UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA
UNAN-LEÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA
DEPARTAMENTO DE MATEMÁTICA Y ESTADÍSTICA

TESIS PARA OPTAR AL TÍTULO DE LICENCIADO EN MATEMÁTICA

TEMA: APAREAMIENTOS EN GRAFOS Y SU IMPLEMENTACIÓN EN C++, CASOS GENERAL Y BIPARTITO

PRESENTADO POR:
BR. JOSÉ OSMÍN MORALES MARTÍNEZ
BR. ELVIN AARÓN GÓMEZ RIZO

BAJO LA DIRECCIÓN DEL DR. RAMIRO JOSÉ CÁCERES ESPINOZA.

“A LA LIBERTAD POR LA UNIVERSIDAD”

Índice

1. INTRODUCCIÓN **6**
1.1. Introducción en general 6
1.2. Estructura y aporte de la tesis 7
1.2.1. Estructura de la tesis 7
1.2.2. Aporte de la tesis 7

2. ANTECEDENTES **8**

3. JUSTIFICACIÓN **11**

4. OBJETIVOS **12**
4.1. Objetivo General 12
4.2. Objetivos Específicos 12

Índice de figuras **13**

5. CONCEPTOS BÁSICOS Y SÍMBOLOS **14**
5.1. Conceptos básicos. 14
5.2. Símbolos. 18

6. FLUJO EN REDES **20**
6.1. Flujo máximo usando el algoritmo de Edmonds-Karp. 20
6.1.1. Los teoremas de Ford y Fulkerson. 20
6.1.2. El algoritmo de Edmonds y Karp. 23
6.1.3. La red auxiliar. 24
6.1.4. El flujo 0-1. 27
6.1.5. Construyendo el bloqueo de flujo. 28

7. APAREAMIENTO EN GRAFOS **30**
7.1. Grafos bipartitos. 30
7.1.1. Teoría. 30
7.1.2. El procedimiento Match. Pseudocódigo. 34
7.2. Grafos generales 35
7.2.1. Los caminos de ampliación. 37
7.2.2. Árboles alternantes y flores. 39
7.2.3. El algoritmo de Edmonds. 42
7.2.4. Algoritmo de Edmonds. Pseudocódigo. 46

8. ALGORITMOS Y SU MODO DE USO **50**
8.1. Sobre grafos bipartitos 50
8.1.1. Algoritmo en C++ para encontrar apareamiento máximo en grafo bipartito. 50
8.1.2. Modo de uso de algoritmo para grafos bipartitos 55
8.2. Sobre grafos generales 56
8.2.1. Algoritmo en C++ para encontrar apareamiento máximo en grafo general. 56
8.2.2. Modo de uso de algoritmo para grafos generales 68

9. APLICACIONES **70**
9.1. Aplicación grafo bipartito 70
9.2. Aplicación grafo general 72

10.CONCLUSIÓN **74**

11.RECOMENDACIONES **75**

12.BIBLIOGRAFÍA **76**

AGRADECIMIENTOS

Hay en verdad muchas personas y entidades a quien debo agradecer. Todas para mí son importantes, así que no prestemos atención al orden en que aparezcan.

- A mi familia, que me ha brindado apoyo en todos los sentidos.
- A la familia Cea-Navas, que constituyó un pilar fundamental para que yo pueda iniciar y finalizar mi carrera. Al mismo tiempo, me ha brindado mucha confianza y estima.
- A la Universidad Nacional Autónoma de Nicaragua Unán-León, por haberme permitido el privilegio de ser uno de sus estudiantes. De esta manera, también ha demostrado en mí su generosidad y solidaridad para con el pueblo Salvadoreño, y también su compromiso para con el desarrollo de la región Centroamericana.
- Al pueblo Nicaragüense en general. Buenos amigos, buenos hermanos.
- A mi ex-profesor y tutor, el Dr Ramiro José Cáceres Espinoza. Un buen maestro. Un buen amigo. Un hombre imprescindible para que esta tesis haya finalizado exitosamente, una persona de quien puede aprenderse bastante, y un modelo a seguir.
- A Gabriel Omón Serrano Hernandez, por ser un visionario. Por haber creído y contribuído a que este proyecto se haga realidad.
- Al M.Sc. José Alberto Cerda Campos. Por haberse involucrado directamente en el proceso para que mis estudios fueran posibles.
- Al Lic. Felipe Martínez Juárez, por su apoyo antes y durante el tiempo que duró la carrera.
- A otras personas que de una u otra forma contribuyeron en este esfuerzo en conjunto.

José Osmín Morales Martínez.

AGRADECIMIENTOS

- A Dios por sobre todas las cosas, quien me ha permitido la vida, la salud y porque me ha dado la fuerza para poder finalizar la carrera con sabiduría.
- A la Universidad Nacional Autónoma De Nicaragua (UNAN-LEÓN). Por haberme dado la oportunidad de estudiar la carrera que me permitirá desempeñar como profesional mis habilidades de estudios aprendidos en el tiempo transcurrido de mi formación.
- A los maestros de la Universidad que me han impartido los componentes para la formación de mi carrera profesional en los años transcurridos.
- A mi exmaestro y tutor Ramiro José Cáceres Espinoza, quien me ha brindado de su tiempo para la contrucción de esta monografía. Él ha sido uno de los maestros esenciales en la formación académica de mis estudios en el transcurso de la carrera.
- A mi familia, quienes con tanto amor y cariño siempre me estuvieron apoyando, animando a seguir adelante, fueron y siguen siendo fundamental para mi, ya que ellos siempre estuvieron conmigo en el momento que los necesité.
- A mis amigos y compañeros que de alguna u otra forma me influyeron para seguir luchando, pues en algún momento fueron de gran ayuda para con mis estudios.

Elvin Aarón Gómez Rizo.

1. INTRODUCCIÓN

1.1. Introducción en general

La teoría de grafos como disciplina matemática fué creada por Euler en su hoy famosa discusión sobre el problema de los puentes de Königsberg. Sin embargo, el artículo de Euler de 1736 permaneció como una contribución aislada por cerca de 100 años. Sobre la mitad del último siglo ha resurgido un renovado interés sobre el estudio de los problemas de la teoría de grafos. Hubieron muchas causas acerca de este renovado interés, entre las cuales podemos mencionar a las ciencias naturales, y su afán por aplicar esta teoría en el estudio de las redes eléctricas, modelos para cristales y el estudio de estructuras moleculares, el desarrollo de lógica formal llevado al estudio de relaciones binarias en forma de grafos.

Un buen número de problemas podrían formularse directamente en términos de grafos. Entre los más célebres está la conjetura del mapa de los 4 colores, la cual fué propuesta por primera vez por el matemático De Morgan allá por el año 1850. Ningún otro problema ha contribuido tanto en ingenio y en número a la teoría de grafos. Debido a su formulación simple y exasperante evasividad queda todavía una emoción fuerte al estudio en lleno de las propiedades de los grafos.

La presente década da testimonio de un desarrollo creciente y sostenido de la teoría de grafos, la cual en los últimos años ha florecido en un período de intensa actividad.
Claramente entendible en este proceso están los efectos de las demandas desde los nuevos campos de aplicación como: teoría de juegos, teoría de las comunicaciones, redes eléctricas y modelado de problemas de biología y psicología.

En esta tesis el asunto que nos ocupará es cómo encontrar en un grafo G un conjunto de aristas sin vértices comunes y con cardinalidad máxima, también llamado apareamiento máximo. Veremos cómo encontrar este conjunto en dos tipos de grafos: grafos bipartitos y grafos generales. En el caso bipartito implementamos el algoritmo de Edmonds-Karp. El algoritmo se basa en la transformación del problema en uno de flujo máximo sobre una red 0-1. En el caso general implementamos el algoritmo de Edmonds. Los dos algoritmos se encuentran en [5]. Ambas implementaciones se dan en lenguaje C++.

Es de dar el mérito a la importante contribución de Jack Edmonds con su trabajo *Paths, Trees and Flowers*, [9] publicado en el año 1965.
A él se debe el uso de los caminos de ampliación y el uso de flores como estrategia para encontrar apareamientos máximos en grafos generales. En el caso del grafo bipartito, es importante mencionar que el Algoritmo de Edmonds-Karp es una modificación del algoritmo de Ford-Fulkerson. Ambos algoritmos fueron originalmente publicados en [10] y en [11] respectivamente.

Para la búsqueda y caracterización de apareamientos en grafos bipartitos, es de recordar la importancia del *Teorema de König*, así como el *Teorema de Menger*, versión vértice y versión arista.

1.2. Estructura y aporte de la tesis

1.2.1. Estructura de la tesis

La parte central de la tesis que presentamos se compone de las siguientes secciones:

En la sección 5 damos a conocer los conceptos básicos usados y los símbolos. Cada autor en sus libros usa una terminología distinta. Aquí usamos la que aparece en [5].

En la sección 6 hablamos de fujo en redes. El algoritmo que encuentra apareamiento máximo en grafos bipartitos usa esta técnica. Los procedimiento usados Auxnet, Block01flow, Augment y Maxflow son todos necesarios para finalmente formar a Match. Más adelante, en la sección 7, se hace referencia a la sección de flujo en redes.

En la sección 7 se presenta formalmente a los apareamientos en grafos. Se inicia con la teoría referente a grafos bipartitos, y además se hace referencia a Match, que se describió en la sección 6. Continuamos con los grafos generales, la teoría y el pseudocódigo de los procedimientos Inmatch, Blossom, Contract y Augment. Al procedimiento Maxmatch que se presenta en esta sección se le conoce como algoritmo de Edmonds.

La sección 8 trata sobre los algoritmos y su modo de uso. Se ha pensado en las personas que posteriormente muestren interés en hacer uso del programa. En la sección 8.1 se muestra el código C++ del programa para grafos bipartitos y a continuación se explica el modo de uso con un ejemplo.
En la sección 8.2 se presenta el código del programa en C++ para grafos generales. Después se explica su modo de uso con un ejemplo.

En la sección 9 damos 2 aplicaciones de las cuales se encargan los programas de apareamiento máximo en grafo bipartito y grafo general respectivamente. Los programas encuentran los apareamiento máximos. El lector podrá verificarlos por sí mismo.

1.2.2. Aporte de la tesis

El aporte de esta tesis consiste en haber estudiado un tema que de manera natural no forma parte del currículum de la carrera Licenciatura en Matemática que oferta la Unan-León.

Además, haber implementado en lenguaje C++ dos algoritmos importantes de la teoría de grafos.
Estos algoritmos ya existían, bajo la forma de pseudocódigo. A partir de esta tesis, estos podrán usarse para resolver situaciones prácticas y de importancia.

2. ANTECEDENTES

El origen de la teoría de grafos se remonta al siglo XVIII con el problema de los puentes de Königsberg, el cual consistía en encontrar un camino que recorriera los siete puentes del río Pregel en la ciudad de Königsberg, actualmente Kaliningrado, de modo que se recorrieran todos los puentes pasando una sola vez por cada uno de ellos.
El trabajo del matemático Suizo Leonhard Euler sobre el problema, titulado **Solutio Problematis ad Geometriam Situs Pertinentis** (La solución de un problema relativo a la geometría de la posición) en 1736, es considerado el primer resultado de la teoría de grafos. También se considera uno de los primeros resultados topológicos en Geometría, esto es, que no depende de ninguna medida. Este ejemplo ilustra la profunda relación entre la Teoría de Grafos y la Topología.

En 1847, Gustav Kirchhoff, de la ciudad Rusa de Kaliningrado, utilizó la teoría de grafos para el análisis de redes eléctricas publicando sus leyes de los circuitos para calcular el voltaje y la corriente en los circuitos eléctricos, conocidas como leyes de Kirchhoff, considerado la primera aplicación de la teoría de grafos a un problema de ingeniería.

En 1852, Francis Guthrie, nacido en Londres y de nacionalidad Sudafricana, planteó el problema de los cuatro colores, el cual puede enunciarse de la siguiente manera: *Dado cualquier mapa geográfico con regiones continuas, éste puede ser coloreado con cuatro colores diferentes, de forma que no queden regiones adyacentes con el mismo color.* Este problema fué resuelto hasta un siglo después por Kenneth Appel y Wolfgang Haken en 1976 con la ayuda de un ordenador. Al tratar de resolverlo, los matemáticos definieron términos y conceptos teóricos fundamentales de la Teoría de grafos. Se considera el problema que más ha contribuído con esta teoría.

El término *grafo*, proviene de la expresión *graphic notation* usada por primera vez por el Inglés Edward Frankland y posteriormente adoptada por Alexander Crum Brown en 1884, y hacía referencia a la representación gráfica de los enlaces entre los átomos de una molécula.

En 1936, luego de haber obtenido el año anterior el cargo de profesor titular, el judío-húngaro Dénes König publicó un libro titulado **Theorie der endlichen und unendlichen Graphen**, que traducido es Teoría de Grafos Finitos e Infinitos.
König anunció en 1914 y publicó en 1916 el siguiente resultado: Cada grafo bipartito regular tiene un apareamiento perfecto.

Hay una serie de algoritmos que de cierta forma se encuentran relacionados. Haremos mención de ellos porque el algoritmo de apareamiento máximo en grafo bipartito de esta tesis se basa en transformar el problema a una situación de flujo máximo en una red de flujo 0-1.

En 1956 salió a la luz el algoritmo de Ford-Fulkerson, el cual propone buscar caminos en los que se pueda aumentar el flujo, hasta que se alcance el flujo máximo. Es aplicable a los flujos maximales. La idea es encontrar una ruta factible

con un flujo positivo que una los nodos origen y destino. Su nombre se debe a sus creadores, L. R. Ford Jr. y Delbert Ray Fulkerson.
Este algoritmo puede usarse para resolver modelos de transporte de mercancías, flujo de líquidos y gases por tuberías, componentes o piezas en líneas de montaje y ensamblaje, corrientes en redes eléctricas, paquetes de información en redes de comunicaciones y tráfico ferroviario.

Otra contribución relacionada con grafos bipartitos podemos encontrarla en el Teorema de Hall, también conocido como Teorema de Matrimonio de Hall. Este fué demostrado por Philip Hall en 1935, y posee dos formulaciones equivalentes:

- La formulación matemática combinatoria trata con una colección de conjuntos finitos. Da una condición necesaria y suficiente para ser capaz de seleccionar un elemento distinto desde cada conjunto.
- La formulación grafo teórica trata con un grafo bipartito. Da una condición necesaria y suficiente para encontrar un apareamiento que cubre al menos un lado del grafo.

La formulación grafo teórica es como sigue: Sea $G = (X + Y, E)$ un grafo bipartito finito con subconjuntos finitos X e Y. Para un conjunto W de vértices en X, $N_G(W)$ denota la vecindad de W en G, es decir, el conjunto de todos los elementos en Y adyacente a algún elemento de W. Entonces hay un apareamiento que cubre por completo a X si y solo sí para cada subconjunto W de X se cumple: $|W| \leq |N_G(W)|$.

Por otra parte, en 1957, Claude Berge, matemático Francés formula el Lema de Berge, que traducido significa lo siguiente:
Un apareamiento M en un grafo G es máximo si y sólo si no hay camino de ampliación con respecto a M.
Información relacionada y de calidad puede encontrarse en [4], [3], [2] y [1].

En 1959, Edsger Wybe Dijkstra contribuye a la teoría de grafos con el famoso algoritmo de Dijkstra, el cual es un algoritmo que determina el camino más corto dado un vértice origen al resto de los vértices, en un grafo con pesos en cada arista. La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo.

En 1961, Jack Edmonds formula el algoritmo de apareamiento máximo.
Este es un algoritmo que construye apareamientos máximos en grafos generales y también el algoritmo en el que se basa esta tesis para resolver el problema de apareamiento máximo. El algoritmo de Edmonds se encuentra en [5]. Fué publicado en 1965 en el artículo *paths, trees and flowers*[9].
El apareamiento máximo es construido iterativamente mejorando el apareamiento actual a través de caminos de ampliación mientras exista al menos uno. La idea esencial del algoritmo es que un ciclo de longitud impar (blossom) se contrae

a un solo vértice para luego continuar la búsqueda de caminos de ampliación en el grafo resultante. La estrategia de contraer los ciclos de longitud impar se debe a que si no se hace, el algoritmo de búsqueda de caminos de ampliación al entrar en uno de estos ciclos podría salir o reportar falsos positivos.
La importancia del algoritmo radica en que dió la primera prueba de que se puede encontrar un apareamiento máximo en tiempo polinomial.

Trasladándonos hasta el año 1973, John Hopcroft y Richard Karp formulan el Algoritmo Hopcroft-Karp. Este toma como valor de entrada un grafo bipartito, y produce como salida un apareamiento de cardinalidad máxima. Al igual que el algoritmo de Edmonds, se basa en aumentar el apareamiento vía caminos de ampliación.

El estudio en la teoría de apareamientos puede ubicarse en el año 1962 con el artículo *College admission and the Stability of Marriage* escrito por D. Gale y L.S. Shapley. Esta es la primera aparición de la teoría de apareamientos en la literatura.

Hasta este momento, solo nos hemos dedicado sobre resultados plenamente teóricos. Entrando al campo de trabajos que incluyen aplicaciones, afirmamos que la teoría de apareamientos ya ha sido aplicada en hospitales. Los hospitales ya han usado algoritmos que aparean internos con programas hospitalarios. Tomando el caso de USA, lo han hecho precisamente con el objetivo de optimizar todos los recursos que tienen a su disposición (en este caso riñones) puesto que se limitan a un programa donante-receptor, en base a la ley de 1984[12]

Acercándonos mas, hacia el año 2000 pudimos ver una clara conexión entre la teoría de apareamientos y el apareo de trasplantes renales. En 2005, Alvin E Roth, Tayfun Sönmez y M. Utku Ünver escribieron el artículo Pairwise Kidney Exchange[6]. En él explican como puede aplicarse la teoría de apareamiento al establecimiento de parejas donante-receptor en el trasplante de riñones y hablan sobre las restricciones posibles.

Por otra parte, puede también verse la importancia de establecer apareamientos máximos en un artículo del año 2005 de la American Medical Association [8] escrito por Dorry L Segev, Sommer E Gentry, Daniel S Warren, Brigitte Reeb y Robert A Montgomery. En este artículo ellos describen como hacen uso de una implementación de un algoritmo optimizado basado en el algoritmo de Edmonds. Por mencionar un caso, con un conjunto de 1000 pares de recipiente-donante, el algoritmo optimizado considera aproximadamente 10^{250} soluciones factibles antes de elegir la mejor solución. El algoritmo optimizado usado en este estudio ha sido probado matemáticamente para producir la solución óptima sobre cualquier conjunto dado de prioridades.

3. JUSTIFICACIÓN

A lo largo del estudio de este tema de la teoría de grafos nos hemos convencido cada vez más de la importancia de encontrar apareamientos máximos en grafos. Supongamos que una firma farmacéutica desea lanzar al mercado un nuevo catálogo de medicamentos. Y se decide a hacer un experimento, para probar la eficacia de los nuevos productos. Para esto, convoca a un grupo de voluntarios, es decir se dispone de un conjunto compuesto por p personas. Pero la asignación de medicamentos a personas no es aleatoria, porque también se sabe de antemano que ciertos voluntarios son alérgicos a medicamentos determinados.
Este problema, de apariencia sencilla, puede fácilmente modelarse usando la teoría de grafos, en específico construir un grafo bipartito entre personas y medicamentos, definiendo aristas entre una persona y un medicamento, si el medicamento puede ser evaluado en esta persona. El propósito es evaluar la mayor cantidad de medicamentos del catálogo. La solución en este caso consiste en hallar en el grafo bipartito el mayor número de emparejamientos, o *apareamientos* posibles.

Veamos otro caso. En los Estados Unidos de norteamérica y algunos países europeos, el número de personas que necesitan un trasplante de riñón asciende a cifras de miles. El problema consiste en que no es del todo fácil a primera vista establecer una alineación correcta entre un receptor y un donante compatibles. A partir de esta situación aseguramos que es posible hallar el mayor número de compatibilidades entre donantes y receptores, para que todo riñón disponible pueda ser trasplantado a un receptor. Una explicación más detallada puede encontrarse en [7]. Cada par que se forma por un receptor y un donante incompatible son visto como uno y se aparean con otro par que está en las mismas condiciones.

Al tener en cuenta estas observaciones, podemos representar este caso como un grafo general. Cada par formado por un donante y un receptor incompatible se toma como un vértice. Tomemos dos vértices $V_i = (d_i, r_i)$ y $V_j = (d_j, r_j)$ donde (d_i, r_i) denota al donante i y receptor i al tiempo que (d_j, r_j) denota al donante j y receptor j respectivamente. Las aristas $\{V_i, V_j\}$ son las compatibilidades entre el donante d_i de V_i y el receptor r_j de V_j, así como también la compatibilidad del receptor r_i de V_i y el donante d_j de V_j.

Basándonos en el ejemplo anterior, se ve fácilmente la importancia del desarrollo de la teoría de apareamientos y el potencial que ofrece en la resolución de problemas en campos como la tecnología, la medicina y las ciencias sociales. Pensamos además que esta tesis podrá ser tomada en cuenta en trabajos y situaciones futuras, tanto en el desarrollo teórico como en aplicaciones.

4. OBJETIVOS

4.1. Objetivo General

Usar la teoría de apareamientos y sus algoritmos básicos para dar respuestas a situaciones de carácter práctico.

4.2. Objetivos Específicos

- Conocer en detalle el procedimiento para construir apareamientos máximos en grafos bipartitos, mediante la solución de problemas de flujo en redes 0-1
- Estudiar el Algoritmo de Edmonds para construir apareamientos máximos en grafos generales.
- Implementar en C++ el Algoritmo de Edmonds para encontrar apareamientos máximos en grafos generales.
- Implementar en C++ el algoritmo que encuentra apareamientos máximos en grafos bipartitos, construyendo un problema de flujo en una red 0-1.

Índice de figuras

1. Grafo. 14
2. Grafo G y un subgrafo de expansión G'. 14
3. Grafo K_4. 15
4. Grafo bipartito completo. 16
5. Apareamiento perfecto en grafo bipartito. 17
6. Vértice expuesto g con respecto al apareamiento M. 17
7. Apareamiento. 30
8. Apareamiento inextendible. 30
9. Camino de ampliación. 37
10. Grafo G con apareamiento M. 38
11. Apareamiento $M' = M \oplus P'$. 38
12. Árbol alternante. 40
13. Caso 3. 41
14. Caso 4. 41
15. Flor. 42
16. Caso 1 del lema 7.2.3.1. 44
17. Caso 2 del lema 7.2.3.1. 44
18. Prueba de lema 7.2.3.2. 45
19. Grafo bipartito. 55
20. Apareamiento máximo en grafo bipartito. 55
21. Grafo general. 68
22. Apareamiento máximo en grafo general. 69
23. Grafo que modela la situación vendedores-compradores. 70
24. Solución de la situación vendedores-compradores. 71
25. Grafo que modela los criterios que cumplen cada uno de los inversores. 72
26. Inversores y sus potenciales parejas en base a los 6 criterios. . . . 73
27. Inversores y sus parejas. 73

5. CONCEPTOS BÁSICOS Y SÍMBOLOS

5.1. Conceptos básicos.

Grafo. Un grafo es un par $G = (V, E)$ que consiste de un conjunto finito de elementos $V \neq \emptyset$ llamados vértices y un conjunto E de subconjuntos de dos elementos de V llamadas aristas.

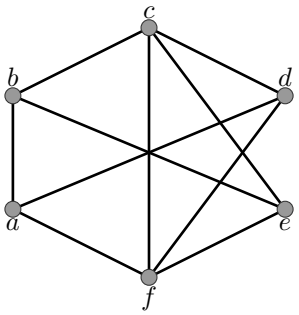

Figura 1: Grafo.

E|V'. Sea $G = (V, E)$ y $V' \subset V$. El símbolo $E|V'$ denota al conjunto de todas las aristas que están en E, las cuales tienen ambos vértices en V'.

Subgrafo. Dado un grafo $G = (V, E)$, un subgrafo de G es cualquier grafo de la forma $G' = (V', E')$ donde $V' \subset V$ y $E' \subset E|V'$.

Subgrafo generado por un subconjunto de vértices. Dado un grafo $G = (V, E)$ y $V' \subset V$, llamamos subgrafo de G generado por V', denotado por $G|V'$ al subgrafo $G|V' = (V', E|V')$.

Subgrafo de expansión. Dado un grafo $G = (V, E)$, un subgrafo $G' = (V', E')$ que cumpla $V' = V$ se llama *subgrafo de expansión de G.*

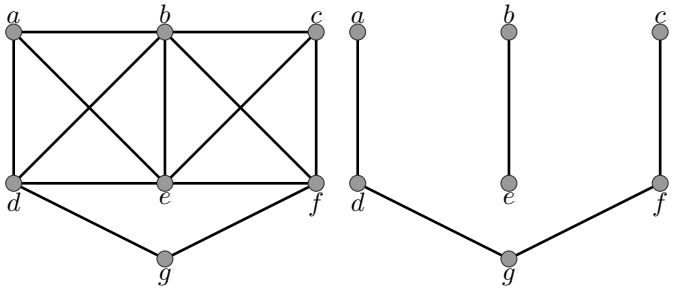

Figura 2: Grafo G y un subgrafo de expansión G'.

Digrafo. Un digrafo o *grafo dirigido* es un par $G = (V, E)$ que consiste de un conjunto finito V y un conjunto E de pares ordenados (a, b), donde $a \neq b$ son elementos de V. Los elementos de V se llaman *vértices* y los elementos de E *arcos*. Denotamos $e = \{a, b\}$ al arco con vértice inicial a y vértice final b.

Camino. Sea $e_1, ..., e_n$ una sucesión de aristas en un grafo G. Si hay vértices $v_0, ..., v_n$ tal que $e_i = \{v_{i-1}, v_i\}$ para $i = 1, ..., n$, la sucesión se llama *paseo*. Si $v_0 = v_n$, es un *paseo cerrado*.
Un *paseo* para el cual las e_i son distintas se llama *sendero*.
Un sendero para el cual $v_0 = v_n$ se llama *sendero cerrado*.
Un *sendero* para el cual los vértices v_j son distintos se le llama *camino*.

Ciclo. En el contexto de la definición anterior, un camino para el cual $v_0 = v_n$ se llama ciclo.

Vértices extremos e incidentes. Sea $G = (V, E)$ con $a, b \in V$ y $e = \{a, b\} \in E$, se dice que:

- a y b son vértices extremos de e.
- a y b son incidentes con la arista e.

Grado de un vértice. Dado cualquier vértice v de un grafo G, el grado del vértice v es el número de aristas incidentes con él, y se denota $grad(v)$.

Vértices y aristas adyacentes. En un grafo $G = (V, E)$

- Si $a, b \in V$ y existe $e = \{a, b\} \in E$, entonces decimos que a y b son vértices adyacentes.
- Si $e, f \in E$ comparten al menos un vértice extremo, e y f se llaman aristas adyacentes.

Grafo completo K_n. El grafo completo K_n es un grafo con n vértices y todos los subconjuntos de dos elementos distintos de V como aristas. El grafo K_n es $(n-1)-$regular.

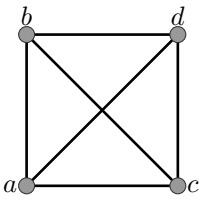

Figura 3: Grafo K_4.

Grafo bipartito. Un grafo $G = (V, E)$ se llama bipartito si hay una partición $V = S \dot{\cup} T$ de su conjunto de vértices tal que los conjuntos de aristas $E|S$ y $E|T$ son vacíos, esto es, cada arista de G es incidente con un vértice en S y un vértice en T.

Grafo bipartito completo. El grafo bipartito completo $K_{m,n}$ tiene como conjunto de vértices la unión disjunta de un conjunto V_1 con m elementos y un conjunto V_2 con n elementos. Las aristas son todos los conjuntos $\{a, b\}$ con $a \in V_1$ y $b \in V_2$.

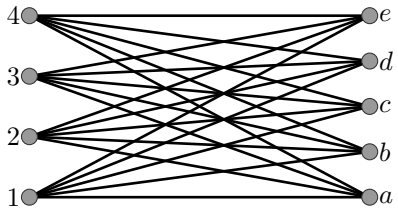

Figura 4: Grafo bipartito completo.

Grafo bipartito regular. Es un grafo bipartito $G = (S \dot{\cup} T, E)$ para el cual todos los vértices tienen el mismo grado $\neq 0$. Esto implica que $|S| = |T|$.

Lista de adyacencia. Sea $G = (V, E)$. La lista de adyacencia A_v de un vértice v es el conjunto de todos los vértices $a \in V$ tales que $\{a, v\} \in E$.

Apareamiento. Sea $G = (V, E)$ un grafo. Un apareamiento M en G es un subgrafo de G que es $1 - regular$. Los apareamiento mostrados en este documento, podemos verlos en los grafos con las aristas remarcadas.

Apareamiento perfecto. Sea $G = (V, E)$ un grafo. Un apareamiento perfecto M en $G = (V, E)$ es un subgrafo de expansión de G que es $1 - regular$.

Apareamiento máximo. Es un apareamiento de cardinalidad máxima.

Apareamiento maximal. Es un apareamiento inextendible M, esto es uno tal que al agregarle cualquier otra arista del grafo éste deja de ser un apareamiento. Un apareamiento inextendible no necesariamente tiene cardinalidad máxima.

Apareamiento completo. Sea $G = (S \dot{\cup} T, E)$ un grafo bipartito. Un apareamiento M de cardinalidad $min\{|S|, |T|\}$ se le llama *apareamiento completo*. De hecho la cardinalidad máxima de un apareamiento en un grafo bipartito está acotada por $min\{|S|, |T|\}$.

Vértice conexo. Dos vértices a y b de un grafo G son llamados *conexos* si existe un *paseo* con vértice inicial a y vértice final b.

Grafo conexo. Dado un grafo $G = (V, E)$, se dice que G es conexo si para cada par de vértices $x, y \in V$ existe un camino con vértice inicial x y vértice final y.

Componente conexa. Sea $G = (V, E)$ un grafo. Se llama componente conexa de G al subgrafo $G|V'$ inducido por V', donde V' es un subconjunto conexo

maximal de V.

Componente impar. Componente conexa de cardinalidad impar.

Componente par. Componente conexa de cardinalidad par.

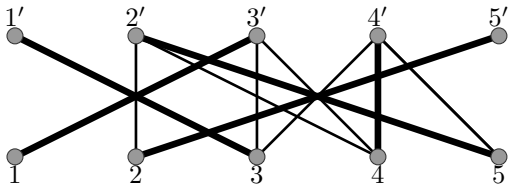

Figura 5: Apareamiento perfecto en grafo bipartito.

Vértice expuesto. Sea $G = (V, E)$ un grafo y M un apareamiento. Entonces a un vértice v que no es incidente con alguna arista en M se le llama *expuesto* con respecto a M.

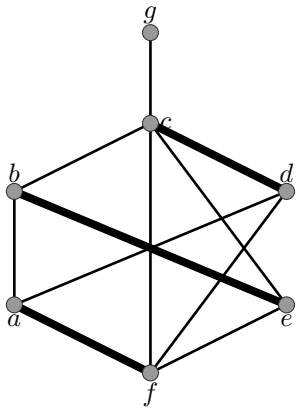

Figura 6: Vértice expuesto g con respecto al apareamiento M.

Vértice saturado o cubierto. Sea M un apareamiento en $G = (V, E)$. Vértices incidentes con alguna arista de M se llaman *saturados* o *cubiertos*.

Compañero. Dado cualquier vértice saturado v, llamamos *compañero* de v al vértice único u tal que $\{u, v\} \in M$.

Árbol. Es un grafo $G = (V, E)$ conexo y sin ciclos.

Lema 1. Dado un grafo $G = (V, E)$, las proposiciones siguientes son equivalentes:

- G es un árbol

- Cualesquiera dos vértices distintos de G son conexos mediante un camino único.
- G es minimalmente conexo. Esto es, G es conexo, pero para cualquiera arista $e \in E$, $G' = (V, E \backslash \{e\})$ es no conexo.
- G es maximalmente acíclico. Esto es, G es acíclico, pero para cualquier par de vértices no adyacentes $a, b \in V$, se cumple que $T' = (V, E \cup \{a, b\})$ es cíclico.

Árbol alternante. Sean G un grafo, M un apareamiento en G y T un árbol que es también subgrafo de G. T se llama *árbol alternante* para M con raíz r, si r es un vértice expuesto respecto a M, y cada camino iniciando en r es un camino alternante.
Los vértices en las capas 0,2,4,... son *vértices exteriores* de T.
Los vértices en las capas 1,3,5,... son *vértices interiores* de T.

Camino alternante. Sea M un apareamiento en $G = (V, E)$. Un *camino alternante* con respecto a M es un camino P en G que inicia en un vértice expuesto, y cuya sucesión de aristas alterna aristas contenidas en M y aristas no contenidas en M.

Camino de ampliación. Es un camino alternante de longitud impar con vértices inicial y final expuestos y distintos.

Flor. Dado $G = (V, E)$, M un apareamiento en G y T un árbol alternante respecto a M. Consideremos dos caminos alternantes $(r, \cdots, w, \cdots, x)$ y $(r, \cdots, w, \cdots, y)$ con la parte común $(r, \cdots, w)$. Llamamos flor al ciclo de longitud impar que se forma cuando agregamos la arista $\{x, y\}$ a $(w, \cdots, x)$ y $(w, \cdots, y)$.

Cobertura de vértice. Sea $G = (V, E)$ un grafo. Una *cobertura de vértice* de G es un subconjunto W de V tal que cada arista de G es incidente con al menos un vértice en W.

5.2. Símbolos.

- $G|U$: Subgrafo de G inducido sobre el conjunto de vértices U.
- $G \backslash e$: G con la arista e desechada.
- $G \backslash T$: Subgrafo de G inducido sobre el conjunto $V \backslash T$.
- G/B: Contracción del grafo G con respecto a la flor B.
- G/e: Contracción del grafo G con respecto a la arista e.
- K_n: Grafo completo sobre n vértices.
- $K_{m,n}$: Grafo bipartito completo sobre $m+n$ vértices.
- N': Red auxiliar para la red de fujo N con respecto al flujo f.

- N'': Red auxiliar de capas para N con respecto al flujo f.
- e^-: Vértice inicial de la arista e.
- e^+: Vértice final de la arista e.
- $\Gamma(J)$: Dado $G = (V, E)$ y $J \subset V$, $\Gamma(J)$ es el conjunto de componentes que son adyacentes a algún vértice en J.
- $\Gamma(v)$: Dado $G = (V, E)$ y $v \in V$, $\Gamma(v)$ es el conjunto de vértices adyacentes al vértice v.
- $\alpha'(G)$: Cardinalidad máxima de un apareamiento de G.
- $\beta(G)$: Cardinalidad mínima de una cobertura de vértice de G.

6. FLUJO EN REDES

6.1. Flujo máximo usando el algoritmo de Edmonds-Karp.

6.1.1. Los teoremas de Ford y Fulkerson.

Sea $G = (V, E)$ un digrafo y sea $c : E \to \mathbb{R}_0^+$ un mapeo; al valor $c(e)$ se le llamará *capacidad* de la arista e. Además, sean s y t dos vértices especiales de G tal que t es accesible desde s. Entonces a $N = (G, c, s, t)$ se le llama una *red de flujo* con *fuente* s y *destino* t. Un *flujo admisible* o *flujo* sobre N es un mapeo $f : R \to \mathbb{R}_0^+$ que satisface dos condiciones:

- F1: $0 \leq f(e) \leq c(e)$ para cada arista e;
- F2: $\sum_{e^+=v} f(e) = \sum_{e^-=v} f(e)$ para cada vértice $v \neq s, t$, donde e^- y e^+ denotan el vértice inicial y final de e.

La condición de factibilidad (F1) requiere que cada arista lleve una cantidad no negativa de flujo, el cual no puede exceder la capacidad de la arista; la condición de conservación del flujo (F2) significa que el flujo se preserva: para cada vértice, excepto para la fuente y el destino, la cantidad que fluye hacia el vértice también fluye desde el vértice. Se ve fácilmente que la cantidad de flujo saliendo de s es la misma cantidad de flujo entrando en t.

Lema 6.1.1.1.

Sea $N = (G, c, s, t)$ una red de flujo con flujo f. Entonces

$$\sum_{e^-=s} f(e) - \sum_{e^+=s} f(e) = \sum_{e^+=t} f(e) - \sum_{e^-=t} f(e). \tag{1}$$

Demostración. Trivialmente,

$$\sum_{e^-=s} f(e) + \sum_{e^-=t} f(e) + \sum_{v\neq s,t} \sum_{e^-=v} f(e)f(e) = \sum_{e} =$$
$$\sum_{e^+=s} f(e) + \sum_{e^+=t} f(e) + \sum_{v\neq s,t} \sum_{e^+=v} f(e).$$

Ahora, la aseveración se sigue inmediatamente desde F2. □

A la cantidad en la ecuación 1 le llamamos *valor de f* y se denota $w(f)$.
Un flujo $w(f)$ se llama *máximo* si se cumple $w(f) \geq w(f')$ para cada flujo f' sobre N.

Sea $N = (G, c, s, t)$ una red de flujo. Un *corte* de N es una partición $V = S \dot{\cup} T$ del conjunto de vértices V de G en dos conjuntos disjuntos S y T con $s \in S$ y $t \in T$.

La capacidad de un corte se define como

$$c(S, T) = \sum_{e^- \in S, e^+ \in T} c(e);$$

así, es sólo la suma de las capacidades de todas las aristas e en el correspondiente cociclo $E(S,T)$ que fué orientado desde S hasta T.
El corte (S,T) se dice *mínimo* si se cumple $c(S,T) \leq c(S',T')$ para cada corte (S',T'). Una arista se llama *saturada* si $f(e) = c(e)$, y *vacía* si $f(e) = 0$.

El lema siguiente muestra que la capacidad de un corte mínimo da la cota superior deseada sobre el valor de un flujo.

Lema 6.1.1.2. *Sea $N = (G, c, s, t)$ una red de flujo, (S,T) un corte y f un flujo. Entonces*

$$w(f) = \sum_{e^- \in S, e^+ \in T} f(e) - \sum_{e^+ \in S, e^- \in T} f(e). \tag{2}$$

En particular, $w(f) \leq c(S,T)$; la igualdad se cumple si y solamente sí cada arista e con $e^- \in S$ y $e^+ \in T$ son saturadas, mientras que cada arista e con $e^- \in T$ y $e^+ \in S$ son vacías.

Demostración. Sumando la ecuación F2 sobre todas las $v \in S$ da
$w(f) = \sum_{v \in S} (\sum_{e^- = v} f(e) - \sum_{e^+ = v} f(e))$
$= \sum_{e^- \in S, e^+ \in S} f(e) + \sum_{e^- \in S, e^+ \in T} f(e) - \sum_{e^+ \in S, e^- \in S} f(e) - \sum_{e^+ \in S, e^- \in T} f(e).$
Tengamos en cuenta que el primer y el tercer término se cancelan. Además, $f(e) \leq c(e)$ para todas las aristas e con $e^- \in S$ y $e^+ \in T$, y $f(e) \geq 0$ para todas las aristas e con $e^+ \in S$ y $e^- \in T$. Esto implica la desigualdad deseada y también el acierto sobre el caso de la igualdad. □

Sea f un flujo en la red $N = (G, c, s, t)$. Un camino W desde s a t se llama *camino de ampliación* con respecto a f si se cumple que $f(e) < c(e)$ para cada arista hacia adelante $e \in W$, mientras que $f(e) > 0$ para cada arista hacia atrás $e \in W$.

Teorema 6.1.1.1 (Teorema camino de ampliación)**.** *Un flujo f sobre una red de flujo $N = (G, c, s, t)$ es máximo si y solo sí no hay caminos de ampliación con respecto a f.*

Demostración. Sea f un flujo máximo. Supongamos que hay un camino de ampliación W. Sea d el mínimo de todos los valores $c(e) - f(e)$. Entonces $d > 0$, por definición de camino de ampliación. Ahora definimos un mapeo $f' : R \to \mathbb{R}_0^+$ como el que sigue:

$$f'(e) = \begin{cases} f(e) + d & \text{si } e \text{ es una arista hacia adelante en } W, \\ f(e) - d & \text{si } e \text{ es una arista hacia atrás en } W, \\ f(e) & \text{en otro caso.} \end{cases}$$

Es fácilmente verificable que f' es un flujo sobre N con valor $w(f') = w(f) + d > w(f)$, contradiciendo la maximalidad de f.

Conversamente, supongamos que no hay camino de ampliación en N con respecto a f. Sea S el conjunto de todos los vértices v tales que existe un camino de ampliación desde s a v (incluyendo s mismo), y pongamos $T = V \backslash S$. Por hipótesis, (S,T) es un corte de N. Tengamos en cuenta que cada arista $e = \{u, v\}$ con $e^- = u \in S$ y $e^+ = v \in T$ debe ser saturada: en otro caso, podría anexarse a un camino de ampliación desde s a u para alcanzar el punto

$v \in T$, una contradicción. De modo similar, cada arista e con $e^- \in T$ y $e^+ \in S$ debe ser vacía. Entonces el lema 6.1.1.2 da $w(f) = c(S,T)$, de modo que f es máximo. □

Corolario 6.1.1.1. *Sea f un flujo sobre una red de flujo $N = (G, c, s, t)$, denotemos por S_f al conjunto de todos los vértices accesibles desde s sobre un camino de ampliación con respecto a f, y pongamos $T_f = V \backslash S_f$. Entonces f es un flujo máximo si y sólo sí $t \in T_f$. En este caso, (S_f, T_f) es un corte mínimo: $w(f) = c(S_f, T_f)$.*

Teorema 6.1.1.2 (Teorema flujo integral)**.** *Sea $N = (G, c, s, t)$ una red de flujo donde todas las capacidades c(e) son enteras. Entonces hay un flujo máximo sobre N de modo que todos los valores f(e) son integrales.*

Demostración. Al establecer $f_0(e) = 0$ para toda e, obtenemos un flujo integral f_0 sobre N con valor 0. Si este flujo trivial no es máximo, existe un camino de ampliación con respecto a f_0. En ese caso el número d que aparece en la demostración del teorema 6.1.1.1 es un entero positivo, y podemos construir un flujo integral f_1 de valor d como en la demostración del teorema 6.1.1.1. Continuamos de la misma manera. Ya que el valor del flujo crece en cada paso por un entero positivo y como la capacidad de cualquier corte es una cota superior sobre el valor del flujo (lema 6.1.1.2) después de un número finito de pasos alcanzamos un flujo integral f para el cual no existen caminos de ampliación. Por teorema 6.1.1.1, este flujo f es máximo. □

Teorema 6.1.1.3 (Teorema flujo máximo - corte mínimo)**.** *El valor máximo de un flujo f sobre una red de flujo N es igual a la capacidad mínima de un corte para N.*

Demostración. Si todas las capacidades son enteras, la aseveración se sigue desde el teorema 6.1.1.2 y el corolario 6.1.1.1. El caso donde todas las capacidades son racionales puede reducirse al caso integral al multiplicar todos los números por su denominador común. Entonces las capacidades de valor real pueden tratarse usando un argumento de continuidad, ya que el conjunto de flujos es un subconjunto compacto de $\mathbb{R}^{|E|}$ y $w(f)$ es una función continua de f. □

Algoritmo 6.1.1.1 (Algoritmo de etiquetado de Ford y Fulkerson)**.** *Sea $N = (G, c, s, t)$ una red de flujo.*

Procedimiento 6.1.1.1.

FORDFULKERSON($N; f, S, T$)

(01) **for** $e \in E$ **do** $f(e) \leftarrow 0$ **od**
(02) etiquetar s con $(-, \infty)$;
(03) **for** $v \in V$ **do** $u(v) \leftarrow$falso; $d(v) \leftarrow \infty$ **od**
(04) **repeat**
(05) escoger un vértice v que está etiquetado y satisface $u(v) =$falso;
(06) **for** $e \in \{e \in E : e^- = v\}$ **do**
(07) **if** $w = e^+$ no está etiquetado **and** $f(e) < c(e)$ **then**
(08) $d(w) \leftarrow \min\{c(e) - f(e), d(v)\}$; etiquetar w con $(v, +, d(w))$ **fi**
(09) **od**
(10) **for** $e \in \{e \in E : e^+ = v\}$ **do**

(11) **if** $w = e^-$ no está etiquetado **and** $f(e) > 0$ **then**
(12) $d(w) \leftarrow \min\{f(e), d(v)\}$; etiquetar w con $(v, -, d(w))$ **fi**
(13) **od**
(14) $u(v) \leftarrow$verdadero;
(15) **if** t está etiquetado
(16) **then** sea d la última componente de la etiqueta de t;
(17) $w \leftarrow t$;
(18) **while** $w \neq s$ **do**
(19) encontrar la primera componente v de la etiqueta de w;
(20) **if** la segunda componente de la etiqueta de w es $+$
(21) **then** establecer $f(e) \leftarrow f(e) + d$ para $e = \{v, w\}$
(22) **else** establecer $f(e) \leftarrow f(e) - d$ para $e = \{w, v\}$
(23) **fi**
(24) $w \leftarrow v$
(25) **od**
(26) borrar todas las etiquetas excepto la etiqueta de s;
(27) **for** $v \in V$ **do** $d(v) \leftarrow \infty$; $u(v) \leftarrow$falso **od**
(28) **fi**
(29) **until** $u(v) =$verdadero para todos los vértices v que están etiquetados;
(30) sea S el conjunto de vértices que están etiquetados y poner $T \leftarrow V \backslash S$

Usando las demostraciones de los teoremas 6.1.1.1 y 6.1.1.2 obtenemos inmediatamente el teorema siguiente, dado por Ford y Fulkerson.

Teorema 6.1.1.4. *Sea N una red de flujo cuya función capacidad c toma solamente valores integrales (o racionales). Entonces el algoritmo 6.1.1.1 determina un flujo máximo f y un corte mínimo (S, T), de manera que se cumple $w(f) = c(S, T)$.*

6.1.2. El algoritmo de Edmonds y Karp.

El algoritmo de Edmonds y Karp, publicado en 1972, es una versión mejorada del algoritmo de etiquetado de Ford y Fulkerson. Bastará si usamos siempre un camino de ampliación de longitud lo más corto posible (esto es, teniendo el mínimo número de aristas) para incrementar el flujo. El algoritmo de Edmonds y Karp es parecido al algoritmo de Ford-Fulkerson; la diferencia está en que tiene un orden definido para ir buscando los caminos de ampliación. El camino encontrado debe ser el camino más corto que tiene capacidad disponible. Esto se puede encontrar mediante una búsqueda del tipo BFS (Breadth First Search), ya que dejamos que las aristas tengan longitud unitaria. Presentamos a continuación el algoritmo de Edmonds y Karp.

Algoritmo 6.1.2.1 (Algoritmo de Edmonds-Karp)**.** *Sea* $N = (G, c, s, t)$ *una red de flujo.*

Procedimiento 6.1.2.1.

EDMONDSKARP($N; f, S, T$)

(01) **for** $e \in E$ **do** $f(e) \leftarrow 0$ **od**
(02) etiquetar s con $(-, \infty)$;
(03) **for** $v \in V$ **do** $u(v) \leftarrow$falso; $d(v) \leftarrow \infty$ **od**
(04) **repeat**
(05) entre todos los vértices con $u(v) =$falso, sea v el vértice que fué etiquetado primero;
(06) **for** $e \in \{e \in E : e^- = v\}$ **do**
(07) **if** $w = e^+$ no está etiquetado **and** $f(e) < c(e)$ **then**
(08) $d(w) \leftarrow \min\{c(e) - f(e), d(v)\}$; etiquetar w con $(v, +, d(w))$ **fi**
(09) **od**
(10) **for** $e \in \{e \in E : e^+ = v\}$ **do**
(11) **if** $w = e^-$ no está etiquetado **and** $f(e) > 0$ **then**
(12) $d(w) \leftarrow \min\{f(e), d(v)\}$; etiquetar w con $(v, -, d(w))$ **fi**
(13) **od**
(14) $u(v) \leftarrow$verdadero;
(15) **if** t está etiquetado
(16) **then** sea d la última componente de la etiqueta de t;
(17) $w \leftarrow t$;
(18) **while** $w \neq s$ **do**
(19) encontrar la primera componente v de la etiqueta de w;
(20) **if** la segunda componente de la etiqueta de w es $+$
(21) **then** poner $f(e) \leftarrow f(e) + d$ para $e = \{v, w\}$
(22) **else** poner $f(e) \leftarrow f(e) - d$ para $e = \{w, v\}$
(23) **fi**
(24) $w \leftarrow v$
(25) **od**
(26) borrar todas las etiquetas excepto la etiqueta de s;
(27) **for** $v \in V$ **do** $d(v) \leftarrow \infty$; $u(v) \leftarrow$falso **od**
(28) **fi**
(29) **until** $u(v) =$verdadero para todos los vértices v que están etiquetados;
(30) sea S el conjunto de vértices que están etiquetados y poner $T \leftarrow V \backslash S$

La diferencia respecto al algoritmo de Ford y Fulkerson se encuentra solamente en la línea 5. Esta modificación hace al algoritmo encontrar con más eficiencia el flujo máximo.

6.1.3. La red auxiliar.

Sea $N = (G, c, s, t)$ una red de flujo con un flujo f. Definimos una red de flujo adicional (G', c', s, t) como sigue. G' tiene el mismo conjunto de vértices que G. Para cada arista $e = \{u, v\}$ de G con $f(e) < c(e)$, hay una arista $e' = \{u, v\}$ en G' con $c'(e') = c(e) - f(e)$; para cada arista $e = \{u, v\}$ con $f(e) \neq 0$, G' contiene una arista $e'' = \{v, u\}$ con $c'(e'') = f(e)$. Llamamos a $N'(f) = (G', c', s, t)$ la *red auxiliar* con respecto a f.

El proceso de etiquetado en el algoritmo de Ford y Fulkerson dado en los pasos (6) a (9) para aristas hacia adelante y en los pasos (10) a (13) para aristas hacia atrás usa solamente aquellas aristas e de G para las cuales G' contiene la arista e' o e'', y un camino de ampliación W (posiblemente dirigido) con respecto a f en G corresponde a un determinado camino único dirigido W' desde s hasta t en G'.

Lema 6.1.3.1. *Sea $N = (G, c, s, t)$ una red de flujo con un flujo f, y sea N' la correspondiente red auxiliar. Entonces f es máximo si y solo sí t no es accesible desde s en G'.*

Vamos a mostrar como podemos usar cualquier flujo en N' para aumentar f cuando construimos un flujo máximo sobre N, y que flujos máximos sobre N' corresponden realmente a flujos máximos sobre N.

Lema 6.1.3.2. *Sea $N = (G, c, s, t)$ una red de flujo con un flujo f, y sea N' la correspondiente red auxiliar. Además, sea f' un flujo sobre N'. Entonces existe un flujo f'' de valor $w(f'') = w(f) + w(f')$ sobre N.*

Demostración. Para cada arista $e = \{u, v\}$ de G, sea $e' = \{u, v\}$ y $e'' = \{v, u\}$. Si e' o e'' no está contenida en N', establecemos $f'(e') = 0$ o $f'(e'') = 0$ respectivamente. Ponemos $f'(e) = f'(e') - f'(e'')$; entonces podemos interpretar f' como un $flujo$ sobre N: f' satisface la condición F2, pero no necesariamente F1. Obviamente, el mapeo f'' definido por

$$f''(e) = f(e) + f'(e') - f'(e'')$$

satisface también la condición F2. Ahora la definición de N' demuestra que las condiciones $0 \leq f'(e') \leq c(e) - f(e)$ y $0 \leq f'(e'') \leq f(e)$ se cumplen para cada arista e, de modo que f'' satisface F1 también. Así f'' es un flujo, y claramente $w(f'') = w(f) + w(f')$. □

Teorema 6.1.3.1. *Sea $N = (G, c, s, t)$ una red de flujo con un flujo f, y sea N' la correspondiente red auxiliar. Denotemos el valor de un flujo máximo sobre N y sobre N' por w y w' respectivamente. Entonces $w = w' + w(f)$.*

Demostración. Por lema 6.1.3.2, $w \geq w' + w(f)$. Ahora sea g un flujo máximo sobre N y definamos un flujo g' sobre N' de la siguiente manera: para cada arista e de G establecemos

$$g'(e') = g(e) - f(e) \text{ si } g(e) > f(e);$$
$$g'(e'') = f(e) - g(e) \text{ si } g(e) < f(e).$$

Tengamos en cuenta que e' y e'' son realmente aristas de N' bajo las condiciones dadas anteriormente y que sus capacidades son lo suficientemente grandes como para asegurar la validez de F1. Para cada otra arista e^* en N', ponemos $g'(e^*) = 0$. Se ve entonces que g' es un flujo de valor $w(g') = w(g) - w(f)$ sobre N'. Esto demuestra que $w' + w(f) \geq w$. □

Teniendo en cuenta nuevamente al algoritmo de Edmonds-Karp, la diferencia que presenta respecto al algoritmo de Ford-Fulkerson en el paso (5) nos asegura que el proceso de etiquetado sobre la red auxiliar $N' = N'(f)$ se comporta como un BFS (Breadth First Search) sobre G'; así el proceso de etiquetado divide a G' en *niveles* o *capas* de vértices teniendo la misma distancia a s. Y como sólo

estamos interesados en encontrar caminos de ampliación de longitud lo más corta, N' contiene usualmente un poco de información que no es relevante, de modo que podemos omitir lo siguiente:

- Todos los vértices $v \neq t$ con $d(s,v) \geq d(s,t)$ junto con todas las aristas incidentes con esos vértices.
- Todas las aristas que van desde algún vértice en la capa j a algún vértice en una capa $i \leq j$

La red resultante $N'' = N''(f) = (G'', c'', s, t)$ se llama la *red auxiliar de capas* con respecto a f. El nombre proviene del hecho de que G'' es un *digrafo por capas*: el conjunto de vértices V de G'' es la unión disjunta de subconjuntos $V_0, \cdots, V_k$, y todas las aristas de G'' tienen la forma $\{u, v\}$ con $u \in V_i$ y $v \in V_{i+1}$ para algún índice i.

El procedimiento siguiente para construir la red auxiliar de capas $N'' = N''(f)$ corresponde al proceso de etiquetado en el algoritmo de Edmonds-Karp. Durante la ejecución del BFS (Breadth First Search), el procedimiento ordena los vértices en capas y omite vértices y aristas superfluos, como lo descrito en la definición de N''. A la variable booleana *max* se le asigna el valor *verdadero* cuando f se vuelve máximo, es decir, cuando t ya no es más accesible desde s; en otro caso tiene valor $falso$. La variable $d + 1$ da el número de capas de N''.

Algoritmo 6.1.3.1. *Sea $N = (G, c, s, t)$ una red de flujo con un flujo f.*

Procedimiento 6.1.3.1.

AUXNET(N, f; N'',max,d)

(01) $i \leftarrow 0$, $V_0 \leftarrow \{s\}$, $E'' \leftarrow \emptyset$, $V'' \leftarrow V_0$;
(02) **repeat**
(03) $i \leftarrow i + 1$, $V_i \leftarrow \emptyset$;
(04) **for** $v \in V_{i-1}$ **do**
(05) **for** $e \in \{e \in E : e^- = v\}$ **do**
(06) **if** $u = e^+ \in V''$ **and** $f(e) < c(e)$
(07) **then** $e' \leftarrow \{v, u\}$, $E'' \leftarrow E'' \cup \{e'\}$, $V_i \leftarrow V_i \cup \{u\}$, $c''(e') \leftarrow c(e) - f(e)$ **fi**
(08) **od**
(09) **for** $e \in \{e \in E : e^+ = v\}$ **do**
(10) **if** $u = e^- \notin V''$ **and** $f(e) \neq 0$
(11) **then** $e'' \leftarrow \{v, u\}$, $E'' \leftarrow E'' \cup \{e''\}$, $V_i \leftarrow V_i \cup \{u\}$, $c''(e'') \leftarrow f(e)$ **fi**
(12) **od**
(13) **od**
(14) **if** $t \in V_i$ **then** remover todos los vértices $v \neq t$ junto con todas las aristas e que satisfacen $e^+ = v$ desde V_i **fi**
(15) $V'' \leftarrow V'' \cup V_i$
(16) **until** $t \in V''$ **or** $V_i = \emptyset$;
(17) **if** $t \in V''$ **then** max←falso; $d \leftarrow i$ **else** max←verdadero **fi**

6.1.4. El flujo 0-1.

Estudiaremos ahora un caso especial que ocurre en muchas aplicaciones combinatorias de la teoría de flujo, llamado flujo integral, el cual toma valores 0 o 1 solamente.

Definición 6.1.4.1. *Red de flujo 0-1.*

Una red de flujo $0-1$ es una red $N=(G,c,s,t)$ que satisface $c(e)=0$ ó $c(e)=1$ para todas las aristas e.

Un flujo f sobre una red de flujo N se llama *flujo 0-1* si f toma valores 0 o 1 solamente.

Lema 6.1.4.1. *Sea $N=(G,c,s,t)$ una red de flujo $0-1$. entonces*

$$d(s,t) \leq \frac{2|V|}{\sqrt{M}}, \tag{3}$$

donde M es el valor máximo de un flujo sobre N. Si además, cada vértice v de N excepto para s y t satisface al menos de las dos condiciones $d_{entrada}(v) \leq 1$ y $d_{salida}(v) \leq 1$, entonces también

$$d(s,t) \leq 1 + \frac{|V|}{M}. \tag{4}$$

Demostración. Sea $D=d(s,t)$, y sea V_i el conjunto de todos los vértices $v \in V$ con $d(s,v)=i$, para $i=0,\cdots,D$. Vértices con distancia más grande a s podrían existir, pero no interesan aquí. Para simplificar la notación, supongamos que no hay tales vértices. Entonces

$$(S_i,T_i)=(V_0 \cup V_1 \cup \cdots \cup V_i, V_{i+1} \cup \cdots \cup V_D)$$

es un corte, para cada $i<D$. Ya que cada arista e con $e^- \in S_i$ y $e^+ \in T_i$ satisface $e^- \in V_i$ y $e^+ \in V_{i+1}$ y como N es una red $0-1$, el lema 6.1.1.2 implica que

$$M \leq c(S_i,T_i) \leq |V_i| \times |V_{i+1}| \text{ para } i=0,\cdots,D-1.$$

Así al menos uno de los dos valores $|V_i|$ o $|V_{i+1}|$ no puede ser más pequeño que $\sqrt{M}$. Por tanto al menos la mitad de las capas $|V_i|$ contienen $\sqrt{M}$ o más vértices. Esto produce

$$D\frac{\sqrt{M}}{2} \leq |V_0| + \cdots + |V_D| \leq |V|,$$

y por tanto 3 se cumple. Ahora supongamos que N satisface la condición adicional establecida en la aseveración. Entonces el flujo a lo largo de cualquier vértice dado no puede exceder uno y obtenemos la desigualdad fuerte

$$M \leq |V_i| \text{ para } i=1,\cdots D-1.$$

Ahora

$$M(D-1) \leq |V_i| + \cdots + |V_{D-1}| \leq V,$$

demostrando 4. □

6.1.5. Construyendo el bloqueo de flujo.

Presentamos a continuación a Block01flow. Este procedimiento construye un bloqueo de flujo después que el procedimiento Maxflow ha usado el procedimiento Auxnet.

Procedimiento 6.1.5.1.

BLOCK01FLOW (N, g)

(01) $L \leftarrow \emptyset$;
(02) **for** $v \in V$ **do** $ind(v) \leftarrow 0$ **od**
(03) **for** $e \in E$ **do** $g(e) \leftarrow 0$; $ind(e^+) \leftarrow ind(e^+) + 1$ **od**
(04) **repeat**
(05) $v \leftarrow t$;
(06) **for** $i = d$ **downto** 1 **do**
(07) escoger una arista $e = \{u, v\}$ y remover e de E;
(08) $ind(v) \leftarrow ind(v) - 1$; $g(e) \leftarrow 1$;
(09) **if** $ind(v) = 0$
(10) **then** anexar v a L;
(11) **while** $L \neq \emptyset$ **do**
(12) remover el primer vértice w de L;
(13) **for** $\{e \in E : e^- = w\}$ **do**
(14) remover e de E; $ind(e^+) \leftarrow ind(e^+) - 1$;
(15) **if** $ind(e^+) = 0$ **then** agregar e^+ a L **fi**
(16) **od**
(17) **od**
(18) **fi**
(19) $v \leftarrow u$
(20) **od**
(21) **until** $ind(t) = 0$

Del procedimiento anterior, queremos usar g para aumentar f. El procedimiento siguiente realiza esta tarea y usa la construcción dada en la demostración de lema 6.1.3.2. Tengamos presente que N' nunca contiene a ambos e' y e''.

Algoritmo 6.1.5.1. *Sea $N = (G, c, s, t)$ una red de flujo dada con un flujo f. Además, supongamos que hemos contruido ya $N'' = N''(f)$ y un bloqueo de flujo g.*

Procedimiento 6.1.5.2.

AUGMENT $(f, g; f)$

(01) **for** $e \in E$ **do**
(02) **if** $e' \in E''$ **then** $f(e) \leftarrow f(e) + g(e')$ **fi**
(03) **if** $e'' \in E''$ **then** $f(e) \leftarrow f(e) - g(e'')$ **fi**
(04) **od**

A continuación, el procedimiento para determinar el flujo máximo.

Algoritmo 6.1.5.2. *Sea $N = (G, c, s, t)$ una red de flujo.*

Procedimiento 6.1.5.3.

MAXFLOW $(N; f)$

(01) **for** $e \in E$ **do** $f(e) \leftarrow 0$ **od**
(02) **repeat**
(03) AUXNET($N, f; N''$,max,d);
(04) **if** max=falso
(05) **then** BLOCK01FLOW($N''; g$); AUGMENT($f, g; f$) **fi**
(06) **until** max=verdadero

Ejemplo 6.1.5.1. *Dado un grafo bipartito $G = (S \dot\cup T)$, encontrar un apareamiento de cardinalidad máxima en G.*

Solución. Este problema es equivalente a encontrar un flujo máximo $0-1$ sobre una red de flujo apropiada. Definimos un digrafo H agregando dos nuevos vértices s y t al conjunto de vértices $S \cup T$ de G. Las aristas de H son todas las $\{s, x\}$ para $x \in S$, y todas las $\{x, y\}$ para quienes $\{x, y\}$ es una arista de G con $x \in S$, y todas las $\{y, t\}$ para $y \in T$. A todas las aristas se les asigna capacidad 1. Hasta aquí hemos definido una red N $0-1$. Tengamos en cuenta que las aristas $\{x_i, y_i\}$, para $i = 1, \cdots, K$, de un apareamiento arbitrario de cardinalidad k para G inducen un flujo de valor k sobre N: pongamos $f(e) = 1$ para todas las aristas $e = \{s, x_i\}$, $e = \{x_i, y_i\}$ y $e = \{y_i, t\}$ para $i = 1, \cdots, k$. Conversamente, un flujo $0-1$ de valor k produce un apareamiento que consiste de k aristas: seleccionar las k aristas del tipo $\{x, y\}$ que realmente llevan un flujo no cero.

El problema anteriormente descrito conlleva a un algoritmo eficiente para determinar apareamientos máximos en grafos bipartitos.
Precisamente, se trata del siguiente algoritmo.

Procedimiento 6.1.5.4.

MATCH $(G; K)$

(01) sean s y t dos nuevos vértices; $V' \leftarrow S \cup T \cup \{s, t\}$;
(02) **for** $e \in E$ **do if** $e = \{x, y\}$ con $x \in S$ **then** $e' = \{x, y\}$ **fi od**
(03) $E' \leftarrow \{\{s, x\} : x \in S\} \cup \{e' : e \in E\} \cup \{\{y, t\} : y \in T\}$;
(04) **for** $e \in E'$ **do** $c(e) \leftarrow 1$ **od**
(05) $H \leftarrow (V', E')$, $N \leftarrow (H, c, s, t)$, $K \leftarrow \emptyset$;
(06) MAXFLOW($N; f$);
(07)**for** $e \in E$ do **if** $f'(e) = 1$ **then** $K \leftarrow K \cup \{e\}$ **fi od**

Proceder con este método para encontrar apareamiento máximo en grafos bipartitos se debe a John Hopcroft y Richard Karp.

7. APAREAMIENTO EN GRAFOS

7.1. Grafos bipartitos.

7.1.1. Teoría.

Supongamos que tenemos un grafo $G = (V, E)$. Un apareamiento en el grafo G es un conjunto M de aristas, de las cuales dos de ellas no tienen ningún vértice en común. Dicho de otra manera, es un conjunto independiente de aristas.

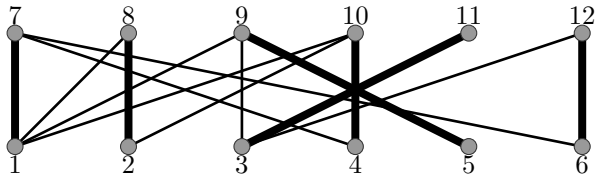

Figura 7: Apareamiento.

La figura 7 muestra un apareamiento M, formado por las aristas $\{1, 7\}$, $\{2, 8\}$, $\{3, 11\}$, $\{4, 10\}$, $\{5, 9\}$ y $\{6, 12\}$.
Generalmente, llamamos *apareamiento máximo* a un apareamiento de cardinalidad máxima. La figura 7 muestra a un apareamiento máximo.

Haremos distinción entre el término *apareamiento máximo* y *apareamiento inextendible*. Un *apareamiento inextendible* M en un grafo G es un apareamiento en el cual la adición al grafo G de una arista $\{v_i, v_{i+1}\}$ que originalmente no pertenece a M, hará que este no sea más un apareamiento. Este *apareamiento inextendible* no necesariamente tiene cardinalidad máxima, como puede verse en la figura 8.

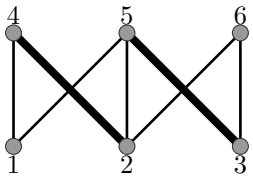

Figura 8: Apareamiento inextendible.

Definición 7.1.1.1 (Separador de arista)**.** *Sea $G = (V, E)$ un grafo, s y t dos vértices de G. Un subconjunto A de E se llama separador de arista para s y t si cada camino desde s hasta t contiene alguna arista de A.*

Definición 7.1.1.2 (Separador de vértice)**.** *Sea $G = (V, E)$ un grafo, s y t dos vértices de G. Un subconjunto X de $V \setminus s,t$ se llama separador de vértice para s y t si cada camino desde s a t concurre a X en algún vértice.*

Teorema 7.1.1.1 (Teorema de Menger, versión arista)**.** *Sea G un grafo o un digrafo, y sean s y t dos vértices de G. Entonces el número máximo de caminos disjuntos en aristas desde s a t es igual a la cardinalidad mínima de un separador de arista para s y t.*

Demostración. En primer lugar, sea G un digrafo. Podemos suponer que t es accesible desde s; en otro caso la aseveración es trivial. Consideremos la red N sobre G donde cada arista tiene capacidad $c(e) = 1$, y denote k el número máximo de caminos dirigidos disjuntos en aristas desde s hasta t. Obviamente, cualquier sistema de tales caminos producen un flujo f de valor k poniendo $f(e) = 1$ si e ocurre en uno de los caminos, y $f(e) = 0$ en otro caso. Por tanto el valor máximo de un flujo sobre N es algún entero $k' \geq k$. Podemos construir un flujo máximo integral empezando con el flujo cero y entonces usar k' caminos de ampliación de capacidad 1. Tengamos en cuenta que estos caminos no son necesariamente dirigidos, ya que podrían ocurrir aristas hacia atrás. Sin embargo, es siempre posible encontrar k' caminos de ampliación sin aristas hacia atrás: supongamos que $e = \{u, v\}$ es una arista hacia atrás que ocurre en el camino W; entonces debe existir un camino W' que fué construido antes que W el cual contiene a e como arista hacia adelante. Así los caminos W y W' tienen la forma

$$W = s \overset{W_1}{-} v \overset{e}{-} u \overset{W_2}{-} t$$

y

$$W^{'} = s \overset{W_1^{'}}{-} u \overset{e}{-} v \overset{W_2^{'}}{-} t.$$

Ahora podemos reemplazar los caminos W y W' por los caminos $W_1 W_2^{'}$ y $W_1^{'} W_2$ y eliminar así la arista e. Podemos suponer que e es la arista hacia atrás que ocurrió primero; entonces W_1, $W_2^{'}$ y $W_1^{'}$ contienen aristas hacia adelante solamente (mientras W_2 podría contener todavía aristas hacia atrás). Repitiendo esta construcción tanto como sea necesario, obtenemos k' caminos de ampliación que consisten de aristas hacia adelante solamente, esto es, caminos dirigidos desde s a t en G. Cualesquiera dos caminos de ampliación que consisten de aristas hacia adelante deben ser disjuntos en aristas, ya que todas las aristas tienen capacidad unidad. Esto implica que $k' \leq k$ y por tanto $k = k'$.
Así el número máximo de caminos disjuntos en aristas desde s a t en G es igual al valor máximo de un flujo sobre N y por tanto, a la capacidad de un corte mínimo en N. Resta mostrar que la cardinalidad mínima de un separador de arista para s y t es igual a la capacidad de un corte mínimo en N. Obviamente, cualquier corte (S, T) en N produce un separador de arista de cardinalidad $c(S, T)$:

$$A = \{e \in E : e^- \in S, e^+ \in T\}.$$

Conversamente, sea A un separador de arista mínimo dado para s y t. Denotemos al conjunto de esos vértices v que son accesibles desde s por un camino dirigido no conteniendo aristas de A por S_A, y pongamos $T_A = V \backslash S_A$. Entonces (S_A, T_A) es un corte en N. Viendo las definiciones de los conjuntos S_A y T_A, es claro que cada arista e con $e^- \in S_A$ y $e^+ \in T_A$ debe estar contenida en A. Como A es mínimo, A consiste de exactamente esas aristas y es por consiguiente inducida por un corte. Esto demuestra el teorema para el caso dirigido.

Ahora, sea G un grafo. Reducimos este caso al caso dirigido al considerar la orientación completa $\overrightarrow{G}$ de G. Obviamente, un sistema de caminos disjuntos en aristas en G inducen un sistema correspondiente de caminos dirigidos disjuntos en aristas en $\overrightarrow{G}$. El converso también se cumple, siempre y cuando los caminos dirigidos disjuntos en aristas en $\overrightarrow{G}$ no contengan ningún par de aristas antiparalelas. Pero tales pares de aristas pueden ser eliminadas, similar a la eliminación de aristas hacia atrás en la primera parte de la demostración. Ahora sea k el número máximo de caminos dirigidos disjuntos en aristas en G y por tanto también en $\overrightarrow{G}$. Entonces existe un separador de arista $\overrightarrow{A}$ en $\overrightarrow{G}$ de cardinalidad k; el correspondiente conjunto de aristas en G es un separador de arista para s y t en G de cardinalidad $\leq k$. Ya que la cardinalidad mínima de un separador de arista para s y t debe ser al menos tan grande como el número máximo de caminos disjuntos desde s hasta t, obtenemos la aseveración. □

Al igual que el Teorema de Menger, versión arista, de gran importancia nos resulta el teorema de Menger, versión vértice.

Teorema 7.1.1.2 (Teorema de Menger, versión vértice)**.** *Sea G un grafo o un digrafo, y sean s y t dos vértices no adyacentes de G. Entonces el número máximo de caminos disjuntos en vértice desde s hasta t es igual a la cardinalidad mínima de un separador de vértice para s y t.*

Demostración. Supongamos que G es un digrafo. Para reducir la aseveración al teorema 7.1.1.1, definimos un nuevo digrafo G'' de la siguiente manera: separamos cada vértice diferente de s y t en dos partes asociadas por una arista; esto resultará en la transformación de caminos disjuntos en vértices a caminos disjuntos en aristas.
De manera formal, los vértices de G' son s y t y para cada vértice $v \neq s, t$ de G, dos nuevos vértices v' y v''. Para cada arista $\{s, v\}$ o $\{v, t\}$ en G, G' contiene la arista $\{s, v'\}$ o $\{v'', t\}$ respectivamente; y para cada arista $\{u, v\}$ en G, donde $u, v \neq s, t$, G' contiene la arista $\{u'', v'\}$. Finalmente, G' contiene también a todas las aristas de la forma $\{v', v''\}$, donde v es un vértice de G con $v \neq s, t$. Es claro que caminos disjuntos en vértices en G corresponden ciertamente a caminos disjuntos en G'.
Por teorema 7.1.1.1, el número máximo de caminos disjuntos en aristas desde s a t en G' es igual a la cardinalidad mínima de un separador de arista A para s y t. Por supuesto, A podría no contener aristas de la forma $\{v', v''\}$, en cuyo caso no lo haría corresponder inmediatamente a un separador de vértice en G. Sin embargo, si alguna arista $\{u'', v'\}$ ocurre en A, podemos reemplazarla por $\{u', u''\}$ y obtener otra vez un separador de arista mínimo. Por tanto podemos restringir nuestras consideraciones al separador de arista mínimo en G' que solamente contiene aristas de la forma $\{v', v''\}$ y por consiguiente corresponder a los separadores de vértice en G. □

Ahora presentamos un ejercicio resuelto:

Ejercicio 7.1.1.1. *Sea G un grafo. Denotemos la cardinalidad máxima de un apareamiento en G por k. Encontrar una cota inferior para el número de aristas de un apareamiento inextendible.*

Solución: Sea $G = (S \dot{\cup} T, E)$ un grafo bipartito. Definimos un nuevo grafo H el cual tiene, además de los vértices de G, dos nuevos vértices s y t y cuyas aristas son las aristas de G más todas las aristas $\{s, x\}$ para $x \in S$ y todas las aristas $\{y, t\}$ para $y \in T$.
Las aristas de un apareamiento M en G corresponden a vértices de caminos disjuntos de s a t en H: asociemos el camino $s - x - y - t$ con la arista $\{x, y\}$ en G, donde $x \in S$ y $y \in T$. Apliquemos el Teorema 7.1.1.2 al grafo H acabado de definir e interpretemos este resultado dentro de G. Como se señaló arriba, vértices de caminos disjuntos en H corresponden a apareamientos en G. También, un separador de vértice para s y t en H es un conjunto X de vértices en G tales que cada arista de G tiene al menos uno de sus vértices extremos en X; esto es, X es una cobertura de vértice para G.
Se denota la cardinalidad máxima de un apareamiento por $\alpha'(G)$, y la cardinalidad mínima de una cobertura de vértice por $\beta(G)$.
Usando esta notación, el Teorema 7.1.1.2 implica inmediatamente el siguiente importante resultado : $\alpha'(G) = \beta(G)$.

A continuación veremos una segunda demostración, basada en contraejemplo mínimo. Hablamos del Teorema de König.

Teorema 7.1.1.3 (Teorema de König)**.** *Sea G un grafo bipartito. Entonces la cardinalidad máxima de un apareamiento en G es igual a la cardinalidad mínima de una cobertura de vértice: $\alpha'(G) = \beta(G)$.*

Demostración. Por definición, ningún vértice puede ser incidente con más de una arista en un apareamiento dado. Por lo tanto una dirección de la afirmación es obvia: $\alpha'(G) \leq \beta(G)$. Así solo necesitamos demostrar la desigualdad inversa.
Ahora supongamos al contrario que $\alpha'(G) < \beta(G)$. Entre todos los grafos bipartitos que violen el teorema, escojamos $G = (S \dot{\cup} T, E)$ como un contraejemplo mínimo: G tiene el número más pequeño posible de vértices, digamos n; y entre todos los contraejemplos sobre n vértices, G también tiene el número mínimo de aristas. Entonces G es conexo. También, G no puede ser un ciclo o un camino, ya que grafos bipartitos de este tipo claramente satisfacen el teorema. Por lo tanto podemos escoger un vértice u con grado $u \geq 3$. Sea v adyacente a u, y consideremos el grafo $G \backslash v$. Ya que G se escogió como un contraejemplo mínimo, $G \backslash v$ satisface el teorema: $\alpha'(G \backslash v) = \beta(G \backslash v)$.
Ahora supongamos que $\alpha'(G \backslash v) < \alpha'(G)$. Entonces podemos adjuntar v a una cobertura de vértice W de $G \backslash v$ con cardinalidad $\alpha'(G \backslash v)$ para obtener una cobertura de vértice para G. Esto implica que $\beta(G) \leq \alpha'(G \backslash v) + 1 \leq \alpha'(G)$, y G satisface el teorema después de todo, una contradicción.
Por tanto debemos tener $\alpha'(G \backslash v) = \alpha'(G)$. Entonces existe un apareamiento máximo M de G para el cual ninguna arista en M es incidente con v. En vista de que grado $u \geq 3$, podemos escoger una arista $e \notin M$ que sea incidente con u, pero no con v. Por la minimalidad de G, el subgrafo $G \backslash e$ satisface el teorema, y obtenemos $\alpha'(G) = \alpha'(G \backslash e) = \beta(G \backslash e)$. Sea W' una cobertura de vértice de $G \backslash e$ con cardinalidad $\alpha'(G)$. Como ninguna arista en M es incidente con v, debemos tener $v \notin W'$. Por lo tanto W' debe contener el otro vértice extremo u de la arista $\{u, v\} \neq e$ y es por consiguiente en realidad una cobertura de vértice para G. De nuevo, G satisface el teorema después de todo; esta contradicción final establece el teorema. □

La cardinalidad máxima de un apareamiento en un grafo bipartito $G = (S \dot\cup T, E)$ está acotada por el $\min\{|S|, |T|\}$. Un apareamiento de esta cardinalidad se llama *apareamiento completo*.

El teorema siguiente, que se debe a Philip Hall, caracteriza a los grafos bipartitos que admiten un apareamiento completo.

Teorema 7.1.1.4. *Sea $G = (S \dot\cup T, E)$ un grafo bipartito con $|S| \geq |T|$. Para $J \subset T$, $\Gamma(J)$ denota el conjunto de todos aquellos vértices en S que son adyacentes a algún vértice en J. Entonces G admite un apareamiento completo si y solo sí se satisface la siguiente condición:*

$$|\Gamma(J)| \geq |J| \quad \forall J \subset T. \tag{5}$$

Demostración. Para ver que la condición 5 es necesaria, sea M un apareamiento completo de G y J cualquier subconjunto de T. Denotar al conjunto de aristas contenidas en M que son incidentes con un vértice en J por $E(J)$. Entonces los vértices extremos de las aristas en $E(J)$ que están contenidas en S forman un subconjunto de cardinalidad $|J|$ de $\Gamma(J)$. Conversamente, supongamos que la condición 5 se satisface y que la cardinalidad máxima de un apareamiento en G es menor que $|T|$. Entonces el Teorema 7.1.1.3 da la existencia de una cobertura de vértice $X = S' \dot\cup T'$ con $S' \subset S$, $T' \subset T$, y $|S'| + |T'| < |T|$. Pero entonces los vértices extremos u de esas aristas $\{u, v\}$ para las cuales v es uno de los $|T| - |T'|$ vértices en $T \backslash T'$ están todos contenidos en S', de modo que

$$|\Gamma(T \backslash T')| \leq |S'| < |T| - |T'| = |T \backslash T'|,$$

una contradicción. □

7.1.2. El procedimiento Match. Pseudocódigo.

El procedimiento Match (6.1.5.4) es un procedimiento de la teoría de grafos que encuentra un apareamiento máximo en grafos bipartitos. Este incluye al procedimiento Maxflow, y Maxflow hace uso de los tres procedimientos auxiliares siguientes: Auxnet, Block01flow y Augment. El procedimiento Auxnet contruye una red auxiliar, Block01flow construye un bloqueo de flujo y Augment aumenta el flujo sobre las aristas de la red.
Para construir el flujo máximo sobre una red, se usa al algoritmo de Edmonds y Karp, es cual es una modificación del algoritmo de etiquetado de Ford y Fulkerson. Para ver en más detalle cada uno los algoritmos y procedimientos usados, ver la sección 6.1.

7.2. Grafos generales

En esta tesis, abordamos el problema de encontrar un apareamiento máximo en un grafo general basándonos en el Algoritmo de Edmonds. El artículo de Jack Edmonds [9] publicado en 1965 sentó las bases para el conocido Algoritmo de Edmonds y también para el posterior desarrollo de algoritmos relacionados al mismo problema. No es fácil reducir el caso general a un problema de flujo. Sin embargo, el algoritmo modifica apropiadamente la situación: trata el problema por partes.
En la primera parte, usa el procedimiento INMATCH. La función de este procedimiento es crear un apareamiento inicial en el grafo. El segundo procedimiento se llama BLOSSOM. El procedimiento BLOSSOM detecta una flor B con base w. Después hace sus funciones el procedimiento CONTRACT el cual toma la flor B detectada anteriormente y la *contrae* a un solo vértice llamado *pseudovértice* b. Finalmente, el procedimiento AUGMENT se encarga de construir caminos de ampliación y actualizar el apareamiento M consecuentemente.
Todos estos procedimientos, unidos apropiadamente forman finalmente el procedimiento MAXMATCH.

Al igual que en el caso bipartito, un $1-factor$ también se llama *apareamiento perfecto* de G.

Teorema 7.2.0.1 (1-factor)**.** *Sea $G = (V, E)$ un grafo. Entonces G tiene un apareamiento perfecto si y solo sí la siguiente condición se cumple:*

$$o(S) \leq |S| \text{ para cada subconjunto } S \text{ de } V \tag{6}$$

donde $o(S)$ denota al número de componentes impares de $G \backslash S$

Sea $G = (V, E)$ un grafo y M un apareamiento. A cualquier vértice v de G que no es incidente con ninguna arista de M se le llama *expuesto* con respecto a M. Los vértices que son incidentes con aristas de M se llaman *saturados* o *cubiertos*. Dado cualquier vértice saturado v, llamamos *compañero* de v al vértice u tal que la arista $\{u, v\} \in M$.

Teorema 7.2.0.2. *(Fórmula de Berge-Tutte) Sea $G = (V, E)$ un grafo, y sea M un apareamiento máximo de G. Hay precisamente*

$$d = max\{o(S) - |S| : S \subset V\} \tag{7}$$

vértices expuestos, y M consiste de $(|V| - d)/2$ aristas.

Demostración. Sea M un apareamiento de G con exactamente e vértices expuestos, y sea S un subconjunto arbitrario de V. Tomar en cuenta que cualquier componente impar de $G \backslash S$ debe contener al menos un vértice expuesto, a menos que uno de sus vértices sea el compañero de algún vértice en S. Por consiguiente, el número de componentes impares en $G \backslash S$ no puede exceder $|S|$ más el número total de vértices expuestos, esto es

$$o(S) - |S| \leq e.$$

Como esto se cumple para todas las elecciones de S vemos que e es al menos el máximo d definido en la ecuación 7.
Ahora haremos inducción sobre el número n de vértices de G para mostrar que un apareamiento con exactamente d vértices expuestos ciertamente existe. Esto es trivial para $n = 0$, pero el paso de inducción toma algún esfuerzo. En lo que sigue, abreviaremos la cantidad $o(S)-|S|$ como $d(S)$, y se llamará la $deficiencia$ de S. Empezamos con una observación simple:

$$d(s) = o(S) - |S| \equiv |V| \pmod 2 \text{ para cada } S \subseteq V \tag{8}$$

Para ver esto, tengamos en cuenta que V equivale a la suma de $|S|$ y las cardinalidades de todas las componentes conexas de $G\backslash S$, de modo que $o(S)+|S| \equiv n \pmod 2$.
Escogemos ahora un conjunto máximo R (con respecto a la inclusión) entre todos los conjuntos S de deficiencia máxima, esto es, entre todos los conjuntos con $d(S) = d$. Solicitamos que cada componente de $G\backslash R$ debe ser impar. Supongamos otro caso, y sea C una componente par. entonces podemos agregar una hoja a de algún árbol de expansión para C a R para obtener un conjunto más grande con la misma deficiencia d, lo cual contradice la maximalidad de R.
Ahora sea R^* el conjunto de todas las componentes (necesariamente impares) de $H = G\backslash R$, y considerar el grafo bipartito B con conjunto de vértices $R \cup R^*$ para el cual un vértice r en R y una componente C en R^* son adyacentes si y sólo sí hay una arista rc en G con $c \in C$. Mostramos que B tiene un apareamiento completo, por verificar la condición 5 del teorema 7.1.1.4. Así sea $J \subseteq R$, y sea $T := \Gamma(J)$ el conjunto de vecinos de J en B, esto es, el conjunto de componentes de H que son adyacentes a algún vértice en J. Entonces los vértices en $R^*\backslash T$ forman las componentes (impares) de $G\backslash(R\backslash J)$, ya que ninguna componente en $R^*\backslash T$ tiene un vecino en J. Así

$$d(R\backslash J) = |R^*\backslash T| - |R\backslash J| \leq d = d(R) = |R^*| - |R|,$$

lo cual se simplifica a $|J| \leq |\Gamma(J)|$, como se deseaba. Así B admite ciertamente una cobertura de apareamiento R.
Por consiguiente, podemos asociar con cada vértice $y \in R$ un vértice x_y de $G\backslash R$ en tal forma que $\{y, x_y\}$ es siempre una arista en G y que la x_y pertenezca a pares apropiados de componentes distintos de $G\backslash R$. Esto produce un apareamiento M_R de G que cubre R y evita exactamente $o(R) - |R| = d$ componentes (impares) de $G\backslash R$. Ahora basta mostrar que, para cada componente C de $G\backslash R$ y cada vértice x de C, el grafo inducido G_C sobre el conjunto de vértices $C\backslash x$ tiene un apareamiento perfecto M_C. Luego la unión de M_R y todos esos apareamientos M_C producirán el apareamiento deseado de G con exactamente d vértices expuestos, siempre y cuando escojamos $x = x_y$ cuanta vez una componente C contenga uno de esos vértices especiales.
Para este fin, verificamos la condición 6 para G_C, y luego usar la hipótesis de inducción. Para un subconjunto S de $C\backslash x$, denote $d'(S)$ la deficiencia de S con respecto al grafo G_C. Entonces

$$\begin{aligned} d(R \cup S \cup \{x\}) \quad &= o(G\backslash(R \cup S \cup \{x\})) - |R \cup S \cup \{x\}| \\ &= o(G\backslash R) + o((C\backslash\{x\}\backslash S) - 1 - (|R| + |S| + 1)) \\ &= d(R) + d'(S) - 2 \end{aligned}$$

Por la escogencia de R como un conjunto máximo de deficiencia máxima,

$$d(R) + d'(S) - 2 = d(R \cup S \cup \{x\}) < d(R). \tag{9}$$

Tener en cuenta que $d'(S)$ debe ser par: esto se sigue de 8 aplicado a G_C, el cual es un grafo con un número par de vértices. Por consiguiente, 9 forza $d'(S) \leq 0$, y por tanto 6 ciertamente se cumple para G_C. □

7.2.1. Los caminos de ampliación.

Cuando tratamos el caso general, atendiendo al Algoritmo de Edmonds, debemos hablar de *caminos de ampliación* con respecto a un apareamiento M dado que todavía no es máximo. Esto lo hacemos en un grafo $G = (V, E)$ para la construcción de un apareamiento M' de cardinalidad mayor. Cuando hablábamos del caso apareamiento máximo en grafo bipartito, determinamos un apareamiento máximo de G haciendo uso de un flujo máximo sobre una red de flujo $0 - 1$. Ahora vamos a describir los caminos de ampliación que ocurren durante este proceso dentro del grafo G.
Sea M un apareamiento de cardinalidad k en G, y denotemos por f al flujo correspondiente $0-1$. Así, un camino de ampliación se ve de la siguiente forma:

$$s \to v_1 \to v_2 \leftarrow ... \to v_{2n-2} \leftarrow v_{2n-1} \to v_{2n} \to t,$$

donde v_1 y v_{2n} son vértices que no son incidentes con ninguna arista saturada y donde las aristas $\{v_{2i}, v_{2i+1}\}$ son aristas hacia atrás (es decir, están saturadas). Así los vértices v_1 y v_{2n} están expuestos con respecto a M, y las aristas $\{v_{2i}, v_{2i+1}\}$ están contenidas en M; ver figura 9, donde las aristas remarcadas están contenidas en el apareamiento M.

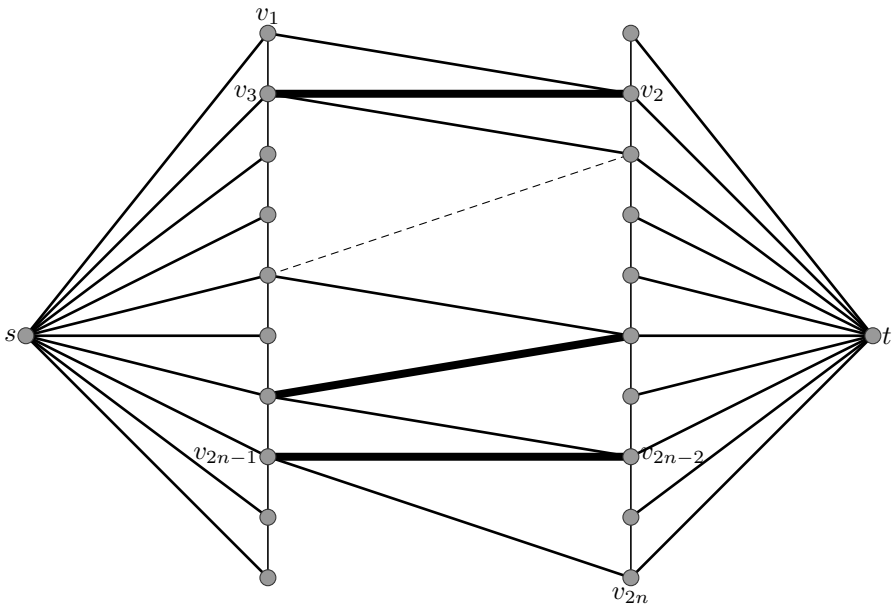

Figura 9: Camino de ampliación.

El caso bipartito nos sugiere la siguiente manera de definir caminos de ampliación en general. Sea M un apareamiento en un grafo arbitrario $G = (V, E)$. Un *camino alternante* con respecto a M es un camino P para el cual aristas contenidas en M se alternan con aristas no contenidas en M. Tal camino se llama *camino de ampliación* si sus vértices inicial y final son vértices expuestos distintos.

Ejemplo 7.2.1.1.

Las aristas remarcadas en el grafo G mostradas en Fig 10 forman un apareamiento M. Los vértices a, f e y son expuestos con respecto a M. La sucesión (y, j, h, i, g, b, c, a) define un camino de ampliación P. Intercambiando los roles de las aristas de M y no aristas de M produce el apareamiento M' de cardinalidad $|M| + 1$ que se muestra en Fig. 11. De manera formal, reemplazamos M por $M \oplus P'$, donde $M \oplus P'$ denota la diferencia simétrica. Tener en cuenta que M' es un apareamiento máximo de G, ya que hay solamente un vértice expuesto.

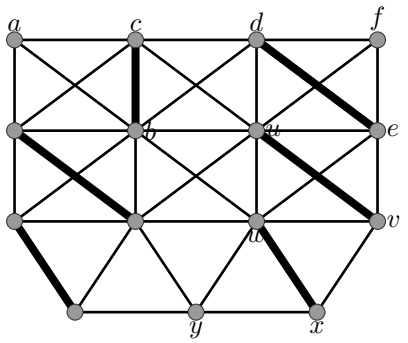

Figura 10: Grafo G con apareamiento M.

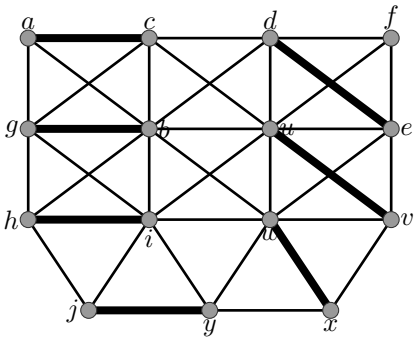

Figura 11: Apareamiento $M' = M \oplus P'$.

Teorema 7.2.1.1 (Teorema Camino de ampliación)**.** *Un apareamiento M en un grafo G es máximo si y solo sí no hay camino de ampliación con respecto a M.*

Demostración. Supongamos primero que M es máximo. Si existe un camino de ampliación P en G, podemos reemplazar M por $M' = M \oplus P$, como en el ejemplo 7.2.1.1. Entonces M' es un apareamiento de cardinalidad $|M| + 1$, una contradicción.
Conversamente, supongamos que M no es máximo; vamos a demostrar la existencia de un camino de ampliación con respecto a M. Sea M' cualquier apareamiento máximo, y consideremos el subgrafo H de G determinado por las aristas en $M \oplus M'$. Tengamos en cuenta que cada vértice de H tiene a lo sumo grado 2; también, un vértice v que tiene grado 2 debe de ser incidente con precisamente una arista de M y una arista de M'.
Por consiguiente una componente conexa de H que consiste de más de un vértice debe ser o bien un ciclo de longitud par (cuando las aristas de M y M' se alternan) o un camino formado por una sucesión alternante de tales aristas. Como M' contiene más aristas que M, existe al menos un tal camino P cuyas primera y última aristas pertenecen a M'. Entonces P es un camino de ampliación con respecto a M, ya que sus vértices finales están obviamente expuestos. □

El teorema 7.2.1.1 es la base de más algoritmos para determinar apareamientos máximos en grafos arbitrarios. La idea básica es esta: iniciamos con cualquier apareamiento dado -por ejemplo, el apareamiento vacío o justo una simple arista- y tratamos de encontrar un camino de ampliación con respecto al apareamiento presente a fin de ampliar el apareamiento hasta que tales caminos no existan más. Para ello, necesitamos una técnica eficiente para encontrar caminos de ampliación; hay que recordar que en general el número de caminos en un grafo crece exponencialmente con el tamaño del grafo.

7.2.2. Árboles alternantes y flores.

El primer algoritmo polinomial para determinar apareamientos máximos se debe a Jack Edmonds; su algoritmo se basa en el uso de caminos de ampliación de acuerdo con el teorema 7.2.1.1.
Aunque es posible usar el apareamiento vacío para inicializar la construcción de un apareamiento máximo a través de caminos de ampliación, desde un punto de vista práctico es obviamente aconsejable determinar un apareamiento inicial razonablemente grande de una manera heurística: podemos esperar de esto resultar en una reducción considerable del número de fases requeridas por el algoritmo. Daremos un método agresivo y a la vez simple para encontrar tal apareamiento inicial.

Algoritmo 7.2.2.1.

Sea $G = (V, E)$ un grafo con conjunto de vértices $V = \{1, ..., n\}$. El algoritmo construye un apareamiento inextendible M descrito por un arreglo *compañero*: para $\{i, j\} \in M$, compañero$(i) = j$ y compañero$(j) = i$, mientras que compañero$(k) = 0$ para vértices expuestos k. La variable $nrex$ denota el número de vértices expuestos con respecto a M.

Procedimiento 7.2.2.1.

INMATCH(G;mate,nrex)

(1) nrex $\leftarrow n$;
(2) **for** $i = 1$ **to** n **do** compañero(i) $\leftarrow 0$ **od**
(3) **for** $k = 1$ **to** $n - 1$ **do**
(4) **if** compañero(k) $= 0$ **and** allí existe $j \in A_k$ con compañero(j) $= 0$
(5) **then** escoger $j \in A_k$ con compañero(j) $= 0$;
(6) compañero(j) $\leftarrow k$; compañero(k) $\leftarrow j$; nrex $\leftarrow$ nrex - 2
(7) **fi**
(8) **od**

El algoritmo toma como valor de entrada un grafo G, y da como valor de salida un arreglo *mate* y una variable *nrex* que denota al número de vértices expuestos con respecto a M.

El siguiente paso es diseñar una técnica eficiente para encontrar caminos de ampliación. Empecemos por escoger un vértice expuesto r con respecto a un apareamiento M de G. Tomemos a r como el vértice inicial para un BFS (Breadth First Search) y pongamos todos los vértices $a_1, \cdots, a_p$ adyacentes a r en la primera capa; tener en cuenta que todos estos vértices son saturados. Como estamos buscando caminos alternantes, ponemos solamente los vértices b_i=compañero(a_i) en la segunda capa. La siguiente capa se compone de todos los vértices $c_1, \cdots, c_q$ que son adyacentes a uno de los b_i, y donde la arista conexa no está contenida en M. Continuando de esta manera; pronto pueden surgir dificultades.
Si encontráramos un vértice expuesto en una de las capas de numeración impar, entonces hemos encontrado un camino de ampliación.
Los vértices en las capas $0, 2, 4, ...$, se llaman *vértices exteriores*, y los vértices en las capas $1, 3, 5, \cdots$ son *vértices interiores* de T. Así un árbol alternante se ve como el árbol mostrado en la Fig. 12, donde las aristas remarcadas pertenecen a M. El propósito de construir un árbol alternante con raíz r es ya sea encontrar un vértice interior expuesto -y por tanto un camino de ampliación- o determinar que tal vértice no existe.

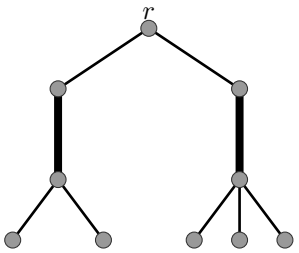

Figura 12: Árbol alternante.

Supongamos que la capa 2i-1 ya se ha construido. Si ningún vértice en esta capa es expuesto, la siguiente capa es fácil de construir: simplemente agregamos

el vértice w =compañero(v) y la arista $\{v, w\}$ a T para cada vértice v en la capa $2i$-1. Por el contrario, pueden surgir dificultades cuando construimos la subsiguiente capa de vértices interiores. Sea x un vértice en la capa $2i$, y sea $y \neq$compañero(x) un vértice adyacente a x. Hay 4 casos posibles:

- *Caso 1:* y es expuesto (y todavía no contenido en T). Entonces hemos encontrado un camino de ampliación.
- *Caso 2:* y no es expuesto, y ni y ni compañero(y) están contenidos en T. Entonces ponemos y en la capa $2i$+1 y compañero(y) en la capa $2i$+2.
- *Caso 3:* y está ya contenido en T como un vértice interior. Notar que la adición de la arista $\{x, y\}$ a T crearía un ciclo de longitud par en T; ver Fig 13. Como T ya contiene un camino alternante de r al vértice interior y, tales aristas deben ser redundantes para nuestros propósitos. Veremos más adelante que esto es cierto, así que podemos ignorar este caso.
- *Caso 4:* y está ya contenido en T como un vértice exterior. Tener en cuenta que la adición de la arista $\{x, y\}$ a T crearía un ciclo de longitud impar $2k + 1$ en T para el cual k aristas pertenecen a M; ver Fig. 14.

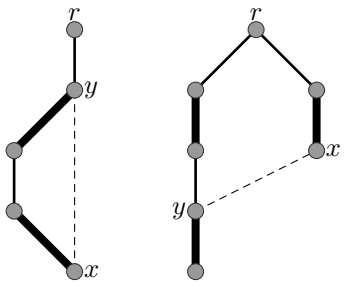

Figura 13: Caso 3.

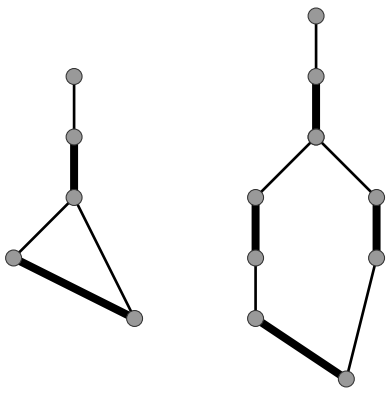

Figura 14: Caso 4.

Los ciclos del caso 4 se llaman *flores*; estas flores -las cuales por supuesto no pueden ocurrir en el caso bipartito- causan las dificultades aludidas anteriormente: las aristas formando una flor con el árbol construido hasta ahora no pueden sencillamente ignorarse. Consideremos por ejemplo la flor exhibida en la figura 15. Cada uno de los vértices a, b, c, d, e, f pueden ser alcanzados a través de dos caminos alternantes diferentes con vértice inicial r: para el camino uno, el vértice será un vértice exterior, y para el camino 2 será un vértice interior. Por ejemplo, a es un vértice interior con respecto a (r, a), y un vértice exterior con respecto a (r, b, d, e, f, c, a). Ahora supongamos que existe una arista $\{a, x\}$ para la cual x es expuesto; entonces (r, b, d, e, f, c, a, x) será un camino de ampliación. Si simplemente omitimos la arista $\{f, c\}$ cuando construimos T, es muy posible que no encontremos ningún camino de ampliación por nuestro enfoque (aunque tal camino exista); el grafo G en la figura 15 proporciona un ejemplo sencillo de este fenómeno.

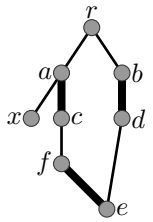

Figura 15: Flor.

Las dificultades que surgen del caso 4 se resuelven en el algoritmo de Edmonds por *encogimiento de flores* a vértices individuales. En un punto posterior del algoritmo, las flores que fueron encogidas anteriormente pueden ser *expandidas* de nuevo.

7.2.3. El algoritmo de Edmonds.

Sea $G = (V, E)$ un grafo conexo con un apareamiento inicial M. Comenzamos por construir un árbol alternante T con raíz r, como se describió anteriormente. Las aristas $\{x, y\}$ cerrando un ciclo de longitud par (Caso 3) serán ignoradas. Siempre que encontremos una arista xy cerrando una flor B (Caso 4), paramos la construcción de T y *encogemos* la flor B.
Formalmente, esta operación se describe como la *contracción* de G con respecto a B a un grafo más pequeño G/B definido de la siguiente manera:

- El conjunto de vértices de G/B es $V/B = (V \backslash B) \cup \{b\}$, donde b es un nuevo vértice.
- El conjunto de aristas E/B de G/B se deduce de E por primero remover todas las aristas $\{u, v\} \in E$ con $u \in B$ o $v \in B$ y luego agregar una arista $\{u, b\}$ para todos aquellos $u \in V \backslash B$ que son adyacentes en G para al menos un vértice de B.

Para distinguirlo de los vértices originales, al nuevo vértice b le llamaremos *pseudovértice* de G/B. Ahora abordemos el asunto sobre cómo el encogimiento de una flor B afecta la construcción de T. Tengamos presente que el apareamiento M de G induce en una forma natural un apareamiento M/B de G/B. Cuando encontramos una arista $\{x, y\}$ cerrando una flor B, sólo conocemos dos vértices de B, llamados x e y. La flor entera puede determinarse por el seguimiento de los caminos desde x e y a la raíz r en T; al primer vértice común w de estos dos caminos le llamaremos la *base* de la flor B. También, w es un punto exterior de T. Entonces B es la unión de $\{x, y\}$ con los dos caminos P_{wx} y P_{wy} desde la base w a los vértices x e y, respectivamente. La omisión de estos dos caminos desde T y el reemplazo de la base w por el pseudovértice b produce un árbol alternante T/B para G/B con respecto al apareamiento M/B. Ahora procedemos con nuestra construcción en G/B, usando T/B; aquí el siguiente vértice exterior que examinamos es el pseudovértice b. Otras flores podrán surgir, en cuyo caso tendremos que hacer una serie de encogimientos.

El proceso de encogimiento para flores trabaja siempre correctamente. Bastará mostrar que el grafo $G' = G/B$ resultante de contraer la primera flor B que encontramos contiene un camino de ampliación con vértice inicial r (o b, si r estuviera contenido en B) si y sólo sí el grafo original G contiene un tal camino. (En el caso que $r \in B$, el vértice r de G se reemplaza con el pseudovértice b en G' por el proceso de encogimiento.) Luego el resultado general se sigue por inducción sobre el número de flores encontradas. Los dos lemas siguientes demuestran nuestra afirmación.

Lema 7.2.3.1. *Sea G un grafo conexo con un apareamiento M, y sea r un vértice expuesto con respecto a M. Supongamos que, durante la construcción de un árbol alternante T con raíz r (según las reglas descritas anteriormente), la primera flor B es encontrada cuando se examina la arista $e = xy$; aquí x denota al vértice exterior que el algoritmo examina esta vez, e y es otro vértice exterior de T. Sea w la base de B, y consideremos el grafo contraído $G' = G/B$ que resulta del reemplazo de B con el pseudovértice b. Si G contiene un camino de ampliación con respecto a M iniciando en r, entonces G' contiene un camino de ampliación con respecto al apareamiento inducido $M' = M/B$ iniciando en r (o en b, cuando $r \in B$).*

Demostración. Sea P un camino de ampliación en G con respecto a M iniciando en r, y denotemos el vértice final de P por s. Como todos los vértices de P son saturados excepto r y s, podemos suponer realmente que r y s son los únicos vértices expuestos de G. (De lo contrario, podemos eliminar todos los vértices expuestos más junto con las aristas incidentes con ellos de G). Supongamos primero que P y B no tienen ningún vértice en común. Entonces la afirmación es obvia: P es también un camino de ampliación en G' con respecto a M'. Así podemos suponer que P y B no son disjuntos. Distinguimos 2 casos:

Caso 1: La raíz r está contenida en la flor B, así que r es la base de B. Trazamos P desde r a s. Sea q el primer vértice de P que no está contenido en B, y denotemos su predecesor sobre P por p; por tanto, p es el último vértice de P contenido en B. (Tener en cuenta que $p = r$ es posible.) Entonces la arista $\{p, q\}$ no está contenida en B. Denotar la parte de P desde r a p por P_1, y la

parte desde q a s por P_2; ver figura 16. Claramente, $P' = b - q \overset{P_2}{-} s$ es un camino de ampliación en G' con respecto a M'.

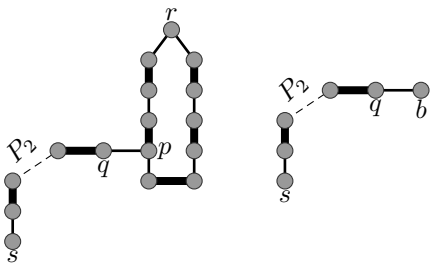

Figura 16: Caso 1 del lema 7.2.3.1.

Caso 2: La raíz r de T no está contenida en B, así que la base de B es un vértice exterior $w \neq r$. Denotemos al camino alternante de longitud par desde r a w en T por S; S se suele llamar el *tallo* de la flor B. Esta vez no es muy obvio cómo el camino de ampliación P con respecto a M interactúa con la flor B. Por consiguiente usaremos un truco el cual nos permite reducir este caso al Caso 1: reemplazamos M por el apareamiento $M_1 = M \oplus S$, que tiene la misma cardinalidad. Entonces w y s son los únicos vértices expuestos con respecto a M_1, de modo que la flor B (que no ha sido cambiada) tiene base w si empezamos construyendo un árbol alternante en el vértice w; ver figura 17. Así la situación para M_1 es realmente como en el Caso 1.
Como existe un camino de ampliación en G con respecto a M, M no fué máximo. Por tanto, M_1 no es máximo; por Teorema 7.2.1.1, existe un camino de ampliación P_1 con respecto a M_1 en G. Según el caso 1, esto implica la existencia de un camino de ampliación P_1' en G' con respecto a M_1/B, de modo que el apareamiento M_1/B no es máximo. De ello se sigue que el apareamiento M/B de G' (el cual tiene la misma cardinalidad que el apareamiento M_1/B) no es máximo tampoco; por tanto debe haber un camino de ampliación en G' con respecto a M/B. Como r y s son los únicos vértices expuestos en G', la aseveración se cumple.

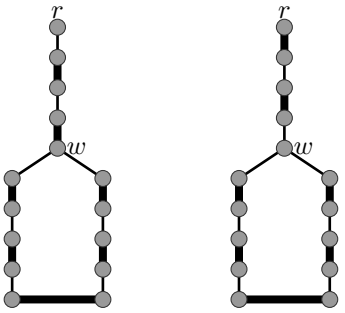

Figura 17: Caso 2 del lema 7.2.3.1.

□

Lema 7.2.3.2. *Sea G un grafo conexo con un apareamiento M, y supongamos que r es un vértice expuesto con respecto a M. Además, sea B una flor con base w y $G' = G/B$ el grafo contraído donde B se reemplaza con el pseudovértice b. Si G' contiene un camino de ampliación con respecto a $M' = M/B$ iniciando en r (o en b, cuando $r \in B$), entonces existe un camino de ampliación en G con respecto a M iniciando en r.*

Demostración. Supongamos primero que el camino de ampliación P' en G' no contiene al pseudovértice b. Entonces P' es también un camino en G, y la afirmación es clara. Por tanto supongamos $b \in P'$. Consideremos solamente el caso $r \notin B$; el caso $r \in B$ es similar y en realidad aún más simple. Sea w la base de B. Supongamos primero que la distancia desde r a b en P' es par. Entonces P' tiene la forma

$$P' : r \overset{P_1}{-} p - b - q \overset{P_3}{-} s,$$

donde P_1 es la parte de P' desde r a p=*compañero(b)*. Ahora q debe ser adyacente a un vértice $q' \in B$. Denotar el camino alternante de longitud par en B desde w hasta q' por P_2; tener en cuenta que $P_2 = \emptyset$ si $w = q'$. Luego

$$P : r \overset{P_1}{-} p - w \overset{P_2}{-} q' - q \overset{P_3}{-} s$$

es un camino de camino de ampliación en G con respecto a M, donde P_3 denota la parte de P' desde q a s; ver figura 18.
Finalmente, si la distancia desde r a b en P' es impar, la distancia desde s a b en P' debe ser par. Luego simplemente cambiamos los roles de r y s y procedemos como antes.

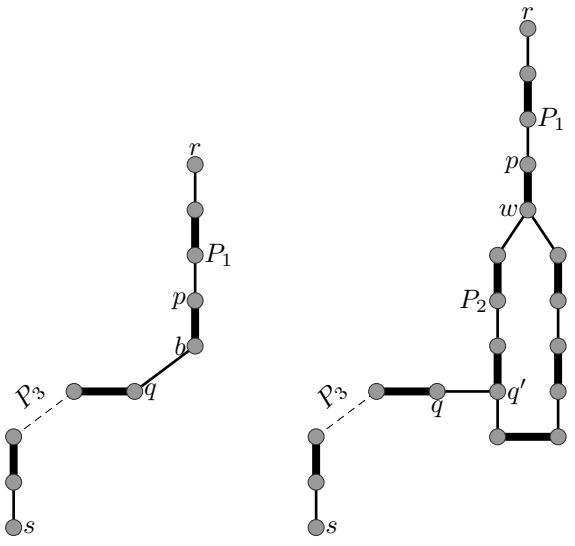

Figura 18: Prueba de lema 7.2.3.2.

□

7.2.4. Algoritmo de Edmonds. Pseudocódigo.

Sea $G = (V, E)$ un grafo sobre el conjunto de vértices $V = \{1, ..., n\}$, dado por la lista de adyacencia A_v. El algoritmo construye un apareamiento máximo M de G descrito por un arreglo *compañero* y determina el número de vértices expuestos de G con respecto a M.

El procedimiento principal MAXMATCH usa el procedimiento INMATCH 7.2.2.1 para determinar un apareamiento inicial, así como otros tres procedimientos auxiliares: BLOSSOM7.2.4.2, CONTRACT7.2.4.3 y AUGMENT7.2.4.4. Estos tres procedimientos forman después MAXMATCH. En MAXMATCH, la función d describe la posición de los vértices en el árbol alternante actual T con raíz r: vértices que aún no están en el árbol tienen $d(y) = -1$; mientras que para todos los otros vértices, $d(y)$ es la distancia entre y y la raíz r de T. En particular, vértices y para los cuales $d(y)$ es impar son vértices interiores, y vértices y para los que $d(y)$ es par son vértices exteriores. Los vértices exteriores se mantienen en una cola prioritaria Q con función prioridad d.
La construcción del árbol alternante T se continúa siempre desde el primer vértice activo de Q. Inicialmente, todos los vértices son activos. Los vértices se vuelven inactivos si están contenidos en una flor que se contrae. El examen de los vecinos de un vértice exterior x se hace como el descrito en 7.2.2, y las flores se contraen inmediatamente cuando son descubiertas.
Ya que más tarde necesitaremos la lista original de adyacencia A_v de G para la expansión de los caminos de ampliación, estas listas no deben cambiarse a lo largo de todo el algoritmo. Por tanto usaremos nuevas listas de adyacencia $CA(v)$ para describir a los grafos contraídos. Los vértices de una flor contraída no se eliminan realmente del grafo, sólo se declaran *inactivos*; por supuesto, originalmente todos los vértices son activos. Para lograr esto, se usa una función booleana a: un vértice v permanece activo con tal que $a(v)$ tenga el valor *verdadero*.
Finalmente, las variables booleanas *aug* y *cont* sirven para controlar el ciclo: la variable *aug* tiene el valor $falso$ hasta que se encuentre un camino de ampliación; y durante el examen de un vértice exterior x, *cont* tiene valor $falso$ hasta que una flor se encuentre y se contraiga.

Procedimiento 7.2.4.1.

MAXMATCH(G;compañero,nrex)

```
(01) INMATCH(G;compañero,nrex);
(02) r ← 0;
(03) while nrex≥2 and r < n do
(04)     r ← r + 1;
(05)     if compañero(r)=0
(06)     then Q ← ∅, aug←falso; m ← 0;
(07)         for v ∈ V do
(08)             p(v) ← 0; d(v) ← −1; a(v) ←verdadero;
(09)             CA(v) ← A_v
(10)         od
(11)         d(r) ← 0; anexar r a Q;
(12)         while aug=falso and Q ≠ ∅ do
(13)             Remover el primer vértice x de Q;
(14)             if a(x)=verdadero
(15)             then cont ← falso
(16)                 for y ∈ CA(x) do u(y) ← falso od
(17)                 repeat
(18)                     escoger y ∈ CA(x) con u(y)=falso; u(y) ←verdadero;
(19)                     if a(y) = verdadero
(20)                     then if d(y) ≡ 0 (mod2)
(21)                                 then m ← m + 1;
(22)                                 BLOSSOM (x, y; B(m), w(m));
(23)                                 CONTRACT (B(m), m, w);
(24)                             else if d(y) = −1
(25)                                 then if compañero(y)=0
(26)                                     then AUGMENT(x, y)
(27)                                     else z ← compañero(y);
(28)                                     p(y) ← x; d(y) ← d(x) + 1;
(29)                                     p(z) ← y; d(z) ← d(y) + 1;
(30)                                     insertar z con prioridad d(z) en Q
(31)                                     fi
(32)                                 fi
(33)                         fi
(34)                     fi
(35)                 until u(y) = verdadero para todo y ∈ CA(v) or aug = verdadero
                     or cont = verdadero
(36)             fi
(37)         od
(38)     fi
(39)od
```

El procedimiento BLOSSOM construye una flor B con base w. Este procedimiento es llamado por MAXMATCH cuando se descubre un vértice exterior adicional y en $CA(x)$ durante el examen de un vértice exterior x (Caso 4). Fig. 14

Procedimiento 7.2.4.2.

BLOSSOM$(x, y; B, w)$

(01) $P \leftarrow \{x\};\ P' \leftarrow \{y\};\ u \leftarrow x; v \leftarrow y;$
(02) **while**$(u \neq r)$ **do**
(03) $\quad P \leftarrow P \cup \{p(u)\};\ u \leftarrow p(u)$
(04) **od**
(05) **while**$(v \neq r)$ **do**
(06) $\quad P' \leftarrow P' \cup \{p(v)\};\ v \leftarrow p(v)$
(07) **od**
(08) $S \leftarrow P \cap P';$
(09) sea w el elemento de S para el cual $d(w) \geq d(z)$ para todo $z \in S;$
(10) $B \leftarrow ((P \cup P') \backslash S) \cup \{w\}$

El procedimiento CONTRACT se usa para la contracción de una flor B, y las listas de adyacencia $CA(v)$ para el grafo contraído se actualizan consecuentemente.

Procedimiento 7.2.4.3.

CONTRACT(B,m,w)

(01) $b \leftarrow n + m;\ a(b) \leftarrow verdadero;$
(02) $p(b) \leftarrow p(w);\ d(b) \leftarrow d(w);$ compañero$(b) \leftarrow$compañero$(w);$
(03) insertar b en Q con prioridad $d(b);$
(04) $CA(b) \leftarrow \bigcup_{z \in B} CA(z);$
(05) **for** $z \in CA(b)$ **do** $CA(z) \leftarrow CA(z) \cup \{b\}$ **od**
(06) **for** $z \in B$ **do** $a(z) \leftarrow$ falso **od**
(07) **for** $z \in CA(b)$ **do**
(08) $\quad$ **if** $a(z)$=verdadero **y** $p(z) \in B$
(09) $\quad$ **then** $d(z) \leftarrow d(b) + 1;\ p(z) \leftarrow b;$
(10) $\quad\quad d($compañero$(z)) \leftarrow d(z)+1;$
(11) $\quad$ **fi**
(12) **od**
(13) cont $\leftarrow$ verdadero

El procedimiento final AUGMENT construye un camino de ampliación y cambia el apareamiento M cuando encuentra un vértice expuesto y durante la construcción del árbol alternante T; esto ocurre en paso 25-26 de MAXMATCH.

Procedimiento 7.2.4.4.

AUGMENT(x, y)

(01) $P \leftarrow \{y, x\};\ v \leftarrow x;$
(02) **while** $p(v) \neq 0$ **do**
(03) $\quad P \leftarrow P \cup \{p(v)\}; v \leftarrow p(v)$
(04) **od**
(05) **while** exista $b \in P$ con $b > n$ **do**
(06) $\quad$ Escoger el más grande $b \in P$ con $b > n$;
(07) $\quad B \leftarrow B(b-n);\ w \leftarrow w(b-n);\ z \leftarrow$ compañero(w);
(08) $\quad$ Sea q el vecino de b en P que es diferente de z;
(09) $\quad$ Escoger algún $q' \in B \cap CA(q)$;
(10) $\quad$ Determinar el camino alternante B' de longitud par en B desde w a q';
(11) $\quad$ Reemplazar b por w en P;
(12) $\quad$ Insertar B' dentro de P entre w y q
(13) **od**
(14) $u \leftarrow y;\ v \leftarrow x;$
(15) **while** $v \neq r$ **do**
(16) $\quad z \leftarrow$ compañero(v); compañero$(v) \leftarrow u$; compañero$(u) \leftarrow v$;
(17) $\quad u \leftarrow z$; sea v el sucesor de z en P
(18) **od**
(19) compañero$(v) \leftarrow u$; compañero$(u) \leftarrow v$;
(20) nrex$\leftarrow$nrex-2; aug$\leftarrow$verdadero

8. ALGORITMOS Y SU MODO DE USO

8.1. Sobre grafos bipartitos

8.1.1. Algoritmo en C++ para encontrar apareamiento máximo en grafo bipartito.

```
#include<iostream>
#include<stdio.h>
using namespace std;

#define nver 200
typedef int matriz[nver][nver];
struct grafbip{int nve1,nve2,mad[nver][nver];};
struct grafo {int nve,mad[nver][nver];};
struct redflujo{grafo g; int cap[nver][nver];int s,t;};


void leer_grafo_bipartito(grafbip& grab)
{
    grab.nve1=6;
    grab.nve2=5;
    for (int i=1;i<=grab.nve1+grab.nve2;i++)
       for (int j=1;j<=grab.nve1+grab.nve2;j++)
          grab.mad[i][j]=0;

        grab.mad[1][8]=1;
        grab.mad[1][9]=1;
        grab.mad[2][7]=1;
        grab.mad[2][11]=1;
        grab.mad[3][8]=1;
        grab.mad[4][8]=1;
        grab.mad[4][11]=1;
        grab.mad[5][8]=1;
        grab.mad[5][10]=1;
        grab.mad[6][8]=1;
        grab.mad[6][11]=1;
}

void construir_red_inicial(grafbip grab, redflujo& ginic)
{
        int m1=grab.nve1;
        int m2=grab.nve2;
        ginic.g.nve=m1+m2+2;
        int m=ginic.g.nve;
    for (int i=1;i<=m;i++)
       for (int j=1;j<=m;j++)
           {
                   ginic.g.mad[i][j]=0;
           ginic.cap[i][j]=0;
           }

        for (int j=2;j<=m1+1;j++)
           ginic.g.mad[1][j]=1;
```

```
    for (int i=m1+2;i<=m1+m2+1;i++)
           ginic.g.mad[i][m1+m2+2]=1;

        for (int i=2;i<=m1+1;i++)
       for (int j=m1+2;j<=m1+m2+1;j++)
                   ginic.g.mad[i][j]=grab.mad[i-1][j-1];
        for (int i=1;i<=m;i++)
           for (int j=1;j<=m;j++)
              ginic.cap[i][j]=ginic.g.mad[i][j];
        ginic.s=1;
        ginic.t=m;
}

void imprimir_grafo_inicial(grafbip grab)
{
    cout<<"Grafo bipartito inicial:"<<endl<<endl;
    for (int i=1;i<=grab.nve1;i++)
           for (int j=grab.nve1+1;j<=grab.nve1+grab.nve2;j++)
              if (grab.mad[i][j]==1) cout<<i<<"-"<<j<<endl;
    cout<<endl;
}

void auxnet(redflujo ginic, redflujo& gauxi, matriz flu,int& d, bool& max1 )
{
        int i,j,k,m,r,u,v,vacio,vdp[nver];
        matriz vi;
        gauxi.s=ginic.s;
        gauxi.t=ginic.t;
        gauxi.g.nve=ginic.g.nve;

        m=ginic.g.nve;

        for (r=1;r<=m;r++)
        {
                vdp[r]=0;
                for (k=1;k<=m;k++)
                        gauxi.g.mad[r][k]=0;
        }

        i=0;vi[i][ginic.s]=1;

        for (k=1;k<=m;k++)
                vdp[k]=vi[0][k];

        do
        {
                i++;
                for (k=1;k<=m;k++)
                        vi[i][k]=0;

                for (k=1;k<=m;k++)
                        if (vi[i-1][k]==1)
                        {
                                v=k;
```

```
                                for(j=1;j<=m;j++)
                                        if (ginic.g.mad[v][j]==1)
                                        {
                                                u=j;
                                                if(vdp[u]==0 && flu[v][u]<ginic.cap[v][u])
                                                {
                                                        gauxi.g.mad[v][u]=1;
                                                        vi[i][u]=1;
                                                gauxi.cap[v][u]=ginic.cap[v][u]-flu[v][u];
                                                }
                                        }
                                for (j=1;j<=m;j++)
                                        if(ginic.g.mad[j][v]==1)
                                        {
                                                u=j;
                                                if(vdp[u]==0 && flu[u][v]!=0)
                                                {
                                                        gauxi.g.mad[v][u]=1;
                                                        vi[i][u]=1;
                                                        gauxi.cap[v][u]=flu[u][v];
                                                }
                                        }
                        }

                if (vi[i][ginic.t]==1)
                {
                   for (j=1;j<=m;j++)
                   {
                          if (vi[i][j]==1)
                          {
                              for (r=1;r<=m;r++)
                                 if(ginic.t!=r && gauxi.g.mad[j][r]==1) gauxi.g.mad[j][r]=0;
                          }
                   }
                }

                for (j=1;j<=m;j++)
                        if (vi[i][j]==1) vdp[j]=1;

                vacio=0;
                for (j=1;j<=m;j++)
                        vacio=vacio+vi[i][j];
        }
        while (vdp[ginic.t]==0 && vacio!=0);

        if (vdp[ginic.t]==1){max1=0;d=i;}
        else max1=1;
        return;
}

void blockflow01(redflujo gauxi, int d, matriz& ge)
{
    bool ele;
        int i,j,r,v,tl,u,w;
```

```
    int ind[nver],L[nver];
    int m=gauxi.g.nve;

    for (i=1;i<=m;i++)
    {
        L[i]=0;
            ind[i]=0;
    }
tl=0;
    for (i=1;i<=m;i++)
       for (j=1;j<=m;j++)
       {
               ge[i][j]=0;
               if (gauxi.g.mad[i][j]==1) ind[j]++;
       }
    do
    {
        v=gauxi.t;
        for (i=d;i>=1;i--)
            {
                ele=0;
                j=0;
                while (ele==0 && j<=m)
                {
                    j++;
                        if (gauxi.g.mad[j][v]==1)
                            {
                                u=j;
                                    gauxi.g.mad[u][v]=0;
                                    ind[v]--;
                                    ge[u][v]=1;
                                    ele=1;
                            }
                    }
                    if (ind[v]==0)
                    {
                        L[tl+1]=v;
                            tl++;
                            while (tl!=0)
                            {
                                w=L[1];
                                for (r=1;r<=tl;r++)
                                       L[r]=L[r+1];
                                tl--;
                                for (r=1;r<=m;r++)
                                {
                                        if (gauxi.g.mad[w][r]==1)
                                        {
                                            gauxi.g.mad[w][r]=0;
                                            ind[r]--;
                                            if (ind[r]==0) {L[tl+1]=r;tl++;}
                                        }
                                }
                            }
```

```
                    }
                        v=u;
                }
    }
    while (ind[gauxi.t]!=0);
}

void augment(redflujo ginic, redflujo gauxi, matriz ge, matriz& fl)
{
    int i,j,m;
    m=ginic.g.nve;
    for (i=1;i<=m;i++)
           for (j=1;j<=m;j++)
           {
                   if (ginic.g.mad[i][j]==1)
                   {
                           if (gauxi.g.mad[i][j]==1) fl[i][j]=fl[i][j]+ge[i][j];
                           if (gauxi.g.mad[j][i]==1) fl[i][j]=fl[i][j]-ge[j][i];
                   }
           }
}

void imprimir_apareamiento(grafbip grab, matriz flu)
{
    int m1=grab.nve1;
    int m2=grab.nve2;
    cout<<"Apareamiento maximal:"<<endl<<endl;
        for (int i=2;i<=m1+1;i++)
           for (int j=m1+2;j<=m1+m2+1;j++)
          if (flu[i][j]==1) cout<<i-1<<"-"<<j-1<<endl;
    cout<<endl;
}

int main()
{
    int d;
    matriz flu,gbf;
    bool max0;
    grafbip grabi;
    redflujo gin,gau;

    leer_grafo_bipartito(grabi);

    construir_red_inicial(grabi,gin);

    int nv=gin.g.nve;
    for (int i=1;i<=nv;i++)
           for (int j=1;j<=nv;j++)
              flu[i][j]=0;
    do
    {
        auxnet(gin,gau,flu,d,max0);
        if (max0==0)
                {
```

```
                    blockflow01(gau,d,gbf);
                        augment(gin, gau, gbf,flu);
                }
        }
        while (max0==0);

        imprimir_grafo_inicial(grabi);
        imprimir_apareamiento(grabi, flu);

        fflush(stdin);
        getchar();
        return(0);

}
```

8.1.2. Modo de uso de algoritmo para grafos bipartitos

Ejemplo 8.1.2.1. *Haciendo uso del programa diseñado en C++ que encuentra apareamiento máximo en grafo bipartito, encontrar el apareamiento máximo del siguiente grafo.*

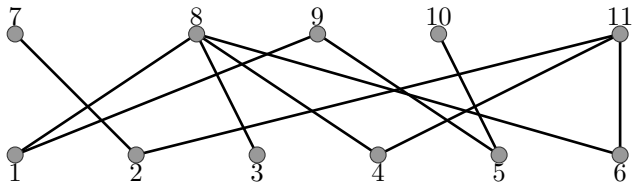

Figura 19: Grafo bipartito.

Solución: El programa inicia con una función llamada *leer_grafo_bipartito* y tiene como valores de entrada números enteros. El formato de entrada es el siguiente:
grab.nve1=m; grab.nve2=n;
donde m y n son el número de vértices de la partición 1 y 2 de vértices del grafo, respectivamente. La lista de adyacencia tiene el aspecto $grab.mad[i][j] = 1$; donde i y j son vértices extremos de la arista $\{i, j\}$. El valor 1 es un valor constante para todas las aristas $\{i, j\}$ del grafo.
El resultado para el ejemplo de la figura 19 es el siguiente:

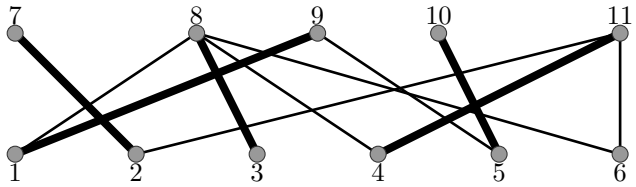

Figura 20: Apareamiento máximo en grafo bipartito.

8.2. Sobre grafos generales

8.2.1. Algoritmo en C++ para encontrar apareamiento máximo en grafo general.

```
#include<iostream>
#include<stdlib.h>
using namespace std;

#define nver 200
typedef int lista[nver];
typedef lista matri[nver];
struct grafo {int nve;matri lad;};

        grafo graf;
        lista w,d,p,mate,q;
        matri b,ca;
        int i,j,r,n,m,nrex,x,z,v,tlq;
        bool cont,aug, u[nver],a[nver],termina, condicion;

void leer_grafo_inicial(grafo& gra)
{
    int i,j;
    gra.nve=24;
    for (i=1;i<=gra.nve;i++)
       for (j=1;j<=gra.nve;j++)
          gra.lad[i][j]=0;


        gra.lad[1][1]=2;
        gra.lad[1][2]=3;
        gra.lad[1][3]=4;

        gra.lad[2][1]=1;
        gra.lad[2][2]=5;
        gra.lad[2][3]=6;

        gra.lad[3][1]=1;
        gra.lad[3][2]=4;
        gra.lad[3][3]=19;

        gra.lad[4][1]=1;
        gra.lad[4][2]=3;
        gra.lad[4][3]=7;

        gra.lad[5][1]=2;
        gra.lad[5][2]=6;
        gra.lad[5][3]=10;

        gra.lad[6][1]=2;
        gra.lad[6][2]=5;
        gra.lad[6][3]=22;

        gra.lad[7][1]=4;
        gra.lad[7][2]=8;
```

```
gra.lad[7][3]=11;

gra.lad[8][1]=7;
gra.lad[8][2]=9;
gra.lad[8][3]=11;

gra.lad[9][1]=8;
gra.lad[9][2]=10;
gra.lad[9][3]=12;

gra.lad[10][1]=5;
gra.lad[10][2]=9;
gra.lad[10][3]=12;

gra.lad[11][1]=7;
gra.lad[11][2]=8;
gra.lad[11][3]=13;

gra.lad[12][1]=9;
gra.lad[12][2]=10;
gra.lad[12][3]=14;

gra.lad[13][1]=11;
gra.lad[13][2]=15;
gra.lad[13][3]=16;

gra.lad[14][1]=12;
gra.lad[14][2]=17;
gra.lad[14][3]=18;

gra.lad[15][1]=13;
gra.lad[15][2]=16;
gra.lad[15][3]=20;

gra.lad[16][1]=13;
gra.lad[16][2]=15;
gra.lad[16][3]=17;

gra.lad[17][1]=14;
gra.lad[17][2]=16;
gra.lad[17][3]=18;

gra.lad[18][1]=14;
gra.lad[18][2]=17;
gra.lad[18][3]=21;

gra.lad[19][1]=3;
gra.lad[19][2]=20;
gra.lad[19][3]=23;

gra.lad[20][1]=15;
gra.lad[20][2]=19;
gra.lad[20][3]=23;
```

```
        gra.lad[21][1]=18;
        gra.lad[21][2]=22;
        gra.lad[21][3]=24;

        gra.lad[22][1]=6;
        gra.lad[22][2]=21;
        gra.lad[22][3]=24;

        gra.lad[23][1]=19;
        gra.lad[23][2]=20;
        gra.lad[23][3]=24;

        gra.lad[24][1]=21;
        gra.lad[24][2]=22;
        gra.lad[24][3]=23;

}


int grado(grafo gra,int v)
{
        int i,n,grad;
        n=gra.nve;
        grad=0;
        for (i=1;i<=n;i++)
                if(gra.lad[v][i]>0) grad++;
        return(grad);
}


void inmatch(grafo gra, lista& mate, int& nrex)
{
        int i,j,k,g;
        bool enc;
        nrex=n;
        for (i=1;i<=n;i++)
                mate[i]=0;
        for (k=1;k<=n-1;k++)
                if (mate[k]==0)
                        {
                        g=grado(gra,k);
                        j=1;
                        enc=false;
                        while (j<=g && enc==false)
                        {
                                if (mate[gra.lad[k][j]]==0)
                                {
                                mate[gra.lad[k][j]]=k;
                                mate[k]=gra.lad[k][j];
                                nrex=nrex-2;
                                enc=true;
                                }
                                j++;
                        }
```

```
                        }
}


int tlista(lista q, int n)
{
        int i;
        i=1;
        while (q[i]!=0 && i<=n-1)
                i++;
        if (q[i]==0) return(i-1);
        else return(i);
}


void eplista(lista q, int n)
{
        int i,k;
        k=tlista(q,n);
        for (i=1;i<=k-1;i++)
                q[i]=q[i+1];
        q[k]=0;
}

void blossom(int x, int y, lista& b, int& w)
{
        int i,j,dw,tlb,tls,tlpe,tlpp,u,v;
        bool repetido;
        lista s,pe,pp;

        pe[1]=x;tlpe=1;
        pp[1]=y;tlpp=1;

        u=x;
        v=y;

        while(u!=r)
        {
                pe[tlpe+1]=p[u];tlpe++;
                u=p[u];
        }

        while (v!=r)
        {
                pp[tlpp+1]=p[v];tlpp++;
                v=p[v];
        }

        tls=0;
        for (i=1;i<=tlpe;i++)
        {
                j=1;
                repetido=false;
                do
```

```
        {
                if (pe[i]==pp[j])
                {
                        s[tls+1]=pe[i];
                        tls++;
                        repetido=true;
                }
                j++;
        }
        while(j<=tlpp && repetido==false);
}

dw=0;w=0;
for (i=1;i<=tls;i++)
        if (d[s[i]]>=dw)
        {
                dw=d[s[i]];
                w=s[i];
        }

b[1]=w;tlb=1;

for (i=1;i<=tlpe;i++)
{
        repetido=false;
        j=1;
        do
        {
                if (pe[i]==pp[j]) repetido=true;
                j++;
        }
        while(j<=tlpp && repetido==false);

        if (repetido==false)
        {
                b[tlb+1]=pe[i];
                tlb++;
        }

}

for (i=1;i<=tlpp;i++)
{
        repetido=false;
        j=1;
        do
        {
                if (pp[i]==pe[j]) repetido=true;
                j++;
        }
        while(j<=tlpe && repetido==false);
        if (repetido==false)
        {
        b[tlb+1]=pp[i];
```

```
                tlb++;
                }
        }

        cout<<"Flor numero "<<m<<endl<<endl;
        for (i=1;i<=tlb;i++)
                cout<<"B("<<m<<","<<i<<")= "<<b[i]<<endl;
        system("pause");
}

void contract(lista b,int w)
{
        int i,j,t,pvb,r0,r1;

        pvb=n+m;
        a[pvb]=true;
        p[pvb]=p[w];
        d[pvb]=d[w];
        mate[pvb]=mate[w];
        q[tlista(q,n+m)+1]=pvb;

        r0=tlista(b,n+m);
        t=0;
        for (i=1;i<=r0;i++)
        {
                r1=tlista(ca[b[i]],n+m);
                for (j=1;j<=r1;j++)
                {
                        bool agregar=true;
                        for (int e=1;e<=t;e++)
                                if (ca[pvb][e]==ca[b[i]][j]) {agregar=false;}
                        if (agregar==true)
                        {
                                t++;
                                ca[pvb][t]=ca[b[i]][j];
                        }
                }
        }

        r0=tlista(ca[pvb],n+m);
        for (i=1;i<=r0;i++)
                {
                        r1=tlista(ca[ca[pvb][i]],n+m);
                        ca[ca[pvb][i]][r1+1]=pvb;
                }

        r0=tlista(b,n+m);
        for (i=1;i<=r0;i++)
                a[b[i]]=false;

        r1=tlista(ca[pvb],n+m);
        for (i=1;i<=r1;i++)
        {
                if (a[ca[pvb][i]]==true)
```

```
                {
                        j=1;
                        bool salir=false;
                        while (j<=r0 && salir)
                        {
                                if (b[j]==ca[pvb][i])
                                {
                                        d[ca[pvb][i]]=d[pvb]+1;
                                        p[ca[pvb][i]]=pvb;
                                        d[mate[ca[pvb][i]]]=d[ca[pvb][i]]+1;
                                        salir=true;
                                }
                                else j++;
                        }
                }
        }
        cont=true;


        cout<<"Listas de Adyacencia"<<endl<<endl;
        for (i=1;i<=n+m;i++)
        {
                cout<<"ca("<<i<<")= ";
                for (j=1;j<=n+m;j++)
                        cout<<ca[i][j]<<" ";
                cout<<endl;
        }
        cout<<endl<<"Estado de los Vertices"<<endl<<endl;
        for (i=1;i<=n+m;i++)
                cout<<"a("<<i<<")= "<<a[i]<<endl;
        system("pause");
}


void intersec(lista a, lista b, lista& c)
{
        int i,j,tlc,tla,tlb;
        bool repetido;

        tlc=0;
        tla=tlista(a,n+m);
        tlb=tlista(b,n+m);
        for (i=1;i<=tla;i++)
        {
                j=1;
                repetido=false;
                do
                {
                        if (a[i]==b[j])
                        {
                                c[tlc+1]=a[i];
                                tlc++;
                                repetido=true;
                        }
```

```
                        j++;
                }
                while(j<=tlb && repetido==false);
        }
}

void augment(int x, int y)
{
        int i,j,tlpe,v,u,z,we,max0,pos,posqu,qu,qp,k0,u0;
        bool repetido,exito;
        lista nl,pe,be,bep;

        for(i=1;i<=n+m;i++)
                {
                pe[i]=0;
                nl[i]=0;
                be[i]=0;
                bep[i]=0;
                }
        pe[1]=y;
        pe[2]=x;
        tlpe=2;
        v=x;
        while (p[v]!=0)
        {
                pe[tlpe+1]=p[v];
                v=p[v];
                tlpe++;
        }

        max0=0;
        pos=0;
        for (i=1;i<=tlpe;i++)
                if(pe[i]>n && max0<pe[i]) {max0=pe[i];pos=i;}

        while(max0>0)
        {
                for (i=1;i<=n+m;i++)
                        be[i]=b[max0-n][i];
                we=w[max0-n];
                z=mate[we];
                if (pe[pos-1]==z) {qu=pe[pos+1];posqu=pos+1;}
                else {qu=pe[pos-1];posqu=pos-1;}
                intersec(be,ca[qu],nl);
                if (tlista(nl,n+m)>0) qp=nl[1];

                lista pp0,pe0,usad;
                for (i=1;i<=n+m;i++)
                        usad[be[i]]=0;
                int flis=tlista(be,n+m);
                u0=we;
                pp0[1]=u0;
                int tlpp0=1;
                bool salir=true;
```

```
do
{
        int ind=1;
        bool encon=false;
        while(ind<=flis && encon==false)
        {
                if (p[be[ind]]==u0 && usad[be[ind]]==0)
                {
                        u0=be[ind];
                        tlpp0++;
                        usad[be[ind]]=1;
                        pp0[tlpp0]=u0;
                        encon=true;
                }
                else ind++;
        }
        if (encon==false) salir=false;
}
while (salir);

bool encon;
u0=we;
pe0[1]=u0;
int tlpe0=1;
salir=true;
do
{
        int ind=1;
        encon=false;
        while(ind<=flis && encon==false)
        {
                if (p[be[ind]]==u0 && usad[be[ind]]==0)
                {
                        u0=be[ind];
                        tlpe0++;
                        usad[be[ind]]=1;
                        pe0[tlpe0]=u0;
                        encon=true;
                }
                else ind++;
        }
        if (encon==false) salir=false;
}
while (salir);

for(i=tlpe0;i>=1;i--)
{
        tlpp0++;
        pp0[tlpp0]=pe0[i];
}

i=1;encon=false;
while(i<=tlpp0 && encon==false)
{
```

```
                if(qp==pp0[i])
                {
                        if (i%2==0)
                        {
                                for(j=i;j<=tlpp0;j++)
                                bep[j-i+1]=pp0[j];
                        }
                        else
                        {
                                for(j=i;j>=1;j--)
                                bep[i+1-j]=pp0[j];
                        }
                        encon=true;
                }
                else
                {
                        i++;
                }
        }

        int tl1=tlista(bep,n+m);
        for (j=pos;j<=tlpe-pos;j++)
                pe[pos+tl1+j]=pe[pos+j];

        for (j=posqu+1;j<=posqu+tl1;j++)
                pe[j]=bep[j-posqu];
        tlpe=tlista(pe,n+m);
        max0=0;
        for (i=1;i<=tlpe;i++)
                if(pe[i]>n && max0<pe[i]) {max0=pe[i];pos=i;}
}

cout<<"Camino alternante "<<endl;
for (i=1;i<=n+m;i++)
                cout<<"p("<<i<<")="<<pe[i]<<endl;
system("pause");
u=pe[1];
v=pe[2];

while (v!=r)
{
        z=mate[v];
        mate[v]=u;
        mate[u]=v;
        u=z;

        exito=false;
        j=1;
        do
        {
                if (pe[j]==z) {v=pe[j+1];exito=true;}
                else j++;
        }
        while(exito==false && j<=tlpe);
```

```
        }
        mate[v]=u;
        mate[u]=v;
        nrex=nrex-2;
        aug=true;
}

int main()
{
        leer_grafo_inicial(graf);
        n=graf.nve;
        inmatch(graf,mate,nrex);
        r=0;
        while (nrex>=2 && r<n)
        {
                r++;
                if (mate[r]==0)
                {
                        for (i=1;i<=n;i++)
                                q[i]=0;
                        aug=false;
                        m=0;
                        for (i=1;i<=n;i++)
                        {
                                p[i]=0;
                                d[i]=-1;
                                a[i]=true;
                                for(j=1;j<=n;j++)
                                        ca[i][j]=graf.lad[i][j];
                        }
                        d[r]=0;
                        tlq=tlista(q,n+m);
                        if (tlq==n+m) cout<<"error 1";
                        else {q[tlq+1]=r;tlq++;}

                        while (aug==false && tlista(q,n+m)!=0)
                        {
                                x=q[1];
                                eplista(q,n+m);
                                if (a[x]==true)
                                {
                                        cont=false;
                                        int t=tlista(ca[x],n+m);
                                        int y;
                                        for (int j=1;j<=t;j++)
                                                u[ca[x][j]]=false;
                                        do
                                        {
                                                bool nenc=true;
                                                int contad=1;
                                                int fin=tlista(ca[x],n+m);
                                                while (nenc==true && contad<=fin)
                                                {
                                                        y=ca[x][contad];
```

```
                        if (u[y]==false)
                        {
                                u[y]=true;
                                nenc=false;
                        }
                        else
                                contad++;
                }
                if (a[y]==true)
                {
                        if (d[y]%2==0)
                        {
                                m++;

                                blossom(x,y,b[m],w[m]);
                                contract(b[m],w[m]);
                        }
                        else
                        {
                                if (d[y]==-1)
                                {
                                        if (mate[y]==0)
                                                augment(x,y);
                                        else
                                        {
                                                z=mate[y];
                                                p[y]=x;
                                                d[y]=d[x]+1;
                                                p[z]=y;
                                                d[z]=d[y]+1;
                                        q[tlista(q,n+m)+1]=z;
                                        }
                                }
                        }
                }
                int t0=tlista(ca[x],n+m);
                int i0=1;
                termina=false;
                while(i0<=t0 && termina==false)
                {
                        if(u[ca[x][i0]]==false) termina=true;
                        else i0++;
                }

        }
        while (termina==true && aug==false && cont==false);


                }
        }
}
}
cout<<"Compa\244eros"<<endl<<endl;
for (i=1;i<=n;i++)
```

```
                    cout<<"mate("<<i<<")="<<mate[i]<<endl;
        cout<<endl;
        system("pause");
        return 0;
}
```

8.2.2. Modo de uso de algoritmo para grafos generales

Ejemplo 8.2.2.1. *Haciendo uso del algoritmo de Edmonds diseñado en C++, encontrar un apareamiento máximo M en el grafo general que se da a continuación:*

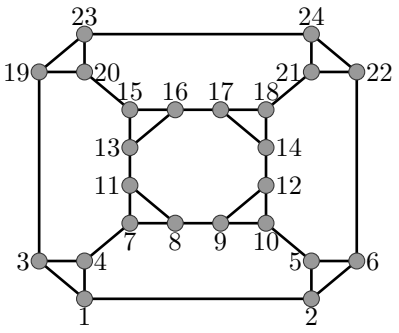

Figura 21: Grafo general.

Solución. El algoritmo de apareamiento máximo, llamado MAXMATCH, ya implementado en C++ toma como valores de entrada número enteros. El algoritmo inicia con una función *leer_grafo_ inicial*. El formato de entrada es el siguiente: *gra.nve=n*, donde n es el número de vértices del grafo. A continuación atendemos a la lista de adyacencia, la cual tiene la siguiente apariencia: $gra.lad[i][j] = k$. Observamos tres valores enteros: i,j y k. El primero y el tercero constituyen los vértices extremos de la arista $\{i, k\}$. El entero j es un valor que va desde 1 hasta m, donde m es el grado del vértice i. El entero i es un valor que va desde 1 hasta n, donde n es el número de vértices del grafo.

La solución del problema del grafo anterior es el apareamiento máximo siguiente:

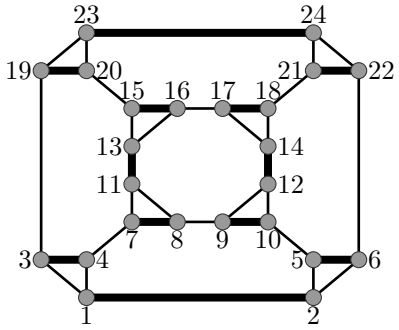

Figura 22: Apareamiento máximo en grafo general.

9. APLICACIONES

9.1. Aplicación grafo bipartito

Aplicación 9.1.1.

En una feria de negocios, temprano por la mañana, aparecen 10 personas V_i que venden al por mayor los siguientes productos: V_1=ropa, V_2=muebles, V_3=medicinas, V_4=alimentos, V_5=insumos agrícolas, V_6=computadoras, V_7=autos, V_8=cosméticos, V_9=zapatos y V_{10}=electrodomésticos. Más tarde, aparece el público, deseoso de comprar, pero se encuentran con la sorpresa de que cada vendedor está dispuesto a vender su productos en su totalidad a una sola persona. La multitud espera una flexibilidad por parte de los vendedores, pero estos alegan que prefieren a los compradores serios, que hayan venido a comprar, y que sepan que el tiempo es importante. Aparecen en ese momento 12 personas que están decididas a comprar y, además, disponen del efectivo para hacerlo. Es claro que no todos se muestran interesados por todos los productos que se ofrecen, pero la mayoría tiene preferencia por más de un producto. Las relaciones existentes entre los compradores y los vendedores que ofertan el producto de su preferencia son las siguientes:

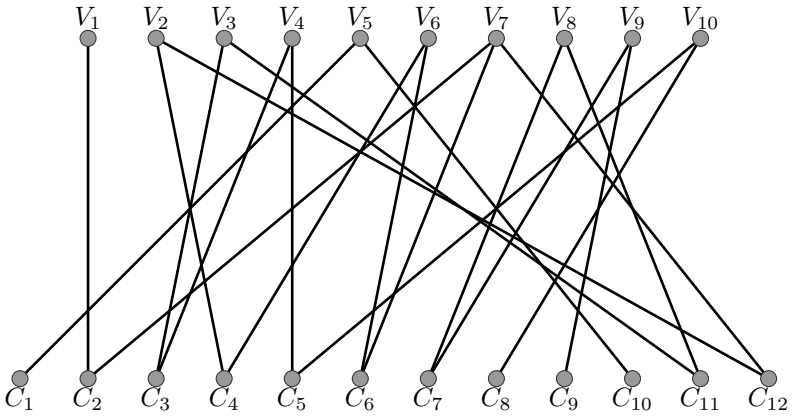

Figura 23: Grafo que modela la situación vendedores-compradores.

Ahora surge la pregunta natural: será posible que cada comprador se vaya satisfecho a casa con la compra que hizo, y que cada vendedor logre vender su producto?

Solución. Aquí entra en escena el uso del algoritmo de apareamiento máximo en grafo bipartito. La solución que el programa nos provee es la siguiente:

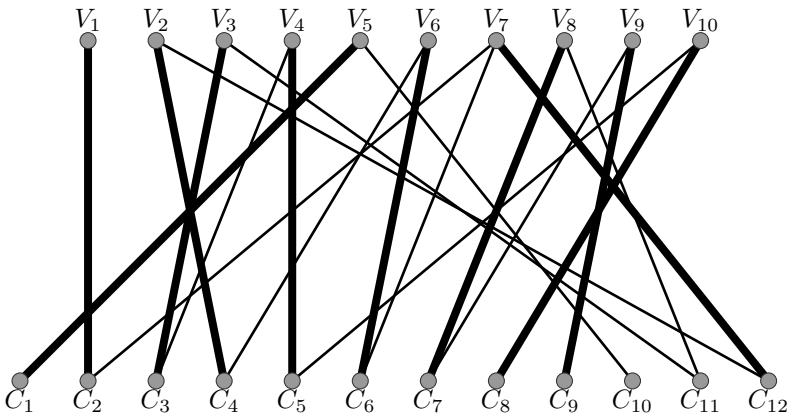

Figura 24: Solución de la situación vendedores-compradores.

El algoritmo encuentra el apareamiento máximo entre los dos conjuntos de vértices. Por tanto, todos los vendedores logran vender su producto, y 10 compradores se van satisfechos. 2 compradores no logran adquirir mercancías. De modo que con las relaciones dadas, se ha movido el máximo de dinero.

9.2. Aplicación grafo general

Aplicación 9.2.1.

En una feria de negocios las personas desean invertir con alguien que comparta alguna característica, condición o estatus económico. En una en particular aparece el siguiente caso. Se establecen 6 criterios, a los cuales llamaremos C_1= que sea Americano, C_2= que sea Europeo, C_3= que el capital que desea invertir sea mayor a \$1,000,000.00, C_4= que hable inglés, C_5= que llegue puntual a las reuniones y C_6= que tenga un título MBA.
Se desea con estos 12 inversores formar el mayor número de parejas para invertir en negocios.
Los inversores I_j, junto con los criterios C_k que cumplen puede representarse en el siguiente grafo:

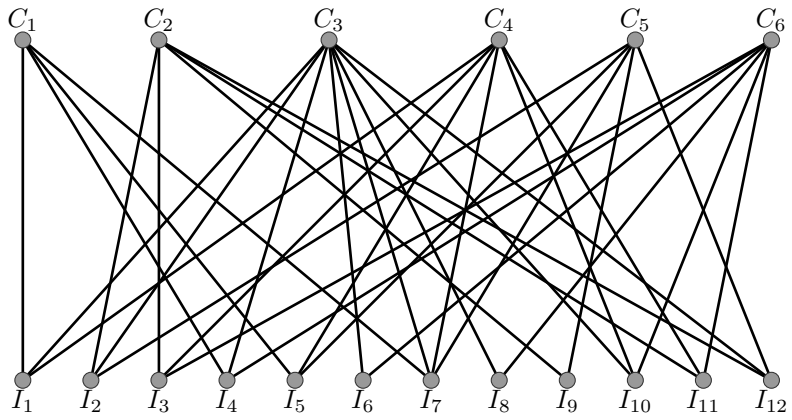

Figura 25: Grafo que modela los criterios que cumplen cada uno de los inversores.

Ahora bien, los negocios se dan entre personas, así que representamos en el grafo general siguiente las compatibilidades entre ellos. Los vértices representan a las personas, y las aristas indican que es posible que esas dos personas vayan juntas en el negocio.

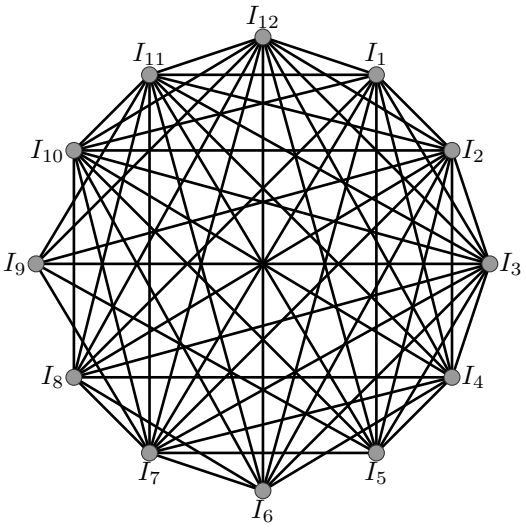

Figura 26: Inversores y sus potenciales parejas en base a los 6 criterios.

Al trabajar el problema anterior en el algoritmo Maxmath implementado, nos da el resultado siguiente:

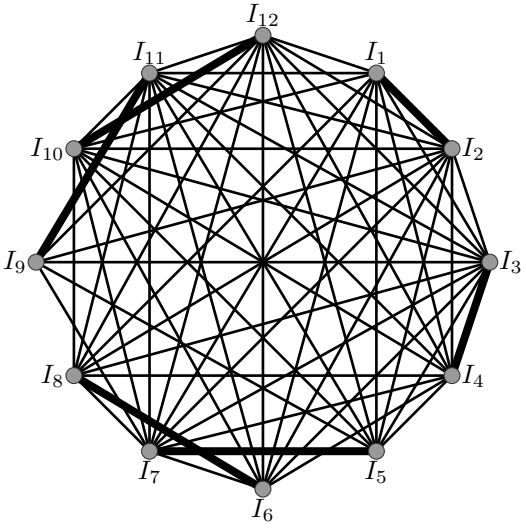

Figura 27: Inversores y sus parejas.

Así, el programa ha encontrado el apareamiento máximo en este grafo general.
El inversor 1 hará pareja con el inversor 2, el inversor 3 con el 4, el inversor 5 con el 7, el inversor 6 con el 8, el inversor 9 con el 11 y el inversor 10 con el 12.

10. CONCLUSIÓN

Después de haber tenido este breve contacto con la teoría de grafos, y en especial con los apareamientos, nos hemos dado cuenta que la teoría de grafos representa un pilar fundamental en la resolución de problemas aplicados. Reconocemos que en el futuro, con el avance de las ciencias y el gradual decaimiento del tiempo de procesamiento, irán surgiendo más áreas en las cuales las personas verán una oportunidad para aplicar esta herramienta.
Remarcando brevemente, es posible usar esta herramienta a nivel de medicina como el mostrado en [8], ya sea usándola directamente o realizando cierta modificación en el programa.
También podemos usarlo en asignación de personal atendiendo a diversas restricciones, como el mostrado en [2].

Los algoritmos usados en esta tesis no son nuevos, su historia se remonta hacia el año 1955 con el algoritmo de Ford y Fulkerson. Después, Edmonds y Karp contribuyeron con una modificación, y nos legaron el algoritmo mostrado en esta tesis. Vemos también importante el hecho de que es necesario el paso del tiempo para que surja una mejora para cada uno de los procedimientos existentes. Por tanto, los autores nos llenaremos de satisfacción cuando el momento llegue en que surja un proyecto ambicioso en miras de mejorar las cosas que hoy presentamos.

El programa que hoy presentamos es eficiente. Más aún si consideramos las capacidades de cálculo y almacenamiento que disponen los ordenadores del momento.
Animamos a las personas a que se atrevan a superar este trabajo, y contribuir de esta forma al avance de la ciencia, al desarrollo de la técnica y al descubrimiento de herramientas potentes para la resolución de problemas para beneficio de la sociedad Nicaragüense.

11. RECOMENDACIONES

Al finalizar esta tesis, consideramos como recomendaciones las siguientes:

- Al departamento de matemática de la Unan-León, para que considere la posibilidad de reforzar la clase de teoría de grafos, dedicando un buen espacio de tiempo al contenido de la teoria de apareamientos.
- A los estudiantes en general, y a las personas interesadas, para que centren su atención en el estudio de algoritmos de la teoría de grafos, ya que estos proveen herramientas poderosas que pueden usarse para modelar y resolver problemas tanto de la matemática pura, como de la vida real.
- A los estudiantes de la Unan-León, para que se interesen por el potencial que encierra la teoría de grafos, la cual consideramos tiene en la actualidad un futuro prometedor con el avance de la era de la información y la computación.
- A los profesionales de distintas áreas, para que conlleven una comunicación más fluida con los matemáticos, pues es la forma en la que pueden ocurrir ideas para inventar estrategias, procedimientos o algoritmos, los cuales pueden implementarse en un lenguaje entendible para una computadora, en lugar de hacerlo manualmente.

12. BIBLIOGRAFÍA

[1] *Theory of graphs*, American Mathematical Society, 1974.

[2] *Ten applications of graph theory*, D. Reidel Publishing Company, 1984.

[3] *Matching theory*, Elsevier Science Publishers B.V., 1986.

[4] *The theory of graphs*, Dover Publications, Inc, 2001.

[5] *Graphs, networks and algorithms*, Springer, 2013.

[6] Roth Alvin, Sonmez Tayfun, and Unver M. Utku, *Pairwise kidney exchange*, Journal of Economics Theory 125(2005) 151-188 (2005).

[7] Kim de Bakker, *Matching theory and the allocation of kidney transplantations*, 2016.

[8] Seguev Dorry, Sommer Gentry, Daniel Warren, Brigitte Reeb, and Montgomery Robert, *Kidney paired donation and optimizing the use of live donors organs*, American Medical Association (2005).

[9] Jack Edmonds, *pahts, trees and flowers*, Combinatorial Mathematics Project (1965).

[10] Jack Edmonds and Richard Karp, *Theoretical improvements in algorithmic efficiency for network flow problems*, Association for Computing Machinery Vol 19 (1972).

[11] Ford and Fulkerson, *Maximal flow through a network*, (1955).

[12] USA, *National organ transplant act*, 1984.