Teoría de Grafos

Redes de Flujo

B.S. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería

Red de Flujo

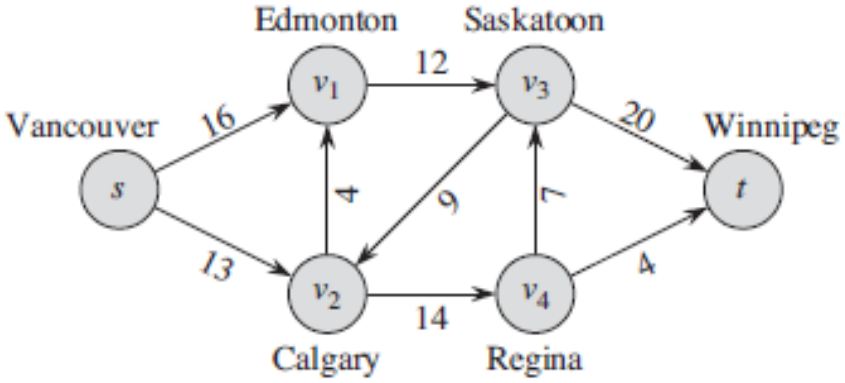
- Es un grafo dirigido $G = (V, E)$ en el cual cada arista $(u, v) \in E$ tiene asociada una capacidad $c(u, v) \geq 0$.

- Distinguiremos dos vértices especiales en el grafo:

  Fuente / source (s) : generalmente no tiene aristas de entrada.

  Sumidero / sink (t) : generalmente no tiene aristas de salida.

Red de Flujo


Edmonton
Saskatoon
Vancouver
Winnipeg
Calgary
Regina
v_1
v_2
v_3
v_4
s
t
16
13
12
14
20
4
9
7
4

Flujo

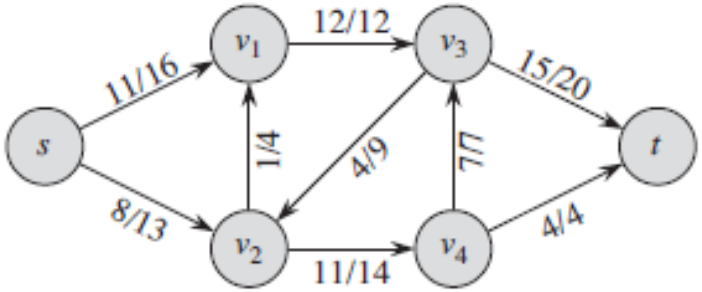
Sea $G = (V, E)$ una red de flujo, se denomina flujo a una función $\boldsymbol{f}$ tal que satisface 2 propiedades:

- Restricción de capacidad : $\forall\ (u, v) \in E,\ \ \mathbf{0} \leq \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{v}) \leq \boldsymbol{c}(\boldsymbol{u}, \boldsymbol{v})$

- Conservación del flujo : $\forall\ u \in V - \{s, t\}\,,$

$$\sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{v}, \boldsymbol{u}) = \sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{v})$$

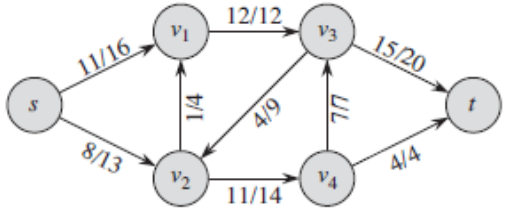
Flujo



Flujo de la Red

El flujo $|\boldsymbol{f}|$ de la red $G = (V, E)$ está definido como:

$$|\boldsymbol{f}| = \sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{v}) - \sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{v}, \boldsymbol{s})$$



Flujo de la Red

Generalmente el source no tiene aristas de entrada, entonces $|\boldsymbol{f}|$ es igual al flujo total que sale de $\boldsymbol{s}$, que es equivalente al flujo total que llega a $\boldsymbol{t}$.

$$|\boldsymbol{f}| = \sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{v}) = \sum_{\boldsymbol{v} \in \boldsymbol{V}} \boldsymbol{f}(\boldsymbol{v}, \boldsymbol{t})$$

Problema del Flujo Máximo

Dada una red de flujo $G = (V, E)$, el problema consiste en encontrar el flujo de máximo valor que admite G.

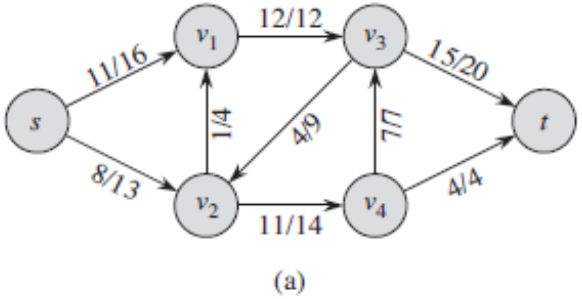
Método Ford-Fulkerson

- Inicializamos el flujo de cada arista y de la red **G** en 0.
- En cada iteración se busca un **camino de aumento** en la **red residual** G_f para incrementar el valor del flujo de la red **G**.
- Termina cuando ya no existe camino de aumento.

Red Residual

- Es una nueva red de flujo que representa cómo podemos variar el flujo sobre las aristas de la red original.
- Dado una red de flujo $G = (V, E)$, una red residual $G_f = (V, E_f)$ es una red con los mismos vértices, pero con las aristas con **capacidades residuales** $c_f(u, v) = c(u, v) - f(u, v)$
- Para tener la posibilidad de disminuir el flujo sobre alguna arista (u, v) debemos agregar la arista reversa (v, u) a G_f con $c_f(v, u) = f(u, v)$
- Las aristas reversas permiten que un algoritmo regrese flujo que ya había enviado a través de una arista.

Red Residual



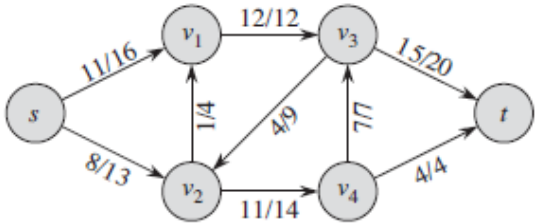
Camino de Aumento

- Dado una red de flujo G, un camino de aumento $\boldsymbol{p}$ es un camino simple desde $\boldsymbol{s}$ hacia $\boldsymbol{t}$ en la red residual G_f.

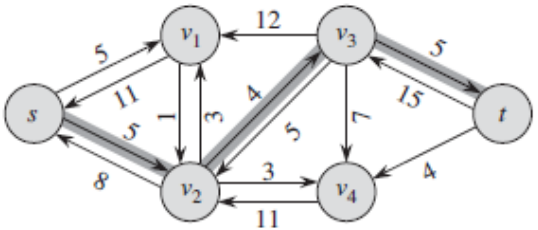
- **La capacidad residual del camino de aumento** $c_f(p)$ se define como la máxima cantidad de flujo que podemos aumentar a cada una de las arista de $\boldsymbol{p}$.

$$\boldsymbol{c_f(p) = \min\{c_f(u,v) : (u,v) \in p\}}$$

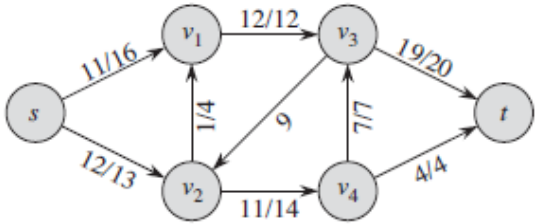
Camino de Aumento



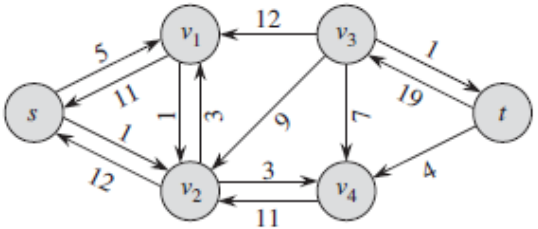
(a)



(b)



(c)

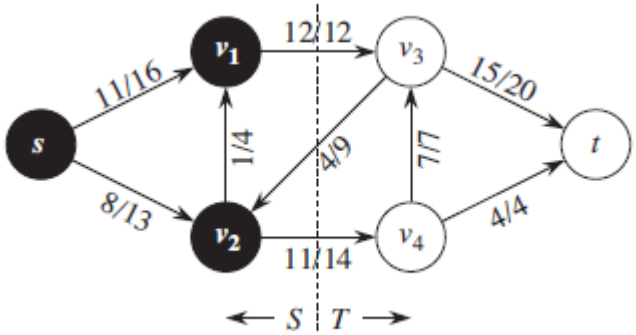


(d)

Cortes

- Dada una red de flujo $G = (\mathrm{V}, \mathrm{E})$, un corte $(\boldsymbol{S}, \boldsymbol{T})$ es una partición de V en $\boldsymbol{S}$ y $\boldsymbol{T} = V - S$, tal que $\boldsymbol{s} \in \boldsymbol{S}$ y $\boldsymbol{t} \in \boldsymbol{T}$.

- Si removemos aristas de **G** de tal forma que no exista camino de **s** a **t ,** también estamos formando cortes.

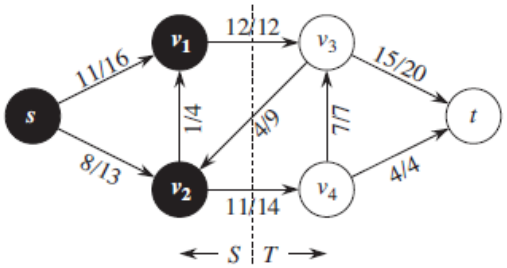


Cortes

- El conjunto de corte (cut-set) X_C de un corte $C = (S, T)$ está formado por todas la aristas que conectan el conjunto S y T.

$$X_C = \{(u, v) \in E : u \in S, v \in T\}$$

- Si removemos de la red de flujo a todas las aristas que aparecen en X_C , entonces no habría forma de pasar flujo de **s** a **t.**



Cortes

- **Flujo Neto de un corte (S, T)**

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u)$$

- **Capacidad de un corte (S, T)**

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

El corte mínimo de una red de flujo, es el corte con menor capacidad.

Cortes

Propiedad 1:

Dada una red de flujo $\boldsymbol{G} = (\mathrm{V}, \mathrm{E})$ con una función de flujo $\boldsymbol{f}$ y sea $(\boldsymbol{S}, \boldsymbol{T})$ cualquier corte de $\boldsymbol{G}$, entonces se cumple que $\boldsymbol{f}(\boldsymbol{S}, \boldsymbol{T}) = |\boldsymbol{f}|$

Cortes

Demostración

De la conservación de flujo, para cualquier nodo $u \in V - \{s, t\}$

$$\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) = 0$$

De la conservación de flujo, para cualquier nodo $u \in S - \{s\}$

$$\sum_{u \in S - \{s\}} \left(\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right) = 0$$

Cortes

Demostración

De la definición de flujo de una red $|f|$:

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in S-\{s\}} \left(\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) \right)$$

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in S-\{s\}} \sum_{v \in V} f(u,v) - \sum_{u \in S-\{s\}} \sum_{v \in V} f(v,u)$$

$$|f| = \sum_{v \in V} \sum_{u \in S} f(u,v) - \sum_{v \in V} \sum_{u \in S} f(v,u)$$

Cortes

Demostración

Debido a que $V = S \cup T \ \ y \ \ S \cap T = 0$:

$$|f| = \sum_{v \in S} \sum_{u \in S} f(u,v) + \sum_{v \in T} \sum_{u \in S} f(u,v) \ - \sum_{v \in S} \sum_{u \in S} f(v,u) - \sum_{v \in T} \sum_{u \in S} f(v,u)$$

$$|f| = \sum_{v \in S} \sum_{u \in S} f(u,v) - \sum_{v \in S} \sum_{u \in S} f(v,u) + \sum_{v \in T} \sum_{u \in S} f(u,v) \ - \sum_{v \in T} \sum_{u \in S} f(v,u)$$

$$|f| = \sum_{v \in T} \sum_{u \in S} f(u,v) \ - \sum_{v \in T} \sum_{u \in S} f(v,u) = f(S,T)$$

Cortes

Propiedad 2:

Dada una red de flujo $\boldsymbol{G} = (\mathrm{V}, \mathrm{E})$, el valor de cualquier flujo $|f|$ esta acotado superiormente por la capacidad de cualquier corte en G.

Cortes

Demostración

Sea (S, T) algún corte en G :

$|f| = f(S, T)$

$$|\boldsymbol{f}| = \sum_{\boldsymbol{u} \in \boldsymbol{S}} \sum_{\boldsymbol{v} \in \boldsymbol{T}} \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{v}) - \sum_{\boldsymbol{u} \in \boldsymbol{S}} \sum_{\boldsymbol{v} \in \boldsymbol{T}} \boldsymbol{f}(\boldsymbol{v}, \boldsymbol{u})$$

$$|\boldsymbol{f}| \leq \sum_{\boldsymbol{v} \in \boldsymbol{S}} \sum_{\boldsymbol{v} \in \boldsymbol{T}} \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{v}) \leq \sum_{\boldsymbol{v} \in \boldsymbol{S}} \sum_{\boldsymbol{v} \in \boldsymbol{T}} \boldsymbol{c}(\boldsymbol{u}, \boldsymbol{v})$$

$$|\boldsymbol{f}| \leq \boldsymbol{c}(\boldsymbol{S}, \boldsymbol{T})$$

Teorema max flow - min cut

Dada una red de flujo $\boldsymbol{G} = (\mathbf{V}, \mathbf{E})$, las siguientes condiciones son equivalentes:

1. f es un flujo máximo en G
2. La red residual G_f no contiene caminos de aumento
3. $|f| = c(S, T)$, para algún corte (S, T) de G.

Teorema max flow - min cut

Demostración

(1) → (2) :

Si el flujo es máximo no puede existir camino de aumento, ya que sino se podría aumentar el flujo.

(2) → (3) :

Si G_f no contiene camino de aumento, entonces no contiene un camino de $\boldsymbol{s}$ a $\boldsymbol{t}$.

Formemos dos particiones

$S = \{v \in V : existe\ camino\ de\ \boldsymbol{s}\ a\ \boldsymbol{v}\ en\ \boldsymbol{G_f}\}$ y $T = V - S$, entonces (S, T) es un corte.

Teorema max flow - min cut

Sea $\boldsymbol{u} \in \boldsymbol{S}$ y $\boldsymbol{v} \in \boldsymbol{T}$:

Si $(u, v) \in E$ → $f(u, v) = c(u, v)$, de lo contrario existiría la arista (u, v) en $\boldsymbol{G_f}$

Si $(v, u) \in E$ → $f(v, u) = 0$, de lo contrario existiría la arista (u, v) en $\boldsymbol{G_f}$

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) = \boldsymbol{c(S, T)} = |f|$$

Teorema max flow - min cut

Demostración

(3) → (1) :

Sabemos que $|f| \leq c(S,T)$, para cualquier corte (S,T), por ello si se cumple que $|f| = c(S,T)$, entonces f es el flujo máximo y $c(S,T)$ es el corte mínimo.

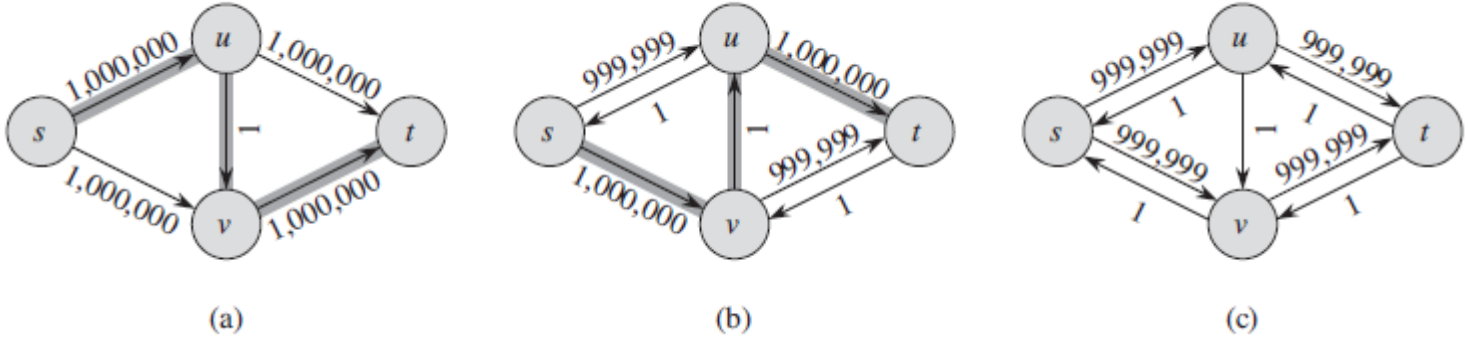
Algoritmo Ford-Fulkerson

```
int n, m, s, t;
vector<int> adj[ MAXV ];
int cap[ MAXV ][ MAXV ];
bool vis[ MAXV ], M[ MAXV ][ MAXV ];

int dfs(int u, int f){
    if( u == t ) return f;
    if( vis[ u ] ) return 0;
    vis[ u ] = 1;
    for(int i = 0; i < adj[ u ].size(); ++i){
        int v = adj[ u ][ i ];
        if( cap[ u ][ v ] == 0 ) continue;
        int ret = dfs(v, min(f, cap[ u ][ v ] ));
        if(ret > 0){
            cap[ u ][ v ] -= ret;
            cap[ v ][ u ] += ret;
            return ret;
        }
    }
    return 0;
}
```

```
int maxFlow(){
    int flow = 0;
    while( 1 ){
        memset(vis, 0, sizeof(bool)*(n));
        int inc = dfs( s, inf );
        if( inc == 0 ) break;
        flow += inc;
    }
    return flow;
}

void add_edge(int u,int v,int uv, int vu = 0){
    if( !M[ u ][ v ] ) adj[ u ].push_back( v );
    if( !M[ v ][ u ] ) adj[ v ].push_back( u );
    M[ u ][ v ] = M[ v ][ u ] = 1;
    cap[ u ][ v ] += uv;
    cap[ v ][ u ] += vu;
}
```

Complejidad: $O(E * maxFlow)$

Desventaja del algoritmo Ford-Fulkerson



Algoritmo Edmonds Karp

- Se basa en el método de Ford Fulkerson.
- Escoge como camino aumentador, al camino más corto que existe desde s a t (omitiendo los pesos de las aristas) en la red residual.

Algoritmo Edmonds Karp

Propiedad 1

Sea la red de flujo $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, entonces para todos los vértices $\mathbf{v} \in \mathbf{V} - \{\mathbf{s}.\mathbf{t}\}$, la distancia del camino más corto $\boldsymbol{d_f(s, v)}$ en la red residual G_f aumenta con cada aumento de flujo.

Algoritmo Edmonds Karp

Demostración

Supondremos que para algún vértice $\mathbf{v} \in \mathbf{V} - \{\mathbf{s}.\mathbf{t}\}$, hay un aumento de flujo que causa que $\boldsymbol{d_f(s,v)}$ disminuya, luego llegaremos a una contradicción.

Algoritmo Edmonds Karp

Demostración

1.- Sea $\boldsymbol{f}$ el flujo antes del primer aumento de flujo que disminuye la distancia de un camino más corto y $\boldsymbol{f}'$ el flujo justo después de ello.

2.- Sea $\boldsymbol{v}$ el vértice con el mínimo $d_{f'}(s, v)$ cuya distancia disminuyó luego del aumento de flujo, es decir : $d_{f'}(s, v) < d_{f}(s, v)$

Algoritmo Edmonds Karp

Demostración

3.- Sea $\boldsymbol{p} = \{\boldsymbol{s}, \dots, \boldsymbol{u}, \boldsymbol{v}\}$ el camino más corto desde $\boldsymbol{s}$ a $\boldsymbol{v}$ en $G_{f'}$,
entonces $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}_{f'}$ y $\boldsymbol{d}_{f'}(\boldsymbol{s}, \boldsymbol{u}) = \boldsymbol{d}_{f'}(\boldsymbol{s}, \boldsymbol{v}) - \boldsymbol{1}$

4.- Debido a como definimos $\boldsymbol{v}$, podemos garantizar que la distancia de $\boldsymbol{s}$ a $\boldsymbol{u}$ no ha disminuido, es decir $\boldsymbol{d}_{f'}(\boldsymbol{s}, \boldsymbol{u}) \geq \boldsymbol{d}_{f}(\boldsymbol{s}, \boldsymbol{u})$

5.- Podemos afirmar que $(\boldsymbol{u}, \boldsymbol{v}) \notin \boldsymbol{E}_{f}$ ya que de lo contrario tendríamos:

$$\boldsymbol{d}_{f}(\boldsymbol{s}, \boldsymbol{v}) \leq \boldsymbol{d}_{f}(\boldsymbol{s}, \boldsymbol{u}) + \boldsymbol{1} \leq \boldsymbol{d}_{f'}(\boldsymbol{s}, \boldsymbol{u}) + \boldsymbol{1}$$
$$\boldsymbol{d}_{f}(\boldsymbol{s}, \boldsymbol{v}) \leq \boldsymbol{d}_{f'}(\boldsymbol{s}, \boldsymbol{v}) \quad \textbf{...} \text{ Contradicción}$$

Algoritmo Edmonds Karp

Demostración

6.- Tenemos que $(u, v) \notin E_f$ y $(u, v) \in E_{f'}$, entonces el flujo que se incrementó debió pasar de v a u, para esto tuvo que existir un camino más corto $p = \{s, \dots, v, u\}$ en G_f. Por ello

$$d_f(s, v) = d_f(s, u) - 1$$
$$d_f(s, v) <= d_{f'}(s, u) - 1$$
$$d_f(s, v) <= d_{f'}(s, v) - 1 - 1$$

$d_f(s, v) <= d_{f'}(s, v) - 2$ **...** Contradicción con suposición original, el vértice v no existe.

Algoritmo Edmonds Karp

Propiedad 2

En el algoritmo de Edmonds Karp el número de aumentos de flujo que se realiza es $\boldsymbol{O(VE)}$

Algoritmo Edmonds Karp

Demostración

La arista (u, v) se denomina **crítica** sobre un camino aumentador de G_f si $c_f(p) = c_f(u, v)$

Luego de aumentar el flujo sobre el camino aumentador, las aristas críticas desaparecen de la red residual.

Al menos una arista del camino aumentador es crítica.

Algoritmo Edmonds Karp

Demostración

Demostraremos que cada una de las $|\boldsymbol{E}|$ aristas puede ser crítica a lo más $|\boldsymbol{V}|/\mathbf{2}$ veces.

Sea la arista $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{E}$, cuando sea crítica por primera vez, tendremos: $\boldsymbol{d_f(s, v) = d_f(s, u) + 1}$

Una vez que aumentamos el flujo, la arista $(\boldsymbol{u}, \boldsymbol{v})$ desparece de G_f hasta que el flujo de $(\boldsymbol{u}, \boldsymbol{v})$ decrece, lo cual ocurre solo si $(\boldsymbol{v}, \boldsymbol{u})$ aparece en un camino aumentador.

En ese momento se cumple que : $\boldsymbol{d_{f'}(s, u) = d_{f'}(s, v) + 1}$

Algoritmo Edmonds Karp

Demostración

De la propiedad 1: $d_f(s, v) \leq d_{f'}(s, v)$

$d_f(s, v) + 1 \leq d_{f'}(s, v) + 1$

$d_f(s, u) + 1 + 1 \leq d_{f'}(s, u)$

$d_f(s, u) + 2 \leq d_{f'}(s, u)$

Desde la primera vez que $(\boldsymbol{u}, \boldsymbol{v})$ fue crítica hasta la siguiente vez, la distancia de **u** se ha incrementado al menos en 2.

Por lo tanto $(\boldsymbol{u}, \boldsymbol{v})$ puede ser crítica a los más $|\boldsymbol{V}|/\boldsymbol{2}$ veces.

Algoritmo Edmonds Karp

```
vector<int> adj[ MAXV ];
int cap[ MAXV ][ MAXV ], parent[ MAXV ];
bool vis[ MAXV ], M[ MAXV ][ MAXV ];

int bfs(int s, int t) {
    deque< pair<int,int> > Q;
    Q.push_back( mk(s, inf) );
    while (!Q.empty()){
        int u = Q.front().first ;
        int cp = Q.front().second;
        Q.pop_front();
        for( int i = 0; i < adj[ u ].size(); ++i ){
          int v = adj[ u ][ i ];
          if( !vis[ v ] && cap[ u ][ v ] > 0 ){
            parent[ v ] = u;
            vis[ v ] = 1;
            int ncp = min( cp, cap[ u ][ v ] );
            if( v == t ) return ncp;
            Q.push_back( mk(v, ncp) );
          }
        }
    }
    return 0;
}
```

```
int maxFlow(int s, int t){
  int flow = 0;
  while( 1 ){
    memset(parent, -1, sizeof(int)*(n));
    memset(vis, 0, sizeof(bool)*(n));
    int inc = bfs( s, t );
    if( inc == 0 ) break;
    flow += inc;
    int cur = t;
    while( cur != s ){
      int u = parent[ cur ], v = cur;
      cap[ u ][ v ] -= inc;
      cap[ v ][ u ] += inc;
      cur = u;
    }
  }
  return flow;
}

void add_edge(int u,int v,int uv, int vu = 0){
  if( !M[ u ][ v ] ) adj[ u ].push_back( v );
  if( !M[ v ][ u ] ) adj[ v ].push_back( u );
  M[ u ][ v ] = M[ v ][ u ] = 1;
  cap[ u ][ v ] += uv;
  cap[ v ][ u ] += vu;
}
```

Complejidad: $\boldsymbol{O(VE^2)}$

Problemas

UVA 820 – Internet Bandwidth

UVA 10330 – Power Transmission

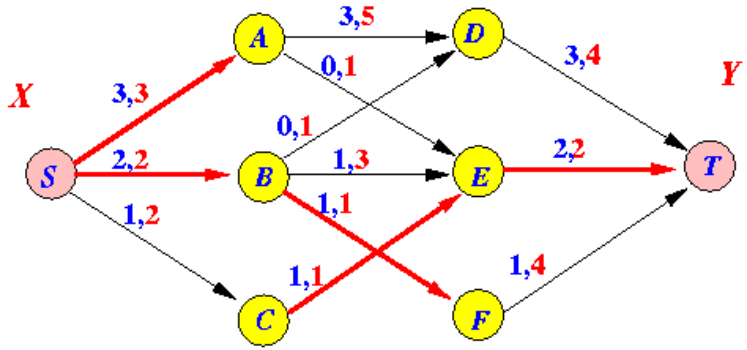
Corte mínimo

- En una red de flujo $\boldsymbol{G}$ el corte mínimo se refiere al corte con capacidad mínima.
- Si una red residual $\boldsymbol{G_f}$ no contiene camino de aumento, entonces no contiene un camino de $\boldsymbol{s}$ a $\boldsymbol{t}$ y podemos definir un corte $C = (S, T)$ de la siguiente manera:

  $S = \{v \in V : existe\ camino\ de\ \boldsymbol{s}\ a\ \boldsymbol{v}\ en\ \boldsymbol{G_f}\}$ y $T = V - S$

  Y de la demostración del teorema max flow – min cut, podemos garantizar que dicho corte C es mínimo.

Reconstruir corte mínimo

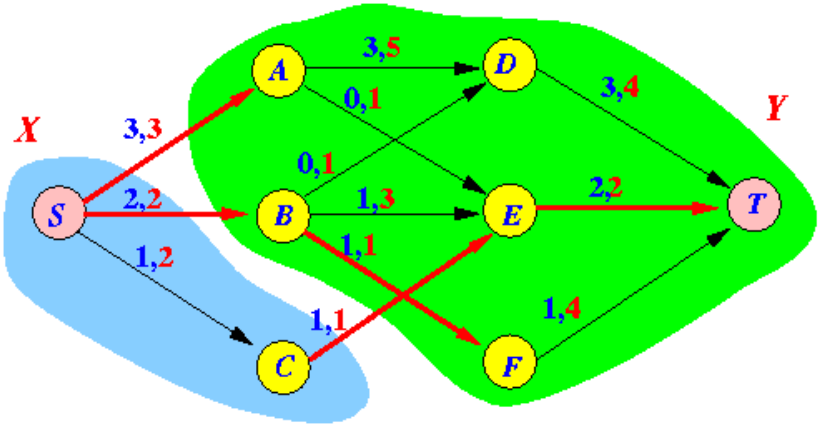


1. Usamos el método Ford Fulkerson para obtener la red residual final.

2. Los nodos alcanzables desde s, serán parte de la partición S y $T = V - S$.

3. Todas las aristas que van de S a T forman parte del conjunto de corte buscado.

Reconstruir corte mínimo

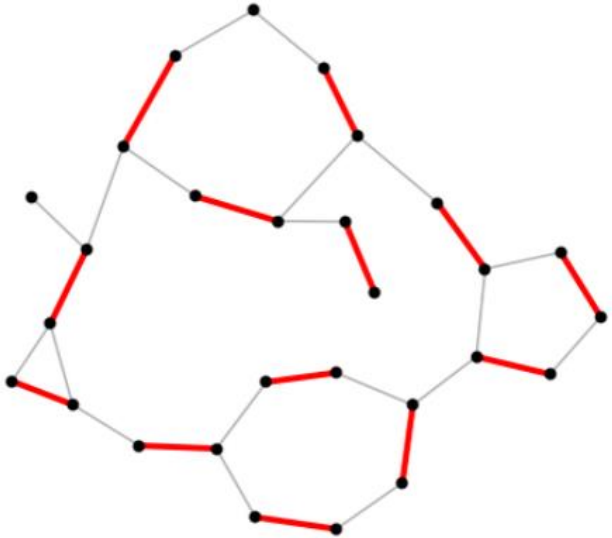


Problemas

SPOJ - COCONUTS

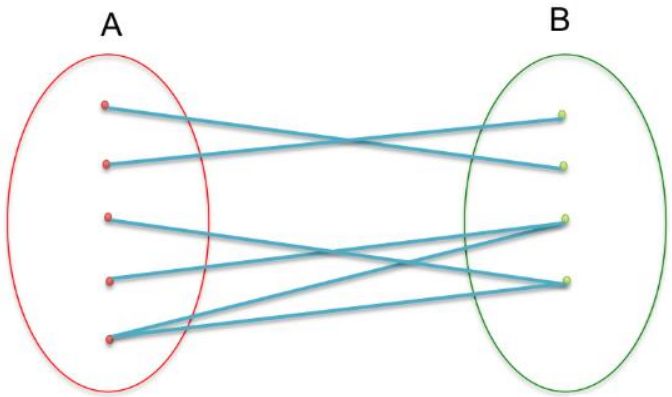
Matching

- Dado un grafo no dirigido $G = (V, E)$, un **matching** es un subconjunto de aristas $M \subseteq E$ tal que ningún par de aristas tenga vértices en común.
- Un **matching** se dice máximo, si no existe otro con mayor cantidad de aristas.



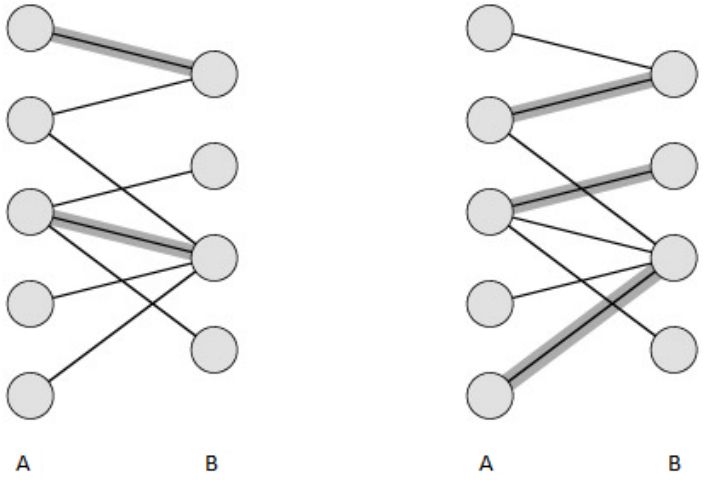
Grafo Bipartito

Un grafo bipartito $G = (V, E)$ es un grafo en el que los vértices se pueden particionar en 2 conjuntos disjuntos A y B $(V = A \cup B)$ y todas las aristas en E conectan un vértice de A y uno de B.



Maximum Bipartite Matching (MBM)

Nos enfocaremos en encontrar el **máximo matching** exclusivamente en grafos bipartitos.



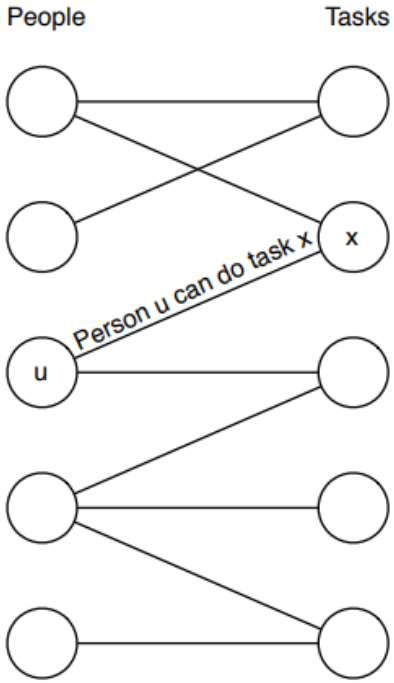
De MBM a Flujo máximo

Veámoslo con un ejemplo:

Dado un conjunto de personas y un conjunto de tareas, donde cada persona solo puede hacer algunas tareas.

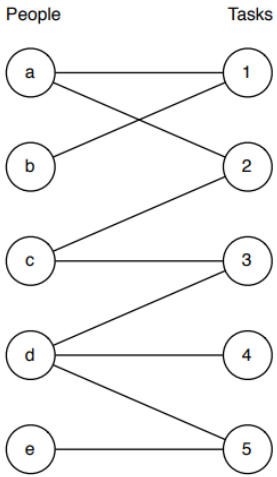
¿Cuál es el máximo número de tareas que se pueden realizar, sabiendo que las personas deben hacer tareas distintas?

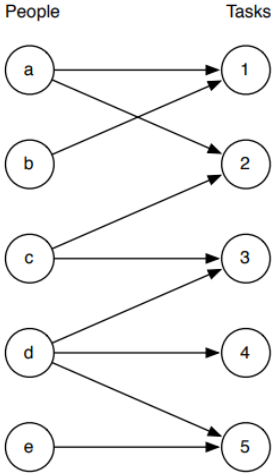
Podemos modelarlo como un grafo bipartito



De MBM a Flujo máximo

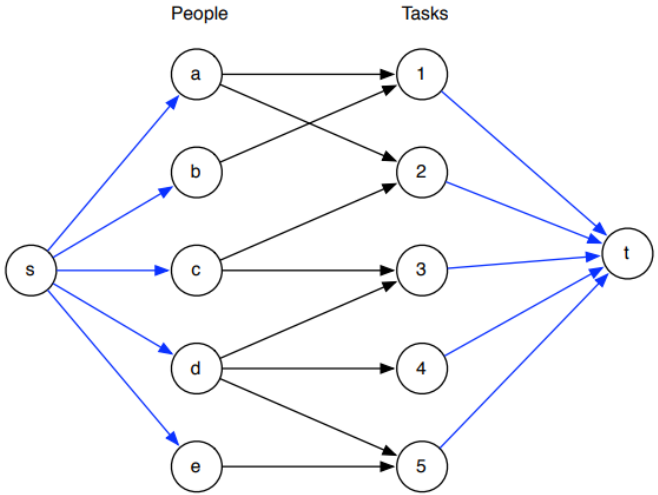
1.- A cada una de las aristas le ponemos dirección desde el conjunto A hacia B.





De MBM a Flujo máximo

2.- Agregamos lo vértices s y t. Además agregamos aristas desde s hacia cada uno de los vértices de A y aristas desde cada vértice de B hacia t.

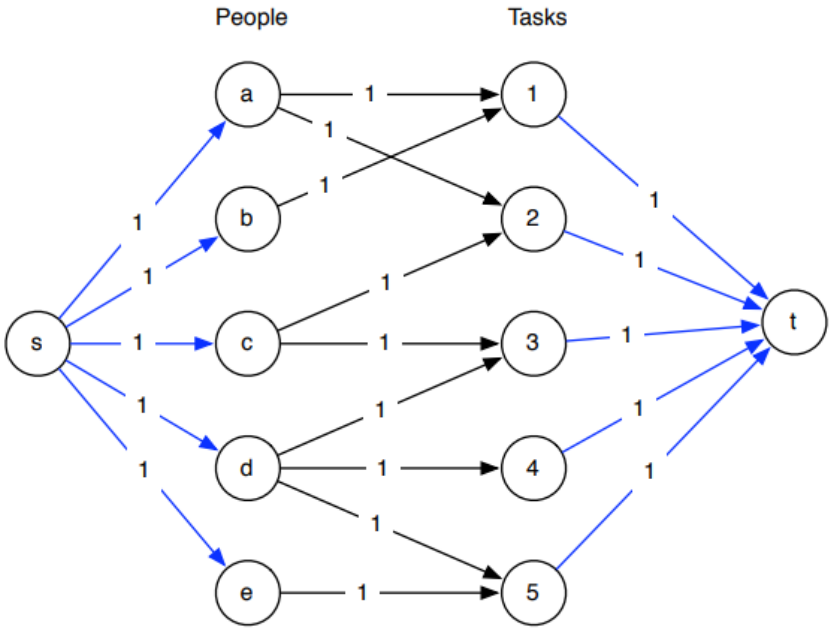


De MBM a Flujo máximo

3.- A cada una de las aristas del grafo formado le asignamos una capacidad igual a 1.

4.- El flujo máximo de la red de flujo formada será igual al máximo matching.

$$Ford\ Fulkerson: O(VE)$$



Algoritmo de MBM

- Podemos reducir la memoria que usa Ford-Fulkerson ya que el grafo es bipartito y todas las capacidades son 0 ó 1.
- Para cada vértice del primer conjunto usaremos un dfs que le encuentre un match en el segundo (camino de aumento).
- En el dfs probaremos con todos los posibles v adyacentes a u para que hagan match. Si el v aún no tiene match, asumimos que hará match con u. Pero si ya tiene match con u', entonces recursivamente revisaremos si a u' se le puede asignar otra pareja v'.

```
vector<int> adj[ MAXV1 ];
bool vis[ MAXV2 ];
int match[ MAXV2 ];
int n, m;

bool dfs( int u ){
  for( int i = 0; i < adj[u].size(); ++i ){
    int v = adj[ u ][ i ];
    if( !vis[ v ] ){
      vis[ v ] = 1;
      if( match[ v ] == -1 || dfs( match[ v ] ) ){
        match[ v ] = u;
        return 1;
      }
    }
  }
  return 0;
}
```

```
int mbm(){
  int flow = 0;
  for( int j = 0; j < m; ++j ) match[ j ] = -1;
  for( int i = 0; i < n; ++i ){
    for( int j = 0; j < m; ++j ) vis[j] = 0;
    flow += dfs( i );
  }
  return flow;
}
```

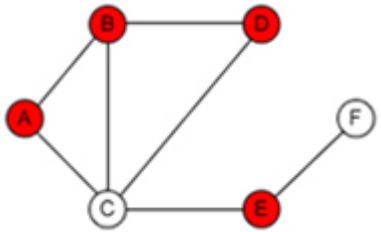
Problemas

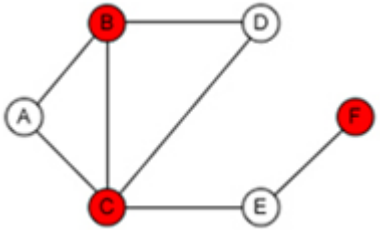
Live Archive 2044 - Courses

SPOJ-Taxi

Vertex Cover

- Dado un grafo $G = (V, E)$, decimos que $S \subseteq V$ es un vertex cover si cada arista del grafo es incidente en al menos un vértice de S.
- Podemos verlo como un conjunto de vértices con el cual puedo cubrir todas las aristas del grafo.
- Un vertex cover se dice mínimo si no existe otro con menor cantidad de vértices.
- El problema de hallar el vertex cover mínimo en un grafo cualquiera es **NP.**





Teorema de Konig

En un grafo bipartito la cantidad de aristas en un matching máximo es igual a la cantidad de vértices en un vertex cover mínimo.

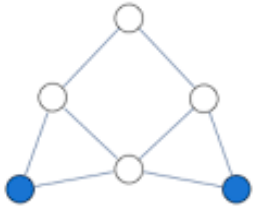
$$\textbf{matching máximo} = \textbf{vertex cover mínimo}\ , si\ G\ es\ bipartito$$

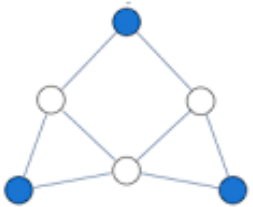
Problemas

SPOJ - Dungeon of Death
UVA 11159 – Factors and Multiples

Independent Set

- Dado un grafo $G = (V, E)$, decimos que $S \subseteq V$ es un independent set si no existe ninguna arista entre sus vértices.
- Un independent set se dice máximo si no existe otro con mayor cantidad de vértices.
- El problema de hallar el independent set máximo en un grafo cualquiera es **NP.**

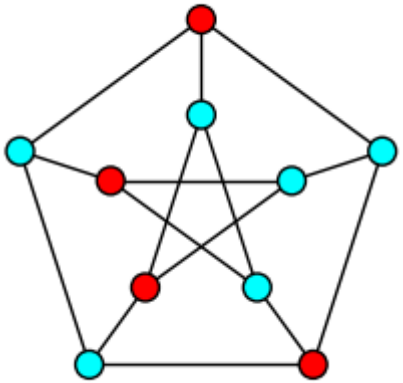




Vertex Cover e Indepent Set

- Para cualquier grafo G, el complemento de un vertex cover es un independent set.
- El complemento de un vertex cover mínimo es un independent set máximo.

$$V = |\boldsymbol{vertex\ cover\ mínimo}| + |\boldsymbol{independent\ set\ máximo}|$$



Problemas

Live Archive 2041 – Girls and boys

Referencias

- T. Cormen, Introduction to Algorithms
- E-maxx

¡ Good luck and have fun !