Graphs

$G(V, E)$ - graph, V - verticies, E - edges ($ab \neq ba$).

Directed Graph

$G(V, E)$ is a pair with V - a set and E - a subset of $V \times V$ ($\times$ - cartesian product: $A \times B = B \times A$, only if $A = B$).

Simple Graph

We say $G = (V, E)$ is simple if $E \subseteq V \times V \setminus \{(i, i), i \in V\}$

Weighted Graph

A weighted graph (directed) $G = (V, E, f)$ is the couple (V, E) and f with $f : E \to A$, with A an arbitrary set. f is called **weight function** (f maps E to some number).

Multigraph

We say $G = (V, E, f)$ is multigraph (directed) if G is a graph (directed) with $f : E \to \mathbb{N}$. f is called **multiplicity function**.

Undirected Graph

We say $G = (V, E)$ is an undirected graph if G can be represented as (V, E) with $E \subseteq \{\{u, v\} | u, v \in V\}$ ($ab = ba$)

Degree of a Vertex

The degree of $u \in V$, $deg_G(u), G = (V, E)$ - undirected graph, is:

$$deg_G(u) = |v \in V | \{u, v\} \in E|$$

Proposition

Let $G = (V, E)$ be a simple undirected graph. Then:

$$\sum_{v \in V} deg_G(v) = 2|E|$$

$$\sum_{v \in V} deg_G(v) = \sum_{v \in V} \{\{u, v\} | \{u, v\} \in E\} = 2|E|$$

For non-simple graph: $\sum_{v \in V} deg_G(v) = 2|E| - N_{loops}$

Handshaking Lemma

The number of guests who shook hands with odd number of people is even.

Isomorphism

Let $G = (V, E)$ and $G' = (V', E')$ be simple undirected graphs. A **bijection** $f : V \to V'$ is isomorphism between V and V' if $\forall u, v \in V$, $\{u, v\} \in E$: $\{f(u), f(v)\} \in E'$

Subgraph

Let $G = (V, E)$ be undirected simple graph. Let $G' = (V', E')$ be a graph. G' is a subgraph of G if $V' \subseteq V, E' \subseteq E$.

Induced Subgraph

Let $G = (V, E)$ be undirected simple graph. Let $G' = (V', E')$ be a graph, subgraph of G. We say G' is induced subgraph of G if $\forall v', u' \in V'$ s.t. $\{u', v'\} \in E$ and $\{u', v'\} \in E'$

Spanning

A graph $G' = (V', E')$ is a spanning subgraph of $G = (V, E)$ if G' is a subgraph of G with $V = V'$ that satisfies the condition $\forall v \in V, \exists e \in E'$ s.t. $v \in e$. Hence, a spanning subgraph of G is a subgraph of G whose edges 'span' the vertices of G ($G'(V', E')$ is spanning if $V' = V$ and E' is a subset of E)

Complete

Graph G is complete graph (denote K_n) if $V = \{v_1, ..., v_n\}$ and $E = \{\{v_i, v_j\}, i, j \in \{1, ..., n\}$ and $i \neq j\}$

Cycle

G s.u.g. We say the cycle (denoted as C_n) is the graph s.t. $V = \{v_1, ..., v_n\}, E = \{\{v_i, v_{i+1}\} | i = 1, ..., n-1\} \bigcup \{v_n, v_1\}$

Path

G s.u.g. We say G is the path (denoted as P_n) if $V = \{v_1, ..., v_n\}, E = \{\{v_i, v_{i+1}\}, i = 1, ..., n\}$ (cycle without $\{v_n, v_1\}$).

Proposition

Let $G = (V, E)$ be s.u.g. We say there is a path from u to v, $u, v \in V$ in G if $\exists$ a subgraph G' s.t. G' is isomorph to the path of length n and $u, v \in V(G')$. We say the same for cycles C_n

Walk

n-length of P_n. Walk from $v_1, ..., v_n$ is $\{v_1, \{v_1, v_2\}, \{v_2, v_3\}, ..., v_n\}$

Bipartite

$G = (V, E)$ is bipartite if $V = V_1 \bigcup V_2$ for some sets V_1 and V_2 and $E \subseteq V_1 \times V_2$ (V_1 and V_2 are disjoint) (a graph whose vertices can be divided into two independent sets, U and V such that every edge (u, v) either connects a vertex from U to V or a vertex from V to U).

Bipartite Complete

Let V and V' - two sets (disjoint) with $|V| = n, |V'| = m$. Then the complete bipartite graph $K_{m,n}$ is s.u.g. with $V(K_{m,n}) = V \bigcup V', E = \{\{u, v\} | u \in V, v \in V'\}$ (map from V to V'). V and V' are the ports of $K_{m,n}$.

Connected

A graph G is connected if for any $u \neq v \in V(G)$, there is a path from u to v.

Distance

$G = (V, E)$ s.u.g. Distance $d_G(v, u) : \{u, v\}, u, v \in V$ and $u \neq v \to$ length of the shortest path between u and v.

$$d_G : \{\{u, v\}, u \neq v, u, v \in V\} \to \mathbb{N}$$

$$\{u, v\} \to n \in \mathbb{N}$$

$n -$ length of the shortest path from u and v

Degree of a Sequence | Score

The degree sequence of $G = (V, E)$ s.u.g. is (a **score** of G): $(deg_G(v_1), ..., deg_G(v_n))$ with $V = \{v_1, ..., v_n\}$

Score Theorem

Let $n \geq 1$ and degree sequence $C = (C_1, ..., C_n), C_1 \leq ... \leq C_n$. Let degree sequence $C' = (C'_1, ..., C'_{n-1})$ s.t.:

$$C'_i = \begin{cases} C_i, & 1 \leq i \leq n - C_n - 1 \\ C_i - 1, & n - C_n \leq i \leq n - 1 \end{cases}$$

Then G' s.u.g. has degree sequence $C' \iff \exists G$ s.u.g. s.t. C is the degree sequence of G.

Eulerian Tour

Let $G = (V, E)$ s.u.g. An Eulerian Tour is a Walk $T = \{v_0, e_1, v_1, ..., e_n, v_0\}$ s.t. $E = \{e_i | i = n\}$ and $e_i \neq e_j \forall i = j, (v_1, ...v_n) = V$. Allows to go back to vertex **without repeating an edge**. Traverses **all edges and verticies**.

We say G is Eulerian if it admits an Eulerian Tour.
Adding loops does not change anything about the existence of Eulerian Tour. We can generalize it to u.g.
A Tour is a Walk between two verticies that are the same.

Proposition

Let $G = (V, E)$ s.u.g. G is Eulerian $\iff$ G is connected and $deg_G(v_i)$ is even $\forall v_i \in V$

Proof

1. Necessity: Let $T = (v_0, e_1, ..., e_t, v_o)$ be an eulerian tour in G. Each time a vertex v appears in the tour, it has an 'ingoing' vertex and 'outgoing' vertex in the tour. Hence, since every edge appears once and only once in the tour, each vertex has an even degree.
2. Sufficiency: Let $G = (V, E)$ be a connected simple graph whose verticies have even degree. Let $T = (v_0, e_1, ..., e_m, v_m)$ be a tour (walk) in G of longest possible length. We prove that (1) $v_m = v_0$ and (2) $E = \{e_i | i \in \{1, ..., m\}\}$
   1) If $v_m \neq v_0$ then v_0 is connected to an odd number of edges of T. Since the degree of v_0 is even, there is an $e \in E \setminus E(T)$.By appending e to the tour T, we obtain a

tour in G which is strictly longer than T, and get a contradiction.

2) First prove that $V(T) = V$. Since G is connected, there is a vertex $v \in V \setminus V(T)$ and a $k \in \{1, ..., m\}$ s.t. $e = \{v, v_k\} \in E$. It follows that $(v, e_{k+1}, v_{k+1}, ..., v_m, e_1, v_1, ..., e_k, v_k)$ is a tour in G which is strictly longer that T, we get a contradiction. Last, we prove that $E(T) = E$. Otherwise, let $e = \{v_l, v_k\} \in E \setminus E(T)$ with $l < k$. Then $\{v_l, e_{l+1}, ..., e_k, v_k, e_{k+1}, ..., v_m, e_1, ..., v_l, e, v_k\}$ is a tour in G which is strictly longer than T, we get a contradiction.

Hierholzer Algorithm | Find an Eulerian Tour

1. Pick any $v \in V(G)$.
2. Construct a Path with no repeated edge from v to v (no way to get stuck because of the even degrees of the verticies)
3. Pick $v' \in V$ s.t. $\{v', v''\}$ does not belong to the path, but v' belongs to it.
4. Build Tour from v' to v' using only edges that are not used already.
5. Iterate $\rightarrow$ get Eulerian Tour.

Inner Degree | Outer Degree

Let $G = (V, E)$ a s.d.g.(**simple** directed graph). We define $\forall v \in V, deg_-(v) = \{v' \in V | (v, v') \in E\}$ (all edges leave v, outer degree), $deg_+(v) = \{v' \in V | (v', v) \in E\}$ (all edges arrive in v, inner degree).

Eulerian Tour for Directed Graph

Let $G = (V, E)$ s.d.g. An Eulerian Tour in G is a sequence $(v_0, e_1, ..., e_n, v_0)$ where $V = \{v_0, ..., v_n\}$, $e_i = (v_{i-1}, v_i), \forall i \neq j, e_i \neq e_j$ and $E = \{e_1, ..., e_n\}$

Let $G = (V, E)$ a s.d.g. G is Eulerian $\iff$ G is connected and $deg_-(v) = deg_+(v) \forall v \in V$

Hamiltonian Tour

Let $G = (V, E)$ a s.u.g. A Hamiltonian Tour is a sequence $(v_0, e_1, v_1, ..., e_n, v_0)$ s.t. $V = \{v_0, ..., v_n\}$ and $v_i \neq v_j \forall i \neq j$ and $e_i \in E \forall i$ (same as Eulerian Tour but do not pass through the same vertex, not edge, or it is an Eulerian Tour for verticies).

Graph Difference

Let $G = (V, E)$ s.u.g. and $S \subseteq V$. Let define $G - S = (V', E')$ the graph s.t. $V' = V/S$ and $E' = \{\{v, u\} \in E | v, u \in V'\}$

Proposition

Let $G = (V, E)$ a s.u.g. and $S \subseteq V$ a non-empty set. If G is Hamiltonian, i.e. has a Hamiltonian Tour, then the number of connected components in $G - S$ is at most $|S|$ (cardinatility of S).

Proof Let C be an hamiltonian cycle in G. Let k be the number of connected components of $G - S$.
If $k = 1$, $|S| \geq 1$, which is true, since $S \neq \emptyset$.
Assume $k > 1$. Let $G_1, ..., G_k$ be connected components of $G - S$. For any $i \in \{1, ..., k\}$, we define u_i and v_i as the last vertex of C that belongs to G_i and its direct successor in C, respectively.
We prove that $\{v_1, ..., v_k\} \subseteq S$. Assume $v_1 \nsubseteq S$. Then $v_1 \in V(G - S)$ and $(u_1, v_1) \in E(G - S)$ (this edge is in E since it appears in cycle C). Since $u_1 \in G_1$, we have $v_1 \in G_1$, a contradiction.
$v_i \neq v_j, \forall i \neq j$ because C is a cycle and $(u_i, v_i) \in E(G), \forall i$. For each $i = 1, \ldots, k$ $\exists v_i \in S, v_i$ are distinct. Thus, $|S| \geq k =$ number of G_i.

Proposition

Let $G = (V, E)$ a s.u.g with $|V| \geq 3$. If $\forall v \in V, deg_G(v) \geq \frac{|V|}{2}, G$ is Hamiltonian.

Proof We denote by n positive integer s.t. $\frac{|V|}{2} \in (n, n+1]$.

1. Proof G is connected. Assume contrary G has atleast 2 connected components, G_1 and G_2. Since any vertex v in G_1 is connected to atleast $n+1$ verticies, but not to itself, we get $|G_1| \geq n+2$ and similarly $G_2 \geq n+2$, a contradiction.
2. Let $P = (x_0, \ldots, x_k)$ be the sequence of verticies of a path of maximal length in G. If this path is Hamiltonian, G is Hamiltonian. We prove $\exists i \in \{0, \ldots, k-1\}, \{x_0, x_{i+1}\} \in E$ and $\{x_k, x_i\} \in E (*)$. If $(*)$ is true, then $(x_0, \ldots, x_i, x_k, x_{k-1}, \ldots, x_{i+1}, x_0)$ is the vertex sequence of a cycle C in G. It is Hamiltonian, otherwise $\exists x \notin \{x_0, \ldots, x_k\}$ and $l \in \{1, \ldots, k-1\}$ s.t. $\{x, x_l\} \in E$. By combining this edge with C we get a longer path than P, a contradiction.

3. It remains to prove that $(*)$ is true. Assume $\nexists$ such i, i.e. $\forall i \in \{0, \ldots, k-1\} : \{x_0, x_{i+1}\} \in E \Rightarrow \{x_k, x_i\} \notin E$ and $\{x_k, x_i\} \in E \Rightarrow \{x_0, x_{i+1}\} \notin E(-)$. By maximality of P any vertex adjacent in G to x_0 belongs to $\{x_1, \ldots, x_k\}$ and similarly any vertex adjacent to x_k belongs to $\{x_0, \ldots, x_{k-1}\}$. Hence, by assumption, $\{x_1, \ldots, x_k\}$ contains atleast $n+1$ verticies adjacent to x_0 and $\{x_0, \ldots, x_{k-1}\}$ contains atleast $n+1$ verticies adjacent to x_k. Moreover, $k+1 \leq |V|$. It follows from $(-)$ that the sets $I = \{0 \leq i \leq k-1 | \{x_0, x_{i+1}\} \in E\}$ and $J = \{0 \leq i \leq k-1 | \{x_k, x_i\} \in E\}$ are disjoint. Since $|I|$ and $|J|$ are atleast $n+1$, we get $I \cup J$ contains atleast $2n+2 \geq |V| \geq k$ verticies, a contradiction.

Tree

Let $G = (V, E)$ a s.u.g. G is a tree if G is **connected** and if there are **no cycles** in G.

Proposition | End Vertex

Let $G = (V, E)$ a s.u.g. $|V| \geq 2$:

1. G is a tree $\iff$ G has atleast 2 end verticies.
2. G is a tree $\iff$ $G - v$ is a tree $\forall v \in V$ s.t. $deg_G(v) = 1$ (end vertex).

Proposition

If $G = (V, E)$ s.u.g. Then the following are equivalent:

1. G is a tree
2. $\forall v_1, v_2 \in V \exists$ unique path P between v_1, v_2
3. G is connected and $G - e := (V, E \setminus e)$ is not connected $\forall e \in E$
4. G contains no cycle and $G + e := (V, E \cup e)$ has a cycle $\forall e = \{v_1, v_2\} \notin E, v_1, v_2 \in V$
5. G is connected and $|V| = |E| + 1$

Labelled Tree

A labelled tree is a tree $T = (V, E)$ with $V = \{1, ..., n\}$.

Prüfer Sequence

Different way (more explicit) to represent a tree.

Algorithm:
Let T-labelled tree, let P-empty list.

Different way (more explicit) to represent a tree.

1. Identify the smallest i s.t. $deg_T(i) = 1$
2. Construct $G \setminus i$
3. Append the unique $j \in V$ s.t. $\{i, j\} \in E$ to the list P.
4. Iterate until $|V| = 2$

Length of the Prüfer Sequence is $|V| - 2$

Reconstruct Tree from Prüfer Sequence

Let $G = (V = \{1, ..., n\}, E =)$ (graph with no connection). Let $P = (P_1, ..., P_{n-2})$ with $P_i \in \{1, ..., n\} \forall i = 1, ..., n$. Let $D = (1, \ldots, n)$ sequence of length $n, n = len(P) + 2$.

1. For i in D:
    1. If $len(D) = 2$:
        1. Add edge to E consisting of the last two elements in D.
        2. Return E
    2. Else:
        1. Find smallest element in D which is not in P and first element in P and add an edge to E constisting of these two element.
        2. Delete these elements from D and P, respectively.

Proposition

Two labelled trees are isomorphic $\iff$ they have the same Prüfer sequence.

Cayley Formula

There is n^{n-2} labelled trees with $|V| = n$.

Spanning Tree

Let $G = (V, E)$ a s.u.g. A spanning tree T is a spanning subgraph of G which is a tree.

If $G = K_n$ (complete graph) there are n^{n-2} spanning (labelled) trees.

Algorithm for Constructing a Spanning Tree

Let $G = (V, E)$ a s.u.g. with $|V| = n, E = (e_1, ..., e_m)$. Let $E_0 =$

1. For $i \geq 1$:
    1. Consider $E_i = E_{i-1} \cup (e_i)$ if $(V, E_{i-1} \cup (e_i))$ has no cycle

    2. Else $E_i = E_{i-1}$
2. Stop when (let say at step k) $|E_i| = n - 1$ or when $i = m$

=> Get spanning tree is $T = (V, E_k)$

Proposition

Let $G = (V, E)$ s.u.g. s.t. $|V| = n$. Let $T = (V, E_T)$ the output of the spanning tree algorithm. Then, T is a (spanning) tree of G if $|E_T| = n - 1$ and G is not connected otherwise.

Minimal Spanning Tree

Let $G = (V, E)$ a s.u.g. weighted. T is a minimal spanning tree if T is a spanning tree and $\sum_{e \in E(T)} w(e)$ is minimal.

Algorithm for Constructing the Minimal Spanning Tree

Let $G = (V, E, w)$ a weighted s.u.g.

1. Construct the set $\{e_1, ..., e_n\}$ where $w(e_i) \leq w(e_j), \forall i \leq j$ and $E = \{e_1, ..., e_n\}$ (order elements by weights)
2. Apply the algorithm to construct a spanning tree.

Depth-First Search Algorithm

We have T a tree, a starting vertex $r \in V(T)$ and we have a condition C:

$$C(T) \to \begin{cases} TRUE \\ FALSE \end{cases}$$

We define function:

explore($T, u \in V(T)$)

1. if $(C(u) = TRUE)$

- return u

2. else label u as explored
3. $\forall v'$ s.t. $\{v', u'\} \in E(T)$ and v' is not labelled :

- explore (T, v')

Breadth-First Search Algorithm

- Input:
    1. a tree $T = (V, E)$
    2. a starting vertex r
    3. a condition C:

$$C \to \begin{cases} TRUE \\ FALSE \end{cases}$$

- Output: a vertex or $\emptyset$.

1. Initialization phase: list L and append r to L.
2. Check if $C(L[1]) holds$.
3. Append the neighbors of $L[1]$ to L.
4. Remove $L[1]$.
5. Iterate.

Shortest Path Problem

Let $G = (V, E, W)$ s.d.w.g. with weight function W. Assume G is connected. We want to find a path of **minimum** length / weight from $u \in V$ to $v \in V$.

More formally, let $W(u, u) = 0, \forall u \in V$. Let $W(u, v) = \infty$ if $(u, v) \notin E$. Length of the path $P(u, e_1, \ldots, e_n, v) = \sum_{i=1}^{n} W(e_i)$

Dijkstra Algorithm | Shortest Path from One Vertex to Another

1. Initialization phase:
    1. $X = \{u\}$
2. $\forall v \in V$
    1. Define $L(v) = W(u, v)$
    2. Set $P(v) = \{u, v\}$
3. While $X \neq V$:
    1. Pick $v_{min} \in V \backslash X$ s.t. $\forall v \in V \backslash X$: $L(v_{min}) \leq L(v)$
    2. $X = X \cup \{v_{min}\}$
    3. $\forall v \in V \backslash X$:
        1. if $L(v) > L(v_{min}) + W(v_{min}, v)$
            1. $L(v) = L(v_{min}) + W(v_{min}, v)$
            2. $P(v) = P(v_{min}) \cup v$

Proposition

At each step of the algorithm we have:

1. $\forall v \in V, P(v) =$ path from u to v of length $L(v)$
2. $\forall v \in X, L(v) =$ the minimal length in path from u to v.

3. $\forall v \in V \backslash X, L(v) =$ the minimal length in path from u to v with verticies, but v, belonging to X.

Proof 1. is true at initialization and at step 3 in algorithm. 2. and 3. are proved by mutual induction on $|X|$.

1. Base case: $|X| = 1$. We only have $u \in X, L(u) = 0$ minimal.
2. Induction step: assume 2. and 3. are true for some X and construct $X' = X \cup v$ where $v = v_{min}$ from 3.1. in the algorithm. We denote $L_X, L_{X'}$ map L at the end of step X and X', respectively.
   1. Let $w \in X'$. WTS $L_{X'}(u, w)$ is the length of the shortest path from u to w. If $w \in X$, then the result is obtained by induction hypothesis. If $w = v = v_{min}$, then $L_{X'}(u, w) = L_X(u, w)$ which by induction hypothesis means $L_{X'}(u, w)$ is shortest by 3.. Assume $L_{X'}(u, v)$ is not shortest $\Rightarrow \exists y \notin X \cup \{v\}$ and path $P(u, v)$ passing by y s.t. $W(P) < L_{X'}(u, v)$. Assume y is the first vertex of $P \notin X$ and we obtain that $L_X(u, y) < L_X(u, v)$, which contradicts $v = v_{min}$.
   2. Let $w \notin X'$. WTS $L_{X'}(u, w)$ is the length of the shortest path from u to w with verticies, but w, belonging to X'. By induction hypothesis $L_X(u, w)$ minimal from u to w with verticies, but w, $\in X$. In algorithm, step 3. states that $L'_X(u, w) = L_X(u, w)$ unless path from u to v followed by (v, w) is shorter than path from u to v, which is sufficient to obtain 3., since by induction hypothesis applied to 2., path from u to w is shortest with all verticies, but v, are in X.

Network

A network $N = (G, C, s, t)$ is a s.d.g. with $C : E(G) \to R^+$, a capacity function and a source vertex $s \in V$ and a target vertex $t \in V$. Here, capacity = weight. Problem: what is the maximum capacity that can flow from source vertex to a target vertex. Edge of zero capacity - nothing can pass through that edge.

Flow

Let N a network. A flow f in N is a map $f : E(G) \to R^+$ that satisfies:

1. $0 \leq f(e) \leq C(e), \forall e \in E(G)$
2. $\sum_{e|e^{(2)}=v} f(e) = \sum_{e|e^{(1)}=v} f(e), \forall v \in V \backslash \{s, t\}, e \in E(G)$

Let define $e^+ = e^{(2)}$ (goes in v) and $e^- = e^{(1)}$ (goes out of v).

Proposition

If f is a flow on N (network), then

$$\sum_{e^+(t)} f(e) - \sum_{e^-(t)} f(e) = \sum_{e^-(s)} - \sum_{e^+(s)}, e \in E(G)$$

You cannot transmit more than what have been received by s.

Value of Flow

Let $v(f) = \sum_{e^-=s} f(e) - \sum_{e^+=s} f(e)$ be the value of f.

Maximal Flow

f is maximal if $v(f) \geq v(f'), \forall f'$ flow.

Network Cut

N is a network. A cut of N is a partition (S, T) of $V(N)$ s.t. $s \in S, t \in T$ and the capacity of the cut (S, T) is $C(S, T) = \sum_{e^- \in S, e^+ \in T} C(e)$

Minimal Cut

We say that (S, T) is the minimal cut if $\forall (S', T')$ partitions of $V(N)$ we have $C(S, T) \leq C(S', T')$

Proposition

Let f a flow over N. For any cut (S, T) of N we have

$$v(f) \leq C(S, T)$$

Augmenting Path

Let N network, f a flow. An augmenting path W (without any cycle) is a path from s to t s.t.

1. $f(e) < C(e), \forall e \in E(W)$
2. $f(e) > 0, \forall e | (e^+, e^-) \in E(W)$

Augmenting Path Theorem

Let $N = (G, C, s, t)$ be a network. A flow f in N is maximal $\iff \nexists$ augmenting path with respect to f.

Proof

1. Sufficiency =>: let f-maximal flow on N. Assume W-augmenting path. Let $d = \min[\min_{e_i \in E(W)}[C(e_i) - f(e_i)], \min_{e_j | (e_j^+, e_j^-) \in E(W)} f(e_j)]$. Note $d > 0$ by construction of W. Let $f' : E(G) \to R^+$ s.t. if $e \in E(W), f'(e) = f(e) + d$. If $(e^+, e^-) \in E(W), f'(e) = f(e) - d$ and $f'(e) = f(e)$ else. We have that f' is a flow s.t. $f(f') = v(f) + d > v(f)$, a contradiction (basically, we can create a new flow by augmenting it by the minimal residual capacity, i.e. as long as there exists an augmenting path).
2. Necessity <=: assume $\nexists$ augmenting path. Let S-set of verticies $v \in V(G)$ s.t. $\exists$ augmenting path from s to v. Let $T = V(G) \backslash S$. (S, T) is a cut of N. Every $e \in E(G)$ s.t. $e^- \in S, e^+ \in T$ satisfies $f(e) = C(e)$ (else it could be to an augmenting path and e^+ would be in S). Also every $e \in E(G)$ s.t. $e^- \in T$ and $e^+ \in S$ should satisfy $f(e) = 0$ (same as earlier). Then:

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

$$= \sum_{e | e^- \in S, e^+ \in T} c(e) - \sum_{e | e^+ \in S, e^- \in T} 0 = C(S, T)$$

Then, $v(f)$ is maximal.

Max-flow Min-cut Theorem for Integer Capacities

Let N a network s.t. $\forall e \in E(N), C(e) \in N_0 (\in R_0^+$ for real capacities). Then, $\exists f$ a flow s.t. f is maximal and the value of f is equal to:

$$v(f) = \min_{\{(S', T') \text{ partitions of } V(N)\}} C(S', T')$$

If $- > O < -$, we increase $- >$, then decrease $< -$ by same amount.

Algorithm for Max Flow: Naive

1. $f(e) = 0 \forall e \in E(N)$.
2. While $\exists$ augmenting path W w.r.t. f:
    1. $W <-$ augmenting path
    2. $d = \min\{\min_{e \in E(W)}(C(e) - f(e)), \min_{(e^{(2)}, e^{(1)}) \in E(W)} f(e)\} \forall e \in E(W)$
    3. If $e \in E(W)$
        - $f(e) = f(e) + d$
    4. Elif $(e^{(2)}, e^{(1)}) \in E(W)$
        - $f(e) = f(e) - d$

Proposition

If $C(e) \in N$, then the algorithm is consistent.

Ford-Fulkerson Algorithm

Let $N = (G, C, s, t)$ a network.

1. Set flow everywhere to 0.
2. At each iteration:
    1. Find an augmenting path.
    2. Compute the bottleneck capacity (edge in the path with the smallest capacity, residual capacity).
    3. Augment each edge and the total flow.
    4. Note that for each node, flow into node = flow out of the node.
    5. If edge is backwards, then subtract the flow from its current flow (e.g. edge has 8/10, bottleneck is 4, this flow of the edge becomes 4/10 and bottleneck edge is 4/4).
    6. Iterate until edges leaving s are full or edges coming into t are full. If there is some flow left in edge leaving s, see if forward edges of the vertex of the edge leaving s are not full or backward edges are not empty (<0 I guess).

Algorithm

1. $\forall e \in E(N)$
    - $f(e) = 0$
2. Put a label on the sourse $s : (-, \infty)$
3. For $v \in V(N)$
    - $u(v) = FALSE$
    - $d(v) = \infty$
4. While $u(v) \neq TRUE$ for every labelled verticies $v \in V(N)$
    1. Pick labelled vertex $v \in V(N)$ (at first it is just s) s.t. $u(v) = FALSE$
    2. For $e \in \{e \in E(N) | e^- = v\}$

1. If $w = e^+$ is not labelled and $f(e) < C(e)$
   - $d(w) = \min(C(e) - f(e), d(v))$
   - Label w with $(v, +, d(w))$
3. For $e \in \{e \in E(N) | e^+ = v\}$
   1. If $w = e^-$ is not labelled and $f(e) > 0$
      - $d(w) = \min(f(e), d(v))$
      - Label w with $(v, -, d(w))$
4. $u(v) = TRUE$
5. If t is labelled
   1. Let d be the last component of the label of t
   2. $w = t$
   3. While $w \neq s$
      1. Select the first component v of the label of w
      2. If the second component is $+$ then
         - $f(e) = f(e) + d$ for $e = vw$
      3. Else
         - $f(e) = f(e) - d$ for $e = wv$
      4. $w = v$
   4. Delete all labels except the one of s
   5. For $v \in V(v)$
      - $u(v) = FALSE$
      - $d(v) = \infty$
5. Let S be the set of labelled verticies and $T = V \backslash S$

Proposition

Let N network. If $C(e) \in Q_0^+, \forall e \in E$, then the output of the F-F algorithm (S, T) is a minimal cut and $\exists$ a maximal flow f s.t. $v(f) = C(S, T)$.

Optimization of F-F

To optimize, we do BFS: pick s, put all verticies connected to s in a list, pick first vertex in list, e.g. A. Put all verticies connected to A in list. Iterate until reach t. The complexity then does not depent on C.

Matching

Let $G = (V, E)$ a s.u.g. A matching M in G is a set $M \subset E$ s.t. $\forall e_1, e_2 \in M : e_1 \cap e_2 = \emptyset$. No two edges are adjacent. A matching is maximal if $|M| \geq |M'|$ for all M'.

Vertex Cover

Let $G = (V, E)$ a s.u.g. A vertex cover V' is a set $V' \subset V$ s.t. $\forall e \in E$ is s.t. $\exists v' \in V'$ with $\{v'\} \cap e \neq \emptyset$. With vertex cover, we cover all the edges. Vertex covex is a set of verticies that includes at least one endpoint of every edge of G.

König's Theorem

Size of the vertex cover serves as the upper bound for the size of a matching in graph.

Let $G = (V, E)$ a bipartite s.u.g.

$$M(G) = \max_{\{M' \text{ matching in } G\}} |M'| =$$

$$\min_{\{V' \text{ s.t. } V' \text{ vertex cover of } G\}} |V'| = C(G)$$

For every edge in the matching, atleast one vertex must be in the cover. All these must be distinct by definition of a matching (no vertex is in two edges).

Application

In a bipartite graph. Getting the cardinality $M(G)$ can be done by taking a set $V' = \emptyset$ and add u to V' if $\exists e = (u, v)$ with $u, v \notin V'$. Then iterate over all edges. We get $O(|E|)$ complexity.

Algorithm for Finding out if a Graph is Bipartite

Remember that G is bipartite $\iff$ G has no odd length cycle.

1. Pick a starting vertex $v \in V(G)$
2. Put $l(v) = 0$ (label)
3. Run BFS algorithm on v (at first $L = [v]$):
   - At each step you get L a list of unexplored verticies.
   - If $L[1]$ has a neighbor in E that shares its label and $L[1]$ is unexplored then:
     - return $FALSE$
   - For all u s.t. $(u, L[1]) \in E$ and u unexplored:
     1. $l(u) = 1$ if $l(L[1]) = 0$, else $l(u) = 0$
     2. label $L[1]$ as explored
     3. add u to L
4. remove $L[1]$ from list.