Graph Algorithms

Table of contents

Chapter 1: O, Θ, Ω

We consider the sets:
$\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}^*$. All the functions will be of the type :
$f : \mathbb{N} \to \mathbb{R}^*$.

1. O

Let $f : \mathbb{N} \to \mathbb{R}^*$.

$$O(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^* \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})[\forall n \geq n_0][t(n) \leq cf(n)]\}$$

In other words, $O(f(n))$ is the set of functions $t(n)$ bounded by a real number multiplied by $f(n)$ for all n enough big ($n \geq n_0$).
Hence $t(n) \in O(f(n))$, we write $t(n) = O(f(n))$.

$O(1)$

We must consider the function $\mathbf{1} : \mathbb{N} \to \mathbb{R}^*$ such that $\mathbf{1}(n) = 1$ for evrey $n \in \mathbb{N}$
$t(n) = O(1) \Rightarrow \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}$ such that for every $n \geq n_0$
$t(n) \leq c\mathbf{1}(n)$. It means that for $n \geq n_0$ $t(n) \leq \mathbf{1}(n)$, $t(n) \leq c$
$t(n) = O(1)$ means that t is bounded by a constant.

$O(n^3)$

$t(n) = 3n^3 + 2n^2$
$t(n) \leq 3n^3$, $n_0 = 1$.
$t(n) = 0(n^3)$.

Homework

Prove that $3^n \neq O(2^n)$
Prove that $O(f(n) + g(n)) = O(max(f(n), g(n))$.

2. Ω

Let $f : \mathbb{N} \to \mathbb{R}^*$.

$$\Omega(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^* \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})[\forall n \geq n_0][t(n) \geq cf(n)]\}$$

$t(n) \in \Omega(g(n))$ is written also $t(n) = \Omega(g(n))$

Proposition

$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$$

3. Θ

$$\Theta(f(n) = O(f(n) \cap \Omega(f(n))$$

Definition

Let $f : \mathbb{N} \to \mathbb{R}^*$.

$$\Theta(f(n)) = \{t : \mathbb{N} \to \mathbb{R}^* \mid (\exists c, d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})[\forall n \geq n_0][cf(n) \leq t(n) \leq df(n)]\}$$

Proposition

$$f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$$

Chapter 2- Graphs-Definitions
1- Directed Graphs

Definition

$G = (X, U)$, X the set of vertices, U the set of arcs.
$T : U \to X$, $a \mapsto \text{the terminal extremity}$
$I : U \to X$, $a \mapsto \text{the initial extremity}$

If $a \in U$ we write also $a = \vec{xy}$ where x and y are the extemities of a.

Definition

If $\vec{xy} \in U$ then we say that x is the predecessor of y and y is the successor of x.

2. Representation of a graph
Adjacency Matrix

Let $G = (X, U)$ be a directed graph. We assume that $|X| = n$, $|U| = m$ and $X = \{1, 2, \dots, n\}$

Definition (Adjacency Matrix)

It is a square matrix $n \times n$ $A[G]$, where (a_{ij}) are the coefficients:

$$A[G] = (a_{ij})_{1\leq i\leq n, 1\leq j\leq n}$$

a_{ij} is the number of arcs with initial vertex equal to i and final vertex equal to j.

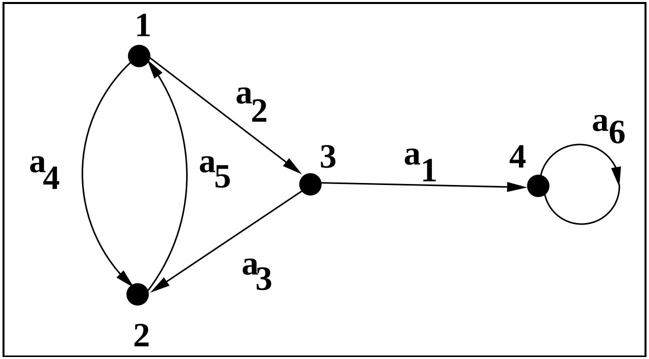


Figure: Graph 2

$$A[G] = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Incidence Matrix

It is a $n \times m$ matrix $B[G] = (b_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$
$U = \{a_1, a_2, \ldots, a_n\}$.

$$b_{ij} = \begin{cases} 1 & \text{if } i \text{ is the initial vertex of } a_j \\ -1 & \text{if } i \text{ is the final vertex of } a_j \\ 0 & \text{otherwise} \end{cases}$$

Incidence Matrix

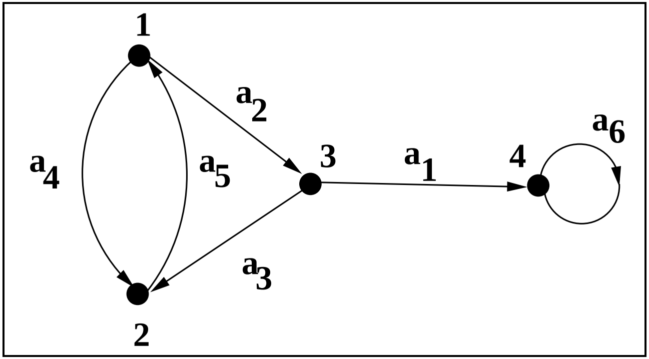


Figure: Graph 2

$$B[G] = \begin{bmatrix} 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Memory space

The needed memory space is in $O(n^2)$ for the matrix $A[G]$, and in $O(n \times m)$ for the matrix $B[G]$

2-Undirected Graphs

Definition

$G = (X, E)$
X is the vertex set. E is the edge set. $E \subset \mathcal{P}_2(X) \bigcup X$ ($\mathcal{P}_2(X)$ is the set of pair of elements of X). Parallel edges are allowed.

In the following example we have:
$X = \{1, 2, 3, 4, 5, 6\}$ et $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

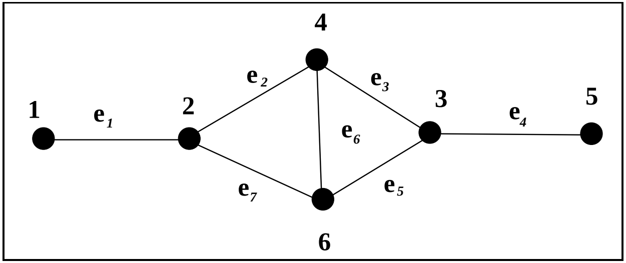


Figure: Graph 3

A graph G is said **simple** if it contains **no loop neither multiple edges (parallel edges)**.

Remarks

Like for the directed graphs, we define $A[G] = (a_{ij})_{1\leq i\leq n, 1\leq j\leq n}$, the adjacency matrix and $B[G] = (b_{ij})_{1\leq i\leq n, 1\leq j\leq m}$, the incidence matrix.

1. For the adjacency matrix, a_{ij} is the number of edges linking i et j.
2. For the incidence matrix, b_{ij} is the number of times that i is incident to a_j. In this representation the loops are present because the value of b_{ij} is equal to 2.

Adjacency, Neighborood

Definition

- Two vertices, x and y, of a directed graph $G = (X, E)$ (or of an undirected graph $G = (X, U)$) are **adjacents** if $xy \in E$ ($\overrightarrow{xy}$ or $\overrightarrow{yx} \in U$)
- Two edges are **adjacent** if they share an extremity.

Definition

1. A vertex x is **incident** to an edge e if $e = xy$ ($e = \overrightarrow{xy}$)
2. The neighboorood of a vertex x ist: $\Gamma(x) = \{y \in X / xy \in E\}$
3. The degree of a vertex x is $d(x) = |\Gamma(x)|$. It is the number of egdes incident to x, the loops are counted twice.

Definition

If $G = (X, U)$ is a directed graph. Let $x \in X$.

1. $$\Gamma^+(x) = \{y \in X / \overrightarrow{xy} \in U\}$$
2. $$\Gamma^-(x) = \{y \in X / \overrightarrow{yx} \in U\}$$
3. $d^+(x) = |\Gamma^+(x)|$ (outdegree of x)
4. $d^-(x) = |\Gamma^-(x)|$ (indegree of x)

Proposition

Let $G = (X, E)$ be a simple unoriented graph (without loop and multiple edges)

- $m \leq \frac{1}{2}n(n-1)$, *m is the number of edges and n is the number of vertices.*
- $\sum_{x \in X} d(x) = 2m$
- *The number of vertices having a odd degree is even. (Homework)*

Proposition

Let $G = (X, U)$ be a directed graph, then
$\sum d^+(x) = \sum d^-(x) = m$ *(m is the number of arcs)*

Data structures

Static structures

1. Adjacency Matrix
2. Incidence Matrix

Dynamic structures

List of successors: A list of lists, in which the first list is a list of indices corresponding to each node in the graph. Each of these refer to another list that stores the label of each adjacent node to this one.

Data structures

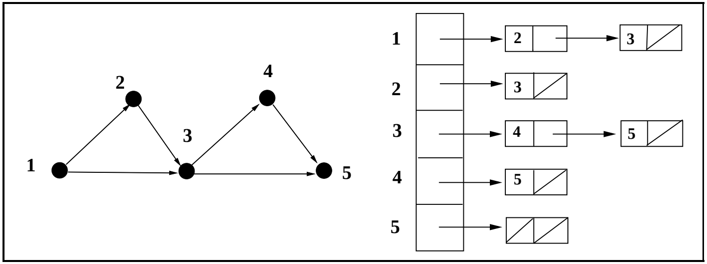


Figure: List of successors

Chapter 3- Graph Traversals

Definition

Let $G = (V, E)$ be a graph.

1. A walk is a sequence of vertices $\mu = (x_1, x_2, \cdots, x_k)$ such that: $x_i x_{i+1} \in E$, for $i \in \{1, \cdots, k-1\}$
2. A walk is closed if $x_1 = x_k$.
3. A path is a walk without repetition of edges.
4. A cycle is a closed path.
5. An elementary path (cycle) is a path (a cycle) without repetition of vertices (for the cycle except for one vertex).

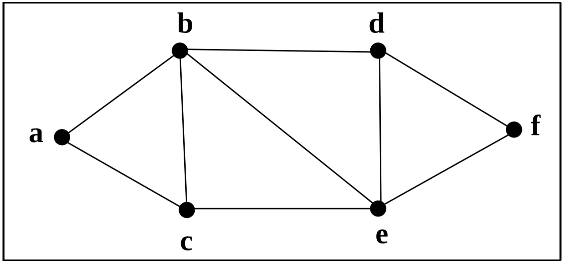


Figure: Walk, Path, Cycle

Closed walk: (a, b, c, e, b, d, b, a)
A path: (a, b, d, e, b, c)
A cycle: (b, d, e, c, b)
An elementary path : (a, c, e, f)

Properties and remarks

1. From any walk we can extract an elementary path.
2. For the directed graphs, we call path or dipath (resp. circuit) a path (resp. cycle) with all the edges having the same direction.
3. Let G be a gaph (digraph) and x, y belonging to V, the distance between x and y is the number of edges of a shortest path linking x and y.

Breadth-first search

```
BFS(G,s)
1  for each $u \in V \setminus s$
2      do color[$u$] $\leftarrow$ WHITE
3         d[$u$] $\leftarrow \infty$
4         $\Pi$[$u$] $\leftarrow$ NIL
5  color[s] $\leftarrow$ GRAY
6  d[s] $\leftarrow$ 0
7  $\Pi$[$s$] $\leftarrow$ NIL
8  F $\leftarrow \emptyset$
9  ENQUEUE(F,s)
10 while F $\neq \emptyset$
11     do $u$ $\leftarrow$ DEQUEUE(F)
12        for each $v \in$ Adj[$u$]
13            do if color[$v$]=WHITE
14                then color[$v$] $\leftarrow$ GRAY
15                     d[$v$] $\leftarrow$ d[$u$] +1
16                     $\Pi$[$v$] $\leftarrow$ $u$
17                     ENQUEUE(F,$v$)
18        color[$u$] $\leftarrow$ BLACK
```

Remark-Complexity

- • At the beginning all the vertices are white. If a vertex is dicovered then it becomes gray. A vertex with all the adjacent vertices are visited becomes black. We construct a spanning tree of shortest paths.
- • $|V| = n$ and $|E| = m$
  1. line 1 to 4: $O(n)$
  2. line 12 : $O(n)$
  3. line 11, loop while, it is done at most $\sum_{x \in V} d(x) = 2m$: $O(m)$

  Total complexity $O(m + n)$

Analysis (I)

We denote by $\delta(x, y)$, the distance between two vertices x and y of G

Lemma

Let G be a graph and s an arbitrary vertex of G. Then for any edge $e = xy$, we have:

$$\delta(s, y) \leq \delta(s, x) + 1$$

Lemma

Let G be a graph and assume that BFS is run on G from a vertex s. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies: $d[v] \geq \delta(s, v)$

Analysis (II)

Lemma

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue F contains the vertices $< v_1, v_2, \cdots, v_r >$, where v_1 is the head of F and v_r is the tail of F. Then:

1. *$d[v_r] \leq d[v_1]+1$.*
2. *$d[v_i] \leq d[v_{i+1}]$ for $i \in \{1, \cdots, r-1\}$*

Corollary

Assume that the vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $d[v_i] \leq d[v_j]]$ at the ime v_j is enqueued.

Analysis (III)

Theorem

Let $G = (V, E)$ a directed or undirected graph, and assume that BFS is run on G from a given source $s \in V$. Then during its execution, BFS discovers every vertex v that is reachable from the source s, and upon termination, d[v]$= \delta(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$, one of the shortest paths from s to v is a shortest path from s to Π[v] followed by the edge Π[v]v.

Depth-First search

The first timestamp $d(v)$ records when v is discovered (and grayed).
The second timestamp $f(v)$ records when search finishes after examining v's adjacency list (and blackens v)

- At the beginning all the vertices are WHITE.
- Any vertex u is WHITE before $d(u)$.
- Any vertex u is GRAY after $d(u)$ and before $f(u)$.
- Any vertex u is black after $f(u)$

Depth-First search

```
DFS(G)
1    for each vertex $u \in V$
2        do color[$u$] $\leftarrow$ WHITE
3           $\Pi[u] \leftarrow$ NIL
4    time $\leftarrow 0$
5    for each vertex $u \in V$
6        do if color[$u$]=WHITE
7           then DFS-Visit($u$)
DFS-Visit($u$)
1    color[$u$] $\leftarrow$ GRAY /* $u$ is discovered */
2    d[$u$] $\leftarrow$ time $\leftarrow$ time + 1
3    for each $v \in$ Adj[$u$]
4        do if color[$v$]=WHITE
5           then $\Pi[v] \leftarrow u$
6                DFS-Visit($v$)
7    Color[$u$] $\leftarrow$ BLACK
8    f($u$) $\leftarrow$ time $\leftarrow$ time + 1
```

Complexity:

$$O(n+m)$$

Theorem

In an DFS of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exacly one of the following conditions holds:

- $[d(u), f(u)] \bigcap [d(v), f(v)] = \emptyset$
- $[d(u), f(u)] \subset [d(v), f(v)]$ *and u is a descendant of v in DFS.*
- $[d(v), f(v)] \subset [d(u), f(u)]$ *and v is a descendant of u in DFS.*

Corollary

A vertex v is a proper descendant of vertex u in the DFS-forest for a (directed or undirected) graph $G \iff$
$d(u) < d(v) < f(v) < f(u)$

Topological sort

Definition

Let $G = (V, E)$ be a directed graph. A topological sort is an injective mapping $f : V \longrightarrow \mathbb{N}$ (it is a linear ordering of the vertices) such that for any x and any y, if $\overrightarrow{xy} \in U$ then $f(x) < f(y)$.

Theorem

A directed graph $G = (V, E)$ has a topological sort $\Longleftrightarrow$ it does not contain any circuit.

A directed graph without circuit is called a DAG (directed acyclic graph).

Topological sort algorithm

Topological-Sort(G)

1 call DFS(G) to compute finishing times $f(v)$ for each vertex v

2 as each vertex is finished, insert it onto the front of a linked list

3 return the linked list.

Complexity: $\theta(n+m)$

Analysis

Lemma

A directed graph is acyclic $\iff$ if a DFS of G yields no back edges.

Theorem

Topological-Sort(G) *produces a topological sort of a directed acyclic graph G.*

Strongly connected components

Let $G = (V, E)$ be a directed graph.

Definition

$x \overrightarrow{C} y$,We define $\overrightarrow{C}$ a binary relation in the set of vertices. For any couple of vertices (x, y) we will write:

$$x \overrightarrow{C} y \Leftrightarrow \begin{cases} \text{x=yor} \ : \\ \text{it exists a dipath from } x \text{ to } y \textbf{ and} \\ \text{it exists a dipath from } y \text{ to } x \end{cases}$$

$\overrightarrow{C}$ is a equivalence relation.

An equivalent class $\dot{x}$ under $\overrightarrow{C}$, $\dot{x} = \{y / x \overrightarrow{C} y\}$ is called a strongly connected component of G.
A graph $G = (V, E)$ is said strongly connected if it contains only one strongly connected component.

Strongly connected component algorithm

Strongly-connected-component(G)

1 call DFS(G) to compute finishing times $f(u)$ for each vertex u

2 compute G^{-1}

3 call DFS(G^{-1}), consider the vertices in order of decreasing $f(u)$ (as computed in line 1)

4 output the vertices of each tree in the DFS forest formed in line 3 as a separated strongly connected component.

Complexity: $\theta(n+m)$

Analysis

Lemma

Two vertices x and y belong to the same strongly connected component of $G \iff$ they belong to the same tree in the DFS forest of G^{-1}.

Theorem

Strongly-connected-component(G) *correctly computes the strongly components of a directed grph G.*

Chapter 4- Minimum spanning trees

We deal with undirected graph.

Definition

- let $G = (V, E)$ be a graph, a **partial subgraph** of G is a graph $G' = (V, F)$ with $F \subset E$
- a tree is an acyclic conneted graph
- a spanning tree of $G = (V, E)$ is a partial subgraph of G that is a tree.

Theorem

Theorem

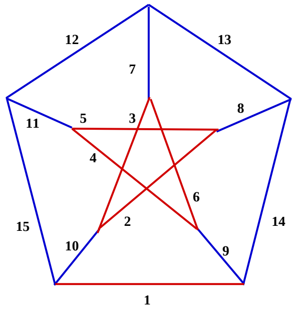
The following conditions are equivalent:

1. *T is a tree*
2. *T is a connected graph and $m = n - 1$ (m number of edges, n number of vertices)*
3. *T is an acyclic graph and $m = n - 1$*
4. *T is a connected graph and every edge of T is a* **bridge**
5. *T is an acyclic graph and if we link any two non adjacent vertices of T then we create a cycle (a unique cycce)*
6. *For any two vertices a et b, it exists a unique path linking a and b.*

Fundamental cycle

Definition

Let $G = (V, E)$ be graph and $T = (V, F)$ a spanning, the fundamental cycle associated to $e \in E \setminus F$ is the unique cycle $C_T(e)$ of $T \cup e$



Minimum spanning tree - Problem

Definition

Let $G = (V, E)$ be a graph $p : E \rightarrow \mathbb{R}$ be a weight funcion. Let $T = (V, F)$ be a spanning tree of G then:
$p(T) = \sum_{e \in F} p(e)$

Problem

Let G be a graph with a weighted function, find a minimum (weighted spanning) tree for G.

Kruskal Algorithm (1956)

```
MST-Kruskal(G,p)
/* the edges are sorted into the nondecreasing order
by weight */
/* $p(e1) \leq p(e_2) \leq \cdots \leq p(e_m)$ */

1   $S \leftarrow \emptyset$
2   $k \leftarrow 0$
3   while $k < \mathrm{m}$ and $|S| < n - 1$
4       do if $(V, S \cup e_{k+1})$ is acyclic.
5                   then $S \leftarrow S \cup e_{k+1}$
6         $k \leftarrow k + 1$
```

Analysis-Complexity

We prove by contradition that:

Theorem

Any spaning tree $T^* = \{e_1, , e_2, \cdots, e_{n-1}\}$ *obtained by Kruskal algorithm is optimal.*

For the acyclic test, we give a numbering to the vertices from 1 to n and we take an edge iff the number of the extremities of the edge are distinct. Then all the vertices having the same number as one of the extremity is renumbered with the samllest nuber of the extremities.
Complexity:

1. to sort the edges: $m \log m$
2. to compare the extremities of the edges $\theta(m)$
3. to renumber the vertices $O(n^2)$

The complexity is: $O(n^2 + m \log m)$

Prim Algorithm (1956)

MST-Prim (G,p,r)

```
1    $S \leftarrow \emptyset$
2    $A \leftarrow r$ /*A is the set of marked vertices*/
3    while $|S| < n-1$
4        do $u \leftarrow xy$ s.t. $x \in A$, $y \notin A$ and $p(xy)$ is minimum.
5            $S \leftarrow S \cup u$
6            $A \leftarrow S \cup y$
```

Analysis

We prove by induction on the number of steps that:

Theorem

If S is the set of selected edges then it exists a minimum spanning tree containg S

Prim Algorithm with priority queue

```
MST-Prim-Q(G,p,r)
1     for each $u \in V$
2         do key$[u] \leftarrow \infty$
3             $\Pi[u] \leftarrow$ NIL
4     Key$[r] \leftarrow 0$
5     Q $\leftarrow V$
6     while Q $\neq \emptyset$
7     do $u \leftarrow$ Extract-Min(Q)
8         for each $v \in$ Adj$[u]$
9             do if $v \in$ Q and $p(uv) <$ Key$[v]$
10                    then  $\Pi[v] \leftarrow u$
11                          Key$[v] \leftarrow p(uv)$
```

Complexity

- Build the heap $O(n)$
- Extract minimum $\log n$, we do it n times
- line 8 to 11 $O(m) \times O(\log n)$
- Total: $O(n \log n + m \log n)$, it is $O(m \log n)$

Chapter 5- Shortest Paths
1- Definitions

Let $G = (V, E)$ be a directed graph.
Let $w : E \to \mathbb{R}$ be a weight function. For $u \in E$ $w(u)$ is the weight of u

Definition

- A path is a alternated sequence of vertices and edges $x_1, u_1, x_2, u_2, \ldots, u_k, x_{k+1}$ such that $u_i = \overrightarrow{x_k x_{k+1}}$.
- If P is a path, the weight of P is $w(P) = \sum_{u \in P} w(u)$
- An **elementary path** does not go through the same vertex twice.
- A root of G is a vertex s of G such that for each $x \in V \setminus \{s\}$ it exists a path from s to x.
- A **circuit** is a closed path.

1- Definitions

Definition

- A circuit C is said **an absorbing circuit** if :
$w(C) = \sum_{u \in C} p(u) < 0$
- An **arborescence with root** r in a directed graph $G = (V, E)$ is a sub-graph such that the underlying undirected graph is a tree and it exists a path from s to any vertex of the arborescence.
- A **shortest path** linking two vertices x and y is a path linking x to y with a smallest weight.

Theorem

Let $G = (V, E)$ be a directed weighted graph. Let $s \in V$, then for any $x \in V \setminus \{s\}$ it exists a path linking s to x iff:

1. *s is a root of G*
2. *G does not contain an absorbing circuit.*

Single-source shortest path
1: The weights are nonnegative: Dijkstra Algorithm

Remark. There is no absorbing circuit
Idea. We start from the vertes s. we build a set of edges by taking successively edges with initial vertex in D (the set of discovered vertices) and the final extremity not in D.

Dijkstra Algorithm

```
Dijkstra(G,s)
1     $D \leftarrow s$; $A(s) \leftarrow \epsilon$; $x_1 \leftarrow s$; $d(s) \leftarrow 0$; $k \leftarrow 1$
2     for each ($x \in V$ and $x \neq s$)
3         do $d(x) \leftarrow \infty$
4     while ($k < n$ and $d(x_k) < \infty$)
5         do for each $u \in E$ s.t. $I(u) = x_k$ and $T(u) \notin D$
6             $x \leftarrow T(u)$
7             if $d(x) > d(x_k) + w(u)$
8                 then $d(x) \leftarrow d(x_k) + w(u)$
9                     $A(x) \leftarrow u$
10        Choose $x \notin D$ s.t. $d(x) = \min\{d(y), y \notin D\}$
11        $k \leftarrow k + 1$
12        $x_k \leftarrow x$
13     $D \leftarrow D \cup \{x_k\}$
```

Complexity

- line 2-3 $O(n)$
- line 5-9 $O(n)$
- line 10 to find the minimum $O(n^2)$
- total: $O(n^2)$

2- Graphs without circuit. DAG. Bellman Algorithm

```
Bellman(G,s)
1   $D \leftarrow s$; $A(s) \leftarrow \epsilon$; $x \leftarrow s$; $d(s) \leftarrow 0$
2   for each ($x \in V$ and $x \neq s$)
3       do $d(x) \leftarrow \infty$
4   while it exists $x \notin D$ with all its predecessors in $D$
5       do $d(x) \leftarrow \min_{u/T(u)=x, I(u) \in D}\{d(I(u)) + w(u)\}$
7           let $\tilde{u}$ s.t. $d(x) = \min_{u/T(u)=x, I(u) \in D}\{d(I(u)) + w(u)\}$
8               $A(x) \leftarrow \tilde{u}$
9               $D \leftarrow D \cup \{x\}$
```

Complexity : $O(n^2)$

$$G = (V, E, w)$$
$$\texttt{de\_num}(1) = s$$

Bellman-with-topological-sort(G,s)

```
1   $\mathcal{A}(s) \leftarrow \epsilon$
2   $d(s) \leftarrow 0$
3   for each $x \in V$, $x \neq s$
4           do $d(x) \leftarrow +\infty$
5   for $j \leftarrow 2$ to $n$
6           do $y \leftarrow \texttt{de\_num}(j)$
7                   $d(y) \leftarrow min_{u/\vec{xy}=u}[d(x) + w(u)]$
8                   let $\tilde{u}$ such that $d(y) = d(x) + w(\tilde{u})$
9                   $\mathcal{A}(y) \leftarrow \tilde{u}$
```

Complexity : $O(n^2)$

All-pairs shortest paths
1-Floyd Algorithm

We assume that the graph $G = (V, E)$ has a weight function ω. $V = \{1, 2, \dots, n\}$ and G does not contain absorbing circuit. We proceed step by step. We consider the paths linking i and j such that all intermediate vertices are in $\{1, \cdots, k\}$ at the step k. we denote by:

- $d_k(i, j)$ the weight of a shortest path using only intermediates vertices belonging to $\{1, \cdots, k\}$
- $pred_k(i, j)$ the predecessor of j in such a path.

We use the $n \times n$ matrix A such that :

$$a_{ij} = \begin{cases} \omega(\overrightarrow{ij}) & si\, \overrightarrow{ij} \in E \\ +\infty & si\, \overrightarrow{ij} \notin E \end{cases}$$

At each step k $a_{ij} = d_k(i,j)$

And the $n \times n$ matrix P :

$$P = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{nn} & \cdots & x_{nn} \end{pmatrix}$$

where $P_{ij} = pred_k(i,j)$ At the beginning $P_{ij} = i$

Floyd(G,w)

```
for $i \leftarrow 1$ to $n$
    do for $j \leftarrow 1$ to $n$
            $A[i,j] \leftarrow \omega(i,j)$
            $P[i,j] \leftarrow i$
do for $k \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n$
        do for $j \leftarrow 1$ to $n$
            do if $A[i,k] + A[k,j] < A[i,j]$
                then $A[i,j] \leftarrow A[i,k] + A[k,j]$
                    $P[i,j] \leftarrow P[k,j]$
```

Complexity : $O(n^3)$

Existence of paths between all pairs of vertices 1-Matrix

Let $G = (V, E)$ be a directed graph with $V = \{1, 2, \dots, n\}$. We want to know if it exists a path from i to j for each couple (i, j) of vertices of G.

Definition

Let $G = (V, E)$ be a directed graph. We define the boolean matrix:

$$A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$$

$$a_{i,j} = 1 \Leftrightarrow \vec{ij} \in E$$

Proposition

Let $p \geq 2$ be a integer, we define

$$A^{(p)} = (a_{ij}^{(p)})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$$

$$A^{(p)} = A^{(p-1)}.A$$

We have : $a_{ij}^{(p)} = 1 \Leftrightarrow$ *it exists a path from* i *to* j *with a length equal to* p

2- Transitive closure- Warshall Algorithm

Definition

Let $G = (V, E)$ be a directed graph. The transitive closure of G is the graph $G^* = (V, E^*)$ such that:

$$\vec{ij} \in E^* \Leftrightarrow \text{it exists a path linking } i \text{ to } j \text{ in } G$$

We will use the boolean matrix:

$$A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$$

for $i = j$ $a_{i,j} = 1$

$$a_{i,j} = 1 \Leftrightarrow \vec{ij} \in E$$

Warshall Algorithm

Warshall(G)

```
1    for $i \leftarrow 1$ to $n$
2        do for $j \leftarrow 1$ to $n$
3            do $A[i,j] \leftarrow c(i,j)$
4    do for $k \leftarrow 1$ to $n$
5        do for $i \leftarrow 1$ to $n$
6            do for $j \leftarrow 1$ to $n$
7                do $A[i,j] \leftarrow (A[i,j] \vee (A[i,k] \wedge A[k,j]))$
```

Complexity : $O(n^3)$

Chapter 6- Maximum Flow
1- Definitions

Definition (Network)

Let $G = (V, E)$ be a directed graph.

- $c : E \rightarrow \mathbb{R}^+$. For each $e \in E$, $c(e)$ is a nonnegative capacity.
- s and p two vertices of G, s is the source, p the sink of the network.
- a special edge $u_r = ps$ is the back edge, $c(u_r) = +\infty$

Flow

Definition

A flow is a real-valued function $f : E \longrightarrow \mathbb{R}^+$ such that:

$$\forall x \in V : \sum_{u/T(u)=x} f(u) = \sum_{u/I(u)=x} f(u)$$

Maximum flow problem

Let $G = (V, E, u_r, c)$ be a network ($c(u_r) = +\infty$).

Maximum flow problem

Find a real-valued function $f : E \to \mathbb{R}^+$ such that:

- f is a flow
- $\forall u \in E : 0 \leq f(u) \leq c(u)$
- $f(u_r)$ is maximum under the previous two conditions

Linear programming formulation

Let A be the incidence matrix of the graph G.

Maximum Flow Problem

$$(F_M)\begin{cases} Af = 0 \\ 0 \leq f \leq c \\ f(u_r) = z(Max) \end{cases}$$

Ford-Fulkerson Algorithm

Let f be a flow, we say that the flow is feasible if $\forall u \in E : 0 \leq f(u) \leq c(u)$.
We begin with a feasible flow . Let P be a path linking s to p with its edges in $E \setminus \{u_r\}$. We denote by P^+ the set of edges of P oriented in the direction s to p and P^- the set of edges of P oriented in the reverse direction. By starting from s we search a path P linking s and p containing no saturated edges. An edge is saturated in the two following cases:

- $u \in P^+$ and $f(u) = c(u)$
- $u \in P^-$ and $f(u) = 0$

A such path P with no saturated edges is called an augmenting path.

We define:

- $\delta_1 = \min_{u \in P^+}\{c(u) - f(u)\}$
- $\delta_2 = \min_{u \in P^-}\{f(u)\}$
- $\delta = \min\{\delta_1, \delta_2\}$

We define a new real-valued function f' such that:

- $f'(u) = f(u)$ if $u \notin P \cup \{u_r\}$
- $f'(u) = f(u) + \delta$ if $u \in P^+$
- $f'(u) = f(u) - \delta$ if $u \in P^-$
- $f'(u_r) = f(u_r) + \delta$

Conclusion

f' is a feasible flow and $f'(u_r) > f(u_r)$

Marking

f is a feasible flow of G.

We start from s. We mark the vertices of G.

s is marked. $\delta(s) = 0$

- if x is marked and it exits $u = \vec{xy}$ with y not marked and $f(u) < c(u)$ then
  - $\delta(y) = \min\{\delta(x), c(u) - f(u)\}$
  - $\mathcal{A}(x) = u$, y is marked.
- if x is marked and it exits $u = \vec{yx}$ with y not marked and $f(u) > 0$ then
  - $\delta(y) = \min\{\delta(x), f(u)\}$
  - $\mathcal{A}(x) = u$, y is marked.

Ford-Fulkerson Algorithm

$$G = (V, E, u_r, c)$$

```
**Marking Algorithm**$(G, f)$      /* $f$ is a feasible flow */
1    $\delta \leftarrow \delta(s) \leftarrow c(u_r) - f(u_r)$
2    $Y \leftarrow \{s\}$
3    while $p \notin Y$ and $\delta > 0$
4          do if it exists $u = \vec{xy}$, $x \in Y$, $y \notin Y$ et $f(u) < c(u)$
5                then  $Y \leftarrow Y \cup \{y\}$
6                      $\mathcal{A}(y) \leftarrow u$
7                      $\delta(y) \leftarrow \min\{\delta(x), c(u) - f(u)\}$
8                else
9                   if it exists $u = \vec{yx}$, $x \in Y$, $y \notin Y$ et $f(u) > 0$
10                        then $Y \leftarrow Y \cup \{y\}$
11                             $\mathcal{A}(y) \leftarrow u$
12                             $\delta(y) \leftarrow \min\{\delta(x), f(u)\}$
13                        else $\delta \leftarrow 0$
14    if $p \in Y$ then $\delta \leftarrow \delta(p)$
```

Ford-Fulkerson Algorithm

$$G = (V, E, u_r, c)$$

```
Changing flow$(G, f, \delta)$
1    $x \leftarrow p$
2    $f(u_r) \leftarrow f(u_r) + \delta$
3    while $x \neq s$
4        do $u \leftarrow \mathcal{A}(x)$
5            if $x = T(u)$ then $f(u) \leftarrow f(u) + \delta$
6                               $x \leftarrow I(u)$
7                          else
8                               $f(u) \leftarrow f(u) - \delta$
6                               $x \leftarrow T(u)$
```

Ford-Fulkerson Algorithm

G=(X,U,c)

```
Maximum flow Algorithm($G$)
1    for each $u \in U$ do $f(u) \leftarrow 0$
2    Iterate { Marking
3                stop $\delta = 0$
4                Changing flow
5              }
```

Max Flow - Min Cut

Let $G = (V, E, c, u_r)$ and $u_r = \vec{ps}$.

Definition

We say that $\mathcal{C} \subset E$ is a separating cut for s and p, if we can find $Y \subset V$ such that $s \in Y$ and $p \notin Y$ and

$$\mathcal{C} = \{u \in E : I(u) \in Y, T(u) \notin Y\}$$

.

Definition

The capacity of a cut $\mathcal{C}$ separating s and p is:

$$c(\mathcal{C}) = \sum_{u \in \mathcal{C}} c(u)$$

Max Flow - Min Cut

Theorem

For any feasible flow f of $G = (V, E, c)$ ($c(u_r) = +\infty$) and for any cut $\mathcal{C}$ separating s and p we have:

$$f(u_r) \leq c(\mathcal{C})$$

Theorem

For any cocyle $\Omega(Y)$:
f is a flow of $G = (V, E)$ $\iff$

$$\textit{For any cocyle } \Omega(Y) : \sum_{u \in \Omega(Y)^+} f(u) = \sum_{u \in \Omega(Y)^-} f(u)$$

Max Flow - Min Cut

Corollary

If the Marking Algorithm terminates without reaching p then the obtained flow by Ford-Fulkerson Algorithm is an optimal solution for the Maximum Flow problem in $G = (V, E, c, u_r)$.

Theorem (Max Flow - Min cut)

The maximum value for a feasible flow of $G = (V, E, c)$ $(f(u_r))$ *is equal to the minimum capacity of a cut separating* s *and* p.

Maximum matching in a bipartite graph

Definition

An undirected graph $G = (V, E)$ is bipartite if $V = Y \cup Z$ with $Y \cap Z = \emptyset$ and Y, Z are independant sets.

Theorem

A graph G is bipartite $\iff$ G does not contain cycles of odd length.

Definition

A matching is a subset K of E such the edges belonging to K are independant.

Definition

A maximum matching is a matching that saturates the maximum number of vertices.

Definition

A transversal of a graph $G = (V, E)$ is a set of vertices $T \subset V$ such that $V \setminus T$ is a stable set, i.e. any edge of G has at least one extremity in T.

Observation

Let K be a matching and T be a transversal of G then $|K| \leq |T|$.

Observation

If $|K| = |T|$ then K is a maximum matching and T is a minimum transversal.

Problem

Find a maximum matching in a bipartite graph.

Maximum matching in a bipartite graph

Let $G = (V, E)$ be a bipartite graph. $E = Y \cup Z$ with $Y \cap Z = \emptyset$ and Y, Z are stable sets of vertices.
We define a network $R = (V', E', c)$:

- $V' = Y \cup Z \cup \{s, p\}$, $\vec{E} = \{\vec{y}z : xy \in E\}$
- $E' = \vec{E} \cup \{\vec{s}y : y \in Y\} \cup \{\vec{z}p : z \in Z\}$
- $c : E' \to \mathbb{R}^+ \cup \{+\infty\}$
  - $\forall u = yz : y \in Y, z \in Z$, $c(u) = c(u_r) = +\infty$
  - $\forall u = sy : y \in Y$ or $u = zp : z \in Z$, $c(u) = 1$.

Maximum matching in a bipartite graph

Let f be a maximum flow:

- $f(u_r) = \mathcal{C}(\Omega^+(Y))$, $\Omega^+(Y)$ is a minimum cut.
- For any edge $u \in E'$, $f(u) = 1$ or $f(u) = 0$.
- Any two edges $u = \vec{y}z$ and $u' = \vec{y}'z'$ such that $f(u) = f(u') = 1$ are not adjacent.
- $K = \{\vec{u} \in E : f(\vec{u}) = 1\}$ is a matching.
- $f(u_r) = \sum_{u \in E} f(u) = |K|$ is the cardinality of the matching.

Maximum matching in a bipartite graph

Let S such that $\Omega^+(S)$ is the minimum cut obtained by the algorithm. $s \in S$, $p \notin S$

- $\Omega^+(S)$ has finite capacity, then it does not contain any edge $u = \vec{y}z$. It implies that if $u = \vec{y}z$ then $y \in \bar{S}$ either $z \in S$.
- $T = (\bar{S} \cap Y) \cup (S \cap Z)$ is a transversal of G.
- the edges of $\Omega^+(S)$ are
  - $\vec{s}y : y \in \bar{S} \cap Y$
  - $\vec{z}p : z \in S \cap Z$
- All this $|T|$ edges have a capacity equal to 1, then $c(\mathcal{C}) = |T|$
- $c(\mathcal{C}) = |T| = f(u_r)$ and $f(u_r) = |K|$, then $|T| = |K|$.
- K is a maximum matching of G.

Maximum matching in a bipartite graph

Theorem

In a bipartite graph, the cardinality of a minimum transversal is equal to the cardinality of a maximum matching.

Menger Theorem

Theorem

Let s and p be two vertices of a directed graph G. The maximum number P_{sp} of edge disjoint paths linking s and p is equal to the minimum cardinality $N_{s,p}$ of a set of edges separating p from s