Graph Algorithms

Table of contents

Chapter 1: Asymptotic notations

Chapter 2: Graph: Definitions

Chapter 2: Traversal of graphs

Chapter 4: Minimum Spanning tree

Chapter 5: Shortest path

Chapter 6: Maximum flow problem

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.
$T : U \to X$, $a \mapsto$ the terminal extremity
$I : U \to X$, $a \mapsto$ the initial extremity

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

Definition

f $\vec{xy} \in U$ 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 $X = \{1, 2, \ldots, n\}$

Definition (Adjacency Matrix)

t 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.

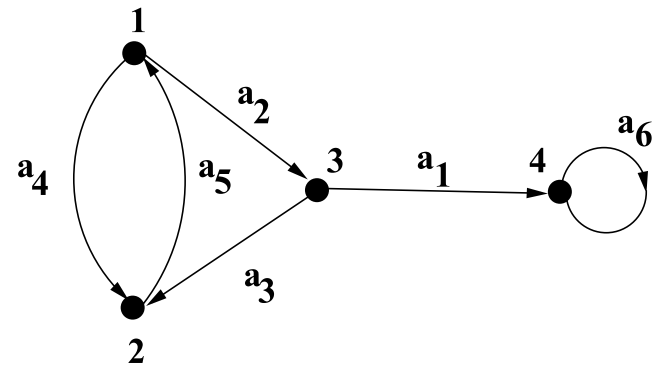

Figure: Graph 2

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

ncidence Matrix

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

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

ncidence Matrix

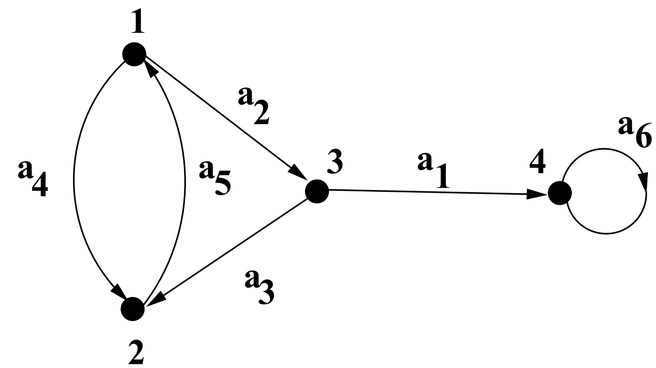

Figure: Graph 2

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

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 $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

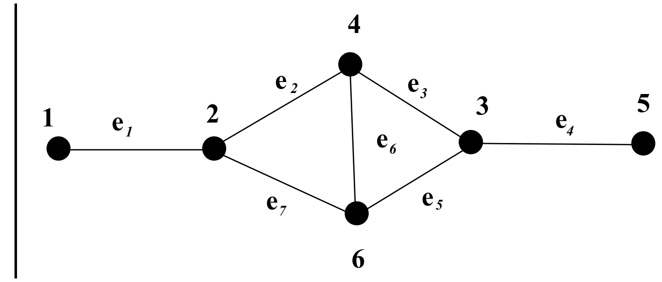

Figure: Graph 3

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, Neighboorood

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

f $G = (X, U)$ 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

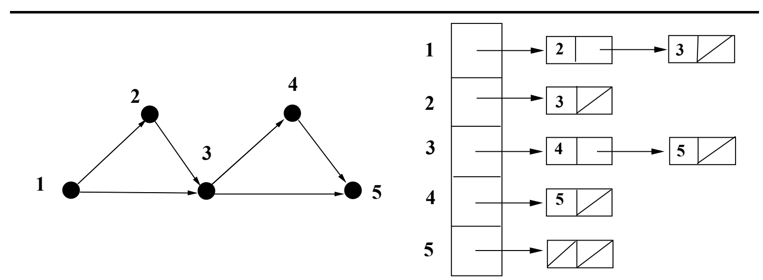

Figure: List of successors

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).

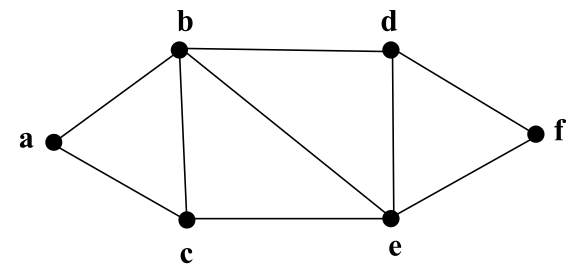

Figure: Walk, Path, Cycle

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)
for each $u \in V \setminus s$
  do color[u] $\leftarrow$ WHITE
    d[u] $\leftarrow \infty$
    Π[u] $\leftarrow$ NIL
color[s] $\leftarrow$ GRAY
d[s] $\leftarrow$ 0
Π[s] $\leftarrow$ NIL
F $\leftarrow \emptyset$
ENQUEUE(F,s)
while F $\neq \emptyset$
  do $u \leftarrow$ DEQUEUE(F)
    for each $v \in$ Adj[u]
      do if color[v]=WHITE
        then color[v] $\leftarrow$ GRAY
          d[v] $\leftarrow$ d[u] +1
          Π[v] $\leftarrow u$
          ENQUEUE(F,v)
    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 $\Pi[v]$ followed by the edge $\Pi[v]v$.

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)
    for  each vertex u ∈ V
         do color[u] ← WHITE
                  Π[u] ← NIL
    time ← 0
    for  each vertex u ∈ V
         do if color[u]=WHITE
                  then DFS-Visit(u)
DFS-Visit(u)
  color[u] ← GRAY /* u is discovered */
  d[u] ← time ← time + 1
  for each v ∈ Adj[u]
      do if color[v]=WHITE
               then  Π[v] ← u
                     DFS-Visit(v)
  Color[u] ← BLACK
  f(u) ← time ← 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 $\iff$ 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 $\Longleftrightarrow$ 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 $\Longleftrightarrow$ 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.*