Graph Algorithms - Lecture 3

October 18, 2024

Table of contents

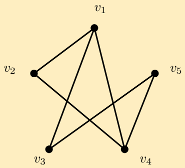
Associated matrices - Adjacency matrix

Let G be a graph with $V(G) = \{v_1, \ldots, v_n\}$. The adjacency matrix of G is the matrix $A = (a_{ij})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_{n\times n}(\{0,1\})$, where

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ adjacent} \\ 0, & \text{otherwise} \end{cases}.$$

Example

A graph and its adjacency matrix.



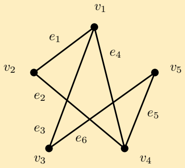
$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Let G be a graph with $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. The incidence matrix of G is the matrix $B = (b_{ij})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_{n \times m}(\{0, 1\})$, where

$$b_{ij} = \begin{cases} 1, & \text{if } e_j \text{ is incident with } v_i \\ 0, & \text{otherwise} \end{cases}.$$

Example

A graph and its incidence matrix.



$$= \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Associated matrices

The eigenvalues, eigen vectors and the characteristic polynomial of the adjacency matrix are called eigenvalues, eigen vectors, and, respectively, characteristic polinomial of the graph. These are the objects of study for spectral graph theory.

For digraphs, similar matrices can be defined with entries in $\{-1, 0, 1\}$ in order to point out the direction of arcs.

Let G be a digraph with $V(G) = \{v_1, \dots, v_n\}$ and $E(G) = \{e_1, \dots, e_m\}$. The vertex-arc incidence matrix of G is the matrix $B = (b_{ij})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_{n \times m}(\{-1, 0, 1\})$, where

$$b_{ij} = \begin{cases} 1, & \text{if } e_j \text{ is incident from } v_i \\ -1, & \text{if } e_j \text{ is incident into } v_i \\ 0, & \text{otherwise} \end{cases}.$$

Data structures for adjacency matrix

Let $G = (V, E)$ be a (di)graph with $V = \{1, 2, \ldots, n\}$.

- If $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$ is the adjacency matrix of G then, representing it as a a 2-dimensional array, we need $\mathcal{O}(n^2)$ time for initialization (depending on the programming language).
- Hence any algorithm that represents G with adjacency matrix has $\Omega(n^2)$ time (and space) complexity.
- Testing if two vertices are adjacent is done in $\mathcal{O}(1)$ time, but passing through the set of neighbors $N_G(u)$ (or $N_{G^+(u)}$), for a certain vertex $u \in V$, needs $\Omega(n)$ time - unpractical for large sparse graphs.

Data structures for adjacency lists

Let $G = (V, E)$ be a (di)graph with $V = \{1, 2, \ldots, n\}$ and $|E| = m$.

- Every vertex $u \in V$ has a list, $A(u)$, of its neighbors in G:
  - when G is a graph $A(u) = N_G(u)$;
  - if G is a digraph, then $A(u) = N_G^+(u) = \{v \in V \ : \ uv \in E\}$;
- If G is a graph, then each edge $uv \in E$ generates two elements in the adjacency lists: one in $A(u)$ and one in $A(v)$; the space needed is $\mathcal{O}(n + 2m)$.
- If G is a digraph, then the space needed is $\mathcal{O}(n + m)$.
- Adjacency lists can be implemented using linked lists or using arrays.
- Testing if a vertex u is adjacent to another vertex v in G needs $\mathcal{O}(d_G(u))$ time, but passing through the set of neighbors $N_G(u)$ (or $N_G^+(u)$), for an arbitrary vertex $u \in V$, can be done in $\Omega(d_G(u))$ time (and not in $\mathcal{O}(n)$ time as in the case of adjacency matrix).

Graph traversal or graph search is an algorithmic paradigm specifying a systematic method to pass through the set of vertices reachable by paths starting from a specified vertex in a (di)graph.

Given a (di)graph $G = (\{1, \ldots, n\}, E)$ and $s \in V(G)$

"efficiently" generate the set

$$S = \{u \in V(G) \ : \ \text{there is a path from } s \text{ to } u \text{ in } G\}.$$

G will be represented with adjacency lists, because, during the traversal process, we need to handle in an efficient way the set of neighbors of the current vertex.

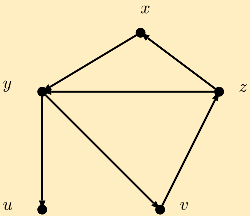
for $v \in V$ do
    $label(v) \leftarrow -1$; $parent(v) \leftarrow -1$;
$label(s) \leftarrow 0$; $parent(s) \leftarrow 0$;
create queue Q containing s;
while $Q \neq \varnothing$ do
    $u \leftarrow pop(Q)$;
    for $v \in A(u)$ do
        if $label(v) < 0$ then
            $label(v) \leftarrow label(u) + 1$;
            $parent(v) \leftarrow u$; $push(Q, v)$;

Properties of BFS. It is not difficult to prove that:

- $S = \{u \in V \ : \ label(u) \geqslant 0\}$;
- $\forall u \in V, label(u) = d_G(s, u)$ (the distance in G from s to u);
- Variable parent defines the bfs-tree associated to the search from s: if G is a graph then the bfs-tree is a spanning tree of the connected component containing s; if G is a digraph then the bfs-tree is an arborescence (directed rooted tree in which all arcs point away from the root s).
- The time complexity of $BFS(s)$ is $\mathcal{O}(n_S + m_S)$, where $n_S = |S| \leqslant |V| = n$, and $m_S = |E([S]_G)| \leqslant |E|$ (this follows easily by observing that each node in the adjacency list of a vertex from S is accessed exactly once).

Example

Two BFS's on the same digraph (starting from z and from v):



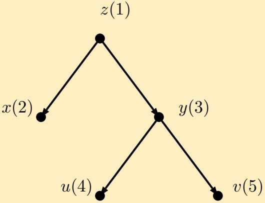
$A(x) = [y]$

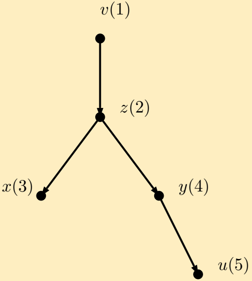
$A(y) = [u, v]$

$A(z) = [x, y]$

$A(u) = \emptyset$

$A(v) = [z]$





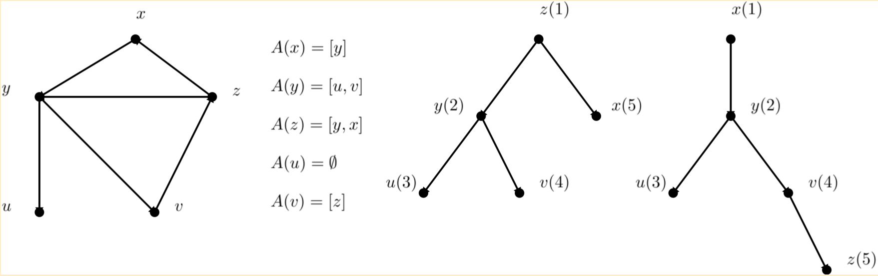
for $v \in V$ do
    $label(u) \leftarrow -1$; $parent(u) \leftarrow -1$;
$label(s) \leftarrow 0$; $parent(s) \leftarrow 0$;
create stack S containing s; $n_S \leftarrow 0$;
while $S \neq \varnothing$ do
    $u \leftarrow top(S)$;
    if $((v \leftarrow next[A(u)]) \neq NULL)$ then
        if $label(v) < 0$ then
            $n_S + +$; $label(v) \leftarrow n_S$;
            $parent(v) \leftarrow u$; $push(S, v)$;
    else
        $delete(S, u)$;

Properties of DFS. It is not difficult to prove that:

- $\{u \in V : label(u) \geqslant 0\}$ is exactly the set S of the vertices reachable by paths from s;
- $\forall u \in V, label(u) =$ visiting time of u (s has visiting time 0);
- Variable parent defines the dfs-tree associated to the search from s;
- The time complexity of $DFS(s)$ is $\mathcal{O}(n_S + m_S)$, where $n_S = |S| \leqslant |V| = n$, and $m_S = |E([S]_G)| \leqslant |E|$ (this follows easily by observing that each node in the adjacency list of a vertex from S is accessed exactly once).

Example

Two DFS's on the same digraph (starting from z and from x):



Shortest paths - Notations

Let $G = (V, E)$ be a digraph, with $V = \{1, \dots, n\}$.

- Each directed edge (arc) $e \in E$ has associated a cost $a(e) \in \mathbb{R}$ (weight, length etc).
- If G is represented with adjacency lists, then $a(ij)$ is a field in the node of adjacency list of i (representing the arc ij).
- For ease of notation we will use the representation of G with the cost-adjacency matrix $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$, where

$$a_{ij} = \begin{cases} a(ij), & \text{if } ij \in E \\ \infty, & \text{otherwise} \end{cases}.$$

Shortest paths - Notations

- Here, ∞ denotes a big real number with respect to actual edge costs (e.g., $\infty > n \cdot \max_{ij \in E} a(ij)$) and we suppose that $\infty \pm a = \infty$, $\infty + \infty = \infty$.
- It is also possible to use ∞ as an unsuccessful access to the data structure used to represent the matrix A.
- For $i, j \in V$, the set of all paths in G from i to j is denoted by $\mathcal{P}_{ij}$:

$$\mathcal{P}_{ij} = \{P \;:\; P \text{ is a path from } i \text{ to } j\}.$$

- If $P_{ij} \in \mathcal{P}_{ij}$, $P_{ij} : (i =)v_0, v_0v_1, v_1, \ldots, v_{r-1}, v_{r-1}v_r, v_r(= j)$, then

$$V(P_{ij}) = \{v_0, v_1, \ldots, v_r\}, E(P_{ij}) = \{v_0v_1, \ldots, v_{r-1}v_r\}.$$

- The cost of $P_{ij} \in \mathcal{P}_{ij}$ is

$$a(P_{ij}) = 0 + \sum_{uv \in E(P_{ij})} a_{uv}.$$

- In particular $a(P_{ii}) = 0$.

Single-pair shortest path problem.

P1 Given $G = (V, E)$ q digraph; $a : E \to \mathbb{R}$; $s, t \in V, s \neq t$.
Find $P^*_{st} \in \mathcal{P}_{st}$, such that $a(P^*_{st}) = \min\{a(P_{st}) \,:\, P_{st} \in \mathcal{P}_{st}\}$.

Single-source shortest path problem.

P2 Given $G = (V, E)$ a digraph; $a : E \to \mathbb{R}$; $s \in V$.
Find $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, s. t. $a(P^*_{si}) = \min\{a(P_{si}) \,:\, P_{si} \in \mathcal{P}_{si}\}$

All-pairs shortest path problem.

P3 Given $G = (V, E)$ a digraph; $a : E \to \mathbb{R}$.
Find $P^*_{ij} \in \mathcal{P}_{ij}, \forall i, j \in V$, s. t. $a(P^*_{ij}) = \min\{a(P_{ij}) \,:\, P_{ij} \in \mathcal{P}_{ij}\}$

Remarks 1

- The cost-adjacency matrix representation of the pair (G, a) implies that $\mathcal{P}_{ij} \neq \varnothing, \forall i, j \in V(G)$: if $a(P_{ij}) < \infty$, then P_{ij} is a true path in G, and if $a(P_{ij}) = \infty$, then P_{ij} is not a path in G but it is a path in the complete symmetric digraph obtained from G by adding all missing arcs (with ∞ costs).
- It follows that all sets over which a minimum cost element is required in the problems P1 - P3 are non-empty and finite and all minimum paths required are well-defined.
- The algorithms for solving the problem P1 are obtained from those solving the problem P2 by adding an (obvious) stopping test.
- The problem P3 can be solved by iterating any algorithm for the problem P2. We'll see that there are more efficient solutions.

1. Communication Networks. The digraph $G = (V, E)$ represents a communication network between the nodes in V and with E modeling the set of directed links between nodes.

- If $a(e) \geqslant 0$ $(\forall e \in E)$ represents the length of the direct connection between the extremities of e, then the problems P1 - P3 are natural shortest paths problems.
- If $a(e) \geqslant 0$ $(\forall e \in E)$ represents the time needed for the direct connection between the extremities of e, then the problems P1 - P3 are natural fastest paths problems.
- If$a(e) \in (0, 1]$ $(\forall e \in E)$ represents the probability that the direct connection between the extremities of e works properly, and we suppose that edges work properly independent of each other, then the problems P1 - P3 become most reliable paths problems:

If $P_{ij} \in \mathcal{P}_{ij}$ for some pair $i, j \in V$, then the probability that this path works properly is (by the independence assumption)

$$Prob(P_{ij}) = \prod_{e \in E(P_{ij})} a(e).$$

By taking $a'(e) = -\log a(e)$,

$$\log Prob(P_{ij}) = \log \left(\prod_{e \in E(P_{ij})} a(e) \right) = -\sum_{e \in E(P_{ij})} a'(e).$$

By the monotonicity of the log function it follows that the problems P1 - P3 with costs a', give the most reliable paths in communication network.

2. PERT Networks - Critical Path method (CPM). PERT (Path Evaluation and Review Technique) is a method to analyze (especially) the completion time of each task in a given complex project.

- Let $P = \{A_1, A_2, \ldots, A_n\}$ be atomic activities of a large project P (n is big). $(P, <)$ is a partially ordered set, where $A_i < A_j$ if $i \neq j$ and activity A_j can be started only after the activity A_i was finished.
- For each activity A_i, its completion time t_i is given (estimated).

Find a scheduling of the activities of the project to minimize its total completion (calendar) time.

Shortest paths - Applications

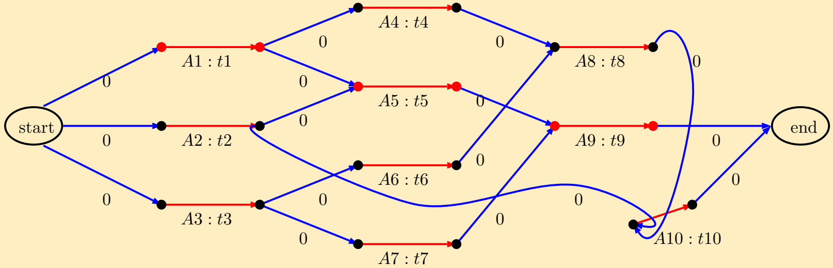
We can associate a directed acyclic graph to the problem in this way:

- to each activity A_p ($p \in \{1, \ldots, n\}$ we add an arc $i_p j_p$ with cost $a(i_p j_p) = t_p$;
- the node i_p corresponds to the beginning event of A_p and the node j_p is associated to the finishing event of it;
- if an activity A_k can start only after the activity A_p we add the arc $j_p i_k$ (dummy activity).

- The construction of the digraph is finished after adding a node s corresponding to the start event of the project linked by arcs si_p for each activity A_p with no incoming arcs, and a node t corresponding to the terminal event of the project linked by arcs j_pt for each activity A_p with no outgoing arcs.
- In the obtained digraph, the maximum cost of a path from s to t is equal to the minimum completion time of the project.
- A maximum cost path is called a critical path since any delay of an activity on this path infers a delay of the whole project.

Shortest paths - Applications

Example



3. Knapsack problem $(0-1)$. We are given a knapsack of size $b \in \mathbb{N}$, and n objects of sizes $a_1, \ldots, a_n \in \mathbb{N}$. Also is known the profit $p_i \in \mathbb{N}$ of inserting the object i ($i \in \{1, \ldots, n\}$) into the knapsack. We are asked to choose a filling of the knapsack of maximum total profit.

Let x_i, for $i \in \{1, \ldots, n\}$, be a boolean variable having the meaning that $x_i = 1$ if and only if the object i is inserted into the knapsack. Then the knapsack problem can be stated as

$$\max \left\{ \sum_{i=1}^{n} p_i x_i \ : \ \sum_{i=1}^{n} a_i x_i \leqslant b, x_i \in \{0,1\}, \forall i = \overline{1,n} \right\}.$$

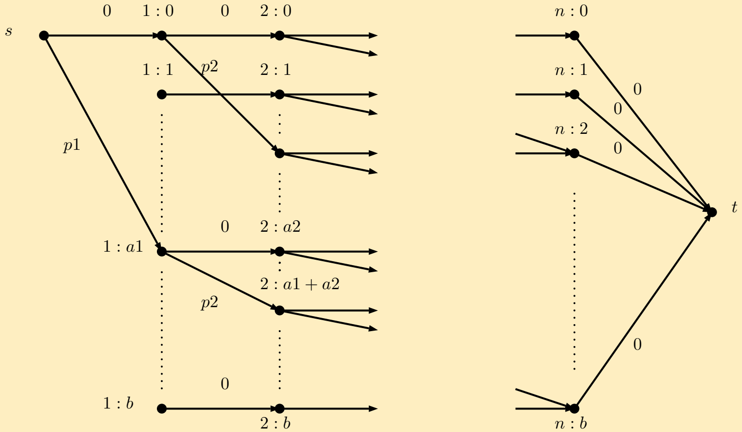
- Let $G = (V, E)$ be the digraph with $V = \{s\} \cup V_1 \cup \ldots \cup V_n \cup \{t\}$, where $V_i = \{i^0, i^1, \ldots, i^b\}$ is associated to object i, $i = \overline{1, n}$.
- The arcs of G and their costs are:
  - $s1^0$ and $s1^{a_1}$ with $a(s1^0) = 0, a(s1^{a_1}) = p_1$ (either the object 1 is added to the knapsack with profit p_1 and filling level a_1 , or it is not added, with the profit and fitting level 0).
  - $(i-1)^j i^j$ with $a((i-1)^j i^j) = 0$, $\forall i = \overline{2, n}, \forall j = \overline{0, b}$ (the object i is not inserted into knapsack: from the filling with the first $i-1$ objects and filling level j, we pass to a filling with the first i objects, without object i; the filling level remains j and the additional profit is 0).
  - If $j - a_i \geqslant 0$, then we have also the arc $(i-1)^{j-a_i} i^j$ with $a((i-1)^{j-a_i} i^j) = p_i$ (we can arrive at the filling level j by inserting the object i to a filling with the first $i-1$ objects, with the filling level $j - a_i$).

- $n^j t$ with $a(n^j t) = 0$, $\forall j = \overline{0, b}$.

Remarks 2

- Each path from s to t in G corresponds to a subset of objects with the filling level $\leqslant b$ and with the total profit equal to the cost of the path. Since, conversely, to each filling of the knapsack corresponds a path from s to t in G, it follows that the knapsack problem can be solved by finding a path of maximum cost in the directed acyclic graph G.
- The static description given above for G can be transformed into a procedural one, giving the usual dynamic programming solution. Note that the problem is NP-hard (the order of G could be exponential in the input size of the problem).

Example



P2 Given $G = (V, E)$ digraph; $a : E \to \mathbb{R}$; $s \in V$.
Find $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, s. t. $a(P^*_{si}) = \min\{a(P_{si}) \ : \ P_{si} \in \mathcal{P}_{si}\}$

Theorem 1

Let G be a digraph, $s \in V(G) = \{1, \ldots, n\}$ and $a : E(G) \to \mathbb{R}$, s. t.

(I) $$a(C) > 0, \text{ for all } C \text{ cycle in } G.$$

Then $(u_1, \ldots, u_n)$ is a solution of the system of equations

(B) $$\begin{cases} u_s = & 0 \\ u_i = & \min\limits_{j \neq i}(u_j + a_{ji}) \end{cases} \text{ if and only if}$$

$\forall i \in V(G), \exists P^*_{si} \in \mathcal{P}_{si}$ *s. t.* $u_i = a(P^*_{si}) = \min\{a(P) \ : \ P \in \mathcal{P}_{si}\}$.

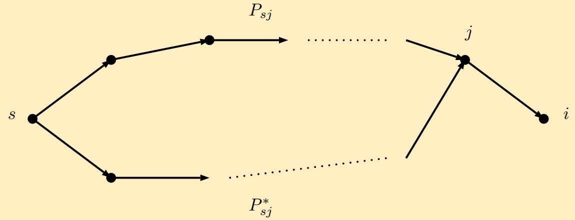
Proof:

"$\Leftarrow$". Let P^*_{si} be an optimal solution of P2 and $u_i = a(P^*_{si})$.
The hypothesis (I) implies that $u_s = 0$, i.e., the first equation of the system (B) is satisfied. For $i \neq s$, the path P^*_{si} has a penultimate vertex j. If P_{sj} is the path from s to j determined on P^*_{si} by j, we have

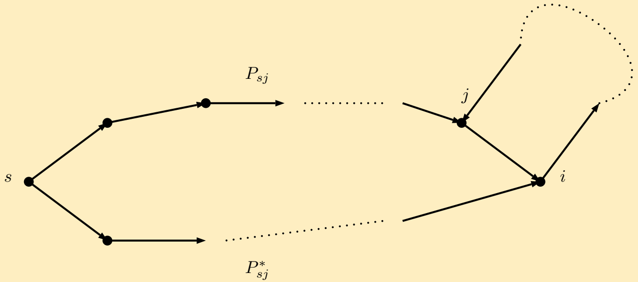
$$u_i = a(P^*_{si}) = a(P_{sj}) + a_{ji} \geqslant a(P^*_{sj}) + a_{ji} = u_j + a_{ji}.$$

Now, we show that $u_i = u_j + a_{ji}$. Suppose that $u_i > u_j + a_{ji}$, i. e., $a(P_{sj}) > a(P^*_{sj})$.

Case 1. $i \notin V(P^*_{sj})$. Then $P^1 = P^*_{sj} \circ (j, ji, i) \in \mathcal{P}_{si}$ and $a(P^1) = a(P^*_{sj}) + a_{ji} < a(P_{sj}) + a_{ji} = a(P^*_{si})$, a contradiction ($P^*_{si}$ is a shortest path).



Case 2. $i \in V(P^*_{sj})$. Let $P^*_{sj} = P_{si} \circ P_{ij}$, the two paths determined by the vertex i on P^*_{sj}. Then the cost of the cycle $C = P_{ij} \circ (j, ji, i)$ is $a(C) = a(P_{ij}) + a_{ji} = a(P^*_{sj}) - a(P_{si}) + a_{ji} = u_j + a_{ji} - a(P_{si})$ which is $\leqslant u_j + a_{ji} - a(P^*_{si}) = u_j + a_{ji} - u_i < 0$, a contradiction (the hypothesis (I) is violated).



Hence the "$\Leftarrow$" part of the theorem is proved.

Remark 1

We proved above that if j is the vertex before i on a shortest path from s to i, then the path from s to j determined by j on this shortest path is a shortest path from s to j. Inductively, it follows:

Bellman's Principle of Optimality: If P^*_{si} is a shortest path from s to i, then $\forall j \in V(P^*_{si})$, if $P^*_{si} = P_{sj} \circ P_{ji}$, then P_{sj} (respectively P_{ji}) is a shortest path from s to j (respectively from j to i).

"$\Rightarrow$". We show that if $(u_1, \ldots, u_n)$ is a solution of (B), then

(a) $\exists P_{si} \in \mathcal{P}_{si}$ such that $u_i = a(P_{si})$, $\forall i \in V$.

(b) $\forall i \in V$, $u_i = \min\{a(P) : P \in \mathcal{P}_{si}\}(= a(P_{si}))$.

(a) If $i = s$, then $u_s = 0$ and the path P_{ss} satisfies $a(P_{ss}) = 0 = u_s$.
If $i \neq s$, let us consider the following algorithm

$v \leftarrow i$; $k \leftarrow 0$;
while $v \neq s$ do
  find w s. t. $u_v = u_w + a_{wv}$;
  // there exists such a w since u_v satisfies (B)
  $i_k \leftarrow v$; $k++$; $v \leftarrow w$;

The algorithm find the path $P : (s =)i_{k+1}, i_{k+1}i_k, i_k, \ldots, i_1, i_1 i_0, i_0(= i)$ with $P \in \mathcal{P}_{si}$ and

$$a(P) = a(i_{k+1} i_k) + \cdots + a(i_1 i_0) =$$

$$(u_{i_k} - u_{i_{k+1}}) + (u_{i_{k-1}} - u_{i_k}) + \cdots + (u_{i_0} - u_{i_1}) =$$

$$u_{i_0} - u_{i_{k+1}} = u_i - u_s = u_i,$$

In each while iteration $w \notin \{i_0, \ldots, i_{k-1}\}$ (else we get a cycle of cost 0, violating the hypothesis (I)).

Note that, with the notations in the above algorithm, we have $u_i = u_{i_1} + a_{i_1 i}$.

(b) Let $\overline{u}_i = a(P^*_{si})$, $\forall i \in V$. By the above proof, $\overline{u}_i$, $i = \overline{1, n}$ satisfy the system (B).

Suppose that $u = (u_1, \ldots, u_n) \neq (\overline{u}_1, \ldots, \overline{u}_n) = \overline{u}$.

Since $u_s = \overline{u}_s = 0$, it follows that there is $i \neq s$ such that $u_i \neq \overline{u}_i$ and $\forall j \in V(P_{si})$, $j \neq i$, $u_j = \overline{u}_j$, where P_{si} is the path built in (a) for $\overline{u}_i$.

Then $u_i > \overline{u}_i = \overline{u}_{i_1} + a_{i_1 i} = u_{i_1} + a_{i_1 i} \geqslant u_i$ (the first inequality holds by the choice of i, the second holds since u_i satisfies (B)).
The contradiction found shows that $u = \overline{u}$, that is the components of u are shortest paths costs.

$\square$

Remark 2

From the above proof it follows that for solving P2 is sufficiently to found a solution of the system of equations (B). The corresponding shortest paths can be obtained as in part (a) of the proof: if we have $u_i = u_k + a_{ki}$ then k is the vertex before i on the shortest path from s to i (of cost u_i). In the algorithm that solves (B) we maintain an array $before[1..n]$ with entries from $V \cup \{0\}$ with the final meaning $before[i]$ =the vertex before i on a shortest path from s to i. The vertices of this path can be found in $\mathcal{O}(n)$ time by constructing the sequence $i, before[i], before[before[i]], \ldots, s$.

Remark 3

If the algorithms solving the system of equations (B) circumvent (by the maintenance of the array *before*) 0-cost cycles, then the problem P2 is solved, even the uniqueness of the solution is lost. Hence these algorithms will solve P2 under the hypothesis

(I') $\qquad a(C) \geqslant 0,$ for all C cycle in G.

Remark 4

If, in the problems P1 - P3, G is a graph and not a digraph, we can use the algorithms for digraphs by replacing each (undirected) edge of G with a symmetric pair of arcs, each having the cost of the edge. Note that this approach works only for non-negative costs of the edges (if an edge has negative cost, then the 2-cycle formed by the two symmetric arcs replacing the edge has negative cost, hence hypothesis (I') is not satisfied).

Remark 5

Since the sets $\mathcal{P}_{ij}$ are finite (and non-empty), we can consider problems similar to P1 - P3 by replacing min with max.

Remark 6

The use of the obvious relation $\max_{x\in A} x = -\min_{x\in A}(-x)$, by replacing the costs a_{ij} by $-a_{ij}$, works only for digraphs in which, for each cycle C, we have $a(C) \leqslant 0$ (in particular, this approach works for digraphs without cycles). If the digraph has cycles, longest path problems are in general NP - hard.

Exercise 1.

We say that a graph $G = (V, E)$ is sparse if $m \leqslant cn^2/\log n$ ($n = |V|, m = |E|$). The reason is that we can represent the adjacency matrix A of G using only $\mathcal{O}(n^2/\log n)$ memory space such that the answer to a query "$a(i, j) = 1$?" could be done in $\mathcal{O}(1)$ time.

Describe such a representation.

Exercise 2.

Show that there is no ordering $e_1, e_2, \ldots, e_{10}$ of the edges of the graph K_5, such that: e_{10} and e_1 are not adjacent and e_i and e_{i+1} are not adjacent for each $1 \leqslant i \leqslant 9$.

Exercise 3. Let $G = (V, E)$ be a graph of order n and size m with adjacency matrix A. From the set of all 2^m possible orientations of all its edges we choose one and consider the vertex-arc incidence matrix $Q \in \mathcal{M}_{n\times m}(\{-1, 0, 1\})$.

$$q_{ve} = \begin{cases} -1, & \text{if } v \text{ is the initial extremity of the arc } e \\ 1, & \text{if } v \text{ is the final extremity of the arc } e \\ 0, & \text{if } e \text{ is not incident with } v. \end{cases}.$$

Prove that $A + QQ^T$ is a diagonal matrix and find the combinatorial interpretation of its diagonal elements.

Exercise 4. Let $D = (V, E)$ be a digraph with $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Let $B = (b_{ij}) \in \mathcal{M}_{n\times m}(\{-1, 0, 1\})$ the incidence matrix of D, where

$$b_{ij} = \begin{cases} 1, & \text{if } e_j \text{ is incident from } v_i \\ -1, & \text{if } e_j \text{ is incident into } v_i \\ 0, & \text{otherwise} \end{cases}.$$

Prove that $\det(M) \in \{-1, 0, 1\}$ for every square submatrix of B (that is, B is a totally unimodular matrix).

Exercise 5. Let $G = (S, T; E)$ be a bipartite graph with $V = S \cup T = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Let $B = (b_{ij}) \in \mathcal{M}_{n\times m}(\{0,1\})$ the incidence matrix of G, where

$$b_{ij} = \begin{cases} 1, & \text{if } e_j \text{ is incident with } v_i \\ 0, & \text{otherwise} \end{cases}.$$

Prove that $\det(M) \in \{-1, 0, 1\}$ for every square submatrix of B (i. e., B is a totally unimodular matrix).

Exercise 6. Let G be a graph with n vertices and m edges.

(a) Prove that if G is bipartite if and only if G doesn't contain odd (induced) cycles.

(b) Devise an $\mathcal{O}(n + m)$ time complexity algorithm for deciding if a given graph is bipartite. (An algorithm for recognizing bipartite graphs.)

Exercise 7. Let M_G be the edge-vertex-edge incidence matrix of a given graph $G = (V, E)$, that is $M_G = (m_{ij})_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant n}}$, where

$$V = \{v_1, v_2, \ldots, v_n\}, E = \{e_1, e_2, \ldots, e_m\}.$$

$$m_{ij} = \begin{cases} 1 & \text{if } e_i \text{ is incident with } v_j \\ 0 & \text{otherwise} \end{cases}$$

(a) Prove that if T is a tree, then by removing from M_T a column corresponding to a given vertex we get a square non-singular matrix.

(b) Prove that if C is a cycle, then M_C is a non-singular matrix if and only if C is odd.

Exercise 8. Let $G = (V, E)$ be a graph with n vertices and $m \geqslant 1$ edges. Consider the following algorithm:

$G' \leftarrow G$;
while ($\exists u \in V(G')$ such that $d_{G'}(u) < m/n$) do
  $G' \leftarrow G' - u$;
return G';

(a) Determine a time complexity of an efficient implemenation of the above algorithm.

(b) Prove that the returned graph, G', cannot be a null graph.

(c) Show that any given graph contains a path of length $\geqslant m/n$.

Exercise 9. The diameter of graph G is the greatest distance between any two vertices in G. Two vertices form a diametral pair of vertices if the distance between them equals the diameter. Show that the following algorithm finds a diametral pair of vertices in a given tree T:

1. starting from some vertex of T, perform a *bfs* (Breadth First Search) algorithm; let u be the last visited vertex by this search.
2. perform another *bfs* on T starting from vertex u; let v be the last visited vertex.
3. return the pair (u, v).

Exercise 10. Show that the DFS traversal can be used to devise an $\mathcal{O}(n)$ algorithm to find an even cycle in a 3-regular graph of order n.

Exercise 11.

(a) Show that for a bipartite graph with n vertices and m edges we have $4m \leqslant n^2$.

(b) Write an $\mathcal{O}(n+m)$ time complexity algorithm which has to test if a graph (with n vertices and m edges) is the complement of a bipartite graph.

Exercise 12. Show that a graph G is bipartite if and only if every induced subgraph H satisfies the inequality: $2\alpha(H) \geqslant |H|$.

Exercise 13. Let $G = (S, T; E)$ a bipartite graph and $X \in \{S, T\}$. G is called X-chain if we can order the vertices of X: $x_1, x_2, \ldots x_k$ ($|X| = k$) such that

$$N_G(x_1) \supseteq N_G(x_2) \supseteq \ldots \supseteq N_G(x_k)$$

(a) Show that G is S-chain if and only if is T-chain.

(b) Suppose that G (which is bipartite) has order n, dimension m, and is represented using the adjacency lists. Describe a S-chain recognition algorithm with $\mathcal{O}(n + m)$ time complexity.

Exercise 14. Let G be a graph; we denote by $b(G)$ the graph obtained from G by inserting a new vertex in the middle of every edge of G.

(a) Show that $b(G)$ is a bipartite graph.

(b) Show that $G \simeq H$ if and only if $b(G) \simeq b(H)$. Using this result prove that the isomorphism testing between two graphs can be polynomial-time reduced at the isomorphism testing between two bipartite graphs.

Exercise 15. Let G a bipartite graph; prove that G is connected if and only if it has only one bipartition with stable sets.