

1 Graph Theory Vocabulary

- Associated matrices

2 Data structures

3 Path problems in digraphs

- (Di)graph traversal

- Shortest paths

4 Exercises for the 4th seminar (october 20-24 week)

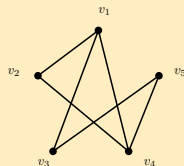
Associated matrices - Adjacency matrix

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



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

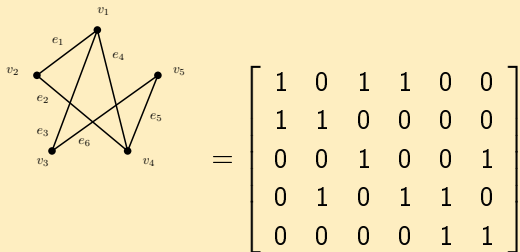
Associated matrices - Incidence matrix

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



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 $B = (b_{ij})_{1 \leq i, j \leq n} \in \mathcal{M}_{n \times m}(\{-1, 0, 1\})$, where

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

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

Path problems - (Di)graph traversal

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

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 $G = (\{1, \dots, n\}, E)$ 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.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Path problems - Breadth-First Search (BFS)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

```
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  $Q \neq \emptyset$  do
     $u \leftarrow pop(Q)$ ;
    for  $v \in A(u)$  do
        if  $label(v) < 0$  then
             $label(v) \leftarrow label(u) + 1$ ;
             $parent(v) \leftarrow u$ ;  $push(Q, v)$ ;
```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Path problems - Breadth-First Search (BFS)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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

- $S = \{u \in V : \text{label}(u) \geq 0\}$;
- $\forall u \in V, \text{label}(u) = d_G(s, u)$ (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 $n_S = |S| \leq |V| = n$, and $m_S = |E([S]_G)| \leq |E|$ (this follows easily by observing that each node in the adjacency list of a vertex from S is accessed exactly once).

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Path problems - Depth-First Search (DFS)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

```
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  $S \neq \emptyset$  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
             $delete(S, u)$ ;
```

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Path problems - Depth-First Search (DFS)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

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

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

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Main shortest path problems

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Single-pair shortest path problem.

P1 Given $G = (V, E)$ a digraph; $a : E \rightarrow \mathbb{R}$; $s, t \in V, s \neq t$.
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; $a : E \rightarrow \mathbb{R}$; $s \in V$.
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; $a : E \rightarrow \mathbb{R}$.
Find $P_{ij}^* \in \mathcal{P}_{ij}, \forall i, j \in V$, s. t. $a(P_{ij}^*) = \min \{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Remarks 1

- The cost-adjacency matrix representation of the pair (G, a) implies that $P_{ij} \neq \emptyset, \forall i, j \in V(G)$: 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 required minimum paths 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 $a(e) \geq 0$ ($\forall e \in E$) represents the length of the direct connection between the extremities of e , then the problems P1 - P3 are natural **shortest paths problems**.
- If $a(e) \geq 0$ ($\forall e \in E$) 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**:

Shortest paths - Applications

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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)

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

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

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

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

Graph Algorithms - C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Shortest paths - Applications

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

2. **PERT Networks - Critical Path method (CPM).** **PERT** (**P**ath **E**valuation and **R**eview **T**echnique) is a method to analyze the completion time of each task in a given complex project.

- Let $P = \{A_1, A_2, \dots, A_n\}$ 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.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Shortest paths - Applications

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

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

Let x_i , for $i \in \{1, \dots, n\}$, 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

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Shortest paths - Applications

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

- Let $G = (V, E)$ be the digraph with $V = \{s\} \cup V_1 \cup \dots \cup V_n \cup \{t\}$, where $V_i = \{i^0, i^1, \dots, i^b\}$ 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 $j - a_i \geq 0$, 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 $j - a_i$).

Shortest paths - Applications

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

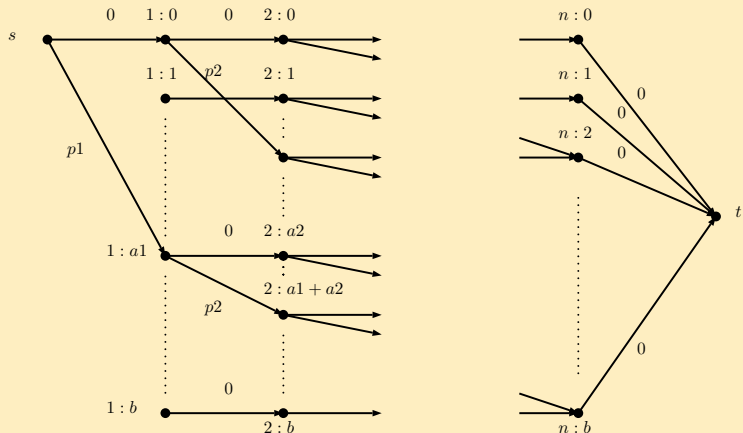
Remarks 2

- Each path from s to t in G corresponds to a subset of objects with the filling level $\leq b$ 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).

Shortest paths - Applications

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Example



- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Shortest paths - Solving problem P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

P2 Given $G = (V, E)$ digraph; $a : E \rightarrow \mathbb{R}$; $s \in V$.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Theorem 1

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

(I) $a(C) > 0$, for all C cycle in G .

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

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

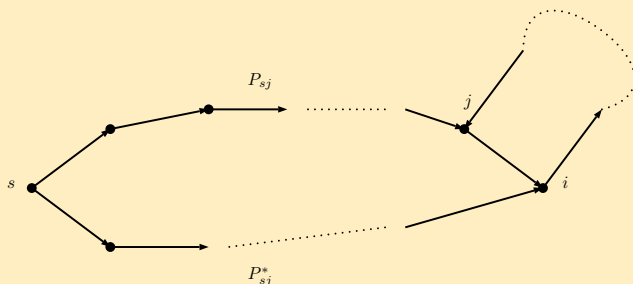
$\forall i \in V(G), \exists P_{si}^* \in \mathcal{P}_{si}$ s. t. $u_i = a(P_{si}^*) = \min \{a(P) : P \in \mathcal{P}_{si}\}$.

Shortest paths - Solving problem P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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 $\leq u_j + a_{ji} - a(P_{si}^*) = u_j + a_{ji} - u_i < 0$, a contradiction (the hypothesis (I) is violated).

Graph Algorithms - C. Croitoru - Graph Algorithms - C. Croitoru - Graph Algorithms - C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.



Shortest paths - Solving problem P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

" \Rightarrow ". We show that if (u_1, \dots, 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. $u_v = u_w + a_{vw};$

 // there exists such a w since u_v satisfies (B)

$i_k \leftarrow v; k ++; v \leftarrow w;$

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

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Shortest paths - Solving problem P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

$$\begin{aligned} a(P) &= a(i_{k+1}i_k) + \cdots + a(i_1i_0) = \\ (u_{i_k} - u_{i_{k+1}}) &+ (u_{i_{k-1}} - u_{i_k}) + \cdots + (u_{i_0} - u_{i_1}) = \\ u_{i_0} - u_{i_{k+1}} &= u_i - u_s = u_i, \end{aligned}$$

In each **while** iteration $w \notin \{i_0, \dots, i_{k-1}\}$ (else we get a cycle of cost 0, violating the hypothesis (I)). (Note that, with the notations in the above algorithm, we have $u_i = u_{i_1} + a_{i_1i}$.)

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

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

Run the algorithm from (a) for \bar{u}_i , $i = \overline{1, n}$ and let \bar{P}_{s_i} , $i = \overline{1, n}$, be the resulted paths.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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

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).

Exercises for the 4th seminar

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercise 1.

We say that a graph $G = (V, E)$ is **sparse** if $m \leq cn^2 / \log n$ ($n = |V|$, $m = |E|$). 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.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exercise 2.

Show that there is no ordering e_1, e_2, \dots, 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 $1 \leq i \leq 9$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercise 3.

Consider an ordering $E = \{e_1, e_2, \dots, e_m\}$ of the edges of a connected graph $G = (V, E)$ of order n . For every subset $A \subseteq E$ we define $x^A \in GF^m$ the m -dimensional characteristic vector: $x_i^A = 1 \Leftrightarrow e_i \in A$. (GF^m is the m -dimensional 0 - 1 vector space over \mathbb{Z}_2 .) Show that:

- the subset of the characteristic vectors corresponding to all the cuts in G completed with zero vector is a subspace X of GF^m ;
- the subset of the characteristic vectors corresponding to all cycles from G spans a subspace, U , of GF^m which is orthogonal on X ;
- $\dim(X) \geq n - 1$;
- $\dim(U) \geq m - n + 1$;
- the above inequalities are in fact equalities.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercise 4. 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.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercise 6. Let $G = (S, T; E)$ be a bipartite graph with $V = S \cup T = \{v_1, v_2, \dots, v_n\}$ and $E = \{e_1, e_2, \dots, e_m\}$. 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 7. Let G be a graph with n vertices and m edges.

- Prove that G is bipartite if and only if G doesn't contain odd (induced) cycles.
- Devise an $\mathcal{O}(n + m)$ time complexity algorithm for deciding if a given graph is bipartite. (An algorithm for recognizing bipartite graphs.)

Exercise 8. Let M_G be the edge-vertex-edge incidence matrix of a given graph $G = (V, E)$, that is $M_G = (m_{ij})_{\substack{1 \leq i \leq m, \\ 1 \leq j \leq n}}$, where

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

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

- (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.

Exercises for the 4th seminar

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercise 9. Let $G = (V, E)$ be a graph with n vertices and $m \geq 1$ 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  $G';$ 
```

- Determine a time complexity of an efficient implementation of the above algorithm.
- Prove that the returned graph, G' , cannot be a null graph.
- Show that any given graph contains a path of length $\geq m/n$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercises for the 4th seminar

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Exercise 11. 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 .

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercise 12.

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

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exercise 13. Show that a graph G is bipartite if and only if every induced subgraph H satisfies the inequality: $2\alpha(H) \geq |H|$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Exercise 14. 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, \dots, x_k ($|X| = k$) such that

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

- (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.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercises for the 4th seminar

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercise 15. 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.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Exercise 16. Let G a bipartite graph; prove that G is connected if and only if it has only one bipartition with stable sets.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *