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 - Lecture 9

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

December 6, 2024

- 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 Algorithms *

Table of contents

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

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

Definition

A preflow in $R = (G, s, t, c)$ is a function $x : E(G) \to \mathbb{R}$ such that

(i) $0 \leqslant x_{ij} \leqslant c_{ij}, \forall ij \in E$;

(ii) $\forall i \neq s,\ e_i = \sum_{ji \in E} x_{ji} - \sum_{ij \in E} x_{ij} \geqslant 0$.

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

e_i (for $i \in V \setminus \{s, t\}$) is called the excess in node i. If $i \in V \setminus \{s, t\}$ and $e_i > 0$, then i is an active node. If $ij \in E$, x_{ij} will be referred as the flow on the arc ij.

If in R there are no active nodes, then the preflow x is a flow with $v(x) = e_t$.

Idea of preflow algorithms: an initial preflow in R is transformed by changing the flow on the arcs in a flow with the property that there are no augmenting paths in R w.r.t. it.

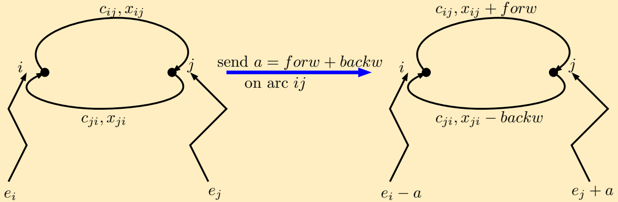
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

G is represented using adjacency lists. We will suppose that if $ij \in E$, then $ji \in E$ too (otherwise, we add the arc ji with capacity 0). Hence, G is a symmetric digraph.

If x is a preflow in R and $ij \in E$, then the residual capacity of ij is

$$r_{ij} = c_{ij} - x_{ij} + x_{ji},$$

(representing the additional flow that can be sent from node i to node j using the arcs ij and ji).



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

In the following, sending flow from i to j means increasing the flow on the arc ij or decreasing the flow on the arc ji.

An A-path in R w.r.t. preflow x, is any path in G having all arcs with strictly positive residual capacity.

A distance function in R w.r.t. preflow x is a function $d : V \to \mathbb{Z}_+$ s.t.

(D_1) $d(t) = 0$,

(D_2) $\forall ij \in E, r_{ij} > 0 \Rightarrow d(i) \leqslant d(j) + 1$.

Remarks

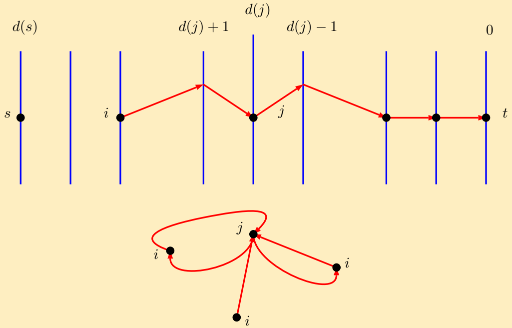
- If P is an A-path w.r.t. preflow x in R from i to t, then $d(i) \leqslant$ *length*(P) (the arcs of P have positive residual capacity and we repeatedly use (D_2)). It follows that $d(i) \leqslant \tau_i$, where τ_i denotes the minimum length of an A-path from i to t.

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

Remarks

- As usual, we denote by $A(i)$ the adjacency list of the node i.

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



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

Definition

Let x be a preflow in R and d a distance function w.r.t. x. An arc $ij \in E$ is called admissible *if* $r_{ij} > 0$ and $d(i) = d(j) + 1$.

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 -

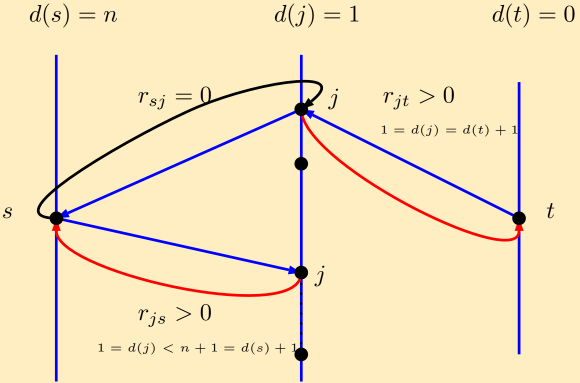
If R is a flow network, we consider the following initialization procedure, which builds in $\mathcal{O}(m)$ a preflow x and a distance function d w.r.t. x.

procedure *initialization*();
for $(ij \in E)$ do
  if $(i = s)$ then
    $x_{sj} \leftarrow c_{sj}$
  else
    $x_{ij} \leftarrow 0$;
$d[s] \leftarrow n$; $d[t] \leftarrow 0$;
for $(i \in V - \{s, t\})$ do
  $d[i] \leftarrow 1$;

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

Note that after the execution of this procedure, we have $r_{sj} = 0$, $\forall sj \in A(s)$. Hence the condition (D_2) is not affected by taking $d(s) = n$. For all arcs ij, (D_2) is fulfilled:



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

The choice of $d(s) = n$ means: there are no A-paths from s to t w.r.t. x (otherwise, if P is such a path, its length must be at least $d(s) = n$, which is impossible).

If this will be an invariant of the preflows algorithms; hence, when x will become a flow, it will be a maximum value flow.

Let us consider the following two procedures:

procedure $push(i)$; // i is a node different from s, t
choose an admissible arc $ij \in A(i)$;
"send" $\delta = \min\{e_i, r_{ij}\}$ (flow units) from i to j;

If $\delta = r_{ij}$ then we have a saturated push, otherwise we have an unsaturated push.

procedure $relabel(i)$; // i is a node different from s, t
$d[i] \leftarrow \min\{d[j] + 1 \,:\, ij \in A(i) \text{ and } r_{ij} > 0\}$;

Preflows - General scheme of a preflow algorithm

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 -

initialization;
while ($\exists$ active nodes in R) do
    choose an active node i;
    if ($\exists$ admissible arcs in $A(i)$) then
        $push(i)$;
    else
        $relabel(i)$;

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

Lemma 1

"d is distance function w.r.t. preflow x" is an invariant of the above algorithm. At each call of $relabel(i)$, $d(i)$ increases.

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

Proof. We already proved that the procedure initialization builds a preflow x and a distance function d w.r.t. x. We show that if the pair (d, x) satisfies (D_1) and (D_2) before an while iteration, then after this iteration the two conditions are fulfilled too.

We have two cases, depending on which procedure push or relabel is called in the current while iteration:

push(i) is called: the only pair that can violate (D_2) is $d(i)$ and $d(j)$. Since ij is admissible, at the *push*(i) call we have $d(i) = d(j) + 1$. After the call of *push*(i), the arc ji could have now $r_{ji} > 0$ (without being before the call), but the condition $d(j) \leqslant d(i) + 1$ is obviously satisfied.

relabel(i) is called: the update of $d(i)$ is such that (D_2) is fulfilled for each arc ij with $r_{ij} > 0$. Since *relabel*(i) is called when $d(i) < d(j) + 1$, $\forall ij$ with $r_{ij} > 0$, it follows that, after the call, $d(i)$ increases (with at least 1). $\square$

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

In order to show that the algorithm terminates, it is necessary to show that (during the execution) if a node i is active then in its adjacency list, $A(i)$, there is at least one arc ij cu $r_{ij} > 0$. This follows from the next lemma.

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

Lemma 2

If i_0 is an x-active node in R, then there is an $i_0 s$ A-path w.r.t. x.

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

Proof. If x is a preflow in R, then x can be decomposed $x = x^1 + x^2 + \cdots + x^p$, where each x^k has the property that the set $A^k = \{ij : ij \in E, x^k_{ij} \neq 0\}$ is

(a) the set of the arcs of a path from s to t, or

(b) the set of arcs of a path from s to an active node, or

(c) the set of arcs of a cycle.

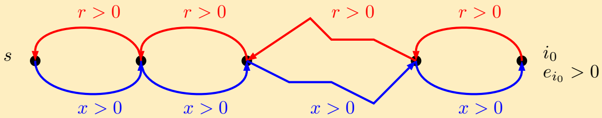
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Moreover, in the cases (a) and (c), x^k is a flow. (The proof is algorithmic: we construct firstly the sets of type (a), than those of type (c) and (b); at each stage, we search the converse of a path of the type (a) or (c) (or (b)); the preflow obtained is subtracted from the current one; since the excesses of the nodes are non-negative, the construction can be realized, whenever the current preflow is non null; the construction is finite, since the number of arcs on which the current flow is 0, increases at each stage of it.)

Since i_0 is an active node in R w.r.t. x, it follows that the case (b) will occur for the node i_0 (the cases (a) and (c) does not affect the excess in the node i_0). The converse arcs of this path have positive residual capacity (see the figure bellow), therefore they form the required A-path. $\square$

- 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



Corollary 1

$\forall i \in V$, $d(i) < 2n$.

Proof. Indeed, if i has not been relabelled, then $d(i) = 1 < 2n$. Otherwise, before the call of *relabel*(i), i is an active node, hence by Lemma 2, there is an is A-path P with *length*$(P) \leqslant n - 1$. By (D_2), it follows that, after the relabel, $d(i) \leqslant d(s) + n - 1 = 2n - 1$ ($d(s) = n$ is never changing). □

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

Corollary 2

The total number of calls of the procedure relabel is not greater than $2n^2$.

Proof. Indeed, there are $n-2$ nodes which can be relabelled. Each of them can be relabelled at most of $2n-1$ times (by Lemma 1, the above Corollary, and the initial distance d). □

Corollary 3

The total number of saturated pushes is not greater than nm.

Proof. Indeed, when an arc ij becomes saturated, we have $d(i) = d(j) + 1$. After that, the algorithm cannot send flow on this arc until it sends flow on the arc ji, when we have $d'(j) = d'(i) + 1 \geqslant d(i) + 1 = d(j) + 2$.

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

Proof cont'd. Hence this flow change on the arc ij does not occur until $d(j)$ increases with 2. It follows that an arc can not become saturated more than n times and there are no more than nm saturated pushes (since the total number of arcs is m). $\square$

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

Lemma 3

(Goldberg and Tarjan, 1986). *The total number of unsaturated pushes is not greater than $2n^2m$.*

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

Proof. Omitted.

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

Lemma 4

The preflow algorithm outputs a maximum value flow x in R.

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 -

Proof. By Lemmas 1 and 3 and by Corollary 3 of Lemma 2, the algorithm terminates in at most $\mathcal{O}(n^2m)$ while iterations. Since $d(s) = n$ is never modified, it follows that there is no augmenting path w.r.t. the flow x obtained, hence x is of maximum value: if P is an augmenting path (in the support graph of G), then, by replacing along P each backward arc with its symmetric arc, we get an A-path from s to t, hence $n = d(s) \leqslant d(t) + n - 1 = n - 1$. $\square$

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

Instead of proving Lemma 3, we will present the Ahuja & Orlin algorithm (1988) that uses a scaling method to reduce the number of unsaturated pushes from $\mathcal{O}(n^2m)$ (Goldberg & Tarjan algorithm) to $\mathcal{O}(n^2 \log U)$.

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

Suppose that $c_{ij} \in \mathbb{N}$ and $U = \max_{ij \in E}(1 + c_{ij})$. Let $K = \lceil \log_2 U \rceil$.

Algorithm's idea: We have $K+1$ stages. For each stage p, with p taking successively the values $K, K-1, \ldots, 1, 0$, the following two conditions are fulfilled:

(a) at the beginning of stage p, $e_i \leqslant 2^p$, $\forall i \in V \setminus \{s, t\}$.

(b) during the stage p, the procedures push-relabel are used in order to eliminate the active nodes, i, with $e_i \in \{2^{p-1} + 1, \ldots 2^p\}$.

By the definition of K, in the first stage ($p = K$), property (a) holds (after the initialisation of the preflow we have $e_i = c_{si}$ or $e_i = 0$, for each $i \neq s, t$, hence $e_i \leqslant U$), and, if property (b) will be maintained during the algorithm (if the integrality of excesses is also maintained during the algorithm), it follows that, after stage $K+1$, the excess of each node $i \in V \setminus \{s, t\}$ is 0, therefore we have a flow of maximum value.

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

In order to maintain the condition (b) during the algorithm, the general scheme of a preflow algorithm is adapted as follows:

- each stage p starts by constructing the list $L(p)$ of all nodes $i_1, i_2, \ldots, i_{l(p)}$ with excesses $e_i > 2^{p-1}$, sorted non-decreasing by d (this can be done with a hash-sorting in $O(n)$ time, since $d(i) \in \{1, 2, \ldots, 2n-1\}$).
- the active node selected for push-relabel during the stage p will be the first node in $L(p)$. It follows that, if a push is done on the admissible arc ij, then $e_i > 2^{p-1}$ and $e_j \leqslant 2^{p-1}$ (because $d(j) = d(i) - 1$ and i is the first node in $L(p)$). If δ, the flow sent from i to j by $push(i)$, is limited to $\delta = \min(e_i, r_{ij}, 2^p - e_j)$, then (since $2^p - e_j \geqslant 2^{p-1}$) it follows that an unsaturated push sends at least 2^{p-1} unit flows.

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

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

After the execution of *push(i)* the excess from node j (the only one for which the excess can increase) will be $e_j + \min(e_i, r_{ij}, 2^p - e_j) \leqslant e_j + 2^p - e_j \leqslant 2^p$, therefore (b) holds.

- the stage p is over when the list $L(p)$ becomes empty.

In order to find efficiently an admissible arc for doing the push, or to inspect all the arcs leaving a node i for doing relabel, we will organize the adjacency lists $A(i)$ as follows:

- each list's node contains: the node j, x_{ij}, r_{ij}, a pointer to the arc ji (from the adjacency list $A(j)$), and a pointer to the next node from the list $A(i)$.
- the list has associated an iterator to enable its traversal.

All these lists are constructed in $O(m)$ time, before the call of the procedure initialization.

initialization; $K \leftarrow \lceil \log_2 U \rceil$; $\Delta \leftarrow 2^{K+1}$;
for ($p = \overline{K, 0}$) do
    construct $L(p)$; $\Delta \leftarrow \Delta/2$;
    while ($L(p) \neq \varnothing$) do
        let i the first node in $L(p)$;
        search in $A(i)$ for an admissible arc;
        if (ij is the admissible arc found) then
            $\delta \leftarrow \min(e_i, r_{ij}, \Delta - e_j)$;
            $e_i \leftarrow e_i - \delta$; $e_j \leftarrow e_j + \delta$;
            "send" δ unit flows from i to j;
            if ($e_i \leqslant \Delta/2$) then
                delete i from $L(p)$;
            if ($e_j > \Delta/2$) then
                add j as the first node in $L(p)$;
        else
            compute $d[i] = \min\{d[j] + 1 \, : \, ij \in A(i) \text{ and } r_{ij} > 0\}$
            reposition i in $L(p)$;
            set the current pointer at the beginning of $A(i)$;

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

The complexity time of the algorithm is dominated by unsaturated pushes (all the remaining parts need $\mathcal{O}(nm)$ time).

Lemma 5

The number of unsaturated pushes is at most $8n^2$ in each stage of the scaling, hence the total number is $\mathcal{O}(n^2 \log U)$.

Proof. Let

$$F(p) = \sum_{i \in V, i \neq s,t} \frac{e_i \cdot d(i)}{2^p}.$$

At the begining of stage p, $F(p) < \sum_{i \in V} \frac{2^p \cdot (2n)}{2^p} = 2n^2$.

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

If, in the stage p, is executed *relabel*(i), then there are no admissible arcs ij, and $d(i)$ is increased with $h \geqslant 1$ units. $F(p)$ will increase by at most h, Since, $\forall i$, $d(i) < 2n$ it follows that $F(p)$ will increase (until the end of the p stage) at most up to $4n^2$.

If, in the stage p, is executed *push*(i), then this sends $\delta \geqslant 2^{p-1}$ on the admissible arc ij with $r_{ij} > 0$ and $d(i) = d(j) + 1$. Hence, after the push, $F(p)$ will have the value $F'(p) = F(p) - \dfrac{\delta \cdot d(i)}{2^p} + \dfrac{\delta \cdot d(j)}{2^p} =$

$$F(p) - \frac{\delta}{2^p} \leqslant F(p) - \frac{2^{p-1}}{2^p} = F(p) - 1/2.$$

This decrease cannot occur more than $8n^2$ times (because $F(p)$ can increase at most up to $4n^2$ and $F(p)$ is non-negative). Clearly, the number of unsaturated pushes is dominated by this number of decreases of $F(p)$.

- 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 *

Summarizing, we have:

Theorem

(Ahuja-Orlin, 1988) The Preflow algorithm with excesses scaling has time complexity $\mathcal{O}(nm + n^2 \log U)$.

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 Algorithms * C. Croitoru - Graph Algorithms

A. Finding a maximum cardinality matching and a maximum stable set in a bipartite graph.

Let $G = (V_1, V_2; E)$ be bipartite graph with n vertices and m edges. Consider the network $R = (G_1, s, t, c)$, where

- $V(G_1) = \{s, t\} \cup V_1 \cup V_2$;
- $E(G_1) = E_1 \cup E_2 \cup E_3$, with

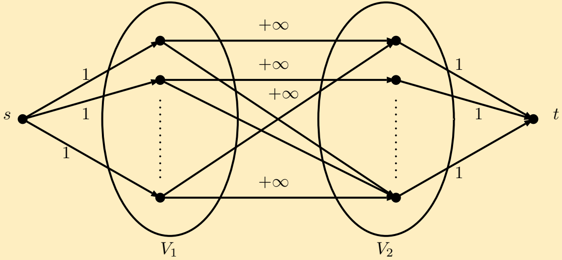
$$E_1 = \{sv_1 \ :\ v_1 \in V_1\}, E_2 = \{v_2 t \ :\ v_2 \in V_2\},$$

$$E_3 = \{v_1 v_2 \ :\ v_1 \in V_1, v_2 \in V_2, v_1 v_2 \in E(G)\},$$

- $c : E(G_1) \to \mathbb{N}$ defined by

$$c(e) = \begin{cases} 1, & \text{if } e \in E_1 \cup E_2 \\ \infty, & \text{if } e \in E_3 \end{cases}$$

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



If $x = (x_{ij})$ is an integral flow in R, then the set $\{ij \ : \ i \in V_1, j \in V_2$ and $x_{ij} = 1\}$ corresponds to a matching M^x in the bipartite graph G, with $|M^x| = v(x)$.

Conversely, any matching $M \in \mathcal{M}_G$ gives rise to a set of non-adjacent arcs in G_1; if on each such arc ij ($i \in V_1$, $j \in V_2$) we consider $x_{ij}^M = 1$ and $x_{si}^M = x_{jt}^M = 1$, and we put $x^M(e) = 0$ on any other arc, then the integral flow x^M satisfies $v(x^M) = |M|$.

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

Hence, if we solve the maximum flow problem in R (starting with the null flow), then we obtain in $\mathcal{O}(nm + n^2 \log n)$ time a maximum cardinality matching in G. (the constant replacing $+\infty$ must be an integer greater than the cardinality of any cut, e. g. $n^2 + 1$ - see below.)

Let (S, T) be the minimum capacity cut (obtained in $\mathcal{O}(m)$ time from the maximum flow found). By the Max-flow Min-cut Theorem, $c(S, T) = \nu(G)$.

Since $\nu(G) < \infty$, taking $S_i = S \cap V_i$ and $T_i = T \cap V_i$ ($i = \overline{1, 2}$), we have $|T_1| + |S_2| = \nu(G)$ and $X = S_1 \cup T_2$ is a stable set in G (in order to have $c(S, T) < \infty$). Moreover, $|X| = |V_1 \setminus T_1| + |V_2 \setminus S_2| = n - \nu(G)$. It follows that X is a maximum cardinality stable set, since $n - \nu(G) = \alpha(G)$) (by König theorem).

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 -

B. Recognizing digraphic degree sequences

Let us consider the following problem:

Given $(d_i^+)_{i=\overline{1,n}}$ and $(d_i^-)_{i=\overline{1,n}}$, is there a digraph $G = (\{1,\ldots,n\}, E)$ such that $d_G^+(i) = d_i^+$ and $d_G^-(i) = d_i^-$, $\forall i = \overline{1,n}$?

Obvious necessary conditions in order to have an "yes instance" are:

$$d_i^+ \in \mathbb{N}, 0 \leqslant d_i^+ \leqslant n-1 \text{ and } d_i^- \in \mathbb{N}, 0 \leqslant d_i^- \leqslant n-1, \forall i = \overline{1,n};$$

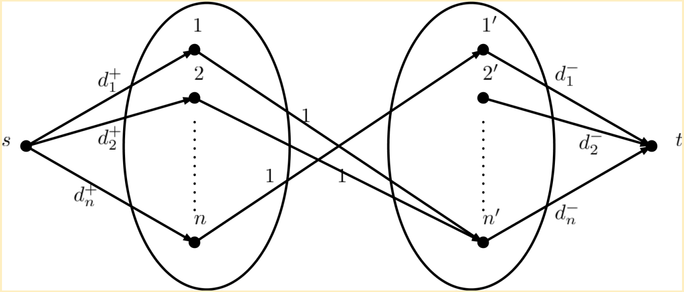
$$\sum_{i=1}^{n} d_i^+ = \sum_{i=1}^{n} d_i^- = m \text{ (where } m = |E|).$$

In this hypothesis, consider the bipartite flow network $R = (G_1, s, t, c)$.

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

G_1 is obtained from a complete bipartite graph $K_{n,n}$ with bipartition $(\{1, 2, \ldots, n\}, \{1', 2', \ldots, n'\})$, by removing the set of edges $\{11', 22', \ldots, nn'\}$ and by orienting each edge ij' $(\forall i \neq j \in \{1, 2, \ldots, n\})$ from i to j', and by adding two new vertices s, t, and all arcs si, $i \in \{1, 2, \ldots, n\}$ and $j't$, $j \in \{1, 2, \ldots, n\}$.
The capacity function: $c(si) = d_i^+$, $c(j't) = d_j^-$, $c(ij') = 1$, $\forall i, j = \overline{1, n}$.



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

If in R there is an integral flow, x, of maximum value m, then from each vertex i will leave exactly d_{ij}^{+} arcs, ij', on which $x_{ij'} = 1$, and in each vertex j' will enter exactly d_i^{-} arcs, ij', on which $x_{ij'} = 1$.

The desired digraph, G, is constructed by taking $V(G) = \{1, 2, \ldots, n\}$ and putting $ij \in E(G)$ if and only if $x_{ij'} = 1$.

Conversely, if G exists, then by inversing the above construction we obtain an integral flow in R of value m (hence, of maximum value).

It follows that, the recognition of digraph sequences (and digraph realization, for positive answer) can be done in $\mathcal{O}(nm + n^2 \log n) = \mathcal{O}(n^3)$.

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. Finding the edge-connectivity number of a graph

Let $G = (V, E)$ be a graph. For $s, t \in V$, $s \neq t$, we denote

- $p_e(s,t)$ = maximum number of edge-disjoint paths from s to t in G,
- $c_e(s,t)$ = minimum cardinality of a set of edges such that there is no path from s to t in the graph obtained by removing it from G.

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

Theorem

$p_e(s,t) = c_e(s,t)$.

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

Proof. Let G_1 be the digraph obtained from G by replacing each edge by a pair of symmetric arcs. Let $c : E(G_1) \to \mathbb{N}$ a capacity function defined by $c(e) = 1$, $\forall e \in E(G_1)$.

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

Proof cont'd. Let x^0 be an integral flow of maximum value in $R = (G_1, s, t, c)$. If there exists a cycle C in G_1 with $x^0_{ij} = 1$ on all arcs of C, then we can put to 0 the flow on the arcs of C without changing the value of the flow x_0. Hence, we can suppose that the flow x^0 is acyclic and then x^0 can be expressed as a sum of $v(x^0)$ integral flows x^k with $v(x^k) = 1$.

Each flow x^k corresponds to a path from s to t in G_1 (by taking the arcs on which the flow is not 0), wich is a path from s to t also in the graph G.

It follows that $v(x^0) = p_e(s, t)$, since any set of edge-disjoint paths from s to t in G generates a $0 - 1$-flow in R of value equal to the number of these paths.

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

Proof cont'd. Let (S, T) be a minimum capacity cut in R; we have $c(S, T) = v(x^0)$, by the Max-flow Min-cut theorem. On the other hand, $c(S, T)$ is the number of arcs with an extremity in S and the other in T (since the arcs capacities are all 1). This set of arcs generates in G a set of edges of the same cardinality and such that there is no path from s to t in the graph obtained by removing it from G.

Hence we obtained a cut of capacity $c(S, T) = v(x^0) = p_e(s, t)$ edges in G which deconnects s from t by their removing from G. It follows that $c_e(s, t) \leqslant p_e(s, t)$. Since the inequality $c_e(s, t) \geqslant p_e(s, t)$ is obvious, the theorem is proved. □

If G is a connected graph, $\lambda(G)$, the maximum value of $p \in \mathbb{N}$ for which G is p-edge-connected, is

$$\min_{s,t \in V(G), s \neq t} c_e(s, t) \quad (*)$$

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

It follows that, to compute $\lambda(G)$, it is necessary to solve the $n(n-1)/2$ maximum flow problems described in the above proof. This number can be reduced if we observe that, for a fixed pair (s, t), if (S, T) is a minimum capacity cut, then

$$\forall v \in S \text{ and } \forall w \in T \; c_e(v, w) \leqslant c(S, T) \; (**)$$

In particular, if (s, t) is the pair for which the minimum in $(*)$ is attained, we have equality in $(**)$.
If we fix a vertex $s_0 \in V$ and solve the $n-1$ maximum flow problems by taking $t_0 \in V \setminus \{s_0\}$ we will obtain a pair (s_0, t_0) with $c(s_0, t_0) = \lambda(G)$ (t_0 will be not in the same class with s_0 in the bi-partition (S, T)).
Conclusion: $\lambda(G)$ can be found in $\mathcal{O}(n \cdot (nm + n^2c)) = \mathcal{O}(n^2m)$ time.

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

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

D. Finding the vertex-connectivity number of a graph.

Let $G = (V, E)$ be a graph. For $s, t \in V$, $s \neq t$, if we denote

- $p(s,t)$ = maximum number of internal vertex disjoint st-paths in G,
- $c(s,t)$ = minimum cardinality of a st-separating set of vertices in G,

then, by Menger Theorem, we have

$$p(s,t) = c(s,t) (***)$$

Moreover, the vertex-connectivity number, $k(G)$, of the graph G (the maximum value of $p \in \mathbb{N}$ for which G este p-connected) is

$$k(G) = \begin{cases} n-1, & \text{if } G = K_n \\ \min\limits_{s,t \in V(G), s \neq t, st \notin E(G)} c(s,t), & \text{if } G \neq K_n \end{cases} \quad (****)$$

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

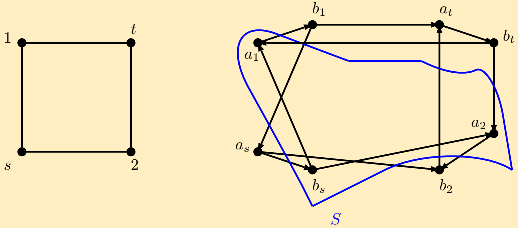
We show that the equality $(***)$ follows from the Max-flow Min-cut theorem, on an appropriate flow network.

Let $G_1 = (V(G_1), E(G_1))$ be the digraph constructed from G as follows:

- $\forall v \in V$, we put $a_v, b_v \in V(G_1)$ and $a_v b_v \in E(G_1)$;
- $\forall vw \in E$, we put $b_v a_w, b_w a_v \in E(G_1)$.

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

Example



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

Also, define $c : E(G_1) \to \mathbb{N}$ by

$$c(e) = \begin{cases} 1, & \text{if } e = a_v b_v \\ \infty, & \text{otherwise} \end{cases}$$

Let us consider the flow network $R = (G_1, b_s, a_t, c)$.

Let x^0 be an integral flow in R of maximum value. In the nodes $b_v (v \in V)$ enters exactly one arc of capacity 1 and from the nodes $a_v (v \in V)$ leaves exactly one arc of capacity 1.

It follows (by the flow equilibrium constraints) that $x^0_{ij} \in \{0, 1\}$, $\forall ij \in E(G_1)$. Therefore x^0 can be decomposed in $v(x^0)$ flows x^k, each of value 1, with the property that the arcs on which x_k is non null correspond to $v(x^0)$ internal disjoint paths in G.

On the other hand, from any set of p internal disjoint st-paths in G, we can construct p internal disjoint $b_s a_t$-paths in G_1, on which we can transport one unit of flow. It follows that $v(x^0) = p(s, t)$.

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

Let (S, T) be a minimum capacity cut in R such that $v(x^0) = c(S, T)$. Since $v(x^0) < \infty$, it follows that $\forall i \in S$, $\forall j \in T$ with $ij \in E(G_1)$; we have $c(ij) < \infty$, therefore $c(ij) = 1$, that is $\exists u \in V$ such that $i = a_u$ and $j = b_u$.

Hence, the cut (S, T) corresponds to a set of vertices $A_0 \subseteq V$ such that $c(S, T) = |A_0|$ and A_0 is a st-separating set.

On the other hand, $\forall A$ st-separating set, $|A| \geqslant p(s, t) = v(x^0)$. Therefore

$$c(s, t) = |A_0| = c(S, T) = v(x^0) = p(s, t).$$

The above proof, which, on one hand, completes the proof of the Menger's theorem from lecture 5, shows, on the other hand, that, in order to find $k(G)$ will be sufficiently to find the minimum in $(****)$ by solving $|E(\overline{G})|$ maximum flow problems, where $\overline{G}$ is the complement of G.

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

This gives an algorithm with time complexity

$$\mathcal{O}\left(\left(\frac{n(n-1)}{2} - m\right)(nm + n^2 \log n)\right).$$

A simple observation gives us a more efficient algorithm. Obviously,

$$k(G) \leqslant \min_{v \in V} d_G(v) = \frac{1}{n}\left(n \cdot \min_{v \in V} d_G(v)\right) \leqslant \frac{1}{n} \sum_{v \in V} d_G(v) = \frac{2m}{n}.$$

If A_0 is a cut-set in G with $|A_0| = k(G)$, then $G \setminus A_0$ is not connected and there exists a partition of $V \setminus A_0$ (V', V'') such that there is no edge in cross between V' and V'' and $\forall v' \in V'$, $\forall v'' \in V''$ we have $p(v', v'') = k(G)$.

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.

It follows that solving a maximum flow problem with $s_0 \in V'$ and $t_0 \in V''$ we obtain that $p(s_0, t_0) =$ maximum flow value $= k(G)$.

We can find such a pair like follows: let $l = \left\lceil \frac{2m}{n} \right\rceil + 1$, choose l vertices arbitrary from $V(G)$, and for each such vertex, v, solve all maximum flow problems $p(v, w)$, with $vw \notin E$. The number of such problems is $\mathcal{O}(nl) = \mathcal{O}\left(n\left(\left(\frac{2m}{n} + 1\right)\right)\right) = \mathcal{O}(m)$.

Hence the time complexity of finding $k(G)$ is $\mathcal{O}(m(nm + n^2 \log n))$.

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

Exercise 1. Show that, by using a maximum flow algorithm (in a certain network), you can find, in a 0 - 1 matrix, a maximum cardinality set of elements equals with 0, in which any two elements are not on the same row or column.

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

Exercise 2. Let S and T be two disjoint, finite, and non-empty sets. We have a function $a : S \cup T \to \mathbb{N}$. The requirement is to decide whether exists a bipartite graph $G = (S, T; E)$ such that $d_G(v) = a(v)$, for all $v \in S \cup T$; if the answer is affirmative you have to return the edges of G (S and T are the classes of the bipartition in G). Show that this problem can be polynomially solved as a maximum value flow problem in a certain network.

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. Every student from a cardinal $n > 0$ set S choose a subset of 4 optional courses from a cardinal $k > 4$ set C. Conceive an algorithm (with polynomial time complexity) which has determine (if exists) an allocation of the students to optional courses from C such that every student will be allocated to exactly 3 courses (from those 4 already chosen) and each course gathers at most $\lceil \alpha \cdot n/k \rceil$ students ($\alpha \geqslant 3$).

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

Exercise 4.

(a) True or false? In a network $R = (G, s, t, c)$ having distinct capacities there is an unique maximum value flow. Why?

(b) Devise and prove the corectness of a polynomial time complexity algorithm which has to decide that if in a given network there is an unique maximum flow.

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

Exercise 5. An IT company has n employees $P_1, P_2, \ldots, P_n$ which has to accomplish m projects $L_1, L_2, \ldots, L_m$. For any employee P_i we have a list $\mathcal{L}_i$ of projects onwhich he can work, and s_i the number of projects from $\mathcal{L}_i$ which can be accomplished by him in a week ($s_i \leqslant |\mathcal{L}_i|$). Any project will be assigned to one employee.
How can you find the minimum number of weeks required to finish all the projects by using flows in networks?

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

Exercise 6. The emergency evacuation plan of a buiding is described as a $n \times n$ grid; the cells borders of this grid are the escape routes to the outside of the building (grid). An instance of the evacuation problem contains the dimension n of the grid an m starting points (corners of the cells).

- 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 6 (cont'd). The instance has a positive answer if there are m disjoint paths towards grid frontier starting in the above m points. If such paths do not exist, then the instance has a negative answer.
Find a representation of this *evacuation problem* as a flow problem in a flow transportation network. Devise an efficient algorithm to recognize a positive instance of the evacuation problem (what is its time complexity?).

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

Exercise 7. Let $G = (V, E)$ be a graph having n vertices $\{v_1, v_2, \ldots, v_n\}$ and $c : E \to \mathbb{R}_+$ a capacity function on the edges of G. A cut in G is a bipartition (S, T) of V. The capacity of a cut (S, T) is $c(S, T) = \sum_{e \in E, |e \cap S| = 1} c(e)$.

- 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

Exercise 7 (cont'd). A minimum cut in G is a cut (S_0, T_0) such that

$$c(S_0, T_0) = \min_{(S,T) \text{ cut in } G} c(S, T)$$

(a) Show that we can determine a minimum cut in polynomial time complexity by solving a polynomial number of maximum flow problems in certain networks.

(b) For $G = C_n$ (the induced cycle of order $n \geqslant 3$) having all capacities 1, prove that there are $\dfrac{n(n-1)}{2}$ minimum cuts.

* 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

Exercise 8. Let $G = (V; E)$ be a digraph and $w : V \to \mathbb{R}$ such that $w(V) \cap \mathbb{R}_+, w(V) \cap \mathbb{R}_- \neq \varnothing$. A subset $A \subseteq V$ is called a *isolated* subset of G if there is no arc that leaves A. The *weight* of $A \subseteq V$ is $w(A) = \sum_{v \in A} w(v)$. Describe a polynomial time complexity using a maximum value flow algorithm in a certain network that has to find a maximum weight isolated subset of G.

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

Exercise 9. At the CS department there are p students ($\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$) who want to graduate and k professors ($\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$). For the final graduate examination (also called exit examination) teams of r professors will judge the students final projects. For a given project each professor either has the competences to judge it or not, i. e., we know the set, $\mathcal{P}_i \subsetneq \mathcal{P}$ ($\mathcal{P}_i \neq \varnothing$), of professors specialized on the project of student S_i. Each professor P_j can participate to at most n_j teams.

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 9 (cont'd) Each student must present this project to a team of $r(\leqslant k)$ professors, $a(\leqslant r)$ of them being specialized on this project and the remaining $(r-a)$ are not.

(a) Devise a network flow model to organize the judging teams (which professor will attend which project presentation).

(b) Give a characterization of the existence of a solution to this problem in terms of maximum flow in the above network.

(c) What is the time complexity for deciding if a solution exists?

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 *