

1

Fluxuri în rețele

- Prefluxuri

- Schema generală a unui algoritm de tip preflux

- Algoritmul lui Ahuja & Orlin

- Aplicații combinatoriale

- Cuplaje în grafuri bipartite

- Recunoașterea secvențelor digrafice

- Conexiunea pe muchii

- Conexiunea pe noduri

2

Exerciții pentru seminarul 10 (săptămâna 8-12 decembrie)

Definiție

Un preflux în $R = (G, s, t, c)$ este o funcție $x : E(G) \rightarrow \mathbb{R}$ astfel încât

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

e_i (pentru $i \in V \setminus \{s, t\}$) este numit **excesul** din nodul i . Dacă $i \in V \setminus \{s, t\}$ și $e_i > 0$, atunci i este un **nod activ**. Dacă $ij \in E$, x_{ij} va fi numit **fluxul de pe arcul ij** .

Dacă în R nu există noduri active, atunci prefluxul x este un flux cu $v(x) = e_t$.

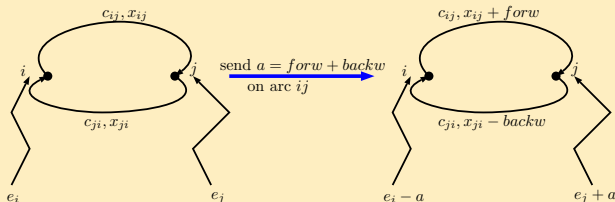
Ideea algoritmilor de tip preflux: un preflux inițial în R este transformat prin modificarea fluxurilor de pe arce într-un flux cu proprietatea că nu există drumuri de creștere în R relativ la el.

G este reprezentat folosind liste de adiacență. Vom presupune că dacă $ij \in E$, atunci $ji \in E$ de asemeni (altfel, adăugăm arcul ji de capacitate 0). Astfel, G devine digraf simetric.

Dacă x este un preflux în R și $ij \in E$, atunci **capacitatea reziduală** a lui ij este

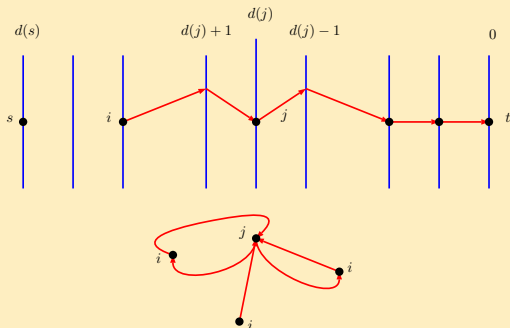
$$r_{ij} = c_{ij} - x_{ij} + x_{ji},$$

(reprezentând fluxul suplimentar care poate fi trimis de la nodul i la nodul j folosind arcele ij și ji).



Remarci

- Ca de obicei, notăm cu $A(i)$ lista de adiacență a nodului i .



Definiție

Fie x un prefix în R și d o funcție distanță relativ la x . Un arc $ij \in E$ este numit **admisibil** dacă $r_{ij} > 0$ și $d(i) = d(j) + 1$.

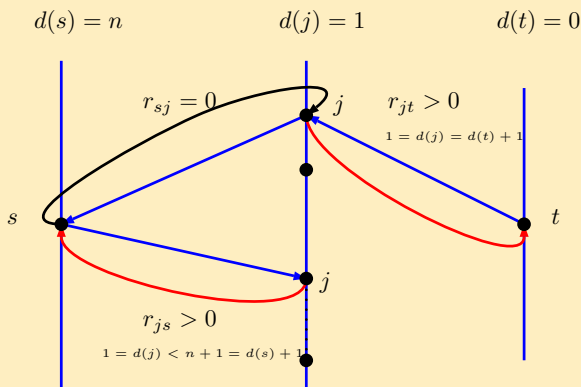
Dacă R este o rețea, considerăm următoarea procedură de inițializare, care construiește în $\mathcal{O}(m)$ un preflux x și o funcție distanță d relativ la x .

```
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  
   $d[i] \leftarrow 1$ ;
```

Se observă că după execuția acestei proceduri, avem $r_{sj} = 0, \forall sj \in A(s)$.

Astfel condiția (D_2) nu este afectată de $d(s) = n$.

Pentru toate arcele ij , (D_2) este îndeplinită:



Alegerea $d(s) = n$ are semnificația: nu există A -drumuri de la s la t relativ la x (altfel, dacă P este un astfel de drum, lungimea lui trebuie să fie cel puțin $d(s) = n$, ceea ce este imposibil).

Dacă acesta va fi un invariant al algoritmilor de tip preflux, atunci când x va deveni un flux, va fi un flux maxim.

Considerăm următoarele două proceduri:

```
procedure push( $i$ ); //  $i$  este un nod diferit de  $s, t$ 
```

```
  alege un arc admisibil  $ij \in A(i)$ ;
```

```
  "trimite"  $\delta = \min \{e_i, r_{ij}\}$  (unități de flux) de la  $i$  la  $j$ ;
```

Dacă $\delta = r_{ij}$ atunci avem o **pompare (push) saturată**, altfel avem o **pompare nesaturată**.

```
procedure relabel( $i$ ); //  $i$  este un nod diferit de  $s, t$ 
```

```
   $d[i] \leftarrow \min \{d[j] + 1 : ij \in A(i) \text{ și } r_{ij} > 0\}$ ;
```

Prefluxuri - Schema generală a unui algoritm de tip preflux

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 noduri active în R) do

 alege un nod activ i ;

 if (\exists arce admisibile în $A(i)$) then

push(i);

 else

relabel(i);

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

Lema 1

" d este funcție distanță relativ la prefluxul x " este un invariant al algoritmului de mai sus. La fiecare apel *relabel*(i), $d(i)$ crește.

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

Demonstrație. Am demonstrat deja că procedura de inițializare construiește un preflux x și o funcție distanță d relativ la x . Arătăm că dacă perechea (d, x) satisface (D_1) și (D_2) înaintea unei iterații **while**, atunci după această iterație cele două condiții sunt de asemeni satisfăcute.

Avem două cazuri, în funcție de care dintre procedurile **push** sau **relabel** este executată în iterația **while** curentă:

este apelată $push(i)$: singura pereche care poate viola (D_2) este $d(i)$ și $d(j)$. Deoarece ij este admisibil, la apelul $push(i)$ avem $d(i) = d(j) + 1$. După execuția $push(i)$, arcul ji poate avea $r_{ji} > 0$ (fără a fi astfel înaintea apelului), dar condiția $d(j) \leq d(i) + 1$ este evident satisfăcută.

este apelată $relabel(i)$: actualizarea lui $d(i)$ este astfel încât (D_2) este satisfăcută pentru fiecare arc ij cu $r_{ij} > 0$. Deoarece $relabel(i)$ este apelată când $d(i) < d(j) + 1$, $\forall ij$ cu $r_{ij} > 0$, urmează că, după apel, $d(i)$ crește (cu cel puțin 1). \square

Pentru a arăta că algoritmul se oprește, este necesar să arătăm că (în timpul execuției) dacă un nod i este activ atunci în lista lui de adiacență, $A(i)$, există cel puțin un arc ij cu $r_{ij} > 0$. Aceasta demonstrează lema următoare.

Lema 2

Dacă i_0 este un x -nod activ în R , atunci există un $i_0 s$ A -drum relativ la x .

Demonstrație. Dacă x este un preflux în R , atunci x poate fi descompus în $x = x^1 + x^2 + \dots + x^p$, unde fiecare x^k are proprietatea că mulțimea $A^k = \{ij : ij \in E, x_{ij}^k \neq 0\}$ este

- mulțimea arcelor unui drum de la s la t , sau
- mulțimea de arce ale unui drum de la s la un nod activ, sau
- mulțimea arcelor unui circuit.

Prefluxuri - Schema generală a unui algoritm de tip preflux

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 -

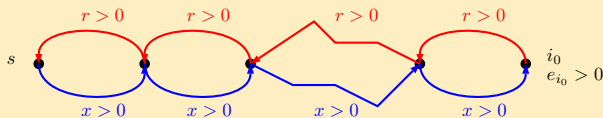
Mai mult, în cazurile (a) și (c), x^k este un flux. (Demonstrația este algoritmică: construim mai întâi mulțimile de tipul (a), apoi pe cele de tipul (c) și (b); la fiecare pas, căutăm inversul unui drum de tip (a) sau (c) (sau (b)); prefluxul obținut este scăzut din cel curent; deoarece excesele nodurilor sunt nenegative, construcția poate fi realizată, ori de câte ori prefluxul curent este nenul; construcția este finită, deoarece numărul de arce pe care fluxul curent este 0, crește la fiecare pas.)

Deoarece i_0 este un nod activ în R relativ la x , cazul (b) va apărea pentru nodul i_0 (cazurile (a) și (c) nu afectează excesul din nodul i_0). Arcele inverse de pe acest drum au capacitatea reziduală pozitivă (vezi imaginea de mai jos), astfel ele formează A -drumul cerut. \square

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 *

Prefluxuri - Schema generală a unui algoritm de tip preflux

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



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

Corolarul 1

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

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

Demonstrație. Într-adevăr, dacă i nu a fost reetichetat, atunci $d(i) = 1 < 2n$. Altfel, înaintea apelului $relabel(i)$, i este un nod activ, astfel din Lema 2, există un is -drum P cu $length(P) \leq n - 1$. Din (D_2) , urmează că, după reetichetare, $d(i) \leq d(s) + n - 1 = 2n - 1$ ($d(s) = n$ nu este modificat vreodată). \square

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

Corolarul 2

Numărul total de apeluri ale procedurii relabel nu este mai mare decât $2n^2$.

Demonstrație. Într-adevăr, există $n - 2$ noduri care pot fi reetichetate. Fiecare dintre ele poate fi reetichetat de cel mult $2n - 1$ ori (din Lema 1, Corolarul de mai sus și distanța inițială d). \square

Corolarul 3

Numărul total de pompări saturate nu este mai mare decât nm .

Demonstrație. Într-adevăr, când un arc ij devine saturat, avem $d(i) = d(j) + 1$. După aceea, algoritmul nu va mai trimite flux pe acest arc până când nu va trimite flux pe arcul ji , când vom avea $d'(j) = d'(i) + 1 \geq d(i) + 1 = d(j) + 2$.

Prefluxuri - Schema generală a unui algoritm de tip preflux

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

Demonstrație (continuare). Astfel o modificare a acestui flux pe arcul ij nu va apărea până când $d(j)$ nu va crește cu 2. Urmează că un arc nu poate deveni saturat de mai mult de n ori și (deoarece numărul total de arce este m), nu există mai mult de nm pompări saturate. \square

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

Lema 3

(Goldberg și Tarjan, 1986). Numărul total de pompări nesaturate nu este mai mare decât $2n^2m$.

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

Demonstrație. Omisă.

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

Lema 4

Algoritmul de tip preflux returnează un flux de valoare maximă în R .

Să presupunem că $c_{ij} \in \mathbb{N}$ și $U = \max_{ij \in E} (1 + c_{ij})$. Fie $K = \lceil \log_2 U \rceil$.

Ideea algoritmului: Avem $K + 1$ etape. În fiecare etapă p , cu p luând succesiv valorile $K, K - 1, \dots, 1, 0$, următoarele două condiții sunt îndeplinite:

- (a) la începutul etapei p , $e_i \leq 2^p, \forall i \in V \setminus \{s, t\}$.
- (b) în timpul etapei p , procedurile push-relabel sunt folosite pentru a elimina nodurile active, i , cu $e_i \in \{2^{p-1} + 1, \dots, 2^p\}$.

Din definiția lui K , în prima etapă ($p = K$) condiția (a) este îndeplinită (datorită modului de inițializare a prefluxului în fiecare nod $i \neq s, t$ avem $e_i = c_{si}$ sau $e_i = 0$, deci $e_i \leq U$), iar dacă proprietatea (b) va fi menținută, păstrând integralitatea exceselor de-a lungul algoritmului, atunci urmează că după ultima etapă **excesul fiecărui nod $i \in V \setminus \{s, t\}$ este 0**, deci avem un flux de valoare maximă.

Pentru a menține (b) de-a lungul execuției algoritmului, schema generală a unui algoritm de tip preflux este adaptată după cum urmează:

- fiecare etapă p începe prin construirea listei $L(p)$ a tuturor nodurilor $i_1, i_2, \dots, i_{l(p)}$ cu excese $e_i > 2^{p-1}$, ordonate crescător după d (aceasta poate fi făcută cu hash-sorting în timpul $O(n)$, deoarece $d(i) \in \{1, 2, \dots, 2n - 1\}$).
- nodul activ selectat pentru **push-relabel** în timpul etapei p va fi **primul nod din $L(p)$** . Urmează că, dacă se face o pompă (push) pe un arc admisibil ij , atunci $e_i > 2^{p-1}$ și $e_j \leq 2^{p-1}$ (deoarece $d(j) = d(i) - 1$ și i este primul nod din $L(p)$). Dacă δ , fluxul trimis de la i la j de către apelul $push(i)$, este limitat la $\delta = \min(e_i, r_{ij}, 2^p - e_j)$, atunci (deoarece $2^p - e_j \geq 2^{p-1}$) urmează că o **pompă nesaturată** trimite cel puțin 2^{p-1} unități de flux.

După apelul $push(i)$ excesul din nodul j (singurul pentru care excesul poate crește) va fi $e_j + \min(e_i, r_{ij}, 2^p - e_j) \leq e_j + 2^p - e_j \leq 2^p$, deci condiția (b) rămâne îndeplinită.

- etapa p este încheiată când lista $L(p)$ devine vidă.

Pentru a determina în mod eficient un arc admisibil pentru o pompare ($push$), sau pentru a inspecta toate arcele care părăsesc un nod i în vederea reetichetării ($relabel$), vom organiza listele de adiacență $A(i)$ după cum urmează:

- fiecare element din listă conține: nodul j , x_{ij} , r_{ij} , un pointer către elementul corespunzător lui i din lista de adiacență $A(j)$ și un pointer către următorul element din lista $A(i)$.
- lista are asociat un iterator pentru a-i înlesni parcurgerea.

Toate aceste liste sunt construite în $O(m)$, înaintea apelului procedurii $initialization$.

```
initialization;  $K \leftarrow \lceil \log_2 U \rceil$ ;  $\Delta \leftarrow 2^{K+1}$ ;  
for ( $p = \overline{K, 0}$ ) do  
  construiește  $L(p)$ ;  $\Delta \leftarrow \Delta/2$ ;  
  while ( $L(p) \neq \emptyset$ ) do  
    fie  $i$  primul nod din  $L(p)$ ;  
    caută în  $A(i)$  primul arc admisibil sau până se ajunge la sfârșit;  
    if ( $ij$  este arcul admisibil găsit) then  
       $\delta \leftarrow \min(e_i, r_{ij}, \Delta - e_j)$ ;  
       $e_i \leftarrow e_i - \delta$ ;  $e_j \leftarrow e_j + \delta$ ;  
      "trimite"  $\delta$  unități de flux de la  $i$  la  $j$ ;  
      if ( $e_i \leq \Delta/2$ ) then  
        șterge  $i$  din  $L(p)$ ;  
      if ( $e_j > \Delta/2$ ) then  
        adaugă  $j$  la începutul listei  $L(p)$ ;  
    else  
      calculează  $d[i] = \min\{d[j] + 1 : ij \in A(i) \text{ și } r_{ij} > 0\}$   
      re poziționează  $i$  în  $L(p)$ ;  
      setează poziția curentă (a pointerului) la începutul listei  $A(i)$ ;
```

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

Complexitatea timp a algoritmului este dominată de **pompările nesaturate** (ceea ce rămâne este de complexitate $\mathcal{O}(nm)$).

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

Lema 5

Numărul pompărilor nesaturate este cel mult $8n^2$ în fiecare etapă a scalării, astfel numărul total este $\mathcal{O}(n^2 \log U)$.

Demonstrație. Fie

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

La începutul etapei p , $F(p) < \sum_{i \in V} \frac{2^p \cdot (2n)}{2^p} = 2n^2$.

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

Dacă, în etapa p , se execută *relabel*(i), atunci nu există arce admisibile ij , și $d(i)$ crește cu $h \geq 1$ unități. $F(p)$ va crește cu cel mult h . Deoarece, $\forall i, d(i) < 2n$ urmează că $F(p)$ va crește (până la finalul etapei p) cel mult până la $4n^2$.

Dacă, în etapa p , se execută *push*(i), atunci aceasta trimite $\delta \geq 2^{p-1}$ pe arcul admisibil ij cu $r_{ij} > 0$ și $d(i) = d(j) + 1$. Astfel, după *push*,

$F(p)$ va avea valoarea $F'(p) = F(p) - \frac{\delta \cdot d(i)}{2^p} + \frac{\delta \cdot d(j)}{2^p} = F(p) - \frac{\delta}{2^p} \leq$

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

Această descreștere nu poate apărea de mai mult de $8n^2$ ori (deoarece $F(p)$ poate crește cel mult până la $4n^2$ și $F(p)$ este nenegativă). Evident, numărul pompărilor nesaturate este dominat de acest număr de descreșteri ale lui $F(p)$.

A. Determinarea unui cuplaj de cardinal maxim și a unei mulțimi stabile de cardinal maxim într-un graf bipartit.

Fie $G = (V_1, V_2; E)$ un graf bipartit cu n noduri și m muchii.

Considerăm rețeaua $R = (G_1, s, t, c)$, unde

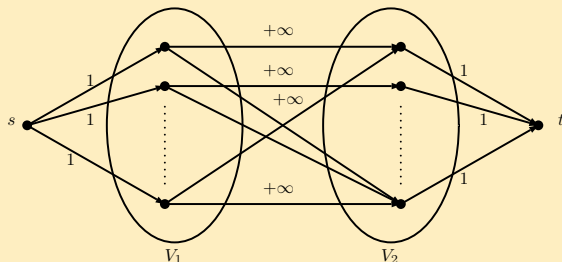
- $V(G_1) = \{s, t\} \cup V_1 \cup V_2$;
- $E(G_1) = E_1 \cup E_2 \cup E_3$, cu

$$E_1 = \{sv_1 : v_1 \in V_1\}, E_2 = \{v_2t : v_2 \in V_2\},$$

$$E_3 = \{v_1v_2 : v_1 \in V_1, v_2 \in V_2, v_1v_2 \in E(G)\},$$

- $c : E(G_1) \rightarrow \mathbb{N}$ definită prin

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



Dacă $x = (x_{ij})$ este un flux întreg în R , atunci mulțimea $\{ij : i \in V_1, j \in V_2 \text{ și } x_{ij} = 1\}$ corespunde unui cuplaj M^x în graful bipartit G , cu $|M^x| = v(x)$.

Reciproc, orice cuplaj $M \in \mathcal{M}_G$ corespunde unei mulțimi de arce neadiacente din G_1 ; dacă pe fiecare astfel de arc ij ($i \in V_1, j \in V_2$) considerăm $x_{ij}^M = 1$ și $x_{si}^M = x_{jt}^M = 1$, și adăugăm $x^M(e) = 0$ pe restul arcelor, atunci fluxul întreg x^M satisface $v(x^M) = |M|$.

Astfel, dacă rezolvăm problema fluxului maxim în R (începând cu fluxul nul), atunci obținem în $\mathcal{O}(nm + n^2 \log n)$ un cuplaj de cardinal maxim în G . (Constanta care înlocuiește în practică $+\infty$ trebuie să fie un întreg mai mare decât cardinalul oricărei tăieturi în graful suport, de exemplu poate fi $n^2 + 1$ - vezi mai jos.)

Fie (S, T) secțiunea de capacitate minimă (obținută în $\mathcal{O}(m)$ dintr-un flux maxim determinat). Din Teorema de flux maxim- secțiune minimă, $c(S, T) = \nu(G)$.

Deoarece $\nu(G) < \infty$, luând $S_i = S \cap V_i$ și $T_i = T \cap V_i$ ($i = \overline{1, 2}$), avem $|T_1| + |S_2| = \nu(G)$ și $X = S_1 \cup T_2$ este o mulțime stabilă în G (pentru a avea $c(S, T) < \infty$). Mai mult, $|X| = |V_1 \setminus T_1| + |V_2 \setminus S_2| = n - \nu(G)$. Urmează că X este o mulțime stabilă de cardinal maxim, deoarece $n - \nu(G) = \alpha(G)$ (din teorema lui König).

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. Recunoașterea secvențelor digrafice

Considerăm următoarea problemă:

Date $(d_i^+)_{i=\overline{1,n}}$ și $(d_i^-)_{i=\overline{1,n}}$, există un digraf $G = (\{1, \dots, n\}, E)$ astfel încât $d_G^+(i) = d_i^+$ și $d_G^-(i) = d_i^-$, $\forall i = \overline{1, n}$?

Condiții evident necesare pentru un răspuns afirmativ sunt:

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

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

În aceste ipoteze considerăm rețeaua bipartită $R = (G_1, s, t, c)$.

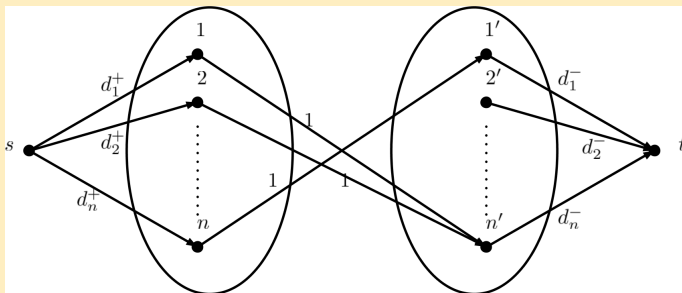
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Aplicații combinatoriale - Recunoașterea secvențelor digrafice

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

G_1 este obținut din graful complet bipartit $K_{n,n}$ cu bipartiția $(\{1, 2, \dots, n\}, \{1', 2', \dots, n'\})$, prin ștergerea mulțimii de muchii $\{11', 22', \dots, nn'\}$ și prin orientarea fiecărei muchii ij' ($\forall i \neq j \in \{1, 2, \dots, n\}$) de la i la j' , și prin adăugarea a două noi noduri s, t , și a tuturor arcelor $si, i \in \{1, 2, \dots, n\}$ și $j't, j \in \{1, 2, \dots, n\}$.

Funcția de capacitate: $c(si) = d_i^+, c(j't) = d_j^-, c(ij') = 1, \forall i, j = \overline{1, n}$.



C. Determinarea numărului de conexiune pe muchii a unui graf

Fie $G = (V, E)$ un graf. Pentru $s, t \in V, s \neq t$, notăm

- $p_e(s, t)$ = numărul maxim de drumuri disjuncte pe muchii de la s la t în G ,
- $c_e(s, t)$ = cardinalul minim al unei mulțimi de muchii astfel încât nu există drum de la s la t în graful obținut prin ștergerea ei din G .

Teoremă

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

Demonstrație. Fie G_1 digraful obținut din G prin înlocuirea fiecărei muchii printr-o pereche de arce simetrice. Fie $c : E(G_1) \rightarrow \mathbb{N}$ o funcție de capacitate definită prin $c(e) = 1, \forall e \in E(G_1)$.

Demonstrație (continuare). Fie x^0 un flux întreg de valoare maximă în $R = (G_1, s, t, c)$. Dacă există un circuit C în G_1 cu $x_{ij}^0 = 1$ pe toate arcele lui C , atunci putem pune fluxul 0 pe toate arcele lui C fără a modifica valoarea fluxului x_0 . Astfel, putem presupune că fluxul x^0 este aciclic și atunci x^0 poate fi scris ca o sumă de $v(x^0)$ fluxuri întregi x^k fiecare cu $v(x^k) = 1$.

Fiecare flux x^k corespunde unui drum de la s la t în G_1 (considerând arcele pe care fluxul nu este 0), care este un drum de la s la t și în graful G .

Urmează că $v(x^0) = p_e(s, t)$, deoarece orice mulțime de drumuri disjuncte pe muchii de la s la t în G generează un flux 0 – 1 în R de valoare egală cu numărul acestor drumuri.

Demonstrație (continuare). Fie (S, T) o secțiune de capacitate minimă în R ; avem $c(S, T) = v(x^0)$, din Teorema flux maxim-secțiune minimă. Pe de altă parte, $c(S, T)$ este numărul de arce cu o extremitate în S și cealaltă în T (deoarece toate capacitățile arcelor sunt 1). Această mulțime de arce generează în G o mulțime de muchii de același cardinal astfel încât nu există vreun drum de la s la t în graful obținut prin ștergerea ei din G .

Astfel am obținut o mulțime de $c(S, T) = v(x^0) = p_e(s, t)$ muchii în G care deconectează pe s de t prin ștergerea lor din G . Urmează că $c_e(s, t) \leq p_e(s, t)$. Deoarece inegalitatea $c_e(s, t) \geq p_e(s, t)$ este evidentă (vezi cursul 5), teorema este demonstrată. \square

Dacă G este un graf conex, $\lambda(G)$, valoarea maximă alui $p \in \mathbb{N}$ pentru care G este p -muchie-conex, este

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

Urmează că, pentru a calcula $\lambda(G)$, este necesar să rezolvăm $n(n-1)/2$ probleme de flux maxim descrise mai sus. Acest număr poate fi redus dacă observăm că, pentru o pereche fixată (s, t) , dacă (S, T) este o secțiune de capacitate minimă, atunci

$$\forall v \in S \text{ și } \forall w \in T \quad c_e(v, w) \leq c(S, T) (**)$$

În particular, dacă (s, t) este perechea pentru care minimul din (*) se atinge, avem egalitate în (**).

Dacă fixăm un nod $s_0 \in V$ și rezolvăm cele $n-1$ probleme de flux maxim luând $t \in V \setminus \{s_0\}$ vom obține o pereche (s_0, t_0) cu $c(s_0, t_0) = \lambda(G)$ (t_0 nu va fi în aceeași clasă a bipartiției cu s_0 în (S, T)).

Concluzie: $\lambda(G)$ poate fi găsit în $\mathcal{O}(n \cdot (nm + n^2 c)) = \mathcal{O}(n^2 m)$ (dacă G este conex, atunci $m \geq n-1$).

D. Determinarea numărului de conexiune pe noduri a unui graf.

Fie $G = (V, E)$ un graf. C $s, t \in V, s \neq t$, dacă notăm

- $p(s, t)$ = numărul maxim de drumuri intern disjuncte (pe noduri) de la s la t în G ,
- $c(s, t)$ = cardinalul minim al unei mulțimi st -separatoare de noduri din G ,

atunci, din teorema lui Menger, avem

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

În plus, numărul de conexiune pe noduri, $k(G)$, al grafului G (valoarea maximă a lui $p \in \mathbb{N}$ pentru care G este p -conex) este

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

Aplicații combinatoriale - Conexiunea pe noduri

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

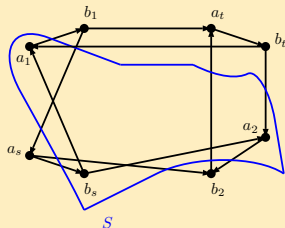
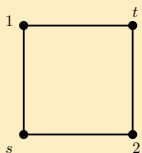
Arătăm că egalitatea (***) provine din Teorema flux maxim-secțiune minimă, aplicată unei rețele corespunzătoare.

Fie $G_1 = (V(G_1), E(G_1))$ digraful construit pornind de la G astfel:

- $\forall v \in V$, adăugăm $a_v, b_v \in V(G_1)$ și $a_v b_v \in E(G_1)$;
- $\forall vw \in E$, adăugăm $b_v a_w, b_w a_v \in E(G_1)$.

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

Exemplu



Mai definim $c : E(G_1) \rightarrow \mathbb{N}$ prin

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

Considerăm rețeaua $R = (G_1, b_s, a_t, c)$.

Fie x^0 un flux întreg în R de valoare maximă. În nodurile b_v ($v \in V$) intră exact câte un arc de capacitate 1 și din nodurile a_v ($v \in V$) pleacă exact câte un arc de capacitate 1.

Urmează că (din legea de conservare a fluxului) că $x_{ij}^0 \in \{0, 1\}$, $\forall ij \in E(G_1)$. Astfel x^0 poate fi descompus în $v(x^0)$ fluxuri x^k , fiecare de valoare 1, cu proprietatea că arcele pe care x_k este nenul corespund la $v(x^0)$ drumuri intern disjuncte din G .

Pe de altă parte, din orice mulțime de p st -drumuri intern disjuncte în G , putem construi p $b_s a_t$ -drumuri intern disjuncte în G_1 , pe care se poate transporta o singură unitate de flux. Urmează că $v(x^0) = p(s, t)$.

Fie (S, T) o secțiune de capacitate minimă în R astfel încât $v(x^0) = c(S, T)$. Deoarece $v(x^0) < \infty$, urmează că $\forall i \in S, \forall j \in T$ cu $ij \in E(G_1)$; avem $c(ij) < \infty$, deci $c(ij) = 1$, adică $\exists u \in V$ astfel încât $i = a_u$ și $j = b_u$.

Astfel, secțiunea (S, T) corespunde unei mulțimi de noduri $A_0 \subseteq V$ astfel încât $c(S, T) = |A_0|$ și A_0 este o mulțime st -separatoare.

Pe de altă parte, pentru orice mulțime st -separatoare, A , avem $|A| \geq p(s, t) = v(x^0)$. Deci

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

Demonstrația de mai sus (pe lângă faptul ca completează demonstrația teoremei lui Menger din cursul 5) arată că, pentru a determina $k(G)$ este suficient să determinăm minimul în (***) rezolvând $|E(\overline{G})|$ probleme de flux maxim, unde \overline{G} este complementul lui G .

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 -

Avem astfel un algoritm cu complexitatea timp

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

O observație simplă ne oferă un algoritm mai eficient. Evident,

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

Dacă A_0 este o mulțime separatoare în G cu $|A_0| = k(G)$, atunci $G \setminus A_0$ nu este conex și există o partiție a lui $V \setminus A_0$ (V' , V'') astfel încât nu există muchii de la V' la V'' și $\forall v' \in V'$, $\forall v'' \in V''$ avem $p(v', v'') = k(G)$.

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

Exerciții pentru seminarul 10

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 -

Exercițiul 1. Arătați că, utilizând un algoritm de flux maxim (într-o anumită rețea), se poate găsi, într-o matrice $0 - 1$, o mulțime de cardinal maxim de elemente egale cu 0, în care oricare două elemente nu se află pe aceeași linie sau pe aceeași coloană.

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

Exercițiul 2. Fie S și T mulțimi nevide, disjuncte și finite. Se dă o funcție $a : S \cup T \rightarrow \mathbb{N}$. Să se decidă dacă există un graf bipartit $G = (S, T; E)$ astfel încât $d_G(v) = a(v)$, pentru orice $v \in S \cup T$; dacă răspunsul este afirmativ trebuie returnate muchiile lui G (S și T sunt clasele bipartiției lui G). Arătați că această problemă poate fi rezolvată în timp polinomial ca o problemă de flux maxim într-o anumită rețea.

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 *

Exercițiul 3. Fiecare student dintr-o mulțime S de cardinal $n > 0$ subscrie la o submulțime de 4 cursuri opționale dintr-o mulțime C de cardinal $k > 4$. Descrieți un algoritm (de complexitate timp polinomială) care să determine (dacă există) o alocare a studenților către cursurile opțional din C astfel încât fiecare student să fie asignat la exact 3 cursuri (din cele 4 la care a scris) și fiecare curs să aibă asignați cel mult $\lceil \alpha \cdot n/k \rceil$ studenți ($\alpha \geq 3$).

Exercițiul 4.

- Adevărat sau fals? Într-o rețea $R = (G, s, t, c)$ cu capacități distincte există un singur flux maxim. De ce?
- Descrieți și demonstrați corectitudinea unui algoritm de complexitate (timp) polinomială care să decidă dacă într-o rețea dată există un singur flux maxim.

Exerciții pentru seminarul 10

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

Exercițiul 5. O companie IT are n angajați P_1, P_2, \dots, P_n care trebuie să lucreze la m proiecte L_1, L_2, \dots, L_m . Pentru orice angajat P_i avem o listă \mathcal{L}_i a proiectelor la care poate lucra și s_i numărul de proiecte din \mathcal{L}_i pe care le poate termina într-o săptămână ($s_i \leq |\mathcal{L}_i|$). Fiecare proiect va fi alocat unui singur angajat.

Cum se poate determina numărul minim de săptămâni necesare terminării tuturor proiectelor folosind fluxuri în rețele.

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

Exercițiul 6. Planul de evacuare de urgență al unei clădiri este descris ca un grid $n \times n$; frontierele celulelor sunt rutele de evacuare către exteriorul clădirii (grid-ului). O instanță a **problemei evacuării** conține dimensiunea n a grid-ului și m puncte de plecare (colțurile celulelor).

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

Exercițiul 6 (continuare). Instanța primește un răspuns afirmativ dacă există m drumuri disjuncte către frontiera grid-ului plecând din cele m puncte de mai sus. Dacă drumurile acestea nu există, atunci instanța primește un răspun negativ.

Determinați o reprezentare a acestei **problem a evacuării** ca o problemă de flux într-o rețea. Descrieți un algoritm eficient pentru a recunoaște instanțele pozitive ale problemei (care este complexitatea timp a algoritmului?).

Exercițiul 7. Fie $G = (V, E)$ un graf cu n noduri $\{v_1, v_2, \dots, v_n\}$ și $c : E \rightarrow \mathbb{R}_+$ o funcție de capacitate pe muchiile lui G . O **tăietură** în G este o bipartiție (S, T) a lui V . Capacitatea unei tăieturi (S, T) este $c(S, T) = \sum_{e \in E, |e \cap S|=1} c(e)$.

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 -

Exercițiul 7 (continuare). O tăietură minimă în G este o tăietură (S_0, T_0) astfel încât

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

- (a) Arătați că se poate determina o tăietură minimă în timp polinomial rezolvând un număr polinomial de probleme de flux maxim în anumite rețele.
- (b) Pentru $G = C_n$ (circuit indus de ordin $n \geq 3$) cu toate capacitățile 1, arătați că există $\frac{n(n-1)}{2}$ tăieturi minime.

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 *

Exercițiul 8. Fie $G = (V; E)$ un digraf și $w : V \rightarrow \mathbb{R}$ astfel ca $w(V) \cap \mathbb{R}_+$ și $w(V) \cap \mathbb{R}_- \neq \emptyset$. O submulțime $A \subseteq V$ se numește *izolată* în G dacă nu există nici un arc care iese din A . *Ponderea* unei submulțimi $A \subseteq V$ este $w(A) = \sum_{v \in A} w(v)$. Descrieți un algoritm de complexitate polinomială bazat pe un algoritm de flux maxim într-o anumită rețea care să determine o submulțime izolată de pondere maximă în G .

Exercițiul 9. La Departamentul de Informatică există p studenți ($S = \{S_1, S_2, \dots, S_p\}$) care doresc să aprobe cu o diplomă de licență și k profesori ($\mathcal{P} = \{P_1, P_2, \dots, P_k\}$). Lucrările de licență ale studenților sunt evaluate de echipe formate din câte r profesori.

Pentru subiectul unei lucrări de licență un profesor poate fi specializat sau nu; se cunoaște mulțimea, $\mathcal{P}_i \subsetneq \mathcal{P}$ ($\mathcal{P}_i \neq \emptyset$), a profesorilor competenți în a judeca lucrarea de licență a studentului S_i , pentru orice i . Fiecare profesor P_l poate participa în cel mult n_l astfel de echipe de

Exercițiul 9 (cont.) Fiecare student trebuie să-și prezinte lucrarea unei echipe de $r (\leq k)$ profesori, $a (\leq r)$ fiind specializați în proiectul respectiv, iar $(r - a)$ nu.

- (a) Descrieți un model cu o rețea de transport pentru a organiza ca mai sus echipele de evaluare formate din profesori (fiecare profesor trebuie asigurat unei mulțimi de lucrări de licență).
- (b) Caracterizați existența unei soluții pentru această problemă în termenii existenței unui anumit flux maxim în rețeaua de mai sus. (Caracterizarea trebuie demonstrată!)
- (c) Care este complexitatea timp necesară pentru a decide dacă există soluții?