Laborator 11 - Flux Maxim

Responsabili:

- Darius-Florentin Neațu (2017-2021) [mailto:neatudarius@gmail.com]
- Radu Nichita (2021) [mailto:radunichita99@gmail.com]

Obiective laborator

- formalizarea noțiunilor de rețea de transport și flux în rețea
- prezentarea unei metode de rezolvare a problemei de flux maxim
- analiza unei implementări a metodei oferite

Importanţă – aplicaţii practice

Un graf orientat poate fi utilizat pentru modelarea unui proces de transport într-o rețea între un producător *s* și un consumator *t*. Destinația nu poate consuma mai mult decât se produce, iar cantitatea trimisă pe o cale nu poate depăși capacitatea sa de transport.

Rețelele de transport pot modela curgerea lichidului în sisteme cu țevi, deplasarea pieselor pe benzi rulante, deplasarea curentului prin rețele electrice, transmiterea informațiilor prin rețele de comunicare etc.

Descrierea problemei şi a rezolvărilor

O problemă des întâlnită într-o rețea de transport este cea a găsirii fluxului maxim posibil prin arcele rețelei astfel încât:

1. să nu fie depășite capacitățile arcelor

2. fluxul să se conserve în drumul său de la *s* la *t*

Definiție 1

O rețea de transport este un graf orientat $G = (V,E)$ cu proprietățile:

1. există două noduri speciale în V: *s* este nodul sursă (sau producătorul) și *t* este nodul terminal (sau consumatorul).

2. este definită o funcție totală de capacitate $c: V \times V \rightarrow R+$ astfel încât:

- $c(u,v) = 0$ dacă $(u,v) \notin E$
- $c(u,v) \geq 0$ daca $(u,v) \in E$

3. pentru orice nod $v \in V \setminus \{s,t\}$ există cel puțin o cale $s \rightarrow v \rightarrow t$.

Definiție 2

Numim flux în rețeaua $G = (V,E)$ o funcție totală $f: V \times V \rightarrow R$ cu proprietățile:

1. Restricție de capacitate:

- $f(u,v) \leq c(u,v), \forall (u,v) \in V$ - fluxul printr-un arc nu poate depăși capacitatea acestuia

2. Antisimetrie:

- $f(u,v) = -f(v,u)$, $\forall u \in V, \forall v \in V$

3. Conservarea fluxului:

- $\Sigma f(u,v) = 0, \forall u \in V \setminus \{s,t\}, v \in V$

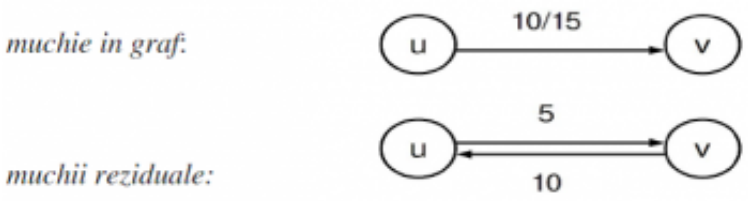
Un flux negativ de la u la v este unul virtual, el nu reprezintă un transport efectiv, ci doar sugerează că există un transport fizic de la v la u (este o convenție asemănătoare cu cea făcută pentru intensitățile curenților într-o rețea electrică). Ultima proprietate ne spune că la trecerea printr-un nod fluxul se conservă: suma fluxurilor ce intră într-un nod este 0 (ținând cont de convenția de semn stabilită).

Numim **capacitate reziduală** a unui arc

$$c_f(u,v) = c(u,v) - f(u,v)$$

și o interpretăm ca fiind cantitatea de flux adițional care poate fi transportat de la *u* la *v*, fără a depăși capacitatea $c(u,v)$.

Exemplu:



Daca avem arcul $(u,v) \in V$ cu $c(u,v) = 15$ și $f(u,v) = 10$, se pot transporta $c_f(u,v) = 5$ unități suplimentare fără a încălca restricția de capacitate. Dar, conform definiției, deși arcul $(v,u) \notin V$ vom avea totuși o capacitate reziduală $c_f(v,u) = c(v,u) - f(v,u) = 0 - (-10) = 10$: as putea transporta 10 unități în sens opus care să le anuleze pe cele 10 ale fluxului direct pe muchia (u,v).

Definiție 3

Fie o rețea de flux $G = (V,E)$, iar *f* fluxul prin G. Numim rețea reziduală a lui G, indusă de *f*, o rețea de flux notată cu $G_f = (V, E_f)$, astfel încât

- $E_f = \{(u,v) \in V \mid c_f(u,v) = c(u,v) - f(u,v) > 0\}$

Este important de observat că E_f și E pot fi disjuncte: un arc rezidual (u,v) apare în rețeaua reziduală doar dacă capacitatea sa este strict pozitivă (ceea ce nu implică existența arcului în rețeaua originală).

Un drum de ameliorare este o cale $(u1, u2, \ldots, uk)$, unde $u1 = s$ și $uk = t$, în graful rezidual cu $c_f(ui, ui+1) > 0, \forall i=1,2,\ldots,k-1$. Practic, un drum de ameliorare va reprezenta o cale în graf prin care se mai poate pompa flux adițional de la sursa la destinație.

Așa cum era de intuit, capacitatea reziduală a unui drum de ameliorare p este cantitatea maximă de flux ce se poate transporta de-a lungul lui:

- $c_f(p) = \min\{c_f(u,v) \mid (u,v) \in p\}$

Acum că am introdus noțiunile necesare pentru formalizarea problemei de flux maxim într-un graf, putem să prezentăm și cea mai utilizată metodă de rezolvare.

Algoritmul Ford-Fulkerson

Aceasta este o metodă iterativă de găsire a fluxului maxim într-un graf care pleacă de la ideea: cât timp mai există un drum de ameliorare (o cale de la sursă la destinație) pot pompa pe această cale un flux suplimentar egal cu capacitatea reziduală a căii.

Acest algoritm reprezintă mai mult un șablon de rezolvare pentru că nu detaliază modul în care se alege drumul de ameliorare din rețeaua reziduală.

```
Ford_Fulkerson
    Input G(V,E), s, t
    Output |fmax|
    |fmax| ← 0
    f{u,v) ← 0, ∀(u,v) ∈ V×V
    while ∃a path p(s ---> t) in Gf such that cf(u,v) > 0, ∀(u,v) ∈ p
        find cf(p) = min{cf(u,v) | cf(u,v) ∈ p}
        |fmax| += cf(p)
        for-each (u,v) ∈ p
            f(u,v) ← f(u,v) + cf(p)
            f(v,u) ← -f(u,v)
    return |fmax|
```

Complexitatea va fi $O(E * f\ max)$ pentru că în ciclul while putem găsi, în cel mai rău caz, doar cai care duc la creșterea fluxului cu doar o unitate la fiecare pas.

Ne punem acum problema dacă acest algoritm este corect, dacă la final vom avea cu adevărat fluxul maxim posibil prin graf. Corectitudinea algoritmului derivă imediat din teorema **Flux maxim - tăietura minimă**.

Numim o **tăietură a unui graf** o partiție (S,T) a nodurilor sale cu proprietatea $s \in S$ si $t \in T$.

Teorema flux maxim - tăietura minimă:

Pentru o rețea de flux $G(V,E)$ următoarele afirmații sunt echivalente:

1. f este fluxul maxim în G

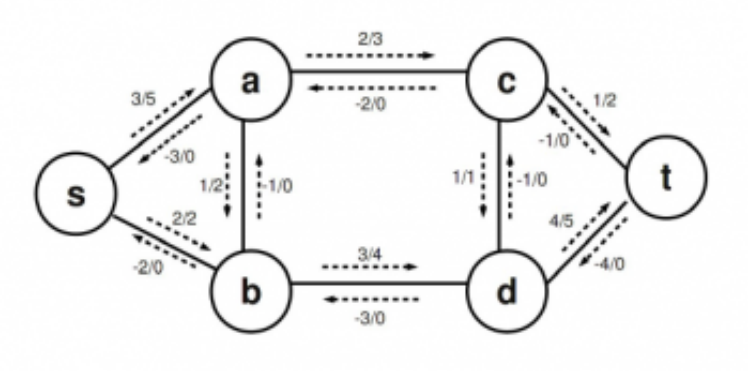
2. Rețeaua reziduală G_f nu conține drumuri de ameliorare

3. Există o tăietură (S,T) a lui G astfel încât fluxul net prin tăietură este egal cu capacitatea acelei tăieturi.

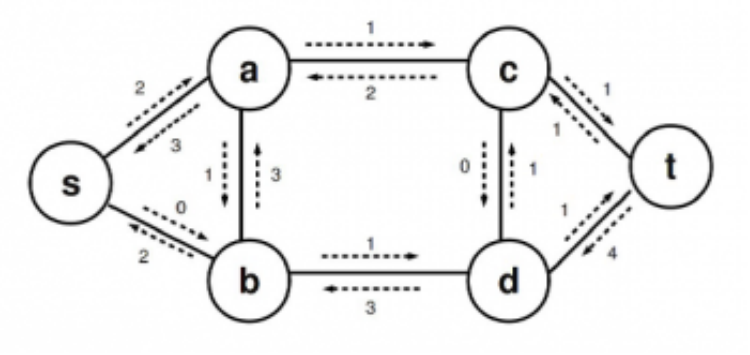
Obs: Prin orice tăietură fluxul este egal cu cel maxim pentru că nu există o altă cale pe care ar putea ajunge flux de la sursă la destinație și care să nu treacă prin tăietură (ar încălca tocmai definiția ei); sau, altfel spus, valoarea unui flux într-o rețea este dată de fluxul oricărei tăieturi. Astfel, fluxul total va fi mărginit de cea mai mică capacitate a unei tăieturi. Dacă este îndeplinit punctul 3. al teoremei atunci știm că acea tăietură nu poate fi decât una de capacitate minimă. Ultima incercare de a gasi o cale de la sursa la drena va rezulta in gasirea doar a elementelor marginite de o astfel de taietura.

Exemplu:

Rețeaua inițială:



Rețeaua reziduală:



Observăm că deși muchia `(d,c)` are capacitate 0 în rețeaua originală (ea nu ar putea transporta flux) în rețeaua reziduală avem `c(d,c) = 1` ceea ce îi permite să facă parte din drumul de ameliorare `p1 = (s, a, b, d, c, t)` de capacitate c_f`(p1) = 1`, astfel că adăugarea acestui flux suplimentar pe muchia `(d,c)` nu va duce la încălcarea restricției de capacitate; sau am fi putut alege drumul `p2=(s, a, c, t)` cu c_f`(p2) = 1` sau `p3 = (s,a,b,d,t)` cu c_f`(p3) = 1`. Performanța algoritmului ține de modul în care va fi ales drumul de ameliorare, se poate întâmpla ca pentru `|fmax|` de valoarea mare o alegere nepotrivită să ducă la timpi de execuție foarte mari.

Implementarea Edmonds-Karp

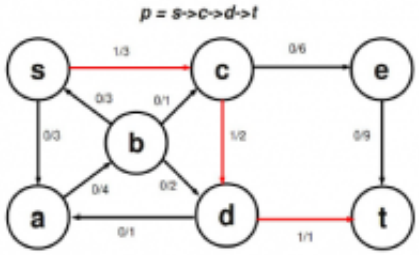
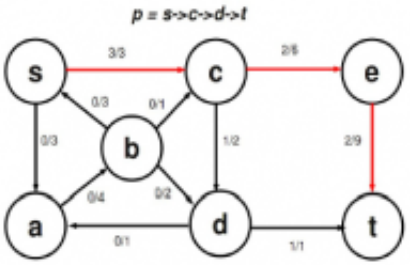
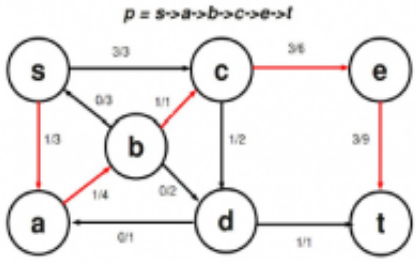
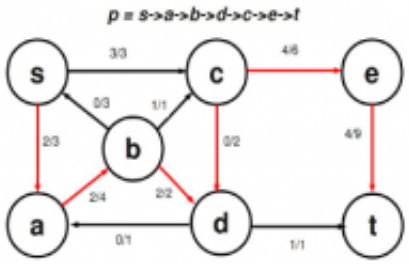
Așa cum am văzut, algoritmul Ford-Fulkerson nu definește o metodă de alegere a drumului de ameliorare pe baza căruia se modifică fluxul în graf. Implementarea Edmonds-Karp alege întotdeauna cea mai scurtă cale folosind o căutare în lățime în graful rezidual unde fiecare arc are ponderea 1. Se poate demonstra că lungimea căilor găsite astfel crește monoton cu fiecare nouă ameliorare.

```
Edmonds-Karp
    Input G(V,E), s, t
    Output |fmax|
    |fmax| ← 0
    f(u,v) ← 0, ∀(u,v) ∈ V×V
    while true
        p(s ⇝ t) ← BFS(Gf,s,t)
        if not ∃p(s ⇝ t)
            break;
        find cf(p) = min{cf(u,v) | cf(u,v) ∈ p}
        |fmax| += cf(p)
        for-each (u,v) ∈ p
            f(u,v) ← f(u,v) + cf(p)
            f(v,u) ← -f(u,v)
    return |fmax|
```

Plecând de la ideea că drumurile de ameliorare găsite au lungimi din ce în ce mai mari se poate arăta că în

această implementare fluxul se mărește de cel mult $O(V*E)$ ori ([1], pag.513).

Complexitatea algoritmului va fi $O(V*E^2)$. Să luăm un exemplu de rulare al acestui algoritm. Vom considera starea rețelei după ce a fost găsita prima cale de pompare flux(inițial toate arcele sunt etichetate cu 0/0 conform notației stabilite):

Capacitatea reziduala a caii de ameliorare	Graful si calea de ameliorare gasita
$c_f(p)=min(c_f(s,c),c_f(c,d),c_f(d,t))=$ $=min(3-0,2-0,1-0)=min(3,2,1)=1$	
$c_f(p)=min(c_f(s,c),c_f(c,e),c_f(e,t))=$ $=min(3-1,6-0,9-0)=min(2,6,9)=2$	
$c_f(p)=min(c_f(s,a),c_f(a,b),c_f(b,c),$ $c_f(c,e),c_f(e,t))=$ $=min(3-0,4-0,1-0,6-2,9-2)=$ $=min(3,4,1,4,7)=1$	
$c_f(p)=min(c_f(s,a),c_f(a,b),c_f(b,d),$ $c_f(d,c),c_f(c,e),c_f(e,t))=$ $=min(3-1,4-1,2-0,0-(-1),6-3,9-3)=$ $=min(2,3,2,1,3,6)=1$	

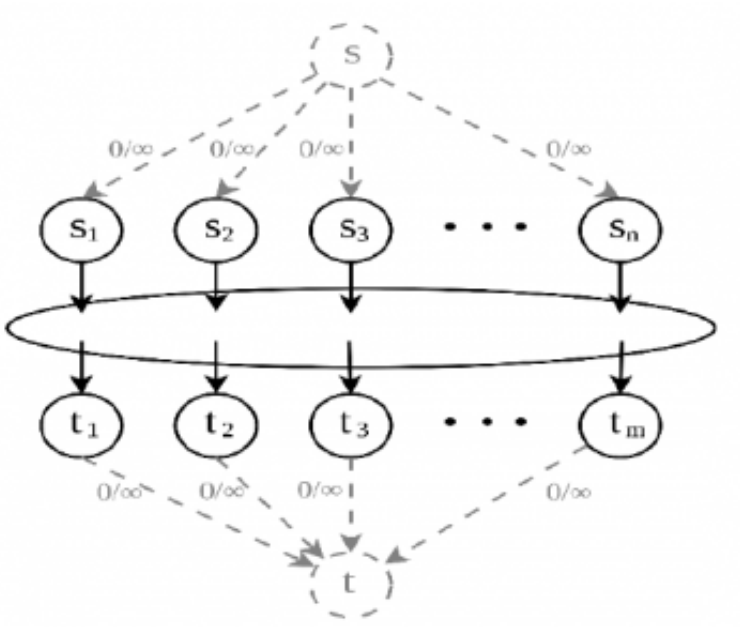
Obs: In cazul ultimului drum de ameliorare găsit in exemplul dat, se observa ca deși nu exista muchia

`(d,c)` se simulează un flux pe aceasta printr-unul negativ in sens opus.

Variații ale problemei clasice

In rețelele clasice studiate pana acum aveam o sursa unica care putea produce oricât, destinația unica consuma oricât, orice nod intermediar conserva fluxul, iar singura constrângere a muchiilor era limitarea superioara a fluxului prin capacitate. Sa vedem cum putem generaliza aceste condiții si daca rețelele obținute ar putea fi reduse la una clasica.

Surse si destinații multiple



Se observă ca putem reduce acest graf la cel cunoscut prin adăugarea unei meta-surse legată de sursele mici prin muchii de capacitate nelimitata si analog un meta-terminal cu aceleași proprietăți. Global comportamentul rețelei de flux nu se va schimba.

Cuplaj bipartit maxim

Fiind dat un graf neorientat `G = (V,E)`, un cuplaj este o submultime de muchii `M` inclusa in `E` astfel incat pentru toate varfurile `v ∈ V`, exista cel mult o muchie in `M` incidenta in `v`. Spunem ca un varf `v ∈ M` este cuplat de cuplajul `M` daca exista o muchie in `M` incidenta in `v`. Un cuplaj maxim este un cuplaj de cardinalitate maxima. In cazul grafurilor bipartite, multimea de varfuri poate fi partitionata in `V = L U R`, unde `L` si `R` sunt disjuncte si toate muchiile din `E` sunt intre `L` si `R`. Problema poate fi rezolvata cu ajutorul notiunii de flux, construind reteaua de transport `G' = (V',E')` pentru graful bipartit `G`. Vom alege sursa *s* si destinatia *t* ca fiind noi varfuri care nu sunt in `V` si vom construi `V' = V U {s,t}`. Arcele orientate ale lui `G'` sunt date de:

- `E' = { (s, u) : u ∈ L } U { (u, v) : u ∈ L, v ∈ R si (u, v) ∈ E} U { (v, t) : v ∈ R}`

Pentru a completa constructia, vom atribui fiecarei muchii din `E'` capacitatea unitate.

Rețea cu noduri ce nu conservă fluxul

Spre deosebire de *s* si *t* care produc/consumă oricât, un nod intermediar ar putea produce sau consuma o cantitate constanta de flux la trecerea prin el. In acest caz vom avea un invariant la nivel de nod care ia forma:

- $f_{in} - f_{out} = di$, unde di este cantitatea produsă suplimentar (> 0) sau solicitată (< 0) de un nod.

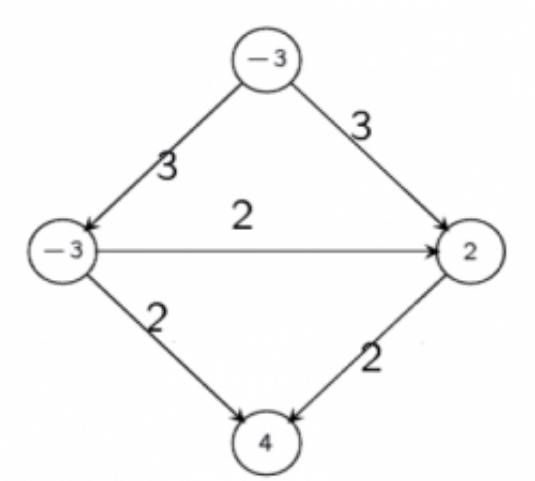
Am putea transforma egalitatea in:

- $(f_{in} + |di|) - f_{out} = 0$, dacă $di < 0$
- $f_{in} - (f_{out} + |di|) = 0$, daca $di > 0$

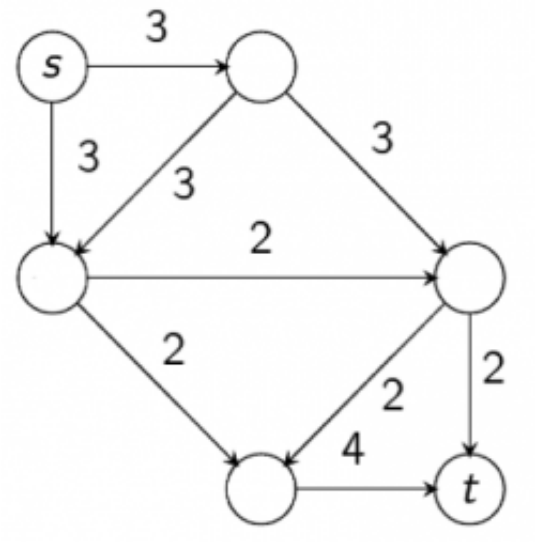
Altfel spus, un nod ce consumă flux poate fi transformat intr-unul ce conservă fluxul și are un in-arc adițional de capacitate $|di|$, iar unul ce produce flux va avea un out-arc de aceeași capacitate.

La nivelul unei rețele întregi se adaugă un nod sursă cu muchii către toate nodurile ce consumau flux si un nod destinație dinspre toate nodurile ce produceau flux.

Exemplu:



se va transforma in:

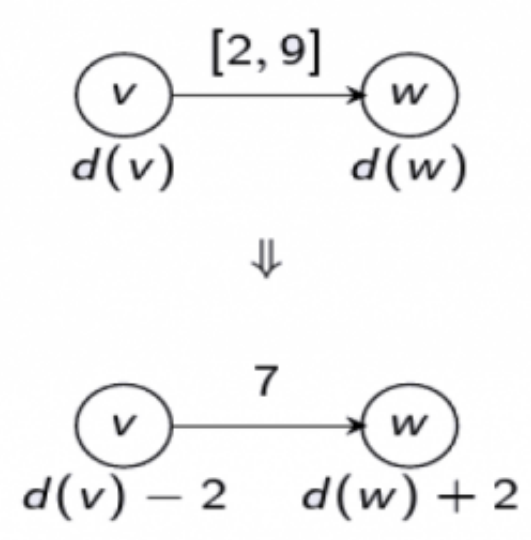


Rețea cu limite inferioare de capacitate

Există cazuri în care aș vrea ca datele de pe muchiile rețelei sa facă parte dintr-un anumit interval `[inf, sup]`. Pe o astfel de muchie fluxul trebuie să respecte inegalitatea `inf ≤ f ≤ sup`

Plecând tot de la condițiile de conservare la nivel de nod putem translata intervalul `[inf, sup]` în `[0, sup-inf]` și să considerăm că nodul sursă a consumat `inf` unități de flux iar nodul destinație a produs `inf` unități. Din exterior entitatea alcătuită din doua noduri și o muchie este văzută ca acționând în același fel asupra fluxului ce o traversează.

Iată un exemplu de transformare:



Bineînțeles că toate aceste transformări pot fi combinate pentru a se ajunge la rețeaua de flux clasică.

Concluzii și observații

Laboratorul de față s-a vrut a fi doar o introducere în domeniul fluxurilor într-un graf – modalitate de a reprezenta probleme de circulație a materialelor atât de frecvent întâlnite. In [1] și [2] găsiți și alți algoritmi interesanți împreună cu studiul complexității lor (algoritmul de pompare preflux, algoritmul „mutare-in-fata"). Spre exemplu, cel mai bun algoritm în prezent pentru cuplajul bipartit maxim se executa in $O(\sqrt{V} * E)$.

Exercitii

Scheletul de laborator se găsește pe pagina pa-lab::skel/lab11 [https://github.com/acs-pa/pa-lab/tree/main/skel/lab11].

Inainte de a rezolva exercitiile, asigurati-va ca ati citit si inteles toate precizarile din sectiunea Precizari laboratoare 07-12 [https://ocw.cs.pub.ro/courses/pa/skel_graph].

Prin citirea acestor precizari va asigurati ca:

- cunoasteti **conventiile** folosite
- evitati **buguri**
- evitati **depunctari** la lab/teme/test

Flux maxim

Se da un graf **orientat** cu **n** noduri si **m** arce. Graful are pe arce **capacitati pozitive**.

Folositi **Edmonds-Karp** pentru a gasi **fluxul maxim** intre nodul sursa **1** si nodul destinatie **n** in graful dat.

Restrictii si precizari:

- $n <= 1.000$
- $m <= 5.000$
- $1 <= c <= 110.000$, unde c este capacitatea unui arc
- Nu exista arc de la **n** la **1**.
- timp de executie
    - C++: 1s
    - Java: 2s

Rezultatul se va returna sub forma unui singur numar **F**, reprezentand fluxul maxim ce poate fi trimis prin retea.

BONUS

Rezolvati problema harta [https://infoarena.ro/problema/harta] pe infoarena.

Referinte

[1] Introducere in algoritmi, Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest – Capitolul VI Algoritmi pe grafuri: Flux maxim

[2] Introducere in analiza algoritmilor, Cristian A. Giumale – Cap. V Algoritmi pr grafuri: Fluxuri maxime intr-un graf

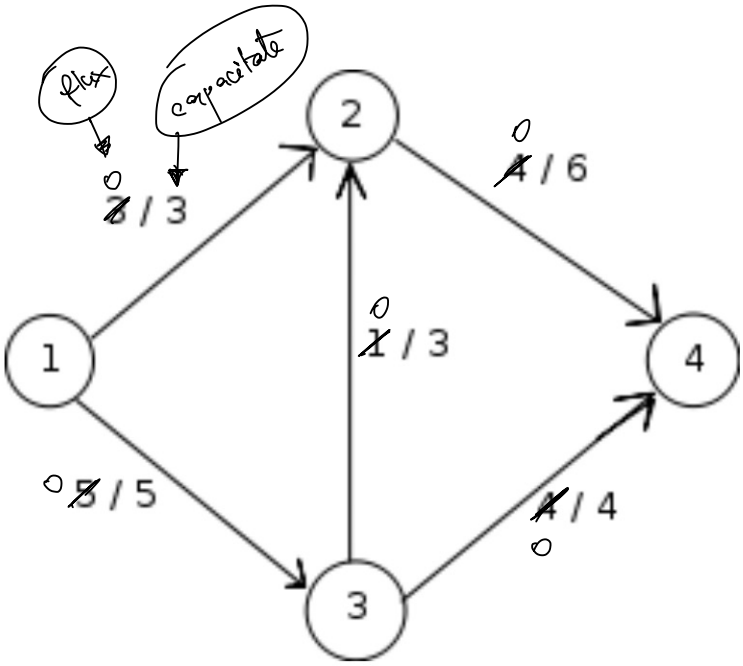
[3] Un articol de la MIT: A Labeling Algorithm for the maximum flow network problem [http://web.mit.edu/15.053/www/AMP-Appendix-C.pdf]

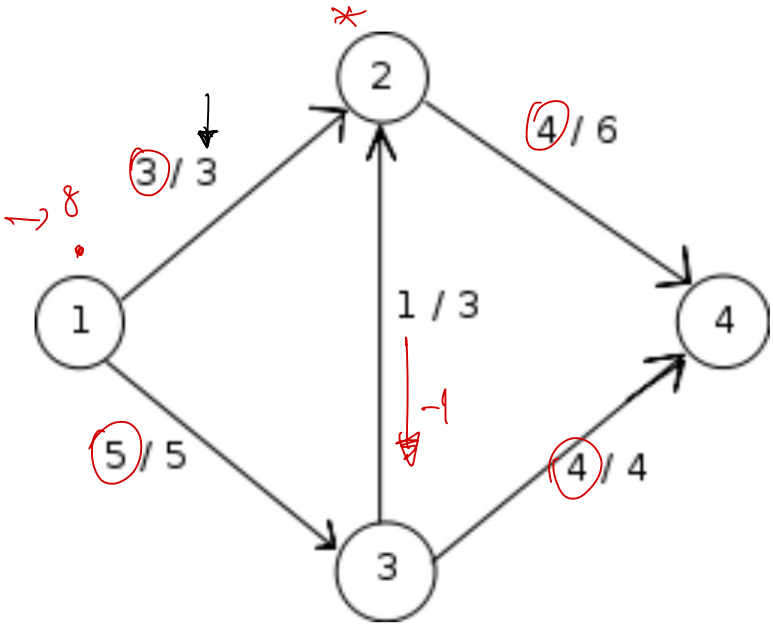
[4] Resurse wiki – Edmonds Karp [http://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm] Max Flow - Min Cut Theorem [http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem]

[5] Aplicatii flux maxim [http://www.cs.princeton.edu/~wayne/cs423/lectures/max-flow-applications-4up.pdf]

pa/laboratoare/laborator-11.txt · Last modified: 2021/05/06 02:09 by darius.neatu

Fluxul maxim





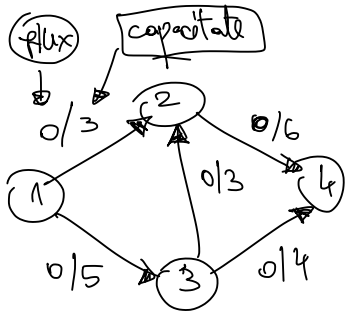
7

Alg. Edmonds-Karp

S = 1

Dest = n

flux / capacitate



fmax = ?

graf initial

n = 4 m = 5

x	y	capacitate
1	2	3
1	3	5
2	4	6
3	4	4
3	2	3

F(nxn)

	1	2	3	4
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

cap

	1	2	3	4
1	0	3	5	0
2	0	0	0	6
3	0	3	0	4
4	0	0	0	0

Lista de vecini

1: 2 3

2: 1 4 3

3: 1 4 2

4: 2 3

(1) BFS

visited

1	2	3	4
1	0	0	0

parent

1	2	3	4
-1	-1	-1	-1

if (F[x][y] != C[x][y] && !visited[y])

q: [1]

x = 1 → y = 2

→ y = 3

visited

1	2	3	4
1	1	1	0

parent

1	2	3	4
-1	1	1	-1

q: [2] [3]

n

S ..... Dest

x = 2 → y = 1

→ y = 4

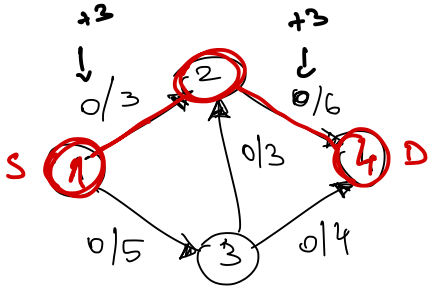
visited

1	2	3	4
1	1	1	1

parent

1	2	3	4
-1	1	1	2

q: [3] [4]



fmin = ∞

node = 4 ; fmin = min(∞, C[2][4] − F[2][4]) = 6 (6 − 0)

node = 2 ; fmin = min(6, C[1][2] − F[1][2]) = 3 (3 − 0)

node = 1

fmin = 3

FMAX = $\sum f_{min}$ = 3

F (nxn)

	1	2	3	4
1	0	3	0	0
2	-3	0	0	3
3	0	0	0	0
4	0	-3	0	0

cap

	1	2	3	4
1	0	3	5	0
2	0	0	0	6
3	0	3	0	4
4	0	0	0	0

if (F[x][y] != c[x][y] && !visited[y])

Lista de vecini

1: 2 3

2: 1 4 3

3: 1 4 2

4: 2 3

(2) BFS

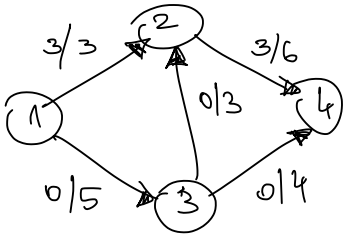
visited

1	2	3	4
1	0	0	0

parent

1	2	3	4
-1	-1	-1	-1

Q: 1



x=1 → y=2

→ y=3

visited

1	2	3	4
1	0	1	0

parent

1	2	3	4
-1	-1	1	-1

Q: 3

x=3 → y=1

→ y=4

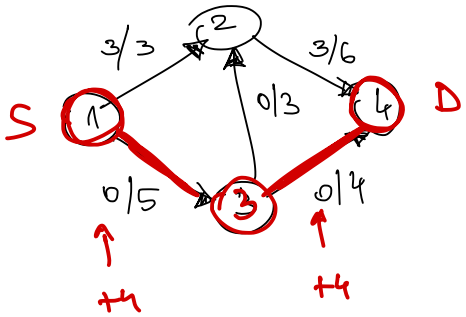
visited

1	2	3	4
1	0	1	1

parent

1	2	3	4
-1	-1	1	3

Q: 4



fmin = ∞

node = 4 ; fmin = min(∞, c[3][4] − F[3][4]) = 4 (4 − 0)

node = 3 ; fmin = min(4, c[1][3] − F[1][3]) = 4 (5 − 0)

node = 1

fmin = 4

F = 3 + 4 = 7

F (nxn)

	1	2	3	4
1	0	3	4	0
2	-3	0	0	3
3	-4	0	0	4
4	0	-3	-4	0

(3) BFS

F (n x n)

	1	2	3	4
1	0	3	4	0
2	-3	0	0	3
3	-4	0	0	4
4	0	-3	-4	0

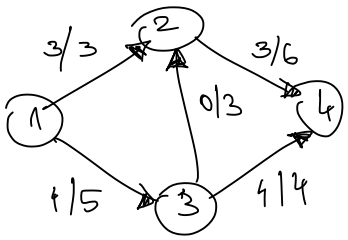
cap

	1	2	3	4
1	0	3	5	0
2	0	0	0	6
3	0	3	0	4
4	0	0	0	0

if (F(x)(y) != C(x)(y) && !visited(y))

Liste de vecini

1: 2 3
2: 1 4 3
→ 3: 1 4 2
4: 2 3



visited 1 2 3 4 : 1 0 0 0

parent 1 2 3 4 : -1 -1 -1 -1

Q: ~~1~~

x = 1 → y = 2
y = 3

visited 1 2 3 4 : 1 0 1 0

parent 1 2 3 4 : -1 -1 1 -1

Q: ~~3~~

x = 3 → y = 1
→ y = 4
→ y = 2

visited 1 2 3 4 : 1 1 1 0

parent 1 2 3 4 : -1 3 1 -1

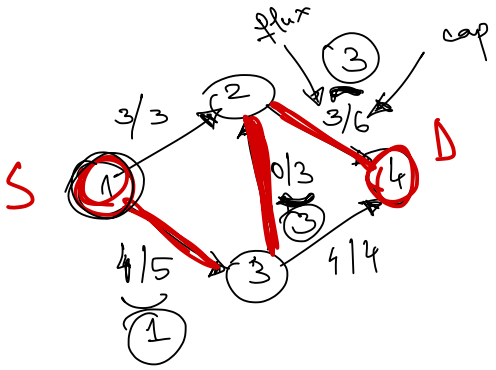
Q: ~~2~~

x = 2 → y = 1
→ y = 4

visited 1 2 3 4 : 1 1 1 1

parent 1 2 3 4 : -1 3 1 2

Q: 4



node=4 $f_{min} = \min(\infty, \underbrace{C[2][4] - F[2][4]}_{6-3}) = 3$

node=2 $f_{min} = \min(3, \underbrace{C[3][2] - F[3][2]}_{3-0}) = 3$

node=3 $f_{min} = \min(3, \underbrace{C[1][3] - F[1][3]}_{5-4}) = 1$

node=1 $f_{min} = 1$

$F = (3+4) + 1 = 8$

F(nxn)

	1	2	3	4
1	0	3	5	0
2	-3	0	-1	4
3	-5	1	0	4
4	0	-4	-4	0

F(nxn)

	1	2	3	4
1	0	3	5	0
2	-3	0	-1	4
3	-5	1	0	4
4	0	-4	-4	0

cap

	1	2	3	4
1	0	3	5	0
2	0	0	0	6
3	0	3	0	4
4	0	0	0	0

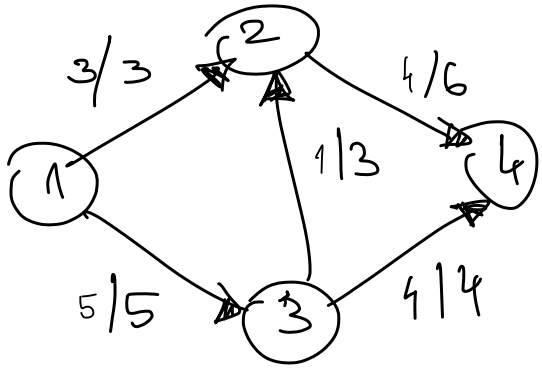
(4) BFS

q: 1

x=1 → y=2 → y=3

q:

8



Lista de vecini

1: 2 3

2: 1 4 3

→ 3: 1 4 2

4: 2 3