Proiectarea Algoritmilor 2009-2010

Laborator 8

Drumuri minime

Cuprins

1 Obiective laborator

- Intelegerea conceptelor de cost, relaxare a unei muchii, drum minim
- Prezentare algoritmi pentru calculul drumurilor minime

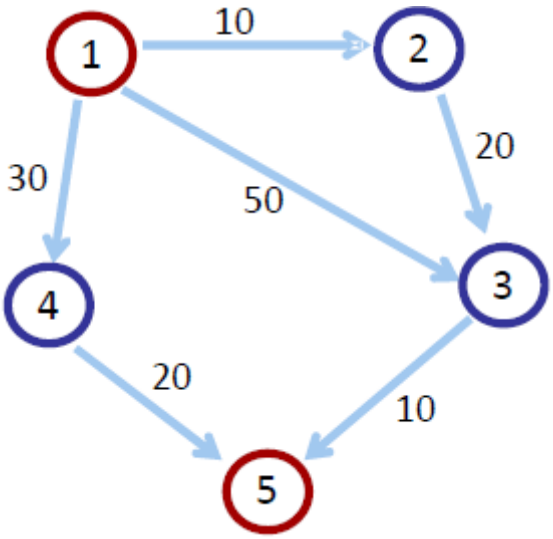
2 Importanţă – aplicaţii practice

- Rutare in cadrul unei retele (telefonice, de calculatoare etc.)
- Gasirea drumului minim dintre doua locatii (Google Maps, GPS etc.)

3 Concepte

3.1 Costul unei muchii si al unui drum

Fiind dat un graf orientat G = (V, E), se considera functia w: E -> W, numita functie de cost, care asociaza fiecarei muchii o valoare numerica. Domeniul functiei poate fi extins, pentru a include si perechile de noduri intre care nu exista muchie directa, caz in care valoarea este $+\infty$. Costul unui drum format din muchiile p_{12} p_{23} … $p_{(n-1)n}$, avand costurile w_{12}, w_{23}, …, $w_{(n-1)n}$, este suma $w = w_{12} + w_{23} + \ldots + w_{(n-1)n}$.



In exemplul alaturat, costul drumului de la nodul 1 la 5 este:

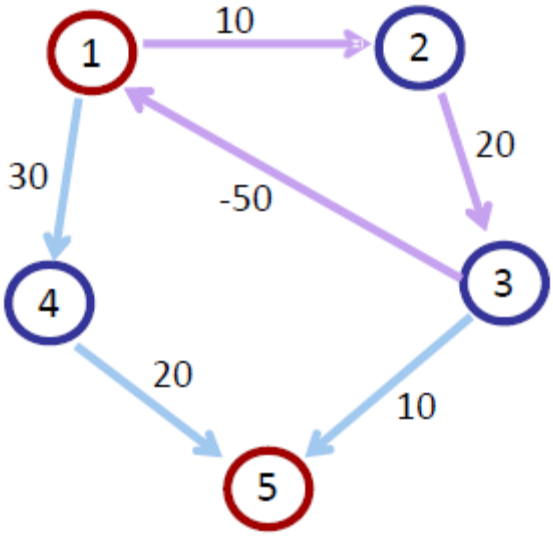
drumul 1: $w_{14} + w_{45} = 30 + 20 = 50$

drumul 2: $w_{12} + w_{23} + w_{35} = 10 + 20 + 10 = 40$

drumul 3: $w_{13} + w_{35} = 50 + 10 = 60$

3.2 Drumul de cost minim

Costul minim al drumului dintre doua noduri este minimul dintre costurile drumurilor existente intre cele doua noduri. In exemplul de mai sus, drumul de cost minim de la nodul 1 la 5 este prin nodurile 2 si 3.



Desi, in cele mai multe cazuri, costul este o functie cu valori nenegative, exista situatii in care un graf cu muchii de cost negativ are

relevanta practica. O parte din algoritmi pot determina drumul corect de cost minim inclusiv pe astfel de grafuri. Totusi, nu are sens cautarea drumului minim in cazurile in care graful contine cicluri de cost negativ – un drum minim ar avea lungimea infinita, intrucat costul sau s-ar reduce la fiecare reparcurgere a ciclului:

In exemplul alaturat, ciclul 1 -> 2 -> 3 -> 1 are costul -20.

drumul 1: $w_{12} + w_{23} + w_{35} = 10 + 20 + 10 = 40$

drumul 2: $(w_{12} + w_{23} + w_{31}) + w_{12} + w_{23} + w_{35} = -20 + 10 + 20 + 10 = 20$

drumul 3: $(w_{12} + w_{23} + w_{31}) + (w_{12} + w_{23} + w_{31}) + w_{12} + w_{23} + w_{35} = -20 + (-20) + 10 + 20 + 10 = 0$

Etc.

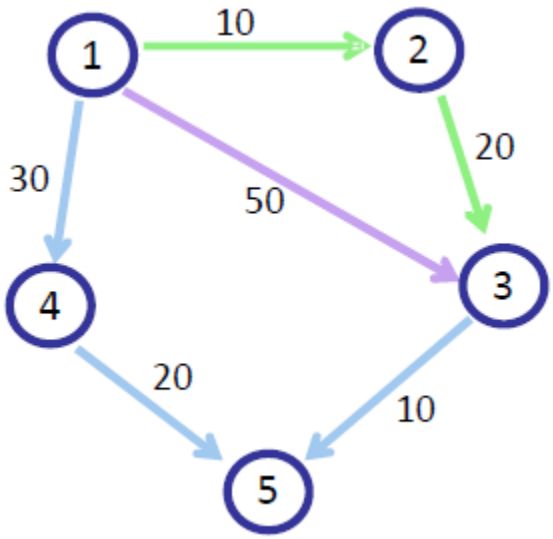
3.3 Relaxarea unei muchii

Relaxarea unei muchii v_1v_2 consta in a testa daca se poate reduce costul ei, trecand printr-un nod intermediar u. Fie w_{12} costul initial al muchiei de la v_1 la v_2, w_{1u} costul muchiei de la v_1 la u, si w_{u2} costul muchiei de la u la v_2. Daca $w > w_{1u} + w_{u2}$, muchia directa este inlocuita cu succesiunea de

muchii v_1u, uv_2.

In exemplul alaturat, muchia de la 1 la 3, de cost $w_{13} = 50$, poate fi relaxata la costul 30, prin nodul intermediar u = 2, fiind inlocuita cu succesiunea w_{12}, w_{23}.

Toti algoritmii prezentati in continuare se bazeaza pe relaxare pentru a determina drumul minim.



4 Drumuri minime de sursa unica

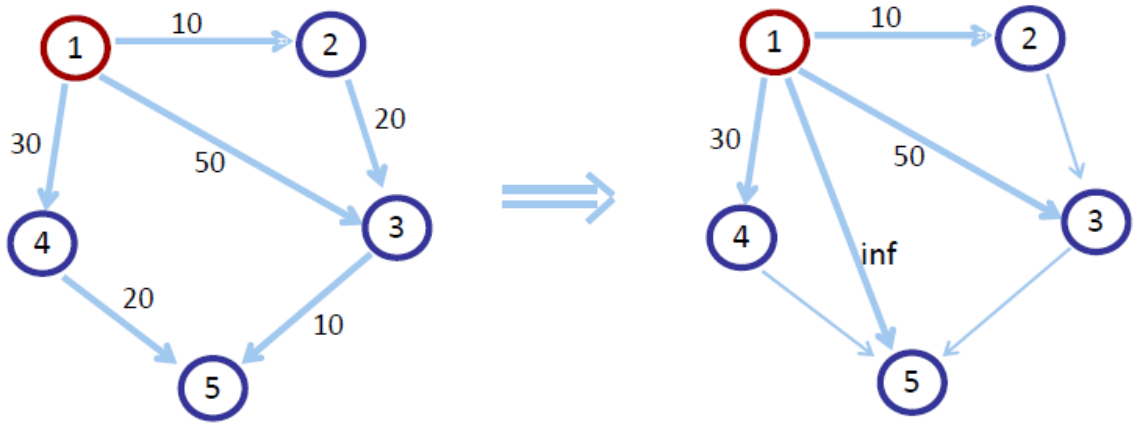
Algoritmii din aceasta sectiune determina drumul de cost minim de la un nod sursa, la restul nodurilor din graf, pe baza de relaxari repetate.

4.1 Algoritmul lui Dijkstra

Dijkstra poate fi folosit doar in grafuri care au toate muchiile nenegative.

Algoritmul este de tip Greedy: optimul local cautat este reprezentat de costul drumului dintre nodul sursa s si un nod v. Pentru fiecare nod se retine un cost estimat d[v], initializat la inceput cu costul muchiei s -> v, sau cu $+\infty$, daca nu exista muchie.

In exemplul urmator, sursa s este nodul 1. Initializarea va fi:



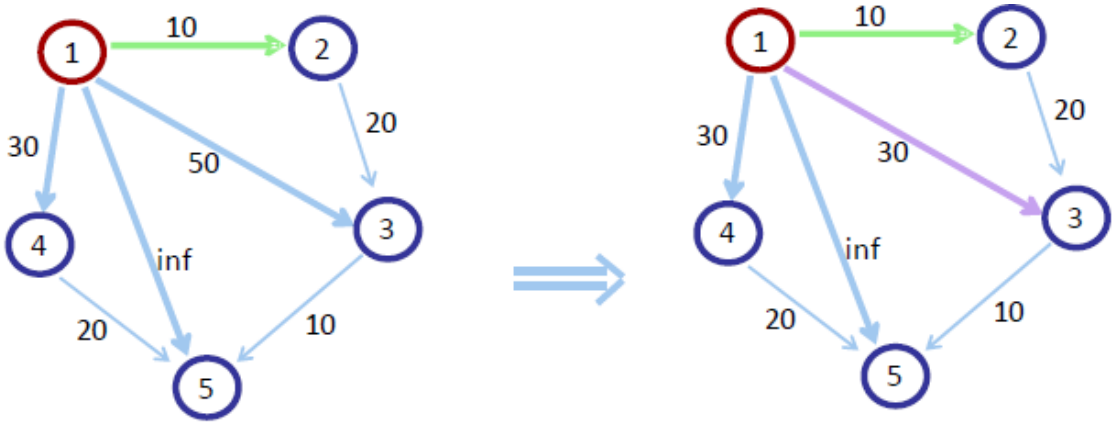
Aceste drumuri sunt imbunatatite la fiecare pas, pe baza celorlalte costuri estimate.

Algoritmul selecteaza, in mod repetat, nodul u care are, la momentul respectiv, costul estimat minim (fata de nodul sursa). In continuare, se incearca sa se relaxeze restul costurilor d[v]. Daca $d[v] < d[u] + w_{uv}$, d[v] ia valoarea $d[u] + w_{uv}$.

Pentru a tine evidenta muchiilor care trebuie relaxate, se folosesc doua structuri: S(multimea de varfuri deja vizitate) si Q (o coada cu prioritati, in care nodurile se afla ordonate dupa distanta fata de sursa) din care este mereu extras nodul aflat la distanta minima. In S se afla initial doar sursa, iar in Q doar nodurile spre care exista muchie directa de la sursa, deci care au $d[nod] < +\infty$.

In exemplul de mai sus, vom initializa S = {1} si Q = {2, 4, 3}.

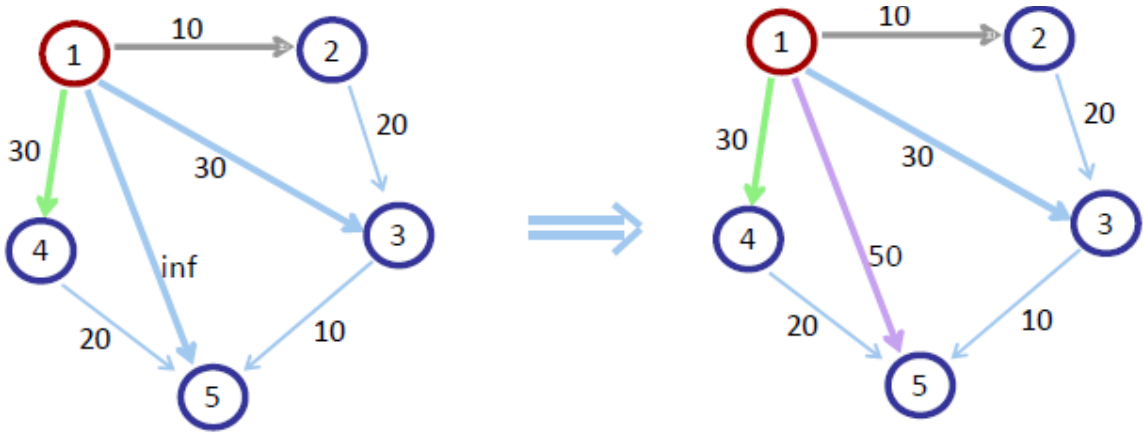
La primul pas este selectat nodul 2, care are d[2] = 10. Singurul nod pentru care d[nod] poate fi relaxat este 3. $d[3] = 50 > d[2] + w_{23} = 10 + 20 = 30$



Dupa primul pas, S = {1, 2} si Q = {4, 3}.

La urmatorul pas este selectat nodul 4, care are d[4] = 30. Pe baza lui, se poate modifica d[5]:

$d[5] = +\infty > d[4] + w_{45} = 30 + 20 = 50$



Dupa al doila pas, S = {1, 2, 4} si Q = {3, 5}.

La urmatorul pas este selectat nodul 3, care are d[3] = 30, si se modifica din nou d[5]:

d[5] = 50 > d[3] + w_{35} = 30 + 10 = 40.

Algoritmul se incheie cand coada Q devine vida, sau cand S contine toate nodurile. Pentru a putea determina si muchiile din care este alcatuit drumul minim cautat, nu doar costul sau final, este necesar sa retinem un vector de parinti P. Pentru nodurile care au muchie directa de la sursa, P[nod] este initializat cu sursa, pentru restul cu null.

Pseudocodul pentru determinarea drumului minim de la o sursa catre celelalte noduri utilizand algoritmul lui Dijkstra este:

```
Dijkstra(sursa, dest):
selectat(sursa) = true
foreach nod in V // V = multimea nodurilor
      daca exista muchie[sursa, nod]
            // initializam distanta pana la nodul respectiv
            d[nod] = w[sursa, nod]
            introdu nod in Q
            // parintele nodului devine sursa
            P[nod] = sursa
      altfel
            d[nod] = +∞ // distanta infinita
            P[nod] = null // nu are parinte
// relaxari succesive
cat timp Q nu e vida
      u = extrage_min (Q)
      selectat(u) = true
      foreach nod in vecini[u] // (*)
```

```
            /* daca drumul de la sursa la nod prin u este mai mic decat
            cel curent */
            daca not selectat(nod) si d[nod] > d[u] + w[u, nod]
                  // actualizeaza distanta si parinte
                  d[nod] = d[u] + w[u, nod]
                  P[nod] = u
                  /* actualizeaza pozitia nodului in coada prioritara */
                  actualizeaza (Q,nod)
// gasirea drumului efectiv
Initializeaza Drum = {}
nod = P[dest]
cat timp nod != null
      insereaza nod la inceputul lui Drum
      nod = P[nod]
```

Reprezentarea grafului ca matrice de adiacenta duce la o implementare ineficienta pentru orice graf care nu este complet, datorita parcurgerii vecinilor nodului u, din linia (*), care se va executa în |V| pasi pentru fiecare extragere din Q, iar pe întreg algoritmul vor rezulta $|V|^2$ pasi. Este preferata reprezentarea grafului cu liste de adiacenta, pentru care numarul total de operatii cauzate de linia (*) va fi egal cu |E|.

Complexitatea algoritmului este $O(|V|^2+|E|)$ în cazul în care coada cu prioritati este implementata ca o cautare liniara. În acest caz functia extrage_min se executa în timp O(|V|), iar actualizeaza(Q) in timp O(1).

O varianta mai eficienta este implementarea cozii ca heap binar. Functia extrage_min se va executa în timp O(lg|V|); functia actualizeaza(Q) se va executa tot în timp O(lg|V|), dar trebuie cunoscuta pozitia cheii nod în heap, adica heapul trebuie sa fie indexat. Complexitatea obtinuta este O(|E|lg|V|) pentru un graf conex.

Cea mai eficienta implementare se obtine folosind un heap Fibonacci pentru coada cu prioritati:

Aceasta este o structura de date complexa, dezvoltata în mod special pentru optimizarea algoritmului Dijkstra, caracterizata de un timp amortizat de O(lg|V|) pentru operatia extrage_min si numai O(1) pentru actualizeaza(Q). Complexitatea obtinuta este O(|V|lg|V| + |E|), foarte buna pentru grafuri rare.

4.2 Algorimtul Bellman – Ford

Algoritmul Bellman Ford poate fi folosit si pentru grafuri ce contin muchii de cost negativ, dar nu poate fi folosit pentru grafuri ce contin cicluri de cost negativ (când

cautarea unui drum minim nu are sens). Cu ajutorul sau putem afla daca un graf contine cicluri.

Algoritmul foloseste acelasi mecanism de relaxare ca si Dijkstra, dar, spre deosebire de acesta, nu optimizeaza o solutie folosind un criteriu de optim local, ci parcurge fiecare muchie de un numar de ori egal cu numarul de noduri si încearca sa o relaxeze de fiecare data, pentru a îmbunatati distanta pâna la nodul destinatie al muchiei curente.

Motivul pentru care se face acest lucru este ca drumul minim dintre sursa si orice nod destinatie poate sa treaca prin maximum |V| noduri (adica toate nodurile grafului); prin urmare, relaxarea tuturor muchiilor de |V| ori este suficienta pentru a propaga pâna la toate nodurile informatia despre distanta minima de la sursa.

Daca, la sfârsitul acestor |E|*|V| relaxari, mai poate fi îmbunatatita o distanta, atunci graful are un ciclu de cost negativ si problema nu are solutie.

Mentinand notatiile anterioare, apseudocodul algoritmului este:

```
BellmanFord(sursa):
// initializari
foreach nod in V // V = multimea nodurilor
      daca muchie[sursa, nod]
            d[nod] = w[sursa, nod]
            P[nod] = sursa
      altfel
            d[nod] = +∞
            P[nod] = null
d[sursa] = 0
p[sursa] = null
// relaxari succesive
for i = 1 to |V|
      foreach (u, v) in E  // E = multimea muchiilor
            daca d[v] > d[u] + w(u,v)
                  d[v] = d[u] + w(u,v)
                  p[v] = u;
// daca se mai pot relaxa muchii
foreach (u, v) in E
      daca d[v] > d[u] + w(u,v)
            fail (”exista cicluri negativ”)
```

Complexitatea algoritmului este în mod evident O(|E|*|V|).

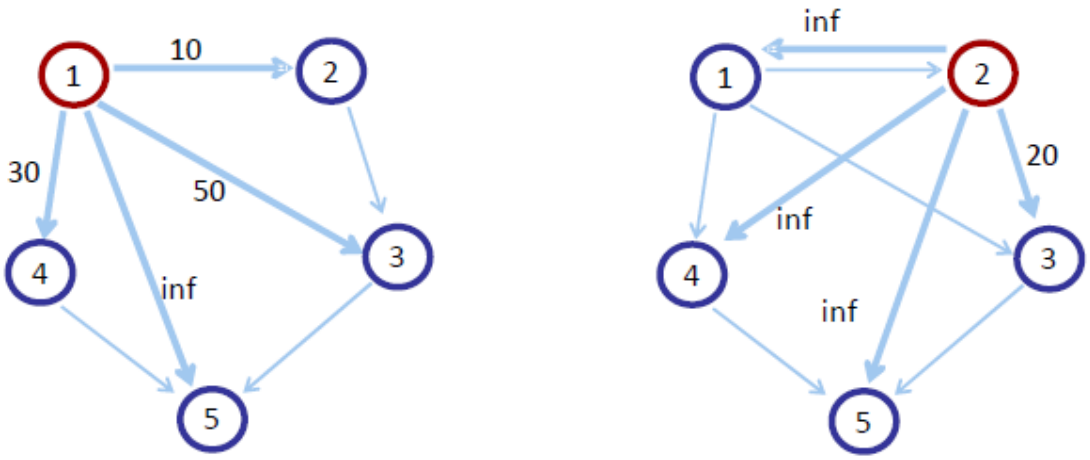
5 Drumuri minime intre oricare doua noduri

5.1 Floyd-Warshall

Algoritmii din aceasta sectiune determina drumul de cost minim dintre oricare doua noduri dintr-un graf. Pentru a rezolva aceasta problema s-ar putea aplica unul din algoritmii de mai sus, considerand ca sursa fiecare nod, pe rand, dar o astfel de abordare ar fi ineficienta.

Algoritmul Floyd-Warshall compara toate drumurile posibile din graf dintre fiecare 2 noduri, si poate fi utilizat si in grafuri cu muchii de cost negativ.

Estimarea drumului optim poate fi retinut intr-o structura tridimensionala d[v_1, v_2, k], cu semnificatia – costul minim al drumului de la v_1 la v_2, folosind ca noduri intermediare doar noduri pana la nodul k. Daca nodurile sunt numerotate de la 1, atunci d[v_1, v_2, 0] reprezinta costul muchiei directe de la v_1 la v_2, considerand $+\infty$ daca aceasta nu exista. Exemplu, pentru v_1 = 1, respectiv 2:



Pornind cu valori ale lui k de la 1 la |V|, ne intereseaza sa gasim cea mai scurta cale de la fiecare v_1 la fiecare v_2 folosind doar noduri intermediare din multimea {1, ..., *k*}. De fiecare data, comparam costul deja estimat al drumului de la v_1 la v_2, deci d[v_1, v_2, k-1] obtinut la pasul anterior, cu costul drumurilor de la v_1 la k si de la k la v_2, adica d[v_1, k, k-1] + d[k, v_2, k-1], obtinute la pasul anterior.

Atunci, d[v_1, v_2, |V|] va contine costul drumului minim de la v_1 la v_2.

Pseudocodul acestui algoritm este:

```
FloydWarshall(G):
n = |V|
int d[n, n, n]
foreach (i, j) in (1..n,1..n)
      d[i, j, 0] = w[i,j] // costul muchiei, sau infinit
```

```
for k = 1 to n
      foreach (i,j) in (1..n,1..n)
            d[i, j, k] = min(d[i, j, k-1], d[i, k, k-1] + d[k, j, k-1])
```

Complexitatea temporala este $O(|V|^3)$, iar cea spatiala este tot $O(|V|^3)$.

O complexitate spatiala cu un ordin mai mic se obtine observând ca la un pas nu este nevoie decât de matricea de la pasul precedent d[i, j, k-1] si cea de la pasul curent d[i, j, k]. O observatie si mai buna este ca, de la un pas k-1 la k, estimarile lungimilor nu pot decât sa scada, deci putem sa lucram pe o singura matrice. Deci, spatiul de memorie necesar este de dimensiune $|V|^2$.

Rescris, pseudocodul algoritmului arata astfel:

```
FloydWarshall(G):
n = |V|
int d[n, n]
foreach (i, j) in (1..n,1..n)
      d[i, j] = w[i,j] // costul muchiei, sau infinit
for k = 1 to n
      foreach (i,j) in (1..n,1..n)
            d[i, j] = min(d[i, j], d[i, k] + d[k, j])
```

Pentru a determina drumul efectiv, nu doar costul acetuia, avem doua variante:

1. Se retine o structura de parinti, similara cu cea de la Dijkstra, dar, bineinteles, bidimensionala.
2. Se foloseste divide et impera astfel:
   - se cauta un pivot k astfel incat cost[i][j] = cost[i][k] + cost[j][k]
   - se apeleaza functia recursiv pentru ambele drumuri -> (i,k),(k,j)
   - daca pivotul nu poate fi gasit, afisam i
   - dupa terminarea functiei recursie afisam extremitatea dreapta a drumului

6 Concluzii

Dijkstra

- calculeaza drumurile minime de la o sursa catre celelalte noduri
- nu poate fi folosit daca exista cicluri de cost negativ
- complexitate minima O(|V|lg|V| + |E|) utilizand heapuri Fibonacci

Bellman – Ford

- calculeaza drumurile minime de la o sursa catre celelalte noduri
- detecteaza existenta ciclurilor de cost negativ
- complexitate O(|V| * |E|)

Floyd – Warshall

- calculeaza drumurile minime intre oricare doua noduri din graf
- poate fi folosit in grafuri cu cicluri de cost negativ, dar nu le detecteaza
- complexitate $O(|V|^3)$

7 Referinţe:

http://en.wikipedia.org/wiki/Dijkstra's_algorithm

http://en.wikipedia.org/wiki/Bellman-Ford_algorithm

http://www.algorithmist.com/index.php/Floyd-Warshall's_Algorithm

http://en.wikipedia.org/wiki/Binary_heap

http://en.wikipedia.org/wiki/Fibonacci_heap

T. Cormen, C. Leiserson, R. Rivest, C. Stein – Introducere în Algoritmi

C. Giumale – Introducere în analiza algoritmilor

Responsabil laborator: Stefan Munteanu (stef8803@gmail.com)

8 Exercitii

8.1

Se da un graf orientat cu muchii de cost nenegativ. Sa se determine costul si muchiile pe care le contine drumul minim intre doua noduri.

8.2

Se da un graf orientat care poate avea si muchii de cost negativ. Sa se determine costul si muchiile pe care le contine drumul minim intre doua noduri.

8.3

Se da un graf orientat. Sa se determine costul si muchiile pe care le contine drumul minim intre oricare doua perechi de noduri.

8.4

Se da un graf orientat. Sa se determine inchiderea tranzitiva. Indicatie - se modifica Floyd Warshall:

8.5

Se da o retea de routere. Intre unele routere exista conexiuni de o anumita latime de banda. Sa se determine drumul optim al unui pachet intre doua routere. Indicatie - se modifica FloydWarshall: