Graf transpus :

- Facem Dijkstra inversă de la 1, 8, 4.
- Mai rămâne Dijkstra cu surse multiple.

1	2	3	4	5	6	7	8
0	8	4	0	3	9	3	0

0 în nodele de start

Sol2: B. Dijkstra (aplicăm alg pe fiecare bd pe graful transpus).

Sol3: Multiple surse Dijkstra pe graful transpus

Întrebare: Care e nr. min. de muchii pe care trebuie să le eliminăm ca să obținem un DAG.

DAG = directed acyclic graph

3 → iau toate submulțimile de noduri; dacă avem ciclu → eliminăm

$2^m(n+m)$ → parcurgere pe fiecare submulțime

6 decembrie 2022

Tema 5 (Seminar)

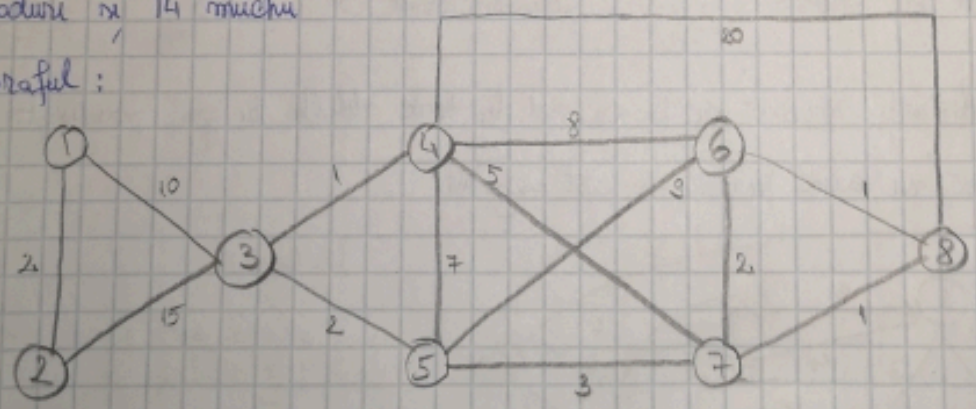
ALGORITMI FUNDAMENTALI

Tema Seminar 5

(Enescu Irina Ștefania 233)

1) Aplicați Bellman Ford pornind din nodul 3 pe un graf neorientat ponderat cu 8 noduri și 14 muchii

Graful :



Algoritmul :

```
pentru fiecare nod:
    d[nod] = ∞
    tata[nod] = 0
d[sursa] = 0

pentru i=1...nr.noduri-1
    pentru fiecare muchie (p,u):
        if d[p] + cost(p,u) < d[u]
            actualizăm d[u]
            tată[u] = p

pentru fiecare muchie (p,u):
    if d[p] + cost(p,u) < d[u]
        ciclu negativ!
```

	1	2	3	4	5	6	7	8
1) d =	∞	∞	0	∞	∞	∞	∞	∞

Muchii:

1 3 : 10
1 2 : 2
2 3 : 15
3 4 : 1
3 5 : 2
4 5 : 7
4 7 : 5
4 6 : 8
5 7 : 3
5 6 : 9
6 7 : 2
6 8 : 1
7 8 : 1
4 8 : 20

2) Iteratia 1:

		1	2	3	4	5	6	7	8	
	d =	~~∞~~	∞	0	∞	∞	∞	∞	∞	
1.	(1,3)-10	10	~~∞~~	0	∞	∞	∞	∞	∞	
2.	(1,2)-2	10	12	0	∞	∞	∞	∞	∞	
3.	(2,3)-15	10	12	0	~~∞~~	∞	∞	∞	∞	(15 > 12)
4.	(3,4)-1	10	12	0	1	~~∞~~	∞	∞	∞	
5.	(3,5)-7	10	12	0	1	8	∞	~~∞~~	∞	
6.	(4,7)-5	10	12	0	1	8	~~∞~~	6	∞	
7.	(4,6)-8	10	12	0	1	8	9	6	∞	
8.	(5,7)-3	10	12	0	1	8	9	6	∞	(11 > 6)
9.	(5,6)-9	10	12	0	1	8	9	6	∞	(17 > 9)
10.	(6,7)-2	10	12	0	1	8	9	6	~~∞~~	(11 > 6)
11.	(6,8)-1	10	12	0	1	8	9	6	~~10~~	
12.	(7,8)-1	10	12	0	1	8	9	6	7	
13.	(4,8)-20	10	12	0	1	8	9	6	7	(21 > 7)

Am actualizat în momentul în care distanța de la nodul x (de plecare) + costul muchiei este mai mică decât distanța de la nodul y (de sosire), înseamnă că am găsit un drum mai bun. Trebuie să refacem actualizările de n (nr. noduri) - 1 ori pentru a ne asigura că am luat cea mai bună variantă în considerare - puteam calcula inițial o distanță cu o distanță intermediară ce ne va minimiza ulterior ⇒ și distanța calculată trebuie minimizată. Pentru a exemplifica acest lucru am lăsat o muchie pentru final:

		1	2	3	4	5	6	7	8
	d =	10	12	0	1	~~8~~	9	6	7
14.	(3,5)-2	10	12	0	1	2	(9)	(6)	(7)

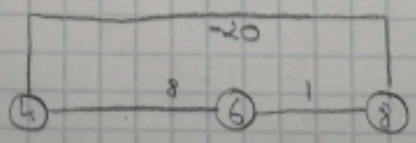
↓

Observăm că acum putem ajunge în 7 mai ieftin, cu 5, nu cu 6. Aceeași problemă apare și la 8 noduri, iar celelalte iterații le vor rezolva. Aceasta deoarece la prima iterație, deci vor păstra doar modificările ⇒ rezultatul final:

	1	2	3	4	5	6	7	8
d =	10	12	0	1	2	7	5	6

3) Observăm că nu avem cicluri negative, deci este ok. Acestea apăreau în cazul în care distanța de la x + costul muchiei este mai mică decât distanța de la y, distanțele fiind calculate după toate iterațiile. În acest caz puteam minimiza la infinit:

exemplu: daca aveam



$8 + 1 - 20 = -11 + 8 + 1 - 20 = -22 + 8 + 1 - 20 = - \dots \to -\infty$

2) Exemplificați algoritmul A* și faceți o comparație între el și Dijkstra.

Atât A*, cât și Dijkstra calculează distanța minimă dintre două noduri într-un graf. A* nu reprezintă însă distanțele, dorind cât mai rapid soluția bună cu ajutorul funcțiilor de estimare a distanței de la un nod la altul:

$f(n) = g(n) + h(n)$ ∀ un nod → combină distanța deja parcursă până la ↓ costul estimat celui mai bune sol. ce trece prin n un nod cu distanța estimată până la final.

Algoritmul A*:

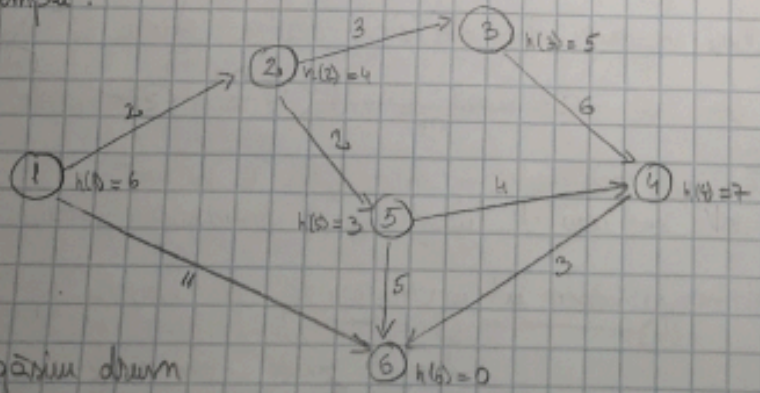
A* pornește de la nodul inițial cu costul 0 și cu o estimare a distanței până la nodul final. Costul nodului și estimarea formează euristica trecerii printr-un nod. Nodul inițial este adăugat în „lista deschisă" - o coadă de priorități (inversată) după $f(n)$ → în vârf avem cea mai mică val.

Se alege primul nod din listă până când lista deschisă este goală (nu există soluții) sau am ajuns la nodul final (returnăm costul). Dacă nu am ajuns la nodul final, verificăm vecinii nodului curent, salvând fiecare vecin al nodului cu costurile cumulative corespunzătoare fiecăruia. Se gestionează de asemenea o „listă închisă" - un vector de vizitat: dacă un vecin e deja în listă cu un cost egal sau mai mic, nu se va lua în considerare nodul sau calea sa. Dacă un nod deja vizitat e accesat cu unul nou, dar cu un cost mai mare, acesta va fi eliminat din lista închisă, iar algoritmul va rula de la nodul nou.

Ulterior, o nouă estimare a distanței de la nodul nou la nodul final este adăugată la cost => se formează valoarea euristică. Se adaugă nodul în listă deschisă doar dacă nu există un nod identic cu valoare euristică mai mică sau egală.

Algoritmul se repetă → fiecare vecin, iar nodul curent e marcat ca vizitat. Următorul nod se extrage din lista deschisă și se repetă procesul.

Exemplu:



Vrem să găsim drum minim (1) → (6).

Diferența față de Dijkstra:

- în Dijkstra reținem d[vnod]
- în A* reținem d[vnod] + h(k) ↓ apare distanța estimată

Pornim de la ① cu $h(1) = 6$: ↑ cost parcă la modul 2.

→ putem merge ① $\xrightarrow{2}$ ② $\Rightarrow f(u) = g(u) + h(u) = 2 + 4 = 6$ ✓

① $\xrightarrow{11}$ ⑥ $\Rightarrow f(u) = g(u) + h(u) = 11 + 0 = 11$ ⓐ

Alegem prima variantă deoarece $6 < 11$, dar păstrăm și a doua variantă pt. verificare finală

→ putem merge ① $\xrightarrow{2}$ ② $\xrightarrow{3}$ ③ $\Rightarrow f(u) = (2+3) + 5 = 10$ ⓑ

① $\xrightarrow{2}$ ② $\xrightarrow{2}$ ⑤ $\Rightarrow f(u) = (2+2) + 3 = 7$ ✓

Alegem a doua variantă deoarece $7 < 10$, dar o păstrăm și pe prima.

→ putem merge ① $\xrightarrow{2}$ ② $\xrightarrow{2}$ ⑤ $\xrightarrow{4}$ ④ $\Rightarrow f(u) = (2+2+4) + 7 = 15$ ⓒ

① $\xrightarrow{2}$ ② $\xrightarrow{2}$ ⑤ $\xrightarrow{5}$ ⑥ $\Rightarrow f(u) = (2+2+5) + 0 = 9$ ✓

Alegem a doua variantă deoarece $9 < 15$, dar o păstrăm și pe prima.

Acum verificăm muchiile salvate anterior deoarece 6 nu mai are vecini, fiind nodul căutat:

ⓐ $11 > 9$ da, păstrăm 9

ⓑ $10 > 9$ da, păstrăm 9

ⓒ $15 > 9$ da, păstrăm 9

Ne oprim deoarece am ajuns la ⑥ : ① – ② – ⑤ – ⑥ : 9