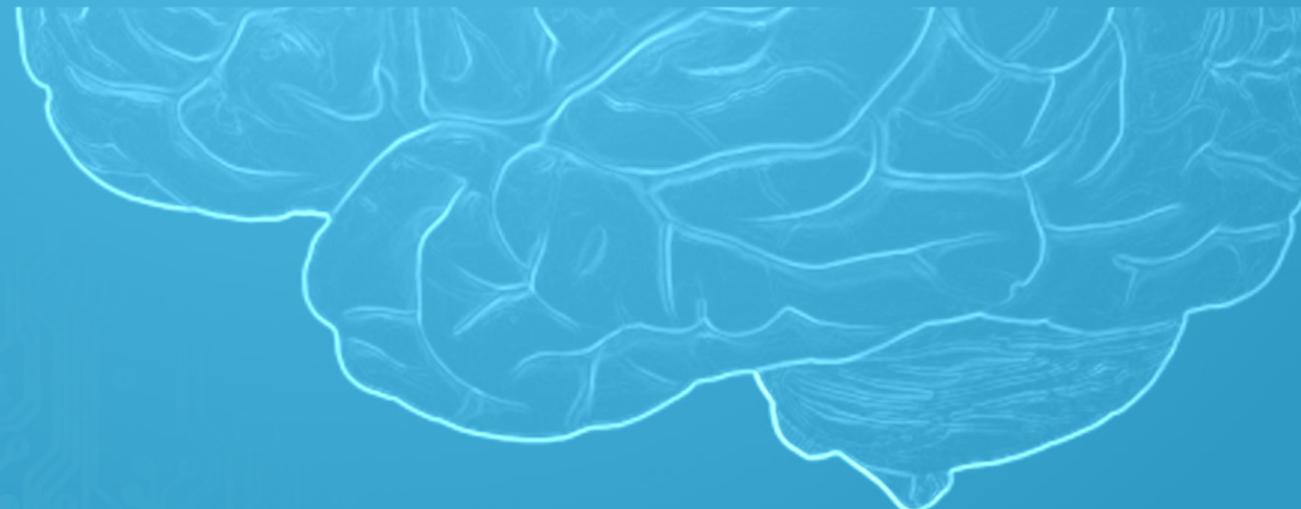
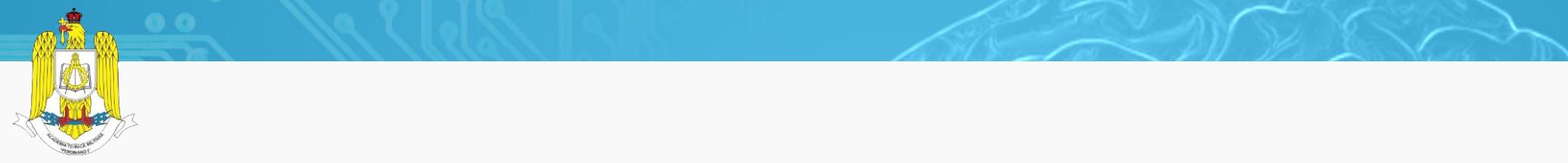


Structuri de date și algoritmi Grafuri – Drumuri Minime

Lect. Dr. Ing. Cristian Chilipirea – cristian.chilipirea@mta.ro







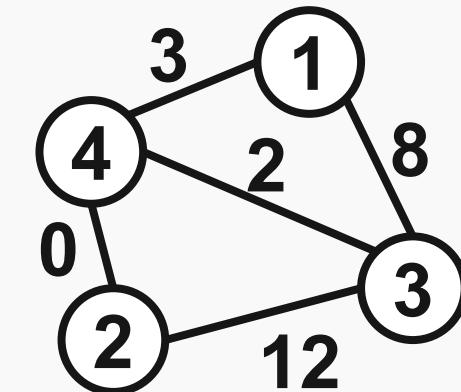
Grafuri ponderate – Weighted Graphs - Reminder

Pentru un graf $G = (V, E)$ se adaugă funcția W ce asociază un cost fiecărei muchii.

$$w(1,4) = 3$$

$$w(2,3) = 12$$

$$w(2,4) = 0$$



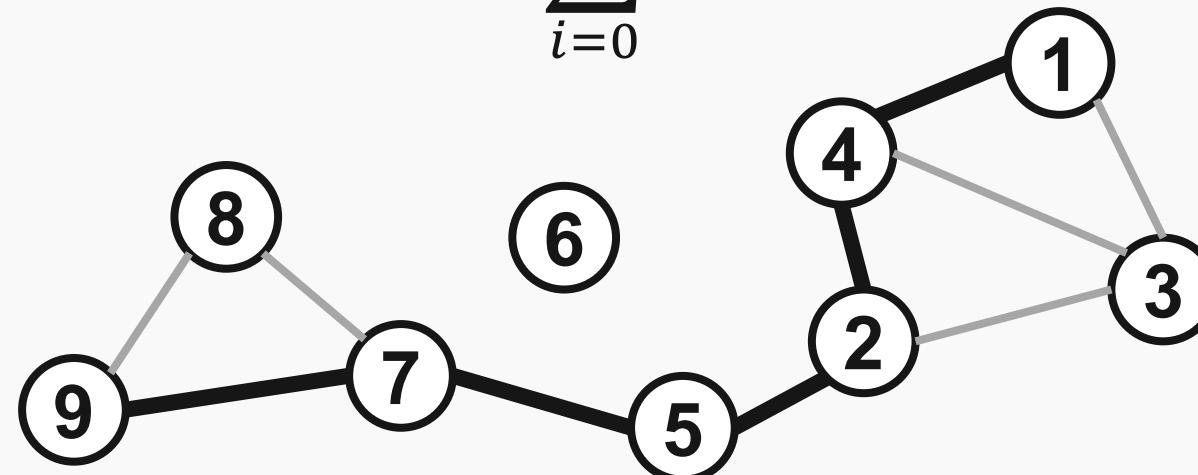


Drumuri – Paths - Reminder

Pentru un graf $G = (V, E)$ un drum este un set de noduri $P = (\nu_1, \nu_2, \nu_3, \dots, \nu_N)$ cu $(\nu_i, \nu_{i+1}) \in E, \forall i$

Lungimea unui drum este:

$$w(P) = \sum_{i=0}^{N-1} w(\nu_i, \nu_{i+1})$$





Aplicații

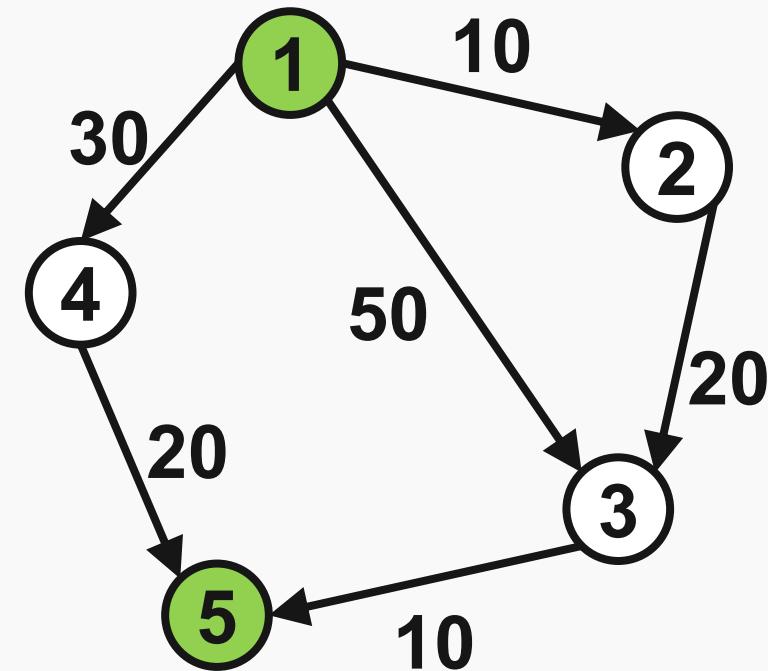
- Trafic rutier, hărți GPS, hărți în jocuri.
- Drum minim în rețea în funcție nu de lungimea fizică dar în funcție de latență sau număr de hop-uri.
- Proiectare hardware, căi circuite electrice; VLSI.



Drumuri Cost Minim - Exemplu

Obiectiv: Fie v și v' se caută $P = (v_1, v_2, v_3, \dots, v_N)$ cu $v = v_1$ și $v' = v_N$ cu $w(P)$ minim.

$P = (1, \dots, 5) ?$



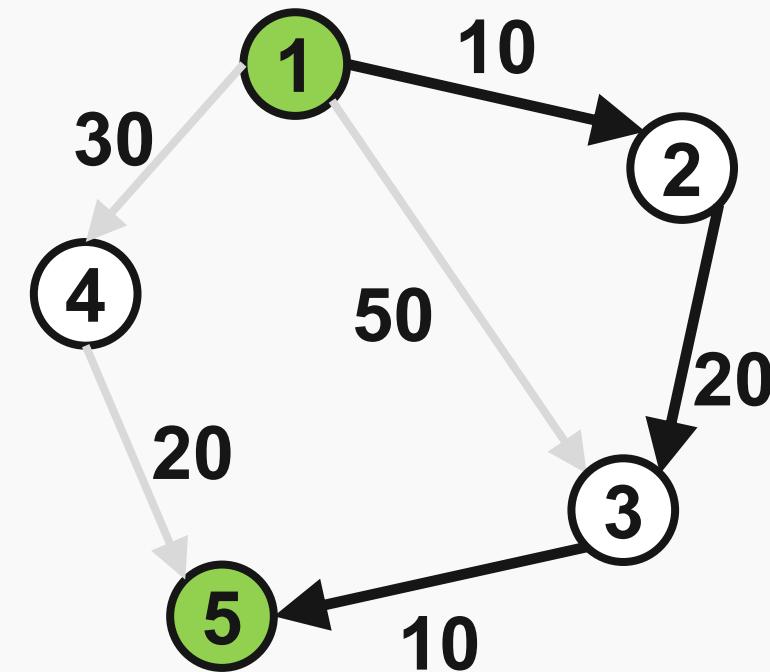


Drumuri Cost Minim - Exemplu

Obiectiv: Fie v și v' se caută $P = (v_1, v_2, v_3, \dots, v_N)$ cu $v = v_1$ și $v' = v_N$ cu $w(P)$ minim.

$$P = (1, \dots, 5) ?$$

$$w(1,2,3,5) = 10 + 20 + 10 = 40$$



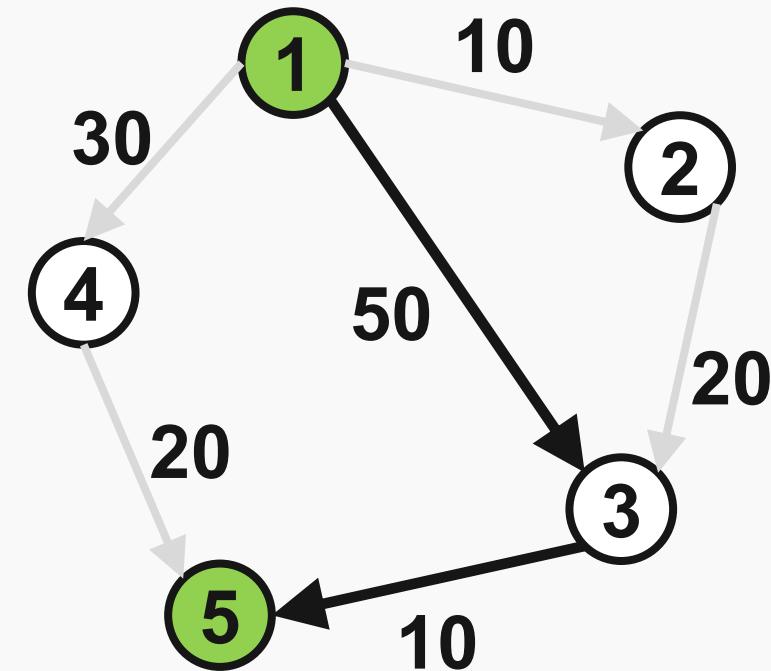


Drumuri Cost Minim - Exemplu

Obiectiv: Fie v și v' se caută $P = (v_1, v_2, v_3, \dots, v_N)$ cu $v = v_1$ și $v' = v_N$ cu $w(P)$ minim.

$P = (1, \dots, 5) ?$

$$w(1,3,5) = 50 + 10 = 60$$



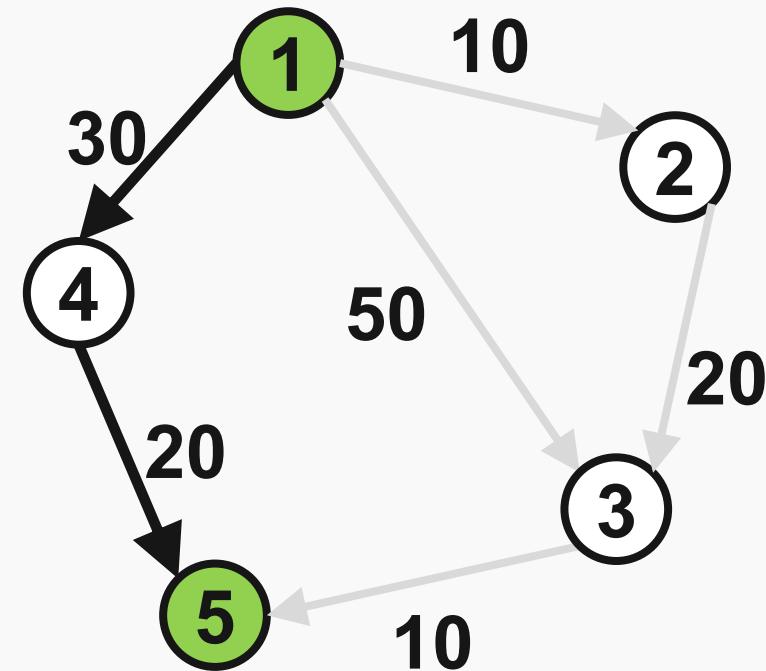


Drumuri Cost Minim - Exemplu

Obiectiv: Fie v și v' se caută $P = (v_1, v_2, v_3, \dots, v_N)$ cu $v = v_1$ și $v' = v_N$ cu $w(P)$ minim.

$P = (1, \dots, 5) ?$

$$w(1,4,5) = 30 + 20 = 50$$

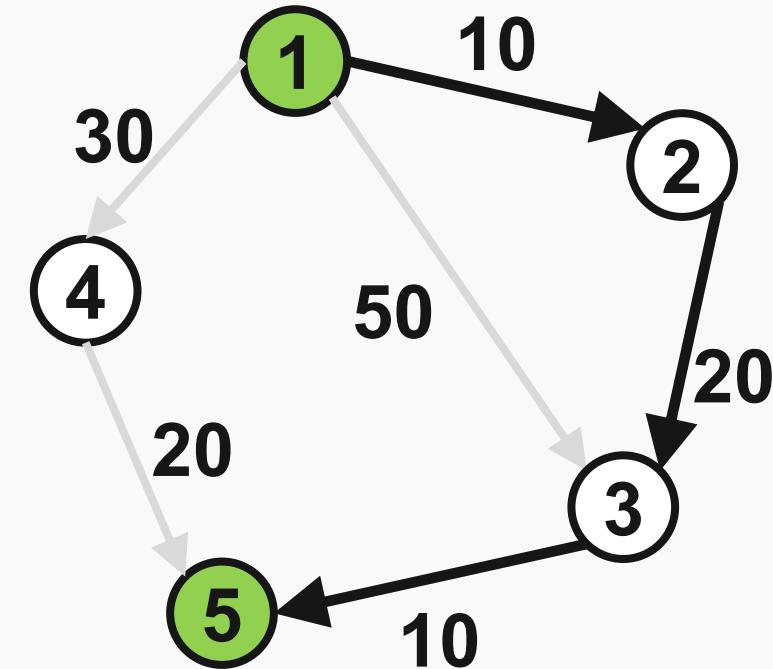




Drumuri Cost Minim - Exemplu

Obiectiv: Fie v și v' se caută $P = (v_1, v_2, v_3, \dots, v_N)$ cu $v = v_1$ și $v' = v_N$ cu $w(P)$ minim.

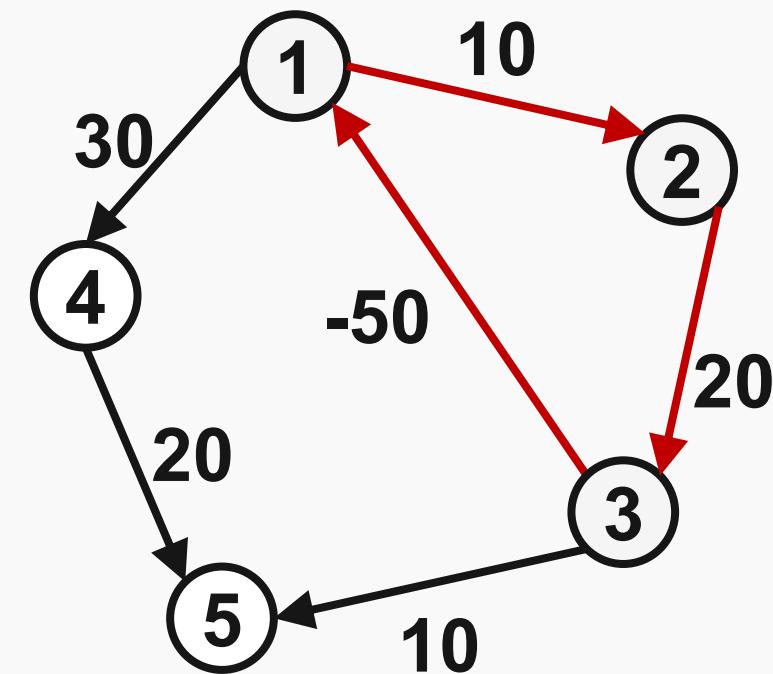
$$\begin{aligned} \min(w(P)) \\ w(1,4,5) &= 30 + 20 = 50 \\ w(1,3,5) &= 50 + 10 = 60 \\ w(1,2,3,5) &= 10 + 20 + 10 = 40 \\) &= w(1,2,3,5) \end{aligned}$$





Cicluri negative

Dacă avem un ciclu cu cost negativ ($10 + 20 - 50 = -20$) am putea merge pe ciclu până ajungem la $-\infty$.





Notății

- Fie $\delta(u, v)$ distanța minimă dintre nodurile u și v .
- Fie $v.d$ este distanța minimă **estimată** de la un nod sursă (s) la v .
 - Toate estimările la început sunt de distanță ∞ .
 - Estimarea $s.d$ este 0.
- Fie $v.\pi$ predecesorul lui v către nodul sursă (s).

INITIALIZE-SINGLE-SOURCE(G, s)

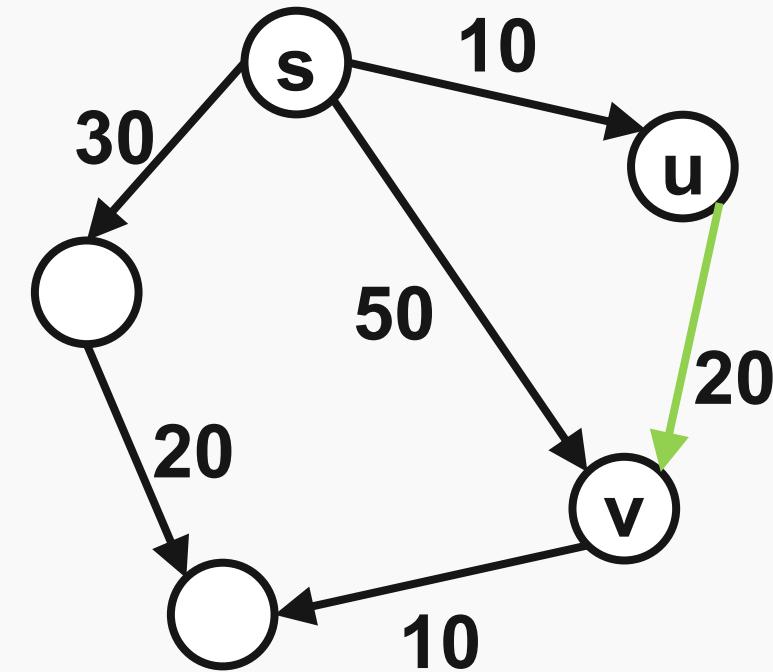
- 1 **for** each vertex $v \in G.V$
- 2 $v.d = \infty$
- 3 $v.\pi = \text{NIL}$
- 4 $s.d = 0$



Relaxarea unei muchii

RELAX(u, v, w)

- 1 **if** $v.d > u.d + w(u, v)$
- 2 $v.d = u.d + w(u, v)$
- 3 $v.\pi = u$

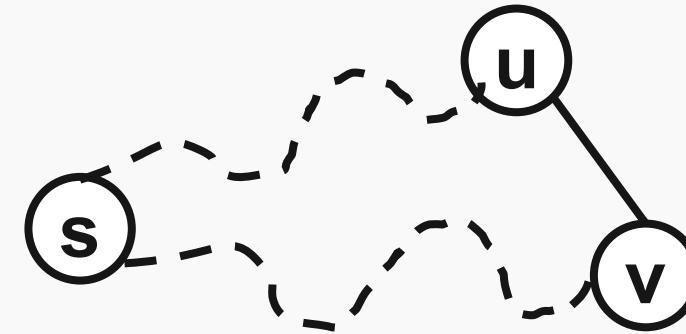




Proprietăți drumuri optime și relaxare

■ Inegalitatea triunghiurilor

$$\forall (u, v) \in E; \delta(s, v) \leq \delta(s, u) + w(u, v)$$



■ Limita superioară

$$v.d \geq \delta(s, v); \forall v \in V;$$

- Când $v.d = \delta(s, v)$ nu se mai schimbă



Proprietăți drumuri optime și relaxare

- **Nu este cale**

$$v.d = \delta(s, v) = \infty$$

- **Convergență**

Dacă muchia (u, v) face parte din calea minimă spre v și $u.d = \delta(s, u)$ atunci după relaxare $v.d = \delta(s, v)$

- **Proprietatea relaxării unei căi**

Dacă $P = (s, v_1, v_2, v_3, \dots, v_k)$ este o cale minimă de la s la v_k și relaxăm muchiile în ordine $(s, v_1), (v_1, v_2) \dots (v_{k-1}, v_k)$, atunci $v_k.d = \delta(s, v_k)$



Proprietăți drumuri optime și relaxare

- Dacă $P = (v_1, v_2, v_3, \dots, v_N)$ este optim atunci $\forall P' = (v_i, \dots, v_j)$ este optim.

Demonstrație: dacă $\exists P'' = (v_i, \dots, v_j)$ aî $\delta(P'') < \delta(P')$ atunci calea P'' ar putea fi folosită pentru a scurta P .

- Un drum optim nu poate avea cicluri pozitive sau negative!
- Un drum optim poate conține maxim $|V|$ noduri.



Algoritmul Bellman Ford



Bellman & Ford

P-923
8-14-56
-1-

ON A ROUTING PROBLEM

By

Richard Bellman

§1. Introduction.

The problem we wish to treat is a combinatorial one involving the determination of an optimal route from one point to another. These problems are usually difficult when we allow a continuum, and when we admit only a discrete set of paths, as we shall do below, they are notoriously so.

The purpose of this paper is to show that the functional equation technique of dynamic programming, [1], [2], combined with the concept of approximation in policy space, yields a method of successive approximations which is readily accessible to either hand or machine computation for problems of realistic magnitude. The method is distinguished by the fact that it is a method of exhaustion, i.e. it converges after a finite number of iterations bounded in advance.

NETWORK FLOW THEORY

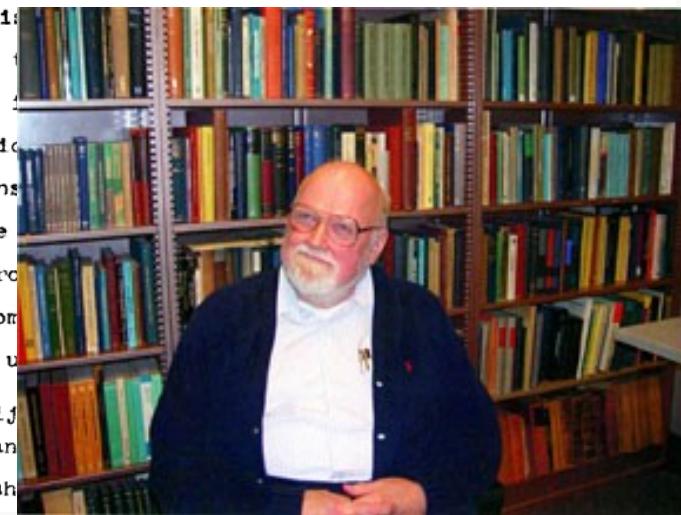
L. R. Ford, Jr.

1. INTRODUCTION

A network (or linear graph) is a collection of points or nodes, some of which may be joined together by arcs. We shall denote the points by P_i , $i = 0, 1, 2, \dots, N$, and denote the arc joining P_i to P_j in that order by A_{ij} . (Note that there may also be an arc A_{ji} joining P_j to P_i .) We may also have associated with the arc A_{ij} a capacity c_{ij} and a length (or cost) ℓ_{ij} . We shall assume these to be positive integers.

We shall also distinguish an origin, and P_N , the destination. If the network represents a rail network, P_i represents a station, P_N a warehouse at which all traffic originates. P_i represents a constraint on the system. The capacity of an arc is the maximum shipping capacity from P_i to P_j or the upper bound on the distance from P_i to P_j or the upper bound on the cost of shipping from P_i to P_j . We agree that if $c_{ij} = 0$, then A_{ij} does not exist.

Evidently many problems can be solved in this framework. We shall





Algoritmul Bellman Ford

- Se poate aplica și pe grafuri ce au muchii cu valori negative.
- Poate fi folosit pentru detectarea ciclurilor negative
- Un drum minim poate conține maxim $|V|$ noduri =>
=> orice cale poate fi relaxată de maxim $|V| - 1$ ori



```
bellmanFordAlgorithm(G, source) {
    int d[|G.V|] = { INFINITY };
    int prev[|G.V|] = { UNDEFINED };
    dist[source] = 0;
    for (i = 0; i < |G.V| - 1; i++)
        for each (edge(u,v) in G.E)
            if (d[u] + w(u, v) < d[v]) {
                d[v] = d[u] + w(u, v);
                prev[v] = u;
            }
    for each (edge(u, v) in G.E)
        if (d[u] + w(u, v) < d[v])
            print "Negative-weight cycle"
    return dist, prev;
}
```



Complexitate?

```
bellmanFordAlgorithm(G, source) {  
    int d[|G.V|] = { INFINITY };  
    int prev[|G.V|] = { UNDEFINED };  
    dist[source] = 0;  
    for (i = 0; i < |G.V| - 1; i++)  
        for each (edge(u,v) in G.E)  
            if (d[u] + w(u, v) < d[v]) {  
                d[v] = d[u] + w(u, v);  
                prev[v] = u;  
            }  
    }  
}
```



Complexitate?

$$O(|V| * |E|)$$



Bellman-Ford Demonstrație corectitudine

- Lemma: După $|V| - 1$ iterații avem $v.d = \delta(s, v) \forall v \in V$
- Demonstrație:
 - Dacă $P = (v_0, v_1, v_2, v_3, \dots, v_k); s = v_0; v = v_k$ și P cale minimă
 - P are maxim $|V| - 1$ muchii și $k \leq |V| - 1$.
 - La iterația i este relaxată și muchia (v_{i-1}, v_i)
 - Din proprietatea relaxării unei căi rezultă
$$v.d = \delta(s, v) \forall v \in V$$



Algoritmul Bellman Ford – caz particular DAG

- Inițializare. $O(|V|)$
- Sortăm topologic graful. $O(|V| + |E|)$
- Aplicăm relaxarea muchiilor o singură dată. $O(|E|)$
- Nodurile din stânga source rămân cu distanță infinită.
- Complexitate totală $O(|V| + |E|)$



Algoritmul lui Dijkstra



Algoritmul lui Dijkstra

Numerische Mathematik 1, 269–271 (1959)

A Note on Two Problems in Connexion with Graphs

By

E. W. DIJKSTRA

We consider n points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict ourselves to the case where at least one path exists between any two nodes. We now consider two problems.

Problem 1. Construct the tree of minimum total length between the n nodes.
(A tree is a graph with one and only one path between every two nodes.)

In the course of the construction that we present here, the branches are subdivided into three sets:

I. the branches definitely assigned to the tree under construction (they will form a subtree);

II. the branches from which the next branch to be added to set I, will be selected;

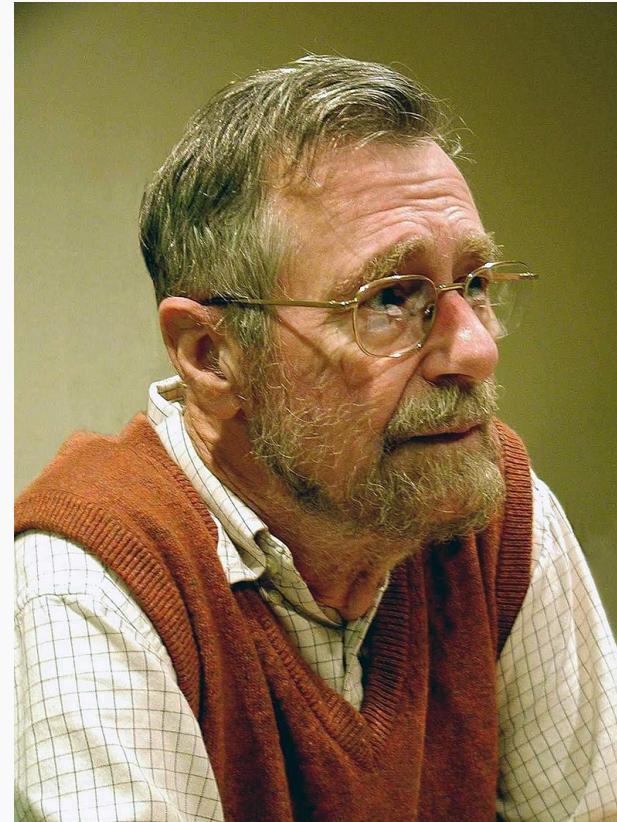
III. the remaining branches (rejected or not yet considered).

The nodes are subdivided into two sets:

A. the nodes connected by the branches of set I,

B. the remaining nodes (one and only one branch of set II will lead to each of these nodes).

We start the construction by choosing an arbitrary node as the only member of set A, and by placing all branches that end in this node in set II. To start with, set I is empty. From then onwards we perform the following two steps repeatedly.





Algoritmul lui Dijkstra

- Funcționează doar pentru muchii de cost pozitiv

Considerăm un drum minim : $s, v_1, v_2 \dots v_k$

Atunci toate subdrumurile $s, \dots, v_i, i < k$ vor fi de asemenea drumuri minime: $\delta(s, s) \leq \delta(s, v_1) \leq \dots \leq \delta(s, v_k)$

Idee: Ordonarea nodurilor din graf în funcție de distanța lor optimală $\delta(u)$ față de nodul sursă s .



Algoritmul lui Dijkstra

```
dijkstraAlgorithm(G, source) {
    int d[|G.V|] = { INFINITY };
    d[source] = 0;
    for each(node in G.V)
        push(LIST, node);

    while (!isEmpty(LIST)) {
        node = pop_MIN(LIST, d);
        for each (neighbor of node) {
            if (d[node] + w(node, neighbor) < d[neighbor]) {
                d[neighbor] = d[node] + w(node, neighbor);
                prev[neighbor] = node;
                update(LIST, d);
            }
        }
    }
    return dist, prev;
}
```



Intuiția Algoritmului Dijkstra

Care este cel mai apropiat nod de sursa s ?



Intuiția Algoritmului Dijkstra

Care este cel mai apropiat nod de sursă s ? Chiar s .

La prima relaxare (muchii care provin din sursă s):

$$d(u) = \infty > d(s) + w(s, u); \mathbf{d}(u) = w(s, u)$$

Care este cel mai apropiat nod de sursă, diferit de s ?



Intuiția Algoritmului Dijkstra

Care este cel mai apropiat nod de sursă s ? Chiar s .

La prima relaxare (muchii care provin din sursă s):

$$d(u) = \infty > d(s) + w(s, u); \mathbf{d}(u) = w(s, u)$$

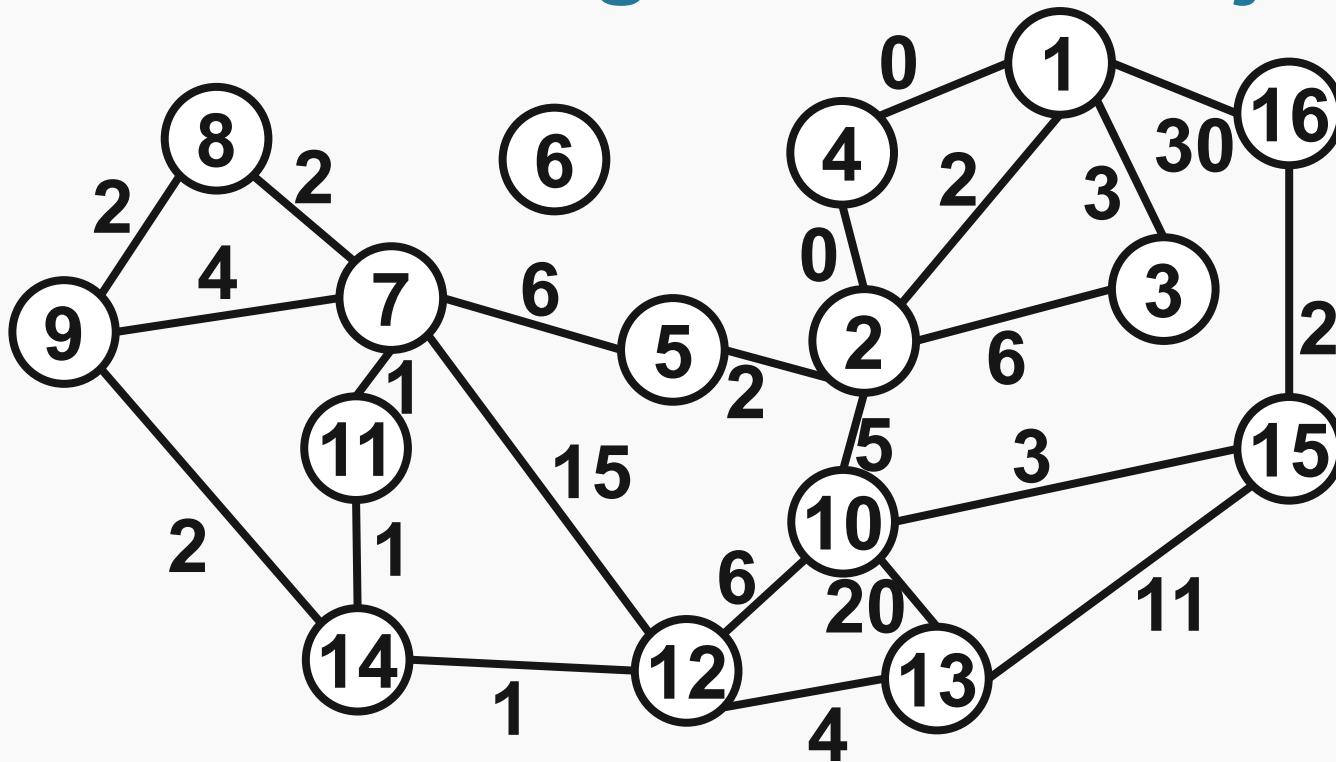
Care este cel mai apropiat nod de sursă, diferit de s ?

- Nodul care este legat de sursă prin muchia minimă.

Deci nodul extras u din mulțimea Q în următoarea
iterație este chiar nodul corect : $\mathbf{d}(u) = w(s, u) = \delta(u)$



Algoritmul lui Dijkstra

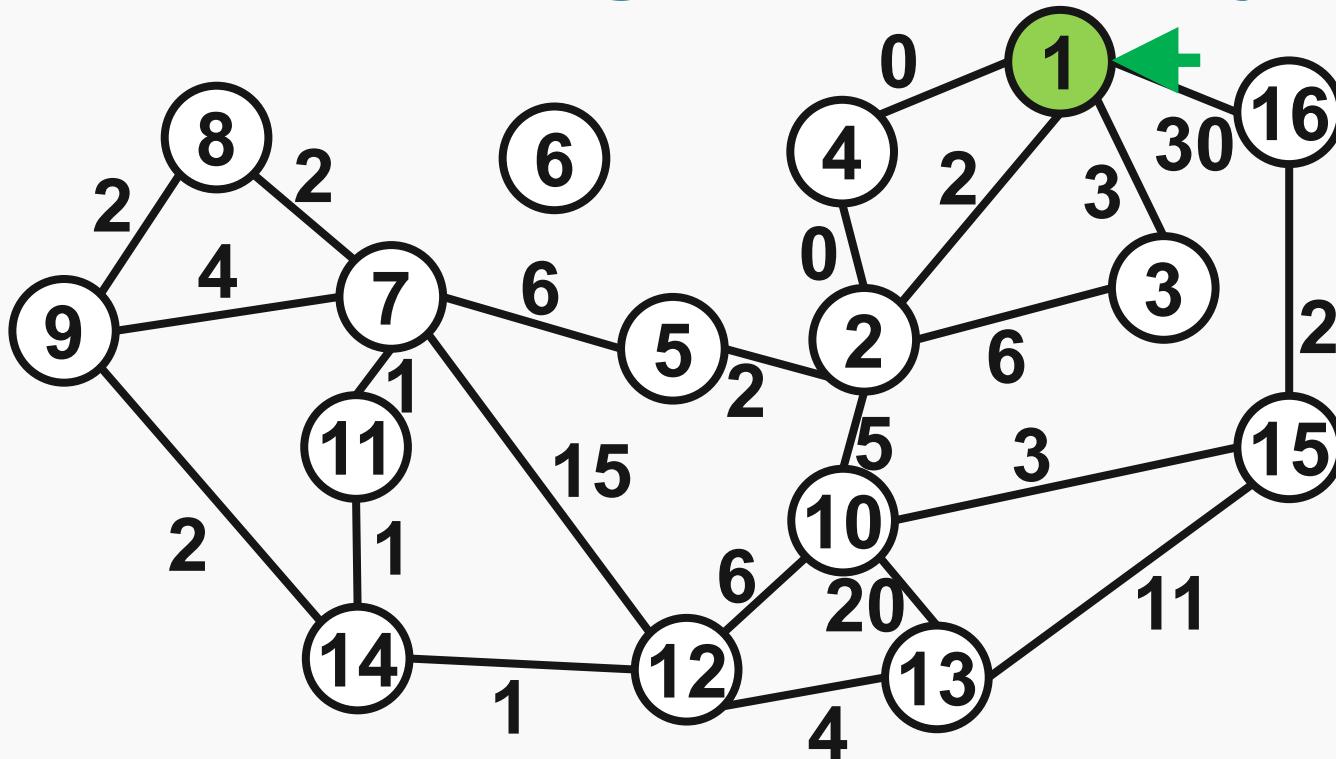


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	∞														

#	D	π
1	0	-
2	∞	-
3	∞	-
4	∞	-
5	∞	-
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	∞	-
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	∞	-



Algoritmul lui Dijkstra

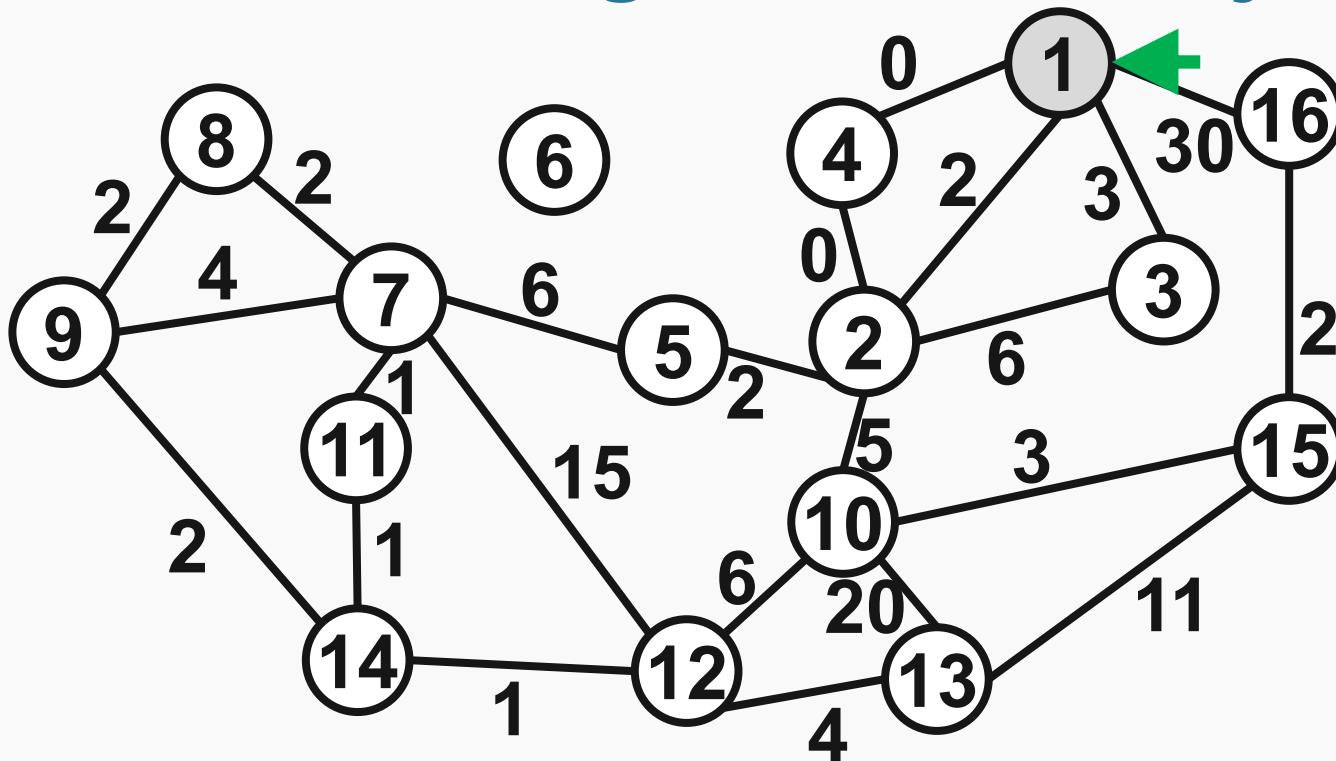


2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	0	∞	30										

#	D	π
1	0	-
2	2	1
3	3	1
4	0	1
5	∞	-
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	∞	-
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

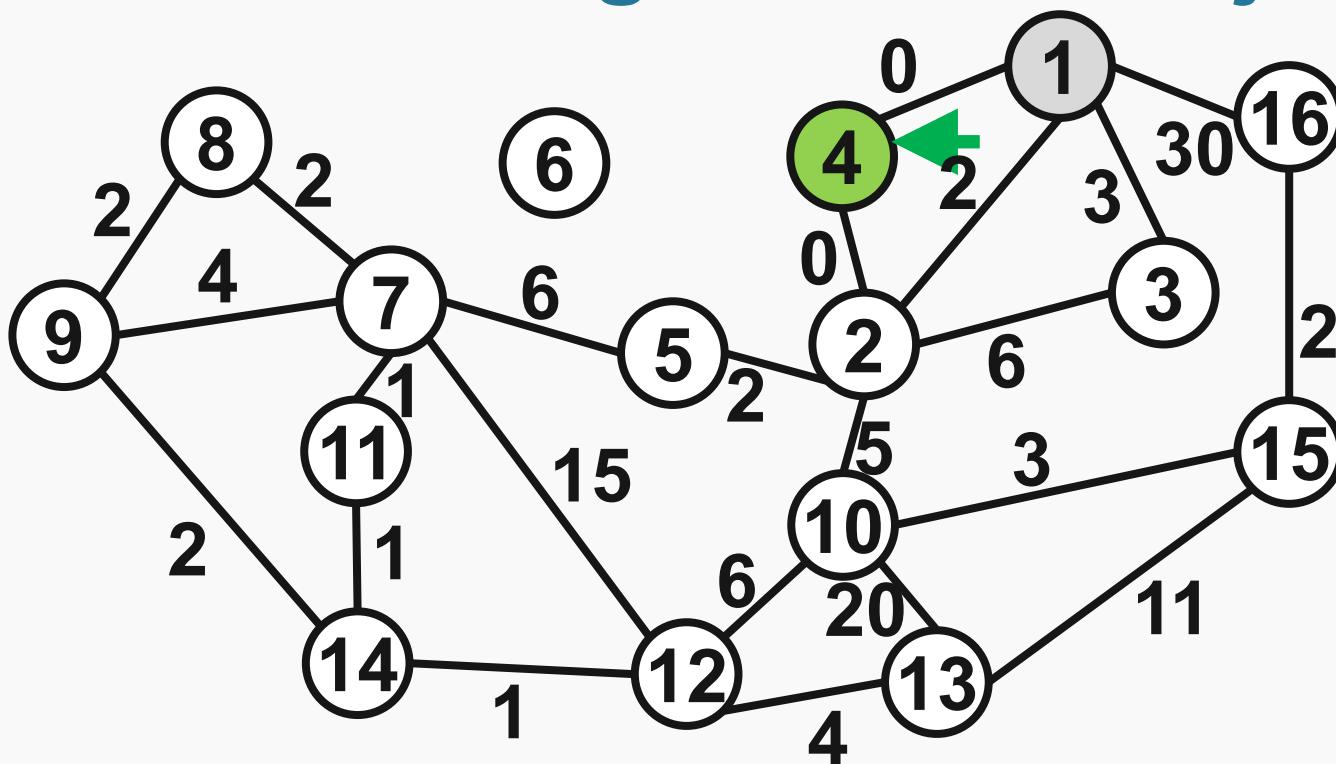


4	2	3	16	5	6	7	8	9	10	11	12	13	14	15
0	2	3	30	∞										

#	D	π
1	0	-
2	2	1
3	3	1
4	0	1
5	∞	-
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	∞	-
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

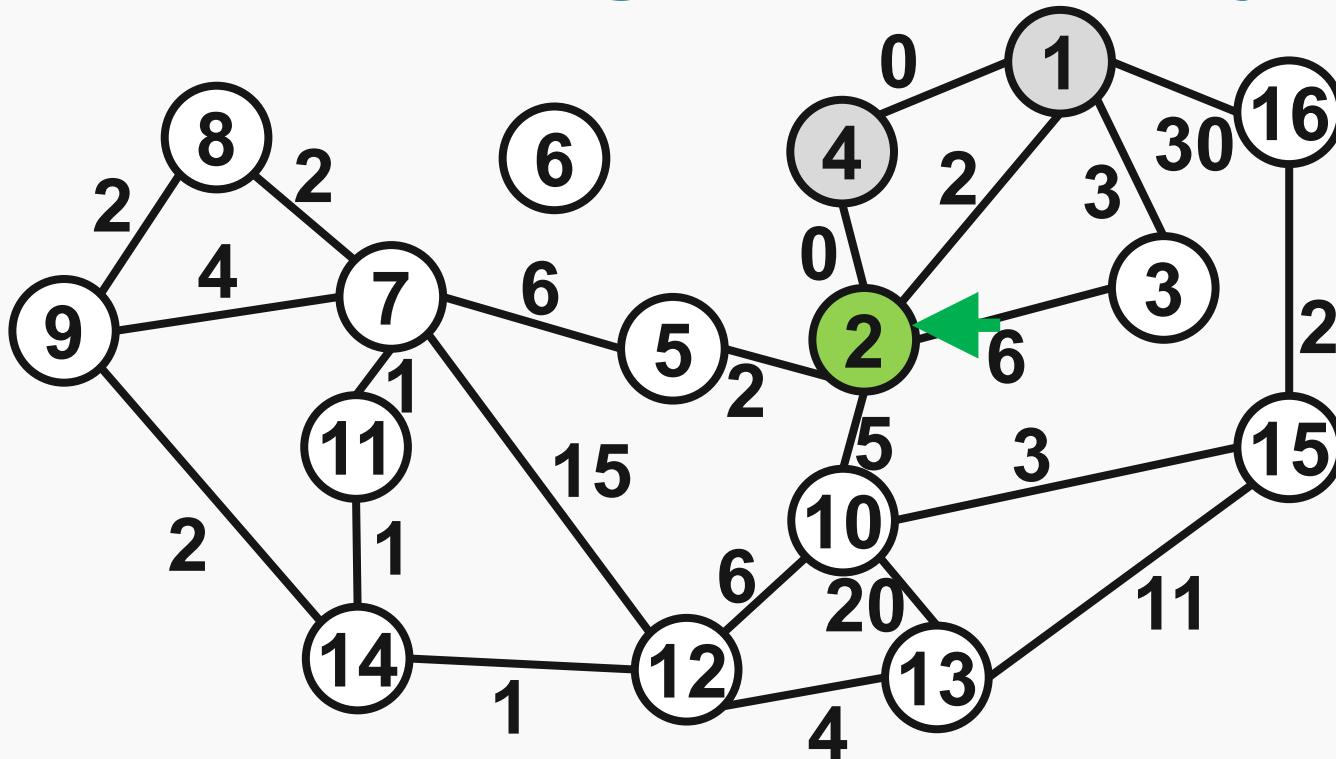


2	3	16	5	6	7	8	9	10	11	12	13	14	15
0	3	30	∞										

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	∞	-
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	∞	-
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

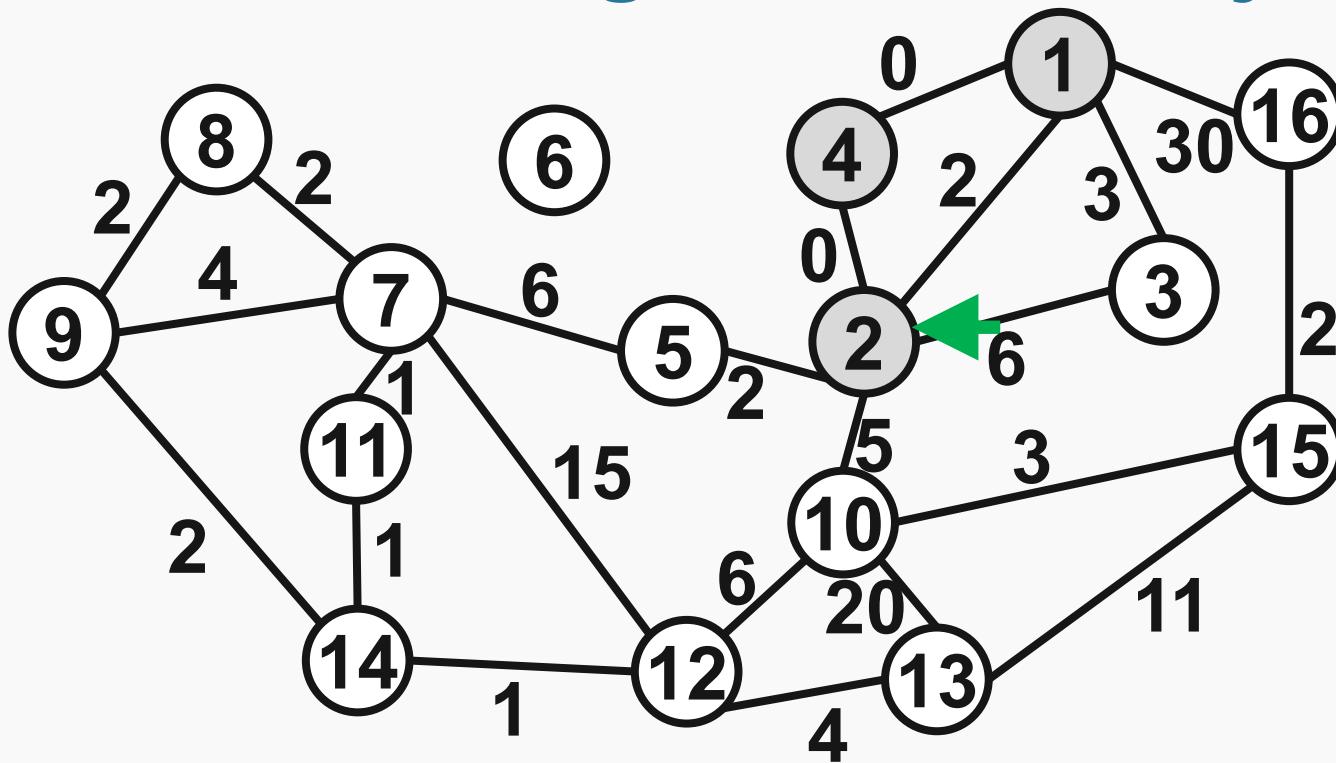


3	16	5	6	7	8	9	10	11	12	13	14	15
3	30	2	∞	∞	∞	∞	5	∞	∞	∞	∞	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

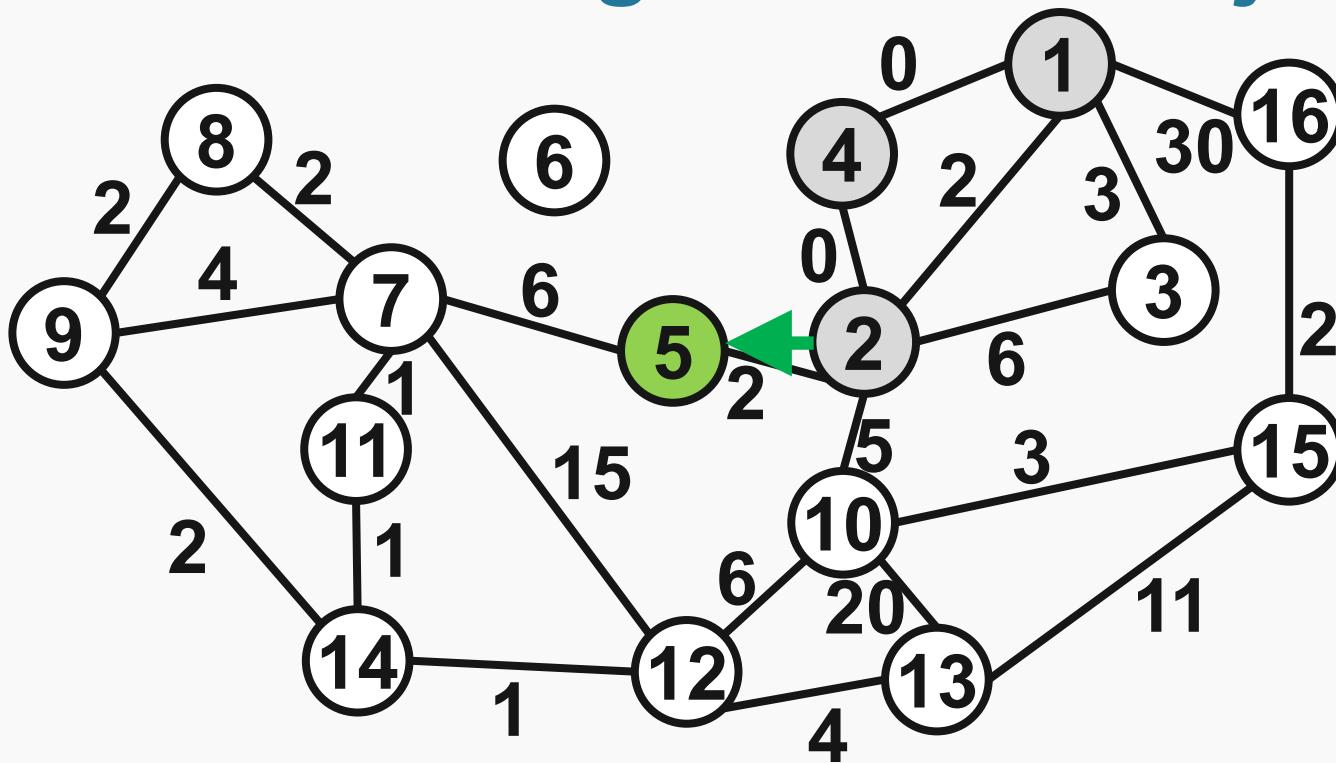


5	3	10	16	6	7	8	9	11	12	13	14	15
2	3	5	30	∞								

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	∞	-
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

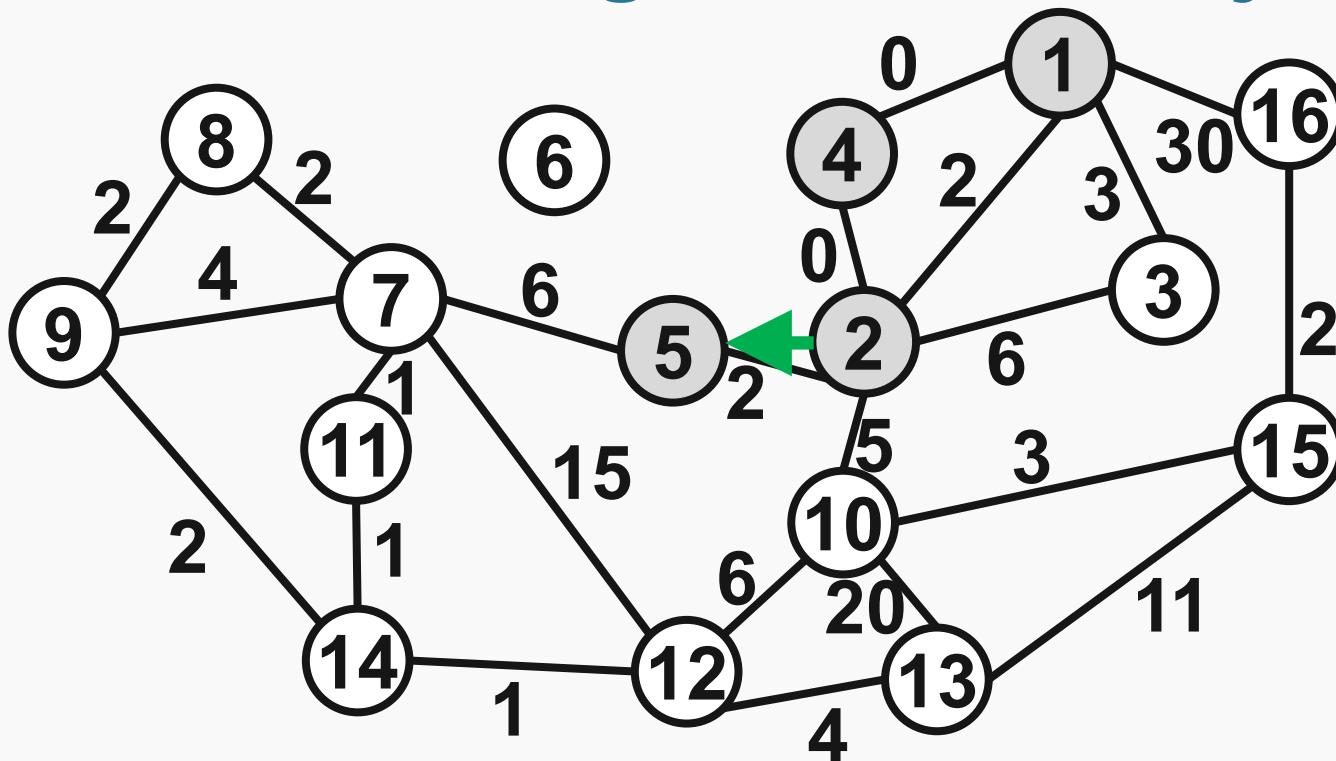


3	10	16	6	7	8	9	11	12	13	14	15
3	5	30	∞	8	∞						

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

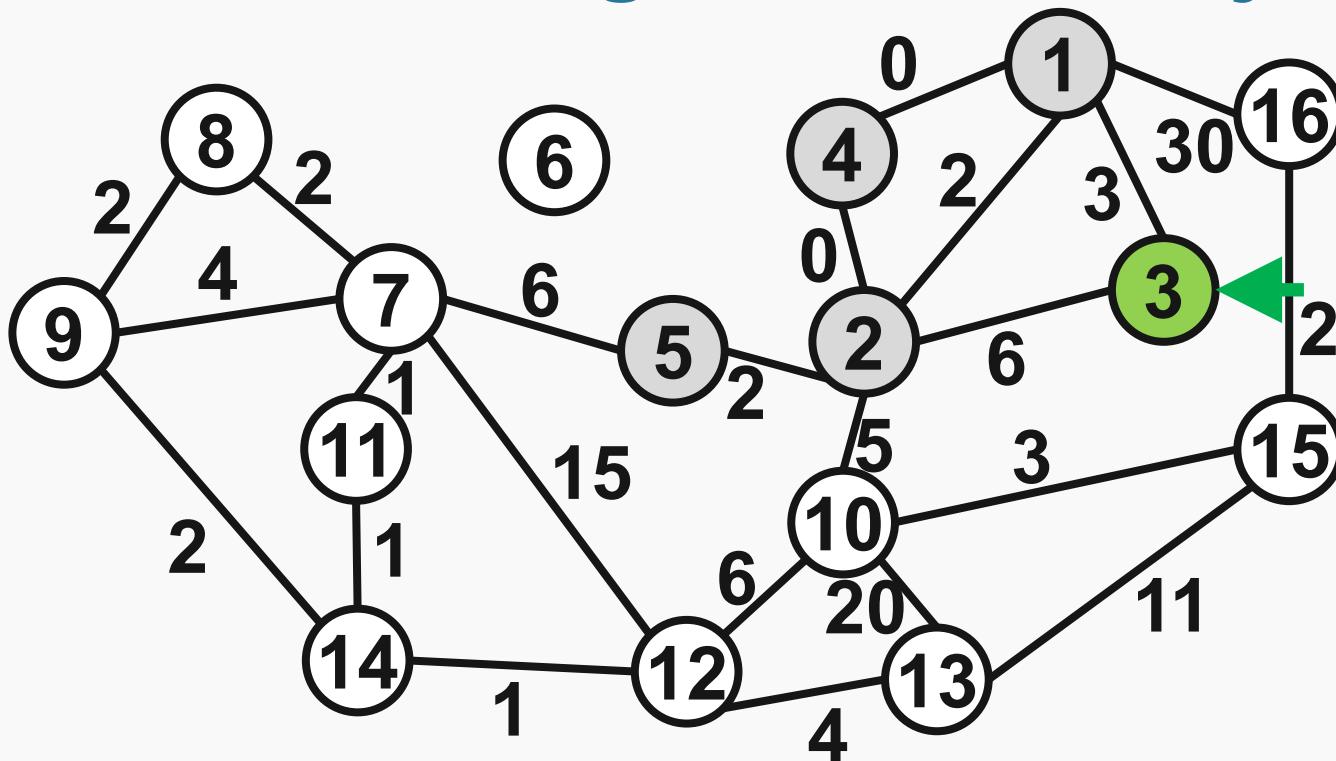


3	10	7	16	6	8	9	11	12	13	14	15
3	5	8	30	∞							

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

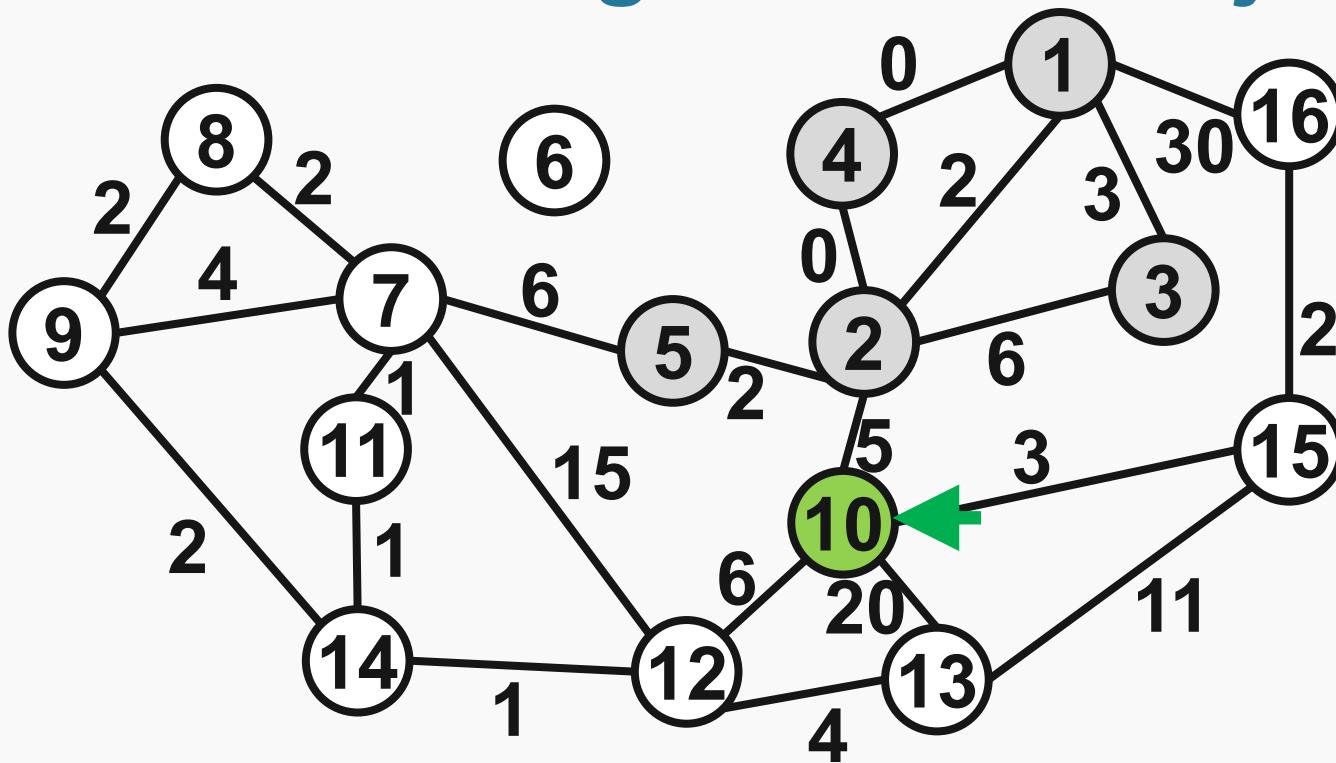


10	7	16	6	8	9	11	12	13	14	15
5	8	30	∞							

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	∞	-
13	∞	-
14	∞	-
15	∞	-
16	30	1



Algoritmul lui Dijkstra

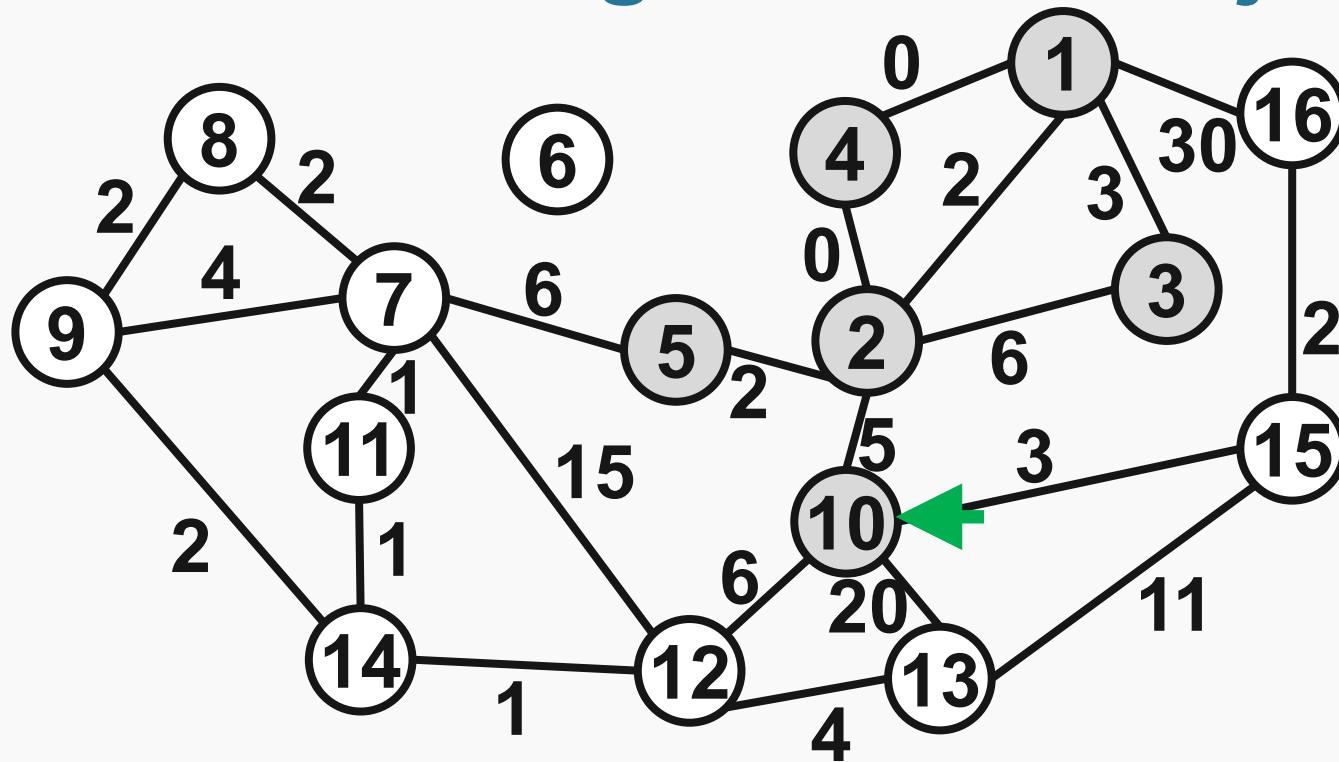


7	16	6	8	9	11	12	13	14	15
8	30	∞	∞	∞	∞	11	25	∞	8

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	11	10
13	25	10
14	∞	-
15	8	10
16	30	1



Algoritmul lui Dijkstra

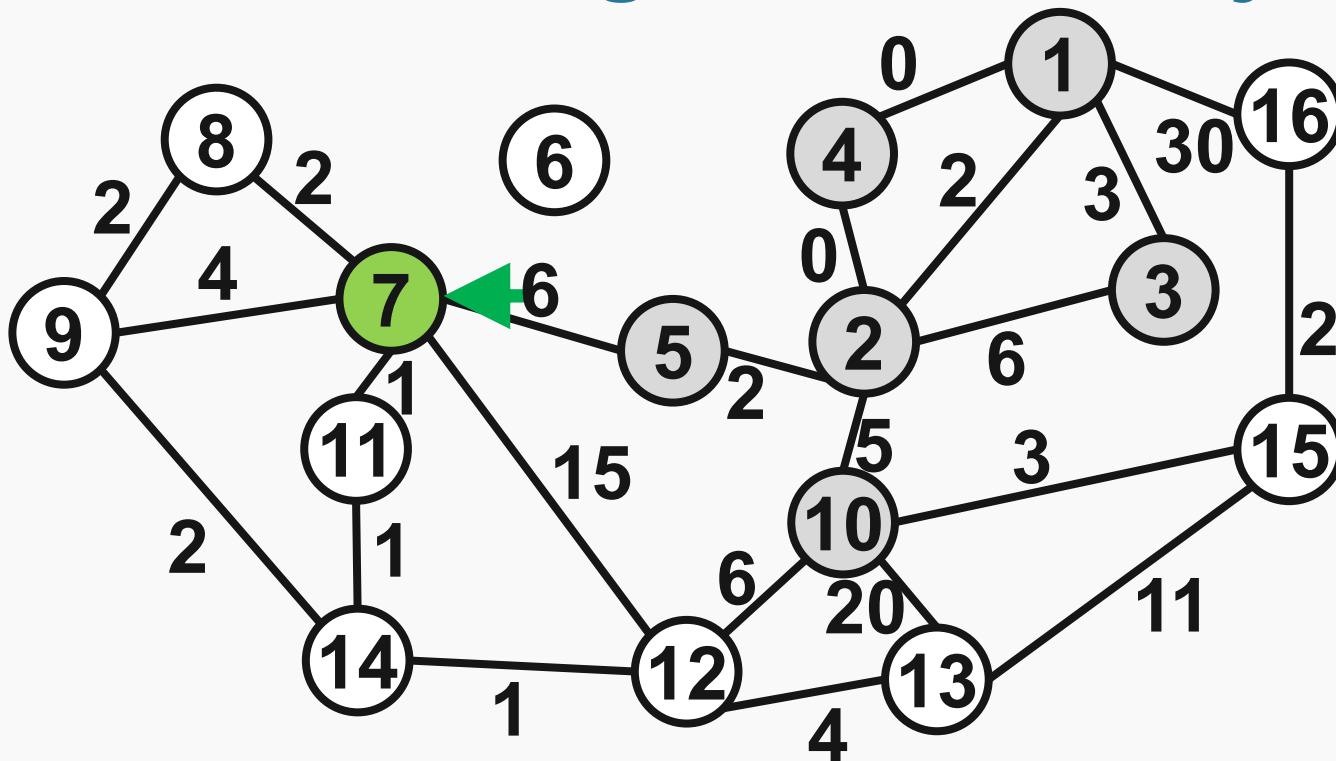


7	15	12	13	16	6	8	9	11	14
8	8	11	25	30	∞	∞	∞	∞	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	∞	-
9	∞	-
10	5	2
11	∞	-
12	11	10
13	25	10
14	∞	-
15	8	10
16	30	1



Algoritmul lui Dijkstra

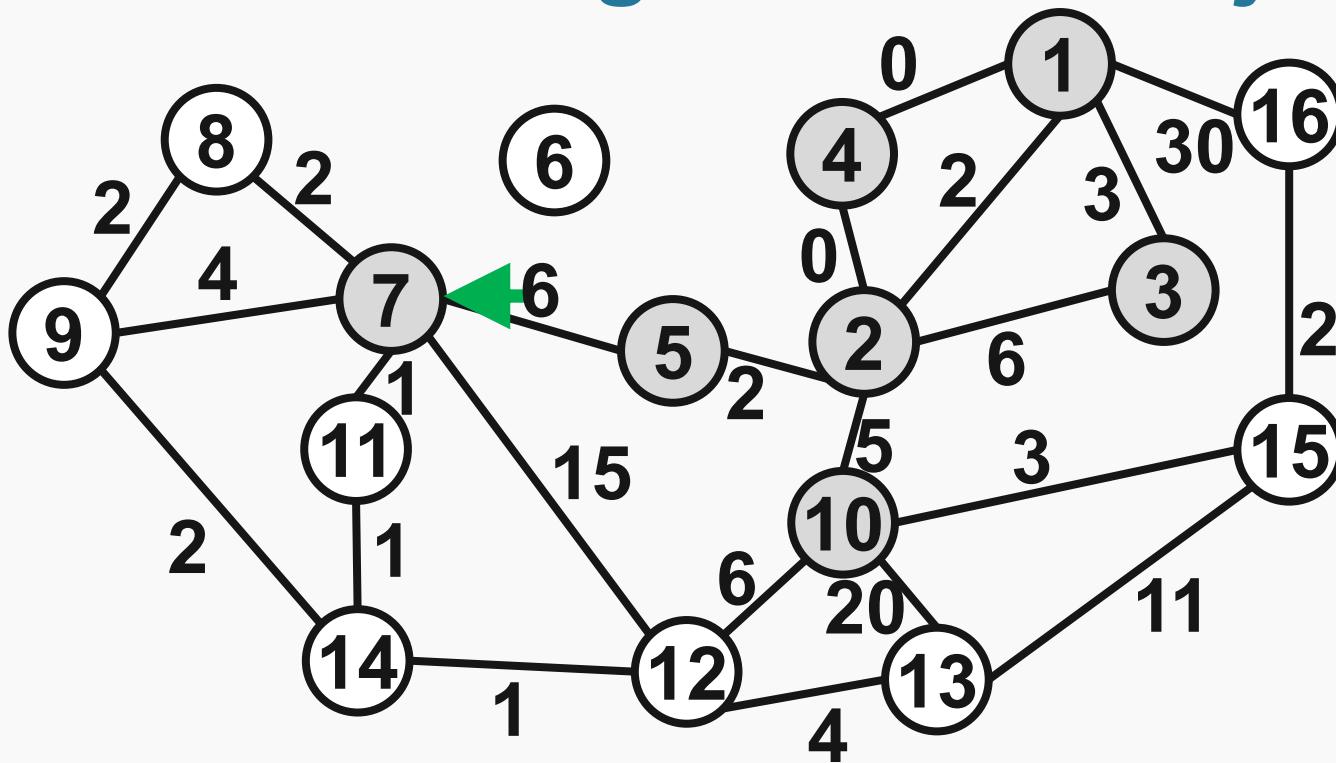


15	12	13	16	6	8	9	11	14
8	11	25	30	∞	10	12	9	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	25	10
14	∞	-
15	8	10
16	30	1



Algoritmul lui Dijkstra

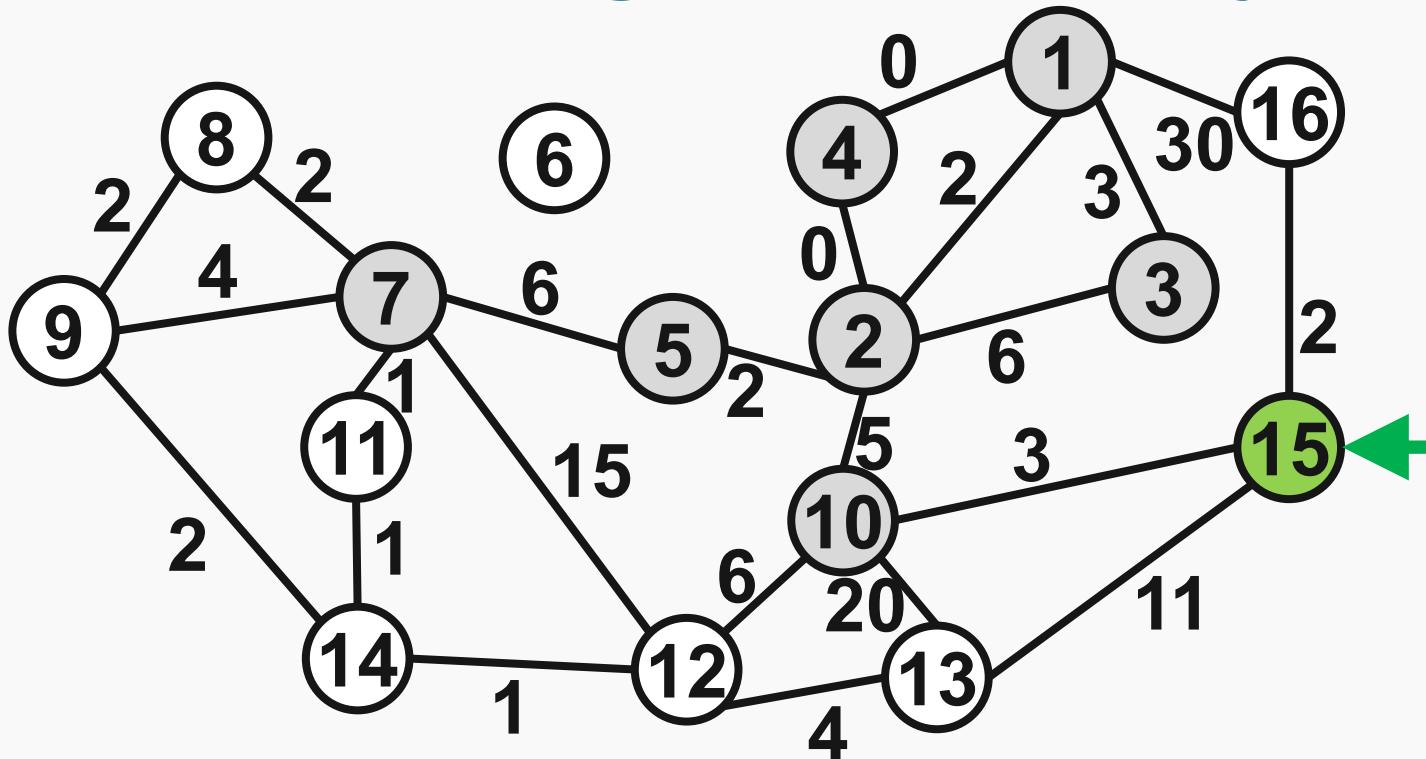


15	11	8	12	9	13	16	6	14
8	9	10	11	12	25	30	∞	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	25	10
14	∞	-
15	8	10
16	30	1



Algoritmul lui Dijkstra

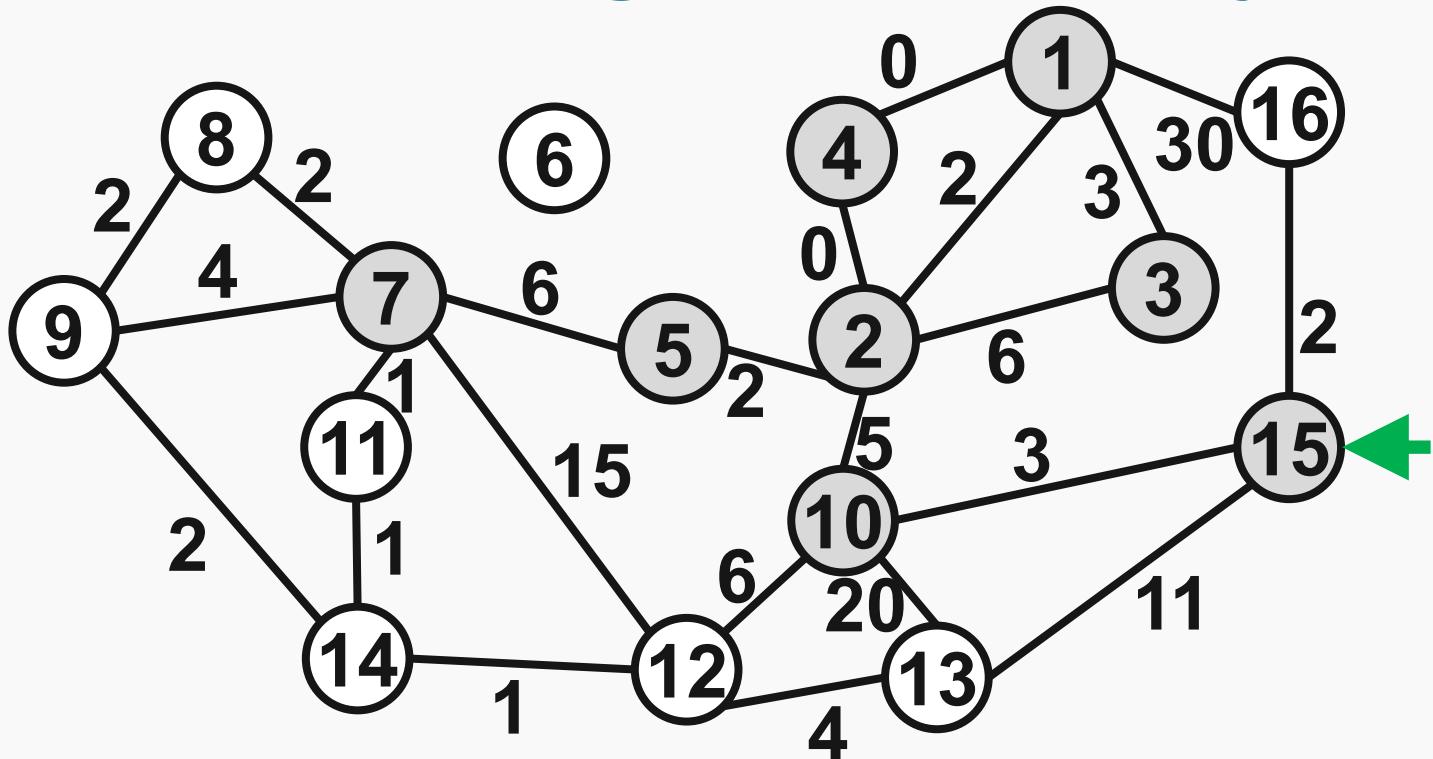


11	8	12	9	13	16	6	14
9	10	11	12	19	10	∞	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	∞	-
15	8	10
16	10	15



Algoritmul lui Dijkstra

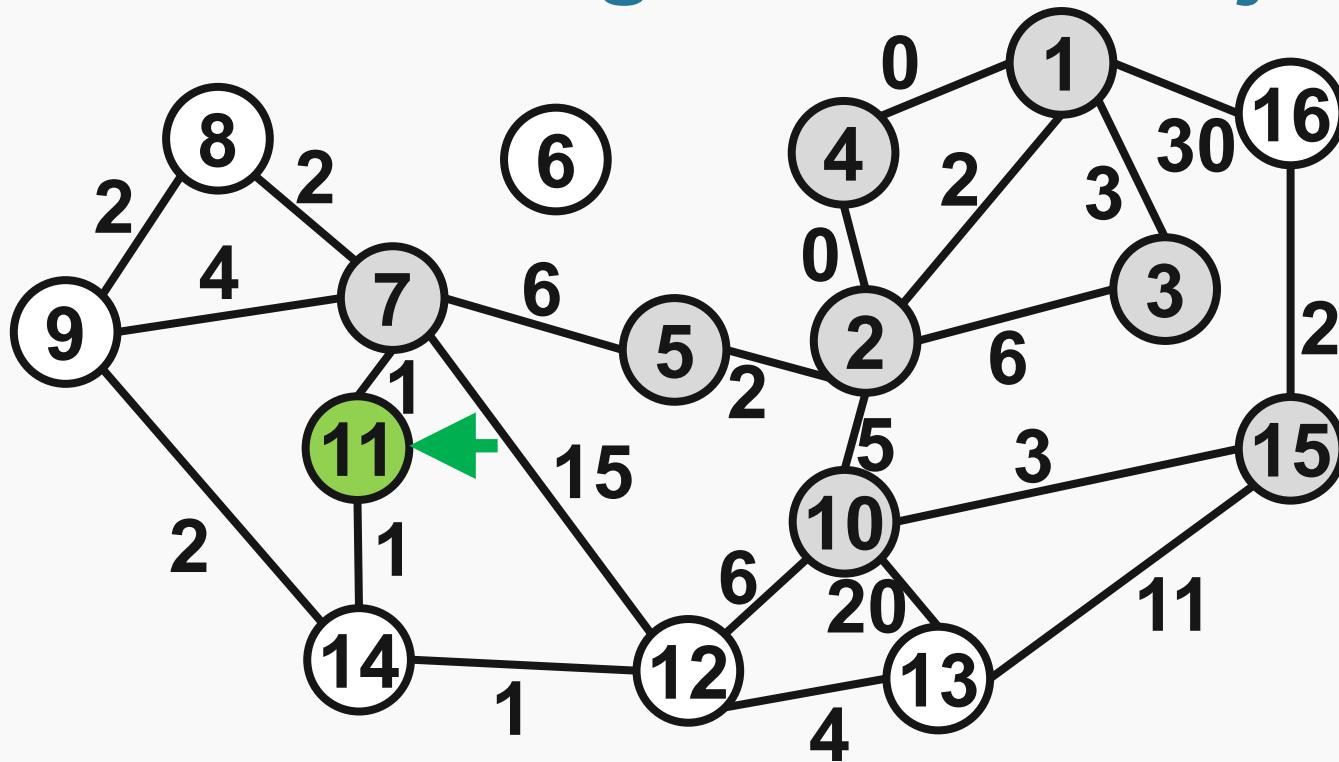


11	8	16	12	9	13	6	14
9	10	10	11	12	19	∞	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	∞	-
15	8	10
16	10	15



Algoritmul lui Dijkstra

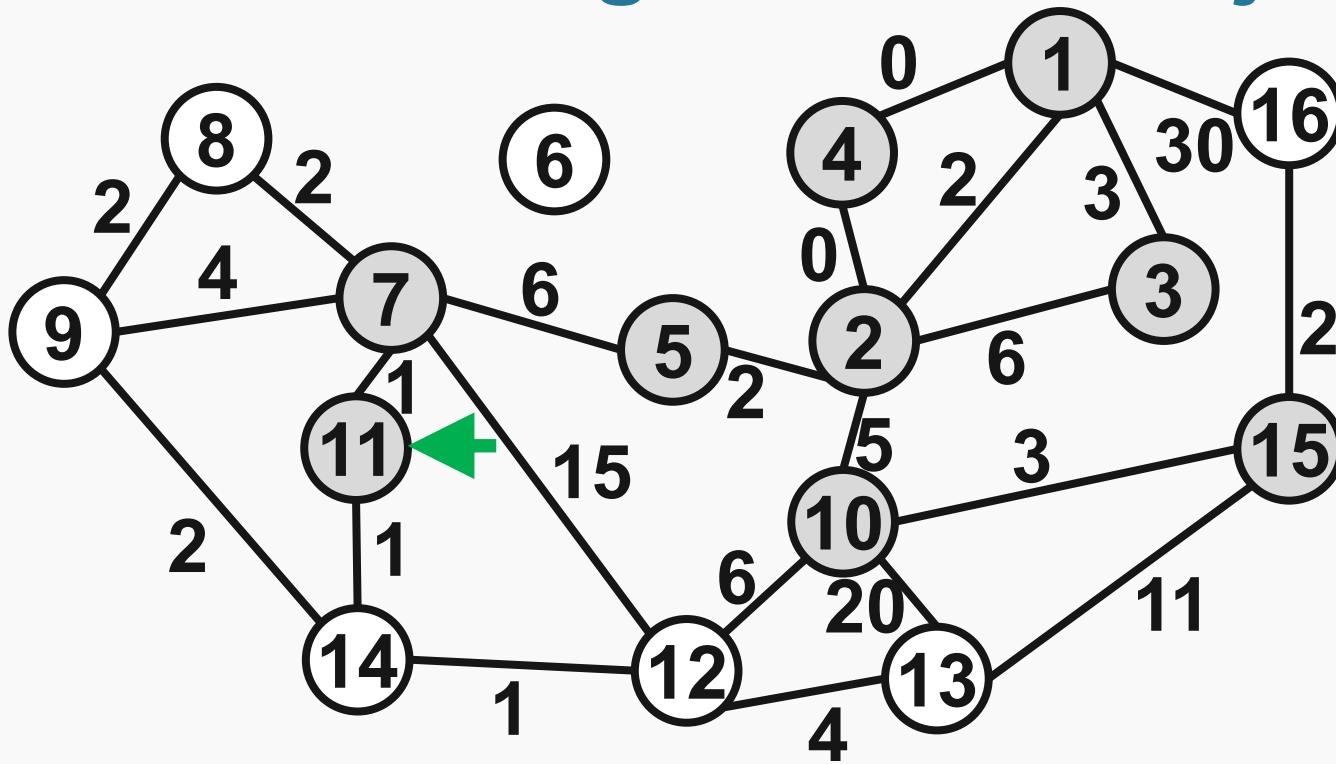


8	16	12	9	13	6	14
10	10	11	12	19	∞	10

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

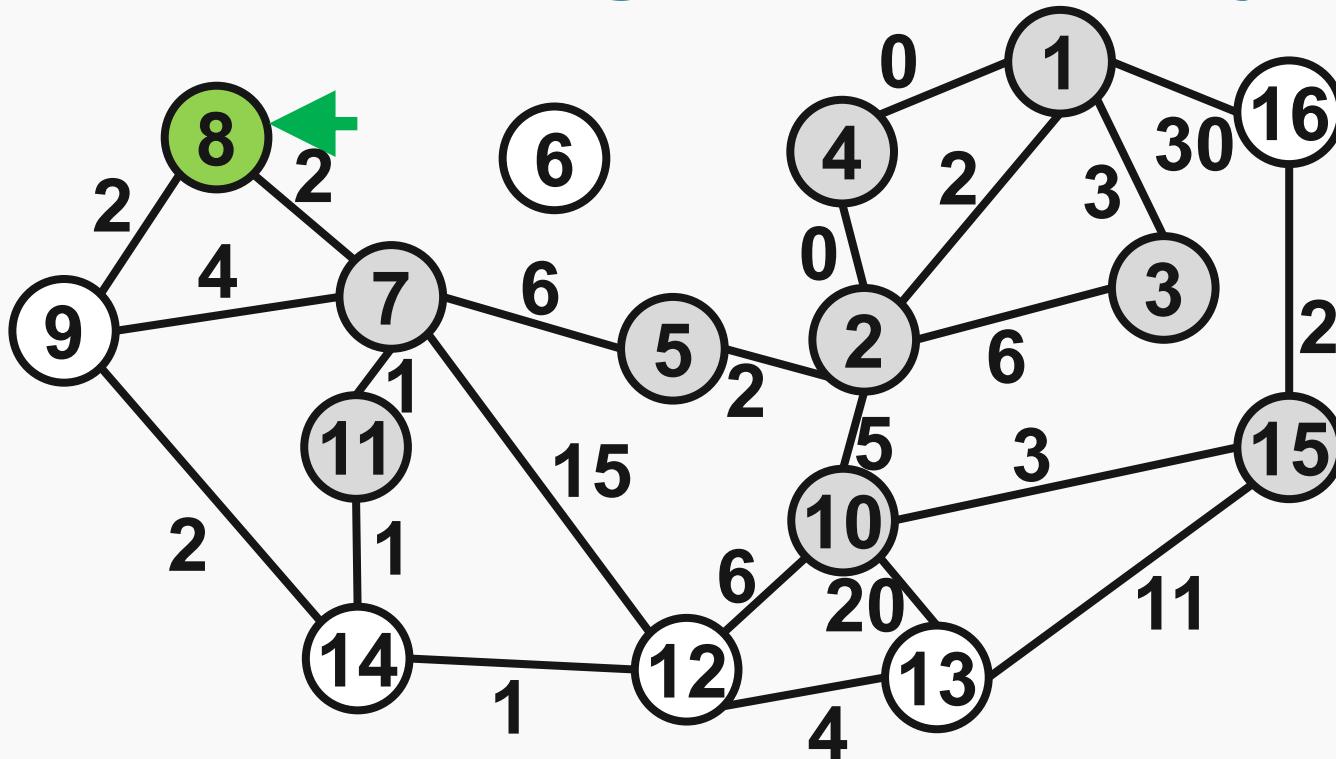


8	16	14	12	9	13	6
10	10	10	11	12	19	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

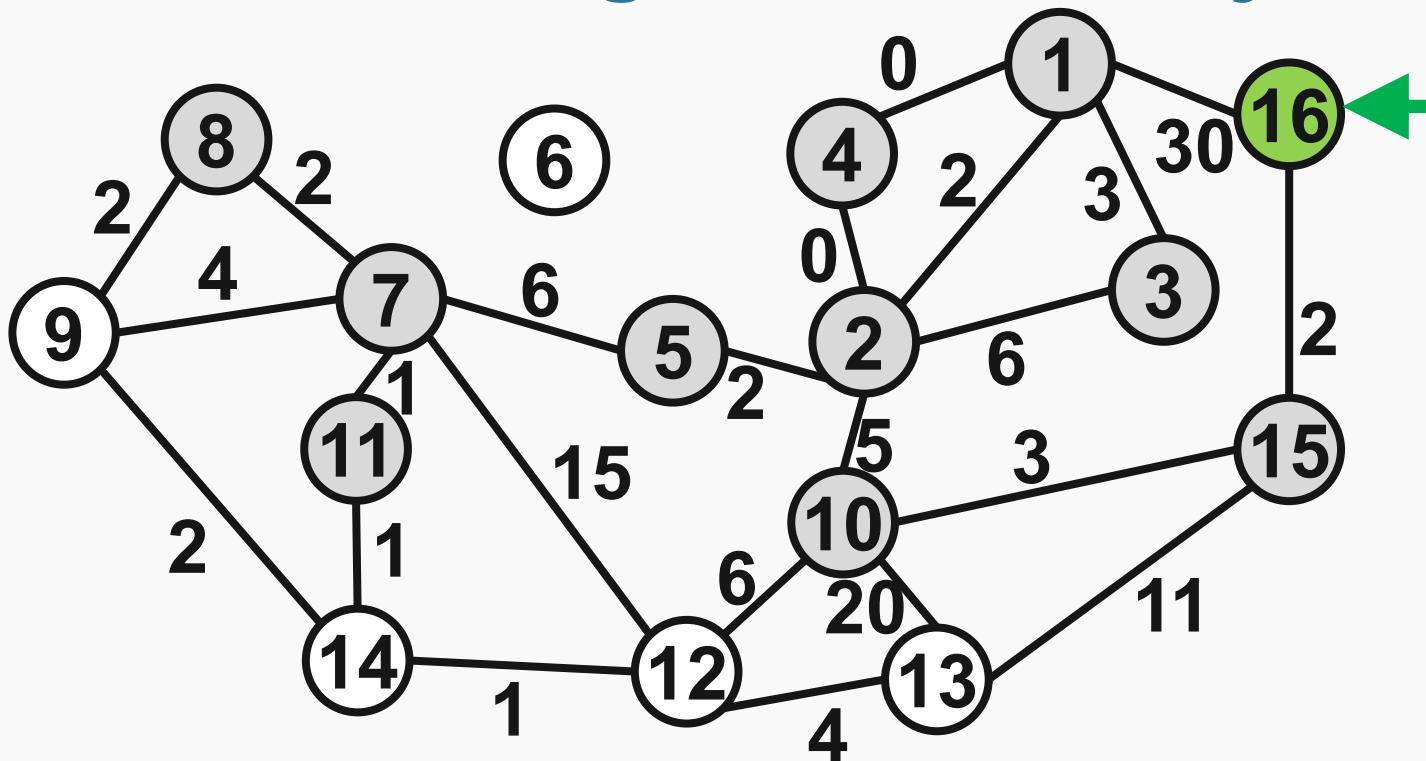


16	14	12	9	13	6
10	10	11	12	19	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

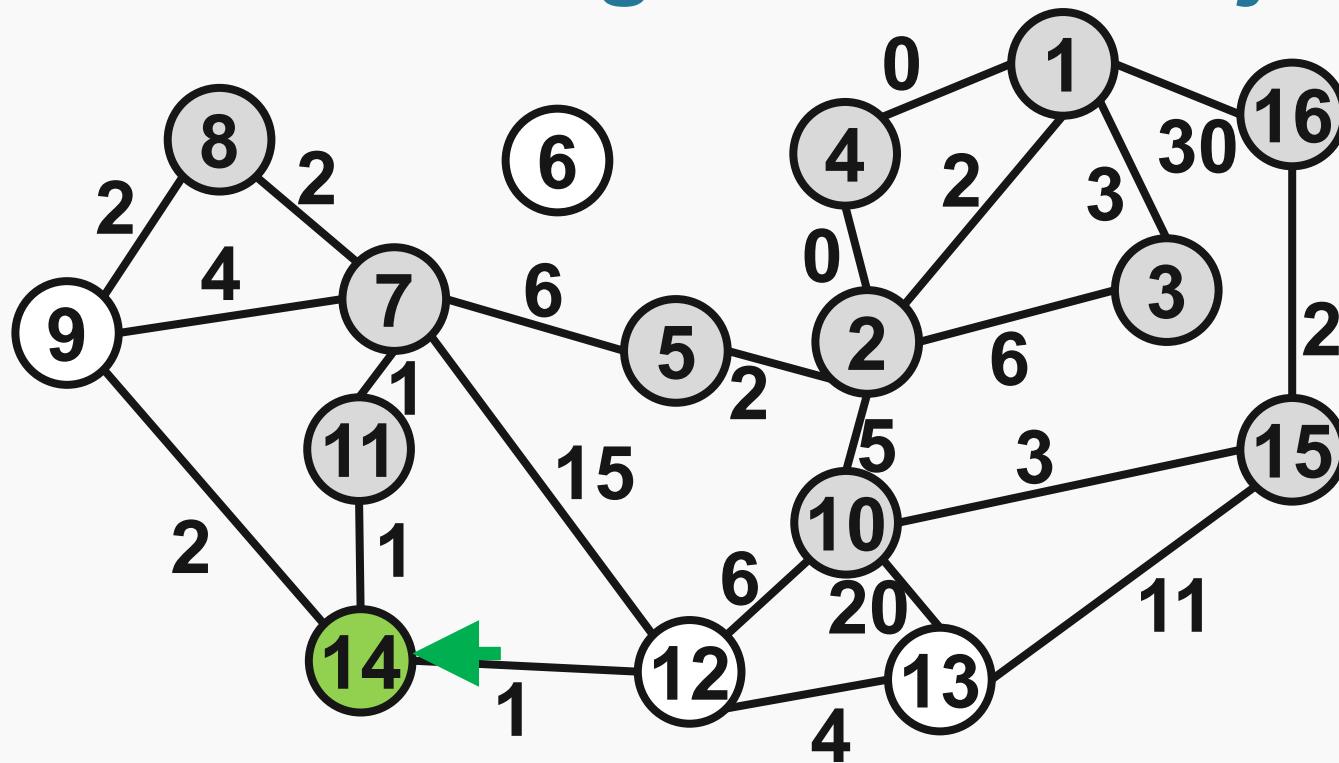


14	12	9	13	6
10	11	12	19	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

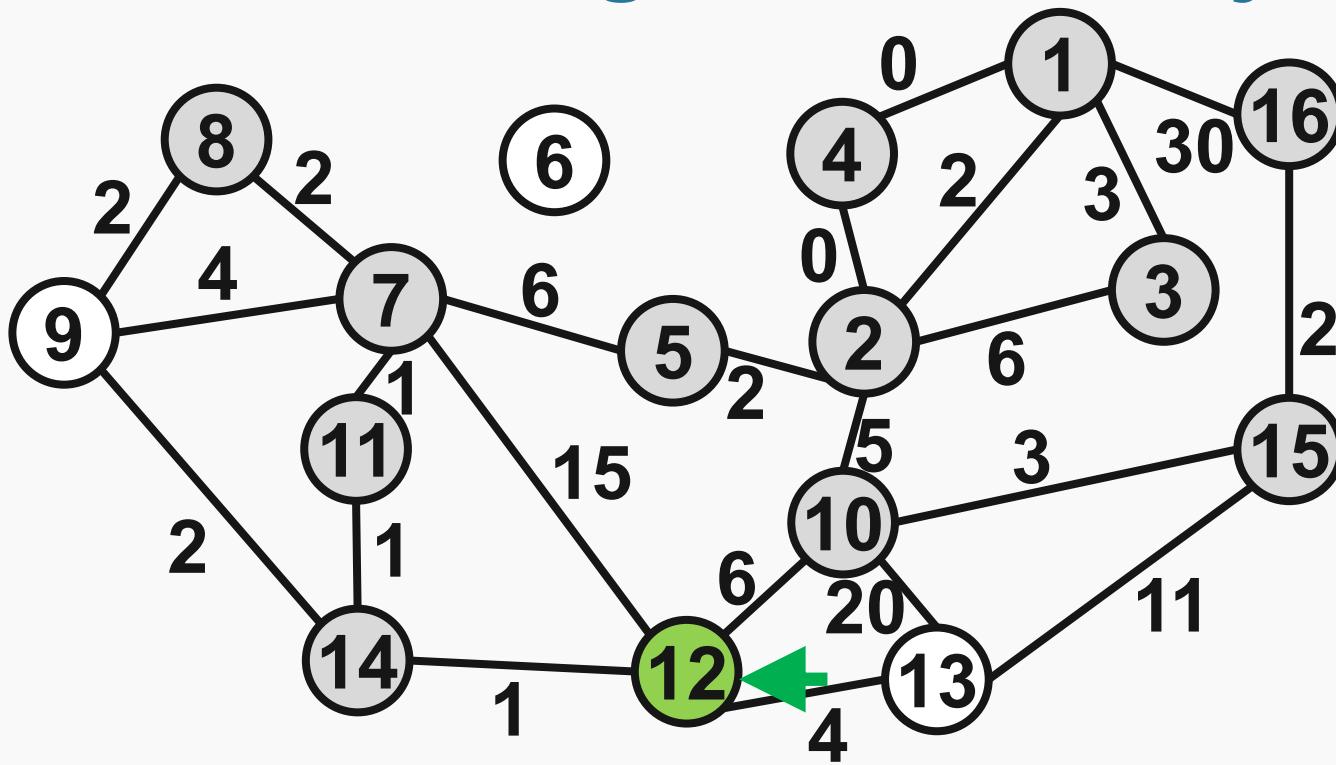


12	9	13	6
11	12	19	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	19	15
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

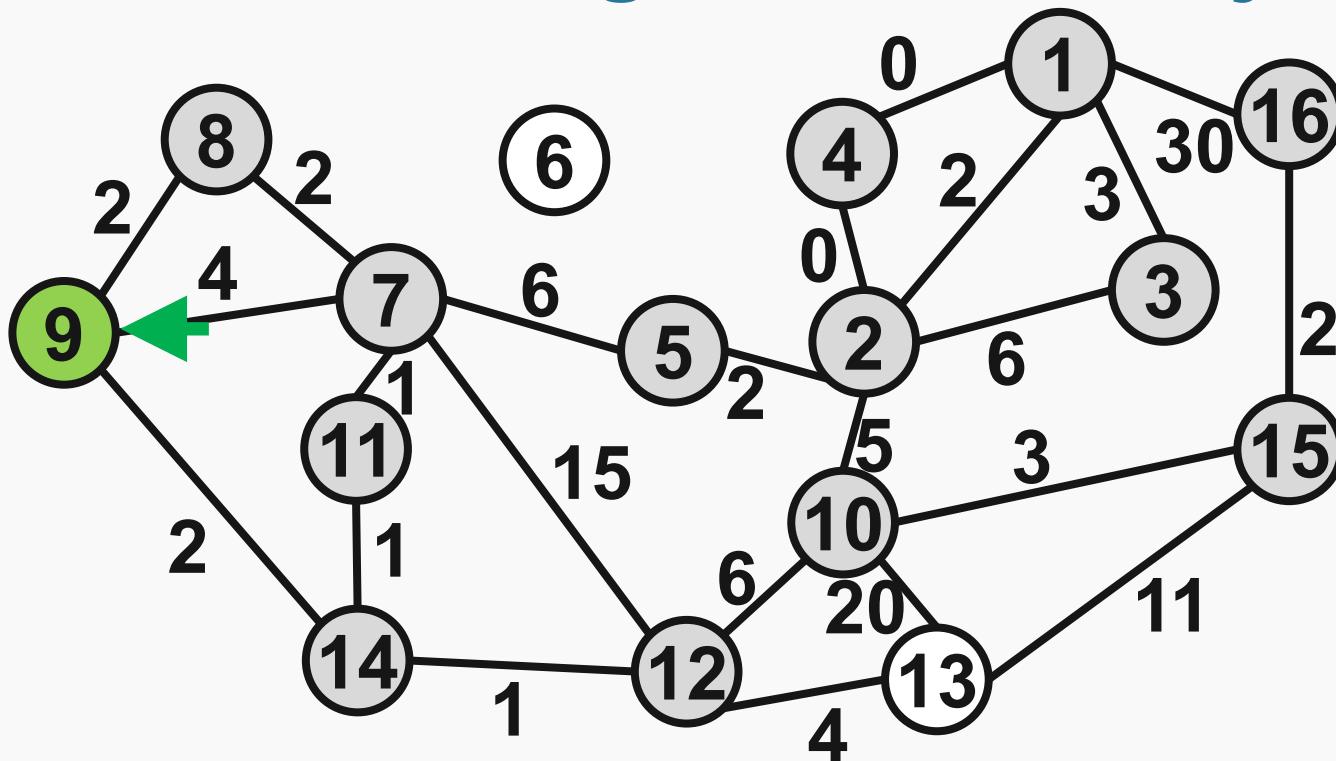


9	13	6
12	15	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	15	12
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

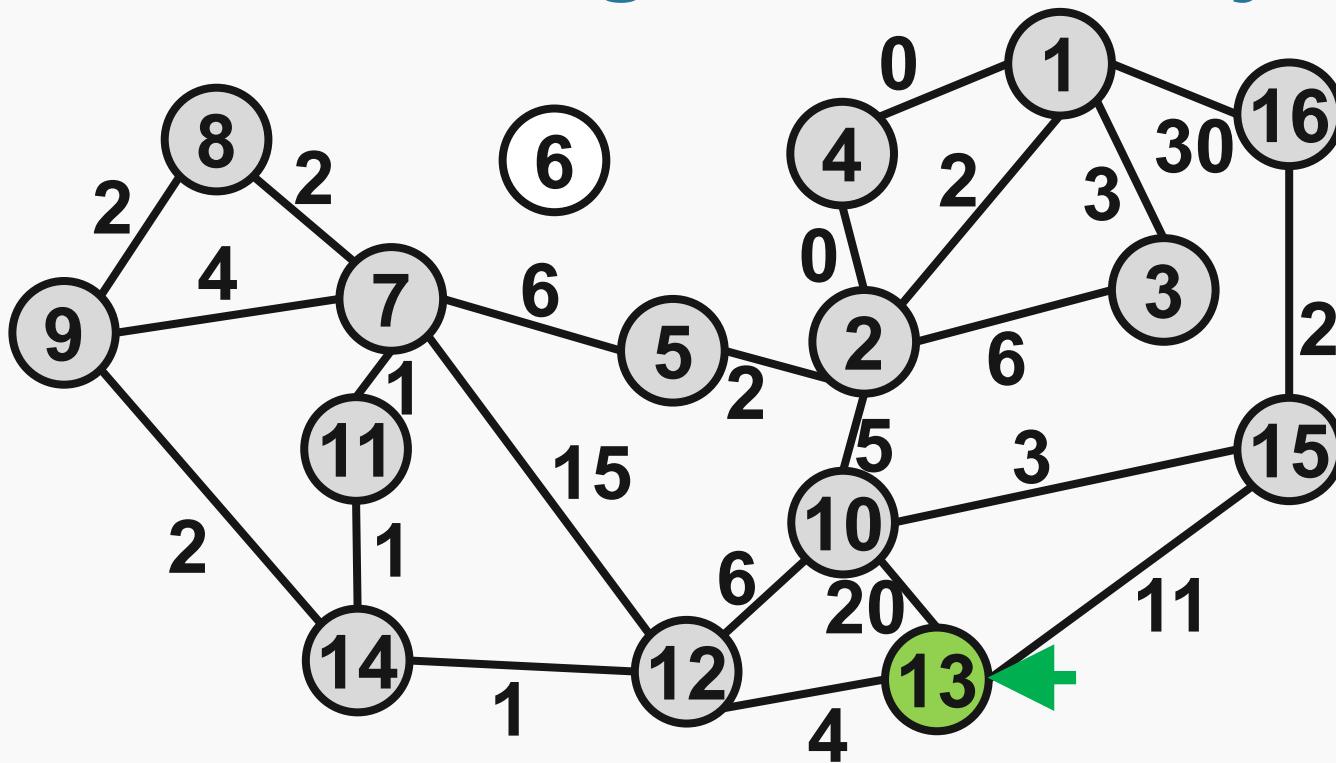


13	6
15	∞

#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	15	12
14	10	11
15	8	10
16	10	15



Algoritmul lui Dijkstra

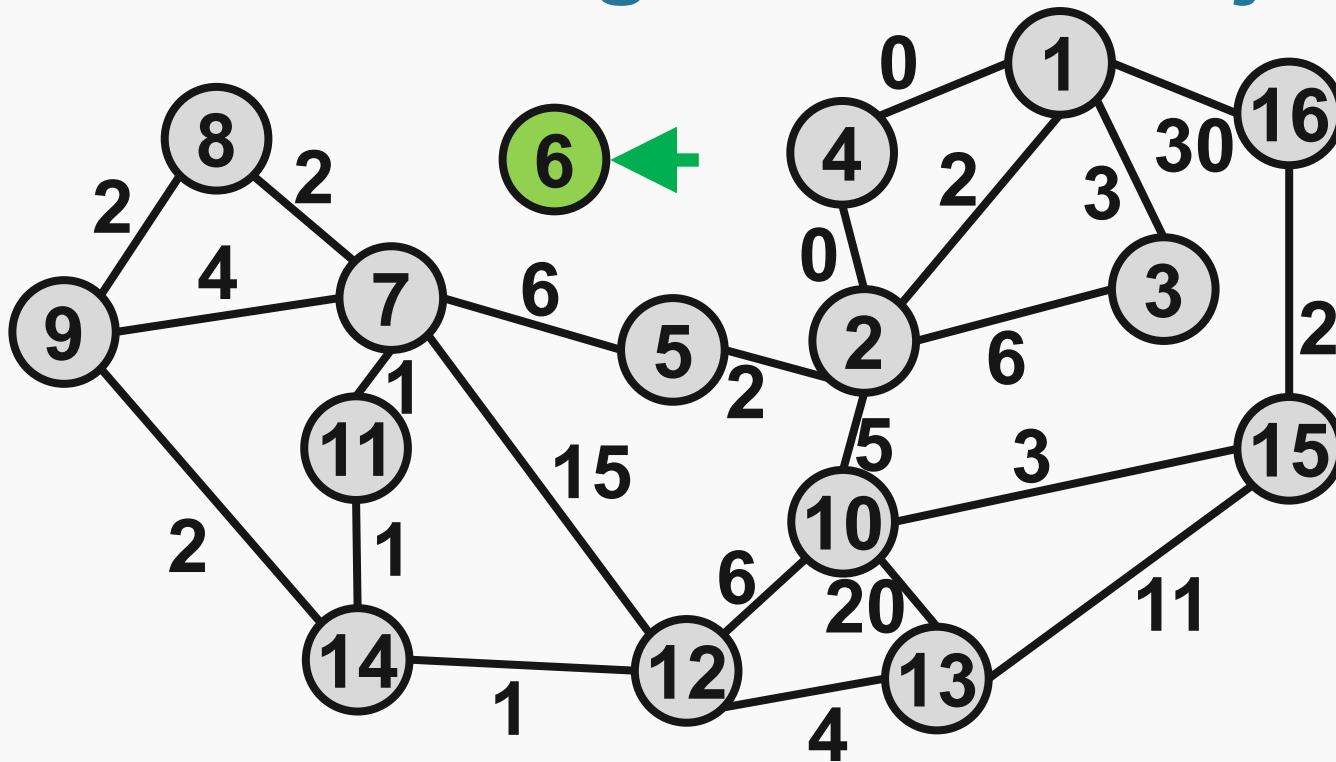


6
∞

#	d	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	15	12
14	10	11
15	8	10
16	10	15



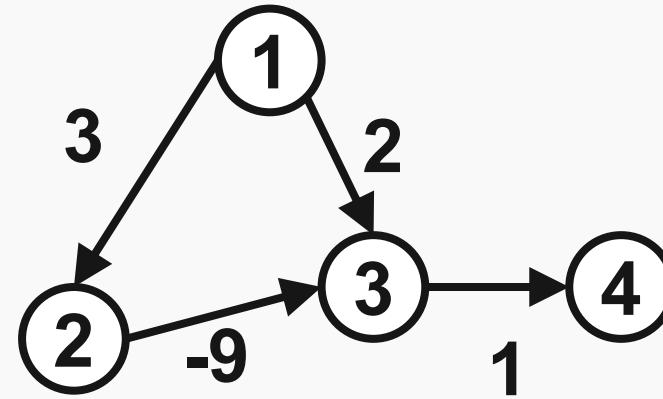
Algoritmul lui Dijkstra



#	D	π
1	0	-
2	0	4
3	3	1
4	0	1
5	2	2
6	∞	-
7	8	5
8	10	7
9	12	7
10	5	2
11	9	7
12	11	10
13	15	12
14	10	11
15	8	10
16	10	15

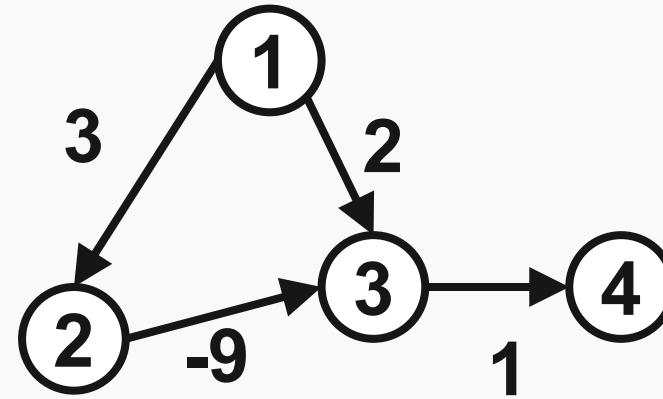


Algoritmul lui Dijkstra. De ce nu muchii negative?



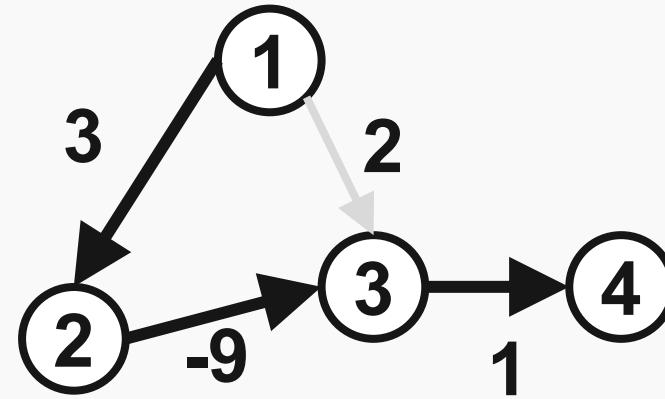


Algoritmul lui Dijkstra. De ce nu muchii negative? Care este drum minim 1->4 ?





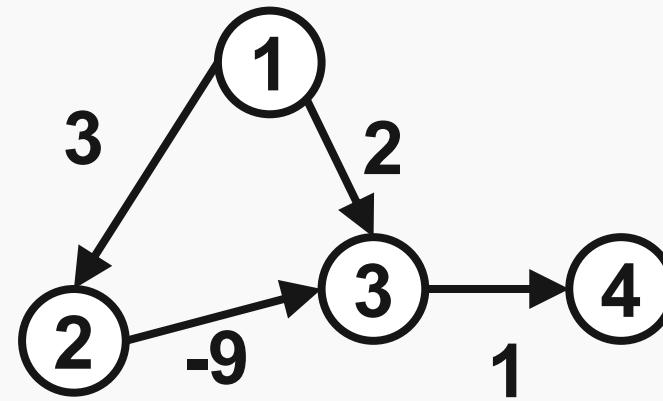
Algoritmul lui Dijkstra. De ce nu muchii negative? Care este drum minim 1->4 ?



(1,2,3,4) cu costul $3+(-9)+1 = -5$



Algoritmul lui Dijkstra. De ce nu muchii negative?

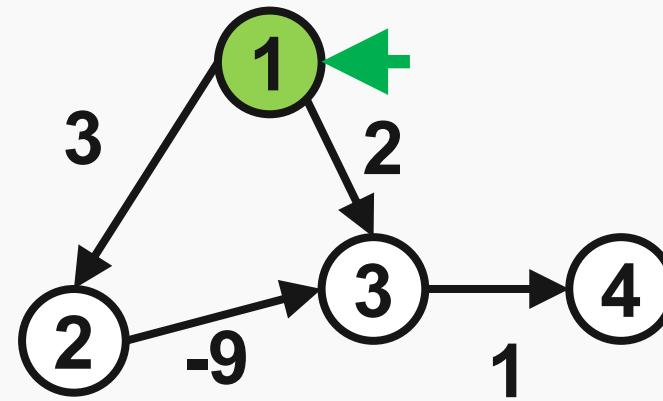


#	D	π
1	0	-
2	∞	-
3	∞	-
4	∞	-

1	2	3	4
0	∞	∞	∞



Algoritmul lui Dijkstra. De ce nu muchii negative?

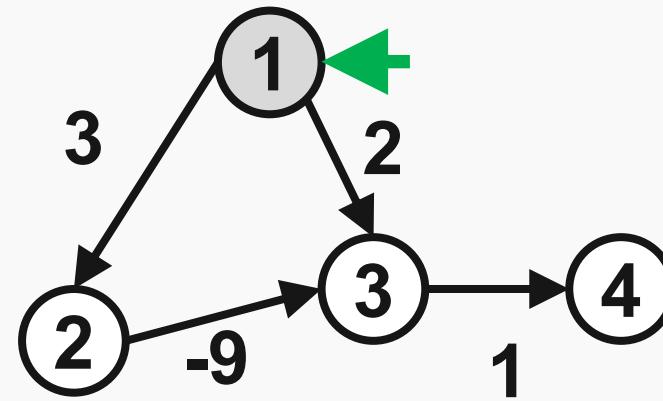


#	D	π
1	0	-
2	3	1
3	2	1
4	∞	-

2	3	4
3	2	∞



Algoritmul lui Dijkstra. De ce nu muchii negative?

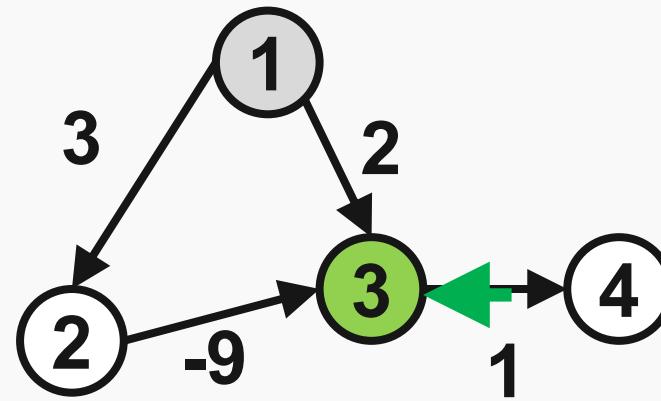


#	D	π
1	0	-
2	3	1
3	2	1
4	∞	-

3	2	4
2	3	∞



Algoritmul lui Dijkstra. De ce nu muchii negative?

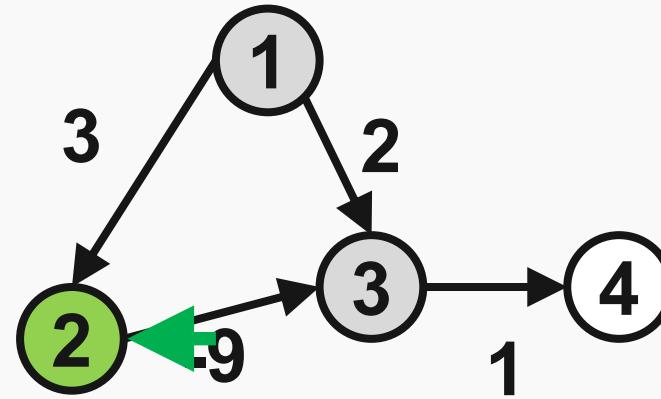


2	4
3	3

#	D	π
1	0	-
2	3	1
3	2	1
4	3	3



Algoritmul lui Dijkstra. De ce nu muchii negative?

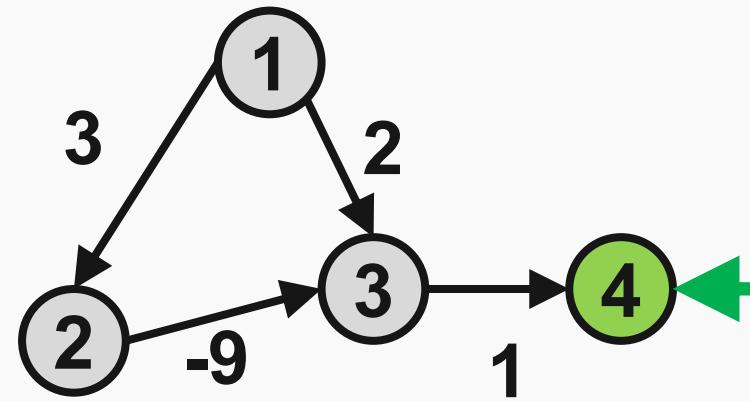


4
3

#	D	π
1	0	-
2	3	1
3	-6	2
4	3	3



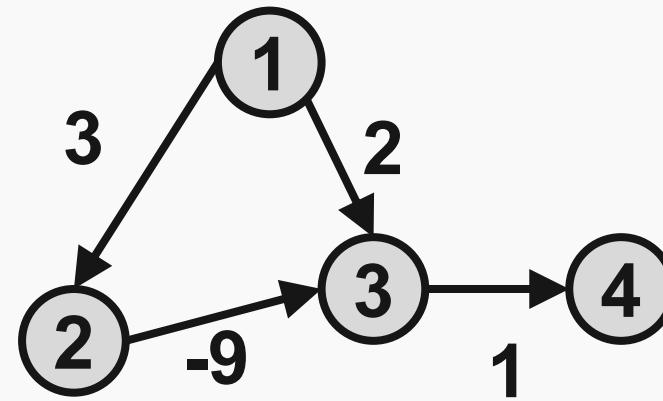
Algoritmul lui Dijkstra. De ce nu muchii negative?



#	D	π
1	0	-
2	3	1
3	-6	2
4	3	3



Algoritmul lui Dijkstra. De ce nu muchii negative?



#	D	π
1	0	-
2	3	1
3	-6	2
4	3	3

Dar știm $(1,2,3,4)$ cu costul $3 + (-9) + 1 = -5$



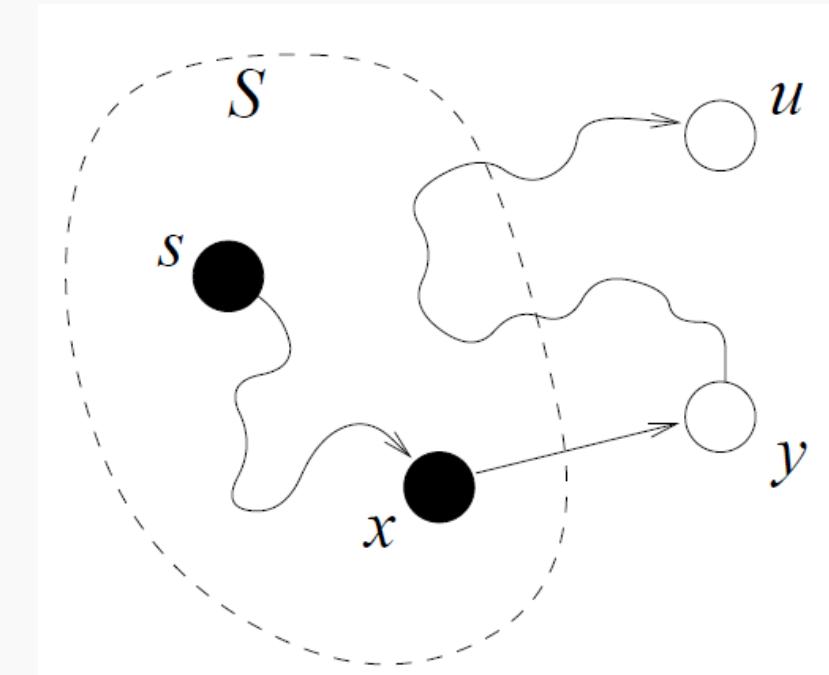
Algoritmul Dijkstra Demonstrație corectitudine

- Teoremă: Pentru un graf cu muchii pozitive la final vom avea $v.d = \delta(s, v) \forall v \in V$
- Demonstrație:
 - Invariant de buclă: La începutul fiecărei iterații nodurile extrase au $v.d = \delta(s, v)$.
 - Este suficient să arătăm că la extragere $v.d = \delta(s, v)$, toate momentele de după sunt confirmate de limita superioară a estimării.
- Inițial: nu este nici un nod extras, invariantul este corect.



Algoritmul Dijkstra Demonstrație corectitudine

- La fiecare pas:
 - Prin contradicție fie $u.d \neq \delta(s, u)$ primul adăugat la S
 - Fie $s \rightsquigarrow x \rightarrow y \rightsquigarrow u; x \in S; y \in V - S$
 - $y.d = \delta(s, y)$ deoarece u primul
 - $y.d = \delta(s, y) \leq \delta(s, u) \leq u.d$
 - Dar $u.d \leq y.d$ deoarece u ales înainte de y
 $\Rightarrow y.d = \delta(s, y) = \delta(s, u) = u.d$
- Terminare: Toate nodurile sunt extrase, deci $v.d = \delta(s, v) \forall v \in V$.





```
dijkstraAlgorithm(G, source) {
    int d[|G.V|] = { INFINITY };
    d[source] = 0;
    for each(node in G.V)      //O(|v|)
        push(LIST, node, d);   //Tinsert

    while (!isEmpty(LIST)) {    //O(|v|)
        node = pop_MIN(LIST, d); //Tmin
        for each (neighbor of node) { //O(|Lu|)
            if (d[node] + w(node, neighbor) < d[neighbor]) {
                d[neighbor] = d[node] + w(node, neighbor);
                prev[neighbor] = node;
                update(LIST, d); //Tacces
            }
        }
    }
    return dist, prev;
}
```



Complexitatea Generală Dijkstra

$$T(V, E) = \theta(VT_{insert}) + \sum_{u \in V} [T_{min} + L_u T_{acces}]$$

$$T(V, E) = \theta(VT_{insert}) + \theta(VT_{min}) + T_{acces} \sum_{u \in V} L_u$$

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$



Complexitatea Dijkstra LISTA ca vector

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$

- $T_{insert} = \theta(1)$
- $T_{min} = \theta(V)$
- $T_{acces} = \theta(1)$

$$T(V, E) = \theta(V^2 + E)$$



Complexitatea Dijkstra LISTA ca listă înlățuită

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$

- $T_{insert} = \theta(1)$
- $T_{min} = \theta(V)$
- $T_{acces} = \theta(V)$

$$E \geq V - 1$$

pentru graf

connex

$$T(V, E) = \theta(V^2 + EV) = \theta(EV)$$



Complexitatea Dijkstra LISTA ca o coadă priorități implementată cu vector

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$

- $T_{insert} = \theta(\ln V)$
- $T_{min} = \theta(1)$
- $T_{acces} = \theta(\ln V)$

$$T(V, E) = \theta((E + V)\ln V)$$



Complexitatea Dijkstra LISTA ca un arbore binar

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$

- $T_{insert} = \theta(\ln V)$
- $T_{min} = \theta(\ln V)$
- $T_{acces} = \theta(\ln V)$

$$T(V, E) = \theta((E + V)\ln V)$$



Complexitatea Dijkstra arbore binar vs vector

$$\theta((E + V) \ln V)$$

$$\theta(V^2 + E)$$

Graf obișnuit : $E \approx kV$;

Graf complet : $E = V(V - 1)/2$



Complexitatea Dijkstra LISTA ca un heap fibonacci

$$T(V, E) = \theta[V(T_{insert} + T_{min})] + \theta(ET_{acces})$$

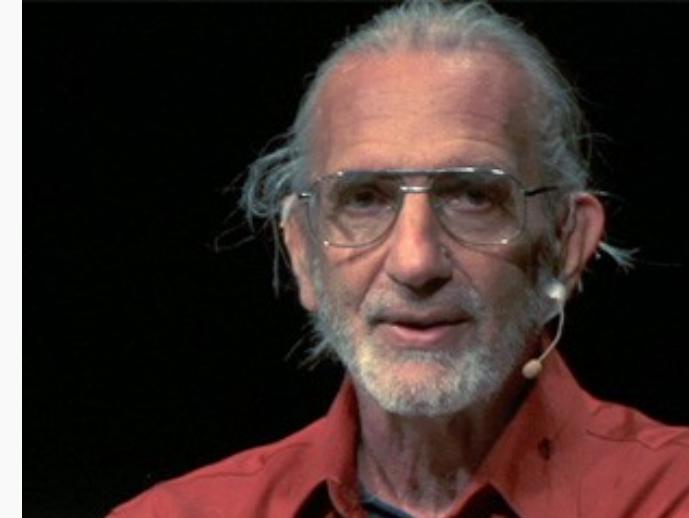
- $T_{insert} = \theta(1)$
- $T_{min} = \theta(1 + \ln V)$
- $T_{acces} = \theta(1)$
- Heap Binar : arbore binar unde *tatal* \leq *ambii copii*
- Heap Fibonacci : colectie specială de heapuri binare

$$T(V, E) = \theta(E + V \ln V)$$



Algoritmul Floyd-Warshall (Roy)

- Găsește drum minim între oricare 2 noduri





Algoritmul Floyd-Warshall (Roy)

```
floydWarshallAlgorithm(G) {  
    for (k = 0; k < |G.V| ; k++)  
        for (i = 0; i < |G.V| ; i++)  
            for (j = 0; j < |G.V| ; j++)  
                if (dist[i][j] > dist[i][k] + dist[k][j])  
                    dist[i][j] = dist[i][k] + dist[k][j];  
}
```



Complexitate?

```
floydWarshallAlgorithm(G) {  
    for (k = 0; k < |G.V| ; k++)  
        for (i = 0; i < |G.V| ; i++)  
            for (j = 0; j < |G.V| ; j++)  
                if (dist[i][j] > dist[i][k] + dist[k][j])  
                    dist[i][j] = dist[i][k] + dist[k][j];  
}
```



Complexitate?

$$O(|V|^3)$$



Algoritmul A*

- Algoritm heuristic
- Nu găsește calea minimă dar una destul de apropiată
- $h(v)$ **estimare** a distanței de la nodul v la destinație
- $g(v)$ distanța de la nodul de start la v
- A* adaugă câte un nod la cale. Este ales mereu nodul care minimizează $f(v) = h(v) + g(v)$
- Dacă $h(x) \leq w(x, y) + h(y) \quad \forall (x, y) \in E$ funcția este numită monotonă sau consistentă.



```
A_Star(start, goal, h, G) {
    int prev[|G.V|] = { UNDEFINED };
    int g[|G.V|] = { INFINITE }; g[start] := 0;
    int f[|G.V|] = { INFINITE }; f[start] := h(start);

    while (!isEmpty(priorityQueue)) {
        node = pop(priorityQueue) //pop based on f[]
        if node = goal
            return;
        for each (neighbor of node) {
            if (g[node] + w(node, neighbor) < g[neighbor]) {
                prev[neighbor] = current
                g[neighbor] = g[node] + w(node, neighbor)
                f[neighbor] = g[neighbor] + h(neighbor)
                if (neighbor not in priorityQueue)
                    push(priorityQueue, neighbor);
            }
        }
    }
}
```



Dijkstra vs A*

