Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

GRAFURI PONDERATE

Introducere

- În afară de direcție, muchiile unui graf pot avea o **pondere**
- Dacă vârfurile unui graf ponderat reprezintă orașe, ponderile muchiilor pot reprezenta distanțele între orașe, costurile deplasării cu avionul sau numărul curselor de autobuze efectuate între orașe

Definirea structurii de graf ponderat

```
typedef struct {
int n, m;          // nr de noduri și nr de arce
int **c;           // matrice de ponderi (costuri)
} GrafP;
```

Funcții cu grafuri ponderate

```
//funcție de adăugare a arcului (v,w) la graful ponderat g
void addArc (GrafP & g, int v, int w, int cost) {
        g.c[v][w] = cost;
        g.m++;
}
//funcție de eliminare a arcului (v,w) din graful ponderat g
void delArc (GrafP & g, int v, int w) {
        g.c[v][w] = INF;
        g.m--;
}
//funcție care întoarce costul arcului (v,w)
int cost_arc (GrafP g, int v, int w) {
        return g.c[v][w];
}
```

Arborele minim de acoperire al unui graf neorientat ponderat

- Crearea arborelui minim de acoperire este mai dificilă într-un graf ponderat, decât într-unul neponderat
- Când se presupune că toate muchiile au lungimi egale, este simplu pentru algoritm să aleagă una dintre muchii și să o adauge la arborele minim de acoperire

Arbore minim de acoperire

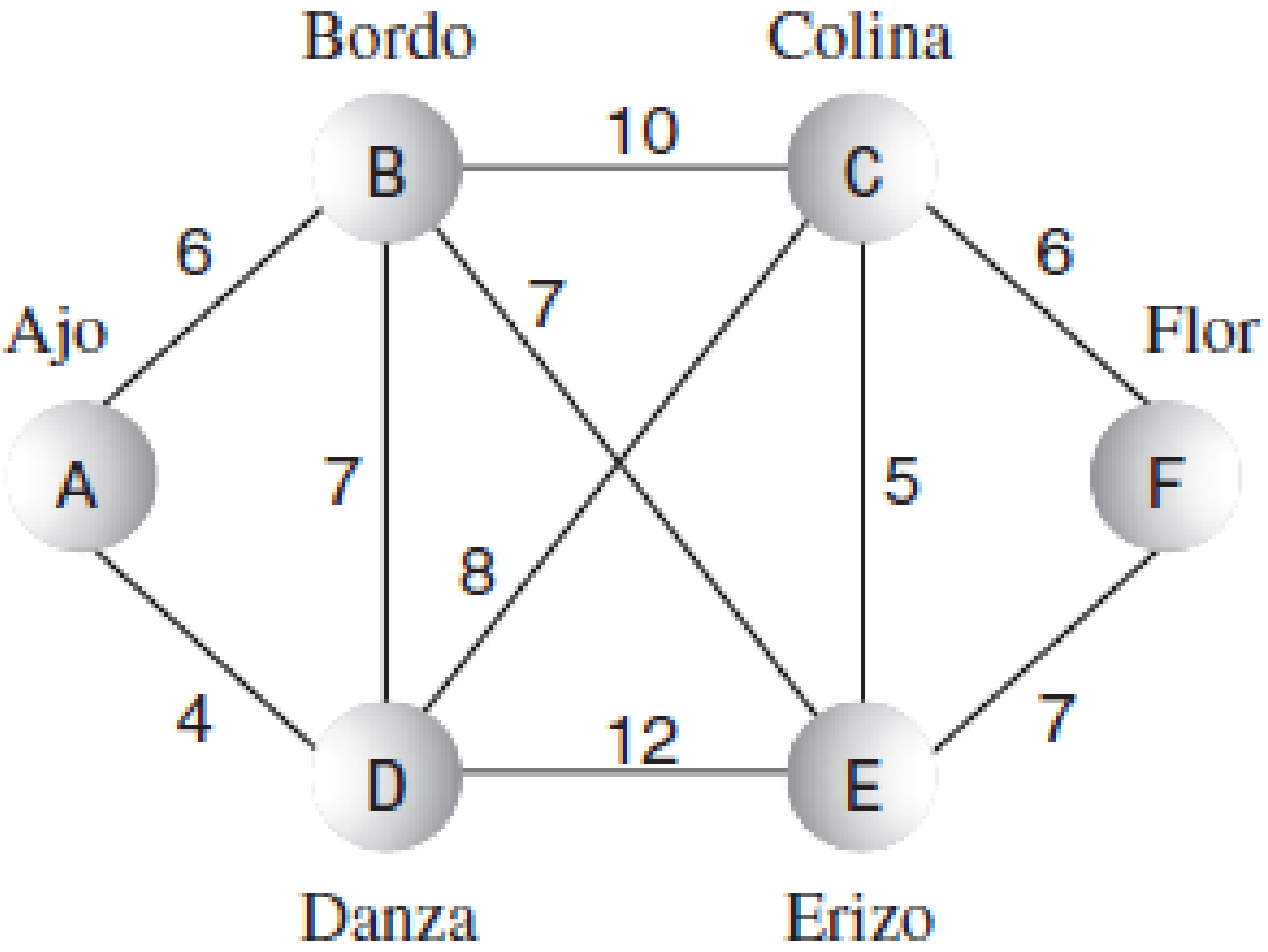
- Un graf conex are mai mulți arbori de acoperire, numărul acestor arbori fiind cu atât mai mare cu cât numărul de cicluri din graful inițial este mai mare
- Pentru un graf conex cu **n** vârfuri, arborii de acoperire au exact **n-1** muchii
- Pentru un graf dat, trebuie găsit arborele de acoperire de cost total minim sau unul dintre aceștia, dacă sunt mai mulți

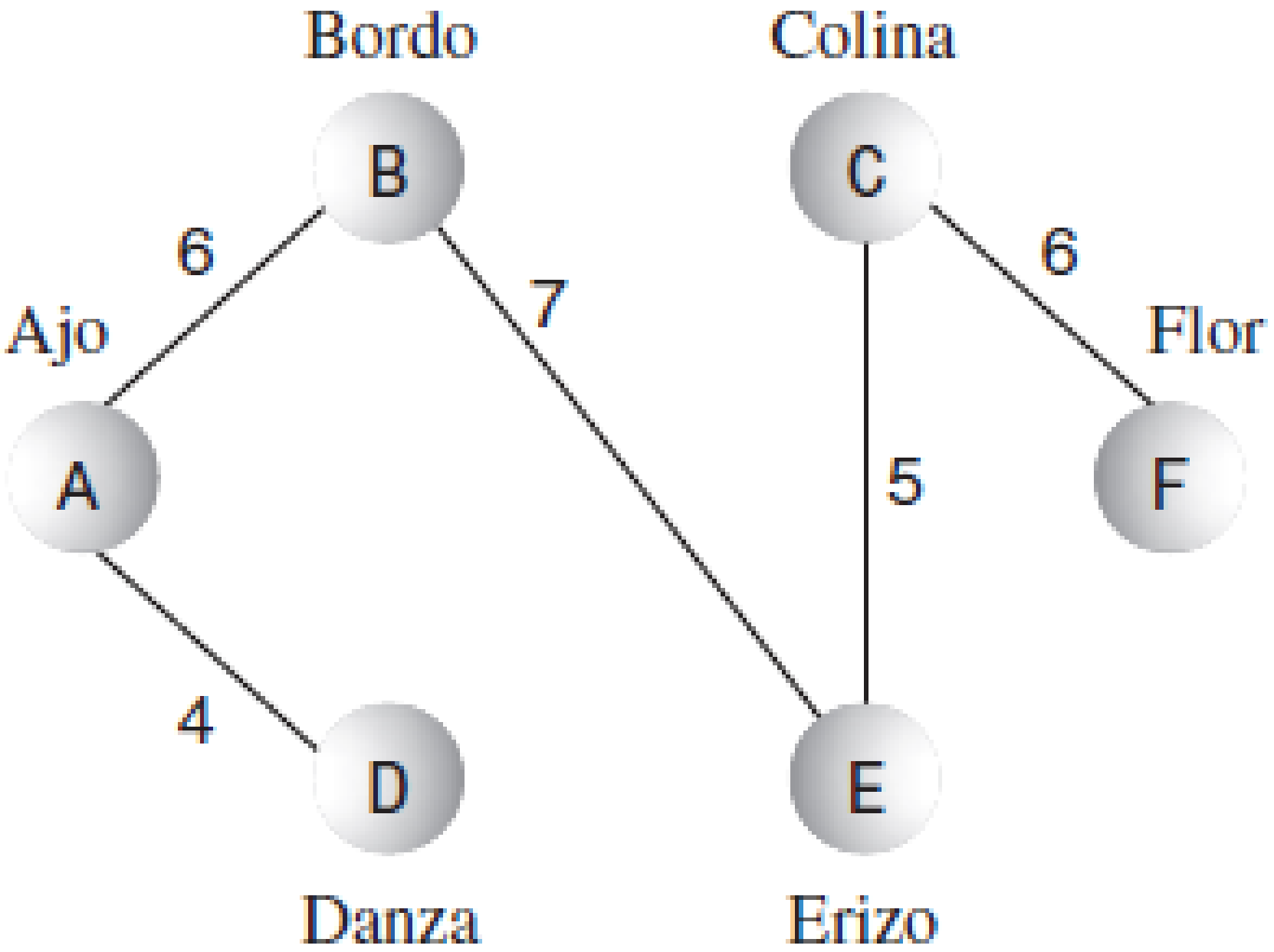
Arbore minim de acoperire

- Când muchiile au lungimi diferite, trebuie să efectuăm anumite calcule, înainte de a alege muchia potrivită
- Exemplu – se instalează o linie de televiziune prin cablu, care să conecteze șase orașe
- Cele șase orașe vor fi conectate prin cinci legături

Arbore minim de acoperire

- Costurile de conectare diferă pentru fiecare pereche de orașe, deci trebuie ales traseul care minimizează costul global
- Se consideră un graf ponderat cu șase vârfuri, reprezentând orașele: **A**jo, **B**ordo, **C**olina, **D**anza, **E**rizo și **F**lor


Bordo
Colina
B
C
10
6
6
7
Ajo
Flor
A
7
5
F
8
4
12
7
D
E
Danza
Erizo


Bordo
Colina
B
C
6
6
7
Ajo
Flor
A
5
F
4
D
E
Danza
Erizo

Modelarea problemei

- Fiecare muchie are asociată o **pondere**, care reprezintă costul pentru instalarea unei legături prin cablu între două orașe
- Se observă că unele dintre aceste legături sunt nepractice, din cauza costurilor prea mari
- Cum se poate alege un traseu care **minimizează** costul de instalare al rețelei de cabluri ?

Soluția problemei

- Se obține calculând un **arbore minim de acoperire**
- Acesta va avea cinci legături, va conecta toate cele șase orașe și va **minimiza** costul total al instalării acestor legături

Observații

- În cazul grafurilor, algoritmii încep parcurgerea cu un anumit vârf și se deplasează din aproape în aproape, examinând mai întâi vârfurile mai apropiate și apoi pe cele mai depărtate de punctul de pornire

Observații

- Nu se cunosc de la început toate costurile de instalare a cablului între oricare două orașe
- Culegerea acestor informații necesită timp și se efectuează pe parcurs

Se începe din **A**jo

- Din **A**jo, există două orașe la care se poate ajunge **direct**: **B**ordo și **D**anza
- Se creează o **coadă cu priorități** cu costurile legăturilor, introduse în ordinea crescătoare a costului
- **A**jo-**D**anza **4**
- **A**jo-**B**ordo **6**

Stabilirea legăturii **A**jo-**D**anza

- Trebuie mai întâi să adăugăm un vârf la arbore și abia apoi să încercăm să determinăm ponderile muchiilor care pleacă din acel vârf

Stabilirea legăturii **A**jo-**B**ordo

- După ce se stabilește legătura **A**jo-**D**anza, se inspectează toate orașele adiacente orașului **D**anza: **B**ordo, **C**olina și **E**rizo
- **A**jo-**B**ordo **6**
- **D**anza-**B**ordo **7**
- **D**anza-**C**olina **8**
- **D**anza-**E**rizo **12**

Observații

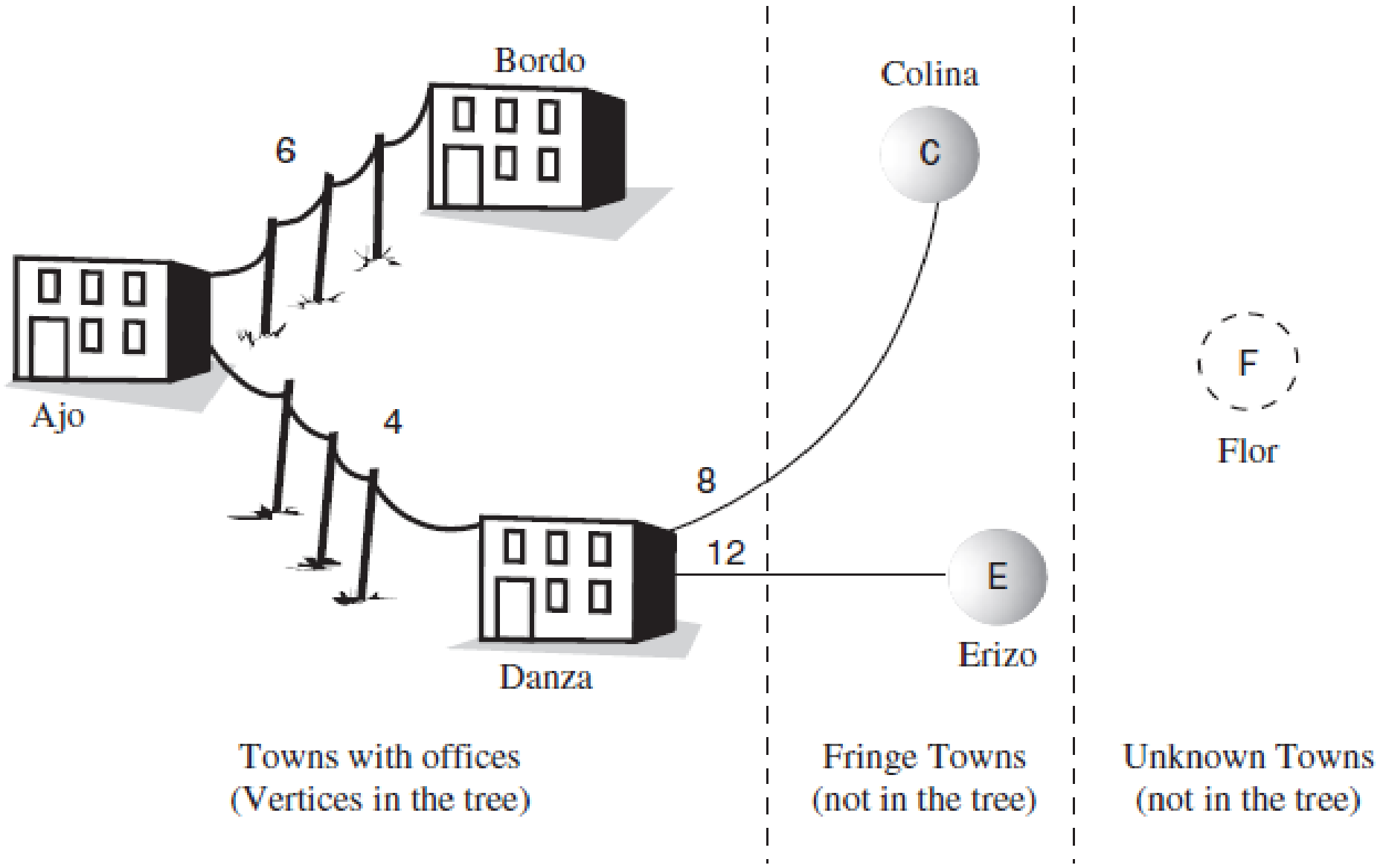
- Legătura **A**jo-**D**anza nu mai apare în listă deoarece s-a instalat deja cablul
- Rutele pe care s-a instalat deja cablul sunt șterse din listă
- *Regulă*: alegem întotdeauna din listă **muchia cea mai scurtă** (sau **cea mai ieftină**)

Observații

- La orice moment din procesul de construcție a rețelei de cabluri, există trei tipuri de orașe:
- **1**. Orașe care sunt deja în arborele minim de acoperire
- **2**. Orașe pentru care se cunoaște costul de conectare cu cel puțin un oraș care se află deja în arborele minim; acestea se numesc orașe de frontieră
- **3**. Orașe despre care nu cunoaștem încă nimic

Observații

- **A**jo, **D**anza și **B**ordo fac parte din prima categorie
- **C**olina și **E**rizo din a doua categorie
- **F**lor din ultima categorie
- Pe măsură ce algoritmul avansează, orașele din categoria a treia trec în cea de-a doua, iar cele de aici trec în prima


Bordo
Colina
C
6
Ajo
4
8
12
E
Erizo
F
Flor
Danza
Towns with offices
(Vertices in the tree)
Fringe Towns
(not in the tree)
Unknown Towns
(not in the tree)

Stabilirea legăturii **B**ordo-**E**rizo

- Lista conține următoarele costuri:
- **B**ordo-**E**rizo **7**
- **D**anza-**C**olina **8**
- **B**ordo-**C**olina **10**
- **D**anza-**E**rizo **12**

Observații

- Legătura **D**anza-**B**ordo exista în vechea listă, dar nu mai există acum, deoarece nu are sens să luăm în considerare legături între orașe deja conectate, chiar printr-o rută indirectă
- Din lista actuală, legătura cea mai ieftină este **B**ordo-**E**rizo, cu costul **7**

Stabilirea legăturii **E**rizo-**C**olina

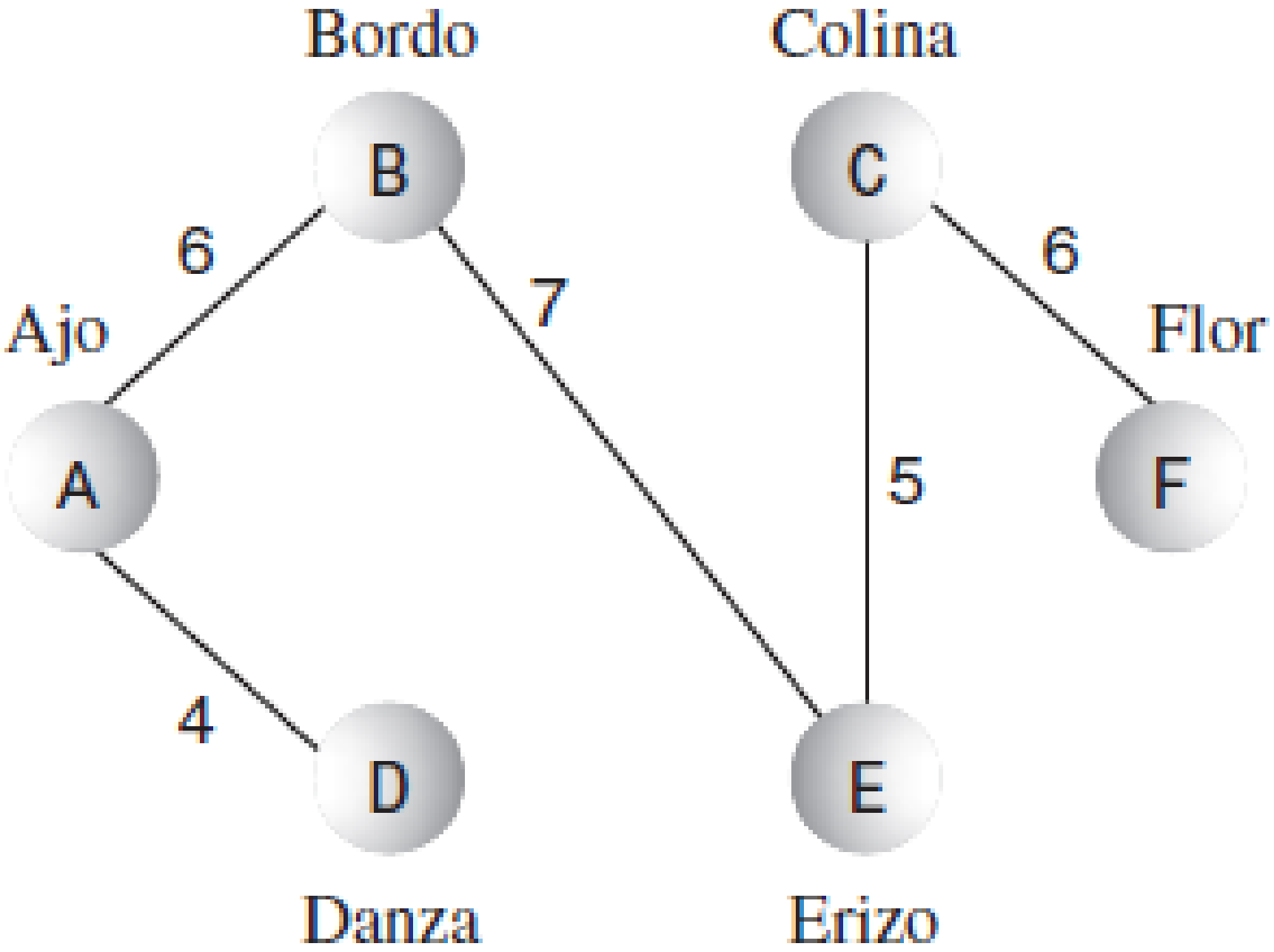
- Din **E**rizo, costul este **5** spre **C**olina și **7** spre **F**lor
- Legătura **D**anza-**E**rizo trebuie ștearsă din listă, deoarece **E**rizo este acum un oraș conectat

Conținutul actualizat al listei

- **E**rizo-**C**olina **5**
- **E**rizo-**F**lor **7**
- **D**anza-**C**olina **8**
- **B**ordo-**C**olina **10**
- Cea mai ieftină dintre aceste legături este **E**rizo-**C**olina

Legătura **C**olina-**F**lor

- După ștergerea orașelor deja conectate, lista conține doar legăturile:
- **C**olina-**F**lor **6**
- **E**rizo-**F**lor **7**
- Se stabilește ultima legătură, **C**olina-**F**lor, cu costul **6**
- Rutele obținute reprezintă **cea mai ieftină** soluție de conectare a tuturor orașelor


Bordo
Colina
B
C
6
7
6
Ajo
Flor
A
5
F
4
D
E
Danza
Erizo

Coada cu priorități

- O **listă** în care se poate selecta în mod repetat elementul cu valoarea **minimă** sugerează utilizarea unei **cozi cu priorități**

Pașii algoritmului

- Începem cu un **vârf**, pe care-l introducem în **arbore**
- *Repetăm* următorii *pași*:
- **1**. Determinăm toate muchiile, de la **cel mai recent** vârf către alte vârfuri, care nu aparțin arborelui; aceste muchii se inserează în **coada cu priorități**
- **2**. Alegem muchia cu **ponderea minimă**, iar aceasta se adaugă (împreună cu vârful de destinație) la **arborele de acoperire**

Observații

- Acești pași se repetă, până când toate vârfurile sunt în arbore
- În pasul **1**, prin "**cel mai recent**" se înțelege nodul cel mai recent adăugat în arbore
- Muchiile necesare sunt găsite în matricea de adiacență
- După pasul **1**, lista va conține toate muchiile cu originea în vârfuri din arbore și destinația în vârfuri de pe frontieră

Observații

- În procesul de menținere a listei de legături, o problemă este de a **șterge** legăturile care au destinația în orașul (vârful) cel mai recent conectat (adăugat în arborele de acoperire)
- Fără această operație, este posibil să instalăm legături prin cablu care nu mai sunt necesare

Observații

- Trebuie să ne asigurăm că în coada cu priorități nu există muchii a căror destinație reprezintă un nod aflat deja în arbore
- Putem parcurge coada, căutând și eliminând toate muchiile de acest fel, de fiecare dată când adăugăm un vârf nou în arbore

Observații

- Este mai simplu să memorăm în coadă, la un moment dat, o singură muchie de la un vârf din arbore către fiecare nod de frontieră dat
- Coada trebuie sa conțină o singură muchie către fiecare vârf din categoria a doua

Step Number List	Unpruned Edge List	Pruned Edge (In Priority Queue)	Duplicate Removed from Priority Queue
1	AB6, AD4	AB6, AD4	
2	DE12, DC8, DB7, AB6	DE12, DC8, AB6	DB7(AB6)
3	DE12, BC10, DC8, BE7	DC8, BE7	DE12(BE7), BC10(DC8)
4	BC10, DC8, EF7, EC5	EF7, EC5	BC10(EC5), DC8(EC5)
5	EF7, CF6	CF6	EF7

Căutarea duplicatelor în coada cu priorități

- De fiecare dată când adăugăm o muchie în coadă, ne asigurăm că nu mai există o altă muchie cu aceeași destinație
- Dacă mai există astfel de muchii, o păstrăm numai pe cea cu **ponderea minimă**
- Această operație necesită căutarea secvențială prin coada cu priorități

Algoritmul lui Prim

- Algoritmul folosește o **coadă cu priorități** de muchii care leagă vârfuri din arborele minim de acoperire cu alte vârfuri (coada se modifică pe măsură ce algoritmul evoluează)

Algoritmul lui Prim

- Algoritmul se bazează pe observația următoare: fie **S** o submulțime a vârfurilor grafului și **R** submulțimea de vârfuri care nu sunt în **S**
- Muchia de **cost minim** care unește vârfurile din **S** cu vârfurile din **R** face parte din **arborele minim de acoperire**

Algoritmul lui Prim

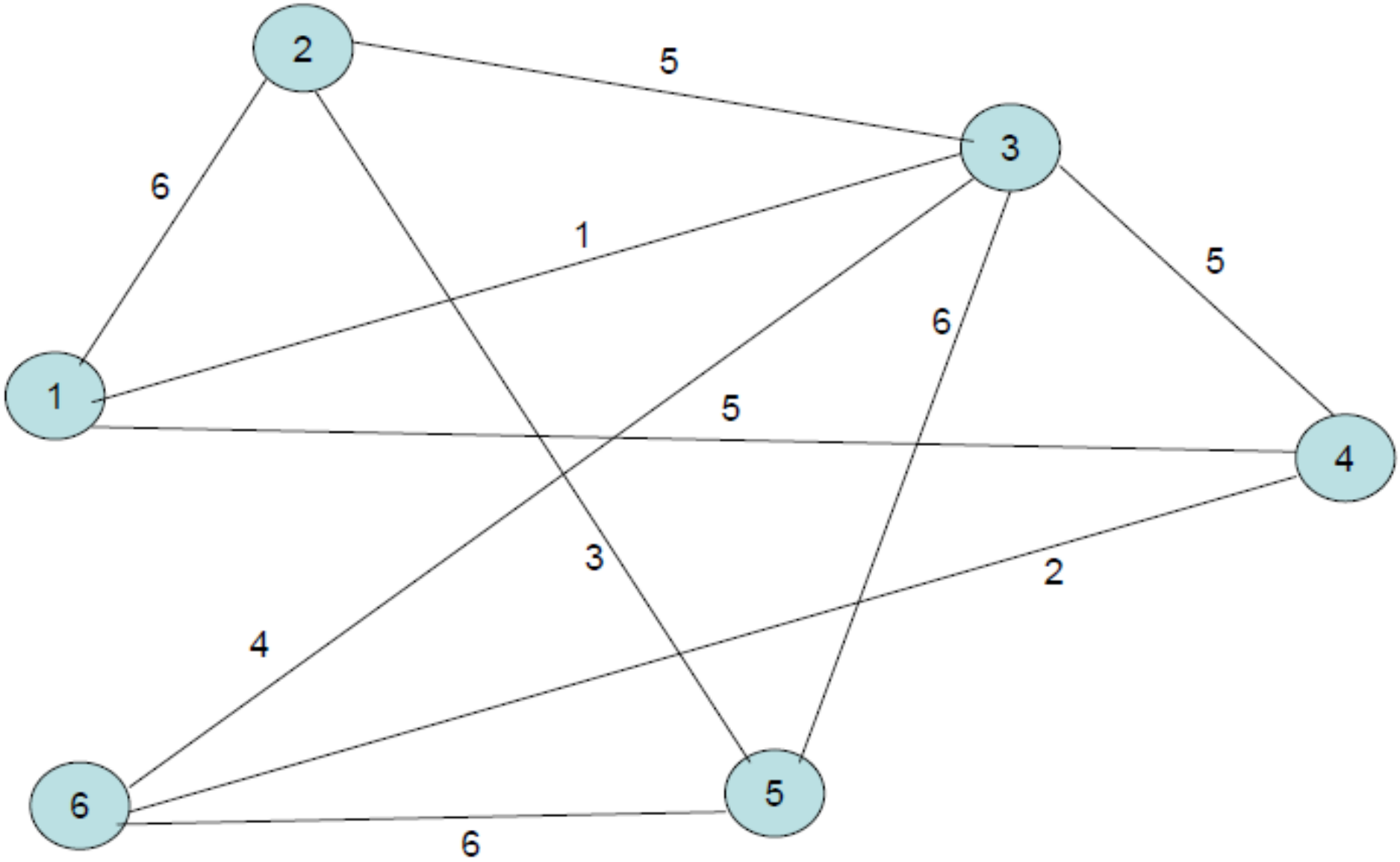
- Se poate folosi noțiunea de “**tăietură**” în graf: se taie toate muchiile care leagă un nod **k** de restul nodurilor din graf și se determină muchia de **cost minim** dintre muchiile tăiate
- Această muchie va face parte din arborele minim de acoperire și va uni nodul **k** cu arborele minim de acoperire al grafului rămas după îndepărtarea nodului **k**
- La fiecare pas se face o nouă tăietură în graful rămas și se determină o altă muchie din arborele minim de acoperire

Algoritmul lui Prim

- Fiecare tăietură în graf împarte mulțimea nodurilor din graf în două submulțimi:
  - **S** (noduri incluse în arborele minim de acoperire)
  - **R** (restul nodurilor)
- Inițial, **S** = {**1**}, dacă se pornește din nodul **1**, iar în final **S** va conține toate nodurile din graf

Exemplu

- Se consideră următorul graf neorientat și ponderat, cu 6 noduri, cu muchiile și costurile asociate:
- (1,2) = 6; (1,3) = 1; (1,4) = 5;
- (2,3) = 5; (2,5) = 3;
- (3,4) = 5; (3,5) = 6; (3,6) = 4;
- (4,6) = 2;
- (5,6) = 6;


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

Arbore minim de acoperire

- Format din muchiile:
- (1,3), (3,6), (4,6), (2,3), (2,5)
- Cost total:
- 1 + 4 + 2 + 5 + 3 = 15

S	Muchii între S și R (muchii tăiate)	Minim	y
1	(1,2)=6; (1,3)=1; (1,4)=5;	(1,3)=1	3
1,3	(1,2)=6; (1,4)=5; (3,2)=5; (3,4)=5; (3,5)=6; (3,6)=4	(3,6)=4	6
1,3,6	(1,2)=6; (1,4)=5; (3,2)=5; (3,4)=5; (3,5)=6; (6,4)=2; (6,5)=6	(6,4)=2	4
1,3,6,4	(1,2)=6; (3,2)=5; (3,5)=6; (6,5)=6	(3,2)=5	2
1,3,6,4,2	(2,5)=3; (3,5)=6; (6,5)=6	(2,5)=3	5

Soluția problemei

- O mulțime de muchii, adică un tablou de perechi de noduri, sau două tablouri de întregi X și Y, cu semnificația că o pereche x[i]-y[i] reprezintă o muchie din **arborele minim de acoperire**

Algoritmul lui Prim

- Este un algoritm “**greedy**”
- Lista de candidați este lista muchiilor “tăiate”, adică muchiile care unesc noduri din S cu noduri din R
- La fiecare pas se alege muchia de cost minim dintre muchiile “tăiate” și se generează o altă listă de candidați