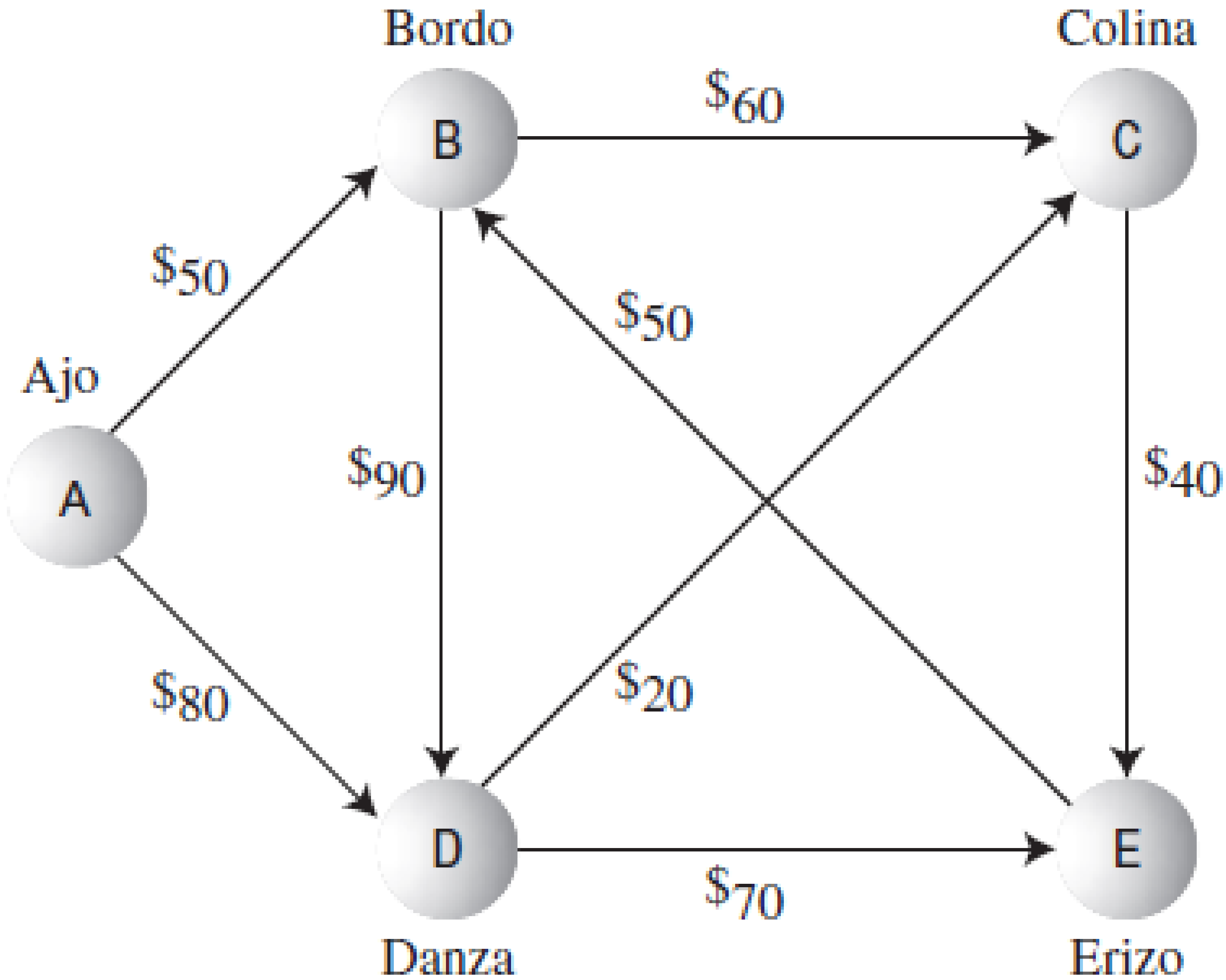
Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

ALGORITMUL LUI DIJKSTRA

Determinarea drumului de lungime minimă

- O aplicație frecventă a grafurilor **orientate și ponderate** este determinarea **drumului cel mai scurt** dintre două vârfuri date
- Dorim să determinăm ruta cea mai ieftină de la un oraș la un alt oraș



Bordo
Colina
B
C
$60
$50
$50
Ajo
A
$90
$40
$80
$20
D
E
$70
Danza
Erizo

Observații

- Muchiile grafului sunt **orientate și ponderate**
- Acestea reprezintă costurile pe calea ferată, pe care se circulă într-un singur sens
- Deși în acest caz suntem interesați de minimizarea unui cost, numele algoritmului este **problema drumului minim**

Observații

- Prin **drum minim** nu se înțelege neapărat drumul cel mai scurt, din punct de vedere fizic
- Poate fi vorba de drumul cel mai ieftin, cel mai rapid sau cel mai bun, dintr-un alt punct de vedere

Costuri minime

- Între oricare două orașe există mai multe drumuri posibile
- Problema **drumului minim** presupune determinarea, pentru un punct de pornire dat și o destinație precizată, a traseului cel mai ieftin

Graf orientat și ponderat

- Rețeaua de cale ferată cuprinde numai linii unidirecționale
- Această situație poate fi modelată printr-un **graf orientat și ponderat**

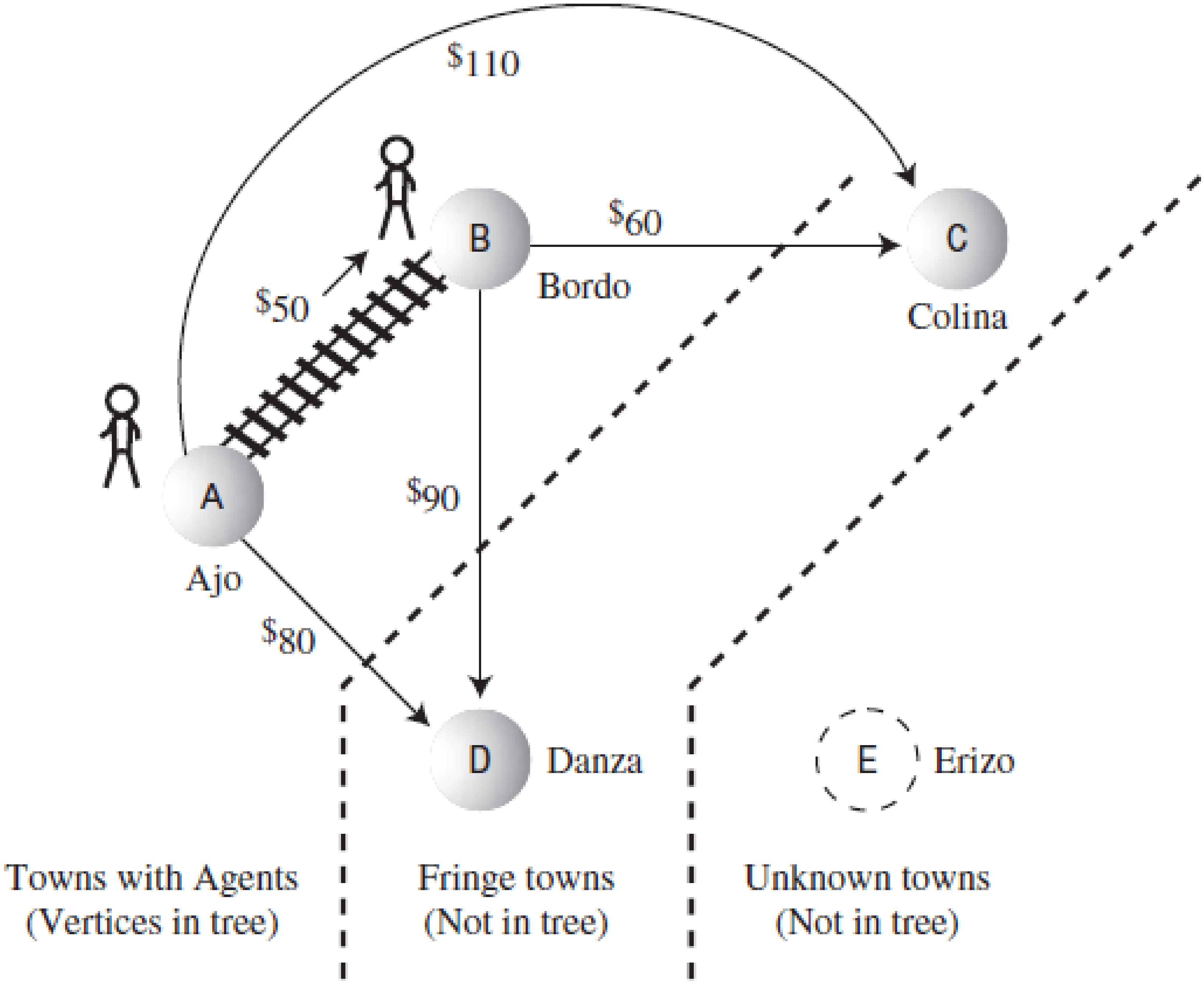
Algoritmul lui Dijkstra

- Soluția la problema drumului minim este numită **algoritmul lui Dijkstra**, după numele lui Edsger Dijkstra, care l-a descoperit în 1959
- Algoritmul se bazează pe reprezentarea grafului cu ajutorul matricei de adiacență
- Algoritmul permite determinarea atât a **drumului minim** dintre un vârf precizat și altul, cât și a **drumurilor minime**, de la vârful precizat la toate celelalte vârfuri

Prezentarea algoritmului

- Determinăm cel mai ieftin mod de a călători de la **A**jo până la orice alt oraș
- Algoritmul trebuie să examineze numai o singură informație la un moment dat
- *Regulă*: Întotdeauna se merge în orașul pentru care costul total calculat din punctul de pornire (**A**jo) este minim

From Ajo to→	Bordo	Colina	Danza	Erizo
Step 1	50 (via Ajo)	inf	80 (via Ajo)	inf



$110
$60
$50
$90
$80
A
Ajo
B
Bordo
C
Colina
D
Danza
E
Erizo
Towns with Agents
(Vertices in tree)
Fringe towns
(Not in tree)
Unknown towns
(Not in tree)

From Ajo to→	Bordo	Colina	Danza	Erizo
Step 1	50 (via Ajo)	inf	80 (via Ajo)	inf
Step 2	50 (via Ajo)*	110 (via Bordo)	80 (via Ajo)	inf

3 tipuri de orașe

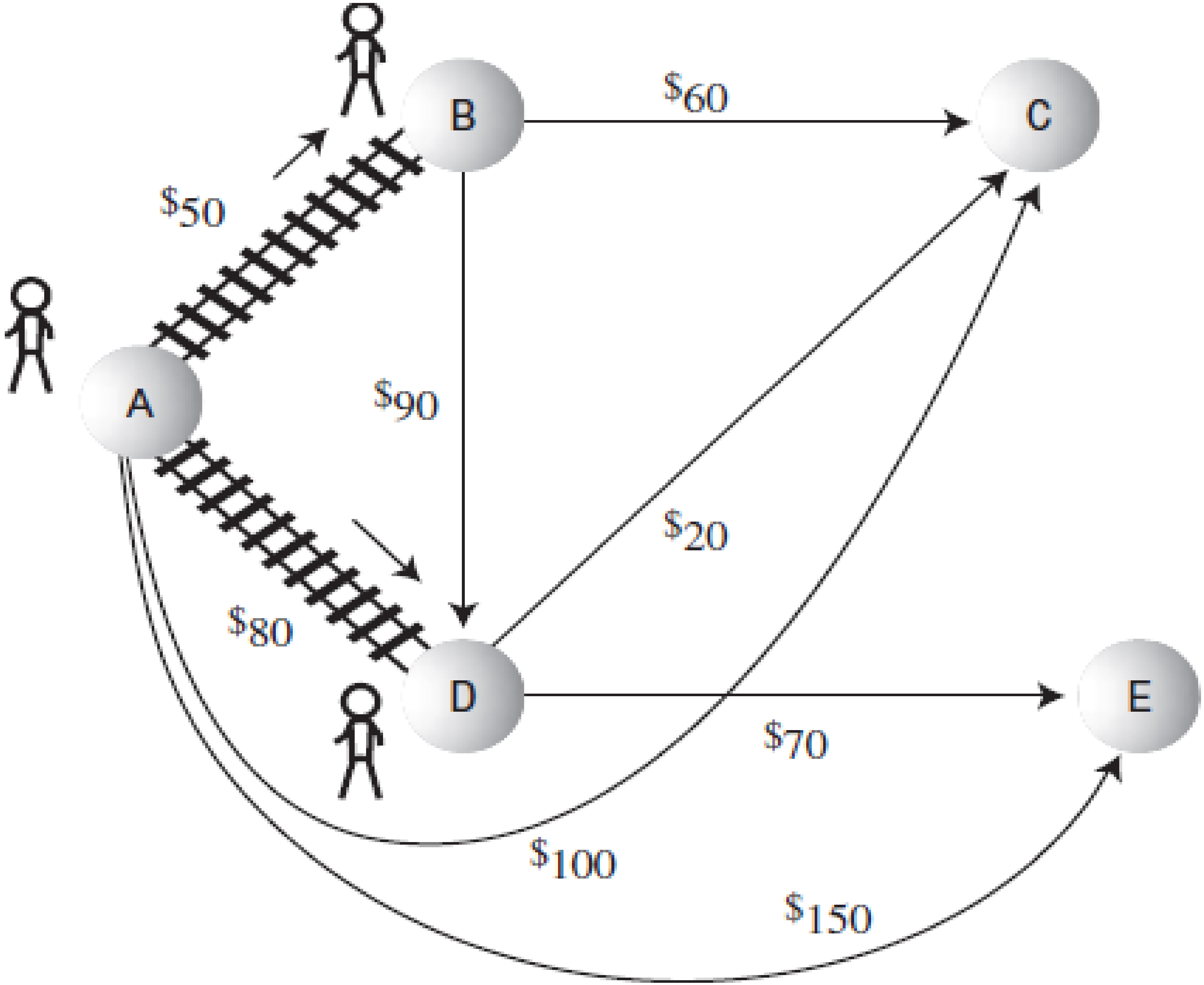
- **1**. Orașe prin care am trecut deja; acestea sunt în arbore
- **2**. Orașe pentru care știm costurile de călătorie; acestea sunt pe frontieră
- **3**. Orașe necunoscute

Observații

- **A**jo și **B**ordo sunt de tipul 1
- Orașele de tipul 1 formează un arbore, care constă din drumurile care încep cu același punct de pornire, fiecare terminându-se cu un alt vârf, diferit, de destinație
- **D**anza și **C**olina sunt orașe de tipul 2 (de frontieră)

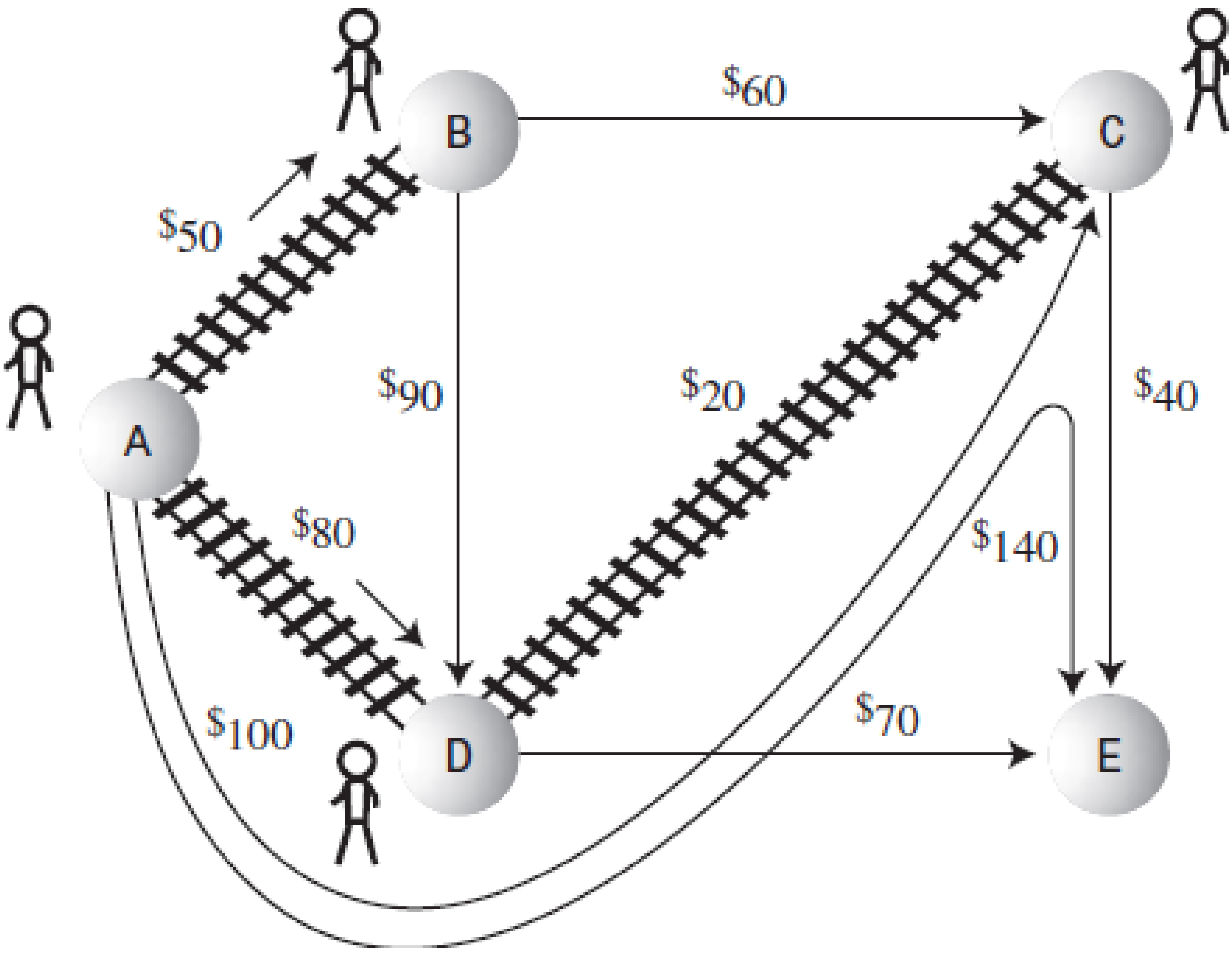
Observații

- Orașele se vor deplasa de la tipul 3 (necunoscute), în cel de-al doilea tip (de frontieră), iar de aici în arbore, pe măsură ce algoritmul avansează



B
C
A
D
E
$60
$50
$90
$20
$80
$70
$100
$150

From Ajo to→	Bordo	Colina	Danza	Erizo
Step 1	50 (via Ajo)	inf	80 (via Ajo)	inf
Step 2	50 (via Ajo)*	110 (via Bordo)	80 (via Ajo)	inf
Step 3	50 (via Ajo)*	100 (via Danza)	80 (via Ajo)*	150 (via Danza)



$60
B
C
$50
$90
$20
$40
A
$80
$140
$100
$70
D
E

From Ajo to→	Bordo	Colina	Danza	Erizo
Step 1	50 (via Ajo)	inf	80 (via Ajo)	inf
Step 2	50 (via Ajo)*	110 (via Bordo)	80 (via Ajo)	inf
Step 3	50 (via Ajo)*	100 (via Danza)	80 (via Ajo)*	150 (via Danza)
Step 4	50 (via Ajo)*	100 (via Danza)*	80 (via Ajo)*	140 (via Colina)

From Ajo to→	Bordo	Colina	Danza	Erizo
Step 1	50 (via Ajo)	inf	80 (via Ajo)	inf
Step 2	50 (via Ajo)*	110 (via Bordo)	80 (via Ajo)	inf
Step 3	50 (via Ajo)*	100 (via Danza)	80 (via Ajo)*	150 (via Danza)
Step 4	50 (via Ajo)*	100 (via Danza)*	80 (via Ajo)*	140 (via Colina)
Step 5	50 (via Ajo)*	100 (via Danza)*	80 (via Ajo)*	140 (via Colina)*

Observații

- Când se cunosc costurile călătoriei din **A**jo până în oricare alt oraș, algoritmul se termină
- Pasul 5 indică rutele cele mai ieftine, din **A**jo până în toate celelalte orașe

Ideile algoritmului lui Dijkstra

- 1. De fiecare dată când ne aflăm într-un oraș nou, actualizăm lista de costuri
- În listă reținem numai drumul de **cost minim** (cunoscut până în momentul curent) dintre punctul de pornire și un alt oraș precizat
- 2. Mergem întotdeauna în orașul care are calea cea mai ieftină față de punctul de pornire

Idei de implementare

- Se consideră ca nod sursă **i** nodul 1 și se determină lungimile drumurilor minime d[2], d[3],..., d[n] până la nodurile 2, 3,..., n
- Pentru memorarea nodurilor de pe un drum minim se folosește un tablou **P**, cu p[i] = nodul precedent lui **i** pe drumul minim de la 1 la i (mulțimea drumurilor minime formează un arbore, iar tabloul P reprezintă acest arbore de căi în graf)

Idei de implementare

- Se folosește un tablou **D**, unde d[i] este distanța minimă de la 1 la i, dintre drumurile care trec prin noduri deja selectate
- O variabilă **S** de tip mulțime memorează numerele nodurilor cu distanță minimă față de nodul 1, găsite până la un moment dat
- Inițial, S = {1} și d[i] = cost[1][i], adică se consideră arcul direct de la 1 la i ca drum minim între 1 și i
- Pe măsură ce algoritmul evoluează, se actualizează **D** și **S**

Pseudocod algoritm Dijkstra

```
S = {1}      // S = mulțime noduri pentru care s-a
             //determinat distanța minimă față de nodul 1
repetă cât timp S conține mai puțin de n noduri {
             găsește muchia (x,y) cu x ∈ S și y ∉ S care
             minimizează d[x] + cost(x,y)
             adaugă y la S
             d[y] = d[x] + cost(x,y)
    }
```

Idei de implementare

- La fiecare pas din algoritmul lui Dijkstra:
- **1**. Se găsește dintre nodurile $j \notin S$ acel nod "jmin" care are distanța minimă față de nodurile din S
- **2**. Se adaugă nodul "jmin" la mulțimea S
- **3**. Se recalculează distanțele de la nodul 1 la nodurile care nu fac parte din S, pentru că distanțele la nodurile din S rămân neschimbate
- **4**. Se reține în p[j] numărul nodului precedent cel mai apropiat de nodul j (de pe drumul minim de la 1 la j)

Exemplu

- Se consideră un graf orientat și ponderat, cu următoarele costuri ale arcelor:
- (1,2) = 5; (1,4) = 2; (1,5) = 6;
- (2,3) = 3;
- (3,2) = 4; (3,5) = 4;
- (4,2) = 2; (4,3) = 7; (4,5) = 3;
- (5,3) = 3;

Exemplu

- Drumurile posibile între 1 și 3 și costul lor:
- 1-2-3 = 8
- 1-4-3 = 9
- 1-4-2-3 = 7
- 1-4-5-3 = 8
- 1-5-3 = 9

Exemplu

- Drumurile minime de la 1 la celelalte noduri din graf:
- 1-4-2 cu costul 4
- 1-4-2-3 cu costul 7
- 1-4 cu costul 2
- 1-4-5 cu costul 5

Observații

- Într-un drum minim, fiecare drum parțial este minim
- În drumul 1-4-2-3, drumurile parțiale 1-4-2 și 1-4 sunt și ele minime
- Evoluția tablourilor **D** și **S** pentru acest graf, în cazul algoritmului lui Dijkstra:

S	d[2]	d[3]	d[4]	d[5]	Nod selectat
1	5	M	2	6	4
1,4	4	9	2	5	2
1,4,2	4	7	2	5	5
1,4,2,5	4	7	2	5	3

Observații

- Tabloul **P** va arăta astfel:

p[2]	p[3]	p[4]	p[5]
4	2	1	4

Funcție Dijkstra

```
void dijkstra (GrafP g, int p[]) {
int d[M], s[M];
// s = noduri pentru care se știe distanța minimă
int dmin;
int jmin, i, j;
for (i = 2; i <= g.n; i++) {
    p[i]=1; d[i]=cost_arc(g, 1, i);
                    // distanțe inițiale de la 1 la alte noduri
  }
s[1] = 1;
```

Funcție Dijkstra

```
for (i = 2; i <= g.n; i++) {                    // repetă de n-1 ori
    // caută nodul j pentru care d[j] este minim
    dmin = MARE;
    for (j = 2; j <= g.n; j++)
      // determină minimul dintre distanțele d[j]
      if (s[j] == 0 && dmin > d[j]) {
            // dacă j ∉ S și este mai aproape de S
            dmin = d[j]; jmin = j;
      }
```

Funcție Dijkstra

```
s[jmin] = 1;                              // adaugă nodul jmin la S
for (j = 2; j <= g.n; j++)
     // recalculare distanțe noduri față de 1
     if ( d[j] > d[jmin] + cost_arc(g, jmin, j) ) {
          d[j] = d[jmin] + cost_arc(g, jmin, j);
          p[j] = jmin;
          // predecesorul lui j pe drumul minim
     }
  }
}
```

Observații

- Valoarea constantei **MARE**, folosită pentru a marca în matricea de costuri absența unui arc, nu poate fi mai mare ca jumătate din valoarea maximă pentru tipul întreg, deoarece la însumarea costurilor a două drumuri se poate depăși cel mai mare întreg (se pot folosi pentru costuri și numere reale foarte mari)