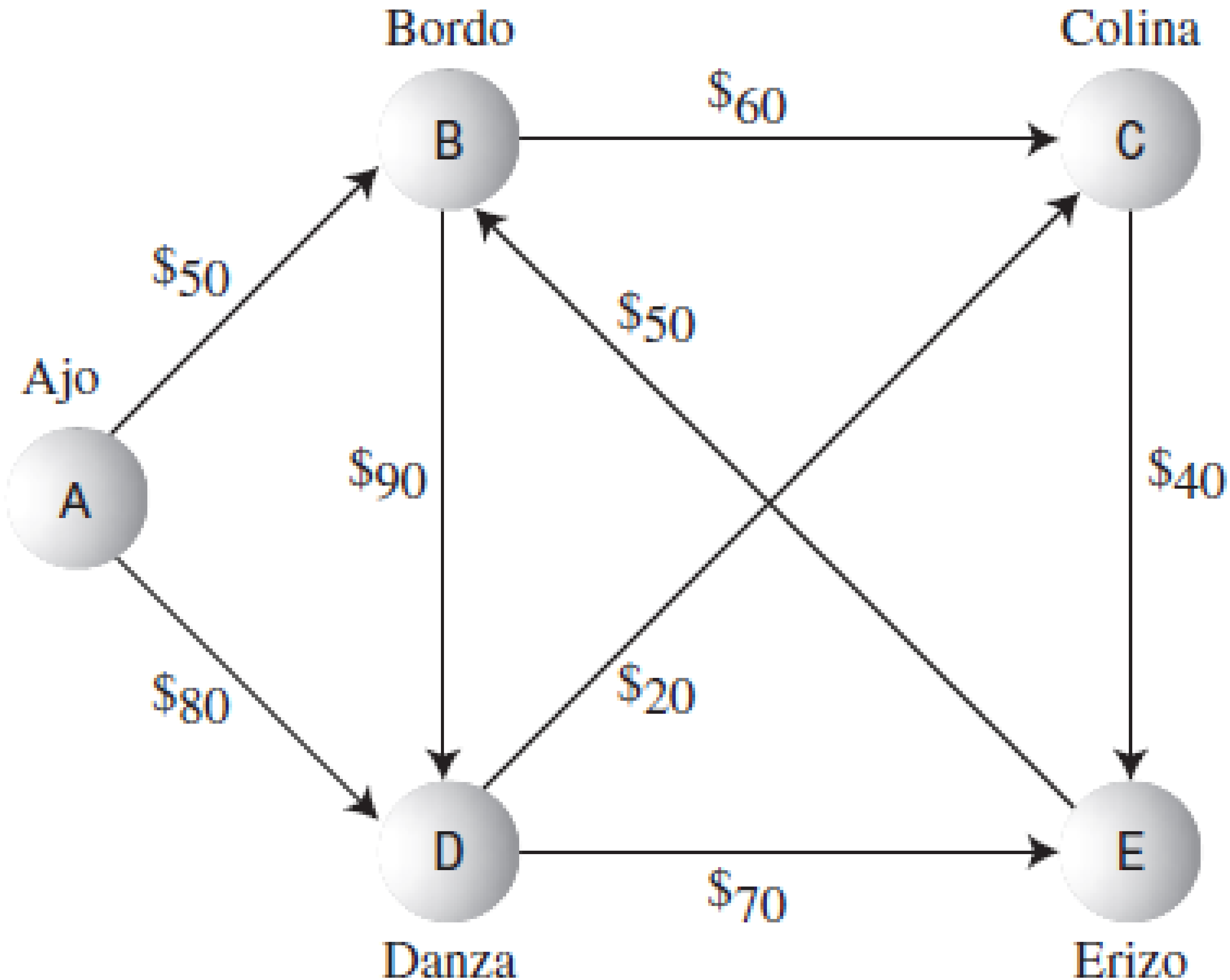
Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

ALGORITMUL LUI FLOYD

Problema drumului de lungime minimă din orice vârf

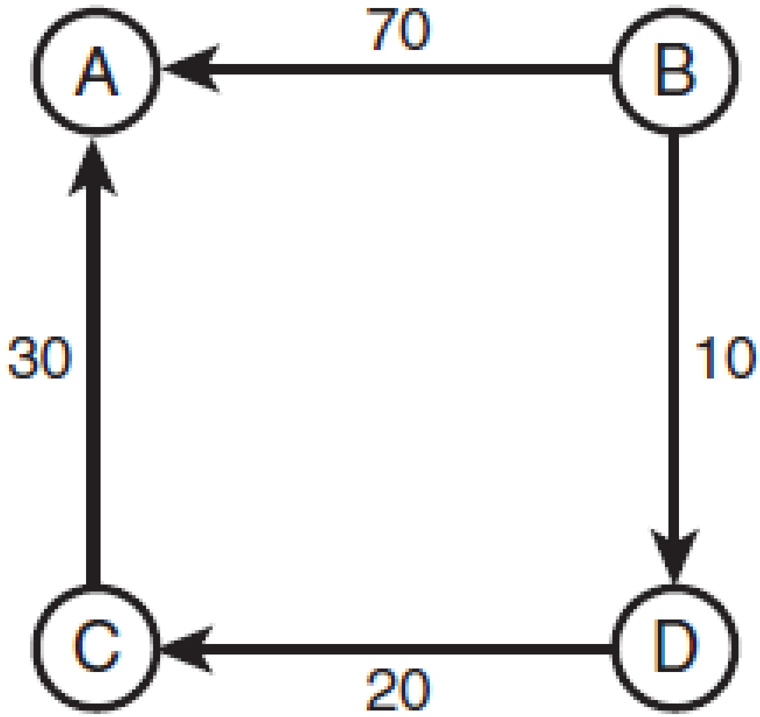
- Problema se referă la aflarea **costului minim** din **orice vârf** către **oricare alt vârf**, folosind muchii multiple
- Aceasta se numește problema drumului de lungime minimă din orice vârf (**all-pairs shortest path** problem), pentru un graf **orientat** și **ponderat**


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

	A	B	C	D	E
A	—-	50	100	80	140
B	—-	—-	60	90	100
C	—-	90	—-	180	40
D	—-	110	20	—-	60
E	—-	50	110	140	—-

Algoritmul lui Floyd

- Algoritmul lui Warshall reprezintă o modalitate rapidă de a crea un tabel care indică vârfurile în care se poate ajunge dintr-un anumit vârf, într-unul sau mai mulți pași, într-un graf orientat
- O abordare similară pentru grafuri orientate și ponderate este folosită de **algoritmul lui Floyd**, descoperit de Robert Floyd în 1962



	A	B	C	D
A				
B	70			10
C	30			
D			20	

Observații

- Matricea de adiacență indică costurile tuturor căilor cu un singur arc
- Se dorește extinderea acestei matrici pentru a indica costurile tuturor căilor, indiferent de lungimea lor
- De exemplu, se poate ajunge de la B la C cu un cost de 30 (10 de la B la D și 20 de la D la C)

Observații

- Similar algoritmului lui Warshall, se modifică matricea de adiacență
- Se examinează fiecare celulă de pe fiecare rând
- Dacă există o pondere pozitivă, de exemplu 30 la intersecția liniei C cu coloana A, atunci se analizează coloana C, deoarece C reprezintă linia unde se află 30

Observații

- Dacă se găsește o celulă în coloana C, de exemplu 20 la linia D, atunci există o cale de la C la A cu o pondere de 30 și o cale de la D la C cu o pondere de 20
- Se poate deduce că există o cale cu două arce, de la D la A, cu o pondere de 50

Observații

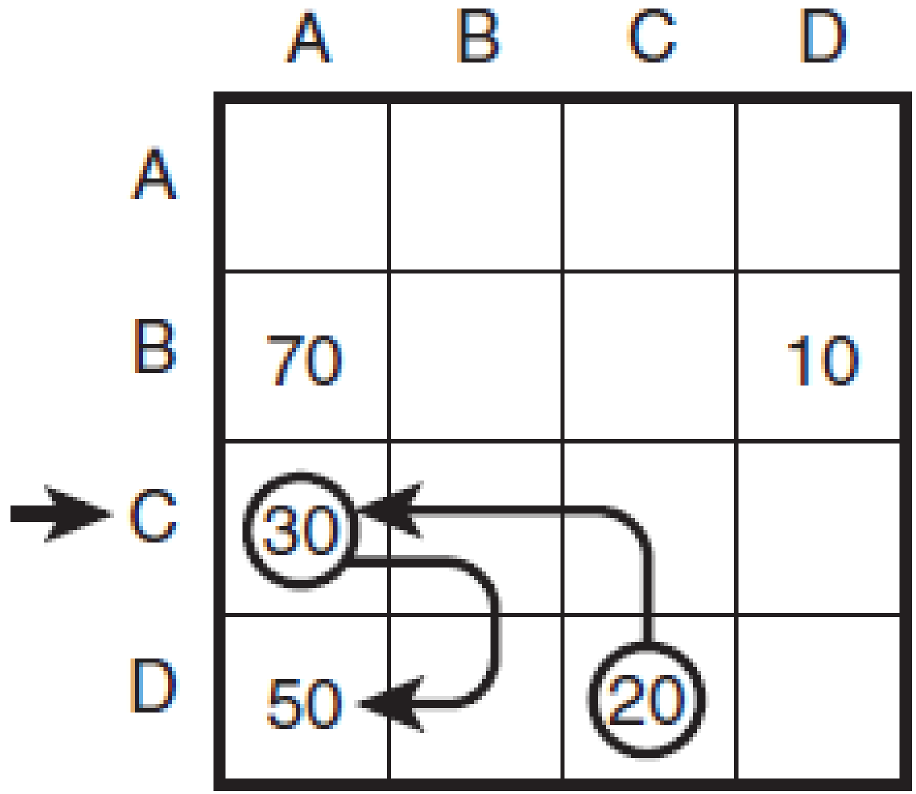
- Linia A este vidă
- Pe linia B este 70 în coloana A și 10 în coloana D, dar coloana B este vidă, deci arcele care încep din B nu pot fi combinate cu niciun arc care se termină în B

Observații

- În linia C se află 30 pe coloana A
- În coloana C se află 20 pe linia D
- Arcul de la C la A are o pondere de 30
- Arcul de la D la C are o pondere de 20
- Se obține calea de la D la A cu ponderea de 50

a)

$y = 2, x = 0, z = 3$



$C \rightarrow A$ & $D \rightarrow C$

30 20

so $D \rightarrow A$

50

Observații

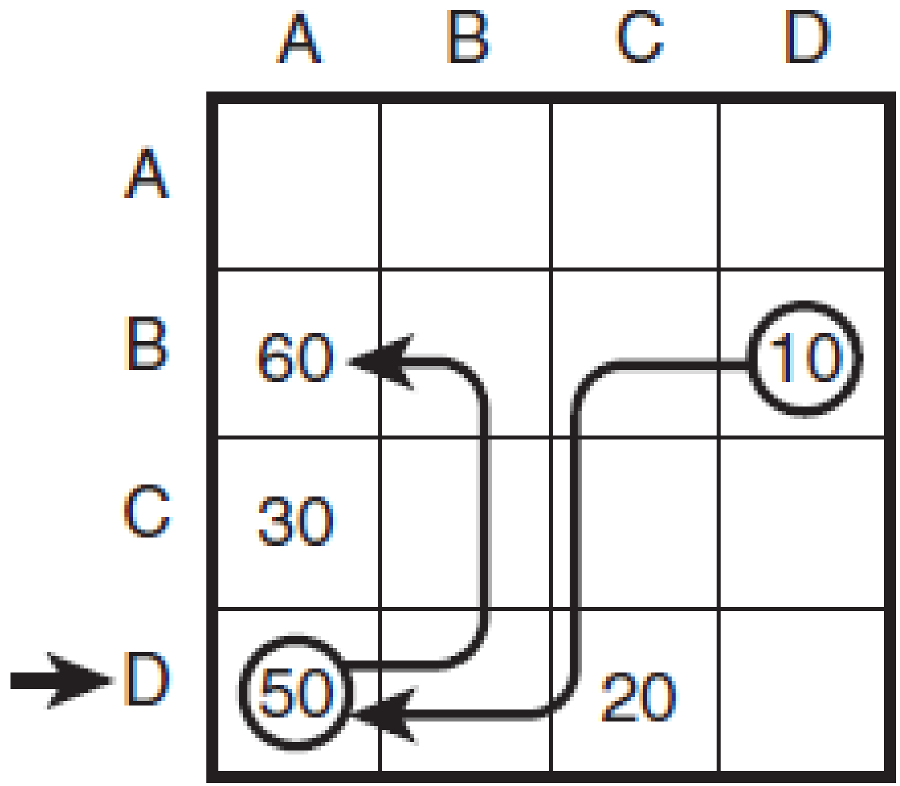
- Linia D arată o situație interesantă – se poate **micșora** un cost existent deja
- Pe linia D există 50 în coloana A
- Pe linia B există 10 în coloana D
- Există o cale de la B la A cu costul 60
- Cu toate acestea, există deja costul 70 pe linia B, în coloana A

Observații

- Deoarece 60 e mai mic decât 70, se înlocuiește 70 cu 60
- În cazul căilor multiple de la un vârf la altul, tabelul indică calea de cost minim

b)

$y = 3,\ x = 0,\ z = 1$



$D \rightarrow A$ & $B \rightarrow D$

50 10

so $B \rightarrow A$

60

c)

$y = 3,\ x = 2,\ z = 1$

	A	B	C	D
A				
B	60		30	10
C	30			
D	50		20	

$D \rightarrow C$ & $B \rightarrow D$

20 10

so $B \rightarrow C$

30

Observații

- Implementarea algoritmului lui Floyd este similară algoritmului lui Warshall
- În locul inserării valorii 1 în tabel, cum se procedează în algoritmul lui Warshall, când se găsește o cale cu două muchii, se adaugă costul căii cu două arce și se inserează suma costurilor celor două arce în tabel

Observații

- Algoritmul lui Floyd se bazează pe utilizarea unei matrice **A** a distanțelor minime, ale cărei valori sunt calculate în mai multe etape
- Inițial:
- $A[i,j] = cost[i,j]$, pentru orice $i \neq j$
- $A[i,j] = 0$, pentru $i=j$
- $A[i,j] = \infty$, dacă nu există arcul (i,j)

Observații

- Calculul distanțelor minime se face în **n** iterații
- La iterația **k**, **A[i,j]** va avea ca valoare cea mai mică distanță între **i** și **j**, pe căi care nu conțin vârfuri numerotate peste **k** (exceptând capetele **i** și **j**)
- Se utilizează formula:
- $A_k[i,j] = \min(A_{k-1}[i,j], A_{k-1}[i,k] + A_{k-1}[k,j])$

Observații

- Deoarece
- $A_k[i,k] = A_{k-1}[i,k]$ și
- $A_k[k,j] = A_{k-1}[k,j]$
- nicio intrare cu unul din indici egal cu **k** nu se modifică la iterația **k**
- Se poate realiza calculul cu o singură copie a matricei A

```
AlgoritmFloyd() {
pentru toate liniile i execută
   pentru toate coloanele j execută
                  A[i,j] ← cost[i,j]
   pentru toate liniile i execută
                  A[i,i] ← 0
   pentru k de la 1 la n execută
      pentru toate liniile i execută
         pentru toate coloanele j execută
            dacă A[i,k] + A[k,j] < A[i,j] atunci
                           A[i,j] ← A[i,k] + A[k,j]
}
```

Eficiența algoritmilor pe grafuri

- Algoritmii lui **Warshall** și **Floyd** au o complexitate de ordinul $O(N^3)$
- Multe probleme reale nu pot fi rezolvate într-un timp rezonabil – astfel de probleme se numesc **NP complete**

Problema calului la șah

- Aceasta este o problemă NP completă, datorită numărului mare de mișcări posibile
- Fiecare mișcare poate genera 8 stări următoare
- Acest număr poate fi redus prin mișcări în afara tablei sau mișcări într-o stare anterioară

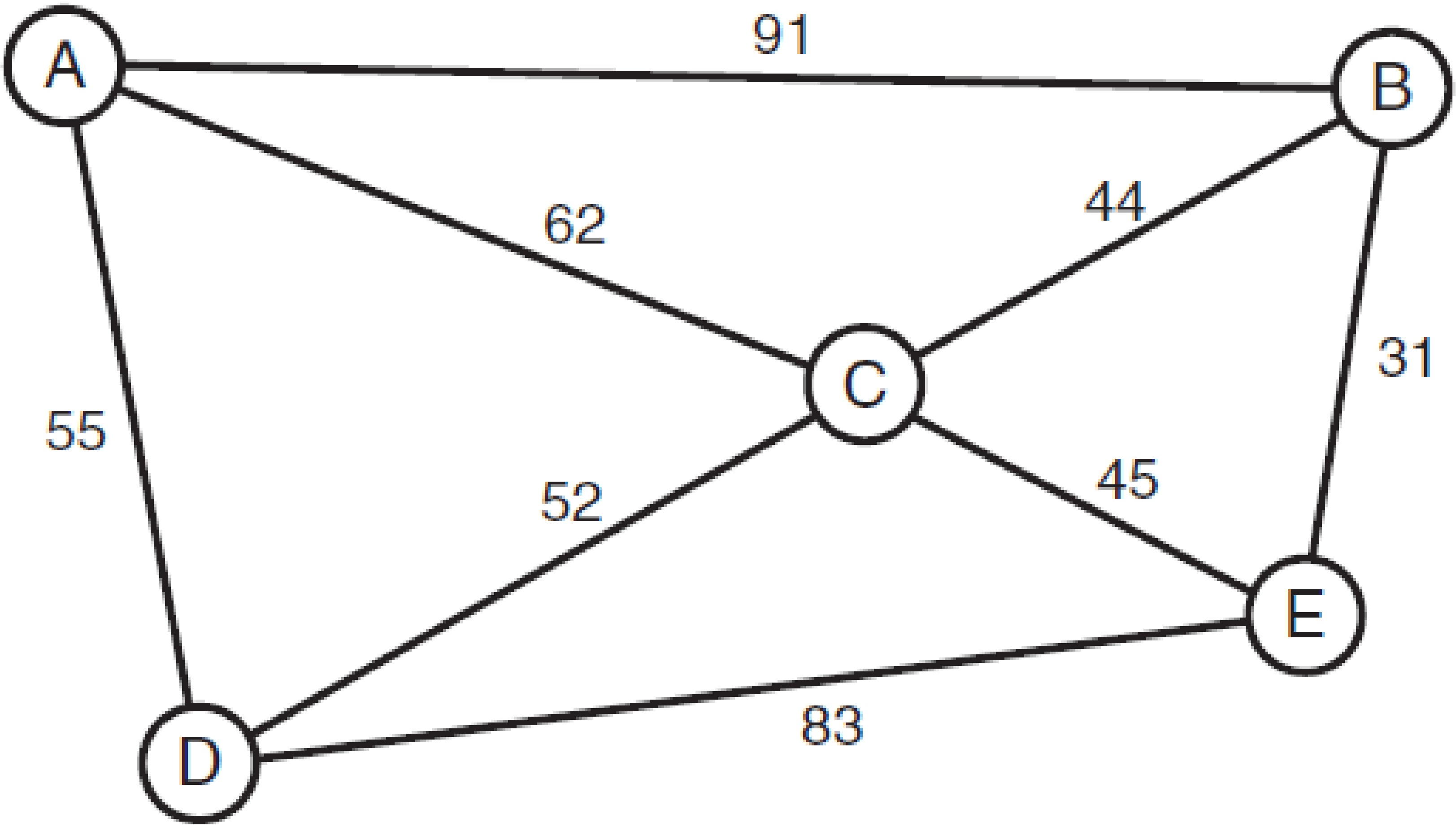
Problema calului la șah

- Se presupune o medie de numai 2 mișcări posibile din fiecare poziție
- Pornind din starea inițială, calul mai are de parcurs 63 de căsuțe
- În total sunt 2^{63} mișcări posibile, adică 10^{19}

Problema calului la șah

- Presupunând că un calculator poate face 10^6 mișcări într-o secundă, cum într-un an sunt 10^7 secunde, calculatorul poate face 10^{13} mișcări într-un an
- Rezolvarea acestei probleme prin “forță brută” necesită 10^6 ani
- Problema poate fi rezolvată folosind **funcții euristice**, care să elimine părți din arborele de joc

Problema comis-voiajorului



Problema comis-voiajorului

- Calea ABCEDA are lungimea totală 318
- Calea ABCDEA este imposibilă, deoarece nu există nicio muchie de la E la A
- Numărul de permutări este foarte mare – factorialul numărului de orașe
- Dacă trebuie vizitate 6 orașe, există 720 de căi posibile

Problema comis-voiajorului

- Există strategii care să reducă numărul de căi posibile
- Graful asociat problemei poate fi un graf ponderat sau un graf orientat și ponderat

Ciclu Hamiltonian

- Un ciclu hamiltonian vizitează fiecare vârf al grafului o singură dată
- Graful asociat este un graf neorientat și neponderat
- Calea ABCEDA este un ciclu hamiltonian
- Calea ABCDEA nu este un ciclu hamiltonian

Ciclu Hamiltonian

- Problema calului la șah poate fi privită ca un exemplu de determinare a ciclului hamiltonian (în cazul când calul se întoarce în căsuța inițială)
- Găsirea unui ciclu hamiltonian are complexitatea O(N!), la fel ca problema comis-voiajorului

Concluzii

- Într-un graf ponderat, fiecare muchie are asociat un număr, numit pondere
- Ponderile pot reprezenta distanțe, costuri, timpi sau alte mărimi
- Arborele minim de acoperire, într-un graf ponderat, minimizează ponderile muchiilor necesare pentru a conecta toate vârfurile

Concluzii

- Pentru determinarea arborelui minim de acoperire al unui graf, putem utiliza o coadă cu priorități
- Problema drumului minim într-un graf neponderat presupune determinarea numărului minim de muchii dintre două vârfuri

Concluzii

- Rezolvarea problemei drumului minim, în cazul grafurilor ponderate, se poate face utilizând algoritmul lui Dijkstra
- Rezolvarea problemei drumului de lungime minimă din orice vârf înseamnă găsirea costurilor totale ale muchiilor între toate perechile de vârfuri ale unui graf; această problemă se rezolvă folosind algoritmul lui Floyd