Algoritmul lui Kruskal

Introducere

- Descoperit de Joseph Kruskal în 1956
- Găsește arborele parțial de cost minim pentru un graf conex ponderat
- Găsește submulțimea muchiilor care formează un arbore care include toate vârfurile și care este minimal din punct de vedere al costului

Introducere

- Dacă graful nu este conex, atunci algoritmul găsește un arbore parțial de cost minim pentru fiecare componentă conexă
- Algoritmul lui Kruskal este un exemplu de algoritm **greedy**

Observații

- Este asemănător cu algoritmul lui **Prim**
- Algoritmul lui **Kruskal** are complexitatea **O(m*log m)**, unde **m** reprezintă numărul de muchii
- Algoritmul lui **Prim** are complexitatea **$O(n^2)$**, unde **n** este numărul de noduri

Observații

- Dacă **numărul de muchii este mic**, este preferabilă folosirea **algoritmului lui Kruskal**
- Dacă **numărul de muchii este mare** și numărul lor se apropie de $\mathbf{n^2}$, atunci este preferabilă folosirea **algoritmului lui Prim**

Pașii algoritmului

- Pasul **1** – Inițial, fiecare nod va fi un arbore; avem o pădure de **n** arbori
- Se execută de **n-1** ori pasul următor:
- Pasul **2** – Se caută muchia de cost minim care unește noduri care aparțin la doi arbori diferiți; se selectează muchia respectivă
- După selectarea a **n-1** muchii, se găsește arborele de cost minim

Detalii despre algoritm

- Pentru implementarea algoritmului este necesară rezolvarea următoarelor două probleme:
- 1) Cum extragem muchia de cost minim ?
- 2) Cum testăm dacă muchia selectată formează sau nu cicluri cu cele deja selectate ?

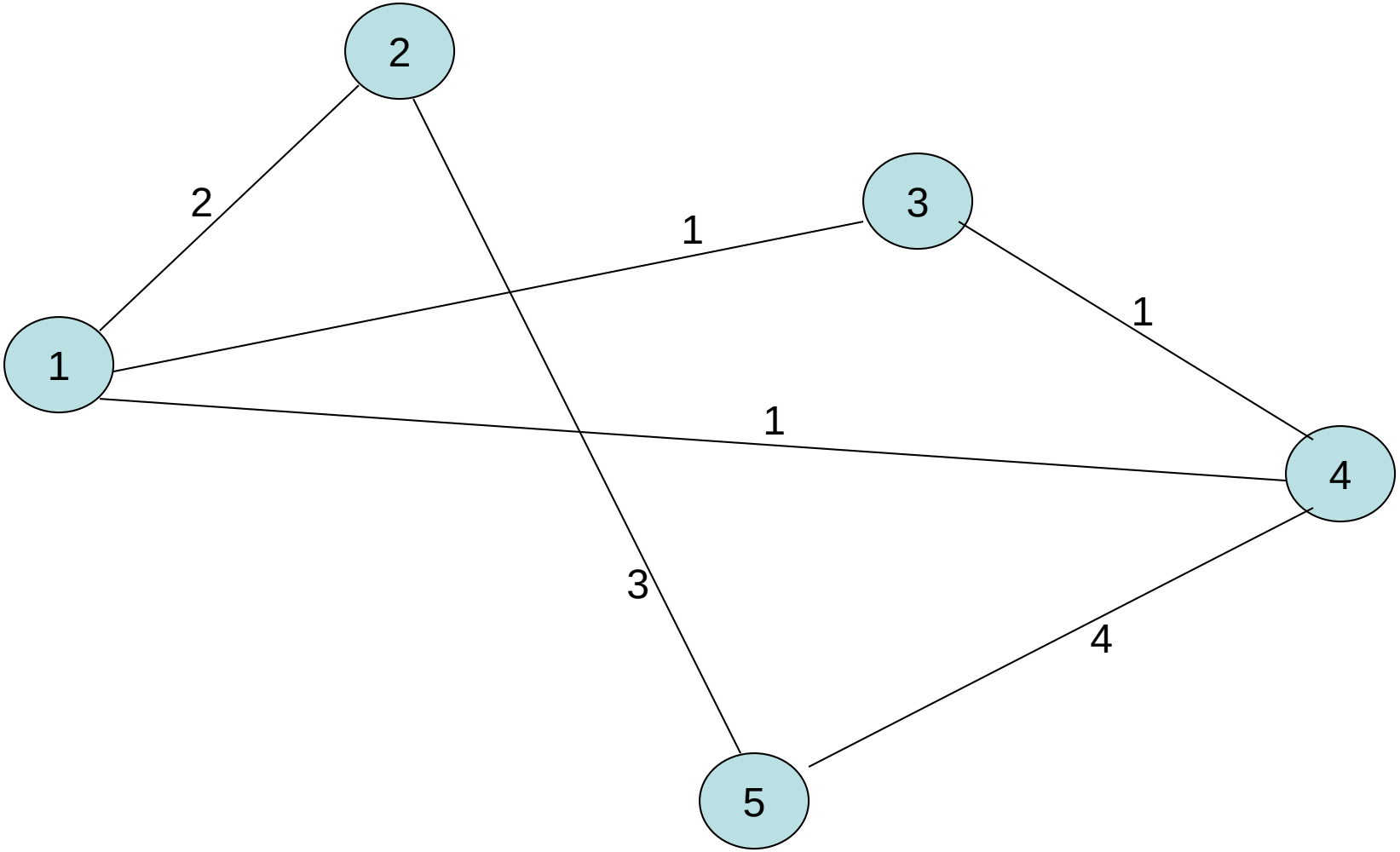
Observații

- Pentru a extrage muchia de cost minim, se sortează muchiile crescător după cost și se parcurg secvențial muchiile ordonate
- Pentru a realiza sortarea muchiilor grafului, se reprezintă graful prin lista muchiilor (un vector cu **m** componente, fiecare componentă fiind o structură în care se reține cele două extremități și costul muchiei)

Observații

- O muchie va forma **cicluri** cu muchiile deja selectate dacă și numai dacă între extremităţile muchiei există cel puțin un **lanț**
- Pentru a testa dacă o muchie formează cicluri cu muchiile deja selectate, se testează dacă extremităţile muchiei se găsesc în aceeași componentă conexă
- Pentru aceasta trebuie să ținem evidența componentelor conexe care se formează

Exemplu



Explicații

- Pasul **1**: Se selectează o muchie de cost minim – în acest caz, de cost 1
- Se observă că în graful parțial selectat există n - 1 = 4 arbori, pentru că se unifică arborii corespunzători extremităților muchiei selectate
- Arborii sunt: { 1, 3 }; { 2 }; { 4 }; { 5 }

Explicații

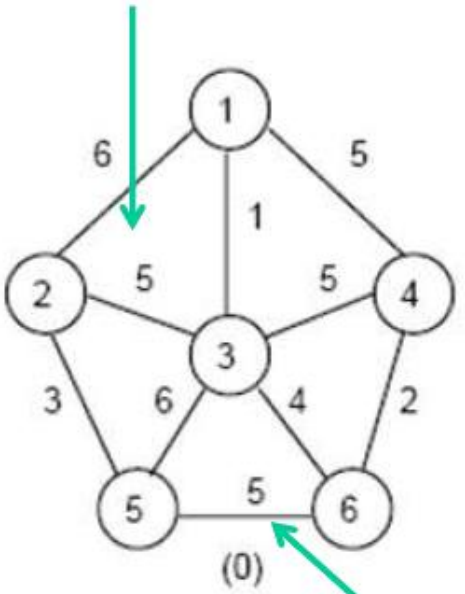
- Pasul **2**: Se selectează din nou o muchie de cost minim - costul minim fiind 1
- Se observă că în graful parțial selectat există n - 2 = 3 arbori
- Arborii sunt: { 1, 3, 4 }; { 2 }; { 5 }

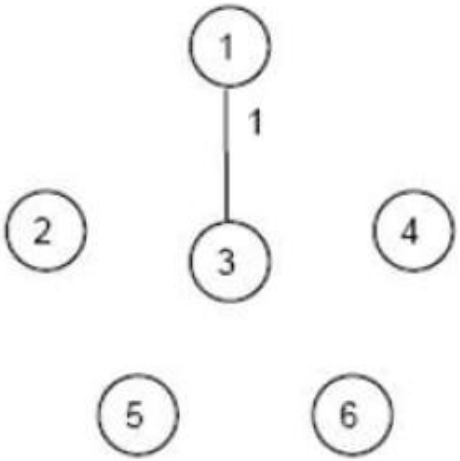
Explicații

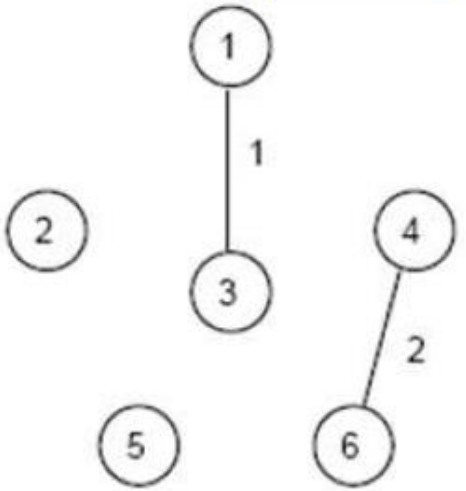
- Pasul **3**: La acest pas nu se mai poate selecta o muchie de cost 1, deoarece s-ar obține un ciclu
- Se selectează muchia de cost 2
- Arborii sunt: { 1, 2, 3, 4 }; { 5 }

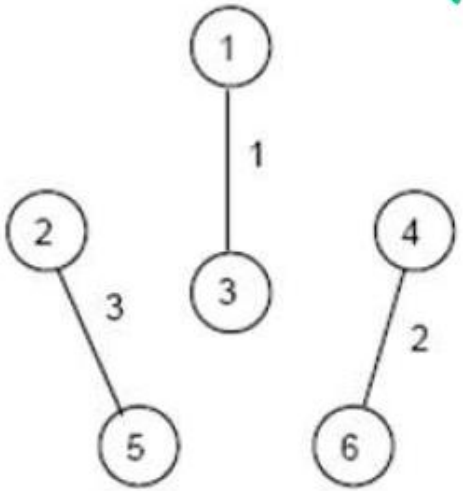
Explicații

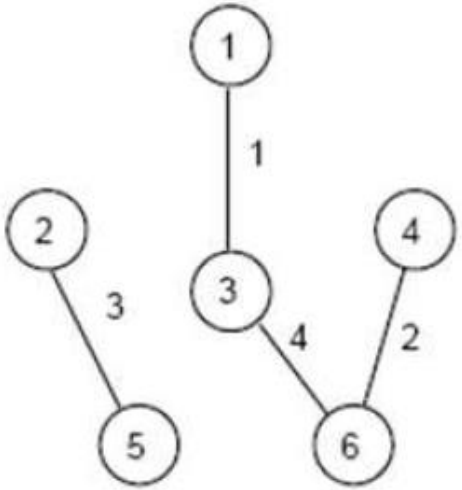
- Pasul **4**: Se selectează, în final, muchia de cost 3
- Se obține un graf, fără cicluri, cu n-1 muchii, deci un arbore

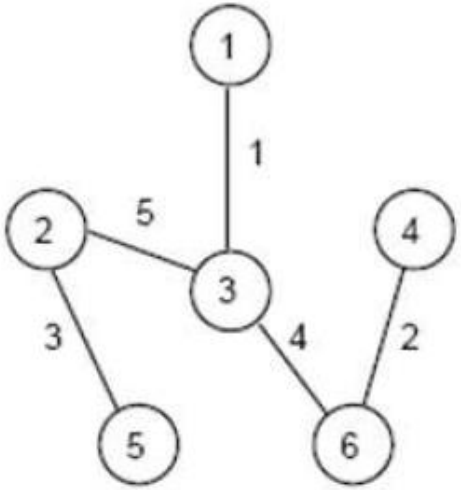

1
2
3
4
5
6
6
5
1
5
5
3
6
4
2
5
6
(0)


1
2
3
4
5
6
1
(1)


1
2
3
4
5
6
1
2
(2)


1
2
3
4
5
6
1
3
2
(3)


1
2
3
4
5
6
1
3
4
2
(4)


1
2
3
4
5
6
1
5
3
4
2
(5)

Detalii algoritm

- Se alege la fiecare pas muchia de cost minim dintre cele rămase încă neselectate, dacă ea nu formează un ciclu cu muchiile deja selectate
- Condiția ca o muchie **(x,y)** să nu formeze ciclu cu celelalte muchii selectate se poate exprima astfel: nodurile **x** și **y** trebuie să se afle în componente conexe diferite

Detalii algoritm

- Inițial, fiecare nod formează o componentă conexă, iar apoi o componentă conexă conține toate nodurile acoperite cu muchii din arborele minim de acoperire, iar nodurile neacoperite formează alte componente conexe
- Algoritmul folosește o **coadă cu priorități** și o **colecție de mulțimi disjuncte**

Pseudocod

- citire date si creare coadă de muchii
- ***repetă*** {
- extrage muchia de **cost minim** din coadă
- ***dacă*** muchia e acceptabilă ***atunci*** {
- afișare muchie
- actualizare componente conexe
- }
- } ***până când*** toate nodurile sunt conectate

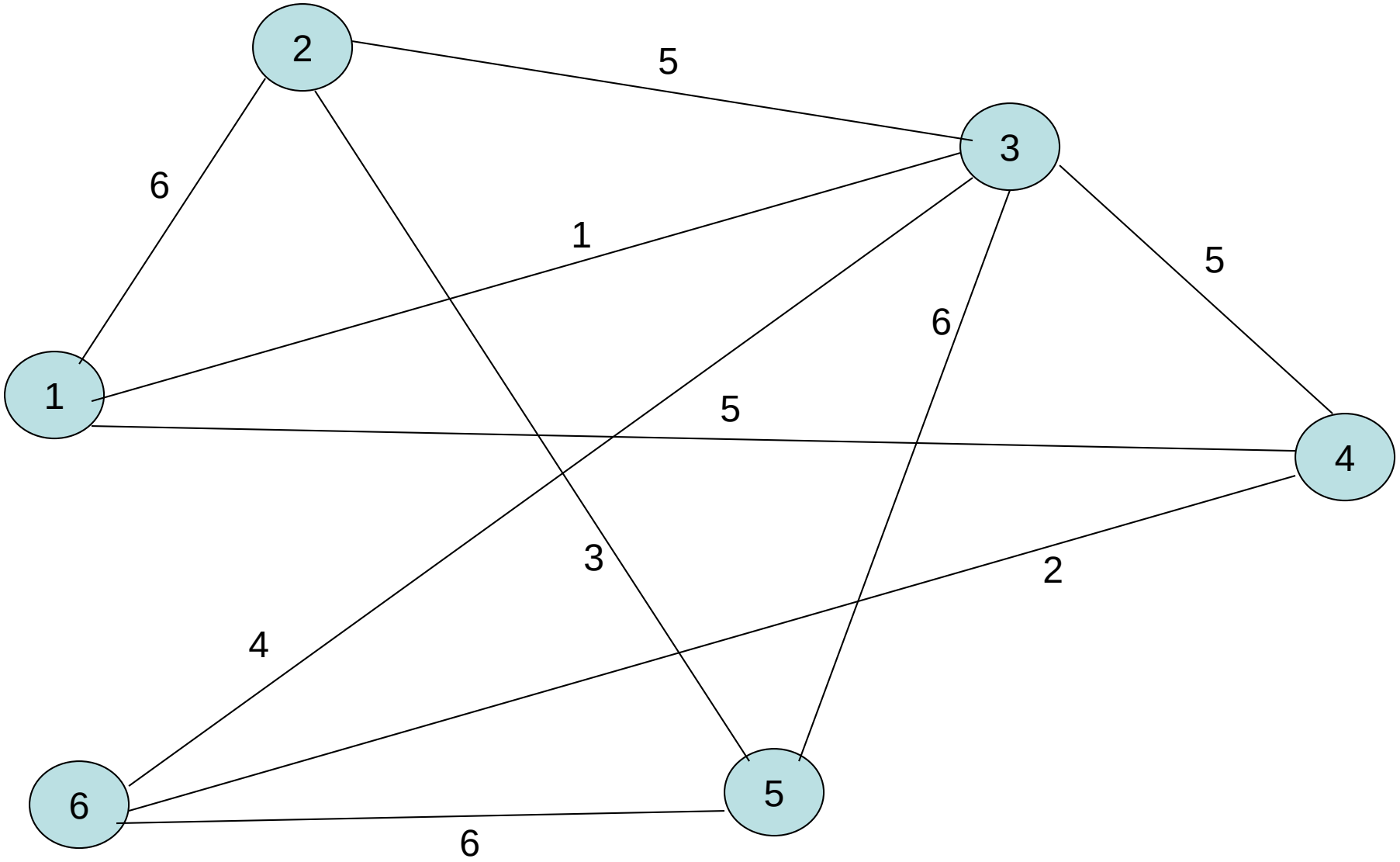
Observații

- O muchie care leagă două noduri dintr-o aceeași componentă conexă va forma un ciclu cu muchiile selectate anterior și nu poate fi acceptată
- Va fi acceptată numai o muchie care leagă între ele noduri aflate în două componente conexe diferite

Exemplu

- Pentru graful cu 6 noduri și 10 muchii, cu 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
- Evoluția algoritmului lui Kruskal este:

Exemplu



Pas	*Muchie (Cost)*	*Acceptabil*	*Cost total*	*Afișare*
1	1,3 (1)	da	1	1 - 3
2	4,6 (2)	da	3	4 - 6
3	2,5 (3)	da	6	2 - 5
4	3,6 (4)	da	10	3 - 6
5	1,4 (5)	nu	10	
6	3,4 (5)	nu	10	
7	2,3 (5)	da	15	2 - 3

Observații

- Toate nodurile din graf trebuie să se afle în componentele conexe asociate lor
- Inițial sunt atâtea componente câte noduri există
- Când o muchie este acceptată, se reunesc cele două componente care conțin extremitățile muchiei în una singură
- Numărul de componente conexe se reduce treptat, până când ajunge egal cu 1

Pas	*Componente conexe*
1	{1}, {2}, {3}, {4}, {5}, {6}
2	{1,3}, {2}, {4}, {5}, {6}
3	{1,3}, {2,5}, {4,6}
4	{1,3,4,6}, {2,5}
5	{1,3,4,6}, {2,5}
6	{1,3,4,6}, {2,5}
7	{1,2,3,4,5,6}