

1

Problema arborelui parțial de cost minim

- Metoda generală MST
- Algoritmul lui Prim
- Algoritmul lui Kruskal

2

Cuplaje

- Cuplaje maxime – acoperiri minime cu muchii
- Problema cuplajului maxim

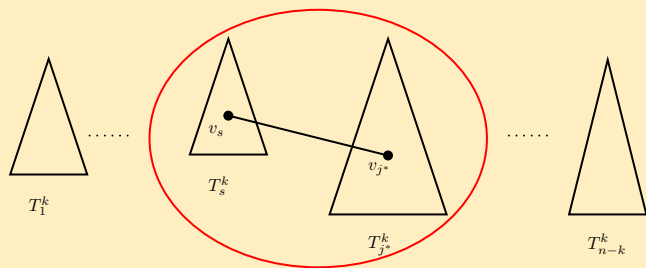
3

Exerciții pentru seminarul 7 (săptămâna 10-14 noiembrie)

- Se pornește cu familia $\mathcal{T}^0 = (T_1^0, T_2^0, \dots, T_n^0)$ de n arbori disjuncți:
 $T_i^0 = (\{i\}, \emptyset)$, $i = \overline{1, n}$.
- La fiecare pas k ($0 \leq k \leq n - 2$), din familia $\mathcal{T}^k = (T_1^k, T_2^k, \dots, T_{n-k}^k)$ de $(n - k)$ arbori disjuncți astfel încât $V = \bigcup_{i=1}^{n-k} V(T_i^k)$ și $\bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E$, construiește \mathcal{T}^{k+1} după cum urmează:
 - alege $T_s^k \in \mathcal{T}^k$;
 - determină o muchie de cost minim $e^* = v_s v_{j^*}$ din mulțimea de muchii ale lui G cu o extremitate $v_s \in V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$ ($v_{j^*} \in V(T_{j^*}^k)$);
 - $\mathcal{T}^{k+1} = (\mathcal{T}^k \setminus \{T_s^k, T_{j^*}^k\}) \cup T$, unde T este arborele obținut din T_s^k și $T_{j^*}^k$ prin adăugarea muchiei e^* .

Remarci

- Observăm că, dacă, la un anumit pas, nu există nicio muchie cu o extremitate în $V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$, atunci G nu este conex și nu există vreun MST în G .
- Construcția de mai sus este sugerată în imaginea de mai jos:



- Familia \mathcal{T}^{n-1} are doar un arbore, T_1^{n-1} .

Teorema 1

Dacă $G = (V, E)$ este un graf conex cu $V = \{1, 2, \dots, n\}$, atunci T_1^{n-1} construit de algoritmul anterior este un MST al lui G .

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

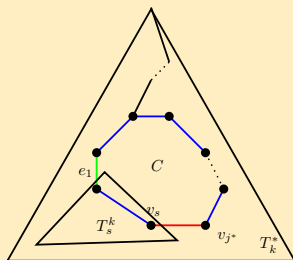
Demonstrație: Demonstrăm (prin inducție) că $(*) \forall k \in \{0, \dots, n-1\}$ există un arbore parțial T_k^* , MST al lui G , astfel încât

$$E(\mathcal{T}^k) = \bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E(T_k^*).$$

În particular, pentru $k = n-1$, $E(\mathcal{T}^{n-1}) = E(T_1^{n-1}) \subseteq E(T_{n-1}^*)$ implică $T_1^{n-1} = T_{n-1}^*$ și teorema este demonstrată.

Pentru $k = 0$, avem $E(\mathcal{T}^0) = \emptyset$ și, deoarece G este conex, există un MST T_0^* ; astfel, proprietatea $(*)$ este adevărată.

Demonstrație (continuare).



Pe de altă parte, deoarece $c(e^*) \leq c(e_1)$, avem $c(T^1) = c(T_k^*) + c(e^*) - c(e_1) \leq c(T_k^*)$.

Deoarece T_k^* este un MST al lui G , urmează că $c(T^1) = c(T_k^*)$, i. e., T^1 este un MST al lui G conținând toate muchiile din $E(\mathcal{T}^{k+1})$. Luând $T_{k+1}^* = T^1$ încheiem demonstrația teoremei. \square

Remarci

- Demonstrația de mai sus rămâne adevărată și pentru funcții de cost $c : \mathcal{T}_G \rightarrow \mathbb{R}$ care satisfac: $\forall T \in \mathcal{T}_G, \forall e \in E(T), \forall e' \notin E(T)$

$$c(e') \leq c(e) \Rightarrow c(T + e' - e) \leq c(T).$$

- În metoda generală prezentată, modul de alegere a arborelui T_s^k nu este precizat în amănunt. Vom discuta două strategii clasice de alegere a acestui arbore.
- Prima strategie alege T_s^k ca fiind arborele de ordin maxim din familia \mathcal{T}^k .
- În cea de-a doua strategie T_s^k este unul dintre cei doi arbori din familia \mathcal{T}^k , legați printr-o muchie de cost minim printre toate muchiile cu extremități în arbori diferiți ai familiei.

Algoritmul lui Prim

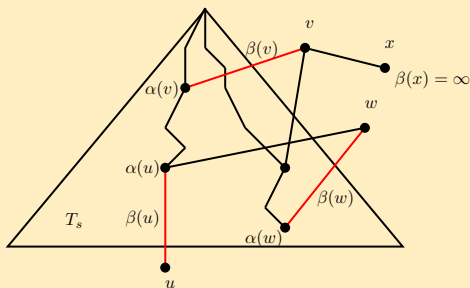
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- În strategia lui Prim T_s^k este arborele de ordin maxim din familia \mathcal{T}^k .
- Urmează că la fiecare pas $k > 0$ al metodei generale, \mathcal{T}^k are un arbore, $T_s^k = (V_s, E_s)$, cu $k + 1$ noduri și $n - k - 1$ arbori fiecare cu câte un nod.
- **Implementarea lui Dijkstra:** Fie α și β doi vectori de dimensiune n ; elementele lui α sunt noduri din $V(G)$ iar elementele lui β sunt numere reale, cu următoarea semnificație:

$$(S) \quad \forall j \in V \setminus V_s, \beta[j] = c(\alpha[j]j) = \min_{i \in V_s, ij \in E} c(ij)$$

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Remarci



- Se observă că (S) este satisfăcută după inițializare. În bucla while, strategia metodei generale este respectată și de asemeni semnificația lui (S) este păstrată de testul din bucla for.
- **Complexitatea timp:** $\mathcal{O}(n - 1) + \mathcal{O}(n - 2) + \dots + \mathcal{O}(1) = \mathcal{O}(n^2)$ care este bună pentru grafuri de dimensiune $\mathcal{O}(n^2)$.

- În algoritmul lui Kruskal T_s^k este unul dintre cei doi arbori din familia \mathcal{T}^k , legați printr-o muchie de cost minim printre toate muchiile cu extremități în arbori diferiți ai familiei.
- Această alegere poate fi făcută prin sortarea inițială a muchiilor crescător după cost și, după aceea, prin parcurgerea listei astfel obținute. Dacă notăm cu T arborele T_s^k , algoritmul poate fi descris astfel.

```
sort  $E = \{e_1, e_2, \dots, e_m\}$  a. î.  $c(e_1) \leq \dots \leq c(e_m)$ ;
```

```
 $T \leftarrow \emptyset$ ;  $i \leftarrow 1$ ;
```

```
while ( $i \leq m$ ) do
```

```
  if ( $\langle T \cup \{e_i\} \rangle_G$  nu are circuite) then
```

```
     $T \leftarrow T \cup \{e_i\}$ ;
```

```
   $i++$ ;
```

- Sortarea poate fi făcută în $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$.
- Pentru implementarea eficientă a testului din bucla while, este necesar să reprezentăm mulțimile de noduri ale arbori, $V(T_1^k), V(T_2^k), \dots, V(T_{n-k}^k)$ (la fiecare pas k al metodei generale), și să testăm dacă muchia curentă are ambele extremități în aceeași mulțime.
- Aceste mulțimi vor fi reprezentate folosind arbori (care nu sunt, în general, subarbori ai grafului G). Fiecare astfel de arbore are o rădăcină care va fi folosită pentru a desemna mulțimea de noduri ale grafului G din acel arbore.
- Mai precis, avem nevoie de o funcție $find(v)$ care determină cărei mulțimi îi aparține nodul v , adică, **returnează rădăcina arborelui care reține mulțimea lui v .**

Algoritmul lui Kruskal – Union-Find

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

- În metoda generală este necesară o reuniune (disjunctă) a mulțimilor de noduri a doi arbori (pentru a obține \mathcal{T}^{k+1}).
- Folosim metoda *union*(u, w) cu următoarea semnificație: realizează reuniunea a două mulțimi de noduri, una căreia îi aparține v și una căreia îi aparține w .
- Putem rescrie bucla while a algoritmului după cum urmează:

```
while ( $i \leq m$ ) do
  let  $e_i = vw$ ;
  if ( $find(v) \neq find(w)$ ) then
    union( $v, w$ );
     $T \leftarrow T \cup \{e_i\}$ ;
   $i++$ ;
```

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – Prima soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

- Tabloul $root[1..n]$ cu elemente din V are semnificația: $root[v] =$ rădăcina arborelui care reține mulțimea căreia îi aparține v .
- Adaugă la pasul de inițializare (corespunzând familiei \mathcal{T}^0):
for ($v \in V$) do
 $root[v] \leftarrow v$;
- Funcția *find* (cu complexitatea timp $\mathcal{O}(1)$):
function $find(v : V)$;
return $root[v]$;
- Metoda *union* (cu complexitatea timp $\mathcal{O}(n)$):
method $union(v, w : V)$;
for ($i \in V$) do
 if ($root[i] = root[v]$) then
 $root[i] = root[w]$;

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

- Tabloul $pred[1..n]$ cu elemente din $V \cup \{0\}$ are semnificația $pred[v] = \text{nodul dinaintea lui } v \text{ pe drumul unic către } v \text{ de la rădăcina arborelui care reține mulțimea căruia îi aparține } v$.
- Aduagă la pasul de inițializare (corespunzând familiei \mathcal{T}^0):
for ($v \in V$) do
 $pred[v] \leftarrow 0$;
- Funcția $find(v)$ are complexitatea în $\mathcal{O}(h(v))$, unde $h(v)$ este lungimea drumului din arbore de la v la rădăcina acestui arbore:
function $find(v : V)$;
 $i \leftarrow v$; // o variabilă locală.
 while ($pred[i] > 0$) do
 $i \leftarrow pred[i]$;
 return i ;

Algoritmul lui Kruskal – Union-Find – A doua soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- Metoda *union* (de complexitate timp $\mathcal{O}(1)$) este apelată doar pentru noduri rădăcină:

```
method union(root1, root2 : V)  
  pred[root1] ← root2;
```

- Bucla *while* a algoritmului este modificată astfel:

```
while (i ≤ m) do  
  let ei = vw; x ← find(w); y ← find(v);  
  if (x ≠ y) then  
    union(x, y);  
    T ← T ∪ {ei};  
  i ++;
```

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

- Astfel, această formă a algoritmului are complexitatea timp $\Omega(n^2)$ (chiar dacă graful este rar).
- Neajunsul acestei implementări este dat de faptul că, în metoda **union**, rădăcină a noului arbore devine rădăcina aceluia arbore care reține un număr mai mic de noduri, ceea ce implică o mărire a lui $h(v)$ la $\mathcal{O}(n)$ în timpul algoritmului.
- Putem evita acest neajuns ținând în rădăcina fiecărui arbore cardinalul mulțimii pe care arborele o reține. Mai precis, semnificația $pred[v]$, unde v este o rădăcină, este:

$pred[v] < 0 \Leftrightarrow v$ este rădăcina unui arbore

care reține o mulțime cu $-pred[v]$ noduri

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A doua soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- Pasul de inițializare

```
for ( $v \in V$ ) do  
     $pred[v] \leftarrow -1$ ;
```

- Metoda *union* are complexitatea $\mathcal{O}(1)$ pentru a întreține noua semnificație:

```
method union( $root_1, root_2 : V$ )  
     $t \leftarrow pred[root_1] + pred[root_2]$ ;  
    if ( $-pred[root_1] \geq -pred[root_2]$ ) then  
         $pred[root_2] \leftarrow root_1$ ;  $pred[root_1] \leftarrow t$ ;  
    else  
         $pred[root_1] \leftarrow root_2$ ;  $pred[root_2] \leftarrow t$ ;
```

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A doua soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Afirmație. Cu aceste implementări ale funcției `find` și metodei `union` algoritmul are următorul invariant:

$$(*) \forall v \in V, -pred[find(v)] \geq 2^{h(v)}$$

Cu alte cuvinte, numărul de noduri din arborele căruia îi aparține v este cel puțin 2 la puterea "distanța de la v la rădăcină".

Demonstrația afirmației. După pasul de inițializare avem $h(v) = 0$, $find(v) = v$, și $-pred[v] = 1$, $\forall v \in V$, deci $(*)$ are loc cu egalitate.

Să presupunem că $(*)$ are loc înaintea unei iterații din bucla `while`. Sunt posibile două cazuri:

- În această iterație `while` nu este apelată `union`. Tabloul `pred` nu este actualizat, deci $(*)$ are loc și după această iterație.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A doua soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- În această iterație **while** avem un apel al metodei **union**. Fie $union(x, y)$ acest apel, și să presupunem că în metoda **union** se execută atribuirea $pred[y] \leftarrow x$. Aceasta înseamnă că înaintea acestei iterații avem $-pred[x] \geq -pred[y]$.

Nodurile v pentru care $h(v)$ se modifică în această iterația sunt acelea pentru care, înaintea iterației aveam $find(v) = y$ și $-pred[y] \geq 2^{h(v)}$.

După iterația **while**, avem $h'(v) = h(v) + 1$ și $find'(v) = x$. Astfel, trebuie să verificăm că $-pred'[x] \geq 2^{h'(v)}$. Într-adevăr, $-pred'[x] = -pred[x] - pred[y] \geq 2 \cdot (-pred[y]) \geq 2 \cdot 2^{h(v)} = 2^{h(v)+1} = 2^{h'(v)}$.

Urmează că (*) este un invariant al algoritmului. \square

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A doua soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Aplicând logaritmul în (*) obținem

$$h(v) \leq \log(-\text{pred}[\text{find}(v)]) \leq \log n, \forall v \in V.$$

Complexitatea timp a buclei **while** este

$$\mathcal{O}(n - 1 + 2m \log n) = \mathcal{O}(m \log n).$$

Astfel, această a doua implementare pentru **union** și **find** oferă o complexitate timp a algoritmului Kruskal de $\mathcal{O}(m \log n)$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Complexitatea timp a buclei `while` din soluția de mai sus este datorată șirului de apeluri `find`. Tarjan (1976) a observat că un apel cu $h(v) > 1$ poate să "colapseze" drumul din arbore de la v la rădăcină, fără a modifica timpul $\mathcal{O}(h(v))$, făcând $h(x) = 1$ pentru toate nodurile x de pe drum. În acest fel, viitoarele apeluri `find` pentru aceste noduri vor fi mai rapide. Mai precis, funcția `find` devine:

```
function find( $v : V$ ); //  $i, j, aux$  sunt variabile locale.  
 $i \leftarrow v$ ;  
while ( $pred[i] > 0$ ) do  
     $i \leftarrow pred[i]$ ;  
 $j \leftarrow v$ ;  
while ( $pred[j] > 0$ ) do  
     $aux \leftarrow pred[j]$ ;  $pred[j] \leftarrow i$ ;  $j \leftarrow aux$ ;  
return  $i$ ;
```

Dacă $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ este funcția lui Ackermann dată prin:

$$(1) \quad A(m, n) = \begin{cases} n + 1, & \text{dacă } m = 0 \\ A(m - 1, 1), & \text{dacă } m > 0 \text{ și } n = 0 \\ A(m - 1, A(m, n - 1)), & \text{dacă } m > 0 \text{ și } n > 0 \end{cases}$$

și dacă notăm, $\forall m \geq n > 0$,

$$\alpha(m, n) = \min \{z : A(z, 4 \lceil m/n \rceil) \geq \log n, z \geq 1\}$$

obținem că **complexitatea timp a buclei while folosind union din cea de-a doua soluție și find de mai sus, devine $\mathcal{O}(m \cdot \alpha(m, n))$.**

Să notăm că $\alpha(m, n)$ este o funcție care crește foarte încet (pentru valori practice ale lui n , $\alpha(m, n) \leq 3$); astfel cea de-a treia soluție este practic o implementare liniară ($\mathcal{O}(m)$) a algoritmului lui Kruskal.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Fie $G = (V, E)$ un (multi)graf. Dacă $A \subseteq E$ și $v \in V$, notăm $d_A(v) = |\{e : e \in A, e \text{ incidentă cu } v\}|$, i. e., gradul lui v în subgraful generat de A , $\langle A \rangle_G$.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Definiție

Un cuplaj (o mulțime independentă de muchii) în G este o mulțime de muchii $M \subseteq E$ astfel încât

$$d_M(v) \leq 1, \forall v \in V.$$

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.

Familia tuturor cuplajelor din graful G este notată cu \mathcal{M}_G :

$$\mathcal{M}_G = \{M : M \subseteq E, M \text{ cuplaj în } G\}.$$

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Se observă că \mathcal{M}_G satisface:

- (i) $\emptyset \in \mathcal{M}_G$;
- (ii) $M \in \mathcal{M}_G, M' \subseteq M \Rightarrow M' \in \mathcal{M}_G$.

Fie $M \in \mathcal{M}_G$ un cuplaj.

- Un nod $v \in V$ cu $d_M(v) = 1$ este numit **saturat** de către M , iar mulțimea tuturor nodurilor lui G saturate de M este notată cu $S(M)$. Evident,

$$S(M) = \bigcup_{e \in M} e, \text{ și } |S(M)| = 2 \cdot |M|.$$

- Un nod $v \in V$ cu $d_M(v) = 0$ este numit **expus** relativ la M (sau **nesaturat** de către M), iar mulțimea tuturor nodurilor lui G expuse relativ la M este notată cu $E(M)$. Evident că $E(M) = V \setminus S(M)$, și $|E(M)| = |V| - 2 \cdot |M|$.

Definiție

O acoperire cu muchii a lui G este o mulțime de muchii $F \subseteq E$ astfel încât

$$d_F(v) \geq 1, \forall v \in V(G).$$

Familia acoperirilor cu muchii ale grafului G este notată cu \mathcal{F}_G :

$$\mathcal{F}_G = \{F : F \subseteq E, F \text{ acoperire cu muchii a lui } G\}.$$

\mathcal{F}_G are următoarele proprietăți

- (i) $\mathcal{F}_G \neq \emptyset \Leftrightarrow G$ nu are noduri izolate (caz în care $E \in \mathcal{F}_G$);
- (ii) $F \in \mathcal{F}_G, F' \supseteq F \Rightarrow F' \in \mathcal{F}_G$.

Problem acoperirii minime cu muchii:

P_2 dat un graf $G = (V, E)$, găsiți $F^* \in \mathcal{F}_G$ astfel încât

$$|F^*| = \min_{F \in \mathcal{F}_G} |F|.$$

Teorema 2

(Norman-Rabin, 1959) Fie $G = (V, E)$ un graf de ordin n , fără noduri izolate. Dacă M^* este un cuplaj de cardinal maxim în G și F^* este o acoperire minimă cu muchii a lui G , atunci

$$|M^*| + |F^*| = n.$$

Demonstrație: " \leq " Fie M^* un cuplaj de cardinal maxim în G ; considerăm următorul algoritm:

$F \leftarrow M^*$;

for ($v \in E(M^*)$) do

 find $v' \in S(M^*)$ a. î. $vv' \in E$;

$F \leftarrow F \cup \{vv'\}$;

Demonstrație (continuare).

Să notăm că, $\forall v \in E(M^*)$, deoarece G nu are noduri izolate, există o muchie incidentă cu v , și, cum M^* este maximal relativ la relația de incluziune, această muchie are celălalt capăt în $S(M^*)$ (altfel putem adăga e la M^* și obținem un cuplaj de cardinal mai mare decât M^*).

Mulțimea F de muchii construită este o acoperire cu muchii și $|F| = |M^*| + |E(M^*)| = |M^*| + n - 2 \cdot |M^*| = n - |M^*|$. Astfel

$$(2) \quad |F^*| \leq |F| = n - |M^*|.$$

" \geq " Fie F^* o acoperire minimă cu muchii a lui G ; considerăm următorul algoritm:

```

M ← F*;
for (∃v ∈ V : d_M(v) > 1) do
  find e ∈ M incidentă cu v;
  M ← M \ {e};

```

Demonstrație (continuare).

Evident că algoritmul construiește un cuplaj M în G . Dacă muchia e incidentă cu v (eliminată din M într-o iterație **while**) este $e = vv'$, atunci $d_M(v') = 1$ și în următoarea iterație vom avea $d_M(v') = 0$; astfel, la fiecare iterație **while** este creat un nod expus relativ la cuplajul final, M (dacă ar exista o altă muchie e' , în mulțimea curentă M , incidentă cu v' , atunci deoarece $e \in F^*$, $F^* \setminus \{e\}$ ar fi o acoperire cu muchii de cardinal mai mic decât F^*).

Astfel, dacă M este cuplajul construit de algoritm, avem: $|F^*| - |M| = |E(M)| = n - 2 \cdot |M|$, i. e.,

$$(3) \quad |F^*| = n - |M| \geq n - |M^*|.$$

Din (2) și (3) derivă concluzia teoremei. \square

Problema cuplajului maxim

Fie $G = (V, E)$ un graf și \mathcal{M}_G familia cuplajelor sale.

Problem cuplajului maximum.

P_1 dat un graf $G = (V, E)$, determinați $M^* \in \mathcal{M}_G$ astfel încât

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

- Folosind formularea ca problemă de programare întreagă liniară problema P_1 este ușoară dacă graful este bipartit.
- Adaptări combinatoriale ale algoritmului simplex pentru rezolvarea problemei de programare liniară au condus la așa numita “metodă ungară” pentru rezolvarea P_1 în grafuri bipartite.
- Vom discuta o soluție mai rapidă găsită de Hopcroft și Karp (1973).
- Cu toate acestea, teorema de dualitate din programarea liniară și integralitatea soluției optime pot fi folosite pentru a obține și explica rezultatele (deja dovedite) despre cuplaje în grafuri bipartite:

Problema cuplajului maxim

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Teorema 3

(Hall, 1935) Fie $G = (S, T; E)$ un graf bipartit. Există un cuplaj în G care saturează toate nodurile din S dacă și numai dacă

$$|N_G(A)| \geq |A|, \forall A \subseteq S.$$

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

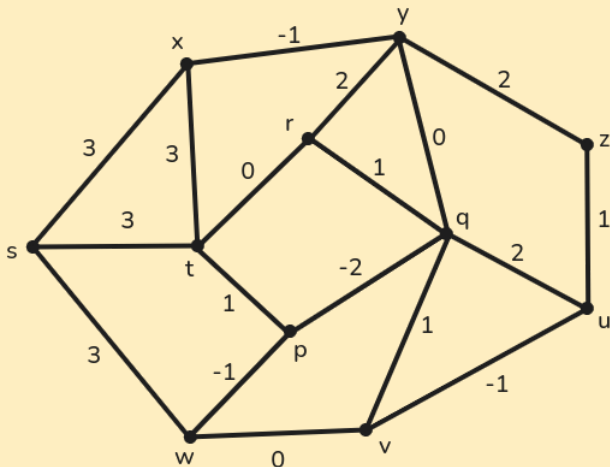
Teorema 4

(Konig, 1930) Fie $G = (S, T; E)$ un graf bipartit. Cardinalul maxim al unui cuplaj în G este egal cu cardinalul minim al unei acoperiri cu noduri a lui G , $\nu(G) = n - \alpha(G)$.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul 7

Exercițiul 1'. Determinați un arbore parțial de cost minim în graful de mai jos.



Exercițiul 1. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost pe muchiile sale. O submulțime $A \subseteq E$ este numită *tăietură* dacă există o bipartiție (S, T) a lui V astfel încât $A = \{uv \in E : u \in S, v \in T\}$ ($G \setminus A$ nu mai este conex).

- Dacă în orice tăietură există o singură muchie de cost minim, atunci G conține un singur arbore parțial de cost minim.
- Arătați că, dacă c este funcție injectivă, atunci G conține un singur arbore parțial de cost minim.
- Reciprocele afirmațiilor de mai sus sunt adevărate?

Exercițiul 2. Fie $G = (V, E)$ un graf conex de ordin n , $c : E \rightarrow \mathbb{R}$, și \mathcal{T}_G^{min} familia arborilor săi parțiali de cost (c) minim. Definim $H = (\mathcal{T}_G^{min}, E(H))$ unde $T_1 T_2 \in E(H) \iff |E(T_1) \Delta E(T_2)| = 2$. Arătați că H este conex și că diametrul său este cel mult $n - 1$.

Exerciții pentru seminarul 7

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exercițiul 3. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$. Pentru un arbore parțial $T = (V, E') \in \mathcal{T}_G$, și $v \neq w \in V$ notăm cu P_{vw}^T singurul vw -drum din T . Arătați că un arbore parțial $T^* = (V, E^*)$ este de cost minim dacă și numai dacă

$$\forall e = vw \in E \setminus E^*, \forall e' \in E(P_{vw}^{T^*}), \text{ avem } c(e) \geq c(e').$$

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exercițiul 4. Fie $G = (V, E)$ un graf 2-muchie-conex și $c : E \rightarrow \mathbb{R}$. Dacă $T = (V, E')$ este arbore parțial de cost minim al lui G și $e \in E'$, $T - e$ are exact două componente conex T'_1 și T'_2 , respectiv. Notăm cu $e_T \neq e$ o muchie de cost minim în tăietura generată de $(V(T'_1), V(T'_2))$ în $G - e$. Arătați că, dacă T^* este un arbore parțial de cost minim al lui G , și $e \in E(T^*)$, atunci $T^* - e + e_{T^*}$ este un arbore parțial de cost minim $G - e$.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Exercițiul 5. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost injectivă pe muchiile sale. Considerăm următorul algoritm

for ($e \in E$) do

$\gamma(e) \leftarrow r$; // toate muchiile sunt colorate cu **roșu**; în timpul execuției vor fi **roșii**, **albastre** sau **verzi**

while ($(\exists A \subseteq E$, tăietură cu $\gamma(e') \neq v$, unde $c(e') = \min_{e \in A} c(e)$) sau

$(\exists C$, un circuit cu $\gamma(e') \neq a$, unde $c(e') = \max_{e \in C} c(e)$)) do

 pentru o tăietură, A , $\gamma(e') \leftarrow v$;

 pentru un circuit, C , $\gamma(e') \leftarrow a$;

return $H = (V, \{e \in E : \gamma(e) = v\})$;

Arătați că

- o muchie aparține unui arbore parțial de cost minim dacă și numai dacă este de cost minim într-o tăietură;
- o muchie nu aparține nici unui arbore parțial de cost minim dacă și numai dacă este de cost maxim într-un circuit;

Exercițiul 5 (continuare).

- (c) algoritmul nu se oprește câtă vreme mai există muchii roșii în graf;
- (d) algoritmul se oprește pentru orice alegere a muchiilor e' , iar H este singurul arbore parțial de cost minim din G .

Exercițiul 6. Fie H un graf conex, $\emptyset \neq A \subseteq V(H)$, și $w : E(H) \rightarrow \mathbb{R}_+$. Un arbore Steiner pentru (H, A, w) este un arbore $T(H, A, w) = (V_T, E_T) \subseteq H$ cu $A \subseteq V_T$ care are costul minim printre toți arborii care conțin A , și care sunt subgrafuri ale lui H :

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) =$$

$$= \min \left\{ \sum_{e \in E_{T'}} w(e) : T' = (V_{T'}, E_{T'}) \text{ arbore în } H, A \subseteq V_{T'} \right\}$$

- (a) Arătați că un arbore Steiner poate fi determinat în timp polinomial dacă $A = V(H)$ sau $|A| = 2$.

Exercițiul 6 (continuare).

- (b) Fie $G = (V, E)$ un graf conex cu $V = \{1, 2, \dots, n\}$, și $A \subseteq V$; avem de asemenea și o funcție de cost $c : E \rightarrow \mathbb{R}_+$. Considerăm graful complet K_n (cu $V(K_n) = V$) și definim $\bar{c} : E(K_n) \rightarrow \mathbb{R}_+$:

$$\bar{c}(ij) = \min \left\{ c(P) = \sum_{e \in E(P)} c(e) : P \text{ este } ij\text{-drum în } G \right\}$$

Demonstrați că $s[T(G, A, c)] = s[T(K_n, A, \bar{c})]$ și arătați cum se poate construi un arbore Steiner $T(K_n, A, \bar{c})$ dintr-un arbore Steiner $T(G, A, c)$.

- (c) Arătați că există un arbore Steiner $T(K_n, A, \bar{c})$ astfel încât toate nodurile sale din afara lui A au gradul cel puțin 3. Folosind această proprietate arătați că există un arbore Steiner $T(K_n, A, \bar{c})$ cu cel mult $2|A| - 2$ noduri.

Exercițiul 7. Considerăm o ordonare $E = \{e_1, e_2, \dots, e_m\}$ a muchiilor unui graf conex $G = (V, E)$ de ordin n . Pentru orice submulțime $A \subseteq E$ definim $x^A \in GF^m$ vectorul caracteristic m -dimensional al mulțimii A : $x_i^A = 1 \Leftrightarrow e_i \in A$. GF^m este spațiul m -dimensional peste \mathbb{Z}_2 .

- Arătați că submulțimea vectorilor caracteristici corespunzători tuturor tăieturilor din G împreună cu vectorul nul este un subspațiu X al lui GF^m .
- Arătați că submulțimea vectorilor caracteristici corespunzători tuturor circuitelor din G generează un subspațiu U al lui GF^m care este ortogonal pe X .
- Arătați că $\dim(X) \geq n - 1$.
- Arătați că $\dim(U) \geq m - n + 1$.
- În final, dovediți că inegalitățile de mai sus sunt de fapt egalități.

Exercițiul 8. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost pe muchiile sale.

a) Fie T^* un arbore parțial de cost c -minim al lui G și $\epsilon > 0$. Arătați că T^* este singurul arbore parțial de cost \bar{c} -minim al lui G , unde

$$\bar{c}(e) = \begin{cases} c(e) - \epsilon, & \text{dacă } e \in E(T^*) \\ c(e), & \text{altfel} \end{cases}$$

b) Deduceți de aici că pentru orice arbore parțial de cost minim, T^* , al lui G există o ordonare a muchiilor lui G astfel încât algoritmul lui Kruskal returnează T^* .

Exercițiul 9. Fie G un graf, $s, t \in V$ și $c : E(G) \rightarrow \mathbb{R}_+$ o funcție de cost definită pe muchiile sale. Pentru orice drum (cu cel puțin o muchie) definim *locul îngust* ca fiind costul minim al uneia dintre muchiile sale. Proiectați un algoritm eficient care să determine un st -drum cu *locul îngust* maxim (printre toate st -drumurile din G).

Exerciții pentru seminarul 7

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 11. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost. Adevărat sau fals? (Justificați răspunsurile!)

- (a) Orice muchie de cost minim din G este conținută într-un anumit arbore parțial de cost minim din G .
- (b) Dacă G are un circuit, C , cu o singură muchie de cost minim, atunci acea muchie este conținută în orice arbore parțial de cost minim din G .
- (c) Dacă o muchie este conținută într-un arbore parțial de cost c minim din G , atunci acea muchie este de cost minim într-o anumită tăietură a lui G .

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul 7

Exercițiul 12. Determinați numărul de cuplaje maxime ale următorului graf:



Exercițiul 13. Doi copii se joacă pe un graf dat, G , astfel: fiecare alege alternativ un nod nou v_0, v_1, \dots așa încât, pentru orice $i > 0$, v_i este adiacent cu v_{i-1} . Jucătorul care nu mai poate alege un nod nou pierde jocul. Arătați că jucătorul care începe jocul are întotdeauna o strategie de câștig dacă și numai dacă G nu are un cuplaj perfect.

Exercițiul 14. Fie S o mulțime nevidă și finită, $k \in \mathbb{N}^*$, iar $\mathcal{A} = (A_i)_{1 \leq i \leq k}$ și $\mathcal{B} = (B_i)_{1 \leq i \leq k}$ două partiții ale lui S . Arătați că \mathcal{A} și \mathcal{B} admit un sistem comun de reprezentanți, i. e., există $r_{\mathcal{A}}, r_{\mathcal{B}} : \{1, 2, \dots, k\} \rightarrow S$ așa încât pentru orice $1 \leq i \leq k$, $r_{\mathcal{A}}(i) \in A_i$ și $r_{\mathcal{B}}(i) \in B_i$, iar cele două funcții au aceeași imagine.

Exerciții pentru seminarul 7

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Exercise 15. Let X be a finite set, $X_1, \dots, X_n \subseteq X$, and $d_1, d_2, \dots, d_n \in \mathbb{N}$. Prove that there are n disjoint subsets $Y_i \subseteq X_i$, $|Y_i| = d_i$, $\forall i = \overline{1, n}$ if and only if

$$\left| \bigcup_{i \in I} X_i \right| \geq \sum_{i \in I} d_i,$$

for all $I \subseteq \{1, \dots, n\}$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Exercise 16. Every p -regular bipartite graph has a perfect matching ($p \geq 1$).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Exercise 17. Let $G = (S, T; E)$ a bipartite graph. Use Hall's theorem to prove that, for every $0 \leq k \leq |S|$, G has a matching of cardinality at least $|S| - k$ if and only if $|N_G(A)| \geq |A| - k$, $\forall A \subseteq S$.