Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

3. Scheme de algoritmi - Greedy

Bibliografie

- Cormen – Introducere în Algoritmi cap. 17
- Giumale – Introducere in Analiza Algoritmilor cap 4.4 ,4.5
- http://www.cs.umass.edu/~barring/cs611/lecture/4.pdf
- http://thor.info.uaic.ro/~dlucanu/cursuri/tpaa/resurse/Curs6.pps
- http://www.math.fau.edu/locke/Greedy.htm
- http://en.wikipedia.org/wiki/Greedoid
- http://activities.tjhsst.edu/sct/lectures/greedy0607.pdf
- http://www.cse.ust.hk/~dekai/271/notes/L12/L12.pdf

Greedy (I)

- Metodă de rezolvare eficientă a unor probleme de optimizare.
- Soluția trebuie să satisfacă un criteriu de optim global (greu de verificat) → optim local mai ușor de verificat.
- Se aleg soluții parțiale ce sunt îmbunătățite repetat pe baza criteriului de optim local până ce se obțin soluții finale.
- Soluțiile parțiale ce nu pot fi îmbunătățite sunt abandonate → proces de rezolvare irevocabil (fără reveniri)!

Greedy (II)

- Schema generală de rezolvare a unei probleme folosind Greedy (programarea lacomă):

- Rezolvare_lacomă(Crit_optim, Problemă)
  - 1. sol_parțiale = sol_inițiale(Problemă); // determinarea soluțiilor parțiale
  - 2. sol_fin = Φ;
  - 3. **Cât timp** (sol_parțiale ≠ Φ)
  - 4. **Pentru fiecare** (s **in** sol_parțiale)
  - 5. **Dacă** (s este o soluție a problemei) { // dacă e soluție
  - 6. sol_fin = sol_fin U {s}; // finală se salvează
  - 7. sol_parțiale = sol_parțiale \ {s};
  - 8. } **Altfel** // se poate optimiza?
  - 9. **Dacă** (optimizare_posibilă (s, Crit_optim, Problemă))
  - 10. sol_parțiale = sol_parțiale \ {s} U // da optimizare(s,Crit_optim,Problemă)
  - 11. **Altfel** sol_parțiale = sol_parțiale \ {s}; // nu
  - 12. **Întoarce** sol_fin;

Arbori Huffman

- Metodă de codificare folosită la compresia fișierelor.
- Construcția unui astfel de arbore se realizează printr-un algoritm greedy.
- Considerăm un text, de exemplu:
  - **ana are mere**
- Vom exemplifica pas cu pas construcția arborelui de codificare pentru acest text si vom defini pe parcurs conceptele de care avem nevoie.

PA

Arbori Huffman – Definitii (I)

- K – mulțimea de simboluri ce vor fi codificate.

- Arbore de codificare a cheilor K este un arbore binar ordonat cu proprietățile:
  - Doar frunzele conțin cheile din K; nu există mai mult de o cheie intr-o frunză;
  - Toate nodurile interne au exact 2 copii;
  - Arcele sunt codificate cu 0 si 1 (arcul din stânga unui nod – codificat cu 0).

- k = Codul unei chei – este șirul etichetelor de pe calea de la rădăcina arborelui la frunza care conține cheia k (k este din K).

- p(k) – frecvența de apariție a cheii k in textul ce trebuie comprimat.

- Ex pentru "ana are mere":
  - p(a) = p(e) = 0.25; p(n) = p(m) = 0.083;p(r) = p( ) = 0.166

Arbori Huffman – Definitii (II)

- A – arborele de codificare a cheilor.
- lg_cod(k) – lungimea codului cheii k conform A.
- nivel(k,A) – nivelul pe care apare in A frunza ce conține cheia K.
- Costul unui arbore de codificare A al unor chei K relativ la o frecventa p este:

$$Cost(A) = \sum_{k \in K} lg_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$

- Un arbore de codificare cu cost minim al unor chei K, relativ la o frecventa p este un arbore Huffman, iar codurile cheilor sunt coduri Huffman.

PA

Arbori Huffman – algoritm de construcție (I)

- 1. pentru fiecare k din K se construiește un arbore cu un singur nod care conține cheia k si este caracterizat de ponderea w = p(k). Subarborii construiți formează o mulțime numită Arb.

- 2. Se aleg doi subarbori a şi b din Arb astfel încât a şi b au pondere minimă.

PA

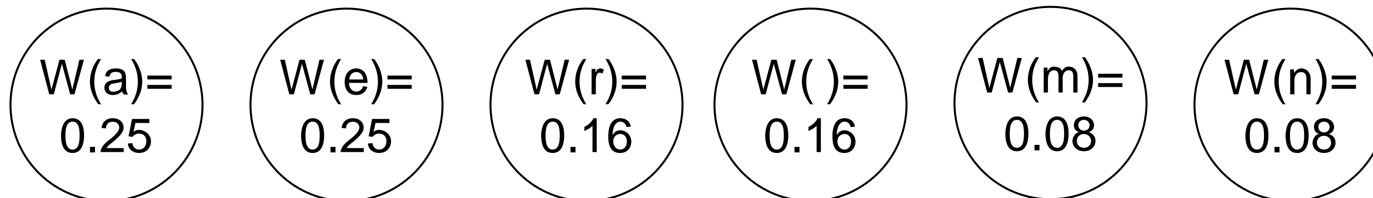
Arbori Huffman – algoritm de construcție (II)

- 3. Se construiește un arbore binar cu o rădăcină *r* care nu conține nici o cheie si cu descendenții a si b. Ponderea arborelui este definită ca w(r) = w(a) + w(b).
- 4. Arborii a si b sunt eliminați din Arb iar r este inserat in Arb.
- 5. Se repeta procesul de construcție descris de pașii 2-4 până când mulțimea Arb conține un singur arbore – Arborele Huffman pentru cheile K.

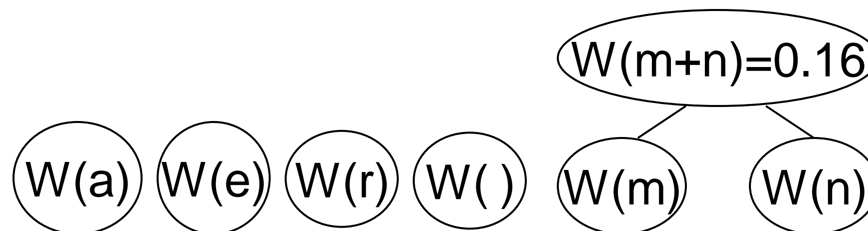
PA

Arbori Huffman – Exemplu (I)

- Text: ana are mere
- p(a) = p(e) = 0.25; p(n) = p(m) = 0.083; p(r) = p( ) = 0.166
- Pasul 1

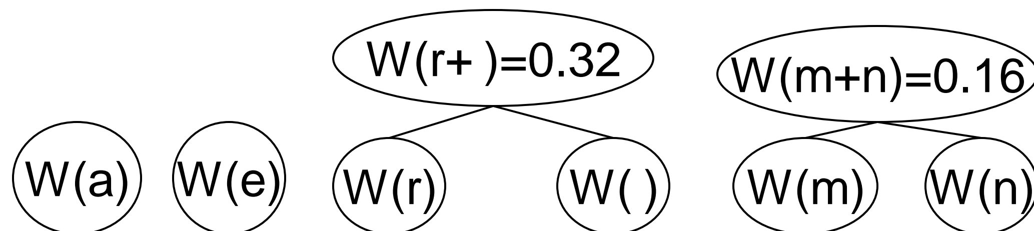


- Pasii 2-4

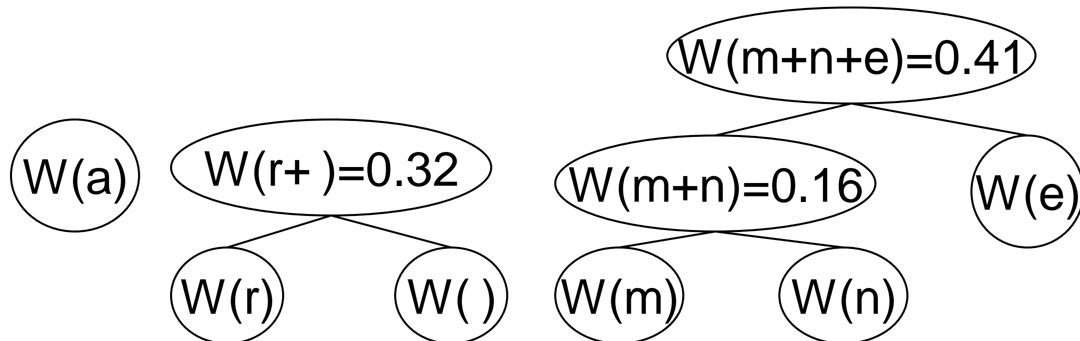


Arbori Huffman – Exemplu (II)

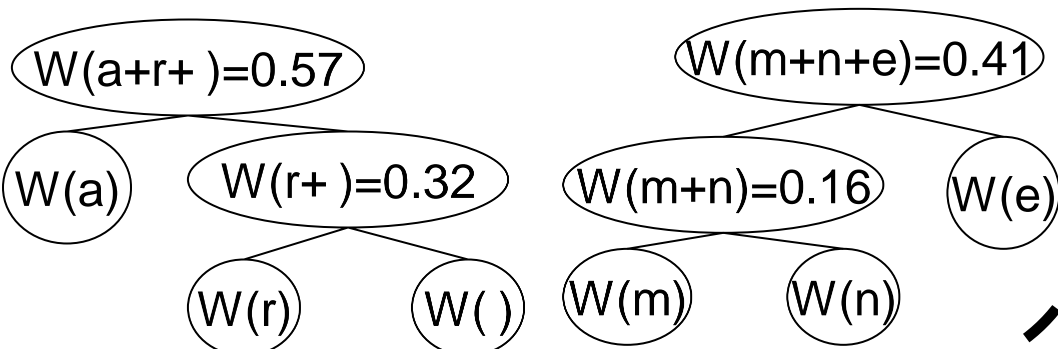
- Pasii 2-4 (II)



- Pasii 2-4 (III)

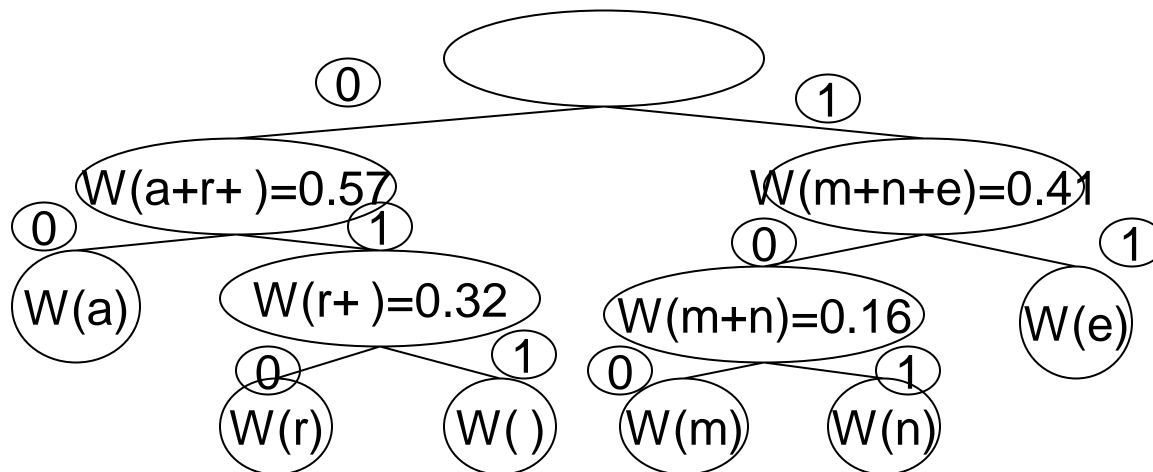


- Pasii 2-4 (IV)



Arbori Huffman – Exemplu (III)

- Pasii 2-4 (V)



- Codificare: a - 00; e -11; r - 010; ' ' - 011; m - 100; n - 101;
- Cost(A) = 2 * 0.25 + 2 * 0.25 + 3 * 0.083 + 3 * 0.083 + 3 * 0.166 + 3 * 0.166 = 1 + 1.2 = 2.2 biti.

Arbori Huffman - pseudocod

- Huffman(K,p){
  - 1. Arb = {k ∈ K | frunză(k, p(k))};
  - 2. **Cât timp** (card (Arb) > 1) // am mai mulți subarbori
  - 3. fie a_1 si a_2 arbori din Arb a.i. $\forall a \in$ Arb $a \neq a_1$ si $a \neq a_2$, avem $w(a_1) \leq w(a)$ si $w(a_2) \leq w(a)$); // practic se extrage // de două ori minimul si se salvează in a_1 si a_2
  - 4. Arb = Arb \ {a_1, a_2} U nod_intern(a_1, a_2, $w(a_1) + w(a_2)$);
  - 5. **Dacă** (Arb = Φ)
  - 6. **Întoarce** arb_vid;
  - 6. **Altfel**
  - 7. fie A singurul arbore din mulțimea Arb;
  - 8. **Întoarce** A;

- Notații folosite:
  - a = frunză (k, p(k)) – subarbore cu un singur nod care conține cheia k, iar w(a) = p(k);
  - a = nod_intern(a_1, a_2, x) – subarbore format dintr-un nod intern cu descendenții a_1 si a_2 si w(a) = x.

Arbori Huffman - Decodificare

- Se încarcă arborele si se decodifică textul din fișier conform algoritmului:

- Decodificare (in, out)

```
  A = restaurare_arbore (in)        // reconstruiesc arborele
 Cât timp (! terminare_cod(in))     // mai am caractere de citit
     nod = A                        // pornesc din rădăcină
     Cât timp (! frunză(nod))       // cât timp nu am determinat caracterul
         Dacă (bit(in) = 1) nod = dreapta(nod)     // avansez in arbore
             Altfel nod = stânga(nod)
     Scrie (out, cheie(nod))        // am determinat caracterul si îl scriu la
                                    // ieșire
```

PA

Demonstrație (I)

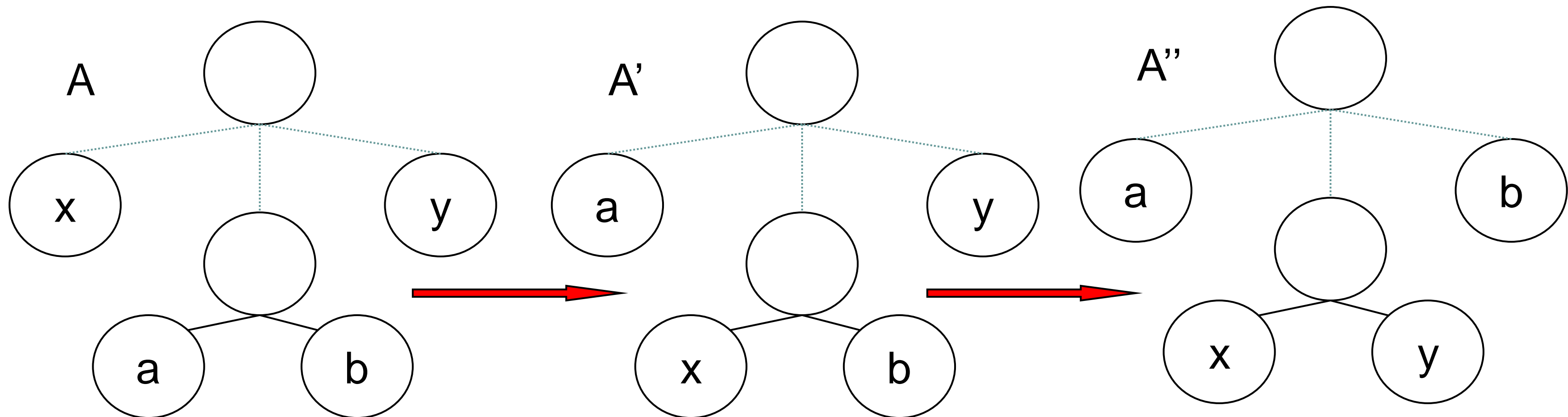
- Arborele de codificare construit trebuie să aibă cost minim pentru a fi arbore Huffman.

- Lema 1. Fie K mulțimea cheilor dintr-un arbore de codificare, card(K) ≥ 2, x, y două chei cu pondere minimă. ∃ un arbore Huffman de înălțime h in care cheile x şi y apar pe nivelul h fiind descendente ale aceluiași nod intern.

Demonstrație (II)

- Demonstrație Lema 1:

$$Cost(A) = \sum_{k \in K} lg_cod(k) * p(k) = \sum_{k \in K} nivel(k, A) * p(k)$$



- Se interschimbă a cu x şi b cu y şi din definiţia costului arborelui => cost(A'') ≤ cost(A') ≤ cost(A) => A'' arbore Huffman

Demonstrație (III)

- Lema 2. Fie A un arbore Huffman cu cheile K, iar x şi y două chei direct descendente ale aceluiaşi nod intern *a*. Fie K' = K \ {x,y} U {z} unde z este o cheie fictivă cu ponderea w(z) = w(x) + w(y). Atunci arborele A' rezultat din A prin înlocuirea subarborelui cu rădăcina *a* si frunzele x, y cu subarborele cu un singur nod care conţine frunza z, este un arbore Huffman cu cheile K'.

- Demonstrație:
  - 1) analog Cost(A') ≤ Cost(A) (Cost(A) = Cost(A') + w(x) + w(y))
  - 2) pp există A'' a.i. Cost(A'') < Cost(A') =>
    - Cost(A'') < Cost(A) - (w(x) + w(y));
    - Cost(A'') + w(x) + w(y) < Cost(A); => A nu este Huffman (contradicţie)

Demonstrație (IV)

- Teoremă – Algoritmul Huffman construiește un arbore Huffman.
- Demonstrație: prin inducție după numărul de chei din mulțimea K.
- $n \le 2$ => evident
- $n > 2$
  - Ip. Inductivă: algoritmul Huffman construiește arbori Huffman pentru orice mulțime cu n-1 chei
  - Fie $K = \{k_1, k_2, \dots, k_n\}$ a.i. $w(k_1) \le w(k_2) \le \dots \le w(k_n)$

Demonstrație (V)

- Cf. Lema 1, ∃ Un arbore Huffman unde cheile k_1, k_2 sunt pe același nivel şi descendente ale aceluiași nod.
- A_{n-1} – arborele cu n-1 chei $K' = K - \{k_1,k_2\} \cup z$ unde $w(z) = w(k_1) + w(k_2)$.
- A_{n-1} rezultă din A_n prin modificările prezentate in Lema 2 => A_{n-1} este Huffman, şi cf. ipotezei inductive e construit prin algoritmul Huffman(K',p').
- => Algoritmul Huffman(K, p) construiește arborele format din k_1 si k_2 si apoi lucrează ca şi algoritmul Huffman(K', p') ce construiește A_{n-1} => construiește arborele Huffman(K, p).

PA

Comparație D&I și Greedy

- Tip abordare
  - D&I: top-down;
  - Greedy: bottom-up.
- Criteriu de optim
  - D&I: nu;
  - Greedy: da.

PA

Alt exemplu (I)

Problema rucsacului

Trebuie să umplem un rucsac de capacitate maximă M kg cu obiecte care au greutatea m_i şi valoarea v_i. Putem alege mai multe obiecte din fiecare tip cu scopul de a maximiza valoarea obiectelor din rucsac.

- Varianta 1: putem alege fracţiuni de obiect – "problema continuă"
- Varianta 2: nu putem alege decât obiecte întregi (număr natural de obiecte din fiecare tip) – "problema 0-1"

PA

Alt exemplu (II)

- Varianta 1: Algoritm Greedy
  - sortăm obiectele după raportul v_i/m_i;
  - adăugăm fracţiuni din obiectul cu cea mai mare valoare per kg până epuizăm stocul şi apoi adăugăm fracţiuni din obiectul cu valoarea următoare.
  - Exemplu: $M = 10$; $m_1 = 5$ kg, $v_1 = 10$, $m_2 = 8$ kg, $v_2 = 19$, $m_3 = 4$ kg, $v_3 = 4$
  - Soluție: (m_2, v_2) 8kg şi 2kg din (m_1,v_1) – valoarea totală: $19 + 2 * 10 / 5 = 23$
- Varianta 2: Algoritmul Greedy nu funcţionează => contraexemplu
  - Exemplu: $M = 10$; $m_1 = 5$ kg, $v_1 = 10$, $m_2 = 8$ kg, $v_2 = 19$, $m_3 = 4$ kg, $v_3 = 4$
  - Rezultat corect – 2 obiecte (m_1,v_1) – valoarea totală: 20
  - Rezultat algoritm Greedy – 1 obiect (m_2,v_2) – valoarea totală: 19

PA

Când funcţionează algoritmii Greedy? (I)

- Problema are proprietatea de substructură optimă
  - soluţia problemei conţine soluţiile subproblemelor.
- Problema are proprietatea alegerii locale
  - alegând soluţia optimă local se ajunge la soluţia optimă global.

PA

Când funcţionează algoritmii Greedy? (II)

- Fie E o mulţime finită nevidă şi $I \subset P(E)$ a.i. $\varnothing \in I$, $\forall X \subseteq Y$ şi $Y \subseteq I \Rightarrow X \subseteq I$. Atunci spunem că (E,I) este un sistem accesibil.
- Submulţimile din I sunt numite submulţimi "independente".
- Exemple:
  - Ex1: $E = \{e_1, e_2, e_3\}$ si $I = \{\varnothing, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_2, e_3\}\}$ – mulțimile ce nu conțin e_1 si e_3.
  - Ex2: E – muchiile unui graf neorientat şi I mulţimea mulţimilor de muchii ce nu conţin un ciclu (mulțimea arborilor).
  - Ex3: E set de vectori dintr-un spaţiu vectorial, I mulţimea mulţimilor de vectori linear independenţi.
  - Ex4: E – muchiile unui graf neorientat şi I mulţimea mulţimilor de muchii în care oricare 2 muchii nu au un vârf comun.

PA

Când funcţionează algoritmii Greedy? (III)

- Un sistem accesibil este un matroid dacă satisface proprietatea de interschimbare:

  $X, Y \subseteq I$ şi $|X| < |Y| \Rightarrow \exists e \in Y \setminus X$ a.i. $X \cup \{e\} \subseteq I$

- Teoremă. Pentru orice subset accesibil (E, I) algoritmul Greedy rezolvă problema de optimizare dacă şi numai dacă (E, I) este matroid.

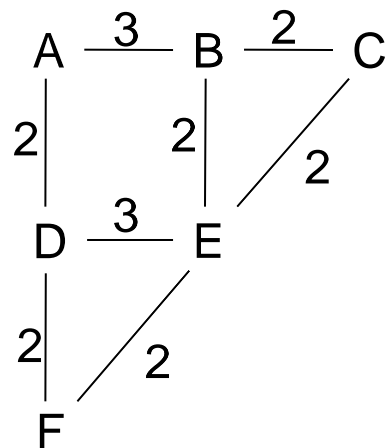
Verificăm exemplele

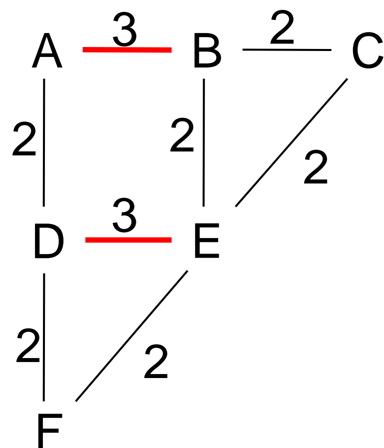
- Ex1: $I = \{\emptyset, \{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}, \{e_2, e_3\}\}$

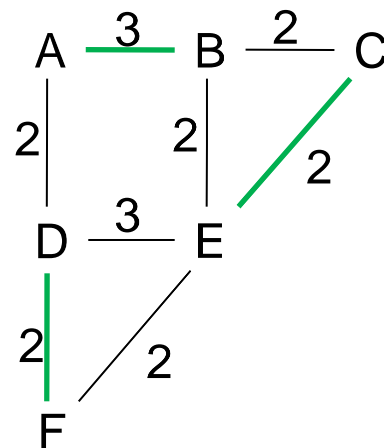
Fie $Y = \{\{e_1\}, \{e_2\}, \{e_3\}, \{e_1, e_2\}\}$ si $X = \{\{e_1\}, \{e_3\}\}$

$\rightarrow Y \setminus X = \{\{e_2\}, \{e_1, e_2\}\} \rightarrow X \cup \{e_2\} \subseteq I \rightarrow$ matroid

- Ex4:







Algoritmul Greedy

- Algoritmul generic Greedy devine:
  - $X = \varnothing$
  - *sortează* elementele din E în ordinea *descrescătoare* a ponderii
  - *Pentru fiecare* element $e \in E$ (sortat) *Repetă*
    - $X = X \cup \{e\}$ dacă şi numai dacă $(X \cup \{e\}) \subseteq I$
  - *Întoarce* X

Greedy – tema de gândire

- Se dă un număr natural n. Să se găsească cel mai mare subset S din {1, 2, ..., n} astfel încât nici un element din S să nu fie divizibil cu nici un alt element din S.
- În plus, dacă există mai multe subseturi maximale ce respectă proprietatea de mai sus, să se determine întotdeauna cel mai mic din punct de vedere lexicografic.
- S este lexicografic mai mic decât T dacă cel mai mic element care este membru din S sau T, dar nu din amândouă, face parte din S.