Proiectarea algoritmilor: Algoritmi Greedy

Ștefan Ciobâcă, Dorel Lucanu

Faculty of Computer Science
Alexandru Ioan Cuza University, Iași, Romania
dlucanu@info.uaic.ro

PA 2015/2016

1. Paradigme de proiectare a algoritmilor

2. Paradigma greedy
    - Problema selecției activităților
    - Algoritmii greedy
    - Problema rucsacului - varianta continuă
    - Problema codurilor Huffman
    - Matroizi
    - Problema APM

Paradigme de proiectare a algoritmilor

Paradigmă de proiectare = o metodă generală de rezolvare a unei clase mai mari de probleme.

1. Divide et impera
2. Greedy
3. Programare dinamică
4. Backtracking
5. ...

Paradigma greedy - probleme de optimizare

Problemă de optimizare:

`Input`: ....

`Output`: cel mai mare/cel mai mic număr cu proprietatea că ....

Exemplu:

`Input`: Un graf $G = (V, E)$ și două noduri $s, t \in V$.

`Output`: Cel mai mic cost *dist* al unui drum de la s la t în graf.

Paradigma greedy - când o folosim

1. Una dintre cele mai simple paradigme de programare.
2. Multe probleme de optimizare pot fi rezolvate eficient folosind algoritmi greedy.
3. Foarte simplu de implementat dar nu neapărat simplu de demonstrat.
4. Pentru alte probleme de optimizare, algoritmii greedy nu produc soluția optimă/cea mai bună (e.g. nu găsesc "soluția" de cost minim, ci o soluție de un cost mai mare). În aceste cazuri, există două strategii:
    1. Dacă se dorește o soluție optimă, se caută altă metodă de rezolvare (folosind, e.g., backtracking sau programare dinamică).
    2. Dacă diferența dintre costul optim și costul produs de algoritmul greedy este tolerabilă în practică, se poate folosi algoritmul greedy.

Paradigma greedy - exemplu (1)

Problema bin-packing.

Strategia greedy: plasează întâi obiectele mari.

Paradigma greedy - exemplu (2)

Problema plății cu număr minim de bancnote.

`Input`: un număr natural n – suma ce trebuie plătită
`Output`: numerele $n_{500}, n_{200}, n_{100}, n_{50}, n_{10}, n_5, n_2, n_1$ (n_i - câte bancnote de i RON folosesc) , astfel încât $\Sigma_i n_i$ să fie minimă și $n = \Sigma_i i n_i$.
Strategia greedy: folosește întâi bancnotele mari.

Problema selecției activităților - exemplu

Azi, un student poate participa la una sau mai multe activități. De exemplu:

1. de la 10 la 12 poate participa la cursul PA;
2. de la 9 la 17 la CodeCamp;
3. de la 18 la 20 la teatru;
4. de la 21 la 22 la film.
5. de la 19 la 24 în club;

Scopul studentului este să participe la **cât mai multe activități** (dar trebuie să participe la fiecare de la început la sfârșit și nu poate fi în două locuri în același timp). Ce activități trebuie să aleagă?

Problema selecției activităților - formal

`Input:` n - numărul de activități, $s[0..n-1]$ - un tablou care conține timpul de început al activităților și $f[0..n-1]$ - un tablou care conține timpul de final al fiecărei activități a.î. f este în ordine crescătoare.
`Output:` $A \subseteq \{0, \dots, n-1\}$ - o mulțime de activități care nu se suprapun de cardinal maxim.
N.B.: două activități i, j se suprapun dacă $[s_i, f_i) \cap [s_j, f_j) \neq \emptyset$.
Exercițiu: arătați că două activități i, j nu se suprapun (sunt compatibile) ddacă $s_i > f_j$ sau $s_j > f_i$.

Problema selecției activităților - exemplu de instanță și răspuns corect

Exemplu. Fie instanța $n = 5$ unde tablourile s și f sunt:

i	0	1	2	3	4
$s[i]$	10	9	18	19	21
$f[i]$	12	17	20	24	22

Un răspuns corect este: $A = \{0, 2, 4\}$.

Problema selecției activităților - ideea de rezolvare

Strategia pe care o vom folosi:

1. începem cu mulțimea $A = \emptyset$;
2. dintre activitățile care au rămas, alegem activitatea i care se termină cel mai devreme (intuitiv, deoarece îmi lasă timp mai mult pentru următoarele activități);
3. adăugăm i la A;
4. ștergem i și toate activitățile care se suprapun cu i din lista de activități disponibile;
5. repetăm procesul dacă mai sunt activități disponibile.

Problema selecției activităților - algoritmul

- `Input`: n, $s[0..n-1]$, $f[0..n-1]$
- `Output`: $A \subseteq \{0, \ldots, n-1\}$, mulțime de activități compatibile de cardinal maxim

- $A = \emptyset$ (mulțimea de activități selectate)
- $time = 0$ (timpul începând cu care sunt disponibil)
- `for` i $= 0$ `to` $n-1$
  - if $s[i] \geq time$ (sunt disponibil pentru activitatea i)
    - $A = A \cup \{i\}$ (selectez activitatea i)
    - $time = f[i]$ (marchez că nu pot accepta activități mai devreme de $f[i]$)

Problema selecției activităților - analiza algoritmului

1. Corectitudine (pe tablă):

Lemma

Fie $S_{time} = \{i \mid s[i] \geq time\}$ o subproblemă a lui $S = \{0, \ldots, n-1\}$. Fie $i \in S_{time}$ activitatea care se termină cel mai devreme din S_{time}. Atunci activitatea i este inclusă într-o submulțime de activități compatibile a lui S_{time} de cardinal maxim.

2. Timp de execuție: $\theta(n)$.
3. Exercițiu. Ce se întâmplă dacă problema nu garantează că vectorul f este în ordine crescătoare?
   Arătați că cele două probleme (în care f este ordonat și respectiv în care f nu este neapărat ordonat) se reduc una la cealaltă.

Algoritmii greedy

1. Algoritmii greedy = secvență de alegeri locale, alegeri care par a fi cele mai bune în momentul respectiv;
2. Odată făcută o alegere, nu ne putem răzgândi.
3. Câteodată această soluție conduce la un optim global (e.g. problema selecției activităților); alteori nu (e.g. problema discretă a rucsacului).

Algoritmii greedy - proces de proiectare

1. Identificăm subprobleme ale problemei de optimizare (e.g. $S_{time} = \{i \mid s[i] \geq time\}$ astfel încât o alegere greedy într-o subproblemă să conducă la o altă subproblemă.
2. (greedy-choice property) Arătăm că există o soluție optimă a problemei inițiale care folosește alegerea greedy.
3. (optimal substructure property) Arătăm că dacă facem o alegere greedy, combinația dintre alegerea greedy și o soluție optimă pentru subproblema rezultată este o soluție optimă pentru problema inițială.

Problema rucsacului - varianta continuă

Un hoț a spart un magazin și a găsit n bunuri. Al i-lea bun valorează v_i lei și cântărește w_i kilograme. Hoțul are un rucsac care poate să ducă cel mult W kilograme de bunuri. Orice bun poate fi secționat iar valoarea unei părți este proporțională cu dimensiunea acesteia (e.g. jumătate dintr-un obiect cu $v_i = 10$ și $w_i = 6$ cântărește 3 kg și valorează 5 lei). Hoțul vrea să maximizeze valorea obiectelor pe care le va pune în rucsac.

Problema rucsacului - formalizare varianta continuă

`Input`: n, $v[0..n-1]$, $w[0..n-1]$, W, toate numere naturale
`Output`: $p[0..n-1]$, $p[i] \in [0,1]$ astfel încât:

1. $\Sigma_i p[i]w[i] \leq W$ (părțile alese ale obiectele încap în rucsac)
2. $\Sigma_i p[i]v[i]$ este maxim (valoarea părților este maximă)

Problema rucsacului - exemplu instanță

Avem $n = 3$ obiecte:

i	0	1	2
$w[i]$	1	2	3
$v[i]$	10	15	20

Rucsacul are capacitate de $W = 5$ kg.
O soluție optimă pentru varianta continuă:

1. iau 100% din primul obiect (deci $p[0] = 1$). Câștig: $1 \times 10 = 10$, capacitate rămasă: $5 - 1 = 4$ kg.
2. iau 100% din al doilea obiect (deci $p[1] = 1$). Câștig: $10 + 1 \times 15 = 25$, capacitate rămasă: $4 - 2 = 2$ kg.
3. iau $2/3$ din al treilea obiect (deci $p[2] = 0.66\ldots$). Câștig: $25 + 2/3 \times 20 = 38.33\ldots$.

Problema rucsacului - abordare greedy

1. subproblemele
2. proprietatea de alegere greedy
3. proprietatea de substructură optimă

(pe tablă).

Problema rucsacului - varianta discretă

Un hoț a spart un magazin și a găsit n bunuri. Al i-lea bun valorează v_i lei și cântărește w_i kilograme. Hoțul are un rucsac care poate să ducă cel mult W kilograme de bunuri. **Niciun bun nu poate fi secționat - obiectul i trebuie furat integral sau deloc**. Hoțul vrea să maximizeze valorea obiectelor pe care le va pune în rucsac.

Problema rucsacului - formalizare varianta discretă

`Input`: n, $v[0..n-1]$, $w[0..n-1]$, W, toate numere naturale
`Output`: $p[0..n-1]$, $p[i] \in \{0, 1\}$ astfel încât:

1. $\Sigma_i p[i]w[i] \leq W$ (obiectele încap în rucsac)
2. $\Sigma_i p[i]v[i]$ este maxim (valoarea obiectelor este maximă)

Problema rucsacului - exemplu instanță

Avem $n = 3$ obiecte:

i	0	1	2
$w[i]$	1	2	3
$v[i]$	10	15	20

Rucsacul are capacitate de $W = 5$ kg.
O soluție optimă pentru varianta discretă:

1. nu iau primul obiect ($p[0] = 0$);
2. iau al doilea obiect ($p[1] = 1$);
3. iau al treilea obiect ($p[2] = 1$).

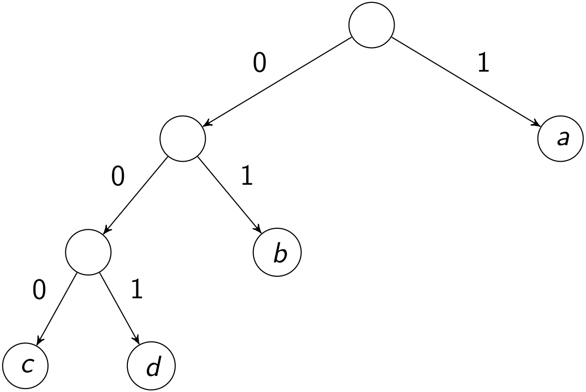
Abordările greedy nu produc soluții optime pentru varianta discretă (vezi capitolele următoare: programare dinamică, backtracking).

Motivație

Să presupunem că avem un fișier care conține 100 de caractere. Caracterul a apare de 45 de ori, b de 30 de ori, c de 10 ori și d de 15 ori.
Fișierul inițial ocupă $100 * 8 = 800$ de biți.
Deoarece în fișier apar doar caracterele a, b, c, d, putem comprima fișierul înlocuind fiecare apariție a lui a cu secvența de biți 00, b cu 01, c cu 10 și d cu 11.
Obținem în acest fel $100 * 2 = 200$ de biți.
Se poate obține o compresie mai bună? Da, folosind coduri de lungime variabilă: $a = 1$, $b = 01$, $c = 000$, $d = 001$. Fișierul comprimat este reprezentat acum prin $1 * 45 + 2 * 30 + 3 * 10 + 3 * 15 = 180$ de biți (90% din dimensiunea inițială).

Coduri prefix

Orice astfel de cod se poate reprezenta în mod unic printr-un arbore binar cu *n* frunze:



Formalizarea problemei

`Input`: n - numărul de caractere distincte; $c[0..n-1]$ - numărul de apariții ale fiecărui caracter.
`Output`: un arbore binar care reprezintă codul prefix optim.
Exemplu: dacă $n = 4$ și $c = [45, 30, 15, 10]$, atunci arborele precedent este un răspuns corect ($a = 1$, $b = 01$, $c = 000$, $d = 001$).
Ideea: combină caracterele cu frecvențele cele mai reduse (pe tablă).

Algoritmul greedy pentru coduri prefix

1. $Q = \emptyset$ (coadă cu prioritate)
2. for $i = 0$ to $n - 1$
   $Q.insert(info : i, frec : c[i], left : null, right : null)$
3. for $i = 0$ to $n - 2$
   $x = new()$
   $x.info = -1$
   $x.left = Q.extractmin()$
   $x.right = Q.extractmin()$
   $x.frec = x.left.frec + x.right.frec$
   $Q.insert(x)$
4. return $Q.extractmin()$

Corectitudinea și analiza algoritmului

1. Există un cod optim în care ultimele două caractere dpdv al frecvenței au aceeași lungime și diferă doar în ultimul bit.
2. Fie x, y cele două caractere cu frecvența cea mai mică. Fie z un nou caracter astfel încât $c[z] = c[x] + c[y]$. Fie T arborele binar care determină codul optim pentru $\Sigma \setminus \{x, y\} \cup \{z\}$ și T' arborele binar obținut din T prin înlocuirea lui z cu un nou nod având copiii x și y. Atunci T' este optim pentru Σ.
3. Timp de rulare: $O(nlogn)$, dacă coada cu prioritate este implementată printr-un heap binar.

Matroid

Mulți algoritmi de tip greedy pot fi demonstrați folosind teoria matroizilor.

Definiție

Un matroid este o pereche $M = (S, I)$ cu proprietățile:

1. *S este o mulțime finită*
2. *(ereditate) I este o mulțime nevidă de submulțimi ale lui S (numite mulțimi independente) și*

   $$\text{dacă } B \in I \text{ și } A \subseteq B, \text{ atunci } A \in I.$$

3. *(interschimbare) dacă $A \in I, B \in I$ și $|A| < |B|$, atunci există $x \in B \setminus A$ astfel încât $A \cup \{x\} \in I$.*

Exercițiu: demonstrați că $\emptyset \in I$.

Matroid - exemplu

Fie $G = (V, E)$ un graf și $M_G = (S_G, I_G)$ definit astfel:

1. $S_G = E$, muchiile grafului
2. $A \subseteq E \in I$ ddacă A nu conține ciclu.

Exercițiu: arătați că M_G este matroid.
Exercițiu: arătați că, dacă la itemul 2 cerem ca A să fie arbore, M_G nu este matroid.
Exercițiu: orice mulțime maximală din I are același cardinal.

Matroid ponderat

Fie $M = (S, I)$.
Fie $w : S \to \mathbb{N}$ o funcție care asociază fiecărui element x din S o pondere $w(x)$.
Funcția w se extinde la mulțimi $A \subseteq S$: $w(A) = \Sigma_{x \in A} w(x)$.
Dacă M este matroid, atunci există un algoritm greedy pentru găsirea unei mulțimi independente de pondere maximă.

Algoritmul greedy (general pentru matroizi)

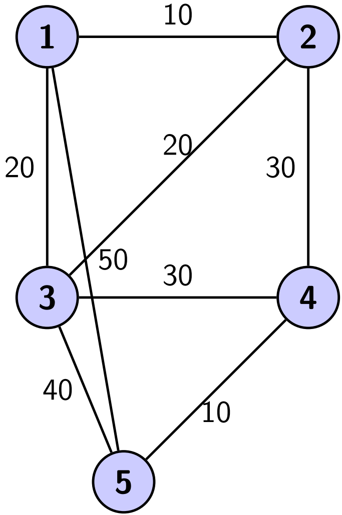
1. $A = \emptyset$ (încep cu mulțimea vidă)
2. sortează $M.S$ în ordine descrescătoare a ponderilor
3. `for each` $x \in M.S$ (în ordine descrescătoare)
   - `if` $A \cup \{x\} \in I$
     $A = A \cup \{x\}$

Teoremă: algoritmul găsește o mulțime independentă de pondere maximă.

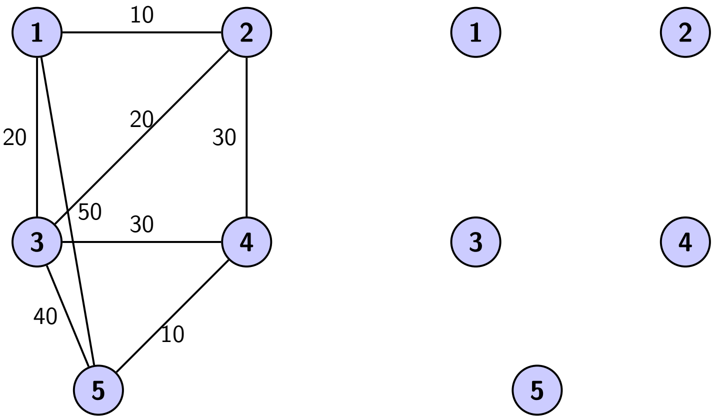
Problema arborelui parțial de cost minim

`Input`: un graf $G = (V, E)$, fiecare muchie $e \in E$ având un cost $l(e)$
`Output`: un arbore $G = (V, A)$, astfel încât suma muchiilor arborelui să fie de cost minim.

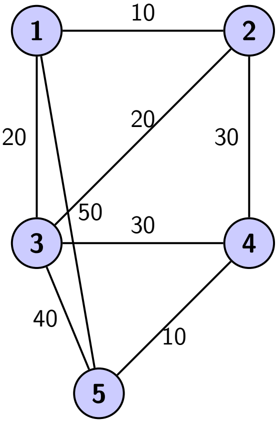
Algoritmul lui Kruskal - instanța

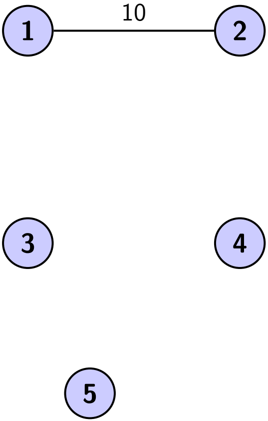


Algoritmul lui Kruskal - pasul 1

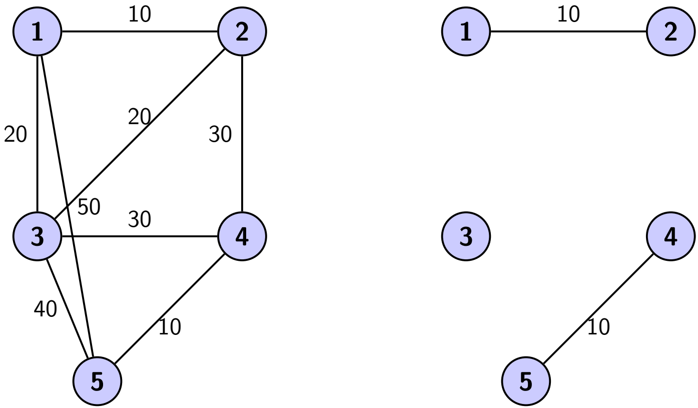


Algoritmul lui Kruskal - pasul 2

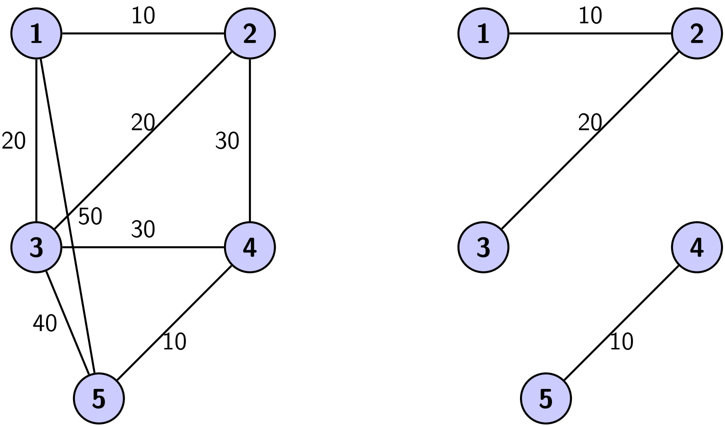




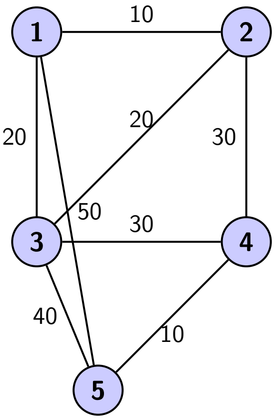
Algoritmul lui Kruskal - pasul 3

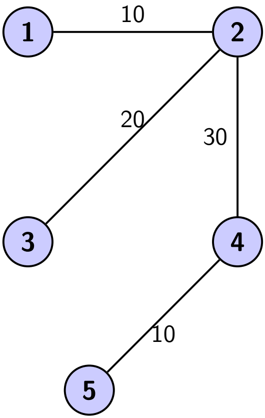


Algoritmul lui Kruskal - pasul 4



Algoritmul lui Kruskal - pasul 5





Algoritmul lui Kruskal ca instanță a unui matroid

Fie $G = (V, E)$ graful pentru care aplicăm algoritmul lui Kruskal.
Definim $M_G = (S_G, I_G)$ ca mai sus (am arătat că este matroid):

1. $S_G = E$, muchiile grafului
2. $A \subseteq E \in I$ ddacă A nu conține ciclu.

Exercițiu: mulțimile maximale din I sunt arbori.
Dacă definim $w(e) = M - l(e)$ (unde M este un număr suficient de mare), atunci algoritmul lui Kruskal este o instanță a algoritmului general pentru matroizi.

Concluzii

1. Greedy = o paradigmă importantă de proiectare a algoritmilor.
2. De obicei algoritmii greedy sunt ușor de implementat (nu neapărat și de demonstrat).
3. Multe probleme de optimizare au soluții optime ce pot fi găsite cu greedy (e.g. problema continuă a rucsacului). Pentru alte probleme (e.g. problema discretă a rucsacului), algoritmii greedy nu produc soluția optimă.
4. Matroid = structură matematică cu ajutorul căreia putem modela diverse probleme de optimizare. Dacă problema se poate modela cu ajutorul unui matroid (ponderat), atunci algoritmul greedy produce soluția optimă.