10 Laborator: Paradigma de programare Greedy

10.1 Obiective

În aceasta lucrare se prezintă paradigma de programare Greedy. Vom descrie cum se poate aplica pe probleme de optimizare. Se vor prezenta atât probleme care se pot rezolva în mod optim cu aceasta paradigmă cât şi exemple unde nu returnează soluţia optimă.

10.2 Noţiuni teoretice

10.2.1 Definiţie

Paradigma sau strategia Greedy se aplică la rezolvarea problemelor de optimizare. O problemă de optimizare urmăreşte minimizarea (sau în mod echivalent maximizarea) unei funcţii de cost care se aplica pe o soluţie. Soluţia se va construi pas cu pas, adăugând un element sau făcând o decizie. Paradigma Greedy (lacom) va alege la fiecare pas decizia care minimizează funcţia de cost local.

Fiind dată o problemă de optimizare paradigma Greedy nu produce neaparat soluţia care corespunde la minimul global. Dacă se doreşte gasirea soluţiei optimale trebuie utilizată o altă abordare (programare dinamică, backtracking, căutare exhaustivă). Există şi posibilitatea ca Greedy să nu gasească o soluţie validă deloc.

În unele situaţii se doreşte o soluţie suboptimală, care poate fi folosită drept limită superioară. În acest caz se poate utiliza paradigma Greedy fiindca ea va fi rapida, intuitivă şi simplu de implementat. Rapiditatea este datorită faptului că nu se explorează spaţiul de căutare în întregime.

Şablon general pentru paradigma Greedy:

```
S = {} //multimea care formeaza solutia
A //multimea actiunilor/elementelor posibile
while S nu este finalizata
        a = alege(A) //alege cel mai bun element disponibil in A
        if S ∪ a satisface restrictiile
                S = S ∪ a
        endif
endwhile
```

Demonstraţia optimalităţii se face de obicei prin metoda reducerii la absurd. Se presupune că soluţia nu este optimă şi se compară cu o soluţie optimă. În general, este mai dificilă demonstraţia corectitudinii decât implementarea metodei.

10.2.2 Exemple de soluţii optimale

În această parte considerăm probleme care au o rezolvare optimă prin strategia Greedy. Se prezintă problema, se defineşte funcţia de cost, se descrie un algoritm pentru rezolvare şi se demonstrează optimalitatea.

- **Descumponerea unui număr x într-o sumă de numere puteri ale lui 2** deci numere din mulţimea $B = \{1, 2, 4, ...\} = \{2^i | i \in \mathbf{N}\}$ cu un număr minim de termeni

  Functia de cost:

  - numarul de termeni: $f(a) = min(length(a)), \sum_i a_i = x, a_i \in B$

  Strategia Greedy:

  - pornim de la soluţia vidă
  - dacă suma soluţiei curente este egală cu x, am terminat
  - altfel alegem numărul cel mai mare din B, care adăugat la suma curentă rezultă într-un număr mai mic sau egal decât x

Demonstraţie optimalitate:

- Pentru orice x există o reprezentare unică în binar care utilizează cel mult un element din B. Fiindcă soluţia greedy alege întotdeauna puterea cea mai mare din B va rămâne de rezolvat o subproblemă $x - 2^k < 2^k$. Deci se obţine reprezentarea binară unică. Observăm că problema este similară cu cea prezentată la contraexemple pentru Greedy, însă am schimbat mulţimea. Această mulţime B trebuie să posede proprietăţi speciale pentru a asigura o soluţie optimă cu Greedy (vezi problema **).

ex. x = 20 solutia Greedy: 2 termeni (20 = 16 + 4);

- **Selecţia activităţilor:** se dă o listă de activităţi $S = a_1, a_2, \ldots a_n$, fiecare având un timp de inceput şi de sfârşit $a_i = (s_i, f_i)$ cu $0 \leq s_i < f_i < \infty$ Toate activităţile sunt la fel de atractive. Dorim să maximizăm numărul de activităţi realizate într-o zi. Vom alege numărul maxim de activităţi care nu se suprapun, adică ele sunt compatibile. Două activităţi a_i şi a_j sunt compatibile dacă $[s_i, f_i) \cap [s_j, f_j) = \emptyset$

Functia de cost:

- cardinalitatea submulţimii: $f(A) = min_A|A|, A \subset S$, a.î. a_i compatibil cu $a_j, \forall a_i \neq a_j \in A$

Strategia Greedy:

- O soluţie optimă pentru alegerea activităţilor este în ordinea descrescătoare a timpului de finalizare.

Pseudocod:

```
Greedy-activity-selector(A)
Sort activities in ascending order of finish time
N = A.length
S={a1}
k = 1
for m = 2 to n do
    if (s_m ≥ f_k) then
        S = S ∪ a_m
        k = m - Stergem din A activitatile care se suprapun cu a_k
Return S
```

Se dă exemplul din figura **??**. Se cunosc timpii de început şi de sfârşit ai fiecărei activităţi.

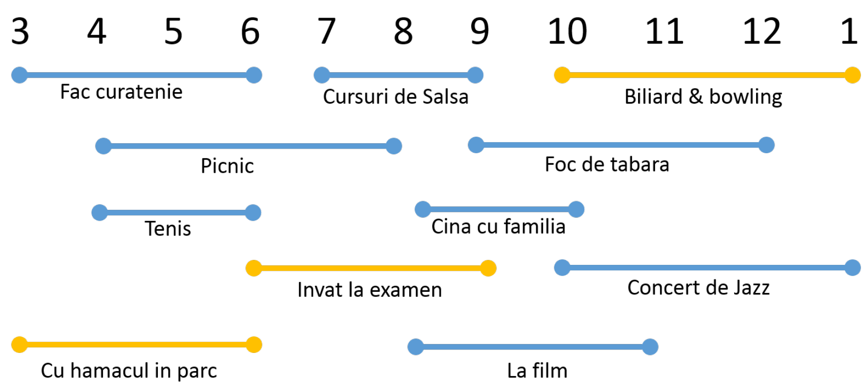

Figure 10.1: Exemple de activitati pentru o zi

Demonstraţie optimalitate:
Teorema – pentru problema selectiei activitatilor metoda greedy alege solutia optima. Inainte de a demonstra acest lucru, consideram urmatoarea observatie.

Observatie: Activitatea k aleasa de metoda greedy se termina mai repede (devreme) decat activitatea k aleasa de orice alta solutie care reprezinta o combinare corecta a activitatilor. E nevoie sa demonstram ca acest lucru este adevarat si sa aratam ca daca e adevarat, algoritmul greedy genereaza solutia optima.

Notam cu $f(i, S)$ timpul de terminare a activitatii i in solutia S.

Lema: Daca S e o solutie data de greedy si $S*$ e solutia optima, atunci pentru oricare $1 \leq i \leq |S|$, avem $f(i, S) \leq f(i, S*)$.

Demonstratie prin inductie:

1. Prima activitate are $f(1, S) \leq f(1, S*)$ pentru ca asa fost selectata – are timpul minim de terminare.

2. Inductia: presupunem ca pentru activitatea i cu $1 \leq i < |S|$ este adevarata proprietatea $f(i, S) \leq f(i, S*)$, activitatea i din solutia S se termina inaintea activitatii i din solutia $S*$.
3. Demonstram pentru $i+1$: in solutia $S*$ activitatea $(i+1)$ incepe dupa ce se termina actitiatea i din $S*$, folosind punctul 2) este adevarat ca activitatea $i+1$ din $S*$ incepe dupa ce se termina activitatea i din S.
4. Astfel activitatea $(i+1)$ din $S*$ trebuie sa fie in multimea activitatilor A cand algoritmul greedy face selectia activitatii $(i+1)$ din S. Cum greedy selecteaza activitatile din A in ordinea crescatoare a timpului de finalizare, avem ca $f(i+1, S) \leq f(i+1, S*)$.

Vom demonstra in continuare ca pentru problema selectiei activitatilor metoda greedy alege solutia optima.

Demonstratie:

- Fie S solutia determinata de metoda greedy si fie $S*$ o solutie optima.
- $S*$ este optima deci $|S| \leq |S*|$.
- Vom demonstra ca $|S| \geq |S*|$.
  * Presupunem prin contradictie ca $|S| < |S*|$. Fie $k = |S|$.
  * Folosind lema de mai sus stim ca $f(k, S) \leq f(k, S*)$, deci activitatea k din S se termina mai repede decat activitatea k din $S*$.
  * Dar s-a presupus ca $|S| < |S*|$ deci exista in $S*$ activitatea $(k+1)$ iar timpul ei de start este dupa timpul de terminare a activitatii k din $S*$, adica $f(k, S*)$, respectiv dupa $f(k, S)$.
  * Deci dupa ce algoritmul greedy a adaugat activitatea k la solutia S, in multimea de activitati A ar mai exista activitatea $k+1$ din $S*$.
  * Dar algoritmul greedy se termina cand nu mai poate adauga activitati deci A este vida! Avem o contradictie. Presupunerea noastra este falsa deci $|S| \geq |S*|$.
  * Stim ca $|S| \leq |S*|$, am demonstrat ca $|S| \geq |S*|$ deci $|S| = |S*|$.

10.2.3 Exemple de soluţii neoptimale

În această parte ilustrăm prin exemple situaţiile unde paradigma Greedy nu produce rezultatul optim. Pentru fiecare caz descriem clar problema de optimizare, funcţia de cost care se minimizează, strategia Greedy şi care este presupunerea greşită. Pornim de la exemple simple unde este evidentă aceasta presupunere greşită si continuam cu probleme mai complicate şi întâlnite în practică.

- Găsirea minimului dintr-un şir de numere a

  Funcţia de cost:

  - valoarea minimului $f(a) = min(a)$

  Strategia Greedy:

  - pornim de la poziţia 0
  - dacă există un vecin mai mic ne poziţionăm pe el
  - în caz contrar returnăm elementul curent

  Presupunerea greşită:

  - dacă vecinul este mai mare, minimul nu poate fi după el.

  ex. a = [2 3 1] soluţia Greedy: 2; soluţia optimală: 1.

- Găsirea drumului cu lungime minimă între două noduri, x şi y, dintr-un graf

  Functia de cost:

  - lungimea drumului: $f(x, y) = min(lungime(drum(x, y)))$

  unde drum(x,y) este mulţimea drumurilor între nodul x şi y; lungime(.) este lungimea unui drum;

  Strategia Greedy:

  - reprezentăm drumul printr-o listă de noduri conectate
  - pornim de la nodul x, care se adaugă la drumul iniţial
  - dacă nodul y este adiecent ultimul nod din drum, îl adăugam şi se termină algoritmul
  - altfel adăugam cel mai apropiat vecin

Presupunerea greşită:

- drumul de lungime minimă conţine vecinul cel mai apropiat

ex. drumul cel mai scurt între nodurile 0 şi 3 soluţia Greedy: 11; soluţia optimală: 10.

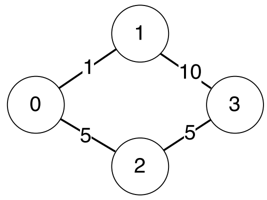

Figure 10.2: Exemplu de graf unde strategia Greedy no obţine soluţia optimă

- Descumponerea unui număr x într-o sumă de numere din mulţimea $Y = \{1, 6, 10\}$ cu un număr minim de termeni

  Functia de cost:

  - numărul de termeni: $f(a) = min(length(a)), \sum_i a_i = x, a_i \in Y$

  Strategia Greedy:

  - pornim de la soluţia vidă
  - dacă suma soluţiei curente este egală cu x, am terminat
  - altfel alegem numărul cel mai mare din Y, care adăugat la suma curentă rezultă într-un număr mai mic sau egal decât x

  Presupunerea greşită:

  - soluţia optimă conţine cel mai mare număr mai mic decât suma x.

  ex. soluţia Greedy: 4 ($x = 13 = 10 + 1 + 1 + 1$); soluţia optimală: 3 ($x = 13 = 6 + 6 + 1$).

10.3 Mersul lucrarii

10.3.1 Probleme obligatorii

Se rezolvă problemele 1,2 şi una la alegere din problemele 3-8. Cuvantul ingrosat reprezintă denumirea codului şablon.

1. **(decompunere)** Se citeşte o sumă de bani x. Să se descompune suma folosind bancnote din mulţimea $a = \{1, 5, 10, 50, 100, 200, 500\}$. Descompunerea trebuie să folosească un număr minim de bancnote.

   Ajutor: Scădem cel mai mare număr care este mai mic sau egal decât suma rămasă.

2. **(activitati)** Se citeşte o listă de activităţi caracterizate prin moment de începere, moment de terminare şi denumire. Să se determine submulţimea de activităţi care rezultă în număr maxim de activităţi realizate dacă două activităţi nu pot să se suprapună în timp.

   Ajutor: Ca şi strategie de rezolvare se considera întotdeauna activitatea cu timp de finalizare minim care începe după ultima sarcină selectată.

3. **(conversie)** Se citeşte un număr întreg x, unde $0 \leq x < 10^{18}$. Să se determine reprezentarea lui x în baza b, $1 < b < 10$ folosind o strategie Greedy.

   Ajutor: Se caută numărul cel mai mare k pentru care $x \geq b^k$. Atunci reprezentarea conţine cifra $[x/b^k]$ pe poziţia k (poziţia 0 este ultima cifră, din dreapta). Scădem din x valoarea $[x/b^k]$ şi continuăm până când x devine 0.

4. **(numere)** Fie $a_i, i = 0, 1, ..., n-1$ o listă de n numere naturale nenule, $n \leq 1000$ şi $a_i \leq 10^6$. Se defineşte $L = \sum_i a_i$, suma valorilor din a. Se cere modul optim de a descompune L în bucăţile a_i. O operaţie de descompunere constă în separarea unei bucăţi x în două u şi v a.î. $x = u + v$. Această descompunere are un cost egal cu x. De exemplu, $a = \{1, 1, 4\}, L = 6$ se poate descompune în două moduri, modul optim este: $6 = ((1 + 1) + 4)$ cu cost 8. Cealaltă variantă este $6 = (1 + (1 + 4))$, cu cost 11.

   Ajutor: Se porneşte de la bucăţi şi se unesc cele 2 bucăţi de mărime minimă. Se formează o listă nouă cu bucăţile unite având $n - 1$ elemente. Se aplică aceeaşi regulă până când avem un singur element de marime L.

5. **(tsp)** Problema comis voiajorului (traveling salesman problem). Fie $G = (V, E)$ un graf neorientat în care oricare două vârfuri diferite ale grafului sunt unite printr-o latură căreia ii este asociat un cost strict pozitiv. Cerinţa este de a determina un ciclu care începe de la un nod aleatoriu al grafului, care trece exact o dată prin toate celelalte noduri şi care se întoarce la nodul iniţial, cu condiţia ca ciclul să aibă un cost minim. Costul unui ciclu este definit ca suma tuturor costurilor ataşate laturilor ciclului.

6. Se citeste un numar x de la tastatura. Sa se realizeze descompunerea numarului intr-o suma de numere puteri ale lui 2 – deci numere din mulţimea $B = \{1, 2, 4, ...\} = \{2^i | i \in \mathbf{N}\}$ cu un număr minim de termeni.

7. Problema rucsacului 0-1. Avem la dispoziţie un rucsac cu capacitatea M. Totodată avem N obiecte caracterizate fiecare prin greutate şi valoare (profit). Cerinţa este să introducem obiecte în rucsac astfel încât să obţinem o valoare (profit) cât mai mare a obiectelor din rucsac, fără a depăşi greutatea maximă a rucsacului (M).

8. Problema rucsacului fractionar. Problema anterioară dar este posibil să adăugăm în rucsac şi o parte fracţionară dintr-un obiect.

10.3.2 Probleme pentru puncte in plus

1. * Implementaţi algoritmul lui Prim în mod eficient. Timpul de execuţie trebuie să fie de ordinea secundelor pentru un graf cu 10^6 muchii. Graful de intrare este conex, neorientat şi ponderat cu numere zecimale.

2. ** Să se determine dacă o mulţime S este un sistem de monede canonic sau nu. Un sistem canonic garantează că orice x se poate descompune în mod optimal folosind abordarea Greedy.

10.3.3 Bibliografie

1. Design and Analysis of Algorithms - The Hong Kong University of Science and Technology

2. Algorithm Design, Kevin Wayne - Princeton University

3. Curs Structuri de date si algoritmi - West University of Timisoara