Metoda greedy euristică

Capitolul 3

- ❖ **Prezentarea metodei**
- ❖ **Implementări sugerate**
- ❖ **Probleme propuse**
- ❖ **Soluțiile problemelor**

3.1. Euristica greedy

Metoda greedy o aplicăm în cazul în care se caută un optim. Dacă în cazul unei probleme această metodă găseşte optimul pentru anumite date de test, iar pentru alte date fie nu găseşte soluția (raportează că nu există soluție), fie oferă o soluție care nu este optimă, spunem că metoda este *euristică*. Chiar dacă originea cuvântului face trimitere la *descoperire*, s-ar putea ca metoda să ne dezamăgească în cazul anumitor probleme.

Totuşi, în cazul acelor probleme pentru care rezultatul exact este furnizat de o metodă de complexitate mărită, cum ar fi metoda backtracking, s-ar putea să ne mulțumească şi rezultatele găsite de o euristică de tip *greedy*.

Exemplu

Fie o problemă clasică din această clasă de probleme. Un vânzător trebuie să dea rest unui client S lei. El are la dispoziție bancnote având valorile b_1, b_2, …, b_n, în număr nelimitat. Trebuie să afişăm o modalitate de a da rest, ştiind că vânzătorul doreşte să folosească cât mai puține bancnote.

Având în vedere cerința problemei de a folosi bancnote cât mai puține, vom încerca să alegem pentru început bancnota cea mai mare. În acest fel se acoperă din sumă o cantitate mare, cu bancnote puține. Pentru suma rămasă de plătit vom proceda la fel, la fiecare pas alegând bancnota cea mai mare posibilă, în număr maxim. Dacă S este suma rămasă de descompus la un moment dat şi cea mai mare bancnotă disponibilă (mai mică sau egală cu S) este b_i, atunci se vor alege $[S / b_i]$ bancnote de acest fel.

În această problemă este mai utilă forma modificată a algoritmului `Greedy`. Vom ordona descrescător şirul bancnotelor, după care vom face selecția lor. Vom memora soluția în şirul x_i, $i = 1, 2, \ldots, n$, unde x_i reprezintă numărul bancnotelor având valoare b_i, $i = 1, 2, \ldots, n$ astfel încât să avem: $S = b_1*x_1 + b_2*x_2 + \ldots + b_n*x_n$.

```
Subalgoritm Selecție-Greedy():
```

```
  Ord_Desc(n,b)
  i ← 0
  cât timp (S > 0) şi (i < n) execută
    i ← i+1
    x_i ← [S/b_i]
    nrb ← nrb+x_i
    S ← S -x_i*nrb
  sfârşit cât timp
  dacă S = 0 atunci
    scrie x_1, ..., x_i
  altfel
    scrie 'Nu s-a gasit solutie.'
  sfârşit dacă
sfârşit subalgoritm
```

Acest algoritm nu furnizează întotdeauna o soluție. De exemplu, dacă avem la dispoziție bancnote de 10, 5 şi 3 lei şi $S = 39$, conținutul şirului x va fi (3, 1, 1), dar acesta nu constituie soluție a problemei, deoarece mai rămâne de descompus suma 1 pentru care nu sunt bancnote.

Analizând exemplul, observăm că problema, totuşi, are soluție. Aceasta corespunde următoarei descompuneri: $3 \cdot 10 + 0 \cdot 5 + 3 \cdot 3 = 39$.

Rezolvarea exactă a problemei se poate face încercând toate combinările posibile de bancnote, folosind metoda backtracking.

Observații

1. Dacă $b_n = 1$, algoritmul furnizează întotdeauna o soluție.
2. În situația în care există bancnote de b^0, b^1, b^2,…b^n unități, disponibile în număr nelimitat, prin aplicarea metodei *greedy* se obține întotdeauna soluția optimă.

Concluzie

O metodă *euristică greedy* presupune folosirea unui algoritm *greedy* care nu determină întotdeauna soluția optimă a unei probleme. Dar există situații, în special în practică, unde este mulțumitoare şi o soluție aproximativă a unei probleme. De cele mai multe ori ea se obține repede şi se implementează uşor. O soluție exactă care s-ar obține cu metoda backtracking, practic, este imposibil de obținut, din cauza timpului de execuție exponențial.

Multe dintre problemele care se rezolvă cu metoda backtracking trebuie să minimizeze o funcție. Avem posibilitatea unei optimizări dacă, înainte de a „intra” în backtracking aplicăm o metodă euristică. În timpul construcției unei soluții cu backtracking, se verifică dacă din soluția parțială găsită se poate obține una mai bună decât cea obținută cu metoda euristică. În caz afirmativ se continuă construirea soluției.

3.2. Implementări sugerate

Pentru a vă familiariza cu modul în care se abordează problemele în care poate fi aplicată o metodă euristică greedy, vă sugerăm să implementaţi algoritmi pentru:

1. plata sumei cu număr minim de bancnote;
2. împachetarea unui rucsac, astfel încât valoarea obiectelor împachetate să fie cât mai mare posibil (problema discretă a rucsacului);
3. problema submarinului (determinarea numărului minim de submarine necesare pentru distrugerea a *N* vapoare);
4. determinarea ordinii în care *M* lucrări independente ale căror timpi de executare se cunosc, se vor executa de către *N* procesoare care pot lucra în paralel, astfel încât durata totală să fie minimă;
5. repartiţia, pe baza unor liste de preferinţe, a *N* apartamente la *N* persoane, astfel încât satisfacerea cerinţelor să fie maximă.

3.3. Probleme propuse

3.3.1. Rucsac[1]

Un hoţ dă o spargere la un magazin, găsind *n* obiecte. Fiecare obiect are o greutate şi o valoare. Ştiind că în rucsac nu poate pune obiecte a căror greutate totală să depăşească valoarea *G*, să se precizeze ce obiecte trebuie să ia hoţul pentru a avea un câştig maxim. *Hoţul nu poate să ia fracţiuni din obiecte*.

Date de intrare

Considerăm obiectele numerotate cu numere naturale de la 1 la *n*. Prima linie a fişierului de intrare `RUCSAC.IN` conţine numărul *n* al obiectelor disponibile şi numărul real *G*, reprezentând capacitatea rucsacului. Linia a doua conţine *n* numere întregi separate prin câte un spaţiu, reprezentând greutăţile celor *n* obiecte. Linia a treia conţine *n* numere întregi, care reprezintă valorile celor *n* obiecte.

Date de ieşire

Fişierul `RUCSAC.OUT` va conţine două linii. Pe prima linie se va scrie valoarea totală a obiectelor selectate (număr real cu două zecimale exacte), iar pe linia a doua se vor scrie numerele de ordine ale obiectelor alese. Ele vor fi separate prin câte un spaţiu.

Restricţii şi precizări

- $1 \le n \le 100$;
- Dacă problema admite mai multe soluţii, în fişier se va scrie una singură.

[1] Problema discretă (0-1) a rucsacului

Exemplu

RUCSAC.IN	**RUCSAC.OUT**
5 10 1 4 3 5 1 8 3 15	24 4 2 1

3.3.2. Expoziție

Într-o sală de expoziție pavilioanele sunt dispuse conform unui caroiaj de dimensiuni $m \times n$. O firmă de pază a încheiat contracte cu o parte din firmele participante, pentru a le supraveghea pavilioanele pe toată durata târgului. Organizatorii pun la dispoziția firmei de pază foişoare care se pot monta deasupra pavilioanelor.

Dintr-un foişor se poate supraveghea un pavilion dacă acesta este situat pe aceeaşi linie sau pe aceeaşi coloană cu foişorul şi dacă între pavilion şi foişor nu mai există alt pavilion. De asemenea, dintr-un foişor nu se poate supraveghea pavilionul deasupra căruia este instalat.

Ajutați directorul firmei de pază să instaleze cât mai puține foişoare care să supravegheze toate pavilioanele contractate.

Date de intrare

Prima linie a fişierului de intrare **EXPO.IN** conține numerele naturale m şi n despărțite de un spațiu. Pe următoarele m linii se află câte n numere întregi, separate prin câte un spațiu, de valori 0 sau 1 având semnificația:

- valoarea 1 corespunde unui pavilion care trebuie păzit;
- valoarea 0 corespunde unui pavilion care nu trebuie păzit.

Date de ieşire

Fişierul **EXPO.OUT** va conține pe prima linie un număr k, reprezentând numărul minim de foişoare care trebuie instalate. Pe următoarele k linii se vor scrie perechi de numere care reprezintă coordonatele foişoarelor. Numerele vor fi separate prin câte un spațiu.

Restricții şi precizări

- $1 \le m, n \le 100$;
- Dacă există mai multe soluții, în fişier se va scrie una singură.

Exemplu

EXPO.IN	**EXPO.OUT**
5 5 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 1	5 1 2 1 1 4 3 5 3

3.3.3. Sportivi

Mai mulți sportivi de performanță trebuie să se antreneze pe aparate identice într-o sală de sport. În funcție de nivelul de pregătire, fiecare sportiv se antrenează un anumit timp t_i, stabilit dinainte. Un sportiv se antrenează fără întrerupere şi fără să își schimbe aparatul la care începe antrenamentul. După ce fiecare sportiv îşi termină antrenamentul, şi numai atunci, toți sportivii au voie să meargă, împreună, la „Festivalul Toamnei", care tocmai a început în oraş. Ştiind că sunt m sportivi şi n aparate, repartizați sportivii la aparate, astfel încât să poată pleca la festival cât mai repede.

Date de intrare

Prima linie a fişierului de intrare **SPORTIVI.IN** conține două numere naturale n *şi* m, reprezentând numărul aparatelor, respectiv numărul sportivilor. Următoarea linie conține m numere întregi, reprezentând durata antrenamentului fiecărui sportiv. Aceste numere vor fi separate prin câte un spațiu.

Date de ieşire

Fişierul de ieşire **SPORTIVI.OUT** va conține $n + 1$ linii. Pe prima linie se va scrie un număr întreg care reprezintă timpul scurs de la începerea antrenamentelor până când termină şi ultimul exercițiile şi sportivii pot pleca la festival. Următoarele n linii vor conține duratele antrenamentelor pe aparate, adică pe linia $i + 1$ se vor scrie duratele antrenamentelor sportivilor cărora li s-a repartizat aparatul i.

Restricții şi precizări

- $1 \le n, m \le 1000$;
- $1 \le t_i \le 1000$, unde t_i reprezintă duratele antrenamentelor, $i = 1, 2, \ldots, m$.

Exemplu

SPORTIV.IN	SPORTIV.OUT
3 6	9
6 5 1 3 4 7	1 7
	3 6
	4 5

3.3.4. Răsplata

Elevii care au fost premiați la Olimpiadele Naționale sunt răsplătiți cu ajutorul mai multor agenții de turism. Pentru cei n elevi premianți se adună n oferte de excursii în țară şi străinătate. Excursiile, numerotate de la 1 la n sunt diferite ca destinație sau durată. Pentru a putea repartiza elevii, acestora li se cere o listă de preferințe. Elevii vor specifica excursiile în care doresc să meargă, începând cu excursia care îi atrage cel mai mult.

Determinați o repartizare a elevilor, astfel încât să fie mulțumiți cât mai mulți elevi.

Date de intrare
Prima linie a fişierului **RASPLATA.IN** conţine un număr natural *n*, reprezentând numărul elevilor şi numărul excursiilor. Următoarele *n* linii conţin opţiunile elevilor, în ordinea numerotării lor. Pe linia *i* + 1 va fi o listă cu opţiunile elevului *i*: mai întâi numărul *nr* de opţiuni, urmat de *nr* numere, reprezentând numerele de ordine ale excursiilor.

Date de ieşire
Pentru a putea selecta excursiile conform cerinţei problemei, vom asocia nişte punctaje excursiilor de pe liste: prima excursie va avea *n* puncte, a doua *n* – 1 puncte, a treia *n* – 2 etc. Vom întocmi un punctaj general *Total* şi în cazul în care se repartizează unui elev o excursie aflată pe poziţia *k* pe lista lui de preferinţe, vom adăuga la *Total* valoarea *n* – *k* + 1. Dacă unui elev nu i se poate repartiza nici o excursie dorită de acesta, *Total* nu se va modifica.

Fişierul de ieşire **RASPLATA.OUT** va conţine trei linii: pe prima linie se scrie valoarea *Total*, valoare care reflectă gradul de mulţumire a elevilor. Pe a doua linie se vor scrie elevii, în ordinea în care au fost repartizaţi. Pe a treia linie se vor scrie numerele de ordine ale excursiilor în conformitate cu linia anterioară.

Restricţii şi precizări
- $1 \le n \le 100$.

Exemplu

RASPLATA.IN	**RASPLATA.OUT**
4 4 3 1 2 4 2 3 2 3 4 1 2 4 2 3 4 1	2 3 4 1 3 4 2 1 15

3.3.5. Reţea liniară
Într-un campus universitar s-au adus *n* calculatoare. Ele s-au dispus în diverse clădiri: bibliotecă, secretariat, cămine etc. Se hotărăşte legarea calculatoarelor într-o reţea liniară: fiecare calculator va fi legat cu alte două calculatoare, într-un lanţ, cu excepţia calculatoarelor de la capetele lui, care se vor conecta doar cu câte un singur calculator. Cunoscându-se coordonatele *x* şi *y* ale fiecărui calculator, să se afişeze o modalitate de legare a calculatoarelor, astfel încât cablul folosit să aibă lungime minimă.

Date de intrare
Prima linie a fişierului de intrare **RETEA.IN** conţine un număr natural *n*, reprezentând numărul calculatoarelor. Următoarele *n* linii conţin perechi de numere reale strict pozitive *x* şi *y*, reprezentând coordonatele în plan ale calculatoarelor.

Date de ieşire
Fişierul de ieşire **RETEA.OUT** va conține două linii: pe prima linie se va scrie un număr real cu două zecimale exacte, reprezentând lungimea cablului necesar interconectării, iar pe a doua linie se scriu *n* numere întregi, reprezentând numerele de ordine ale calculatoarelor, în ordinea în care apar de-a lungul rețelei.

Restricții şi precizări
- $1 \le n \le 100$

Exemplu

RETEA.IN	RETEA.OUT
4	3.00
1 1	3 1 2 4
2 1	
1 2	
3 1	

3.4. Soluțiile problemelor

3.4.1. Rucsac

Deoarece obiectele nu pot fi luate decât întregi, (nu se pot tăia în bucăți), problema se mai numeşte şi *problema* 0-1 *a rucsacului.*

Această problemă o vom rezolva folosind o metodă euristică. Soluția găsită nu va fi întotdeauna optimă, dar va fi apropiată de aceasta. Se va calcula rapid şi va fi uşor de implementat. O soluție exactă se poate afla pentru seturi de date relativ mici dacă se foloseşte metoda *backtracking*.

Vom sorta obiectele descrescător după valoarea câştigului unitar şi vom alege obiecte până se întâlneşte una din următoarele situații:
- obiectele alese au o greutate totală egală cu capacitatea rucsacului (soluție optimă);
- mai există loc în rucsac, dar nu se mai pot selecta obiecte care să acopere greutatea rămasă (în unele situații soluția poate fi optimă, iar în altele, nu).

```
Algoritm Selecție-Greedy(gr,val,n,G):
                        { ordonăm obiectele descrescător după valoarea câştigului }
  Ordonare(n,gr,val)
  i ← 0
  cât timp (G > 0) şi (i < n) execută:
    i ← i + 1
    dacă gr_i ≤ G atunci
      G ← G - gr_i                                  { obiectul i se poate selecta }
```

```
      câştig ← câştig + val_i
      k ← k + 1
      x_k ← i
    sfârşit dacă
  sfârşit cât timp
  scrie x_1, ..., x_k
sfârşit algoritm
```

Exemple

1. Fie $n = 5$, $G = 10$.

	1	2	3	4	5
Greutate	2	4	5	2	6
Valoare	4	20	1	3	3

Vom calcula, pentru fiecare obiect, valoarea pe unitatea de greutate.

Valoare/greutate	2	5	0.2	1.5	1.5

Vom selecta obiectele în ordine descrescătoare a raportului valoare/greutate.

Se obține soluția 1, 2, 4 care este soluție optimă a problemei.

2. $n = 3$, $G=8$

	1	2	3
Greutate	5	4	4
Valoare	6	4	3

În acest caz soluția optimă este formată din obiectele 2 şi 3, dar algoritmul construieşte doar soluția formată din primul obiect.

Algoritmul nu asigură obținerea soluției optime, dar este foarte rapid.

3.4.2. Expoziție

Vom rezolva problema folosind o euristică greedy. Soluția găsită nu va fi întotdeauna optimă, dar va fi apropiată de aceasta şi va avea avantajul că se va calcula rapid şi va fi uşor de implementat. O soluție exactă se poate afla pentru seturi de date relativ mici dacă se foloseşte metoda backtracking.

Algoritmul transcrie, cu mici modificări, algoritmul general *greedy*.

Din mulțimea A = { mulțimea pavilioanelor de coordonate (i, j) din matrice } se va construi o submulțime B = { mulțimea pavilioanelor în care se instalează foişoare }.

Inițial $B = \varnothing$. Treptat se adaugă din A elementele cele mai „promițătoare”. Un pavilion (i, j) este „promițător”, dacă din foişorul corespunzător lui se supraveghează un număr maxim de pavilioane.

Algoritmul se termină când se asigură supravegherea tuturor pavilioanelor.

În algoritmul descris alegem de fiecare dată un optim local. Acest fapt ***nu*** asigură optimalitatea globală a soluției.

Exemplu

Fie o hartă de dimensiuni 2 × 10. Valorile 1 reprezintă pavilioane care trebuie supravegheate (obiective), iar valorile 0 reprezintă pavilioane care nu trebuie păzite.

1 0 0 0 1 0 1 0 0 1
1 0 0 0 0 1 0 0 0 0

Se vor alege, în ordine, elementele mulțimii B:
1. (1, 6) asigură supravegherea a 3 obiective: (1, 5), (1, 7) şi (2, 6);
2. (1, 1) asigură supravegherea unui obiectiv: (2, 1);
3. (1, 7) asigură supravegherea unui obiectiv: (1, 10);
4. (2, 1) asigură supravegherea unui obiectiv: (1, 1).

Deci $|B| = 4$. Dar mulțimea $B' = \{(1, 2), (1, 8), (2, 2)\}$ are doar 3 elemente şi este soluția optimă a problemei.

În algoritm se repetă următorii paşi până la verificarea tuturor obiectivelor:
1. Se determină locul (i, j) în care trebuie instalat un foişor care asigură supravegherea unui număr maxim de obiective.
2. Se adaugă (i, j) la soluție.
3. Se marchează obiectivele supravegheate din (i, j), pentru ca acestea să nu fie luate în considerare în continuare.
4. Se actualizează numărul de obiective supravegheate.

3.4.3. Sportivi

Cele m numere care reprezintă duratele antrenamentelor sportivilor trebuie împărțite în n grupe, astfel încât maximul sumelor elementelor din fiecare grupă să aibă valoare minimă.

Pentru această problemă se poate obține rezultatul exact aplicând metoda backtracking, dar pentru seturi mari de date ea este practic inutilizabilă, având un timp de execuție foarte mare.

În această situație este mai utilă o metodă euristică de rezolvare a problemei. Prin aceasta vom obține o soluție care nu va fi întotdeauna cea optimă, dar va fi apropiată de ea. Programul va fi uşor de implementat şi va avea un timp de execuție rapid. Vom încerca să dispunem sportivii la aparate, astfel încât timpii de ocupare ai fiecărui aparat să aibă valori apropiate.

```
Subalgoritm Euristică_Greedy:
  dacă n ≥ m atunci
    scrie Maximum(t_i, i = 1, 2, …, m)
    pentru i=1,m execută:
      scrie t_i
    sfârşit pentru
  altfel
    ordonează descrescător şirul t
    pentru i=1,m execută:
      repartizează pentru aparatul i (sportivul care se antrenează) timpul t_i
    sfârşit pentru
    pentru i=m+1,n execută:
      selectează aparat dintre cele disponibile (k)
      repartizează pentru aparatul k (sportivul care se antrenează) timpul t_i
    sfârşit pentru
  sfârşit dacă
sfârşit subalgoritm
```

Dacă $n \geq m$ (numărul aparatelor este mai mare sau egal cu numărul sportivilor), vom plasa din start câte un sportiv la un aparat. Timpul minim necesar terminării tuturor antrenamentelor va fi egal cu valoarea elementului maxim din şirul duratelor.

Dacă $n < m$, vom sorta descrescător şirul care memorează duratele de antrenament. Parcurgem acest şir şi repartizăm sportivii (reordonați) după cum urmează:

a) plasăm primii n sportivi la aparate;
b) fiecare sportiv rămas îl plasăm la aparatul care devine disponibil cel mai repede.

În implementarea algoritmului memorăm într-un tablou de înregistrări datele despre fiecare aparat:

- timpul total de ocupare;
- indicele unui sportiv x, care se antrenează la acest aparat.

Lista sportivilor care se antrenează la acelaşi aparat o reconstituim plecând de la sportivul x, folosind tabloul *next*, ştiind că:
$next_x = y$ dacă la aparatul la care se antrenează x, urmează y;
$next_x = 0$ dacă la aparatul respectiv x este ultimul.

Pentru a afişa timpul minim cerut, vom determina valoarea maximă a timpilor de ocupare a fiecărui aparat.

3.4.4. Răsplata

Vom stabili criterii referitoare la ordinea în care vor fi selectați elevii. Aceste criterii le vom verifica la fiecare pas, în următoarea ordine:

a) vom alege elevii cu număr minim de opțiuni;
b) vom alege elevii pentru care prima opțiune are punctaj maxim;

c) vom alege o opțiune a cărei apariții pe listele celorlalte persoane conduce la un punctaj minim.

Exemplu

Fie $n = 3$ şi listele de opțiuni:

Persoana 1	3 opțiuni	3 , 2, 1
Persoana 2	o opțiune	3
Persoana 3	o opțiune	2

La prima alegere, aplicăm criteriul a). Vom avea de ales între elevii 2 şi 3. Aplicăm criteriul b), dar pentru că ambele opțiuni au punctaj maxim (3 puncte) aplicăm criteriul c). Alegerea elevului 2 cu excursia 3 duce la obținerea unui punctaj de 2 puncte. Alegerea elevului 3 cu excursia 2 duce la obținerea unui punctaj de 3 puncte. Deci vom alege întâi elevul 3 cu opțiunea 2.

În continuare vom aplica din nou criteriile, pentru a doua alegere. În urma aplicării criteriului a) se va alege elevul al doilea, cu opțiunea 3. Ultima dată se alege primul elev, cu singura opțiune posibil de îndeplinit în acest moment.

Vom avea un indice de mulțumire a persoanelor *Total* = 3 + 3 +1 = 7 puncte.

Observații

- Vom memora opțiunile elevilor într-o matrice x de dimensiuni $n \times (n + 1)$ astfel:
- în $x[i, 0]$ memorăm numărul de opțiuni ale elevului i;
- în $x[i, 1]$, $x[i, 2]$, … memorăm opțiunile elevului i.
- Când se „rezolvă” un elev i, se codifică $x[i, 0] = n + 1$.
- Soluția se memorează în şirurile y (elevii) şi z (excursiile alese);

```
Subalgoritm Determină_elevi(n,x,total,y,z):
  nr ← 0
  total ← 0
  repetă                              { se alege elevul aplicând criteriile pe rând }
    elev_ales ← alege_minim(opt)
    nr ← nr + 1
    y[nr] ← elev_ales
    z[nr] ← x[elev_ales,opt]
    șterge_opțiuni(elev_ales,opt)     { se actualizează opțiunile celorlalți }
    total ← total + n - opt + 1 { se actualizează total în urma alegerii }
  până când nu mai există elevi cu opțiuni posibil de îndeplinit
     { se repartizează în excursiile rămase elevii care nu mai pot fi satisfăcuți deloc }
  repartizează_restul
sfârşit subalgoritm
```

Problema se poate rezolva mai economic din punct de vedere al memoriei, folosind alocarea dinamică (vezi capitolul 9).

3.4.5. Rețea liniară

Deoarece toți algoritmii de rezolvare a acestei probleme sunt exponențiali, vom rezolva problema folosind o metodă euristică simplă.

Vom alege ca fiind unul din capetele lanțului primul calculator. Pe acesta îl vom lega de cel mai apropiat calculator. Vom repeta algoritmul, selectând la fiecare pas dintre calculatoarele nelegate încă, pe cel mai apropiat de ultimul introdus în rețeaua liniară.

Acest algoritm îl vom repeta alegând ca şi calculator de start pe fiecare dintre calculatoarele existente. Se va reține lanțul al cărui lungime este minimă.