INFORMATICĂ · CLASA A XI-A · METODE DE PROGRAMARE (INTENSIV) · LECȚIA 15

Metoda Greedy

Durată: 60 min Nivel: intensiv informatică / olimpiadă Înainte: lecțiile x-04, x-12

CE VEI ȘTI DUPĂ LECȚIE

- Înțelegi ideea Greedy: la fiecare pas alegi varianta care pare cea mai bună acum.
- Rezolvi problemele clasice: spectacole, rucsac fracționar, plata cu număr minim de monede (când merge).
- Știi să verifici (sau să combați cu un contraexemplu) corectitudinea unei strategii greedy.

1. Ideea: alegerea „lacomă”

Ai de dat rest 276 de lei și vrei cât mai puține bancnote. Natural: dai întâi cea mai mare bancnotă care încape (200), apoi 50, 10, 10, 5, 1. La fiecare pas ai făcut **alegerea cea mai bună în acel moment** și nu te-ai mai răzgândit. Asta este **metoda Greedy** (lacomă).

Schema generală:

1. ordonezi candidații după un **criteriu** bine ales (aici e toată arta);
2. parcurgi candidații în ordine; pe fiecare îl **iei** dacă nu strică ce ai deja;
3. nu te întorci niciodată asupra unei alegeri.

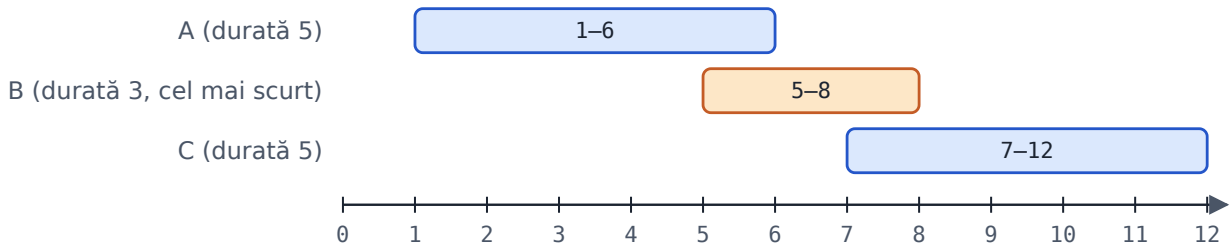
ATENȚIE — GREȘELI FRECVENTE

Greedy este rapid (de obicei O(n log n), cât costă sortarea), dar **nu merge întotdeauna**. Un criteriu care „pare” bun poate da răspunsuri greșite. Regula de aur: **caută un contraexemplu înainte să scrii codul.**

2. Problema spectacolelor

Sunt n spectacole, fiecare cu ora de început și ora de sfârșit. Ce număr maxim de spectacole poate vedea o persoană fără suprapuneri?

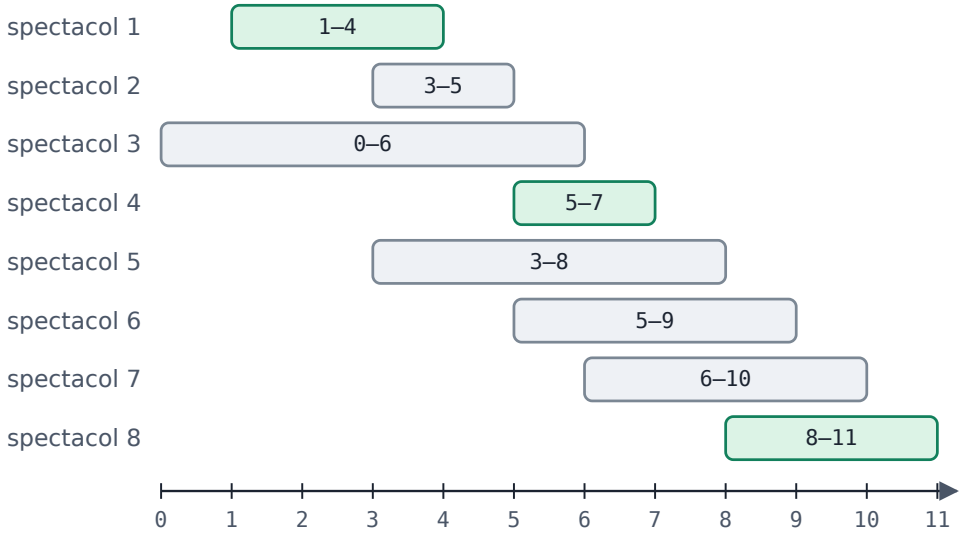
Trei criterii posibile: (a) după ora de început, (b) după durată, (c) după **ora de sfârșit**. Doar al treilea este corect.



cel mai scurt este B, dar se suprapune cu A și C: 1 spectacol; optim: A și C = 2

Criteriul durata, greșit

De ce după sfârșit? Spectacolul care se termină cel mai devreme îți lasă cel mai mult timp liber pentru restul. Orice soluție optimă poate fi „schimbată" să înceapă cu el fără să piardă niciun spectacol (argument de schimb: dacă soluția optimă începe cu alt spectacol, care se termină mai târziu, îl înlocuiești cu cel mai devreme terminat; nu se suprapune cu nimic din restul).



alegem 1–4, apoi primul care începe la ≥ 4 (5–7), apoi primul care începe la ≥ 7 (8–11): 3 spectacole

Spectacole

Trace pe cele 8 spectacole, sortate după sfârșit:

Spectacol (st–sf)	`st ≥ ultimSf` ?	Ce fac	ultimSf
1–4	1 ≥ 0 da	îl aleg	4
3–5	3 ≥ 4 nu	sar	4
0–6	0 ≥ 4 nu	sar	4
5–7	5 ≥ 4 da	îl aleg	7
3–8	3 ≥ 7 nu	sar	7
5–9	5 ≥ 7 nu	sar	7
6–10	6 ≥ 7 nu	sar	7
8–11	8 ≥ 7 da	îl aleg	11

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

struct spectacol { int st, sf; } v[1001];
int n;

bool cmp(spectacol a, spectacol b)
{
    return a.sf < b.sf;                      // crescător după ora de sfârșit
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> v[i].st >> v[i].sf;
    sort(v + 1, v + n + 1, cmp);

    int ultim = 0, nr = 0;
    spectacol ales[1001];
    for (int i = 1; i <= n; i++)
        if (v[i].st >= ultim)                // nu se suprapune cu ultimul ales
        {
            ales[++nr] = v[i];
            ultim = v[i].sf;
        }
    cout << nr << '\n';
    for (int i = 1; i <= nr; i++) cout << ales[i].st << ' ' << ales[i].sf << '\n';
    return 0;
}
```

DE CE AȘA?

De ce `>=` și nu `>`? Un spectacol care începe exact când se termină celălalt nu se suprapune (de la 4 la 4 nu e timp comun). Verifică în enunț dacă „se suprapun” include și capetele. **De ce sortare cu o funcție `cmp`?** `sort` știe să ordoneze numere, nu structuri; îi spui tu cum compară două spectacole.

Intrare / ieșire

Intrare	Ieșire
`8` · `1 4` · `3 5` · `0 6` · `5 7` · `3 8` · `5 9` · `6 10` · `8 11`	`3` · `1 4` · `5 7` · `8 11`

3. Rucsacul fracționar

Ai un rucsac de capacitate C și n obiecte, fiecare cu greutate G și valoare V. Poți lua **fracțiuni** din obiecte (ca la făină sau nisip). Care este valoarea maximă?

Criteriul: raportul **V/G** (lei pe kilogram), în ordine descrescătoare. Umpli rucsacul cu „cea mai scumpă făină” mai întâi.

	obiect 1	obiect 2	obiect 3
greutate G	10	20	30
valoare V	60	100	120
raport V/G	**6**	**5**	4
luăm	**10 kg (tot)**	**20 kg (tot)**	**20 din 30 kg**

valoare: 60 + 100 + 120 · 20/30 = 240 lei

Rucsac

```cpp
#include <iostream>
#include <algorithm>
#include <iomanip>
using namespace std;

struct obiect { double g, v; } o[101];
int n;

bool cmp(obiect a, obiect b)
{
    return a.v / a.g > b.v / b.g;             // raport descrescător
}

int main()
{
    double C;
    cin >> n >> C;
    for (int i = 1; i <= n; i++) cin >> o[i].g >> o[i].v;
    sort(o + 1, o + n + 1, cmp);

    double total = 0;
    for (int i = 1; i <= n && C > 0; i++)
    {
        if (o[i].g <= C)                      // încape întreg
        {
            total += o[i].v;
            C -= o[i].g;
        }
        else                                  // luăm doar o fracțiune
        {
            total += o[i].v * C / o[i].g;
            C = 0;
        }
    }
    cout << fixed << setprecision(2) << total;
    return 0;
}
```

DE CE AȘA?

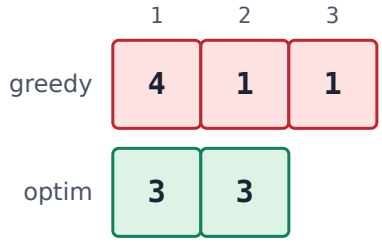
De ce raportul V/G și nu doar V? Un obiect cu valoare mare dar foarte greu „consumă” capacitatea. Contează câtă valoare obții pe **unitatea de capacitate**. **De ce nu merge pentru obiecte întregi?** Cu aceleași date și obiecte nefracționabile, greedy ia obiectele 1 și 2 (30 kg, 160 lei) și nu mai încape al treilea; dar obiectele 2 și 3 încap (50 kg, 220 lei). Fracțiunea este ceea ce face criteriul optim; fără ea trebuie programare dinamică (lecția XI.16).

4. Rest cu număr minim de bancnote

Cu bancnotele românești 500, 200, 100, 50, 10, 5, 1: dai mereu cea mai mare bancnotă care încape. Pentru 276: 200 + 50 + 10 + 10 + 5 + 1.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int b[] = {500, 200, 100, 50, 10, 5, 1};
    int S;
    cin >> S;
    for (int i = 0; i < 7; i++)
        while (S >= b[i])
        {
            cout << b[i] << ' ';
            S -= b[i];
        }
    return 0;
}
```



greedy: 3 monede · optim: 2 monede → pentru acest set de monede, greedy GREȘEȘTE

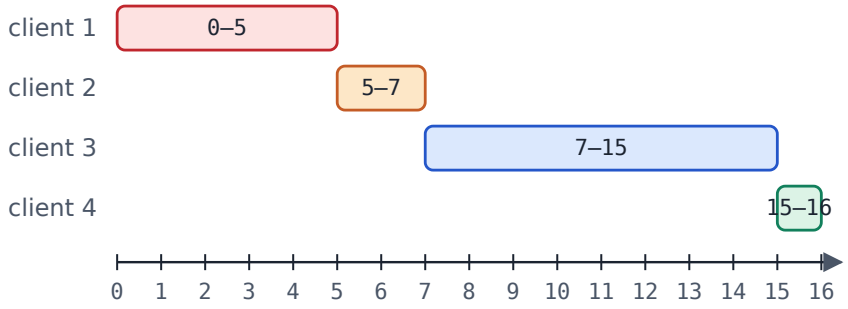
Greedy greșește

ATENȚIE — GREȘELI FRECVENTE

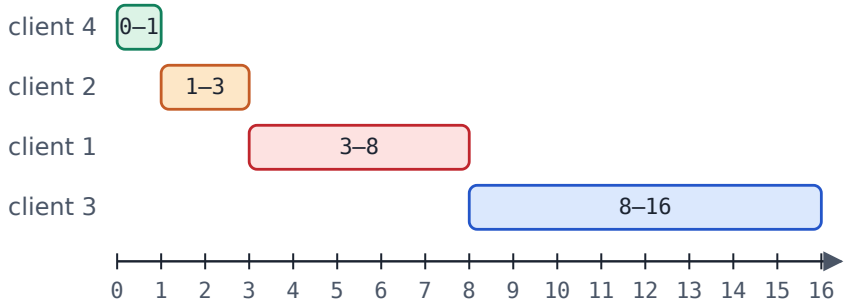
Pentru **alt** sistem de monede greedy poate greși: cu monede {1, 3, 4} și suma 6, greedy dă 4 + 1 + 1 (3 monede), iar optimul este 3 + 3 (2 monede). Greedy funcționează pe bancnotele reale pentru că fiecare valoare este multiplu convenabil al celor mai mici; pentru un set oarecare de monede se folosește programare dinamică.

5. Clienți la casă: cei rapizi primii

n clienți, fiecare cu durata de servire t_i, la o singură casă. În ce ordine îi servim ca **suma timpilor de așteptare** (până când începe să fie servit fiecare) să fie minimă?



ordinea 5, 2, 8, 1: așteptări 0 + 5 + 7 + 15 = 27



ordinea 1, 2, 5, 8 (cei rapizi întâi): 0 + 1 + 3 + 8 = 12

Clienți

De ce cei rapizi primii? Durata primului client se adaugă la așteptarea tuturor celorlalți, durata celui de-al doilea la așteptarea tuturor celor de după el, și tot așa. Un client cu durată mare pus primul ar „taxa” pe toți ceilalți.

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int n, t[101];
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> t[i];
    sort(t + 1, t + n + 1);                    // cei mai rapizi primii

    long long asteptare = 0, curent = 0;
    for (int i = 1; i <= n; i++)
    {
        asteptare += curent;                    // clientul i așteaptă cât au durat cei
dinaintea lui
        curent += t[i];
    }
    cout << asteptare;
    return 0;
}
```

DE CE AȘA?

De ce `long long`? Pentru 100 000 de clienți cu timpi mari, suma depășește $2 \cdot 10^9$. Fă un obicei: dacă adunăm multe valori, `long long`.

Trace pentru `t = 5 2 8 1` (după sortare 1 2 5 8):

i	t[i]	asteptare += curent	curent
1	1	0	1
2	2	1	3
3	5	4	8
4	8	12	16

6. Cum alegi criteriul și cum verifici

Problemă	Criteriu corect	Criteriu care pare bun, dar greșește
spectacole	ora de **sfârșit**	durata; ora de început
rucsac fracționar	raport V/G	doar V; doar G
clienți la casă	durata crescător	ordinea sosirii
rest cu bancnote reale	cea mai mare care încape	(merge)
rest cu monede {1, 3, 4}	—	cea mai mare care încape (greșește)

Ghid în trei pași:

1. propune un criteriu;
2. încearcă să-l spargi cu cazuri mici (n = 3 sau 4), cu cazuri „la limită" (egalități);
3. dacă nu găsești contraexemple, încearcă un argument de schimb: „dacă soluția optimă ar face altfel, o pot transforma în una cel puțin la fel de bună care face ca criteriul meu".

7. Greșeli frecvente

Greșeala	Efect	Corect
spectacole sortate după început	rezultat prea mic	după sfârșit
`cmp` cu `<=`	`sort` poate bloca sau da eroare	`<` strict
rucsac cu `int` și diviziune	`V / G` întreg, zero	`double`
`ultim = 0` și spectacole care încep la 0, cu `>`	pierzi primul	`>=`
criteriu neverificat	greșit pe unele teste	contraexemple

8. Exersează pe hârtie

1. Găsește un contraexemplu pentru „la spectacole, alege mereu spectacolul care începe cel mai devreme".

2. Spectacole: (1,3), (2,5), (4,6), (6,8), (5,9), (8,10). Câte alege greedy (după sfârșit)?

3. Rucsac fracționar, C = 10, obiecte (G, V): (4, 8), (5, 6), (6, 12). Care este valoarea maximă?

4. Rest de 289 lei cu bancnotele 500, 200, 100, 50, 10, 5, 1. Câte bancnote minime?

5. Cu monede {1, 5, 6, 9} și suma 11, ce dă greedy? Care este optimul?

6. Clienți cu duratele 3, 1, 4, 2: care este suma timpilor de așteptare în ordinea sosirii? Dar în ordinea optimă?

7. Se dau n numere întregi și un număr k. Alege exact k numere cu suma maximă. Ce strategie folosești?

8. (BAC) Se dau n activități cu termen limită și profit, fiecare durând o unitate de timp. Dacă poți face o activitate pe unitate de timp, în ce ordine ai alege activitățile? (Idee: cele cu profit maxim primele.)

9. Se dă un număr cu cifre. Elimină k cifre ca numărul rămas să fie **maxim**. Pentru 1924 și k = 2 ce obții?

10. Spune dacă greedy-ul „ia mereu cea mai mare monedă care încape" este corect pentru monedele {1, 2, 5, 10, 20, 50}. Încearcă să găsești un contraexemplu.

9. Mai departe (pentru cine vrea mai mult)

Elimină k cifre pentru numărul maxim. Parcurgi cifrele; cât timp ultima cifră păstrată este mai mică decât cea curentă și mai ai de eliminat, o scoți.

```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s[105], r[105];
    int k, lung = 0;
    cin >> s >> k;
    int n = strlen(s);
    for (int i = 0; i < n; i++)
    {
        while (k > 0 && lung > 0 && r[lung - 1] < s[i])
        {
            lung--;                                  // scoatem o cifră mai mică din urmă
            k--;
        }
        r[lung++] = s[i];
    }
    lung -= k;                                        // dacă au rămas de eliminat, tăiem de la
coadă
    r[lung] = 0;
    cout << r;
    return 0;
}
```

Argumentul de schimb, pe scurt. Pentru a arăta că un greedy e corect, presupui o soluție optimă care nu respectă alegerea greedy și arăți că o poți modifica, fără să o înrăutățești, ca să o respecte. Se repetă pas cu pas.

Alte probleme clasice. Acoperire de intervale cu număr minim de segmente; cel mai scurt drum cu costuri pozitive (Dijkstra, lecția XI.19); arbore parțial de cost minim (Prim, Kruskal); coduri Huffman (compresie).

Greedy sau programare dinamică? Dacă ai un contraexemplu pentru alegerea locală, sau dacă alegerea de acum depinde de ce faci mai târziu, ai nevoie de programare dinamică (XI.16).

Rezumat

- Greedy: sortezi după un criteriu, iei ce încape, nu te mai întorci.
- Spectacole: după **ora de sfârșit**. Rucsac fracționar: după V/G. Clienți: durata crescător.
- Nu merge întotdeauna (monede {1, 3, 4}, rucsac discret): caută contraexemple.
- Sortare cu `cmp` strict (`<`); sume mari în `long long`.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] #353 **spectacole** — ușoară
  https://www.pbinfo.ro/probleme/353/spectacole

- [ ] #91 **masini** — ușoară
  https://www.pbinfo.ro/probleme/91/masini
- [ ] #1877 **kmax** — ușoară
  https://www.pbinfo.ro/probleme/1877/kmax

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #92 **proiecte** — ușoară
  https://www.pbinfo.ro/probleme/92/proiecte
- [ ] #341 **bomboane** — ușoară
  https://www.pbinfo.ro/probleme/341/bomboane
- [ ] #2271 **prodmax1** — ușoară
  https://www.pbinfo.ro/probleme/2271/prodmax1
- [ ] #4220 **alimentarea-masinii** — ușoară
  https://www.pbinfo.ro/probleme/4220/alimentarea-masinii
- [ ] #1340 **rucsac** — medie
  Varianta fracționară: sortare după valoare/greutate.
  https://www.pbinfo.ro/probleme/1340/rucsac
- [ ] #400 **pachete** — medie
  https://www.pbinfo.ro/probleme/400/pachete

Citește și exersează în plus

- Metoda Greedy — prezentare generală https://www.pbinfo.ro/articole/16619/metoda-greedy
- USACO Guide — Greedy Algorithms with Sorting — aceeași idee, probleme de concurs, explicații pe niveluri (engleză) https://usaco.guide/silver/greedy-sorting
- CSES — Movie Festival / Tasks and Deadlines — spectacole (festival de filme) și clienți cu deadline-uri (engleză) https://cses.fi/problemset/task/1629
- Metoda Greedy — Wikipedia — definiție, proprietăți, când este optimă alegerea locală https://ro.wikipedia.org/wiki/Algoritm_greedy

Răspunsuri

Exersează pe hârtie

1. Spectacole (0,10), (1,2), (3,4): cel mai devreme începe (0,10), care acoperă tot; greedy ia 1 spectacol, optimul este 2 ((1,2) și (3,4)).

2. Sortate după sfârșit: (1,3), (2,5), (4,6), (6,8), (5,9), (8,10). Alege (1,3), apoi (4,6) (4 ≥ 3), apoi (6,8) (6 ≥ 6), apoi (8,10): **4** spectacole.

3. Rapoarte: 2, 1,2, 2. Obiectele 1 (4 kg, 8 lei) și 3 (6 kg, 12 lei) au raportul 2; le luăm întregi: 10 kg, 20 lei. Valoarea maximă: **20**.

4. 289 = 200 + 50 + 10 + 10 + 10 + 5 + 1 + 1 + 1 + 1: **10** bancnote. Verificare: 200 + 50 = 250; + 30 (3 × 10) = 280; + 5 = 285; + 4 × 1 = 289.

5. Greedy: 9 + 1 + 1 (3 monede); optim: 5 + 6 (2 monede).

6. În ordinea sosirii (3, 1, 4, 2): așteptări 0 + 3 + 4 + 8 = **15**. Ordinea optimă (1, 2, 3, 4): 0 + 1 + 3 + 6 = **10**.

7. Sortezi descrescător și aduni primele k (dacă toate pozitive); cu numere negative tot primele k, fiind cele mai mari.

8. Sortezi activitățile descrescător după profit; fiecare se pune în ultima unitate de timp liberă dinaintea termenului (dacă există); dacă nu există loc, se sare.

9. Rezultatul este **94** (programul din „Mai departe”).

10. Da, pentru {1, 2, 5, 10, 20, 50} greedy este corect (fiecare monedă este multiplu al unei monede mai mici sau are structură care se potrivește): nu există contraexemplu. Greedy nu este corect, de exemplu, pentru {1, 3, 4}.