CURS 10

METODA GREEDY - CONTINUARE

- **Problema rucsacului**
  - n obiecte $\begin{cases} g_1, g_2, \dots, g_n & \text{greutăți} \\ c_1, c_2, \dots, c_n & \text{câștiguri întregi} \end{cases}$

    G = capacitatea unui rucsac
  - O modalitate de încărcare a rucsacului e.î. câștigul total să fie max?
  - Variante → continuă = orice ob. poate fi tăiat => se face prin Greedy => $O(n \log_2 n)$
    → discretă (0/1) = orice ob. poate fi încărcat doar întreg
  - Criterii → doar greutăți // nu e optim.
    → doar câștiguri // nu e optim
    → cât mai mici și valoroase.

      Câștig unitar = $\frac{c_i}{g_i}$
  - Algoritmul Greedy
    1) Sortăm descresc. după câștigul unitar
    2) Pt. fiecare obiect:
       → dacă ob. încape complet => îl încărcăm complet.
       → dacă ob nu încape complet => încărcăm o parte din el a.î. să umplem rucsacul și după STOP

- Exemplu + dem. corectitudine

$G_1 = 65$ kg

$m = 7$

$c = \{100, 50, 150, 100, 20, 200, 50\}$ (O_1 O_2 O_3 O_4 O_5 O_6 O_7)

$g = \{20, 5, 30, 10, 10, 10, 25\}$

$Cu = \{5, 10, 5, 10, 2, 20, 2\}$

Obiect	Spațiu liber	Câștig total
—	65	0
O_6	$65 - 10 = 55$	$0 + 200 = 200$
O_2	$55 - 5 = 50$	$200 + 50 = 250$
O_4	$50 - 10 = 40$	$250 + 100 = 350$
O_1	$40 - 20 = 20$	$350 + 100 = 450$
O_3	$20 - 2/3 \cdot 30 = 0$	$450 + 2 \cdot 150/3 = 550$

O_5 și O_7 nu mai încap

Exchange arg e mai greu (avem în curs)

METODA DIVIDE ET IMPERA

- Condiția 1 = Divide

Problema se poate împărți în mai multe probleme de același tip și care au dim. datelor de intrare egale

- Condiția 2 = Impera

Soluția unei probleme se poate obține combinând sol. subpb.

- Algoritm general Divide et Impera:

```
def divimp(L, st, dr):
    if dr - st <= 1:
        return sol. pb. dir. rez
    mij = (st + dr) // 2
    sol_st = divimp(L, st, mij)
    sol_dr = divimp(L, mij+1, dr)
    return solutia(sol_st, sol_dr) // funcție / expr
```

- Exemplu: suma elem. dintr-o listă

```
def suma(L, st, dr):
    if dr == st:
        return L[dr]                    => 1
    mij = (st + dr) // 2                => 1
    sol_st = suma(L, st, mij)           => T(m/2)
    sol_dr = suma(L, mij+1, dr)         => T(m/2)
    return sol_st + sol_dr              => 1
// apel: s = suma(L, 0, len(L)-1)
```

$\Rightarrow 2^k + 2^{k+1} = 3m - 2 \Rightarrow O(3m-2)$

- Complexitate = T(m) = complex. rezolvării unei pb având datele de intrare egale cu $m = dr - st + 1$.

Pp $m = 2^k$ pt exemplul cu suma