# Seminar 1 -grupa 234 Problema rucsacului:

**Forma generala:**Avem *n* obiecte, fiecare obiect fiind caracterizat de valoare si greutate. Avem un rucsac de capacitate W.

* Doi vectori de lungime *n*; val[] si w[]; respective o valoare intreaga pozitiva W.

1. Problema continua a rucsacului.

Trebuie sa selectez obiecte sau bucati de obiecte astfel incat greutatea totala sa nu depaseasca valoarea W iar profitul strans sa fie maximizat.

Solutie:  
 - Sortam obiectele crescator greutate/valoare. Iteram prin lista sortata. Adaugam in rucsac fiecare obiect, cu conditia ca sa nu se depaseasca limita W. Daca odata cu ultimul obiect adaugat se depaseste greutatea W, acel obiect va fi taiat si se va adauga doar o bucata astfel incat greutatea totala sa fie exact W.

Val/greutate: [50/10; 30/5; 40/20; 50/20; 30/10, 20/15]

W=50;

Obiectele: [**30/5**; **50/10**; **30/10**; **50/20**; ***40/20***, 20/15] – sortate

Sol: 30+50+30+50+40\*5/20= 170  
  
Complexitate timp?  
O (n log n) – data de sortare, in rest linear.

Tehnica Greedy: se aplica in cazul problemelor de optim (tb sa construiesc o “solutie” care sa respecte anumite restrictii – restrictia greutatii toale – si sa maximizeze/minimizeze o functi – maximizarea profitului). La fiecare pas aleg elemental cel mai “atragator” in momentul respective si il adaug la constructive daca adaugarea lui este valida. Evident selectarea unor obiecte, altele devin indisponibile. Deci alegerea se face pe baza unei “euristici” – deci trebuie justificata corectitudinea algoritmului.   
  
Demonstratie corectitudine:

Reducere la absurd + “exchange argument”

Fie G – elementele selectate, in ordine, de catre algoritmul Greedy de mai sus. Fie O -o solutie optima, diferita de G, dar cu elementele sortate in rucsac dupa acelasi criteriu ca cel din G.

Din faptul ca G=/=O rezulta ca exista cu siguranta un ‘j’ astfel incat G[j]=/=O[j]

val(G[j])/greutate(G[j]) >= val(O[j])/greutate(O[j])

caz 1: ‘>’ interschimbam cel putin o bucata din O[j] cu o bucata de aceiasi greutate (dar valoare mai mare) din G[j] si obtinem o noua solutie “mai optima” decat O. Contradictie!

Caz 2: ‘=’ Gasim o solutie echivalenta cu G pana la acel pas. Continuam pana la urmatoarea diferenta.

O alternativa mai usoara de a justifica: Alegem O – solutie optima dar care se aseamana cel maim ult cu G (diferenta apare cel mai tarziu). Din exchange contruiesc o solutie “la fel de optima” ca O, dar care deifera de G mai tarziu. Contradictie!

1. 1/0 Knapsack Problem

Fiecare obiect ori este selectat in intregime, ori este ignorat. (Nu se pot taia obiectele)

Ce se intampla daca aplic aceiasi idee si in acest caz?

W=50  
obj=[60/10; 100/20; 120/30]

Greedy ar da solutia: 160  
optim: 220

Spoiler alert! Acest algoritm este un algoritm 0.5 aproximativ pt problema 1/0 a rucsacului.   
  
Solutia 1:

O solutie exhaustive. Pentru fiecare obiect calculez toate configuratiile in care acesta este selectat, respective neselectat si o aleg pe cea mai buna. Cod sursa: <https://onlinegdb.com/Hyb50ZubO>  
  
Complexitate: O(2^n) – worst case generam toate submultimile (pt fiecare element threadul de calcul se bifurca in doua)   
  
Solutia 2:

Programare Dinamica!  
  
Care este cerinta problemei: “Din n obiecte sa se selecteze unele astfel incat greutatea totala sa nu depaseasca W iar valoarea totala sa fie maxima”

Cum generalizam problema: “Din primele i obiecte, sa se selecteze unele astfel incat greutatea totala sa nu depaseasca un j, iar valoarea totala sa fie maxima”

DP[i][j]=valoarea maxima obtinuta folosind doar dintre primele i obiecte, fara a depasi greutatea totala j

DP[n][W] va fi raspunsul pt problema noastra.

Cum calculam matricea DP?

DP[0][j]=DP[i][0]=0

Pt celelalte pozitii (i,j>0)?

DP[i][j] = max(DP[i-1][j], val[i-1]+DP[i-1][j-w[i-1]]).

Complexitate timp: O(nxW) – complexitate **pseudo-polinomiala**   
Complexitate spatiu: se poate reduce la O(2xW) retinand doar ultimele 2 linii ale matricei.   
  
Greutatile sunt numere intregi! Asta imi permite maparea lui DP.