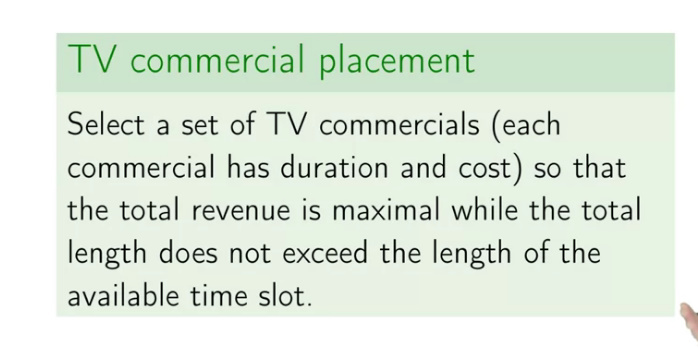
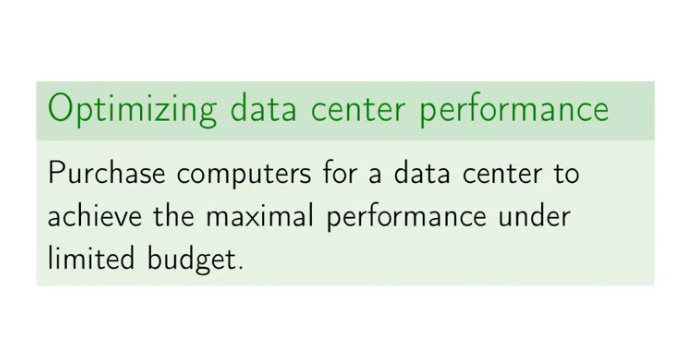
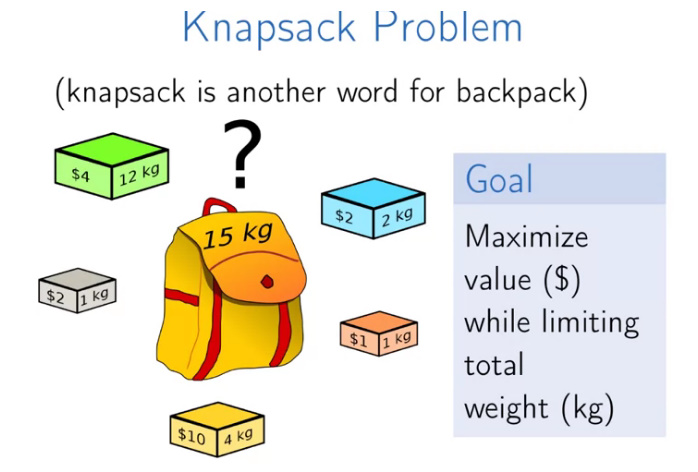
**Dynammic Programming Knapsack**

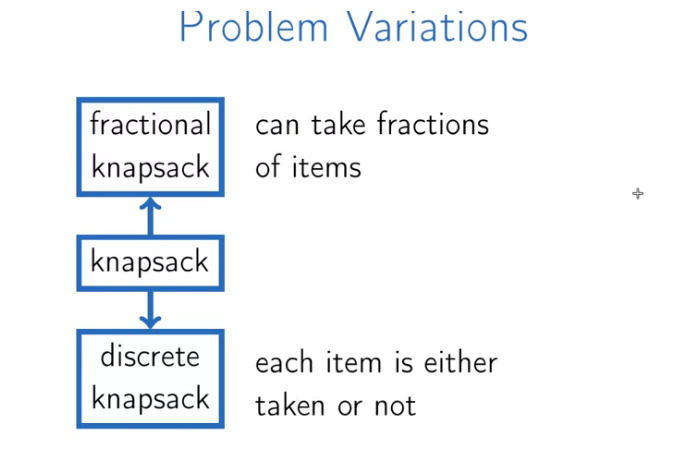
Probleme reale

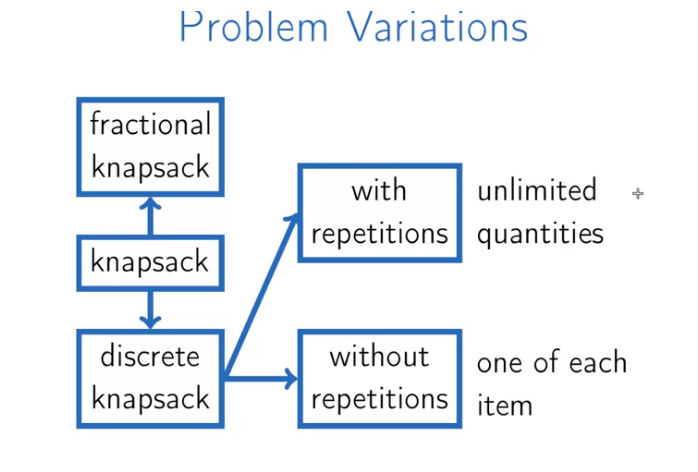


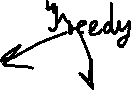


Knapsack problem:

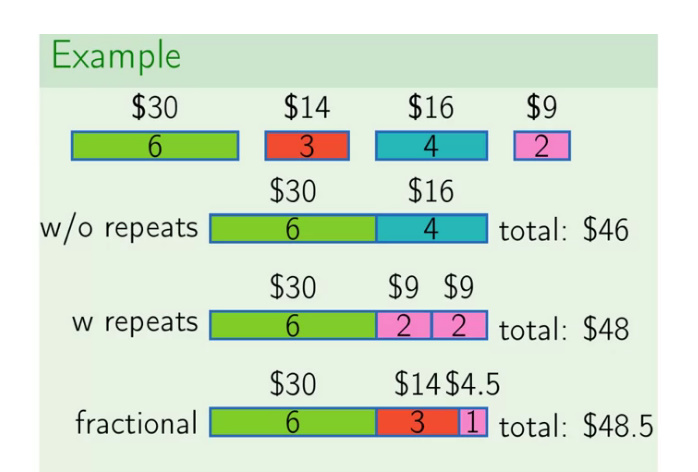


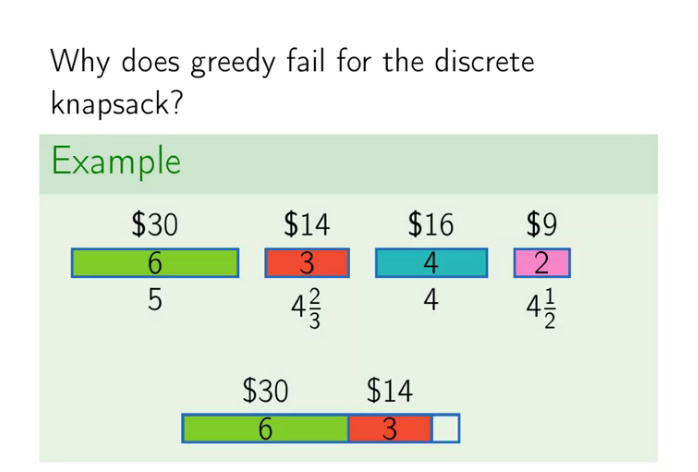






Totusi, problemele cu cantitate nelimitata de obiecte nu sunt practice, deci nici algoritmul Greedy.

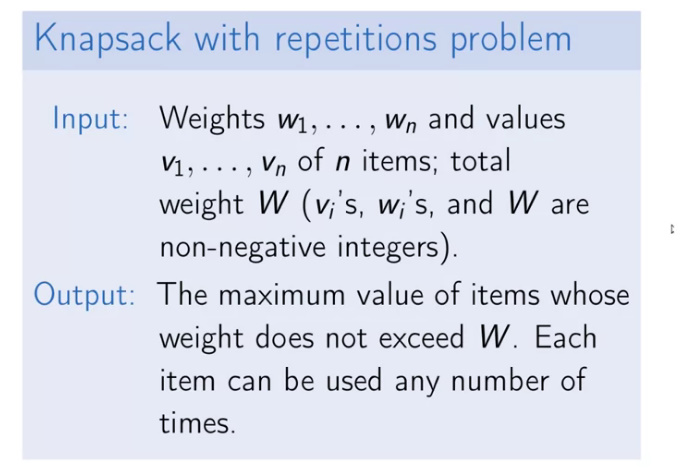


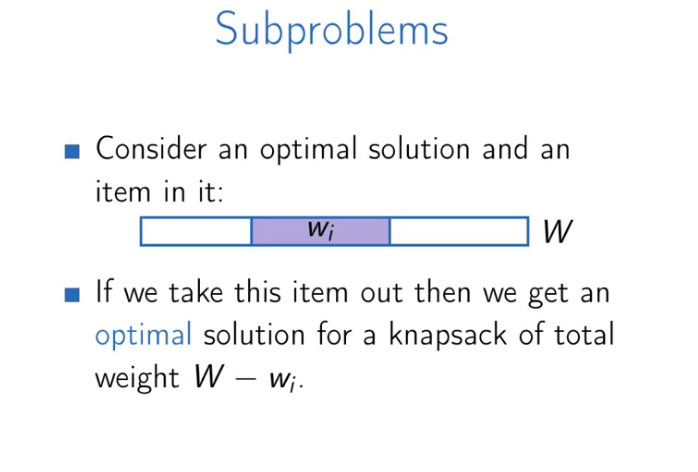


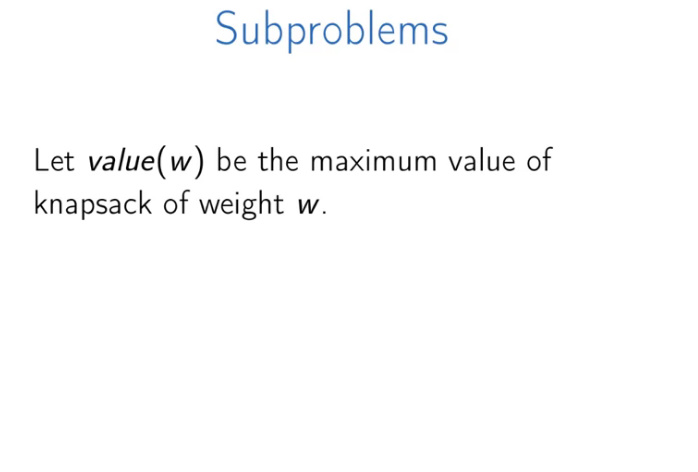
Deci cu Algoritmul Greedy mai e si problema ca spatiul ce ramane ar putea sa nu fie suficient pentru nici-un item ramas,cum e mai sus. Nu avem un item ce ocupa spatiul 1 si nu putem lua bucati din ele.

Deci nu e prea sigur sa luam mereu elementele cu cel mai mare pret per unitate raportat la greutate.

**Problema cu repetitii**

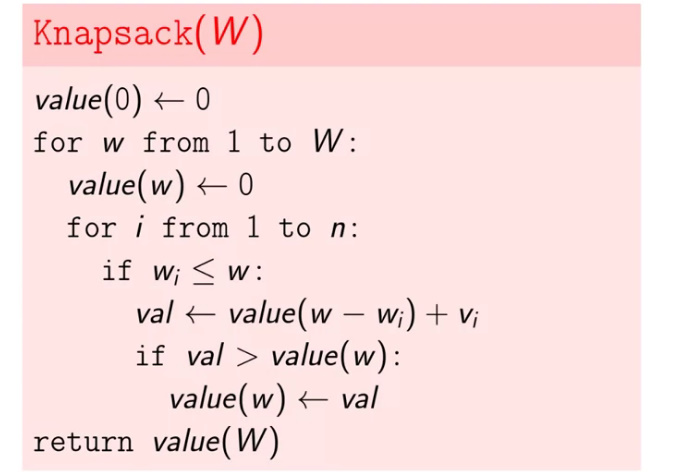






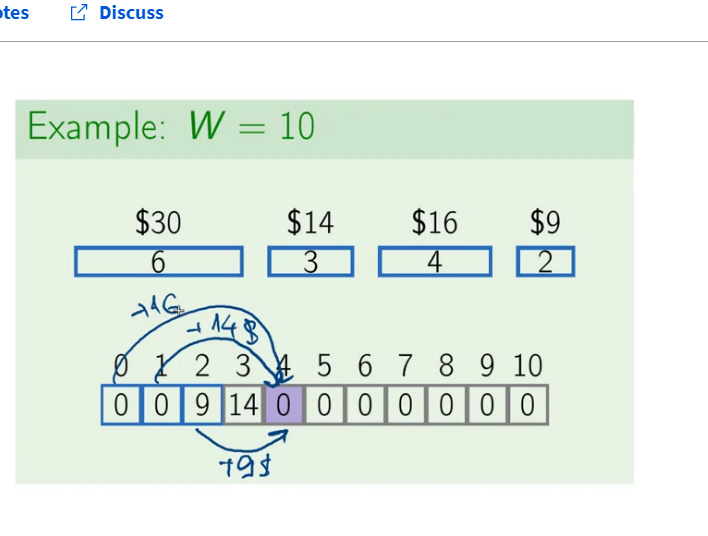
Cu alte cuvinte, daca avem un ruczac cu capacitatea W, atunci cand vom stabili o solutie optima pentru el, daca scoatem un item cu greutatea de exemplu w1, atunci itemele ramase for fi o solutie optima pentru un ruczac cu greutatea W-w1

Deci nu vom afla deodata solutia optima pentru ruczagul nostru, ci pentru ruczace mai mici si apoi vom merge pana la el



vi este valoarea itemului cu weight wi

Algoritmul lucreaza cam asa



Deci, pentru fiecare ruczag, care incepe de la greutatea 1 pana la 10, aa cum ruczagul nostru are 10, aflam optimal solution bazandune pe rezultatele de la ruczagurile precedente.

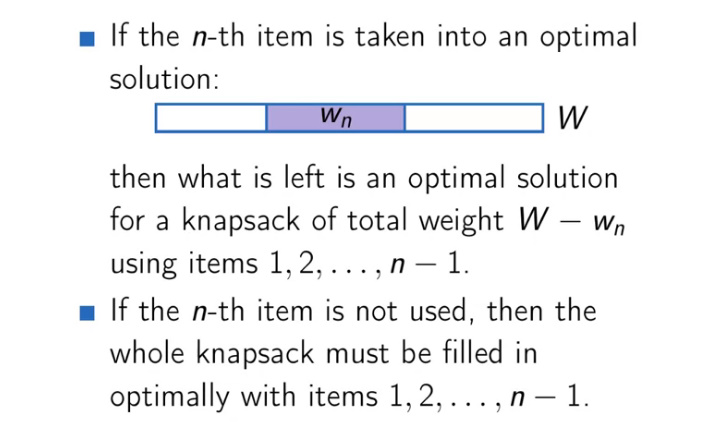
* Pentru 0 logic ca e 0
* Pentru weight 1 nu avem nimic, deci tot e 0
* Pentru 2 putem lua doar itemul cu weight 2 = 9$, caci avem 2-2 = 0, si la 0 nu avem nimic, deci avem 9$
* Pentru 3 putem avea un item cu weigh 2 = 9$, si deci 3-2 = 1, la 1 e 0, deci e doar 9$, sau putem avea un item cu weight 3=14$, deci 3-3 = 0 si la 0 e 0, asa dar aici e 14$. 14>9
* Pentru 4 putem avea optiunea ca :

1. luam un item de 2 = 9$, 4-2 = 2 , si la ruczagul cu weight 2 optim e 9$, deci avem 9$ + 9$ si la 2 avem 2 - 2 care e 0 si la 0 e 0$, deci 9$+9$ = 18$
2. luam un item cu weight 3 = 14$, apoi 4-3 = 1, vedem ce e la rucazgul cu weight 1, si acolo e 0$, deci 14$
3. luam un item cu weight 4 = 16$, apoi 4 – 4 = 0, deci doar 16$

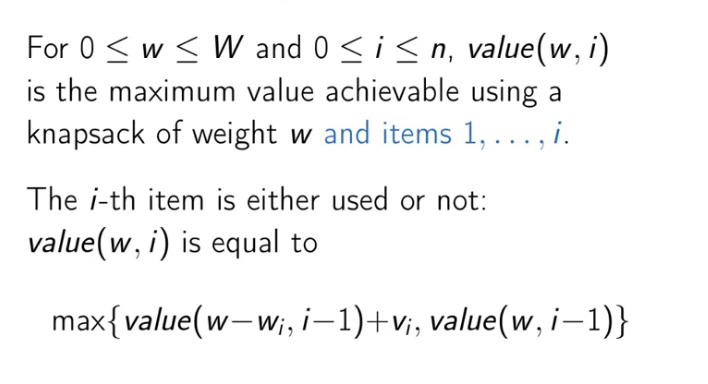
18 e cel mai mare, deci e optim.

**Knapsack fara repetitii**

* Aici deja nu mai putem folosi oridecate ori vrem unul si acelasi item, asa ca mai sus, ci doar o singura data.
* Aici ne bazam pe acelasi principiu ca daca ruczagul are weight maxim de N, atunci aflam valoarea optima pentru fiecare ruczac ce ar avea weight de la 0 la N.
* Totusi, nu mai putem adauga copii ale elementelor precedente, asa cum avem doar cate un item.



* Deci, cream tot ruczaguri de la de la 0 la W, folosint itemii de la 0 la n
* Acum, cand ajungem la ruczagul nostru, avem 2 situatii, fie el contine ultimul item, fie nu. Daca da, si il excludem, obtinem solutia pentru un ruczag cu weight W – wn si deci foloseste itemele de la 1 pana la n-1
* Daca nu il contine pe ultimul, atunci va contine iteme de la 1 la n-1



n – numarul de iteme

i – numarului unui item concret

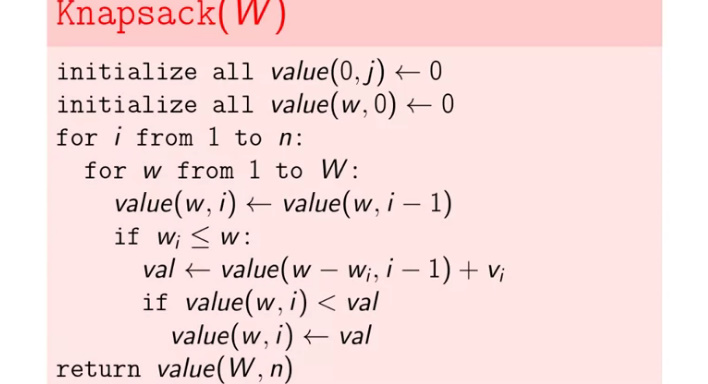
value(w,i) – valaorea unui knapsack cu weight w si cu primele i iteme

vi – valoarea itemului v

in max folosim cele 2 cazuri:

* elementul i este prezent, deci luam valoarea optima a unui knapsack precedent si punem valoarea itemului i
* elementul i nu e prezent, deci nu putem pune + vi, si lum valoare optima a knapsackului nostru, dar fara acel item.

Asa, le comparam si vedem care varianta e mai buna, cand e prezent itemul i sau nu.

* 

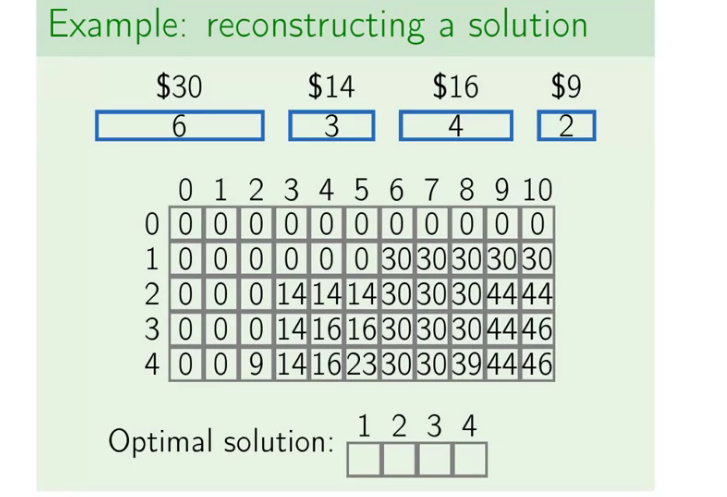
j – numar de knapsacks

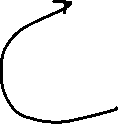
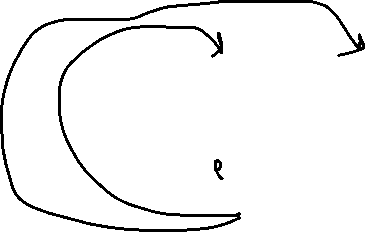
n – tot e nr de knapsacks

wi<=w verifica daca itemul incape in rugzac. Deci, daca de ex avem ruczag cu w = 2, e logic ca primul item nu va incapea in el, caci 6 > 2. w**i** e weight la itemul i

Pe coloana sunt puse itemele, pe linie ruczagurile. Mergem pe linie, nu pe coloana!

Deci, incepem cu un for ce merge pe linie prin fiecare coloana.

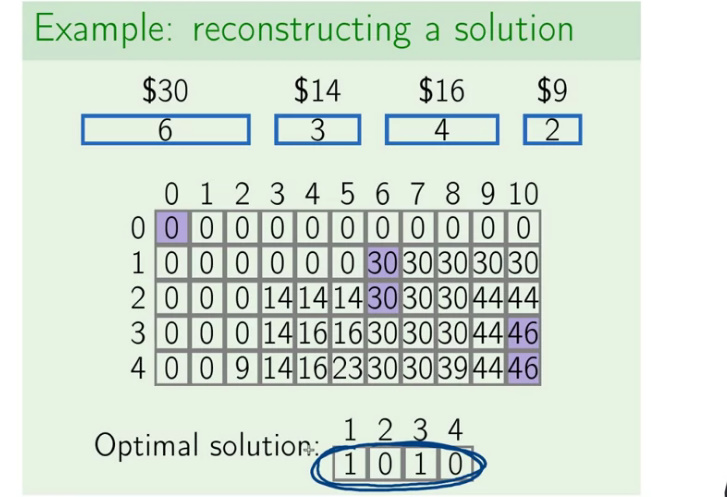




Ideea e ca la fiecare element, presupunem ca deocamdata valoareaa optima e a cea a ruczagului de mai sus, unde elementul i la sigur nu este. Acum, trebuie sa vedem care e valoarea unui ruczac ce nu contine acest item, dar care are spatiu minim suficient pentru a-l mai pastra si pe el, si vedem daca inlocuim tot ce s-a pus la ruczac de mai sus dupa acea weight cu el e mai mare. Pentru asta, din w actual scadem weight a itemului i, si ne ridicam cu o linie mai sus, si asa ajunge spatiu si pentru itemul testat.

* Luam linia 1:
* primul element nu incape in ruczaguri cu masa 1, 2,3,4,5. In schimb, incape in cele cu weight 6 7 8 9 si 10
* Luam linia 2, deci aici luam deja primele 2 elemente:
* in 1 si 2 nu incape nici unul din ele. In 3 4 si 5 incape doar al 2. In 6 ambele tot nu incap, dar deja incape unul din ambele. Luam valoarea optima de la ruczac de mai sus, adica care nu contine deocamdata al 2 element,dar nu are spatiu nici pentru al 3, si vedem acum, daca am scadea din w actual weight la item i, vom afla care ar fi valaorea optima la un ruczag cu weight = w – i, cu o linie mai sus, unde acel item nu exista la sigur, dar exista spatiu minim suficient pentru acest item testa.Deci, ne uitam la elementul 6-3 = 3 , cu o linie mai sus, si vedem ca e 0,deci daca are valoarea 0, are inca suficient spatiu pentru el, iar al 2 element e cu valoarea 14, deci 0 + 14 nu e > ca 30. Ramane 30 cum e mai sus
* Luam linia 3, deci luam primele 3 elemente
* in 1 si 2 nu incape nimic, in 3 doar itemul al 2
* in 4 incape deja al 2 sau al 3. Luam deocamdata valoarea de mai sus, unde la sigur al 3 item nu e, si presupunem ca ea e optima, deocamdata. Acum, ne ducem la elementul 4 – 4 = 0 dar din linia 2, unde e valoarea optima fara al 3 si e suficient spatiu si pentru i,si e 0. 0 +16 > 14, deci 16 va fi, nu 14 cum era
* La coloana 10, luam optim ca 44, deocamdata, de sus. Acum aici stim la sigur ca al 3 item nu e, deci ne uitam la w = 10 – 4 = 6(ruczac cu spatiu pentru el), o linie mai sus, adica 2,caci acolo mai e spatiu minim suficient pentru acest al 3 item, si il putem pune pe al 3 asa, sa vedem daca e mai bun decat ce s-a luat pana la acea valaore optima presupusa(de mai sus), si vedem ca e 30, acum 30 + 16 < 44 , deci va fi 46 in loc de 44.

**Reconstruirea solutiei**



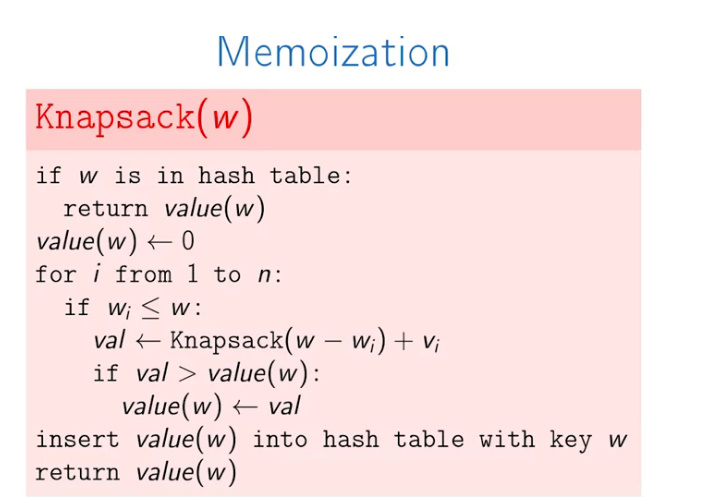
Facem un tabel boolean pentru a vedea daca itemul i e folosit sau nu

Solutia optima e mereu ultima casuta, deci 46

* Pentru a vedea daca itemul al 4 a fost folosit,ne uitam la ruczag de mai sus. si vedem ca tot e 46. Totusi, al 3 si al 4 item ar putea avea valori identice, de aceea scadem din 46 valaoreaa la al 4 item, deci 46 – 9 = 37, deci un ruczag cu primele 3 iteme, dar fara al 4,cu masa 10-2 = 8, ar trebui sa fie 37, dar vedem ca e 30. Deci ultimul item nu e folosit la sigur.
* Luam urmatoarea casuta de mai sus de 46, care tot are valoarea 46. Acum aceast nou 46 il comparam cu 44 de mai sus, si sunt diferite, deci vrem sa vedem daca al 3 item a fost folosit. Din 44 scadem al 3 item, adica 46-16 = 30. Acum, ne uitam la ruczagul 10-4 = 6, la coloana 6 linia 2, si e 30, deci al 3 item e folosit.
* Acum ne ducem la acel 30. Mai sus tot e 30. Vedem daca al 2 item e prezent, deci 30 – 14 = 16, si ne uitam la coloana 6 – 3 = 3, linia 1, si vedem ca e 0, nu 16, deci pass la al 2 item
* Acum ne uitam la 30 de mai sus

**Memoization**

* Daca chiar avem nevoie sa folosim algoritmi recursivi, trebuie cel putin sa evitam repetarea unor solutii deja existente.
* Memoization prevede sa cream un Hash Table sau Hash Map cu solutiile deja gasite, si inainte de a incepe sa aflam o solutie pentru o problema, verificam daca ea nu exista deja in acel tabel.



* Recursia e buna cand nu avem nevoie sa aflam solutiile pentru toate subproblemele.