Metoda Programării Dinamice

Problemele trebuie rezolvate folosind metoda programării dinamice.
Pentru fiecare problemă trebuie justificată relația de recurență obținută (pornind de la principiul de optimalitate, evidențiind subproblemele, relațiile de recurență și ordinea de calcul)

1. Subsecvența de sumă maximă a unui șir: Se dă un șir de numere (în fișier, separate prin spații). Să se afișeze o subsecvenţă de sumă maximă a șirului (formată cu elemente consecutive) O(n)

date.in	date.out
1 -2 3 -1 5 2 -6 1 3	3 -1 5 2

Indicație: Subproblema s[i] = subsecvența de sumă maxima care se termină pe poziția i
Recurența: s[i] = max{s[i-1]+v[i], v[i]} (unde v este șirul dat) – deoarece subsecvența care se termină pe poziția i este formată din elementul v[i] la care se adaugă eventual subsevența care se termină pe poziția i-1 (daca astfel se obține o suma mai mare)

2. Se consideră o tablă de șah nxm (n,m date). Pe fiecare careul al tablei este plasat câte un obiect, fiecare cu o anumită valoare (cunoscută, număr natural). Pe tablă se deplasează un robot astfel: pornește de pe prima linie și prima coloană (un colț al tablei) și se poate deplasa numai în direcțiile sud și est. La parcurgerea unei celule robotul adună obiectul din celulă. Să se determine un traseu al robotului până în poziția (n, m) (până în colțul opus celui din care a plecat) astfel încât valoarea totală a obiectelor adunate să fie maximă. Se vor afișa valoarea totală obţinută şi traseul optim **O(nm)**

date.in	date.out
3 3 **2 1** 4 1 **3** 2 1 **6 1**	13 1 1 1 2 2 2 3 2 3 3

3. Se consideră un șir de n cuburi colorate (n dat), pentru fiecare cub cunoscându-se lungimea laturii și culoarea sa, codificată cu un număr de la 1 la p (p dat). Să se determine un turn de înălțime maximă în care un cub nu poate fi așezat peste un cub de aceeași culoare sau cu latură mai mică sau egală cu a sa. Afișați și câte astfel de turnuri există. Structura fișierului de intrare: pe prima linie sunt n și p, pe următoarele linii lungimea laturii și culoarea câte unui cub. **O(n^2)**

date.in	date.out
7 3 8 3 10 2 9 2 10 1 8 1 5 2 6 2	10 1 9 2 8 1 6 2 2

4. Dat un șir de cuvinte formate cu litere mici, să se determine cel mai lung subșir al său astfel încât pentru orice două cuvinte consecutive din subșir ultimele două litere din primul să coincidă cu primele două litere din cel de al doilea. **Exemplu**: Pentru șirul: seara, carte, teorema, temperatura, rar, mare, arbore
cel mai lung subşir care verifică cerinţele este - carte, temperatura, rar, arbore **O(n^2) /O(n)**

date.in	date.out
masa carte sac teatru tema rustic sare	carte teatru rustic

5. Date o mulțime de n numere naturale și un număr natural M, M < 10000, să se determine, dacă există, o submulțime a mulțimii date de sumă M **O(nM)**

date.in	date.out
6 12 1 3 4 5 7 14	3 4 7

6. Moș Crăciun a poposit la bradul a doi frați, unde și-a golit sacul. Când s-au trezit, frații au intrat într-o mare dilemă: cum își vor împărți ei cadourile? Știind că fiecare cadou are o valoare cuprinsă între 1 și 100 și că sunt maxim 100 de cadouri, scrieți un program care să determine sumele cadourilor fraților precum și modul de împărțire, astfel încât sumele obținute să fie cele mai apropiate posibil. Exemplu: pentru 7 cadouri cu valorile 28, 7, 11, 8, 9, 7, 27 sumele sunt 48 și 49, o împărțire a cadourilor fiind 28, 11, 9, respectiv 7, 8, 7, 27. (Indicație: problema se reduce la a **determina o submulțime de sumă maximă, care nu depășește însă valoarea M**=jumătate din suma valorilor) **O(nS),** cu S=valoarea totală a cadourilor

7. **Generalizarea problemei spectacolelor (planificării activităților) discutată la curs la Greedy.** Se dau n activități prin timpul de început, timpul de sfârșit și profitul asociat desfășurării activității (n intervale închise cu extremităţi numere reale care au asociate ponderi). Să se determine o submulțime de activități compatibile (intervale disjuncte două câte două) care au profitul total maxim. Se vor afișa profitul total şi activităţile **$O(n^2)$/O(nlog n)**

   Jon Kleinberg, Éva Tardos, ***Algorithm Design****, Addison-Wesley 2005*

   https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/06DynamicProgrammingI.pdf

date.in	date.out
4 1 3 1 2 6 8 4 7 2 10 11 5	13 2 6 10 11

8. **(suplimentar) Generalizarea problemei *Maximizarea profitului cu respectarea termenelor limită* de la Greedy (*scheduling jobs with deadlines profits and durations*).** Acelaşi enunţ, dar pentru o activitate se cunoaşte în plus şi durata acesteia l_i (se renunţă la ipoteza toate activităţile au aceeaşi durată şi la faptul că $1 \le t_i \le n$) - **O(nT+nlog(n)),** unde T=max{ti|i=1,n}.

   Exemplu. Pentru n = 4 şi

   p1 = 3, t1 = 5, $l_1 = 3$

   p2 = 2, t2 = 2, $l_2 = 1$

   p3 = 3, t3 = 2, $l_3 = 2$

   p4 = 5, t4 = 4, $l_4 = 3$

   o soluţie optimă se obţine dacă planificăm activitățile în ordinea 2, 4, profitul fiind 7

   Observaţie: *Problema discretă a rucsacului poate fi privită ca un caz particular al acestei probleme (obiectele sunt activităţi de durată g_i, profit c_i şi termen limită G)*

date.in	date.out
4 3 5 3 2 2 1 3 2 2 5 4 3	7 2 4

9. **(Suplimentar) Generalizarea problemei joc de la Greedy** Pentru jocul descris în problema 10 de la Greedy, renunţând la ipoteza că numărul de elemente n este par, determinaţi dacă primul jucător are o strategie de câştig şi, în caz afirmativ, **cu cât va câştiga minim** (cu cât va fi mai mare sigur suma lui decât a adversarului). Implementaţi un joc de două persoane în care pentru primul jucător mută calculatorul conform strategiei optime determinate, iar pentru al doilea joacă utilizatorul. La fiecare pas anunţaţi-l pe utilizator cu cât sunteţi sigur că va fi mai mare suma obţinută de calculator faţă de a sa **$O(n^2)$** (vezi şi http://www.infoarena.ro/problema/joculet)