Metoda Greedy

Problemele trebuie însoțite de demonstrația corectitudinii și justiificarea complexității.

0. Implementați problemele discutate la curs

1. Cuburi. Se dau n cuburi cu laturile **diferite două câte două**. Fiecare cub are o culoare, codificată cu un număr de la 1 la p (p dat).

a) Să se construiască un turn de înălțime maximă de cuburi în care un cub nu poate fi aşezat pe un cub de aceeași culoare sau cu latură mai mică decât a sa – **O(n logn).** Pe prima linie a fişierului de intrare se dau n şi p, iar pe următoarele linii latura şi culoarea fiecărui cub. În fişierul de ieşire se vor afişa înălțimea turnului obținut şi indicele cuburilor folosite (de la bază la vârf)

date.in	date.out
4 2 5 1 10 1 9 1 8 2	23 2 4 1

b) În cazul în care lungimile laturilor cuburilor nu erau diferite mai este valabilă metoda propusă? Justificaţi.

2. http://varena.ro/problema/acoperire

3. Problema partiţionării intervalelor – Se consideră n intervale închise (interpretare: n cursuri, pentru care se cunosc ora de început şi ora de sfârşit). Se cere să se împartă (partiţioneze) această mulţime de intervale într-un **număr minim de submulţimi** cu proprietatea că oricare două intervale dintr-o submulţime nu se intersectează şi să se afişeze aceste submulţimi (interpretare: să se determine numărul minim de săli necesare pentru a putea programa aceste cursuri în aceeaşi zi şi afişaţi o astfel de programare). Care dintre următorii algoritmi Greedy sunt corecţi pentru a rezolva această problemă? Pentru fiecare algoritm (subpunct) justificaţi corectitudinea sau daţi un contraexemplu:

a) Se sortează intervalele crescător după extremitatea iniţială. Pentru fiecare interval I în această ordine execută: se adaugă I la o submulţime deja construită, dacă se poate (nu se intersectează cu niciun interval din ea), altfel se creează o nouă submulţime cu intervalul I.

b) Se sortează intervalele crescător după extremitatea finală. Pentru fiecare interval I în această ordine execută: se adaugă I la o submulţime deja construită, dacă se poate, altfel se creează o nouă submulţime cu intervalul I. Dacă există mai multe mulţimi la care se poate adăuga I, se alege una arbitrar.

c) Se sortează intervalele crescător după extremitatea finală Pentru fiecare interval I în această ordine execută: se adaugă I la o mulţime deja construită, dacă se poate, altfel se creează o nouă mulţime cu intervalul I. Dacă există mai multe mulţimi la care se poate adăuga I, se alege acea submulţime care conţine intervalul cu extremitatea finală cea mai mare (care se termină cât mai aproape de începutul intervalului I)

d) Cât timp mai sunt intervale nedistribuite în submulţimi repetă: construieşte o submulţime de cardinal maxim de intervale disjuncte din mulţimea de intervale (folosind algoritmul de la problema spectacolelor discutată la curs) şi se elimină aceste intervale din mulţimea de intervale (altfel spus, se construieşte prima submulţime folosind un număr maxim de intervale, apoi a doua submulţime folosind un număr maxim de intervale din cele rămase etc).

e) Implementaţi un algoritm **O(n log n)** pentru rezolvarea acestei probleme.

Exemplu: pentru n=3 cursuri, care trebuie să se desfăşoare în intervalele: [10, 14], [12, 16], respectiv [17, 18], sunt necesare 2 săli, o programare optimă fiind:

- **Sala 1**: [10, 14] – activitatea 1, [17, 18] – activitatea 3
- **Sala 2**: [12, 16] – activitatea 2

date.in	**date.out**
3 10 14 12 16 17 18	2 1 3 2

4. Memorarea textelor pe bandă cu frecvență de acces (v. problema textelor pe banda din curs)

n texte cu lungimile L(1),...,L(n) urmează a fi aşezate pe o bandă. Pentru a citi textul de pe poziţia k, trebuie citite textele de pe poziţiile 1,2,...,k. Pentru fiecare text i se cunoaşte şi frecvenţa f(i) cu care acesta este citit. Să se determine o modalitate de aşezare a textelor pe bandă astfel încât timpul total de acces să fie minimizat. Timpul total de acces pentru o aşezare a textelor corespunzătoare unei permutări p se defineşte ca fiind $\sum_{i=1}^{n} f(p_i)[L(p_1)+...+L(p_i)]$ **O(n log n)** Pe prima linie a fişierului de intrare se dă n, iar pe următoarele linii lungimea și frecvența fiecărui text.

date.in	**date.out**
3 5 25 1 25 5 50	2 3 1

Problemă echivalentă - programarea taskurilor cu priorități diferite*. n activități cu duratele L(1),...,L(n) urmează a fi programate pentru executare (secvențială) folosind o aceeași resursă. Fiecare activitate i are asociată o prioritate w(i). O ordine în care se execută aceste activități se poate reprezenta printr-o permutare* $p \in S_n$. *În raport cu această ordine, a i-a activitate executată are timpul de terminare* $L(p_1)+...+L(p_i)$. *Să se determine o ordine în care trebuie planificate activitățile astfel încât să fie minimizată suma ponderată a timpilor de finalizare a acțiunilor, dată de formula* $\sum_{i=1}^{n} w(p_i)[L(p_1)+...+L(p_i)]$

5. Planificare cu minimizarea întârzierii maxime – Se consideră o mulţime de n activităţi care trebuie planificate pentru a folosi o aceeaşi resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Pentru fiecare activitate i se cunosc durata l_i şi termenul limită până la care se poate executa t_i (raportat la ora de început 0). Dorim să planificăm aceste activităţi astfel încât întârzierea fiecărei activităţi să fie cât mai mică. Mai exact, pentru o planificare a acestor activităţi astfel încât activitatea *i* este programată în intervalul de timp [s_i, f_i), definim întârzierea activităţii *i* ca fiind durata cu care a depăşit termenul limită: $\boldsymbol{p_i = \max\{0, f_i - t_i\}}$.

Întârzierea planificării se defineşte ca fiind **maximul întârzierilor activităţilor**:

Exemplu. Pentru n = 3 şi $l_1 = 1$, $t_1 = 3$ / $l_2 = 2$, $t_2 = 2$ / $l_3 = 3$, $t_3 = 3$

o soluţie optimă se obţine dacă planificăm activităţile în ordinea 2, 3, 1; astfel:

- activitatea 2 în intervalul [0, 2) – întârziere 0
- activitatea 3 în intervalul [2, 5) – întârziere 5 – t_3 = 5 - 3 = 2
- activitatea 1 în intervalul [5, 6) – întârziere 6 – t_1 = 6 - 3 = 3

Întârzierea planificării este max{0, 2, 3}=3 $\boldsymbol{P = \max\{p_1, p_2, ..., p_n\}}$

a) Să se determine o planificare a activităţilor date care să aibă întârzierea P minimă. Se vor afişa pentru fiecare activitate intervalul de desfăşurare şi întârzierea – **O(n log n)**

date.in	**date.out**
3 1 3 2 2 3 3	activitatea 2: intervalul 0 2 intarziere 0 activitatea 3: intervalul 2 5 intarziere 2 activitatea 1: intervalul 5 6 intarziere 3 Intarziere planificare 3

b) Este corect un algoritm Greedy bazat pe următoarea idee: planificăm activităţile în ordine crescătoare în raport l_i (adică în raport cu durata)? Justificaţi.
c) Este corect un algoritm Greedy bazat pe următoarea idee: planificăm activităţile în ordine crescătoare în raport cu diferenţa t_i-l_i (adică în raport cu timpul maxim la care trebuie să înceapă activitatea i pentru a respecta termenul limită)? Justificaţi.

6. Maximizarea profitului cu respectarea termenelor limită Se consideră o mulţime de n activităţi care trebuie planificate pentru a folosi o aceeaşi resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Toate activităţile au aceeaşi durată (să presupunem 1). Pentru fiecare activitate i se cunosc termenul limită până la care se poate executa t_i (raportat la ora de început 0, $1 \leq t_i \leq n$) şi profitul p_i care se primeşte dacă activitatea i se execută la timp (cu respectarea termenului limită). Să se determine o submulţime de activităţi care se pot planifica astfel încât profitul total obţinut să fie maxim.
Exemplu. Pentru n = 4 şi
$p_1 = 4,\ t_1 = 3$
$p_2 = 1,\ t_2 = 1$
$p_3 = 2,\ t_3 = 1$
$p_4 = 5,\ t_4 = 3$
o soluţie optimă se obţine dacă planificăm activităţile în ordinea 3, 4, 1, profitul obţinut fiind $p_3 + p_4 + p_1 = 2 + 5 + 4 = 11$. **O(n logn) (4p)**

date.in	**date.out**
4 4 3 1 1 2 1 5 3	11 3 4 1

7. Interclasarea optimă a n șiruri ordonate – Se dau lungimile a n șiruri ordonate $L_1, L_2, \ldots, L_n$. Dorim să obținem un șir ordonat crescător care conține toate elementele celor n șiruri inițiale, interclasând succesiv perechi de șiruri. Știind că interclasarea a două șiruri de lungimi A respectiv B necesită A+B deplasări, să se determine o ordine în care trebuie să se realizeze interclasările astfel încât numărul total de deplasări să fie minim – **O(nlogn)**

Exemplu: Pentru șirurile de lungimi $L_1 = 20$, $L_2 = 30$, $L_3 = 20$, $L_4 = 35$

Pentru ordinea de interclasare:

(sirul 1, sirul 2) => sirul 5 de lungime $L_5 = L_1 + L_2 = 50$ (50 de deplasari)

(sirul 5, sirul 3) => sirul 6 de lungime $L_6 = L_5 + L_3 = 70$ (70 de deplasari)

(sirul 6, sirul 4) => sirul 7 de lungime $L_7 = L_6 + L_4 = 105$ (105 de deplasari)

numărul total de deplasări va fi 50+70+105 = 225

(=strategia interclaseaza(interclaseaza(interclaseaza(1,2), 3), 4))

O ordine de interclasare cu număr minim de deplasări ar fi:

(sirul 1, sirul 3) => sirul 5 de lungime $L_5 = L_1 + L_3 = 20+20=40$ (40 de deplasări)

(sirul 2, sirul 4) => sirul 6 de lungime $L_6 = L_2 + L_4 = 30+35=65$ (65 de deplasări)

(sirul 5, sirul 6) => sirul 7 de lungime $L_7 = 105$ cu $L_5 + L_6 = 40+65 = 105$ (105 deplasari)

Numărul total de deplasări va fi: 40 + 65 + 105= 210

(=strategia interclaseaza(interclaseaza(1,3), interclaseaza(2,4)))

8. Considerăm următorul **joc pentru două persoane**. Tabla de joc este o secvenţă de n numere întregi pozitive, iar cei doi jucători mută alternativ. Când un jucător mută, el selectează un număr ori de la stânga ori de la dreapta secvenţei. Numărul selectat este şters de pe tablă. Jocul se termină când toate numerele au fost selectate. Primul jucător câştigă dacă suma numerelor pe care le-a selectat este **cel puţin egală** cu suma selectată de al doilea jucător. Al doilea jucător joacă cât de bine poate. Primul jucător începe jocul. Ştim că tablă se află la început un număr **par** de elemente n.

a) Să se scrie un program astfel încât, indiferent cum va juca al doilea jucător, primul jucător câştigă. Scrieţi programul astfel încât primul jucător să mute cu ajutorul programului, iar calculatorul să mute aleator de la stânga sau de la dreapta. La ieşire se va scrie suma obţinută de primul jucător, suma obţinută de cea de al doilea şi secvenţele de mutări sub forma unor şiruri cu caracterele S pentru stânga şi D pentru dreapta. **Exemplu**: pentru tabla cu numerele 2 1 4 3 o soluţie câştigătoare este următoarea:

Pasul 1 - primul jucător alege S (valoarea 2); rămân pe tablă 1 4 3

Pasul 2 - calculatorul are două posibilităţi: S (valoarea 1) sau D (valoarea 3).

Pasul 3 – dacă la pasul 2 calculatorul a ales S, atunci primul jucător alege S (valoarea 4);
dacă la pasul 2 calculatorul a ales D, atunci primul jucător alege D (valoarea 4);

Pasul 4 – pe tablă a mai rămas doar o valoare (3 respectiv 1, în funcţie de alegerea de la pasul 2), pe care o alege calculatorul

Astfel, primul jucător a adunat suma 2+4, iar calculatorul suma 1+3 (respectiv 3+1), deci primul jucător a câştigat

b) În cazul în care urma strategiei implementate la a) pentru o secvenţă de numere primul jucător obţine o sumă egală cu cel de al doilea, este posibil ca prin altă strategie totuşi primul jucător să câştige cu o sumă strict mai mare pentru aceeaşi secvenţă? Justificaţi