

Varianta 1

Toate problemele trebuie însoțite de demonstrația scrisă a corectitudinii și complexității

Pentru testul de laborator și examen trebuie știute problemele din toate cele 3 variante

1. **Acoperire** - Se dau un interval închis $[a,b]$ și o mulțime de alte n intervale închise $[a_i, b_i]$, $1 \leq i \leq N$. Scrieți un program care determină și afișează o submulțime de cardinal minim de intervale închise din mulțimea dată cu proprietatea că prin reuniunea acestora se obține un interval care include pe $[a,b]$. Dacă prin reuniunea tuturor intervalelor nu putem obține un interval care să includă intervalul $[a,b]$, se va afișa -1 **$O(n \log n)$** <http://varena.ro/problema/acoperire> (2p)

date.in	date.out
10 20	9 15
5	14 21
9 15	
4 16	
5 12	
14 21	
11 18	

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

Întârzierea planificării se definește ca fiind **maximul întârzierilor activităților**:

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

o soluție optimă se obține dacă planificăm activitățile 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 $\max\{0, 2, 3\} = 3$ **$P = \max\{p_1, p_2, \dots, p_n\}$**

a) Să se determine o planificare a activităților date care să aibă întârzierea P minimă. Se vor afișa pentru fiecare activitate intervalul de desfășurare și întârzierea – **$O(n \log n)$**

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

b) Este corect un algoritm Greedy bazat pe următoarea idee: planificăm activitățile în ordine crescătoare în raport l_i (adică în raport cu durata)? Justificați.

c) Este corect un algoritm Greedy bazat pe următoarea idee: planificăm activitățile în ordine crescătoare în raport cu diferența $t_i - l_i$ (adică în raport cu timpul maxim la care trebuie să înceapă activitatea i pentru a respecta termenul limită)? Justificați. (2,5p)

3. **Descompunere în subșiruri descrescătoare**: Se dă un șir de numere naturale. Să se descompună șirul de numere într-un număr minim de subșiruri descrescătoare (nestrict) și să se afișeze aceste subșiruri.

- a) Implementați un algoritm – $O(n \log n)$ pentru rezolvarea acestei probleme. (Indicație **pentru demonstrarea corectitudinii**: Numărul optim de subșiruri este egal cu lungimea maximă a unui subșir crescător al șirului inițial).

date.in	date.out
6 11 13 10 15 12 7	11 10 7 13 12 15

- b) Este corect următorul algoritm Greedy pentru rezolvarea acestei probleme? Justificați
Cât timp mai sunt elemente în șir repetă:
- construim un subșir descrescător care începe cu primul element din șir, adăugând pe rând la subșir următorul element din șir care se poate adăuga (mai mic sau egal cu ultimul element adăugat la subșir)
- afișează acest subșir și elimină-l din șirul inițial
- c) Implementați Patience Sorting – o metodă de sortare bazată pe împărțirea unui șir în subșiruri descrescătoare, memorate ca stive (după această împărțire se repetă de n ori următoarea operație: mută cea mai mică valoare din topul unei stive în vectorul rezultat, inițial vid) - $O(n \log n)$. Pentru mai multe detalii și imagini sugestive pentru înțelegerea problemei, dar și legătura cu problema determinării celui mai lung subșir crescător al unui șir – pe care o vom studia la programare dinamică – se pot consulta spre exemplu:

https://en.wikipedia.org/wiki/Patience_sorting

<https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/LongestIncreasingSubsequence.pdf> (4,5p)

Variantă 2

Toate problemele trebuie însoțite de demonstrația scrisă a corectitudinii și complexității

Pentru testul de laborator și examen trebuie știute problemele din toate cele 3 variante

1. 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 așezat pe un cub de aceeași culoare sau cu latură mai mică decât a sa – **$O(n \log n)$** . Pe prima linie a fișierului de intrare se dau n și p , iar pe următoarele linii latura și culoarea fiecărui cub. În fișierul de ieșire se vor afișa înălțimea turnului obținut și indicele cuburilor folosite (de la bază la vârf)

date.in	date.out
4 2	2 3
5 1	2 4 1
10 1	
9 1	
8 2	

b) În cazul în care lungimile laturilor cuburilor nu erau diferite mai este valabilă metoda propusă? Justificați. **(2p)**

2. Memorarea textelor pe banda – variante

a) n texte cu lungimile $L(1), \dots, L(n)$ urmează a fi așezate pe o bandă. Pentru a citi textul de pe poziția k , trebuie citite textele de pe pozițiile $1, 2, \dots, k$. Pentru fiecare text i se cunoaște și frecvența $f(i)$ cu care acesta este citit. Să se determine o modalitate de așezare a textelor pe bandă astfel încât timpul total de acces să fie minimizat. Timpul total de acces pentru o așezare a textelor corespunzătoare unei

permutări p se definește ca fiind $\sum_{i=1}^n f(p_i)[L(p_1) + \dots + L(p_i)]$ **$O(n \log n)$** Pe prima linie a fișierului de intrare se dă n , iar pe următoarele linii lungimea și frecvența fiecărui text.

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

***Problemă echivalentă - programarea taskurilor cu priorități diferite.** n activități cu duratele $L(1), \dots, L(n)$ 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 $L(p_1) + \dots + L(p_i)$. 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*

$$\sum_{i=1}^n w(p_i)[L(p_1) + \dots + L(p_i)]$$

b) n texte cu lungimile $L(1), \dots, L(n)$ urmează a fi așezate pe p benzi (p dat). Pentru a citi textul de pe poziția k de pe o bandă, trebuie citite textele de pe pozițiile $1, 2, \dots, k$ de pe aceea bandă. Fiecare text are aceeași frecvență de citire. Să se determine o modalitate de așezare a textelor pe cele p benzi astfel încât timpul total de acces să fie minimizat (se obține adunând timpul total de acces pentru fiecare din cele p benzi). **$O(n \log n)$ (3p)**

3. Maximizarea profitului cu respectarea termenelor limită Se consideră o mulțime de n activități care trebuie planificate pentru a folosi o aceeași resursă. Această resursă poate fi folosită de o singură activitate la un moment dat. Toate activitățile au aceeași 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, $1 \leq t_i \leq n$) și profitul p_i care se primește dacă activitatea i se execută la timp (cu respectarea termenului

limită). Să se determine o submulțime de activități care se pot planifica astfel încât profitul total obținut să fie maxim.

Exemplu. Pentru $n = 4$ și

$$p_1 = 4, t_1 = 3$$

$$p_2 = 1, t_2 = 1$$

$$p_3 = 2, t_3 = 1$$

$$p_4 = 5, t_4 = 3$$

o soluție optimă se obține dacă planificăm activitățile în ordinea 3, 4, 1, profitul obținut fiind

$$p_3 + p_4 + p_1 = 2 + 5 + 4 = 11. \mathbf{O(n \log n)} \quad (4p)$$

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

Varianta 3

Toate problemele trebuie însoțite de demonstrația scrisă a corectitudinii și complexității

Pentru testul de laborator și examen trebuie știute problemele din toate cele 3 variante

1. Considerăm următorul **joc pentru două persoane**. Tabla de joc este o secvență 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 secvenței. Numărul selectat este șters de pe tablă. Jocul se termină când toate numerele au fost selectate. Primul jucător câștigă dacă suma numerelor pe care le-a selectat este **cel puțin egală** cu suma selectată de al doilea jucător. Al doilea jucător joacă cât de bine poate. Primul jucător începe jocul. Știm 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 câștigă. Scrieți programul astfel încât primul jucător să mute cu ajutorul programului, iar calculatorul să mute aleator de la stânga sau de la dreapta. La ieșire se va scrie suma obținută de primul jucător, suma obținută de cea de al doilea și secvențele de mutări sub forma unor șiruri cu caracterele S pentru stânga și D pentru dreapta. **Exemplu:** pentru tabla cu numerele 2 1 4 3 o soluție câștigătoare este următoarea:

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

Pasul 2 - calculatorul are două posibilități: 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 funcție 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 câștigat - **O(n)**

b) În cazul în care urma strategiei implementate la a) pentru o secvență de numere primul jucător obține o sumă egală cu cel de al doilea, este posibil ca prin altă strategie totuși primul jucător să câștige cu o sumă strict mai mare pentru aceeași secvență? Justificați (**1,5p**)

2. Dat un arbore cu n vârfuri, să se determine o mulțime de vârfuri neadiacente de cardinal maxim (o submulțime independentă maximă a mulțimii vârfurilor). **Exemplu:** Pentru un arbore cu 8 vârfuri și muchiile $\{(1,2),(1,3),(2,4),(2,5),(3,6),(3,7),(5,8)\}$ se va afișa mulțimea $\{1,4,6,7,8\}$, nu neapărat în această ordine. Pentru un graf neorientat oarecare mai este corectă o metodă de tip Greedy similară? Justificați. **O(n) (3p)**

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

3. **Problema partiționării intervalelor** – Se consideră n intervale închise (interpretare: n cursuri, pentru care se cunosc ora de început și ora de sfârșit). Se cere să se împartă (partiționeze) această mulțime de intervale într-un **număr minim de submulțimi** cu proprietatea că oricare două intervale dintr-o submulțime nu se intersectează și să se afișeze aceste submulțimi (interpretare: să se determine numărul minim de săli necesare pentru a putea programa aceste cursuri în aceeași zi și afișați o astfel de programare). Care dintre următorii algoritmi Greedy sunt corecți pentru a rezolva această problemă? Pentru fiecare algoritm (subpunct) justificați corectitudinea sau dați un contraexemplu:

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

- b) Se sortează intervalele crescător după extremitatea finală.. Pentru fiecare interval I în această ordine execută: se adaugă I la o submulțime deja construită, dacă se poate, altfel se creează o nouă submulțime cu intervalul I. Dacă există mai multe mulțimi 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 mulțime deja construită, dacă se poate, altfel se creează o nouă mulțime cu intervalul I. Dacă există mai multe mulțimi la care se poate adăuga I, se alege cea submulțime care conține 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 submulțimi repetă: construiește o submulțime de cardinal maxim de intervale disjuncte din mulțimea de intervale (folosind algoritmul de la problema spectacolelor discutată la curs) și se elimină aceste intervale din mulțimea de intervale (altfel spus, se construiește prima submulțime folosind un număr maxim de intervale, apoi a doua submulțime folosind un număr maxim de intervale din cele rămase etc).
- e) Implementați un algoritm **$O(n \log n)$** pentru rezolvarea acestei probleme.

Exemplu: pentru $n=3$ cursuri, care trebuie să se desfășoare 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 **(4,5p)**

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