Clasele XI-XII
Soluţiile problemelor din a doua zi

proc

Testele au fost structurate astfel încât diverse abordări ale problemei să obţină un număr diferit de puncte, astfel:

- **20%** din punctaj, pentru o soluţie cu complexitate exponenţială
- **40%** din punctaj pentru o soluţie greedy cu complexitatea **O(N*P)**
- **70%** din punctaj pentru o soluţie greedy cu complexitatea **O(N*logP)**
- **100%** din punctaj pentru o soluţie cu complexitatea **O(P*logN)**

Soluţia exponenţială

Există mai multe posibilităţi de a implementa o soluţie care foloseste backtracking. Se poate varia penru fiecare fragment de cod procesorul pe care va fi executat – **$O(N^P)$** – sau se poate stabili pentru fiecare procesor câte fragmente de cod de tipul **1**, respectiv **2** vor fi executate pe el. O optimizare a celei de-a doua variante ar fi că timpul după care un procesor **i** devine disponibil, notat prin **T_i**, trebuie să respecte relatia **$T_i/i <= T_j/j$** , unde **j** este numărul unui procesor mai rapid decât o (adică **j<i**).

Soluţia greedy cu complexitate O(N*P)

Se iniţializează timpii după care fiecare procesor redevine disponibil cu **0**. Se stabileşte procesorul pe care va fi executat fiecare fragment de cod cu timpul **2**. Apoi se stabileşte acelaşi lucru (adică procesorul pe care va fi executat) pentru fiecare fragment de cod cu timpul **1**. Pentru a stabili procesorul pe care va fi executat un fragment de cod, se încearcă executarea sa pe orice procesor de la **1** la **P** şi se alege acel procesor pentru care se găseşte minimul dintre timpii după care procesoarele redevin disponibile. Altfel spus, să presupunem că până la adăugarea fragmentului de cod curent, timpul de revenire al fiecarui procesor **i** este **T_i** şi să presupunem că fragmentul de cod actual are timpul de execuţie **Q** (**Q** este **1** sau **2**). Atunci se alege procesorul **i**, pentru care **T_i+Q*i** este minim.

Soluţia greedy cu complexitate O(N*logP)

Ideea acestei solutii este la fel ca cea descrisă mai sus, numai că determinarea procesorului pentru care se obţine timpul de revenire minim se face logaritmic, folosind o structura de heap.

Soluţia (comisiei) cu complexitatea O(P*logN)

Se observă că timpul minim după care se va termina aplicaţia este între **0** si **2*N** (în cazul când toate fragmentele de cod au timpul **2** şi sunt executate pe primul procesor). Se va folosi o căutare binară pentru determinarea acestui timp. La un moment dat, în cadrul căutării, se va testa dacă timpul minim este mai mic sau egal decât timpul **T**. Dacă da, atunci se încearcă un **T** mai mic. Altfel, se măreşte **T**-ul. Funcţia care decide dacă timpul minim este mai mic sau egal cu **T** are complexitatea **O(P)**. Se observă că dacă timpul minim este cel mult **T**, atunci suma timpilor elementari pe fiecare procesor **k** trebuie să fie cel mult **[T/k]**. Se calculează $\mathbf{N_2}=\sum_{k=1}^{P}\left\lfloor\frac{\left\lfloor\frac{T}{k}\right\rfloor}{2}\right\rfloor$, având semnificaţia: numărul maxim de fragmente de cod cu timpul **2** ce pot fi executate dacă timpul minim este cel mult **T**. Dacă $\mathbf{N_2}$ este mai mare sau egal cu numărul de fragmente de cod cu timpul **2** existente (să notăm acest numar cu **DOI**), se verifică dacă se pot executa destule fragmente de cod cu timpul **1**. Pentru aceasta se calculeaza $\mathbf{N_1}=\sum_{k=1}^{P}\left(\left\lfloor\frac{T}{k}\right\rfloor \bmod 2\right)$. Dacă **$N_1$+2*($N_2$-DOI)** este mai mare sau egal cu numărul de fragmente de cod cu timpul **1**, atunci timpul minim dupa care se termină aplicaţia este mai mic sau egal cu **T**. Altfel, nu.

inter

Sunt N*(N-1)/2 puncte de intersecţie. Dar din acestea maxim 2*N ne intereseaza (extremităţile de pe drepte), dar de fapt sunt maxim N. Pentru aflarea acestor puncte, care pot fi pe înfăşurătoarea convexă a celor N*(N-1)/2 puncte, se sortează dreptele dupa pantă. Punctele de interes sunt intersecţia a doua drepte consecutive în sortare, şi în plus prima cu ultima.
Pe aceste N puncte se face infăşurătoarea convexa.
Se calculează aria zonei rezultate...

Complexitate :
1 – sortare dupa pantă – O(NlogN)
2 – calcularea punctelor candidate de a fi pe înfăşurătoare – O(N)
3 – înfăşurătoarea convexă – O(NlogN)
4 – calcularea ariei – O(N)
În total: O(NlogN)

nr

1. Determinam numărul de apariţii pentru fiecare cifră (0..9) în numărul dat (x). Acest număr trebuie să fie par (cu excepţia eventuală a unei singure cifre; în caz că o astfel de cifră există trebuie să fie plasată la mijlocul palindromului), altfel nu există soluţie.
2. Determinăm numărul de apariţii ale fiecărei cifre (0..9) în prima jumătate a numărului x, precum şi numărul de apariţii necesar (pentru nr cerut). Dacă pentru o cifră numărul de apariţii existent în prima jumătate a numărului dat este prea mare, elimin apariţiile suplimentare, începând de la dreapta spre stânga.
3. Încerc să distribui cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să obţin cel mai mic număr posibil >=x (cu o eventuală rearanjare a cifrelor existente) Distribuirea cifrelor de face căutând de la stânga la dreapta să completăm poziţiile cu cifre egale cu cele din numărul dat. Când nu mai este posibil :

- am obtinut o jumatate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate, determin anagrama imediat următoare din pdv lexicografic lui x – dacă există;
- am gasit o cea mai mică cifră mai mare decât cea de pe poziţia curentă, o plasez, ordonez crescător cifrele de după => cea mai mica jumătate >x.