TABĂRA DE PREGĂTIRE ȘI SELECȚIE A LOTULUI NAȚIONAL DE INFORMATICĂ
MAI 2023, SLATINA

DESCRIEREA SOLUȚIILOR, BARAJ 1, JUNIORI

Ziua 1 — Problema 1: Pătrate cu cifre permutate (PCP)

Propusă de: prof. Chește Ciprian, Liceul Tehnologic "Grigore C. Moisil", Buzău

Brute force. O primă abordare, nu tocmai eficientă, ar putea consta în construirea tuturor permutărilor cifrelor numărului N și verificarea proprietății ca numărul obținut să fie pătrat perfect. Această abordare presupune utilizarea unui mecanism de tip backtracking sau un algoritm echivalent pentru generarea tuturor permutărilor și care poate fi optimizat prin vectori de frecvență a cifrelor.

Soluție eficientă. O soluție eficientă constă în generarea tuturor pătratelor perfecte mai mici decât N. Pentru fiecare pătrat perfect generat se va verifica dacă cifrele sale coincid cu cifrele numărului N cu excepția cifrelor de 0. Având în vedere că se cere și numărul acestora, numerele cu proprietatea cerută trebuie memorate într-o structură de date adecvată(vector) și afișate ulterior fără a fi sortate deoarece au fost generate în ordine crescătoare.
Ordinul de complexitate al soluției este $O(\sqrt{N} * log(N))$.

Soluție eficientă optimizată - prof. Adrian Panaete, student Gabriel Theodor Tulbă-Lecu. Se pot realiza următoarele optimizări:

- Se va construi un vector de frecvențe, pe care-l vom nota cu FR pentru cifrele nenule disponibile generând succesiv toate pătratele $\leq VALMAX$ și verificăm la fiecare pas dacă vectorul de frecvențe al pătratului la care am ajuns coincide cu vectorul FR, caz în care am descoperit o soluție.
- În locul efectuării înmulțirilor, pentru a genera pătratele perfecte se pot utiliza adunări folosind ideea ca $n^2 = 1 + 3 + 5 + ... + (2n - 1)$
- Deoarece frecvența unei cifre este cel mult 14, putem simula vectorul de frecvențe utilizând un număr pe 64 de biți.
  Astfel fiecărei cifre i se alocă câte 4 biți:
  pentru cifra 1 - primii 4 biți,
  pentru cifra 2 - următorii 4 biți,
  ...
  pentru cifra 9 - ultimii 4 biți.
  În total 36 de biți.
  Incrementarea frecvenței cifrei 1 se va realiza prin adunare cu 16^0,
  Incrementarea frecvenței cifrei 2 se va realiza prin adunare cu 16^1,
  ....
  Incrementarea frecvenței cifrei 9 se va realiza prin adunare cu 16^8,
  Decrementarea frecvenței cifrelor se va face similar (frecvența cifrei C se realizează scăzând din numărul pe 64 de biți valoarea 16^{C-1}
- Cât timp verificăm un pătrat generat cu valoarea mai mică decât $VALMAX$ ($\leq$ numărul maxim ce se poate forma cu cifrele disponibile), frecvența cifrei 0 nu ne interesează.
- Vectorul de frecvente al unei valori $(k + 1)^2$ poate fi "recalculat" din vectorul de frecvențe al lui k^2 astfel: Parcurgem succesiv și simultan cifrele celor două pătrate începând de la cifra unităților și continuând cu cifra zecilor, a sutelor, etc..
  La fiecare ordin scădem frecvența cifrei la care am ajuns în k^2 și o creștem pe cea de la $(k + 1)^2$ și eliminăm cifrele respective. În acest mod, în momentul în care cele

două valori obținute coincid am realizat transformarea vectorului de frecvențe deoarece cifrele rămase neprocesate din cele două valori k^2 și $(k+1)^2$ coincid.

Soluție de 100 puncte - prof. Mihai Bunget. Se determină cifrele numărului N și se află cel mai mic număr (A) și cel mai mare număr (B) care se pot forma cu cifrele lui N. Se parcurg valorile de la $[\sqrt{A}]$ la $[\sqrt{B}]$, iar pentru fiecare valoare x se determină cifrele lui $x \cdot x$ și, folosind vectori de frecvență, se verifică dacă acestea coincid cu cifrele lui N. Optimizarea se produce dacă observăm că în funcție de restul împărțirii lui N la 9 putem alege să verificăm numai anumite valori ale lui x. Astfel, dacă restul R al împărțirii lui N la 9 este 0 trebuie să parcurgem doar numerele x divizibile cu 3, dacă R este 1 parcurgem numerele x ce dau restul 1 sau 8 la împărțirea cu 9, dacă R este 4 parcurgem numerele x ce dau restul 2 sau 7 la împărțirea cu 9, iar dacă R este 7 parcurgem numerele x ce dau restul 4 sau 5 la împărțirea cu 9. Aceasta se explică prin faptul că restul împărțirii unui pătrat perfect la 9 poate fi doar 0,1,4 sau 7, iar acest pătrat perfect trebuie să aibă aceleași cifre cu N deci ambele vor da același rest la împărțirea cu 9. Matematic restul împărtirii unui număr la 9 este egal cu restul împărțirii sumei cifrelor sale la 9. Pentru afișarea soluțiilor în ordine crescătoare se sortează vectorul soluțiilor.

Restul împărțirii unui pătrat perfect la 9 poate fi doar 0,1,4 sau 7. Deci dacă N nu dă unul din aceste resturi la împărțirea cu 9 se va afișa nu există. Altfel, se va verifica dacă un pătrat perfect are aceleași cifre cu N dar și același rest la împărțirea cu 9.

Ziua 1 — Problema 2: Excursie

Propusă de: stud. Andrei Onuț, Universitatea Yale, S.U.A.,
prof. Daniela Lica, Centrul Județean de Excelență Prahova

Observație inițială: Fără a pierde din generalitatea rezolvării, considerăm că $x < y$, pentru fiecare dintre cele Q excursii. Fie k satul pentru care se va obține un cost total minim (în cadrul unei excursii); pot exista mai multe astfel de sate k, însă vom considera doar unul dintre ele, din moment ce toate vor rezulta în același cost total.

Observăm că: $x \leq k \leq y$ (orice sat k, cu $k < x$, va genera un cost mai mare sau egal decât satul x, de vreme ce, de exemplu, John ar putea rămâne pe întreaga durată a traseului în satul x; similar raționăm și pentru orice sat k, cu $k > y$.).

Idee: Pentru fiecare excursie ($x < y$), considerăm fiecare punct posibil de întâlnire k și calculăm costul pentru a ajunge în punctul k din x și y. Pentru a ajunge dintr-o poziție x în poziția k, unde $x < k$, trebuie ca toate caracterele din secvența $x...k-1$ să fie **R**. Pentru a ajunge dintr-o poziție y în poziția k, unde $y > k$, trebuie ca toate caracterele din secvența $k+1...y$ să fie **L**.

Vom precalcula următorii vectori:

- $Pref[i]$- numărul de caractere **L** din secvența $1...i$
- $Suff[i]$- numărul de caractere **R** din secvența $i...N$

Costul pentru ca o excursie $x < y$ să se întâlnească în punctul k ($x \leq k \leq y$) este:

$$cost(k) = Pref[k-1] - Pref[x-1] + Suff[k+1] - Suff[y+1]$$

$$cost(k) = (Pref[k-1] + Suff[k+1]) - (Pref[x-1] + Suff[y+1])$$

Pentru claritate, notăm cu $Value(k) = Pref[k-1] + Suff[k+1]$. Deci, $cost(k) = Value(k) - (Pref[x-1] + Suff[y+1])$.

Facem următoarele 2 observații:

(1) $Value(k)$ nu depinde de valorile lui x și y.
(2) $(Pref[x-1] + Suff[y+1])$ nu depinde de valoarea lui k.

Mai concret, $cost(k)$ este diferența dintre o valoare prestabilită ($Value$ nu își modifică valoarea da la o excursie la alta) și o constantă. Cum $Value$ diferă în funcție de k, răspunsul nostru este dat de valoarea minimă a lui $Value(k)$.

Astfel, problema noastră s-a redus la a afla k între x și y, pentru care $Value(k)$ este *minim*.

Există mai multe abordări pentru a afla minimul dintr-un interval.

Pentru determinarea optimă a minimului pe un interval, vom construi o matrice RMQ (Range Minimum Query), care reține minimul pe intervale de lungimi puteri de 2, după care, aflarea minimului pe orice interval (x, y) se obține în $O(1)$.

O altă variantă de calcul a RMQ, **propusă de Cătălin Frâncu**, răspunde celor Q interogări în timp total $\mathcal{O}(Q \log N)$, folosind memorie suplimentară doar $\mathcal{O}(N)$ în loc de $\mathcal{O}(N \log N)$. Procedăm astfel:

(1) Ordonăm excursiile descrescător după capătul stîng.
(2) Trecem prin vector de la dreapta la stînga și, la fiecare poziție x, adăugăm $Value(x)$ într-o stivă de maxime ordonată descrescător. Așadar, $Value(x)$ va elimina din vîrful stivei toate valorile mai mici decît ea însăși.
(3) După adăugarea lui $Value(x)$ în stivă, răspundem la toate interogările care încep la poziția x. Pentru o interogare (x, y), răspunsul este cea mai mare valoare aflată în stivă pe o poziție mai mică sau egală cu y. Întrucît stiva este ordonată, putem căuta binar acea valoare.

Ziua 1 — Problema 3: F1

Propusă de: prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta-Turnu Severin

Soluție în $\mathcal{O}(N^6)$: Un algoritm forță brută stochează efectiv matricea de $N \times N$ elemente. Apoi iterează prin toate cele $\mathcal{O}(N^4)$ combinații de colțuri stînga-sus și dreapta-jos ale unei submatrice. Pentru o submatrice fixată, algoritmul calculează în $\mathcal{O}(N^2)$ suma submatricei. Dacă suma este 1, atunci matricea contribuie la puterea punctului pe care îl conține.

Dar trebuie să răspundem la întrebarea: despre care punct este vorba? Pentru a determina în $\mathcal{O}(1)$ punctul conținut, putem folosi un artificiu. În loc să stocăm în matrice valori 1, stocăm valori distincte pentru fiecare punct: 1 000, 1 001, 1 002... Atunci suma unei submatrice va fi $S \in [1\,000, 1\,999]$ dacă și numai dacă ea conține exact punctul $S - 1\,000$.

Soluție în $\mathcal{O}(N^4)$: Putem îmbunătăți soluția de mai sus dacă stocăm sume parțiale în locul matricei propriu-zise. Atunci putem calcula suma oricărei submatrice în $\mathcal{O}(1)$.

Soluție în $\mathcal{O}(N^3)$: Continuăm să îmbunătățim soluția de mai sus. Orice submatrice are o primă și o ultimă coloană. Să iterăm prin toate perechile de coloane (S, D) și să aflăm toate submatricele care se întind de la coloana S și pînă la coloana D și care conțin exact un punct. Colectăm, pentru fiecare dintre cele N linii, numărul de puncte pe acea linie între coloanele S și D. Putem face acest lucru în $\mathcal{O}(1)$ per linie, fie folosind matricea de sume parțiale, fie actualizînd aceste valori pe măsură ce coloana dreaptă D avansează spre dreapta.

Odată obținută lista de linii care conțin puncte, pentru fiecare punct X care este singur pe linie și care se numără printre cele P puncte de interes, puterea lui X crește cu $L_1 \times L_2$, unde L_1 și L_2 reprezintă distanțele pînă la precedenta, respectiv următoarea linie care conține puncte. Aceasta deoarece un dreptunghi care îl conține pe X poate să înceapă oriunde între linia anterioară care conține puncte (exclusiv) și linia lui X (inclusiv).

Soluție în $\mathcal{O}(Q \cdot N^2)$: Soluția de mai sus poate fi îmbunătățită. Pentru o pereche de coloane (S, D) nu este nevoie să iterăm prin toate liniile matricei. Putem să iterăm prin cele Q puncte și să le colectăm pe cele a căror coloană este cuprinsă între S și D. Avînd în vedere că la fiecare colectare dorim punctele sortate după linie, le sortăm o singură dată imediat după citirea datelor. Apoi, ca mai sus, puterea fiecărui punct crește cu $L_1 \times L_2$, unde L_1 și L_2 sînt distanțele pe linie pînă la punctele anterior și următor.

Remarcăm că acest algoritm tratează corect, fără cod suplimentar, cazul în care mai multe puncte se află pe aceeași linie. Ele vor fi consecutive în lista ordonată și toate vor avea putere 0, căci fie L_1, fie L_2 vor fi 0.

Un beneficiu suplimentar al acestei soluții este că nu mai construiește efectiv matricea, deci consumul de memorie este $\mathcal{O}(Q)$.

Soluție în $\mathcal{O}(Q^2 \cdot N)$: Față de algoritmul anterior facem următoarea observație. Pentru o coloană S fixată, cînd avansăm de la coloana D la $D+1$, dacă pe coloana $D+1$ nu se află niciun punct, atunci puterea totală pe intervalul $[S, D+1]$ va fi egală cu puterea pe intervalul $[S, D]$. De ce? Să ne amintim că algoritmul colectează punctele de pe coloane aflate în acel interval. Dacă coloana $D+1$ este goală, atunci lista de puncte colectate nu se modifică, deci implicit nici puterea totală nu se modifică.

Rezultă că D nu trebuie să itereze prin toate cele (maximum) N coloane, ci doar prin cele Q care conțin puncte. Odată calculată puterea pentru o pereche de coloane (S, D), o putem multiplica cu C, unde C este distanța de la D pînă la următoarea coloană care conține puncte.

Soluție în $\mathcal{O}(Q^3)$: Putem proceda pentru coloana S exact cum procedăm în soluția anterioară pentru coloana D. Așadar, S și D iterează prin cele Q puncte. Odată calculată puterea unui interval $[S, D]$, o multiplicăm cu $C_1 \times C_2$, unde C_1 și C_2 sînt distanțele pînă la punctul anterior lui S, respectiv următor lui D.

Soluție în $\mathcal{O}(P \cdot Q^2)$ (Cătălin Frâncu): Putem calcula explicit puterea fiecărui punct în $\mathcal{O}(Q^2)$ per punct. Grupăm punctele în benzi, după coloană. Apoi fixăm un punct X și iterăm prin perechi de benzi (i, j). Pentru fiecare pereche ne întrebăm: cîte submatrice conțin doar punctul X, au colțul stînga-sus între benzile i și $i+1$ și au colțul dreapta-jos între benzile j și $j+1$?

Putem răspunde în $\mathcal{O}(1)$ la fiecare întrebare, calculînd răspunsul pentru perechea $(i, j+1)$ din răspunsul la perechea (i, j). Menținem întinderea pe verticală a acestor dreptunghiuri. Inițial, pentru $i = j =$ banda din care face parte X, întinderea este dată de punctul anterior și următor lui X în banda sa (sau 0, respectiv $N+1$ dacă punctul X este primul, respectiv ultimul în banda sa).

Cînd trecem la următorul j, evaluăm toate punctele din noua bandă și, dacă ele sînt cuprinse în întinderea curentă, restrîngem această întindere.

Soluție în $\mathcal{O}(Q^2 \log Q)$ (Cătălin Frâncu): Soluția în $\mathcal{O}(Q^3)$ este ineficientă deoarece, pentru fiecare pereche de puncte (S, D), ea colectează naiv, în $\mathcal{O}(Q)$, punctele dintre acele coloane. Dar dacă am putea să menținem interactiv această colecție, cu efort doar $\mathcal{O}(\log Q)$ pentru a trece de la punctul D la $D+1$? Dorim să efectuăm eficient operațiile:

(1) Actualizare: Adaugă un punct la colecție.
(2) Interogare: Care este puterea colecției?

(**Notă**: Pentru brevitate spunem „puterea colecției". Formularea completă ar fi „contribuția acestei colecții, dintre coloanele S și D, la puterile punctelor pe care le conține și care sînt și printre cele P puncte de interes".)

Cum recalculăm puterea? Să considerăm o colecție cu 4 puncte pe liniile A, B, C și D. Atunci contribuția lui B este $(B-A)(C-B)$, iar contribuția lui C este $(C-B)(D-C)$. Acum să considerăm că adăugăm un punct pe linia X, iar colecția devine A, B, X, C, D. Trebuie să scădem vechile contribuții ale lui B și C și să adăugăm noile contribuții ale lui B, X și C. Puterile celorlalte puncte din colecție nu se modifică, deci putem recalcula puterea în timp constant.

Rezultă că trebuie să putem accesa în $\mathcal{O}(\log Q)$ punctele anterior și următor unui punct dat. Putem implementa această colecție cu diverse structuri de date. Soluția propusă sortează punctele după linie. Apoi menține un arbore Fenwick (AIB) de 0 și 1 în care bifează cu 1 indicii punctelor din colecția prezentă.

La adăugarea unui punct în colecție, să zicem cel cu indicele 17, vom interoga AIB-ul pentru suma pînă la poziția 17. Să presupunem că primim răspunsul 8, deci punctul tocmai inserat este al 8-lea în ordine în colecția curentă. Atunci putem căuta binar în AIB poziția pe care se ating sumele 7 sau 9 pentru a afla pozițiile elementului anterior sau următor.

Remarcăm că o căutare binară în AIB, implementată naiv, va face $\log Q$ calculări de sumă, fiecare cu costul $\mathcal{O}(\log Q)$, deci va avea complexitatea $\mathcal{O}(\log^2 Q)$. Dar ea poate fi implementată și în $\mathcal{O}(\log Q)$. Detalii aici. drive

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Panaete Adrian, Colegiul Național "A.T. Laurian", Botoșani
- Prof. Șerban Marinel, Colegiul Național "Emil Racoviță", Iași
- Prof. Chește Ciprian, Liceu Tehnologic "Grigore C. Moisil", Buzău
- Prof. Nodea Gheorghe Eugen, Colegiul Național "Tudor Vladimirescu", Târgu Jiu
- Prof. Piț-Rada Ionel-Vasile, Colegiul Național "Traian", Drobeta-Turnu Severin
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova
- Prof. Costineanu Raluca, Colegiul Național "Ștefan cel Mare", Suceava
- Prof. Cerchez Emanuela, Colegiul Național "Emil Racoviță", Iași
- Prof. Bunget Mihai, Colegiul Național "Tudor Vladimirescu", Târgu Jiu
- Prof. Boian Flavius, Colegiul Național "Spiru Haret", Târgu Jiu
- Prof. Szabo Zoltan, Inspectoratul Școlar Mureș
- Inst. Frâncu Cătălin, Clubul Nerdvana, București
- Stud. Tulbă-Lecu Theodor Gabriel, Universitatea Politehnică din București
- Stud. Bogdan Vlad Mihai, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Pop Ioan Cristian, Universitatea Politehnică din București
- Stud. Onuț Andrei, Universitatea Yale, S.U.A.

Cursurile de pregătire au fost susținute de:

- Prof. Marius Nicoli, Colegiul Național "Frații Buzești", Craiova
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova
- Inst. Frâncu Cristian, Clubul Nerdvana, București
- Stud. Popa Bogdan Ioan, Facultatea de Matematică și Informatică, Universitatea București
- Stud. Dăscălescu Ștefan-Cosmin, Facultatea de Matematică și Informatică, Universitatea București

Comisia tehnică a fost formată din:

- Prof. Opriță Petru Simion, Liceul "Regina Maria", Dorohoi
- Prof. Bolohan Mihai, Liceul "Regina Maria", Dorohoi