RAUCODER 2020

SOLUȚII

Data: **30 mai 2020**

Vă propunem mai jos soluțiile noastre.

Problema 1. Război

Autori: Adam Altăr, Daniela Alexandra Crișan

Cuvinte cheie: vectori de diferențe (Șmenul lui Mars), structuri (containere) dinamice

Ideea care stă la baza soluției noastre este utilizarea unui vector de diferențe. Astfel, în loc să se rețină vectorul A de vieți, se reține un vector D al diferențelor între oricare două elemente consecutive din A. Inițial D[1]=A[1] (presupunem că indexăm vectorii de la 1), apoi D[i] = A[i]-A[i-1], $2 \le i \le N$ și D[N+1] =0. Un update de forma (i, j, k) va consta în două operații: D[i] = D[i] – k și D[j+1] = D[j+1] + k. O interogare va presupune parcurgerea vectorului de diferențe D și refacerea lui A: A[1] = D[1] și A[i] = A[i-1] + D[i], $2 \le i \le N$, însoțită de contorizarea elementelor nenule.

Pentru că în specificația problemei se cere un N foarte mare, vectorul de mai sus va fi înlocuit cu un container dinamic (poate fi STL lista, vector sau map, etc.) de perechi de forma <int = indice, int = damage>. Să presupunem că folosim un vector STL. Inițial vectorul conține perechea pair<1, A[1]>. Apoi, la fiecare update, se adaugă în vector două elemente: pair<i, -k> și pair<j+1,k>. In final, vectorul va avea 2M noduri. Este evident că indicii se pot repeta. Dacă se utilizează un container map, atunci se vor face actualizari de elemente și indicii nu se vor mai repeta.

Pentru interogare, se sortează vectorul dupa indice și se parcurge de la st la dr, ținând cont că unii indici consecutivi (în urma sortării) se pot repeta.

Complexitate:

- de timp: $O(M*\log_2 M)$
  - update: adăugarea/accesarea unui element în container: timp constant (cf. specificației librăriei), poate fi $O(\log_{10} N)$, dat de numărul de cifre ale indicelui, în cazul memorării în bucket-uri (map).
  - sortare container: $O(M*\log_2 M)$
  - query: O(M)
- de memorie: O(M)

Problema 2. Pixeli

Autor: Daniela Alexandra Crișan

Cuvinte cheie: structura Trie

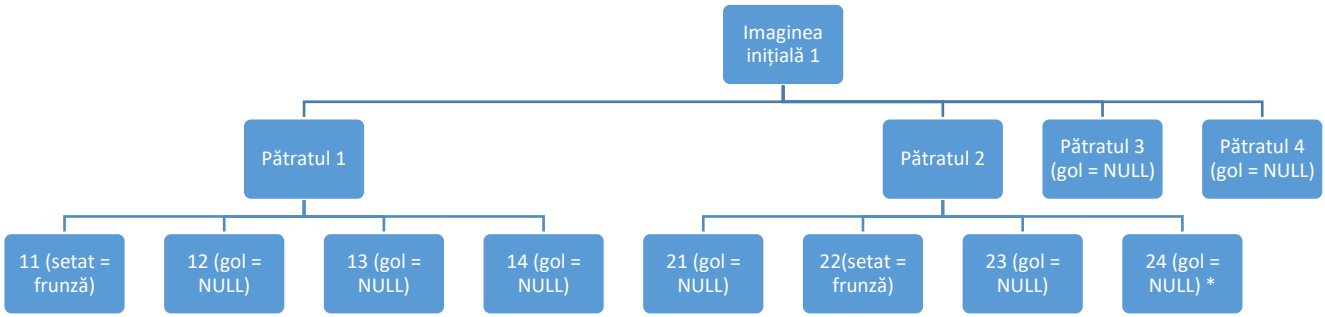
În soluția noastă am utilizat o structură de tip TRIE, la modă anul acesta 😊 . Pentru problema Pixeli, vom avea un arbore în care fiecare nod poate avea 4 fii, corespunzători direcțiilor 1, 2, 3 și respectiv 4, și înălțimea maximă $K=\log_2 N$. Un update poate fi de adăugare în arbore, atunci când pixelul urmează să fie setat, sau de ștergere, când pixelul este resetat. Adăugarea presupune inserarea unei frunze pe ultimul nivel (K+1, dacă se consideră că rădăcina este nivelul 1 și corespunde imaginii goale, inițială). Resetarea unui element presupune ștergerea frunzei corespunzătoare lui de pe nivelul K+1 și a tuturor nodurilor ascendente lui, care nu mai au alți fii.

Un astfel de arbore nu conține decât informația utilă, adică pixelii setați din imagine. Pentru $N = 2*10^9$, înălțimea arborelui nu va depăși 32.

Pentru exemplul furnizat în concurs, arborele asociat matricei

0 0 0 0	0 1 0 0
0 0 0 0	0 0 0 1

este:



Numai 2 pixeli sunt setați, deci arborele are numai 2 frunze, corespunzătoare direcțiilor 11 și 22. Restul fiilor sunt nuli.

O setare de pixel, 24 de exemplu, presupune adăugarea unei frunze pe ultimul nivel, pe direcția 24: nodul marcat cu * va deveni 24 (setat=frunza). Este posibil ca o adăugare să presupună inserarea unui șir întreg de noduri până la ultimul nivel.

O resetare de pixel, 11 de exemplu, presupune ștergerea frunzei 11 și a nodului notat Pătratul 1, întrucât în urma ștergerii frunzei, acesta devine frunză.

Query-ul presupune parcurgerea de la rădăcină în jos, pe direcțiile conținute în interogare, până fie se ajunge la o frunză de pe ultimul nivel (dacă pixelul este setat), fie ne oprim pe un nivel intermediar care ne dă răspunsul la problemă.

Complexitate:

- de timp: $M*O(\log_2 N)$
  - update: $O(\log_2 N)$
  - query: $O(\log_2 N)$
- de memorie: $O(N^2)$

Problema 3. Joc de șah

Autor: Daniela Alexandra Crișan

Cuvinte cheie: măști de biți, parcurgerea grafurilor/algoritmul lui Lee

Pentru aflarea lungimii minime a drumului dintre un rege și o regină se utilizează BFS sau algoritmul lui Lee. Acesta se rulează de mai multe ori, pentru diferite combinații de piese care pot fi folosite la o parcurgere, plecând de la combinația cu pierdere minimă. De exemplu, se începe cu damage 0 vieți (posibilă atunci un rege și o regină sunt alăturate), apoi se încearcă parcurgerea numai cu pioni (pierdere o viață), apoi numai cu cai (pierdere 2 vieți), urmează parcurgere cu pioni și cai (pierdere 1+2 = 3 vieți). Ordinea încercărilor este: 0 – nicio piesă, 1 – pioni, 2 – cai, 3 – pioni și cai, 4 – numai nebuni, 5 – pioni și nebuni, 6 – cai și nebuni, ..., 15 – toate piesele: pioni (1), cai (2), nebuni (4) și ture (8). Pentru a genera aceste combinații, se va lucra pe numere de la 0 la 15, la nivel de biți.

Ne oprim din căutare la prima parcurgere (cu damage minim) care ne arată că există un drum intre un rege și o regină. Evident, pot fi mai multe drumuri. Ca să refacem drumul conform cerințelor problemei, noi am recurs la o strategie: am plecat de la regine către regi, astfel că acum avem pentru toți regii lungimile drumurilor minime către o regină. Alegem primul rege (lexigografic) cu cel mai mic drum, fie *cost* lungimea acestuia. Plecând din el, alegem dintre vecinii săi nodul cel mai mic lexicografic (în ordinea N, V, E, S) cu costul *cost-1*. Și tot așa până ajungem la o regină.

Și pentru că vorbim despre un concurs, iar un concurs este despre învățare, ne asumăm că era bine ca regii să fi avut inițial 16 vieți, ca să le mai rămână cel puțin una indiferent de combinația de piese pe care le atacau în drumul lor.

Complexitate:

- de timp: $O(N^2)$
  - parcurgere: $O(N^2)$
  - refacere drum: $O(N^2)$
- de memorie: $O(N^2)$