ROSEdu Summer Workshops
- Tabele de hashing -

1. Despre mine

- proaspat absolvent al Liceului International de Informatica Bucuresti

- pe parcursul liceului am participat la olimpiada, cele mai importante distinctii fiind doua premii I la nationala si doua medalii de argint la BOI

- momentan particip la Google Summer of Code, lucrez pentru organizatia KDE, la proiectul Marble – un glob virtual, asemanator cu Google Earth

- de la sfarsitul lunii august voi fi student la Yale University, unde cel mai probabil voi face un major in informatica&psihologie.

- fac parte din echipa Infoarena, asa am aflat de eveniment

2. Hashing – importanta

- hashurile, in opinia mea, cea mai importanta / folosita structura de date, cel putin in concursurile de informatica / problemele de interviuri

- aplicatii practice in multe domenii ale informaticii - criptografie, string-uri, baze de date, etc. dar probabil voi stiti asta mult mai bine decat mine :)

3. Prezentarea structurii

Utilitate

- o structura de date extrem de rapida pe operatii de inserare / cautare / stergere - timp mediu de executie → O(1)

Cum functioneaza?

- ideea de baza: alocarea unei chei fiecarui element si inserarea lui la cheia respectiva in structura de date; cautarea se face tot dupa acea cheie;

Exemplu:

Se da o multime, initial vida si trei operatii pe ea:

a) UPDATE: Insereaza elementul X in multime

b) UPDATE: Sterge elementul X din multime (daca acesta exista)

c) QUERY: Exista elementul X in multime?

- cu precizarea ca toate numerele prezente in problema sunt <= 10^6 si nu se repeta (e multime)

Solutie: in mod evident, din cauza restrictiei de mai sus, cheia atribuita numarului X poate fi chiar X => un vector caracteristic, care ne va spune daca elementul e sau nu prezent in multime.

- ce se intampla daca restrictia cu X <= 10^6 se transforma in X <= 10^18?
=> Nu o sa ne mai permita memoria sa tinem cheia X=> se ridica necesitatea unei functii de atribuire a cheii

Functii de hashing

1) Pentru numere intregi

- cea mai folosita functie de hashing pe numere intregi este modulo → $f(x) = x \bmod p$. Pentru a evita coliziunile (despre care o sa discutam putin mai tarziu), de obicei se foloseste un numar p prim.

2) Pentru stringuri

- se considera stringul ca fiind un numar intr-o baza mai mare sau egala cu dimensiunea alfabetului
- se aplica functia de hashing pe numere intregi
- algoritmul Rabin-Karp pentru pattern matching →
    - se dau doua siruri de caractere, A si B.
    - se cere sa se determine daca B apare in A ca subsecventa.

In functie de structura care trebuie hash-uita se gasesc functii de hashing potrivite.

Coliziuni

- dupa cum probabil ca ati observat, se poate sa existe doua valori x diferite pentru care functia de hashing sa returneze aceeasi valoare. Aceste cazuri se numesc coliziuni
- de exemplu, luam functia de hashing f(x) = x % 7. Functia va returna aceeasi valoare pentru 5 si 12.
- daca functia este aleasa in asa fel incat sa nu existe coliziuni => hashing perfect. (de exemplu f(x) = x).

- problema este ca de cele mai multe ori un hash perfect este extrem de costisitor in materie de timp / memorie asa ca se prefera tratarea coliziunilor prin alte metode

- una dintre cele mai folosite metode → pentru fiecare cheie se pastreaza o lista cu toate valorile diferite care duc in cheia respectiva (se poate implementa ca lista simplu inlantuita, vector STL sau orice altceva ce permite alocare dinamica a memoriei)

- timp / memorie?

- o alta metoda, mai eficienta ca timp dar mai putin precisa: double hashing

- dupa cum spune si numele, se tin doua tabele hash in loc de una, fiecare cu functia ei

- fiecare element se adauga in ambele

- in cazul problemei initiale: un element se afla in multime doar daca ambele hash-uri ne confirma asta.

- timp / memorie?

4. Implementare

- de mana
- cu map din STL
- cu set din STL

5. Probleme

1. Loto (http://infoarena.ro/problema/loto)

2. Eqs (http://infoarena.ro/problema/eqs)

3. Ratina (http://infoarena.ro/problema/ratina)

4. Secv5 (http://infoarena.ro/problema/secv5) – are mai putina legatura cu hash-urile, dar e la nivel de dificultate de interviu, poate putin mai grea

5. Ograzi (http://infoarena.ro/problema/ograzi)

6. Regiuni (http://infoarena.ro/problema/regiuni)

7. Poze (http://infoarena.ro/problema/poze)

8. Siruri (http://infoarena.ro/problema/siruri)

9. Parpal (http://infoarena.ro/problema/parpal)

10. Se da un poligon convex. Sa se determine axele sale de simetrie. (http://infoarena.ro/problema/dmg, enunt adaptat)

6. Solutii

- Loto

- se memoreaza toate sumele de cate 3 intr-un hash, apoi se fac din nou sume de cate 3 si se cauta in hash S – suma curenta

- Eqs

- asemanator cu loto, din cauza ca fiecare X e cuprins intre -50 si 50

- Ratina

- pentru fiecare string se construieste H[i] = functia de hash a caracterelor de la 1 pana la i in stringul respectiv.

- pentru un query se face o cautare binara

- Secv5

- se normalizeaza valorile (aici se golosesc hash-uri / map / set)

- se creaza functia solve(X) care returneaza cate subsecvente au intre 1 si X elemente distincte, rezultatul final fiind solve(U) – solve(L - 1)

- in functia solve se parcurge sirul cu doi pointeri, unul la stanga si unul la dreapta; in momentul in care pointerul din dreapta da un nou element, si numarul de elemente distincte devine mai mare decat este permis, se muta pointerul din stanga inspre dreapta, astfel incat numarul de elemente distincte din secventa determinata de cei doi pointeri sa revina in limitele admise

- Ograzi

- se imparte planul intr-un grid cu celula de dimensiune WxL, fiecare punct o sa fie intr-o casuta

- se face hash pe casute si se pune fiecare punct in lista corespunzatoare casutei in care se afla

- un query (dreptunghi WxL) se intinde pe maxim 4 casute

- din cauza ca sunt complet disjuncte query-urile, fiecare punct o sa fie parcurs de maxim 4 ori – deseneaza daca nu ma crezi ;)

- Poze

- se cauta binar latura

- pentru o latura fixata, X, se face un hash 2D pe toate submatricele de latura X si se numara.

- daca numarul de submatrice identice e >= K, se incearca cu o latura mai mare in cautarea binara, daca nu, cu una mai mica

- Regiuni

- fiecare regiune e unic determinata de partile in care punctele din regiunea respectiva sunt fata de toate dreptele

- se construieste pentru fiecare punct un sir de lungime M, continand 1 sau -1, care reprezinta in ce parte a fiecarei drepte se afla punctul respectiv.

- pentru a verifica numarul maxim de puncte din aceeasi regiune in timp rapid se hash-uiesc sirurile construite

- Siruri

- se cauta binar K

- pentru un K fixat se hash-uiesc toate subsecventele de lungime K din ambele siruri si se cauta egalitati (hash egal = subsecvente egale)

- Parpal

- D[i] = 1 daca secventa de la 1 la i se poate scrie ca si concatenare de palindroame pare; altfel D[i] = 0

- se parcurge sirul cu un indice, i, prin care se fixeaza mijlocul palindromului curent

- se cauta binar lungimea maxima a palindromului curent → pentru a verifica daca secventa de la x la y din sir este palindromica, se tin 2 hash-uri: unul pe sirul de la 1 la N si unul pe sirul de la N la 1

- fie 2 * Z lungimea palindromului determinat. Daca exista vreun 1 in intervalul i-Z .. I (fie acesta la distanta T de pozitia i) => am gasit o continuare, updatam D[i+T] cu 1

- Dmg

- axa de simetrie = doua bucati (linii frante) congruente

- doua linii frante sunt congruente daca toate segmentele si unghiurile dintre ele sunt congruente, in ordinea potrivita

- se parcurge poligonul cu doi pointeri, “diametral opusi”, si se mentine valoarea hashului facut pe lungimi+unghiuri pentru fiecare dintre cele 2 jumatati (atentie, la una dintre ele elementele intra prin fata, la cealalta prin spate)