**Кумулативне табеле, (Binary Indexed Trees – BIT), (Fenwick tree)**

Jedan od čestih problem koje srećemo u programiranju jeste problem Kumulativnih tabela, koji

se zasniva na sledećim operacijama nad “dinamičkim” nizom (PrefixSum ovde ne pomaže jer se članovi niza menjaju u toku izvršenja programa):

• ***add*(*v*, *x*)** – povećaj *v* -ti član niza za *x*

• ***sum*(*v*)** – nađi sumu prvih *v* -članova niza

Naravno da se ovo može primeniti na sumu bilo kog uzastopnog intervala odnosno na postavljanje vrednosti nekog člana niza.

Nameće se rešenje, gde ćemo u nizu *d*[*i*] , čuvati vrednosti niza, a traženje sume prvih članova ćemo

naći sumiranjem prvih n članova niza. To povlači sa sobom složenosti *O(1)* za postavljanje (ili sumiranje)i *O(n) z*a sumiranje. Do bolje optimizacije rešenja dolazi se ukoliko pokušamo da smanjimo složenost funkcije ***sum.*** Struktura podataka koja rešava ovaj problem u složenosti *O*(log *n*), za obe operacije, jesu Kumulativne tabele.

Dokaz da Kumulativne tabele rade ispravno ćemo preskočiti. Suština je u tome da se suma odredjenog dela niza (matrice, trodim niza, ...) ne čuva kao suma svih članova od prvog do datog, nego kao suma samo nekih, odredjenih, članova. Ti određeni članovi predstavljaju sume nekih članova BIT-a, ali tako da unija (zbir) tih članova daje sumu celog opsega.

Procedura **dodavanja** ide na sledeći način C++:

**int KT[N+5];**

**void add(int x, int kolko) {**

**while (x <= N) {**

**KT[x] = KT[x] + kolko; //** neka mesta u strukturi se uvećavaju za kolko

**x = x + (x & (-x));** // x and (-x) je izraz koji odredjuje ta mesta

**}**

**}**

U slučaju da se **postavlja** vrednost a ne dodaje kao u gornjem slučaju, treba prvo prvo dodati **-onoliko** koliko ima na tom polju, pa onda dodati vresnost koja se postavlja.

Funkcija koja **dešifrije sumu** svih članova od početka niza do tog člana izgleda ovako C++:

**int sum(int x){** // zbir svih članova iz intervala od 1 do x (ako postojoi opasnost od velikog zbira sve je LL)

**int PomSum = 0;**

**while (x>0) {**

**PomSum = PomSum + KT[x];** // neka mesta iz strukture ulaze u zbir

**x = x – (x & (-x));** // x - x and (-x) je izraz koji odredjuje ta mesta

**}**

**Return PomSum;**

**}**

Gore je opisan jednodimenzioni slučaj Kumulativnoh tabela. Pri tome ako je potreban neka suma intervala od A do B, tada koristimo izraz **Sum(B)-Sum(A-1).**Kod dvodimenzionalnih Kumulativnih tabela (struktura matrice), optimalnost algoritma je još izraženija. Kod obične strukture, *O(1)* za postavljanje (ili sumiranje)i *O(n2) z*a sumiranje. Dok je kod Kumulativnih matrica i postavljanje i sabiranje složenosti Log2(n). Uporedjenja radi, matrica 1000x1000 treba   
  
Obična matrica – 1 000 000 operacija (1000 \* 1000)  
Kumulativnom tabelom – 800 operacija (2 \* 20 \* 20)

Donji deo je uradjen u PASCAL-u

**Var KT : array [1..N, 1..M] of LongInt;**

**Procedure Add(a, b, Broj: LongInt);**

**Var PomB : LongInt;**

**Begin**

**PomB := b;**

**While a <= n do { u neke redove (promenljiva a) na**

**begin**

**b := PomB; neka mesta (promenljiva b)**

**While b <= m do**

**Begin**

**KT[a,b] := KT[a,b] + Broj; dodajemo vrednost (Broj)}**

**b := b + b and -b;**

**end;**

**a := a + a and -a;**

**end;**

**end;**

Kao i kod jednodimenzionih Kumulativnih Tabela i kod dvodimenzionih Kumulativnih Tabela moramo da izvršimo korekciju vrednosti u slučaju da se ne dodaje nego postavlja vrednost u polje.

**Procedure Set(a, b, Broj: LongInt);**

**Var PomB, Korekcija : LongInt;**

**Begin**

**Korekcija := Sum(a,b) - Sum(a-1,b) - Sum(a,b-1) + Sum(a-1,b-1);**

**PomB := b;**

**While a <= n do { u neke redove (promenljiva a) na**

**begin**

**b := PomB; neka mesta (promenljiva b)**

**While b <= m do**

**Begin**

**KT[a,b] := KT[a,b] – Korekcija + Broj; dodajemo vrednost (Broj) a oduzimamo staru }**

**b := b + b and -b;**

**end;**

**a := a + a and -a;**

**end;**

**end;**

**Function Sum(a, b: LongInt): LongInt;**

**Var PomSum, PomB : LongInt;**

**Begin**

**PomSum := 0;**

**PomB := b;**

**While a > 0 do**

**begin**

**b := PomB;**

**While b > 0 do**

**Begin**

**PomSum := PomSum + KT[a, b];**

**b := b - b and -b;**

**end;**

**a := a - a and -a;**

**end;**

**exit (PomSum);**

**end;**

Ako je potrebna suma pravougaone oblasti koja ne ide od polja (2,4) do (6,7) tada je to:

| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |
| **.** | **.** | **.** | **.** | **.** | **.** | **.** | **.** |

**Suma osenčene oblasti**:  **= Sum(6,7) - Sum(1,7) - Sum(6,3) + Sum(1,3);**

**U svim primerima, gornji levi ugao je predstavljao prvo polje strukture.**

**Analogno ovome može se uraditi i proširenje na tro i više dimenzione kumulativne tabele.**

**Linkovi:**[**http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees**](http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees)