**КУБЧЕТА**

**I начин:** Прочитат се първо данните в масив А[], които описват как действа всяко дете. Понеже заявките В[] са offline, те може да се сортират. След това започваме да обработваме подред данните от масива А /в един вектор записваме поредната купчина от кубчета/. При достигане на някоя заявка от B[] запомняме получения до момента вектор. Това е решението в ***kub.cpp***.

**II начин:** Четем данните, които описват действията на децата и запомняме за всяко дете I кое дете J е сложило предишното кубче и кое е то.

Ако сме на позиция I, ни интересува предното дете с номер I-1. Ако то е сложило кубче, насочваме I към I-1. Когато предишното дете е махнало кубче, насочваме I към този номер, към който е насочено I-1-то дете:

Използва се структура за всеки елемент на масива a[]: ***st*** е стойността на кубчето, а ***p*** сочи предишния:

if(c=='-')

{ // **Когато i-я ученик маха най-горното кубче**

 a[i].st=0;

 if(a[i-1].st>0) // Предишният не е 0

 a[i].p=a[i-1].p; // насочваме към предходника на предишния

 else // иначе

 a[i].p=a[a[i-1].p].p; // насочваме към предходника на предходника му

}

else{ // **Когато i-я ученик поставя ново кубче**

 cin>>k;

 a[i].st=k;

 if(a[i-1].st==0)

 a[i].p=a[i-1].p;

 else

 a[i].p=i-1;

}

При четене на заявка за дете с номер I просто се връщаме назад и възстановяваме колоната на кубчетата, след това изкарваме номерата им отзад-напред.

Такова решение е реализирано в ***kub\_p.cpp***.

Първият начин използва малко повече памет – два масива от тип структура и един масив от вектори. Вторият начин изисква само един масив, но при отпечатването се връщаме назад колкото кубчета има съответното дете.

*Автор: Павел Петров*