АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧАТА ЗВЕЗДИЧКИ

За да разберем кое подред трябва да застане патето за да спечели трябва да разберем колко различни фигури можем да направим като свържем между **K1** и **K2** различни точки и да разделим броя им на броя патета **M**, получилият се остатък ще е позицията, на която патето трябва да застане. За да решим по-лесно задачата може да я сведем до нейна подзадача, когато **K1** и **K2** съвпадат. В такъв случай, лесно се забелязва, че за да се намерят броят на всички фигури от **K** точки при **N** възможни точки, трябва да се направят комбинации **N** елемента от **K**-ти клас. Сега вече могат да се разгледат случаите, когато **K1** и **K2** са различни. В тези случаи общия брой на всички възможни фигури е сбора на комбинациите на **N** елемента между **K1** и **K2** клас.

Поради малкото **N** и малкия брой заявки **T**, такова е и решението за 40 точки. За всяка една от заявките се намира сбора на комбинациите, от комбинации **N** елемента **K1** клас до комбинации **N** елемента **K2** клас, и за да се получи съответния отговор за всяка заявка, се взима остатъка при деление на **M**.

Поради големината на **N** при задачите за 80 и 100 точки се изисква да се използва Триъгълникът на Паскал. Той представлява вид двумерно динамично оптимиране, а получената матрица има свойството, че на **i**-ти ред **j**-та колона се пази комбинациите на **i** елемента **j**-ти клас. Формулата, по която се получава матрицата е

dp[i][j]=dp[i-1][j-1]+dp[i-1][j]

Това значи, че ако числата в матрицата се наклонят, така че да наподобяват триъгълник, всяко число в този триъгълни ще зависи само и единствено от двете числа над него.

Задачата за 80 точки се решава точно като се прави тази матрица до **N**-тия ред включително, където всъщност ще се намират всички комбинации на **N** елемента. За да се получи отговора трябва, да се сумират стойностите в клетките между **K1** и **K2** и да се вземе остатъка при делене на **M**. За да може обаче да се достигне до **N**=10 000 ще е нужна и една оптимизация. Ако пазим в матрицата самите числа тя ще продължава да връща верни отговори най-много до **N**=64, за да оправим този проблем ще се наложи да държим стойностите в матрицата под модул. Тъй като все пак нашият отговор вече се иска да се изведе под модул **M**, се оказва удобно и в самата матрица те да са под модул **M**.

```
for(i=0; i<=n; i++)
{
  pascal[i][0] =1%m;
  for( j=1; j<=i; j++ )
    pascal[i][j]+=( (pascal[i-1][j-1]  +  pascal[i-1][j] ) %m);
}
```

При задачата за 100 точки се използва същият принцип, но поради, това че **N**=25 000, а толкова голяма матрица не може да се създаде и поради големия брой заявки ще са нужни още 2 оптимизации. Първата ще поправи проблема с големината на **N**. Можем да забележим, че всеки следващ ред в матрицата зависи само от предходния, а тъй като на нас ни трябва само **N**-тия ред, това означава че останалата част от матрицата няма да ни е нужна и следователно само заема излишна памет. Следователно можем да оставим матрицата само от 2 реда, а да я направим с желания брой колони.

```
for(i=0; i<=n; i++)
  {
   pascal[i&1][0] =1%m;
   for( j=1; j<=i; j++ )
     pascal[i&1][j]=( pascal[(i-1)&1][j-1] + pascal[(i-1)&1][j] ) %m;
  }
```

Втората оптимизация ще свежда скоростта за изпълнение на заявка от приблизително **O(N)** до **O(1)**.Ако вместо на позиция **j** на **N**-тия ред пазим не комбинациите **N** елемента **j**-ти клас, а сбора от всички комбинации до текущата включително нея самата. После ще можем да отоговаряме на заявка с **O(1)** като от **K2** клетка извадим **K1-1** клетка, това ще ни даде сбора на всички клетки от **K1** до **K2**. Това можем да постигнем като, след като построим триъгълника на Паскал по горе споментаия начин минем отново по последния ред и на текущатата клетка **i** запишем собра от **i**-тата и **i-1**-та клетка.

```
for(i=1; i<=n; i++)
  pascal[n&1][i]+=pascal[n&1][i-1];
```

Автор: Теодор Тодоров