```
{fn=in;
fscanf(fn,"%d",&c,&m);
nei.resize(c);
back=nei;
v.resize(c);
res.resize(c);
for(q1=0;q1<c;q1++)
  {fscanf(fn,"%d",&c1);
  for(q2=0;q2<c1;q2++)
    {fscanf(fn,"%d",&c2);
    c2--;
    nei[q1].pb(c2);
    back[c2].pb(q1);}
  }
fclose(fn);
for(q1=0;q1<c;q1++)if(back[q1].sz==0)break;
dfs(q1);
res[top[c-1]]=1;
for(q1=c-2;q1>=0;q1--)
  for(q2=0;q2<back[top[q1]].sz;q2++)
    res[top[q1]]=res[top[q1]]+res[back[top[q1]][q2]];
write(res[top[0]]);
return 0;}
```

Задача А3. ИЗРАЗ

Киро и приятелите му играят странна игра. Кодират математически израз с 0 и 1 в строго определен формат. Всеки от играчите има право да промени израза с максимум **K** замени. Печели този от тях, който след като се пресметне променения израз получи, в резултат на пресмятането на израза, максимално число. Изразът е дефиниран с правилата:

<**израз**>:=<цифра><знак><цифра><знак>...<цифра>

<**цифра**>:=0,1,2,3,4,5,6,7,8,9 <**знак**>:=+,-

Записването на цифрите и знаците в израза с 0 и 1 става по следния начин (удебеленият шрифт е за да личат по ясно нулите и единиците) :

0	1	2	3	4	5	6	7	8	9	+	–
111	00**1**	**111**	**111**	**101**	**111**	**111**	**111**	**111**	**111**	000	000
10**1**	00**1**	00**1**	00**1**	**1**0**1**	**1**00	**1**00	00**1**	**1**0**1**	**1**0**1**	0**1**0	000
10**1**	00**1**	**111**	**111**	**111**	**111**	**111**	00**1**	**111**	**111**	**111**	**111**
10**1**	00**1**	**1**00	00**1**	00**1**	00**1**	**1**0**1**	00**1**	**1**0**1**	00**1**	0**1**0	000
111	00**1**	**111**	**111**	00**1**	**111**	**111**	00**1**	**111**	**111**	000	000

Под промяна на израз с **K** замени се разбира инвертирането (замяна на 1 с 0 и обратно) на **K** цифри, така че общият брой на единиците в новия израз да е равен на броя на единиците в оригиналния израз. При промяна на израз, знак може да се трансформира само в знак и цифра може да се трансформира само в цифра, т.е. не може цифра да стане знак и обратното. В израз участва минимум една цифра и максимум 50. Пример за промяна на израз с 4 замени (с удебелен шрифт са промените):

Изразът преди промяната	Изразът след промяната
111000**111**	111000**00**1
100000000**1**	10**1**000001
11111100**1** <=> 5-7 = -2	111111001 <=> 8-1 = 7

00**1**000000**1**	**1**01000001
111000000**1**	111000001

Напишете програма **EXPRES,** която по зададен първоначален израз и число **K** пресмята колко е максималното число, на което може да се оцени променен израз ако се направят максимално **K** замени.

Вход: Данните са записани в 6 реда на стандартния вход. На първия ред стоят две числа: **N** - броят на цифрите участващи в израза и **K** - броят на максимално разрешените замени. На следващите 5 реда е записан изразът.

Изход: На първия ред на стандартния вход програмата трябва да запише максималната възможна стойност на променения израз, на останалите 5 реда – променениият израз. Ако има няколко решения програмата да изведе кое да е от тях.

ПРИМЕР

Вход:	**Изход:**
3 4	11
1110001110001 11	111000**00**1000111
1000000010000 01	100000001**1**0001
1111110011110 01	111111001111001
0010000010000 01	001000001**1**0001
1110000010000 01	111000001000001

Решение:

Задачата се решава динамично.

1. Прочитат се входните данни.
2. Изчисляват се разликата между всеки две от числата от -9..9 като се слага допълнително и -0.
3. Създава се тримерен масив по max[1..N,0..K,-K..K], които попълваме динамично: първо се движим по цифрите, после по промените до сега и накрая по баланса на 1-ците т.е. дали досега сме ползвали повече 1-ци или са ни останали в излишни. На всяка стъпка минаваме през всички числа от -9 до 9 (като се специални случаи са -0 и първата цифра в израза, която няма знак) и трансформираме текущото число от израза до него с разлика d и разлика в броя на 1-ците r тогава max[currentNumber,currentDifference,currentBalance] = max(max[currentNumber,currentDifference,currentBalance],max[currentNumber-1,currentDifference - d,currentBalance - r])
4. Накрая извеждаме максимума по i от max[N,i,0]

```
{$H+}
uses SysUtils;

Var
   FIn,FOut : Text;

Const
   BitMap : array[0..11] of string =
      ('111101101101111', // 0
       '001001001001001', // 1
       '111001111100111', // 2
       '111001111001111', // 3
       '101101111001001', // 4
       '111100111001111', // 5
```