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
```

```
        '111100111101111', // 6
        '111001001001001', // 7
        '111101111101111', // 8
        '111101111001111', // 9
        '000010111010000', // +
        '000000111000000'  // -
        );

    PLUS_INDEX : integer = 10;
    MINUS_INDEX : integer = 11;

    MAXN = 100;
    MAXK = 50;

Var
    numbers : array [1..MAXN] of integer;
    signs : array [1..MAXN] of integer;
    inputStrings : array [1..MAXN * 2] of string;
    N,K : integer;
    max : array [1..MAXN,0..MAXK,-MAXK..MAXK] of integer;
    solution : array [1..MAXN,0..MAXK,-MAXK..MAXK] of string;
    diff : array[0..15,0..15] of integer;
    ones : array[0..15] of integer;

Function GetDifference(index1,index2 : integer) : integer;
Var
    i,sum : integer;
    s1,s2:string;
Begin
    s1 := BitMap[index1];
    s2 := BitMap[index2];
    sum := 0;
    for i := 1 to length(s1) do
        if s1[i] <> s2[i] then
            inc(sum);
    GetDifference := sum;
End;

Function GetSignBitMapIndex(sign : integer) : integer;
Begin
    if sign = 1 then GetSignBitMapIndex := PLUS_INDEX
    else GetSignBitMapIndex := MINUS_INDEX;
End;

Function GetOnesCount(index:integer) : integer;
Var
    i,sum : integer;
    s: string;
Begin
    s := BitMap[index];
    sum := 0;
    for i := 1 to length(s) do
```

```
      if s[i] = '1' then inc(sum);

   GetOnesCount := sum;
End;

Procedure Init;
Var
   i,j : integer;
Begin
   for i := 0 to 11 do
      for j := 0 to 11 do
         diff[i,j] := GetDifference(i,j);

   for i := 0 to 11 do
      ones[i] := GetOnesCount(i);
End;

Procedure Solve(var result : integer; var solutionString : string);
Var
   numIndex,diffIndex,balIndex : integer;
   sign,number : integer;
   currentDiff,currentBalance : integer;
   oldDiff,oldBalance : integer;
   currentPath : string;
   currentValue : integer;

Begin
   for numIndex := 1 to N do begin
      for diffIndex := 0 to K do begin
         for balIndex := -K to K do begin
            max[numIndex,diffIndex,balIndex] := -10000;
            sign := 1;
            while sign >= -1 do begin

               if (sign = -1) and (numIndex = 1) then break;

               for number := 0 to 9 do begin
                  currentDiff :=
                     diff[GetSignBitMapIndex(sign),GetSignBitMapIndex(signs[numIndex])] +
                     diff[number, numbers[numINdex]];

                  if currentDiff > K then continue;

                  oldDiff := diffIndex - currentDiff;

                  if (oldDiff > K) or
                    (oldDiff < 0) then
                     continue;

                  currentBalance :=
                     ones[GetSignBitMapIndex(sign)] -
                     ones[GetSignBitMapIndex(signs[numIndex])] +
```

```
                    ones[number] -
                    ones[numbers[numIndex]];
                oldBalance := balIndex - currentBalance;

                if (oldBalance < -K) or
                  (oldBalance > K) then
                    continue;

                currentValue := sign * number;

                if numIndex > 1 then begin
                    currentPath := Chr( Ord('0') + number);
                    if sign = -1 then
                        currentPath := solution[numIndex-1,oldDiff,oldBalance] + '-' + currentPath
                    else
                        currentPath := solution[numIndex-1,oldDiff,oldBalance] + '+' + currentPath;

                    currentValue := currentValue + max[numIndex - 1,oldDiff,oldBalance];
                end
                else begin
                    if (currentDiff <> diffIndex) or
                      (currentBalance <> balIndex) then continue;

                    currentPath := Chr( Ord('0') + number);
                end;

                if currentValue > max[numIndex,diffIndex,balIndex] then
                begin
                    max[numIndex,diffIndex,balIndex] := currentValue;
                    solution[numIndex,diffIndex,balIndex] := currentPath;
                end;
            end;
            sign := sign - 2;
        end;
    end;
  end;

  result := -1000;
  for diffIndex := 0 to K do begin
    if result < max[N,diffIndex,0] then begin
      result := max[N,diffIndex,0];
      solutionString := solution[N,diffIndex,0];
    end;
  end;
End;

Procedure ReadInput;
Var
  lineIndex,i,j,stringIndex,sign : integer;
  line : string;
```

```
Begin
   assign(FIn,'Expression.inp');
   reset(FIn);
   readln(FIn,N,K);

   for lineIndex := 1 to 5 do begin
      Readln(FIn,line);
      stringIndex := 0;
      for i := 1 to (N*2 - 1) * 3 do begin
         if i mod 3 = 1 then inc(stringIndex);
         inputStrings[stringIndex] := inputStrings[stringIndex] + line[i];
      end;
   end;

   stringIndex := 1;
   for i := 1 to N do begin
      sign := 1;

      if i > 1  then begin
         if inputStrings[stringIndex] = BitMap[MINUS_INDEX] then // - sign
            sign := -1;
         inc(stringIndex);
      end;

      for j := 0 to 9 do
         if BitMap[j] = inputStrings[stringIndex] then begin
            numbers[i] := j;
            signs[i] := sign;
            break;
         end;

      inc(stringIndex);
   end;

   close(FIn);
End;

Function GetCharBitMapIndex(c:char) : integer;
begin
   if c in ['0'..'9'] then
      GetCharBitMapIndex := ord(C) - ord('0')
   else if c = '+' then
      GetCharBitMapIndex := PLUS_INDEX
   else
      GetCharBitMapIndex := MINUS_INDEX;
end;

Procedure WriteSolution(result : integer; solution : string);
Var
   line,i,j : integer;
   s : string;
begin
```

```
   assign(FOut,'expression.out');
   rewrite(FOut);
   writeln(FOut,result);
   for line := 0 to 4 do begin
      for i := 1 to length(solution) do begin
         s := BitMap[ GetCharBitMapIndex(solution[i])];
         for j := 1 to 3 do
            write(FOut,s[ line * 3 + j]);
      end;
      writeln(FOut);
   end;
   close(FOut);
End;

var
   r : integer;
   s : string;

Begin
   ReadInput;
   init;
   solve(r,s);
   WriteSolution(r,s);
End.
```

ТЕМА ЗА ГРУПА В

Задача В1. БЛИЗКИ ДУМИ

Крайната азбука **A** се състои от главните латински букви, арабските цифри и подчертаващо тире. За всеки знак a от **A** имаме зададено цяло число $V(a)$ – видимост на знака, $0 \le V(a) \le 100$. Видимостта $V(a)$ на знака a е числова оценка за това, колко е вероятно човек да пропусне този знак при четене. Например, буквата **'W'** е много по-видим знак от подчертаващото тире (**'_'**). За всеки два знака **a** и **b** от **A** е зададено цяло число $D(a,b)$ – различност на двата знака, $0 \le D(a,b) \le 100$, $D(a,b) = D(b,a)$ и $D(a,a)=0$. Различността $D(a,b)$ е толкова по-малка, колкото е по-вероятно знакът a да бъде сбъркан със знака **b** при четене. Например, различността между буквата '**O**' и цифрата '**0**' е малко число. Да разгледаме низа $\alpha=a_1a_2...a_m$ над **A**. Дефинираме следните "претеглени" операции върху низа α:

insert – вмъкване на знак на произволна позиция в низа α. Тази операция има тегло $V(a)$, където a е вмъкнатият знак.

update – замяна на един знак от низа α с друг. Ако знакът a е заменен със знака b, теглото на операцията е $D(a,b)$.

delete – изтриване на знак. Тежестта на операцията е $V(a)$, където a е изтритият знак.

Разстояние между низовете $\alpha=a_1a_2...a_m$ и $\beta=b_1b_2...b_m$ наричаме минималната сума от тегла на редица от операции, чрез които от α може да се получи β. Напишете програма **LDIST**, която при зададени $V(a)$ и $D(a,b)$ да намира разстоянието между два низа α и β.