**http://www.math.bas.bg/~nkirov/2009/PFC/lecture02.html**

**2. Динамично оптимиране**

**Задача за раницата [8.2.1]**

Дадена е раница с вместимост *M* килограма и *N* предмета, всеки от които се характеризира с две числа - тегло *mi* и стойност *ci*. Да се избере такова множество от предмети, чиято сумарна стойност е максимална, а сумата от теглата не надвишава *M*.

Дефинираме рекурентна целева функция:

*F*(0) = 0;  *F*(*i*) = max { *cj* + *F*(*i - mj*),  *j =* 1, 2, ..., *N*,  *mj ≤ i* }, *i >* 0

Методът на динамичното оптимиране изисква последователно пресмятане на стойностите на *F*(*i*), като за това пресмятане се използват вече пресметнатите стойности за по-малки *i*.

Да разгледаме примера:

N = 8;  
index  1  2  3  4  5  6  7  8  
m[8] 3, 7, 6, 1, 2, 4, 5, 5  
c[8] 5, 3, 9, 1, 1, 2, 5, 2  
M = 8;  
  
Fn[0] = 0;  
Fn[1] = max{c[4]+Fn[0]} = 1;   
 set[1][4]=1; set[1] = {0,0,0,1,0,0,0}  
  
Fn[2] = max{c[4]+Fn[1], c[5]+Fn[0]} = 1  
 set[2][5]=1; set[2] = {0,0,0,0,1,0,0}  
  
Fn[3] = max{c[1]+Fn[0], c[4]+Fn[2], c[5]+Fn[1]} =   
 max{5 +0, 1 +1, 1 +1} = 5  
 set[3][1]=1; set[3] = {1,0,0,0,0,0,0}  
  
Fn[4] = max{c[1]+Fn[1], c[4]+Fn[3], c[5]+Fn[2], c[6]+Fn[0]} =   
 max{5 +1, 1 +5, 1 +1, 2 +0} = 6  
 set[4][1]=1; set[4] = {1,0,0,1,0,0,0}   
  
Fn[5] = max{c[1]+Fn[2],c[4]+Fn[4],c[5]+Fn[3],c[6]+Fn[1],c[7]+Fn[0],c[8]+Fn[0]} =   
 max{5 +1, 1 +6, 1 +5, 2 +1, 5 +0, 2 +0} = 6  
 set[5][1]=1; set[5] = {1,0,0,0,1,0,0}  
  
Fn[6] = max{c[1]+Fn[3],c[3]+Fn[0],c[4]+Fn[5],c[5]+Fn[4],c[6]+Fn[2],c[7]+Fn[2],c[8]+Fn[1]} =   
 max{5 +5, 9 +0, 1 +5, 1 +6, 2 +1, 5 +1, 2 +1} = 9  
 set[6][3]=1; set[6] = {0,0,1,0,0,0,0}  
  
Fn[7] = max{c[1]+Fn[4],c[2]+Fn[0],c[3]+Fn[1],c[4]+Fn[6],c[6]+Fn[5],c[7]+Fn[2],c[8]+Fn[2]} =   
 max{5 +6, 5 +0, 9 +1, 1 +9, 2 +6, 5 +1, 2 +1} = 10  
 set[7][3]=1; set[7] = {0,0,1,1,0,0,0}  
  
Fn[8] = max{c[1]+Fn[5],c[2]+Fn[1],c[3]+Fn[2],c[4]+Fn[7],c[6]+Fn[4],c[7]+Fn[3],c[8]+Fn[3]} =   
 max{5 +6, 5 +1, 9 +1, 1 +10, 2 +6, 5 +1, 2 +1} = 10  
 set[8][3]=1; set[8] = {0,0,1,0,1,0,0}

[Програма на С++ за решаване на задачата.](http://www.math.bas.bg/~nkirov/2009/PFC/knapsack.cpp)

Варианти на алгоритъма за решаване на задчата:

* с пресмятане на решението за всички стойности на капацитете
* без пресмятане на всички стойности на капацитета

Варианти на задачата за раницата:

* с определен брой предмети от всеки вид
* с неограничен брой предмети от всеки вид

**Преглед на различни "лесни" задачи**

XXII.Национална олимпиада по информатика - 2006   
**Първи кръг** (21 януари 2006) [тук](http://www.math.bas.bg/%7Enkirov/2006/INF297/l02/TemiABCDE.pdf) или от [infoman.musala.com](http://infoman.musala.com/index.html?211+2+%u0441%u044A%u0441%u0442%u0435%u0437%u0430%u043D%u0438%u044F%20%26gt%3B%20%u043D%u0430%u0446%u0438%u043E%u043D%u0430%u043B%u043D%u0438%20%26gt%3B%20%u0443%u0447%u0435%u043D%u0438%u0447%u0435%u0441%u043A%u0438%20%26gt%3B%20%u043D%u0430%u0446%u0438%u043E%u043D%u0430%u043B%u043D%u0430%20%u043E%u043B%u0438%u043C%u043F%u0438%u0430%u0434%u0430%20%u043F%u043E%20%u0438%u043D%u0444%u043E%u0440%u043C%u0430%u0442%u0438%u043A%u0430+http://infoman.musala.com/contests/noi/main.html).

* Да се направи преглед на задачите, като се даде идея за решаването на всяка задача.
* Да се намерят онези задачи, които могат да се решат по метода на динамичното оптимиране.
* Да се посочат подходящи средства за програмирането на всяка задача.
* **Да се напишат програми за решаване на 2 задачи по избор (2a и 2b).**

**3. Динамично оптимиране**

**Най-дълга обща подредица [8.2.6]**

Дадени са две редици (от числа или символи):

*X =* (*x*1, *x*2, ..., *xm*)и  *Y* = ( *y*1, *y*2, ..., *yn*)

Търси се най-дълга редица *Z =* (*z*1, *z*2, ..., *zk*), която е подредица на *X*и *Y* едновременно. *Z* е подредица на  *X*, ако *Z* може да бъде получена чрез премахване на (0 или няколко) членове на *X*.

Най-напред ще търсим само дължината на най-дългата обща подредица. Ще приложим метода на динамичното оптимиране, като *F*(*i, j*) е търсената дължина за първите *i* члена на редицата *X* и първите *j* члена на редицата *Y*. Очевидно *F*(*i,*0) = 0 за всяко *i* и *F*(0*, j*) = 0 за всяко *j*. *F*(*i, j*) = *F*(*i -*1*, j -*1) + 1 за *xi* = *yj*, а *F*(*i, j*) = max {*F*(*i -*1*, j*), *F*(*i, j -*1)} в противен случай. Намираме последователно стойностите на *F*(*i, j*) и последната намерена стойност *F*(*m,n*) е решението на задачата.

Намирането на една най-дълга подредица става по същия начин, като тръгваме от последния елемент и следим откъде идва максималната стойност.

// lsc.c  
#include <stdio.h>  
#include <string.h>  
#define MAXN 100  
#define MAX(a,b) (((a)>(b)) ? (a) : (b))  
  
char F[MAXN][MAXN];  
const char x[MAXN] = "acbcacbcaba";  
const char y[MAXN] = "abacacacababa";  
unsigned m,n;  
/\*  
 "acb cacbcaba " "a cbcac bcaba"  
 "a bacacacababa" "abacacacab aba"  
solution "a b cac caba " "a c cac b aba"  
\*/  
unsigned lcs\_len(void)  
{ unsigned i,j;  
 for (i=0; i<=m; i++) F[i][0]= 0;  
 for (j=0; j<=n; j++) F[0][j]= 0;  
 for (i=1; i<=m; i++)  
 for (j=1; j<=n; j++)  
 if (x[i-1] == y[j-1]) F[i][j]=F[i-1][j-1]+1;  
 else F[i][j] = MAX(F[i-1][j], F[i][j-1]);  
return F[m][n];  
}  
void print(unsigned i, unsigned j)  
{ if (i == 0 || j == 0) return;  
 if (x[i-1] == y[j-1])  
 { print(i-1,j-1);  
 printf("%c", x[i-1]);  
 }  
 else if (F[i][j] == F[i-1][j]) print(i-1,j);  
 else print(i,j-1);  
}  
int main()  
{ m = strlen(x);  
 n = strlen(y);  
 printf("%u\n", lcs\_len());  
 print(m,n);  
 return 0;  
}

|  |
| --- |
| 9  accacbab |

**Сравнение на символни низове**   
    Дадени са два символни низа *s*1 и *s*2. Задачата е от първия низ *s*1 да се получи втория низ *s*2, като се използват операциите:   
    - replace(i, x)- замества i-тия символ на *s*1 със символа x;   
    - insert(i, x) - вмъква символа x на позиция i в *s*1;   
    - delete(i) - изтрива i-тия символ на *s*1.   
За всяка от трите операции е зададена цена. Търсим крайна редица от операции за решаване на задачата, която е с минимална цена.   
    Нека *F*(*i, j*) е минималната цената за уеднаквяване на началните последователности (префикси)

*s*1[1], *s*1[2], ..., *s*1[i]  и *s*2[1], *s*2[2], ..., *s*2[j].

Индексите са взети по началните низове. Нека е *cr* цената за заместване на символ от *s*1 със символ от *s*2; *ci* е цената за вмъкване на символ в *s*1 и *cd* е цената за изтриване на символ от *s*1.

    Ако *s*1[i] = *s*2[j], то

*F*(*i, j*) = *F*(*i -* 1*, j -* 1),

в противен случай ще се вземе минималното от трите числа:

*F*(*i -* 1*, j -* 1) + *cr*   
*F*(*i,  j -* 1) + *ci*   
*F*(*i -* 1*,  j*) + *cd*

Остава да се пресметнат стойностите на *F*(*i,* 0) и *F*(0*, j*). *F*(*i,* 0) означава празен втори низ, следователно *F*(*i,* 0) = *i* *cd*. Аналогично при празен първи низ получаваме *F*(0*, j*) = *j* *ci*.

**// trans.c**   
**#include <stdio.h>**   
**#include <string.h>**   
**#define MAX 100**   
**#define min2(a,b) (((a)<(b)) ? (a) : (b))**   
**#define min3(a,b,c) (min2(min2(a,b),(c)))**

**#define COST\_DELETE 1**   
**#define COST\_INSERT 2**   
**#define COST\_REPLACE(i,j) ((s1[i] == s2[j]) ? 0 : 3)**

**unsigned F[MAX+1][MAX+1];**   
**unsigned n1, n2;**

**const char \*s1 = "\_abracadabra";**   
**const char \*s2 = "\_mabragabra";**

**unsigned editDistance(void)**   
**{ unsigned i,j;**   
**for (i=0; i<=n1; i++) F[i][0] = i\*COST\_DELETE;**   
**for (j=0; j<=n2; j++) F[0][j] = j\*COST\_INSERT;**

**for (i=1; i<=n1; i++)**   
**for (j=1; j<=n2; j++)**   
**F[i][j] = min3(F[i-1][j-1] + COST\_REPLACE(i,j),**   
**F[i  ][j-1] + COST\_INSERT,**   
**F[i-1][j  ] + COST\_DELETE);**   
**return F[n1][n2];**   
**}**

**void printEditOperations(unsigned i, unsigned j)**   
**{**   
**if (j==0) for (j=1; j<=i; j++) printf("DELETE(%u) ",j);**   
**else if (i==0) for (i=1; i<=j; i++) printf("INSERT(%u, %c) ",i, s2[i]);**   
**else if (i>0 && j>0)**   
**{ if ( F[i][j] == F[i-1][j-1] + COST\_REPLACE(i,j) )**   
**{ printEditOperations(i-1,j-1);**   
**if (COST\_REPLACE(i,j)>0) printf("REPLACE(%u, %c) ", i, s2[j]);**   
**}**   
**else if (F[i][j] == F[i][j-1] + COST\_INSERT)**   
**{ printEditOperations(i,j-1);**   
**printf("INSERT(%u, %c) ", i, s2[j]);**   
**}**   
**else if (F[i][j] == F[i-1][j] + COST\_DELETE)**   
**{ printEditOperations(i-1,j);**   
**printf("DELETE(%u) ", i);**   
**}**   
**}**   
**}**

**int main()**   
**{ n1 = strlen(s1)-1; n2 = strlen(s2)-1;**   
**printf("Distance: %u\n", editDistance());**   
**printEditOperations(n1,n2);**   
**printf("\n");**   
**return 0;**   
**}**

|  |
| --- |
| **Distance: 7**  **INSERT(1, m) DELETE(4) DELETE(5) REPLACE(7, g)** |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | **\_abracadabra** |
| **INSERT(1, m)** | 2 | **012345678901**  **\_abracadabra** | **\_mabracadabra** |
| **DELETE(4)** | 1 | **012345678901**  **\_abracadabra** | **\_mabrcadabra** |
| **DELETE(5)** | 1 | **012345678901**  **\_abracadabra** | **\_mabradabra** |
| **REPLACE(7, g)** | 3 | **012345678901**  **\_abracadabra** | **\_mabragabra** |
|  | **7** |  |  |

Southeastern European Regional Programming Contest   
Bucharest, Romania, October 18, 2003

Problem F   
**Common Subsequence**

Input File: F.IN   
Program Source File: F.PAS or F.C or F.CPP or F.JAVA

A subsequence of a given sequence is the given sequence with some elements (possible none) left out. Given a sequence *X* = *<x*1, *x*2, ..., *xm*> another sequence *Z* = <*z*1, *z*2, ..., *zk*> is a subsequence of *X* if there exists a strictly increasing sequence <*i*1, *i*2, ..., *ik*> of indices of *X* such that for all *j* = 1,2,...,*k*, *xij* = *zj*. For example, *Z* = <*a, b, f, c*> is a subsequence of *X* = <*a, b, c, f, b, c*> with index sequence <1, 2, 4, 6>. Given two sequences *X* and *Y* the problem is to find the length of the maximum-length common subsequence of *X* and *Y*.

The program input is from a text file. Each data set in the file contains two strings representing the given sequences. The sequences are separated by any number of white spaces. The input data are correct. For each set of data the program prints on the standard output the length of the maximum-length common subsequence from the beginning of a separate line.

Input   
**abcfbc         abfcab**   
**programming    contest**   
**abcd           mnp**

Output   
4   
2   
0