**ДИНАМИЧНО ОПТИМИРАНЕ**

*Динамическое программирование - это когда у насесть одна большая задача, которую непонятно как решать,и мы разбиваем ее на меньшие задачи, которые тоженепонятно как решать.(с)А.Кумок*

Динамичното оптимиране е техника на програмиране, при която се извършва запълване на таблица с резултатите на вече решени подзадачи с цел избягване на повторни пресмятания. В резултат на това времето за решаване на даден проблем значително се съкращава.

Съществуват няколко различни вида задачи, които се решават с помощта на динамично оптимиране.

**Зад. 1.** При зададено цяло число к пресметнете стойността на израза:

а/ б/ S = 1! + 2! + ... + k!

Наивният подход е да изчисляваме p!, за всяко p от 1 до k и след това да добавим частното към текущата стойност на сумата.

Като използваме равенството , можем да запазим вече пресметнатата стойност на произведението на числата до p и да го използваме за пресмятане на произведението до p+1.

#include <iostream>

using namespace std;

int main()

{

double S = 0, P = 1;

int n;

cin>>n;

for (int i=1; i<=n; i++)

{

P=P\*i;

S = S + 1/P;

}

cout<< S<<endl;

return 0;

}

**Зад. 2.** [**Числа на Фибоначи**](http://informatics.mccme.ru/moodle/mod/statements/view.php?id=649#1)

Редицата: 1, 1, 2, 3, 5, 8, 13, 21, ..., където всяко следващо число е сума от предходните две, се нарича редица на Фибоначи. Съставете програма, която въвежда от клавиатурата число n и намира n-тото число на Фибоначи 0 < n < 50.

Забележка: Редицата на Фибоначи започва с 0, която се счита за нулев член на редицата. Това число не променя изчисленията, затова е игнорирано.

Примерен вход: 7 Примерен изход: 13

**// I начин – рекурсия**

#include <iostream>

using namespace std;

int fib (int n)

{

if (n==1 || n==2) return 1;

else return fib(n-1)+fib(n-2);

}

int main()

{

int n;

cin>>n;

cout << fib(n) <<endl;

return 0;

}

Тази проста програма обаче води до прекалено много пресмятания. Даже извикването F(6) поражда сравнително голямо дърво от извиквания на функцията с по-малки стойности на аргумента

------------ F(6) ------------

/ \

---- F(5) --- F(4)

/ \ / \

F(4) F(3) F(3) F(2)

/ \ / \ / \ / \

F(3) F(2) F(2) F(1) F(2) F(1) F(1) F(0)

/ \ / \ / \ / \

F(2) F(1) F(1) F(0) F(1) F(0) F(1) F(0)

/ \

F(1) F(0)

Пресмятането на числата на Фибоначи може да се организира така, че да е необходимо линейно време по *n*, като запазваме всички пресметнати стойности в масив.

**// II начин – динамично със запазване на стойностите в масив**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | .. | a[n-2] | a[n-1] | a[n] |
|  | **1** | **1** | **2** | **3** | **5** | **.** |  |  |  |

#include <iostream>

using namespace std;

int main()

{

int n, a[50];

cin>>n;

a[1]=a[2]=1;

for (int i=3; i<=n; i++)

a[i]=a[i-1]+a[i-2];

cout << a[n] <<endl;

return 0;

}

Лесно се забелязва, че не е необходимо да запазваме всички пресметнати до текущия момент числа. Достатъчно е да имаме на разположение само двете предишни стойности.

**// ІІІ начин – динамично – запазват се само предходните две стойности**

#include <iostream>

using namespace std;

int main()

{

int n, f1=1, f2=1, fn;

cin>>n;

for (int i=3; i<=n; i++)

{

fn=f1+f2;

f1=f2;

f2=fn;

}

cout << fn <<endl;

return 0;

}

Можем да изчислим около 50 числа от редицата /на 32 битов компютър/. Следващите вече излизат от границите на целия тип. Можем да изведем около 100 числа, ако променим типа на променливите от int в long long или unsigned long long.

**Зад. 3.** [**Последна цифра на n-тото число на Фибоначи**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=649&chapterid=842#1)

Да се намери и изведе последната цифра на n-тото число на Фибоначи, където 0 < n < 1000.

Упътване: Ясно е, че ако вместо цялото число, съхраняваме само последната цифра, това няма да промени резултата, който търсим. Затова a[i]:=(a[i-1]+a[i-2])%10 или fn = (f1+f2)%10. .

**Зад. 4.** Редици от 0 и 1 с дължина N

Да се намери броят на редиците с дължина N, състоящи се от нули и единици.

|  |  |  |  |
| --- | --- | --- | --- |
| N = 1 | 0  1 | Br = 2 | Редиците с дължина 1 са две. |
| N = 2 | 00, 01  10, 11 | Br = 4 | След всяка от предходните две редици поставяме 0 или 1 и така получаваме нови 4 редици. |
| N = 3 | 000, 001  010, 011  100, 101  110, 111 | Br = 8 | От всяка от предходните редици получаваме две нови редици, затова общия брой e 8. |

Oт всяка редица с дължина *k*–1 се получават две нови редици – едната с присъединяване на 0, а другата с присъединяване на 1. Затова, ако знаем броя *Bk*–1 на редиците с дължина *k*–1, тогава броят на редиците с дължина *k* е равен на *Bk* = 2\**Bk*–1.

Можем да пазим стойностите в масив, на който първия елемент е 2, а всеки следващо елемент се получава чрез удвояване на предходния:

Br[0]=2;

for (i=1;i<N;i++) Br[i]=2\*Br[i-1];

Тъй като индексите на масива са от 0, елементът Br[n-1] съдържа търсения брой.

Използването на масив в случая не е наложително, понеже за пресмятането на всяка следваща стойност се използва само непосредствено предишната.

Br = 2;

for (i=1; i <N; i++) Br = Br\*2;

Броят може да се пресметне и по формулата Br = 2N по следния начин:

Br = 1;

for (i=1; i <=N; i++) Br = Br\*2;

**Зад. 5.** [**Редици от 0 и 1 с дължина N без повтарящи се единици**](http://informatics.mccme.ru/moodle/mod/statements/view.php?id=654#1)

Да се намери броят на редиците с дължина N, състоящи се от нули и единици и такива, че в тези редици не се срещат никъде две единици непосредствено разположени една до друга.

|  |  |  |  |
| --- | --- | --- | --- |
| N = 1 | 0  1 | Br = 2 | Редиците с дължина 1 са две. |
| N = 2 | 00, 01  10 | Br = 3 | След 0 можем да поставим 0 или 1 и така получаваме нови 2 редици.  След 1 можем да поставим само 0. |
| N = 3 | 000, 001  010  100, 101 | Br = 5 |  |
| N = 4 | 000, 0001  0010  0100, 0101  1000, 1001  1010 | Br = 8 |  |

Забелязваме, че за броя на редиците получаваме числата на Фибоначи, които вече сме пресмятали.

Друг начин за намиране на зависимост е следният:

След всички редици с дължина к-1 можем да поставим 0 и получаваме редица с дължина к.

Единица можем да поставим след редица с дължина к-1, завършваща на 0, а броят на тези редици е равен на броя на редиците с дължина к-2, защото след всяка от тях може да се постави 0.

Така получаваме зависимостта, *Bk* = *Bk*–1+*Bk*–2, което отново е формулата на Фибоначи.

Забележка: Задачата на сайта е с ограничение до 100 и изисква реализация с дълги числа, иначе тестовете не минават.

**Зад. 6.** [**Редици от 0 и 1 без повтарящи се три единици**](http://informatics.mccme.ru/moodle/mod/statements/view.php?id=654#1)

Да се намери броят на редиците с дължина N, състоящи се от нули и единици и такива, че в тези редици не се срещат никъде **три** единици непосредствено разположени една до друга.

Упътване: Разсъжденията са аналогични на предходната задача, зависимостта е: *Bk* = *Bk*–1+*Bk*–2 +*Bk*–3.

**Зад. 7.** [**Взривоопасност**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=654&chapterid=913#1)

При преработка на радиоактивни материали се образуват два вида отпадъци – особено опасни (тип A) и безопасни (тип B). За съхранението им се използват еднакви контейнери. След това контейнерите се подреждат в специално помещение за отпадъци един върху друг. Един стълб с вертикално подредени контейнери се счита за взривоопасен, ако в него има един до друг повече от един контейнер с отпадък от тип A. За зададен брой контейнери N в стълб определете броя на безопасните стълбове.

Вход: Едно цяло число 1 <= N <= 20 – брой контейнери в стълб.

Изход: Едно цяло число – броя на безопасните варианти на подредба на контейнерите в стълба.

Пример: Вход: 2 Изход: 3

**Упътване:** Обозначаваме с 0 и 1 отпадъците съответно от безопасния и опасния вид. Търсим редица с дължина N, в която не трябва да има две единици една до друга. Това е всъщност задача 5.

**Зад. 8.** [**Взривоопасност 2**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=654&chapterid=914#1)

При преработка на радиоактивни материали се образуват три вида отпадъци – особено опасни (тип A), с малка степен на опасност (тип В) и безопасни (тип С). За съхранението им се използват еднакви контейнери. След това контейнерите се подреждат в специално помещение за отпадъци един върху друг. Един стълб с вертикално подредени контейнери се счита за взривоопасен, ако в него има един до друг повече от един контейнер с отпадък от тип A. За зададен брой контейнери N в стълб определете броя на безопасните стълбове.

Вход: Едно цяло число 1 <= N <= 20 – брой контейнери в стълб.

Изход: Едно цяло число – броя на безопасните варианти на подредба на контейнерите в стълба.

Пример: Вход: 2 Изход: 8

**Зад. 9.** [**Топка на стълбата**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=649&chapterid=203#1)

На върха на стълба с N стъпала е поставена топка, която започва да пада по стъпалата надолу. Топката пада на следващото стъпало, през 1 стъпало или през 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | ball.gif |  |
|  |  |  |  |  | a[0] = 1 |
|  |  |  |  |  | a[1] = 2 |
|  |  |  |  |  | a[2] = 4 |
|  |  |  |  |  | a[3] = 7 |
|  |  |  |  |  | a[4] |

стъпала, т.е. ако топката е на осмо стъпало, може да падне на седмо, шесто или пето стъпало. Определете броя на възможните маршрути на топката от върха до земята.

Вход: едно цяло число 0 < N < 31

Изход: едно цяло число – броя на маршрутите

Пример: Вход: 4 Изход: 7

Използваме масив с n елемента.

Елементът a[0] = 1 съдържа броя начини по които топката да слезе едно стъпало надолу.

Елементът a[1] = 2 съдържа броя начини, по които топката слиза две стъпала.

Елементът a[2] = 4 съдържа броя начини, по които топката слиза три стъпала.

По условие до едно стъпало се достига от трите горни стъпала, затова за всички елементи от a[3] нататък броя начини се получава като сума на начините от предходните три стъпала, т.е. a[i] = a[i-1] + a[i-2] + a[i-3], за всяко i от 3 до N-1.

Ако стъпалата са 5, то а[4] ще съдържа броя начини, по които топката слиза долу. Следователно, ако стълбата е с N стъпала, трябва да намерим и изведем a[N-1].

#include <iostream>

using namespace std;

int main()

{

int n, i, a[31];

cin >>n;

a[0]=1; a[1]=2; a[2]=4;

for (i=3; i<n; i++)

a[i] = a[i-1]+a[i-2]+a[i-3];

cout << a[n-1]<< endl;

return 0;

}

**Зад. 10.** [**Платена стълба**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=654&chapterid=915#1)

Момче се движи по стълба, за всяка стъпало на която трябва да плати определена такса. Момчето може да премине на следващата или на по-следващата стълба. Каква минимална сума трябва да плати момчето, за да стигне до края на стълбата.

Вход: На първи ред е дадено едно цяло число *N*http://informatics.mccme.ru/moodle/lib/jsMath/fonts/cmsy10/alpha/120/char14.png100 – брой стъпала. На втория ред са дадени N естествени числа, *N*http://informatics.mccme.ru/moodle/lib/jsMath/fonts/cmsy10/alpha/120/char14.png100 – такса на всяко стъпало.

Изход: Едно число – най-малката възможна сума за преминаване по стълбата.

Пример:

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 1 2 3 | 4 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://www.freecoloring.info/img/new/child%20(10).png |  |  |  |  |  |  |
|  |  |  |  |  | a[4] |
|  |  |  |  |  | a[3] |
|  |  | 3 |  |  | a[2] = min (a [0], a[1]) + a[2] = min (1, 2) + 3 = 1 + 3 = 4 |
|  | 2 |  |  |  | a[1] = 2 |
| 1 |  |  |  |  | a[0] = 1 |

Задачата прилича на предходната. Използваме масив с n елемента – за всяко стъпало ще запазваме сумата, която трябва да се плати за достигане до това стъпало.

Елементът a[0] = 1 е стойността на първото стъпало. До него стигаме директно.

Елементът a[1] = 2 е стойността на второто стъпало стъпало. До него стигаме директно или като преминем през стъпало 1, но тогава сумата се увеличава, а ние търсим минималната.

До следващото стъпало може да стигнем или от предходното или от по-предното. Например до стъпало 3 достигаме от 2 или от 1. От двата варианта избираме този, който е с по-малка стойност и добавяме стойността на текущото стъпало. Така за елемента a[i] получаваме: a[i] = min (a[i-1], a[i-2]) + a[i];

Тъй като индексите на масива са от 0, в елемента a[n-1] ще се съдържа търсената сума до n-тото стъпало на стълбата.

#include <iostream>

using namespace std;

int main()

{

int n, i, a[100];

cin >>n;

for (i = 0; i<n; i++) cin>>a[i];

for (i=2; i<n; i++)

a[i] = min (a[i-1], a[i-2]) + a[i];

cout << a[n-1]<< endl;

return 0;

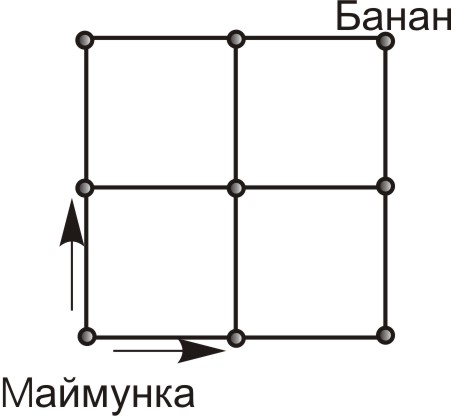
}

**Зад. 11. МАЙМУНКА /НОИ 2006, І кръг, Задача B1, Стоян Капралов/**

В правоъгълна област с размери *М* x *N* (*M* ≤ 10, *N* ≤ 10) равномерно са засадени палми (във всяка точка с целочислени координати спрямо югозападния ъгъл на областта, включително и по нейния контур, има точно по една палма). Върху палмата в югозападния ъгъл на областта се намира маймунка. За съжаление единствения узрял банан е на палмата в североизточния ъгъл на градината. Маймунката може да скача само на съседната палма в източна посока или на съседната палма в северна посока, без да се връща назад.

Напишете програма **MONKEY**, която пресмята по колко различни начина маймунката може да стигне до банана.

От стандартния вход се въвеждат числата *M* и *N*, разделени с един интервал.

На стандартния изход да се изведе търсеното число.

ПРИМЕР: Вход: 2 2 Изход: 6

Упътване:

Представяме областта чрез двумерен масив. Започвайки от началото запълваме масива по редове със стойности – броя начини по които може да се стигне до клетката. За всяка клетка броя пътища до нея се намира като сума на пътищата до долната клетка и клетката вляво. За удобство запълваме І ред и І стълб с 0 и след това за останалите клетки прилагаме горната зависимост.

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

int m, n;

cin >> m >> n;

int a[11][11];

//попълване на таблицата със стойности

a[0][0]=0;

for (int i=1; i<=m; i++) a[i][0]=1;

for (int i=1; i<=n; i++) a[0][i]=1;

for (int i=1; i<=m; i++)

for (int j=1; j<=n; j++)

a[i][j]=a[i-1][j]+a[i][j-1];

cout<< ”Брой начини: ” << a[m][n]<<endl;

return 0;

}

**Зад. 12.** [**Триъгълник на Паскал**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=943#1)

Триъгълникът на Паскал има следния вид: първи ред се състои само от едно число – единица. Всеки следващ ред съдържа едно число повече от предходния ред, като първото и последното число са единици, а всички останали се изчисляват като сума на числата на предходния ред над него и вляво от него.

Вход: цяло число N (0<=N<=30)

Изход: N-редов триъгълник на Паскал

Пример:

|  |  |
| --- | --- |
| Вход | Изход |
| 5 | 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 |

#include <iostream>

using namespace std;

const int MAXN=50;

int t[MAXN][MAXN];

int main()

{

int i,j, N;

cin >>N;

for(i=0;i<N;i++)

{t[i][0]=1;t[i][i]=1;}

for(i=1;i<N;i++)

for(j=1;j<i;j++)

t[i][j]=t[i-1][j-1]+t[i-1][j];

for(i=0;i<N;i++)

{for(j=0;j<=i;j++) cout << t[i][j] << ' ';

cout << endl;}

return 0;

}

**Зад. 13.** [**Най-евтин път**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=944#1)

Дадена е правоъгълна таблица NxM с цели числа. Играчът се намира в горната лява клетка на таблицата. На един ход може да премине в съседната клетка вдясно или в клетката надолу. При преминаването през клетка играчът трябва да остави такса – толкова, колкото е числото в клетката /включително в първата и в последната клетка/. Намерете минималната такса, която ще остави играчът, преминавайки от горна лява клетка до клетката най-долу вдясно.

Вход: На първи ред се въвеждат се числа *N* и *M* – размери на таблицата, 1<=N <=20, 1<=M<=20. На следващите N реда се въвеждат M числа – такса за всяка клетка от таблицата. Числата са от 1 до 100.

Изход: Едно число – минималното сума, които трябва да остави като такси, за да премине от горен ляв до долен десен ъгъл.

Пример

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 4 1 1 1 1 5 2 2 100 9 4 2 1 | 8 |

Упътване:

Попълваме нова таблица със стойността на всяка клетка – таксата, която трябва да остави играча, за да достигне до тази клетка. Във всяка клетка на таблицата можем да стигнем от клетката вляво или от клетката горе. За да бъде сумата минимална, ще изберем по-малката от двете стойности отгоре и вляво и ще я добавим към стойността на текущата клетка. Затова стойността на клетката b[i][j] ще бъде min (a[i][j-1], a[i-1][j]) + a[i][j].

Можем да работим с един масив и да го запълваме по време на въвеждане или да използваме втори масив.

Стойността която търсим е в долната дясна клетка.

За удобство за изчисленията може да запълним първия ред и първата колона на таблицата с нулеви елементи.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | 2 | 7 | 5 | 6 |
| 4 | 5 | 2 | 6 | 5 |
| 1 | 3 | 3 | 10 | 4 |
| 1 | 2 | 7 | 1 | 2 |

**Зад.14. Движение на североизток**

Задачата е подобна на горната, но движението е в посока горе-дясно и се търси максимална сума.

Дадена е таблица с числа. Движим се само нагоре и надясно. Трябва да стигнем до горна дясна клетка с максимална сума.

Упътване:

За всяка клетка помним максималната сума, която се получава достигайки клетката. Започваме от началото и за всяка клетка b[i][j] вземаме максималната от клетките вляво и отдолу + стойността в самата клетка от първата таблица.

За да спестим if добавяме 0 отдолу и вдясно или І стълб /вляво/ и І ред /отдолу/ запълваме автоматично от първата таблица.

**Зад. 15.** [**Пешка в дама**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=945#1)

На шахматна дъска (8х8) има една бяла пешка. Бялата пешка се движи по диагонал, една клетка нагоре-надясно или нагоре-наляво. Пешката се превръща в царица когато попадне на най-горния хоризонтал /когато стигне края на полето/. По колко различни начина тя може да се превърне в царица?

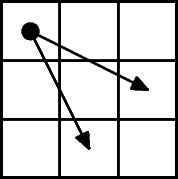
Вход: две числа от 1 до 8 – номер на стълб и ред, където е поставена пешката

Изход: едно число – броя начини до производство на царица

Примери:

|  |  |
| --- | --- |
| Вход | Изход |
| 3 7 | 2 |
| 1 8 | 1 |
| 3 6 | 4 |

**Зад. 16.** [**Ход на коня**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=946#1)

Дадена е правоъгълна дъска *N* × *M* (*N* реда и *M* стълба). В горния ляв ъгъл се намира шахматен кон, който трябва да се премести в долния десен ъгъл на дъската. Конят се движи само две клетки надолу и една вдясно, или две клетки надясно и една клета надолу, както е показано на рисунката.

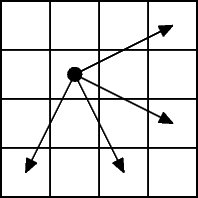
Да се определи колко различни маршрута, водещи от горния ляв ъгъл до долния десен ъгъл, съществуват.

Вход: На един ред са въведени две естествени числа *N* и *M* (1 ≤ *N*,*M*≤ 50)

Изход: Брой начини за преминаване от горен ляв до долен десен ъгъл на дъската с ход на коня надясно и надолу.

Пример

|  |  |
| --- | --- |
| Вход | Изход |
| 3 2 | 1 |
| 31 34 | 293930 |
| 1 1 | 1 |

**Зад. 17.** [**Ход на коня – 2**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=2962#1)

Дадена е правоъгълна дъска *N* × *M* (*N* реда и *M* стълба). В горния ляв ъгъл се намира шахматен кон, който трябва да се премести в долния десен ъгъл на дъската. Конят се движи както е показано на рисунката.

Да се определи колко различни маршрута за придвижване от горен ляв до долен десен ъгъл съществуват.

Вход: На един ред са въведени две естествени числа *N* и *M* (1 ≤ *N*,*M*≤ 15).

Изход: Брой начини за преминаване от горен ляв в долен десен ъгъл по указания начин.

Пример**:**

|  |  |
| --- | --- |
| Вход | Изход |
| 4 4 | 2 |
| 15 14 | 7884330 |

**Зад. 18.** [**Опит за бягство**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=2964#1)

Затворник се опитва да избяга от замък, който се състои от N×M квадратни стаи, разположени във вид на правоъгълник с размери NxM. Между всеки две стаи има врата, като на някои стаи вратите за затворени и не може да се влезе в тях. В началото затворника се намира в стаята в горния ляв ъгъл и за да се спаси, той трябва да стигне до стаята в долния десен ъгъл. Той не разполага с много време. Той може да премине през не повече от N+M-1 стаи в своя път, като може да се движи само направо или надолу. Определете броя на маршрутите, които го водят към изхода.

Вход: На първия ред се въвеждат 2 цели числа n и m – брой редове и стълбове, като 0< n <= 1000 и 0 < m <= 1000. На следващите n реда се въвеждат по m числа – 0 /за затворена стая/ или 1 /за отворена стая/.

Изход**:** Програмата трябва да отпечата броя на маршрутите, водещи затворника към изход и преминаващи през M+N-1 стаи или думата Impossible, ако такъв маршрут не съществува. Броят на маршрутите не надвишава 2 000 000 000.

Пример:

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 5 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 | 3 |
| 3 5 1 0 1 1 1 1 0 1 0 1 1 1 1 0 1 | Impossible |

**Зад. 19.** [**Костенурка**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=2965#1)

В горния ляв ъгъл на правоъгълна таблица с размери N×M се намира костенурка. Във всяка клетка на таблицата е записано някакво число. Костенурката може да се движи надясно или надолу, като маршрута на костенурката завършва в долния десен ъгъл на таблицата. Изчислете сумата на числата, записани в клетките, през които преминава костенурката, включително началната и крайната клетка. Намерете най-голямата възможна стойност на тази сума.

Вход: На първия ред се въвеждат две естествени числа N и M, определящи размера на таблицата, като 0 < N <= 100, 0 < M <= 100. На следващите N реда са въведени по M числа, разделени с интервал, всяко от които приема стойност от 0 до 100.

Изход: Максималната възможна стойност на маршрута на костенурката.

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 4 1 1 2 1 2 2 1 1 2 1 2 1 | 9 |

**Зад. 20.** [**Маршрут на костенурката**](http://informatics.mccme.ru/moodle/mod/statements/view3.php?id=656&chapterid=2966#1)

В горния ляв ъгъл на правоъгълна таблица с размери N×M се намира костенурка. Във всяка клетка на таблицата е записано някакво число. Костенурката може да се движи надясно или надолу, като маршрута на костенурката завършва в долния десен ъгъл на таблицата. Изчислете сумата на числата, записани в клетките, през които преминава костенурката, включително началната и крайната клетка. Намерете най-голямата възможна стойност на тази сума и маршрутът, при който се достига тази сума.

Вход: На първия ред се въвеждат две естествени числа N и M, определящи размера на таблицата, като 0 < N <= 100, 0 < M <= 100. На следващите N реда са въведени по M числа, разделени с интервал, всяко от които приема стойност от 0 до 100.

Изход: На първия ред изведете максималната възможна сума, а на втория ред – маршрута, при който се достига тази сума. Маршрутът да се изведе като последователност от N-1 букви D, означаващи придвижване надолу, и М-1 букви R, означаващи придвижване надясно. Ако тези последователности са повече от 1, изведете само една от тях.

Пример:

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3 4 1 1 2 1 2 2 1 1 2 1 2 1 | 9 D R D R R |

**Задачите, на които са дадени хипервръзки, могат да се тестват на сайта.**