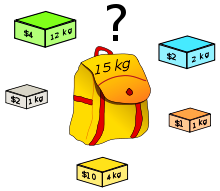
PROGRAMACIÓN DINÁMICA

PRÁCTICA 3



AUTORES:

Manuel Ariza Ortiz

Javier Bueno López

Sergio Cruz Pérez

Carlos Enríquez López

**Índice:**

1. Ejercicio 1.Producto máximo con suma fija.

-Enunciado.

-Solución.

-Salida.

-Eficiencia.

1. Ejercicio 2.La conexión de “El Pedregal”.

-Enunciado.

-Solución.

1. Ejercicio 3.Regalos por la fama.

-Enunciado.

-Solución.

-Ejemplos.

1. Ejercicio 4.El problema del turista en Manhattan.

-Enunciado.

-Enfoque.

-Solución.

-Eficiencia.

1. Apéndice de códigos.

**Ejercicio 1.**Producto máximo con suma fija

**Enunciado**

Dado M, un número natural no negativo, ¿cuál es el mayor valor que podemos conseguir multiplicando n números naturales que sumen M?

Es decir, deseamos lo siguiente:

-maximizar x1\*x2\*...\*xn

-sujeto a x1+x2+...+xn=M

Para aplicar PD consideramos la siguiente función MAX(k,C) = Máximo valor de x1 \* x2 \*…\* xk sujeto a x1 + x2 +…+xk = C, siendo x1, x2,…, xk números naturales.

**Solución**

Podemos aplicar programación dinámica ya que:

1. Podemos dividir el problema en subproblemas más pequeños.
2. Podemos resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.
3. Podemos usar estas soluciones óptimas para construir una solución óptima al problema original.

Además, se cumple el principio de optimalidad de Bellman que dicta que «dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima».

Procedemos a explicar porque se cumple el principio de optimalidad de Bellman en este problema:

Si partimos de un número natural x1, entonces x2,...,xn debe de ser una sucesión optimal para MAX(n,M).

Si no lo fuera, sería porque habría otra sucesión z1,z2,z3,...,zn tal que

∑1<=i<=n xi =M y ℼ1<=i<=n zi > ℼ1<=i<=n xi

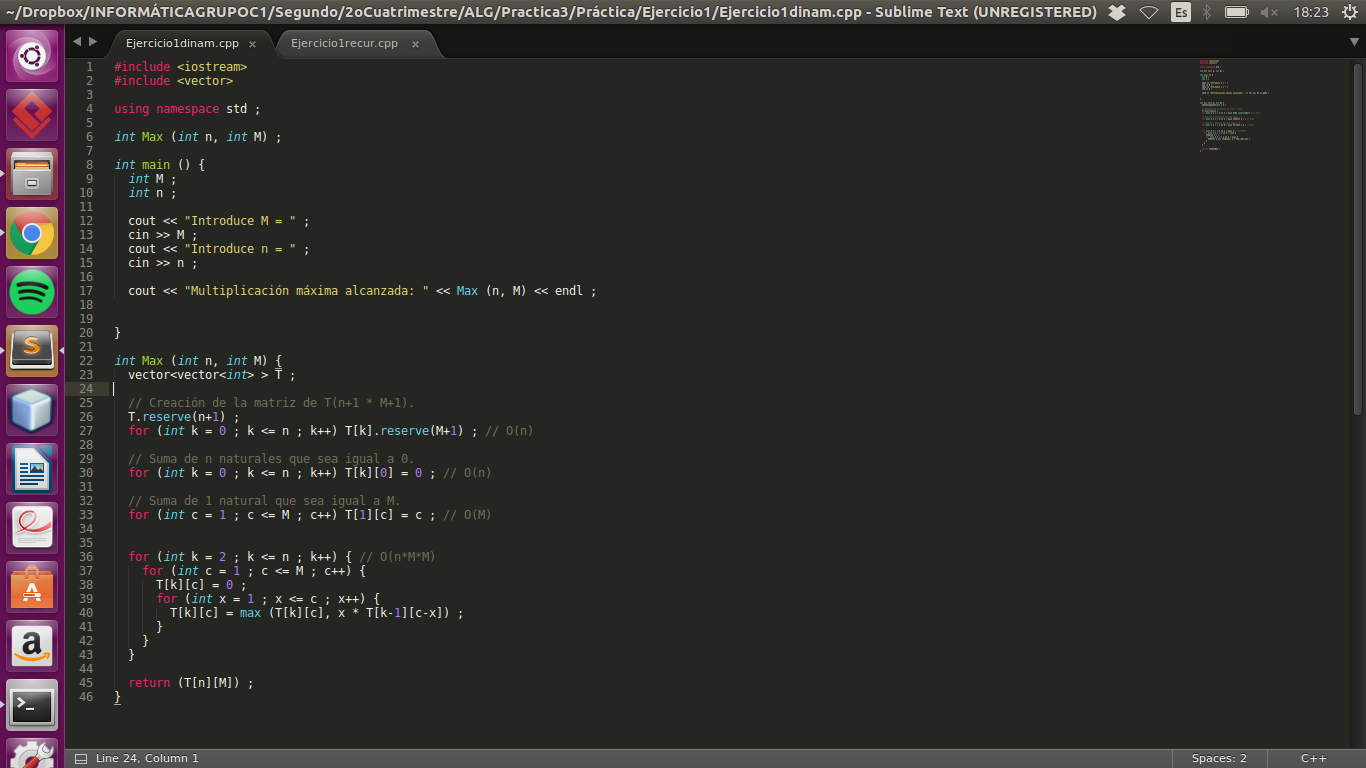
y por lo tanto la sucesión z1,z2,...zn es una sucesión para el problema de partida con mayor valor para el producto , lo que es contradictorio.

Por lo tanto puede aplicarse el principio de optimalidad de Bellman, ya que éste se cumple.

En definitiva nos damos cuenta que la columna inmediatamente a la izquierda en la matriz T, contiene los valores óptimos a buscar para un conjunto en un elemento menor.

Se adjunta en el apéndice de códigos el algoritmo que calcula lo pedido de manera recursiva,

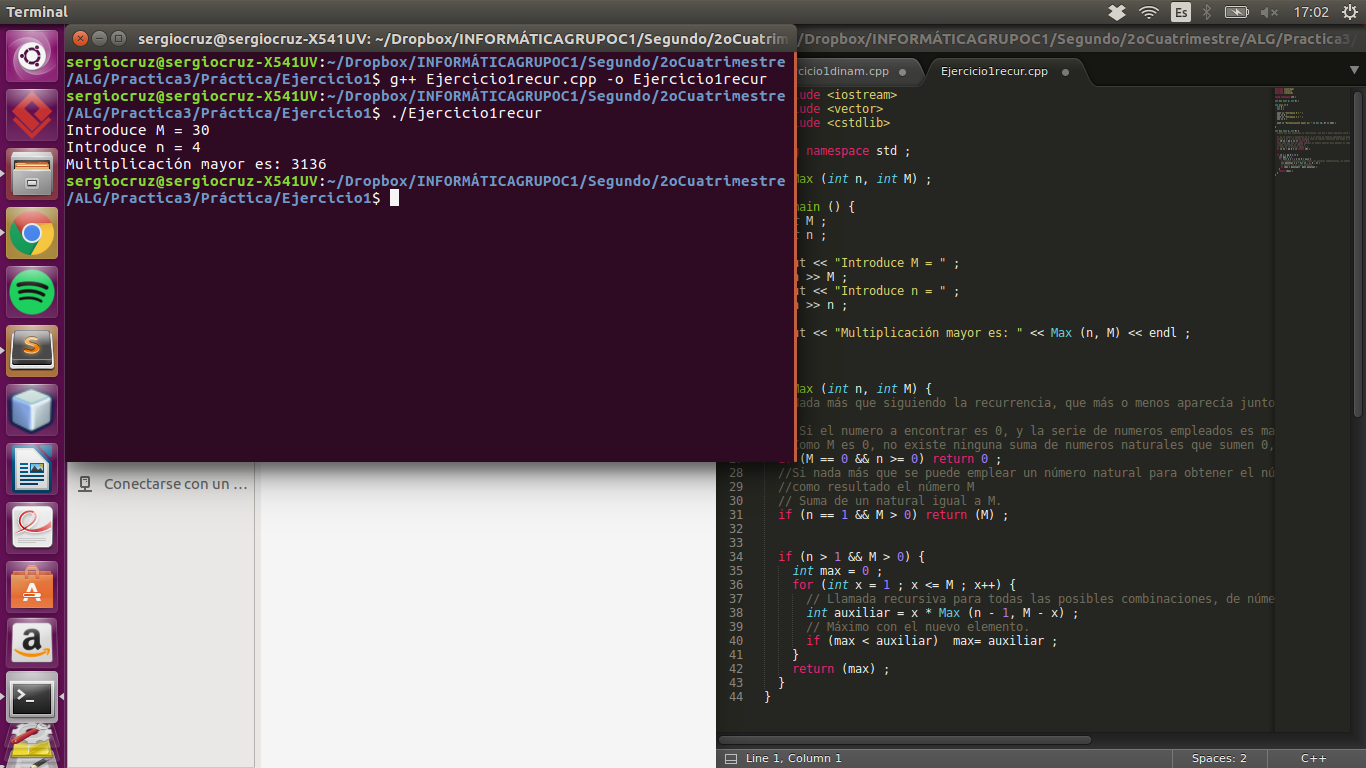
Dicho algoritmo actúa como el pseudocódigo que aparece en las diapositivas de la práctica



Lo que hacemos en la función MAX, que es donde se lleva a cabo todo el algoritmo, es crear de primeras la matriz de n+1 filas(representan los números naturales que queremos emplear) y M+1 columnas, es decir, el número que queremos calcular más uno.

Posteriormente, inicializamos que la suma de los n naturales sea igual a 0 y también inicializamos la suma de 1 natural a M. Una vez concluido esto, ya procedemos mediante el bucle anidado de tres for, al cálculo del producto de los n naturales que sumen M, y que además sea máximo, que será el que devuelva la función como resultado.

**Salida que proporciona el programa,.**

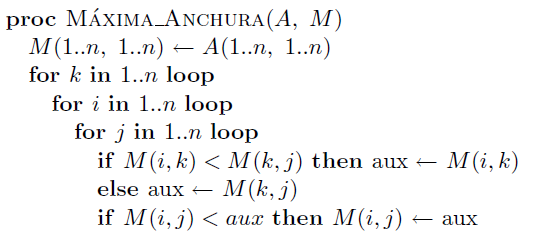
Para mostrar el correcto funcionamiento:

**Eficiencia**

Por último, para explicar la eficiencia, que es O(nM), basta con observar que para realizar el problema hace falta una matriz de tamaño n \* M, y los bucles anidados realizan exactamente ese número de iteraciones.

**Ejercicio 2.**La conexión de “El Pedregal”

**Enunciado:**En un archipiélago, con multitud de pequeñas islas cercanas, hay puentes que unen ciertos pares de islas entre sí. Para cada puente (que puede ser de dirección única), además de saber la isla de origen y la isla de destino, se conoce su anchura (número entero mayor que 0).  
  
La anchura de un camino, formado por una sucesión de puentes, es la anchura mínima de las anchuras de todos los puentes que lo forman.  
  
Para cada par de islas se desea saber cuál es el camino de anchura máxima que las une (siempre que exista alguno).  
  
Diseñe un algoritmo con la técnica de Programación Dinámica que resuelva este problema. Analice su algoritmo.  
  
**Solución:**En este problema usamos programación dinámica ya que cumple el principio de optimalidad de Bellman, el cual comprobamos de la siguiente manera:  
Sea i,i1,...,ik, j el camino de anchura máxima desde i hasta j.  
Comenzamos con el vértice inicial i, se ha tomado la decisión de ir al vértice i1.  
Como resultado, ahora el estado del problema está definido por el vértice i1, y lo que se necesita es encontrar un camino desde i1 hasta j.  
Está claro que la sucesión i1,i2...,ik,j debe constituir un camino de anchura máxima entre i1 y j, Sí no:  
Sea i1,r1,...rq,j un camino más ancho entre i1 y j entonces i,r1,r2,...,rq,j es un camino entre i y j que es más ancho que el camino i,i1,i2,...,ik,j.  
Como eso es una contradicción, se verifica el POB por lo que puede aplicarse a este problema.  
  
  
Denominaremos A(i,j) a la anchura del puente que va de la isla i a la j.  
En caso de que no haya un puente entre una isla y otra A(i,j) = 0 y para ir de una isla a ella misma A(i,i) = -1 que representa el infinito.  
  
Definimos, de manera recursiva, la siguiente función:  
Max Anchura(i, j, k) = máxima anchura de los caminos que van de la isla i a la isla j, pudiendo pasar por las islas {1,...,k}  
  
Max Anchura(i, j, 0) = A(i, j)  
Max Anchura(i, j, k) = Max{Max Anchura(i, j, k − 1), Min{Max Anchura(i, k, k − 1),   
Max Anchura(k, j, k − 1)}} si k>0  
  
Así podemos diseñar un algoritmo como el siguiente, que calcula los valores de una tabla M(1..n, 1..n), de manera que al terminar:  
M(i, j) = Max Anchura(i, j, n) ∀i, j  
  
  
  
Para ello se define el siguiente pseudocódigo:

  
  
Viendo el pseudocódigo se observa que el algoritmo tiene una eficiencia de 0(n³).  
El cual implementamos en C++ (El código se encuentra al final).  
Un ejemplo puede ser:  


De la isla 0 a 3 la anchura máxima será 3, ya que  
para k = 2:  
0-1-2-3 tenemos anchura 1.  
Para k = 1  
0-1-3 tenemos anchura 2  
Como ningún camino es más ancho que para A(0,3) directamente:  
0-3 tenemos anchura 3.  
La anchura máxima que buscamos entre 0 y 3 es 3.  
Si por ejemplo hubiéramos tenido el caso de:  
  
Vemos cómo A(0,3) directamente la anchura es 3.  
Pero si pasamos por 0-1-2-3 (k = 2) la anchura máxima es 4, por tanto en M cambiamos la posición en la que antes había un 3 por un 4 como nueva anchura máxima.

**Ejercicio 3.**Regalos por la fama

**Enunciado:**

Mª Gabriela y Eduardo Fernando han recibido un montón de regalos por su estupendo trabajo en una serie de televisión de reconocida fama. Cada regalo viene en una caja destinada a ambos. Como no tienen suficiente tiempo para desempaquetar y mirar que es cada cosa, han decidido utilizar el siguiente criterio para repartirse los regalos:

Cada uno debe quedarse con la misma cantidad de peso

Para ello cuentan con los pesos de cada una de las cajas P1,..., Pn (números enteros positivos).

Al cabo de un buen rato, todavía no han conseguido hacer el reparto según ese criterio.

Diseñar un algoritmo, utilizando la técnica de Programación Dinámica, que resuelva el problema de esta pareja y determine una forma de reparto de los regalos, si es que es posible y analizar el algoritmo.

**Solución:**

En este apartado se adjunta la solución proporcionada junto con una explicación de la misma.

Sea D = P1 + . . . + Pn la suma de los pesos. Asumimos que D es par, puesto que en otro caso no es posible ese reparto paritario. El problema consiste en determinar si existe un subconjunto de las cajas tal que la suma de sus pesos sea D .

Definimos la función booleana siguiente para aplicar PD, S(m, k) = True si existe un subconjunto de {P1, . . . , Pk} que sume m. En caso contrario S(m, k) = False

S(0, k) = True ∀k ≥ 0

Refleja que para la primera situación del sistema en la que no hemos repartido ningún peso (m=0) da igual el subconjunto de paquetes que tengamos ya que la función será verdadera.

S(m, 0) = False si m > 0

Refleja que si no tenemos paquetes y tenemos que repartir un peso mayor que cero no podemos hacerlo por lo que la función se evalúa como falsa

S(m, k) = S(m, k − 1) ∨ S(m − Pk, k − 1) si m ≥ Pk y k > 0

Refleja que si para un peso y un paquete determinados el restarle el peso del paquete al peso actual da una combinación válida de paquetes la inclusión de dicho paquete junto con su peso en el subconjunto también da una solución válida.

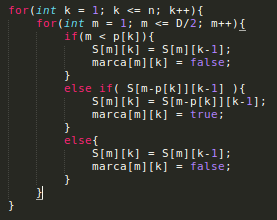
En caso de que el subconjunto sin el paquete actual no diera una solución válida su inclusión en el subconjunto tampoco lo haría.

S(m, k) = S(m, k − 1) si m < Pk y k > 0

Refleja como si el peso del paquete a incluir en el conjunto es mayor que el peso a completar el valor de S es el mismo que el del subconjunto sin dicho paquete.

Por lo tanto necesitamos el valor S(D/2, n) para saber si hay un subconjunto de paquetes que reúna exactamente la mitad del peso total de todos los paquetes.

A partir del pseudo código proporcionado para este fin hemos generado una implementación en C++ (revisar apéndice de códigos) que calcula el valor que necesitamos.



La eficiencia del algoritmo es de O(nD). La explicación está en los únicos dos bucles ‘for’ anidados ya que el externo tiene una eficiencia de O(n) mientras que en el interno la eficiencia es de O(D). Por la regla del producto tenemos una eficiencia resultante de O(nD).

El problema es resoluble mediante programación dinámica porque reúne una serie de características.

* La naturaleza N-etápica del problema bajo consideración, para caracterizar la estructura de una solución optimal.
* Una condición necesaria para la aplicación del Principio de Optimalidad de Bellman (POB) ya mencionada. S(0, k) = True ∀k ≥ 0 y S(m, 0) = False si m > 0
* Una ecuación de recurrencia que represente la forma de lograr etapa por etapa la solución optimal. S(m, k) = S(m, k − 1) ∨ S(m − Pk, k − 1) si m ≥ Pk y k > 0
* Por último, encontrar la solución optimal resolviendo problemas encajados para construir la solución.

**Ejemplos:**

Para representar lo que hace el algoritmo utilizaremos un subconjunto de 5 paquetes con unos pesos respectivos (P1, P2 … P5) de 2, 2, 1, 5, 2 y D = 12.

El algoritmo calcula el valor de S el primer paquete junto con todos los pesos posibles desde 0 hasta D/2. Realiza la misma operación para todos los paquetes.

A la vez que evalúa para S a cada pareja de paquete y peso también comprueba para cada paquete de una pareja si este está incluido está incluido en el subconjunto que da la solución de S (reflejandolo mediante la marca). Esto lo hace mediante la evaluación de las parejas formadas por el paquete que se evaluó antes que él.

Los valores para nuestro ejemplo son los siguientes: (M = valor de la marca)

S / M S / M

(1 , 1) =0 / 0 (1 , 2)= 0 / 0

(2 , 1) =1 / 1 (2 , 2)= 1 / 1

(3 , 1) =0 / 0 (3 , 2)= 0 / 0

(4 , 1) =0 / 0 (4 , 2)= 1 / 1

(5 , 1) =0 / 0 (5 , 2)= 0 / 0

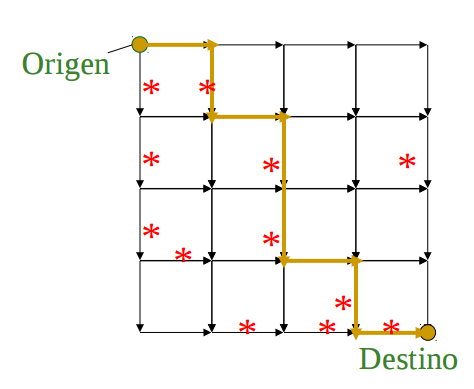
(6 , 1) =0 / 0 (6 , 2)= 0 / 0

Se puede ver como para los dos primeros paquetes de peso 2 y para un peso de m = 2 ambos paquetes confirma el valor de S de manera autónoma pues ellos mismos por separado forman una solución.

Sin embargo para un peso de m = 4 solo P2 puede proporcionar una solución ya que cuenta con la evaluación de P1 para m = 2 que es el peso que queda tras restarle a 4 el peso de P2 que también es de 2.

De esta manera se avanza en la evaluación de todos los paquetes hasta llegar a conocer el valor de S(D/2, n) y saber si tenemos una solución.

**Ejercicio 4.**El problema del turista en Manhattan

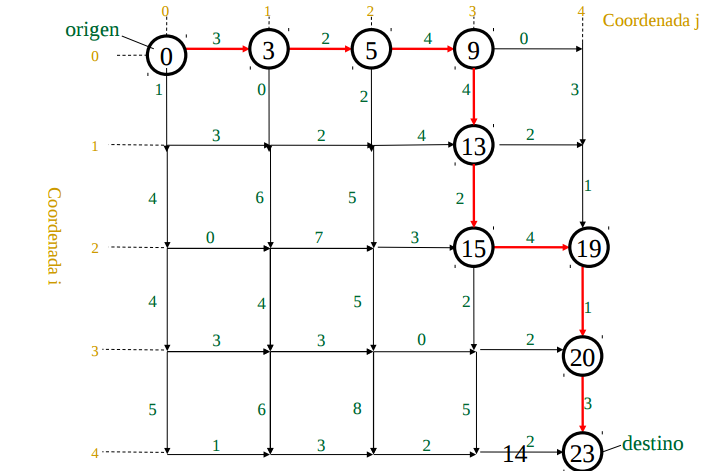


**Enunciado:**

Se trata de encontrar un camino (desde un origen a un destino) tal que caminando solo hacia el sur y hacia el este, nos permita visitar el mayor número lugares turísticos (\*) en el mapa de Manhattan.

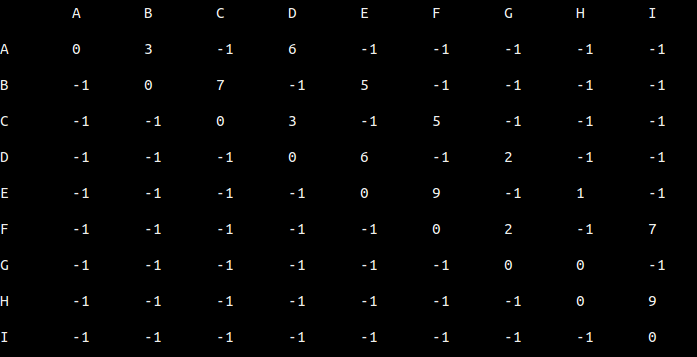
**Enfoque:**

Este problema no cumple el Principio de Optimalidad de Bellman, pero al ser n-etápico podemos enfocarlo como un problema de programación dinámica.

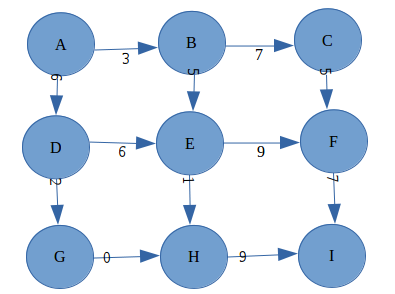


El problema que se nos plantea se puede resolver mediante una matriz de costes, siendo el coste más alto en el que se encuentran más lugares turisticos o más se acerque a los mismos.

La forma de resolver el problema que planteamos consiste en construir una matriz de adyacencia (en este caso la creamos aleatoriamente teniendo en cuenta las direcciones permitidas).

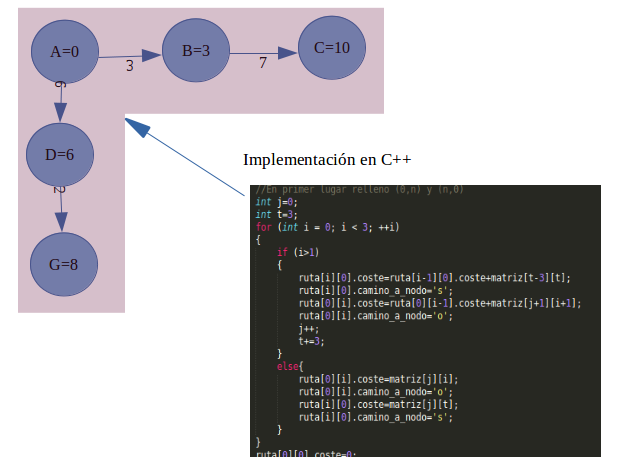


Esta matriz nos generaría el siguiente grafo:

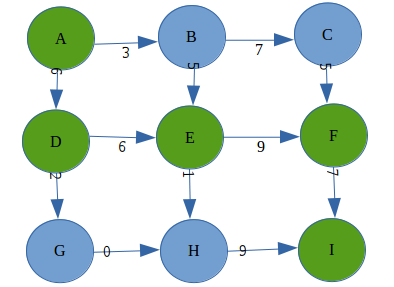


**Solución:**

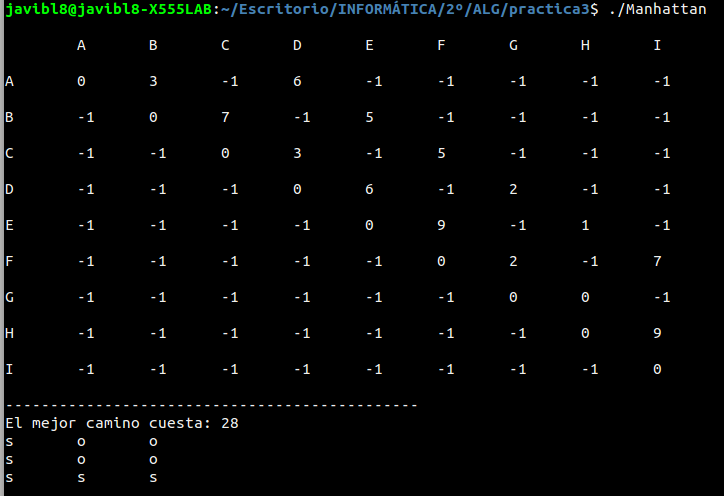
Como nosotros queremos pasar por el mayor número de sitios turísticos tendremos que calcular el camino mayor y para ello lo primero que hacemos es calcular el coste de los nodos al sur y al oeste. Hacemos esto en primer lugar porque son nodos que solo son accesibles desde un punto y por lo tanto no hay que hacer ninguna comprobación previa.



Posteriormente y para los elementos centrales iremos comprobando desde que nodo es mejor acceder, desde la cuál obtenemos mayor coste. Repetimos el proceso de forma iterativa ya al final nos queda que el mejor camino es:



Si calculamos el coste de este camino nos da: 6+6+7+9=28



Como output de nuestro programa además podemos ver el camino óptimo a cada nodo,para ello nos situamos en el nodo que queremos visitar y seguimos la secuencia 's'(sur) 'o'(oeste) de forma inversa; en este caso sería al nodo abajo del todo a la derecha:

PRIMER NODO->Sur->Oeste->Oeste->Sur->OBJETIVO

Por lo tanto el algoritmo que hemos empleado en este caso práctico sería:

Algoritmo Manhattan:

Begin

For s:= 1 to i+j do begin 

coste(0,s) Rellenar [0,n] y [n,0]

coste(s,0)

For k = 1 to s-1 do

calcular\_coste(k,s-k) Calcular costes para puntos múltiplemente accesibles

end

end;

End;

**Eficiencia:**

El lazo más externo se lleva un tiempo O(n2) cuando i+j = n.

Por tanto, el uso de la PD supone un tiempo O(n), muy inferior al del método directo.

**Apéndice de códigos:**

**Ejercicio 1:**

//Ejercicio dinámico.

#include <iostream>

#include <vector>

using namespace std ;

int Max (int n, int M) ;

int main () {

int M ;

int n ;

cout << "Introduce M = " ;

cin >> M ;

cout << "Introduce n = " ;

cin >> n ;

cout << "Multiplicación máxima alcanzada: " << Max (n, M) << endl ;

}

int Max (int n, int M) {

vector<vector<int> > T ;

// Creación de la matriz de T(n+1 \* M+1).

T.reserve(n+1) ;

for (int k = 0 ; k <= n ; k++) T[k].reserve(M+1) ; // O(n)

// Suma de n naturales que sea igual a 0.

for (int k = 0 ; k <= n ; k++) T[k][0] = 0 ; // O(n)

// Suma de 1 natural que sea igual a M.

for (int c = 1 ; c <= M ; c++) T[1][c] = c ; // O(M)

for (int k = 2 ; k <= n ; k++) { // O(n\*M\*M)

for (int c = 1 ; c <= M ; c++) {

T[k][c] = 0 ;

for (int x = 1 ; x <= c ; x++) {

T[k][c] = max (T[k][c], x \* T[k-1][c-x]) ;

}

}

}

return (T[n][M]) ;

}

//Ejercicio recursivo.

#include <iostream>

#include <vector>

#include <cstdlib>

using namespace std ;

int Max (int n, int M) ;

int main () {

int M ;

int n ;

cout << "Introduce M = " ;

cin >> M ;

cout << "Introduce n = " ;

cin >> n ;

cout << "Multiplicación mayor es: " << Max (n, M) << endl ;

}

int Max (int n, int M) {

//Nada más que siguiendo la declaración recurrente que aparecía junto al problema.

// Si el numero a encontrar es 0, y la serie de numeros empleados es mayor o igual que 0

//Como M es 0, no existe ninguna suma de numeros naturales que sumen 0, por lo tanto la función tendrá que devolver 0

if (M == 0 && n >= 0) return 0 ;

//Si nada más que se puede emplear un número natural para obtener el número M, la función obligatoriamente deberá devolver

//como resultado el número M

// Suma de un natural igual a M.

if (n == 1 && M > 0) return M ;

if (n > 1 && M > 0) {

int max = 0 ;

for (int x = 1 ; x <= M ; x++) {

// Llamada recursiva para todas las posibles combinaciones de números naturales que existan hasta llegar a M

int auxiliar = x \* Max (n - 1, M - x) ;

// Máximo con el nuevo elemento.

if (max < auxiliar) max= auxiliar ;

}

return max ;

}

}

**Ejercicio 2:**

#include<iostream>  
#include <vector>  
using namespace std;  
  
vector<vector<int> > Maxima\_anchura(int k);  
  
int main(){  
//llamamos a la funcion Maxima\_anchura la cual  
//tiene dentro la matriz de anchuras entre islas  
//y la salida que nos da este problema es la matriz con  
//la maxima anchura entre unas islas y otras  
//El resultado lo guardamos en la matriz M:  
vector<vector<int> > M = Maxima\_anchura(1);  
//Mostramos por pantalla por ejemplo el camino más optimo entre  
//0 y 3  
cout << M[0][3] << endl;  
}  
//funcion  
vector<vector<int> > Maxima\_anchura(int k){  
int aux=0;  
//Creación de la matriz anchura A  
 vector<vector<int> > A;  
   
 A.reserve(4);  
  
 for(int i= 0; i < 4; i++)  
 A[i].reserve(4);  
  
 A[0].push\_back(-1);  
 A[0].push\_back(2);  
 A[0].push\_back(0);  
 A[0].push\_back(3);  
 A[1].push\_back(2);  
 A[1].push\_back(-1);  
 A[1].push\_back(1);  
 A[1].push\_back(4);  
 A[2].push\_back(0);  
 A[2].push\_back(1);  
 A[2].push\_back(-1);  
 A[2].push\_back(2);  
 A[3].push\_back(3);  
 A[3].push\_back(4);  
 A[3].push\_back(2);  
 A[3].push\_back(-1);  
  
//Creación de la matriz M  
 vector<vector<int> > M;  
  
 M.reserve(4);  
  
 for(int i=0;i<4;i++)  
 M[i].reserve(4);  
  
 M[0].push\_back(-1);  
 M[0].push\_back(2);  
 M[0].push\_back(0);  
 M[0].push\_back(3);  
 M[1].push\_back(2);  
 M[1].push\_back(-1);  
 M[1].push\_back(1);  
 M[1].push\_back(4);  
 M[2].push\_back(0);  
 M[2].push\_back(1);  
 M[2].push\_back(-1);  
 M[2].push\_back(2);  
 M[3].push\_back(3);  
 M[3].push\_back(4);  
 M[3].push\_back(2);  
 M[3].push\_back(-1);  
  
 for(int k=k; k<4; k++)  
 {  
 for(int i=0; i<4; i++)  
 {  
 for(int j=0;j<4; j++)  
 {  
 if(M[i][k] < M[k][j])  
 {  
 aux=M[i][k];  
 }  
   
 else  
 aux=M[k][j];  
  
 if( (M[i][j] < aux) && (i!=j) )  
 M[i][j]=aux;  
 }  
 }  
 }  
  
 return M;  
}

**Ejercicio 3:**

//Compilar con: g++ ej3.cpp -o ej3 -std=gnu++0x

#include <iostream>

#include <vector>

#include <set>

#include <cstdlib>

using namespace std;

set<int> hayReparto(const vector<int> &p){ //El vector viene inicializado con una componente mas, la 0 q no usaremos

int D = 0;

int n = p.size() - 1;

set<int> C;

bool S[500][500];

bool marca[500][500];

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

D = D + p[i];

for(int k = 0; k <= n; k++)

S[0][k] = true;

for(int m = 1; m <= D/2; m++)

S[m][0] = false;

for(int k = 1; k <= n; k++){

for(int m = 1; m <= D/2; m++){

if(m < p[k]){

S[m][k] = S[m][k-1];

marca[m][k] = false;

}

else if( S[m-p[k]][k-1] ){

S[m][k] = S[m-p[k]][k-1];

marca[m][k] = true;

}

else{

S[m][k] = S[m][k-1];

marca[m][k] = false;

}

cout << "S[" << m << "][" << k << "(" << p[k] << ")] = " << S[m][k] << "/" << marca[m][k] << endl;

}

cout << endl;

}

if( S[D/2][n] ){

int i = D/2;

int j = n;

while (j > 0){

if( marca[i][j] ){

C.insert(j);

i = i - p[j];

}

j = j - 1;

}

}

return C;

}

int main(){

set<int> C;

vector<int> p;

p.push\_back(0); //0

p.push\_back(2); //1

p.push\_back(2); //2

p.push\_back(1); //3

p.push\_back(5); //4

p.push\_back(2); //5

C = hayReparto(p);

for(auto it = C.begin(); it != C.end(); it++)

cout << \*it << " ";

cout << endl;

}

**Ejercicio 4:**

void Ruta(int matriz[fc][fc]){

camino ruta[3][3];

//En primer lugar relleno (0,n) y (n,0)

int j=0;

int t=3;

for (int i = 0; i < 3; ++i)

{

if (i>1)

{

ruta[i][0].coste=ruta[i-1][0].coste+matriz[t-3][t];

ruta[i][0].camino\_a\_nodo='s';

ruta[0][i].coste=ruta[0][i-1].coste+matriz[j+1][i+1];

ruta[0][i].camino\_a\_nodo='o';

j++;

t+=3;

}

else{

ruta[0][i].coste=matriz[j][i];

ruta[0][i].camino\_a\_nodo='o';

ruta[i][0].coste=matriz[j][t];

ruta[i][0].camino\_a\_nodo='s';

}

}

ruta[0][0].coste=0;

if (ruta[0][1].coste+matriz[1][4]>ruta[1][0].coste+matriz[3][4])

{

ruta[1][1].coste=ruta[0][1].coste+matriz[1][4];

ruta[1][1].camino\_a\_nodo='s';

}

else{

ruta[1][1].coste=ruta[1][0].coste+matriz[3][4];

ruta[1][1].camino\_a\_nodo='o';

}

if (ruta[1][1].coste+matriz[4][7]>ruta[2][0].coste+matriz[6][7])

{

ruta[2][1].coste=ruta[1][1].coste+matriz[4][7];

ruta[2][1].camino\_a\_nodo='s';

}

else{

ruta[2][1].coste=ruta[2][0].coste+matriz[6][7];

ruta[2][1].camino\_a\_nodo='o';

}

if (ruta[0][2].coste+matriz[2][5]>ruta[1][1].coste+matriz[4][5])

{

ruta[1][2].coste=ruta[0][2].coste+matriz[2][5];

ruta[1][2].camino\_a\_nodo='s';

}

else{

ruta[1][2].coste=ruta[1][1].coste+matriz[4][5];

ruta[1][2].camino\_a\_nodo='o';

}

if (ruta[1][2].coste+matriz[5][8]>ruta[2][1].coste+matriz[7][8])

{

ruta[2][2].coste=ruta[1][2].coste+matriz[5][8];

ruta[2][2].camino\_a\_nodo='s';

}

else{

ruta[2][2].coste=ruta[2][1].coste+matriz[7][8];

ruta[2][2].camino\_a\_nodo='o';

}

imprimir\_camino(ruta);

}