ANÁLISIS DE ALGORITMOS

Alumno: Martín Eduardo Barriga Vargas

Grupo 3CM3

Ejercicio 07: Diseño de soluciones de Programación Dinámica

13 de mayo de 2019

Longest common subsequence

Puntos	13.76	Límite de memoria	32 MiB
Límite de tiempo (caso)	500ms	Límite de tiempo (total)	7.5s
Límite de entrada (bytes)	10 KiB		

Descripción

Al finalizar su viaje por cuba, Edgardo se puso a pensar acerca de problemas mas interesantes que sus alumnos podrian resolver.

En esta ocasión tu trabajo es el siguiente:

Dadas 2 cadenas A y B, debes de encontrar la subsecuencia común mas larga entre ambas cadenas.

Entrada

La primera linea contendra la cadena A. En la segunda linea vendra la cadena B.

Salida

La longitud de la subsecuencia común mas larga.

Ejemplo

Entrada	Salida	Descripción
AGCT AMGXTP	3	La subsecuencia comun mas larga es: AGT

Límites

- $1 \leq |A| \leq 10$
- $1 \leq |B| \leq 10$

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2019-05-09 18:28:28	441ad610	Respuesta correcta	100.00%	cpp11	7.03 MB	0.07 s	

```
1. #include <iostream>
2. #include <string>
3. #include <vector>
4. using namespace std;
5. vector<int> arr(1001);
6. vector<vector<int> > mat(1001, arr);
7. int lcs(string x, string y){
8.     for(int i = x.size(); i >=0 ; i--){
9.         for(int j = y.size(); j >=0; j--){
10.             if(i==x.size() || j==y.size() ) mat[i]
    [j] = 0;
11.             else{
12.                 if(x[i]==y[j]){
13.                     mat[i][j] = 1 + mat[i+1][j+1];
14.                 }
15.                 else{
```

```
16.                     mat[i][j] = max(mat[i+1][j], mat[i]
   [j+1]);
17.                 }
18.             }
19.         }
20.     }
21.     return mat[0][0];
22. }
23. int main(){
24.     string x,y;
25.     cin >> x >> y;
26.
27.     int resultado = lcs(x,y);
28.     /*for(int i =0;  i <=x. size(); i++){
29.         for(int j = 0; j <= y.size(); j++){
30.             cout << mat[i][j] << " ";
31.         }
32.         cout << endl;
33.     }*/
34.     cout << resultado << endl;
```

```
35.     return 0;
36. }
```

Explicación de la solución:

De primera instancia se elaboró el código pensando en una solución Top Down, tomando en cuenta que podían existir tres casos posibles al recorrer nuestras dos cadenas, la primera es que al analizar el caracter en la posición i de la cadena uno y el de la posición j de la cadena dos estos sean iguales. como nuestro objetivo es encontrar la subsecuencia más larga, se podría decir que estos dos caracteres forman parte de una posible solución, para lo cual vamos a regresar uno (debido a que sabemos que al coincidir se incrementa la longitud de nuestra subcadena) más el resultado a avanzar los dos indices en una unidad. El segundo caso es que sean diferentes, entonces vamos a regresar el máximo valor entre haber escogido avanzar el indice i y haber avanzado el indice j, finalmente, en caso de que alguno de los índices tenga mayor longitud que su cadena correspondiente se regresa un cero, pues sabemos que no hubo coincidencia en ese punto. Siempre se pregunta al inicio si el resultado para esa combinación de índices ya existe, si existe se regresa ese valor, pues ya sabemos la solución y no hay necesidad de volverla a calcular, sino entonces obtenemos la solución de esa combinación y la guardamos en la matriz.
Sin embargo, esta solución es más tardada la solución Botton-Up debido a la cantidad de llamadas recursivas que se hacen, por lo cuál, utilizamos de nuevo una matriz pero no hacemos recursión, aún así, utilizamos el mismo razonamiento (sólo que comenzamos por el final y terminamos donde los dos indices valen 0) y llenando con ceros las posiciones donde alguno de los índices tiene el valor de la longitud de su respectiva cadena, se puede notar que efectivamente sigue el mismo patrón que la solución Top, pero gracias a que no usamos llamadas recursivas, tarda menos y la solución es aceptada.
La complejidad es O(nm).

Modelo de implementación: **Botton-Up.**

Bacterias

Puntos	14.22	Límite de memoria	32 MiB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

Descripción.

Un grupo de biólogos computacionales ha diseñado un experimento para decidir si una colonia de microbios es capaz de resolver problemas cuando se le estimula de ciertas formas específicas. En este experimento se construye un recipiente con la forma de una cuadricula rectangular con m renglones y n columnas. En cada uno de los cuadritos de la cuadricula se coloca cierta cantidad de un compuesto químico que le es muy desagradable a los microbios y que, por lo tanto, los microbios preferirían evitar lo más posible. El recipiente está inclinado de tal forma que los microbios solo puede avanzar hacia el este o hacia el sur. Por supuesto, los microbios tampoco pueden salir del recipiente. Al principio la colonia de microbios está localizada en el cuadrito correspondiente al primer renglón y primera columna del recipiente. Al final se espera que la colonia de microbios termine en el cuadrito correspondiente al último renglón y última columna del recipiente. En términos de un sistema de coordenadas, los microbios comienzan en la coordenada (1, 1) y terminan en la coordenada (m, n). Antes de llevar a cabo el experimento, los científicos desean calcular la cantidad c de unidades del compuesto químico que deberá soportar la colonia en su trayecto, esto es, la suma de todas las cantidades del compuesto químico que fueron depositadas en todos los cuadritos por los que pasen. En el ejemplo se muestra un recipiente donde el mínimo valor posible de c es 17.

Entrada

Dos enteros m y n separados por un espacio, seguidos de m renglones cada uno con n enteros separados por espacios. Estos enteros representan las

cantidades del compuesto químico. Puedes suponer que $2 \leq m \leq 100$, $2 \leq n \leq 100$ y que todas las demás cantidades están entre 1 y 9, incluyéndolos.

Entrada

Dos enteros m y n separados por un espacio, seguidos de m renglones cada uno con n enteros separados por espacios. Estos enteros representan las cantidades del compuesto químico. Puedes suponer que $2 \leq m \leq 100$, $2 \leq n \leq 100$ y que todas las demás cantidades están entre 1 y 9, incluyéndolos.

Salida

Deberás escribir en pantalla un entero c el cual representa la cantidad mínima del compuesto químico al que puede estar expuesto la colonia durante su recorrido. Consideraciones Tu programa se evaluará con varios casos de prueba

Ejemplo

Entrada	Salida
2 2 8 1 8 3	12

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2019-05-11 19:49:32	d2079592	Respuesta correcta	100.00%	cpp11	3.25 MB	0.02 s	
2019-05-11 19:48:17	37fd7fbf	Error de ejecución	70.00%	cpp11	3.24 MB	0.04 s	

Nuevo envío

```
1. #include <iostream>
2. #include <math.h>
3. #include <vector>
4. using namespace std;
5. int mat[100][100];
6. int m, n;
7. vector <int> arr(101,-1);
8. vector <vector <int> > respuesta(101, arr);
9.
10.int camino(int i, int j){
11.     if(respuesta[i][j]!= -1){
12.         return respuesta[i][j];
13.     }
14.     else if(i==m || j == n) {
15.         respuesta[i][j] = 90001;
16.     }
```

```
17.     else if( (i == m -1) && (j == n-1)){
18.         respuesta[i][j] = mat[i][j];
19.     }
20.     else{
21.         int a,b;
22.         a = camino(i+1, j);
23.         b = camino(i,j+1);
24.         respuesta[i][j] = mat[i][j] + min(camino(i+1,
    j),camino(i,j+1));
25.     }
26.     return respuesta[i][j];
27. }
28. int main(){
29.     int res;
30.     cin >> m >> n;
31.     for(int i = 0; i < m; i++){
32.         for(int j =0; j <n; j++){
33.             cin >> mat[i][j];
34.         }
35.     }
```

```
36.     res = camino(0, 0);
37.     cout << res << endl;
38.     return 0;
```

Explicación de la solución:

Una vez que hemos leído la matriz, lo que tenemos que hacer es mandar a llamar a nuestra función con un valor 0 para el índice i, 0 para el j. Se encargará de analizar tres cosas: Saber si el indice i o el indice j ya superó los límites de la matríz, para este caso se regresará un valor super grandísimo, puesto que lo que queremos es el mínimo camino y eso estará diciendo que por ahí no hay que pasar, en caso de que podamos avanzar tanto para el índice i más uno y al j más uno, calcularemos cada uno de los dos y elegiremos el más chico. En caso de que no podamos avanzar ni para abajo ni a la derecha, significará que hemos llegado a el valor de la meta, por lo cual sólo regresamos ese valor, pues no podemos avanzar a ningún otro lugar de ahí.
Toda este razonamiento se implementa con DP inicializando una matriz con puros -1, así al inicio de nuestra función preguntaremos por el valor de la posición i y j en nuestra matriz, si es -1 entonces significa que aún no hemos calculado el camino mínimo para esa posición en la matriz, en caso contrario sólo se regresa el valor que se tiene en dicha posición de la matriz.
El orden de complejidad es O(nm)

Modelo de implementación: **Top-Down.**

Escaleras

Puntos	14.15	Límite de memoria	128 MiB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	1m0s
Límite de entrada (bytes)	10 KiB		

Una rana quiere subir a la parte mas alta de una escalera con exactamente N escalones. Para subir escalones, inevitablemente tiene que saltar. En cada salto, la rana puede subir desde 1 hasta k escalones. Si la escalera tuviera 4 escalones, y k=3 la rana podría subir la escalera con 7 sucesiones de saltos distintas:

1, 1, 1, 1 1, 1, 2 1, 2, 1 1, 3 2, 1, 1 2, 2 3, 1

Problema

Escribe un programa que dados N y k, determine el número de formas en que la rana puede subir la escalera.

Entrada

Línea 1: Dos números enteros positivos separados por un espacio, representando N y K respectivamente

Ejemplo: 4 3

Salida

Línea 1: Un solo número entero representando el número de formas de las que es posible que la rana suba la escalera.

Ejemplo: 7

Consideraciones

- $0 < N < 10000$
- $0 < k < 100$

- Nunca habrá mas de 2^62 formas de subir la escalera

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Detalles
2019-05-10 08:57:59	29206314	Respuesta correcta	100.00%	cpp11	10.94 MB	0.03 s	
2019-05-10 08:47:30	a28a6537	Respuesta parcialmente correcta	83.33%	cpp11	11.04 MB	0.03 s	

```
1. #include <iostream>

2. #include <vector>

3. using namespace std;

4. vector <int> paso;

5. vector<long long int> arr(10002, -1);

6. vector <vector <long long int> > mat(102, arr);

7. long long int escalera(int indice, int escalon){

8.     if(mat[indice][escalon] != -1) return mat[indice]
   [escalon];

9.     if(indice == paso.size() || (escalon -
    paso[indice]) <0 ){

10.        mat[indice][escalon] = 0;

11.    }

12.    else if(escalon - paso[indice]  == 0) mat[indice]
   [escalon] = 1;

13.    else {
```

```
14.         mat[indice][escalon] = escalera(0, escalon-
   paso[indice]) + escalera(indice+1, escalon);
15.     }
16.     return mat[indice][escalon];
17. }
18. int main(){
19.     int escalon, salto;
20.     long int res;
21.     cin >> escalon >> salto;
22.
23.     for(int i = 0; i < salto; i++){
24.         paso.push_back(i+1);
25.     }
26.     res = escalera(0, escalon);
27.     cout << res << endl;
28.     return 0;
29. }
```

Explicación de la solución:

Para la solución de este problema debemos de pensar en que podemos tanto tener varias monedas de una misma, ninguna de algún tipo o sólo una de algún tipo. Por lo cuál lo que se hará es primeramente preguntar si en nuestra matriz de solución ya existe la solución para el indice que representa la posición en el vector de monedas que estamos calculando y la cantidad de escalones que nos hacen falta. En caso de que no exista preguntamos si el índice ya se salió del tamaño de nuestro vector de tipos de saltos o si la magnitud del salto en el que nos encontramos ya superó la cantidad de escalones que nos hacen falta avanzar, en ese caso regresamos un 0, pues esa no es una solución posible. Una solución posible sería que justamente el valor de escalones que nos hacen falta recorrer menos el tipo de salto que vamos a dar sea igual a 0, entonces en nuestra matriz guardamos un uno en esa posición. En caso contrario entonces guardamos en la posición de la matriz la suma de llamar a la función, manteniéndonos restando el mismo tipo de salto y el no tomar ese tipo de salto y avanzar nuestro índice de tipo de salto en 1.
Cabe señalar que una vez que nos han dado el valor máximo que un salto puede tener, se genera un vector con los valores del 1 hasta ese valor que nos dieron, para así poder ir analizando los valores dentro del arreglo. Sin embargo, pensándolo nuevamente, este paso se puede omitir, pues el valor del arreglo en algún índice solamente es ese índice+1.
El orden de complejidad es O(NK).

Modelo de implementación: **Top-Down.**

Dollars

New Zealand currency consists of $100, $50, $20, $10, and $5 notes and $2, $1, 50c, 20c, 10c and 5c coins. Write a program that will determine, for any given amount, in how many ways that amount may be made up. Changing the order of listing does not increase the count. Thus 20c may be made up in 4 ways: 1×20c, 2×10c, 10c+2×5c, and 4×5c.

Input
Input will consist of a series of real numbers no greater than $300.00 each on a separate line. Each amount will be valid, that is will be a multiple of 5c. The file will be terminated by a line containing zero (0.00).

Output
Output will consist of a line for each of the amounts in the input, each line consisting of the amount of money (with two decimal places and right justified in a field of width 6), followed by the number of ways in which that amount may be made up, right justified in a field of width 17.

Sample Input
```
0.20
2.00
0.00
```

Sample Output
```
  0.20                4
  2.00              293
```

23319416	147 Dollars	Accepted	C++11	0.010	2019-05-11 21:22:54

```
1. #include<iostream>
2. #include <vector>
3. #include <math.h>
4. using namespace std;
5. vector <float> moneda(11);
6. vector <long long int> arr(30001, -1);
7. vector< vector<long long int> > mat(12, arr);
8. long long int dollars(int indice, float cantidad){
9.     if(mat[indice][cantidad*100] != -1){
10.         return mat[indice][cantidad*100];
11.     }
12.     else if(cantidad == 0) {
13.         mat[indice][cantidad*100] = 1;
14.     }
15.     else if(indice > 10){
16.         mat[indice][cantidad*100] = 0;
17.     }
18.     else if(moneda[indice] > cantidad){
```

```
19.         mat[indice][cantidad*100]  = dollars(indice+1,
    cantidad);

20.     }

21.     else{

22.

23.         mat[indice][cantidad*100] = dollars(indice,
    round((cantidad -moneda[indice])*100)/
    100) + dollars(indice+1, cantidad);

24.     }

25.     return mat[indice][cantidad*100];

26. }

27. int main(){

28.     moneda[0] = 100.0;

29.     moneda[1] = 50.0;

30.     moneda[2] = 20.0;

31.     moneda[3] = 10.0;

32.     moneda[4] = 5.0;

33.     moneda[5] = 2.0;

34.     moneda[6] = 1.0;

35.     moneda[7] = 0.5;

36.     moneda[8] = 0.2;
```

```
37.     moneda[9] = 0.1;
38.     moneda[10] = 0.05;
39.     float prueba;
40.     long long int resultado;
41.     int espacio1,espacio2, i;
42.     while(cin >> prueba){
43.         if(prueba == 0.00) return 0;
44.         resultado = dollars(0, prueba);
45.         i=1;
46.         while(((int)prueba)/pow(10, i)>=1){
47.             i++;
48.         }
49.         espacio1=i+3;
50.         i=1;
51.         while(resultado/pow(10, i)>=1){
52.             i++;
53.         }
54.         espacio2=i;
55.         for(i = 6; i >espacio1; i--) cout << " ";
56.         printf("%.2f", prueba);
```

```
57.         for( i = 17; i > espacio2; i--) cout << " " ;
58.         cout << resultado << endl;
59.     }
60.     return 0;
61.}
```

Explicación de la solución:

Como ya se conoce las denominaciones de las monedas que se tendrán, se elabora un arreglo de tamaño 11 y le metemos los valores de las monedas. Luego se manda a nuestra función el índice 0, y el valor al que tenemos que llegar.
Dentro de la función, se puede dar el caso de que el índice se haya salido del tamaño del arreglo, es decir 11, entonces se regresará un 0, indicando que ese no es una posible solución. Si justamente nuestra cantidad a la que queremos llegar ya es un 0, regresamos un 1, indicando que es una posible solución. En caso de que la denominación sea más grande que el valor al que queremos llegar entonces regresamos el valor obtenido probando con la denominación siguiente, llamando a la función con índice + 1. En caso contrario, obtenemos la suma de probar con el índice +1 y probando con el mismo índice y restando ala denominación en el índice al valor meta.
Una vez planteada esta solución procedemos a implementarla con DP, donde únicamente tenemos que preguntar al inicio de la función si ya existe un valor diferente a -1 en la matriz en los índices que manejamos, si es así entonces sólo regresamos ese valor, en caso contrario calculamos todo lo que ya anteriormente mencionamos, guardamos el valor en la matriz y lo regresamos.

El orden de complejidad es O(NumeroMonedas*Dinero)

Modelo de implementación: **Top-Down.**

ELIS - Easy Longest Increasing Subsequence

no tags

Given a list of numbers A output the length of the longest increasing subsequence. An increasing subsequence is defined as a set {i0 , i1 , i2 , i3 , ... , ik} such that 0 <= i0 < i1 < i2 < i3 < ... < ik < N and A[i0] < A[i1] < A[i2] < ... < A[ik]. A longest increasing subsequence is a subsequence with the maximum k (length).

i.e. in the list {33 , 11 , 22 , 44}

the subsequence {33 , 44} and {11} are increasing subsequences while {11 , 22 , 44} is the longest increasing subsequence.

Input

First line contain one number N (1 <= N <= 10) the length of the list A.

Second line contains N numbers (1 <= each number <= 20), the numbers in the list A separated by spaces.

Output

One line containing the lenght of the longest increasing subsequence in A.

Example

Input:
5
1 4 2 4 3
Output:
3

23751747	2019-05-12 06:37:38	EscubyDubyDu	Easy Longest Increasing Subsequence	**accepted** edit ideone it	0.00	16M	CPP14

```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. vector <int> numero(11);
5. vector <int> arr(12,-1);
6. vector <vector <int> > mat(22, arr);
7. long long int elis(int indice, int n, int anterior){
8.     if(mat[anterior][indice]!=-1){
9.         return mat[anterior][indice];
10.    }
11.    if(indice == n) {
12.        mat[anterior][indice] = 0;
13.    }
14.    else if(numero[indice] <= anterior){
15.        mat[anterior][indice] =  elis(indice+1, n,
   anterior);
16.    }
17.    else{
```

```
18.         mat[anterior][indice] = max(elis(indice+1, n,
    anterior), 1 + elis(indice+1, n, numero[indice]));
19.     }
20.     return mat[anterior][indice];
21. }
22. int main(){
23.     int n;
24.     cin >> n;
25.     for(int i = 0; i < n ; i++){
26.         cin >> numero[i];
27.     }
28.     cout << elis(0, n, 0) << endl;
29.     return 0;
30. }
```

Explicación de la solución:

Para este problema tenemos 3 casos, sin contar la parte de la DP. El primer caso es que nuestro índice con el cuál vamos iterando sobre nuestro vector de números, se haya pasado de la cantidad de elementos que se tienen, es decir n, para lo cual regresamos 0 indicando que ese elemento no forma parte de la subsecuencia.
El segundo caso es que nuestro número en el índice sea menor al último número que agregamos a la subsecuencia, en dicho caso no se contará en cuenta ese número y regresaremos el resultado de la función en el índice +1.
El tercer caso es que efectivamente sea mayor al número anterior, para lo cual regresaremos el número máximo de lo que regrese la función al tomar ese número en cuenta, actualizando el índice, el número anterior y sumándole 1, y entre lo que regrese la función sin tomar en cuenta ese número.
Al igual que en los otros problemas, una vez planteada la solución procedemos a implementarla con DP, donde únicamente tenemos que preguntar al inicio de la función si ya existe un valor diferente a -1 en la matriz en los índices que manejamos, si es así entonces sólo regresamos ese valor, en caso contrario calculamos todo lo que ya anteriormente mencionamos, guardamos el valor en la matriz y lo regresamos.
La complejidad es O(NV). Donde N es la cantidad de números y V es el valor máximo del número más grande

Modelo de implementación: **Top-Down.**

KNAPSACK - The Knapsack Problem

no tags

The famous knapsack problem. You are packing for a vacation on the sea side and you are going to carry only one bag with capacity S (1 <= S <= 2000). You also have N (1<= N <= 2000) items that you might want to take with you to the sea side. Unfortunately you can not fit all of them in the knapsack so you will have to choose. For each item you are given its size and its value. You want to maximize the total value of all the items you are going to bring. What is this maximum total value?

Input

On the first line you are given S and N. N lines follow with two integers on each line describing one of your items. The first number is the size of the item and the next is the value of the item.

Output

You should output a single integer on one like - the total maximum value from the best choice of items for your trip.

Example

Input:
```
4 5
1 8
2 4
3 0
2 5
2 3
```

Output:
```
13
```

ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
23747524	2019-05-11 07:25:27	EscubyDubyDu	The Knapsack Problem	**accepted** edit ideone it	0.00	31M	CPP14

```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. vector <int> valor;
5. vector <int> tam;
6. vector <int> arr(2001, -1);
7. vector <vector <int> > mat(2001, arr);
8. int knapsack( int indice, int capacidad){
9.     if(mat[indice][capacidad] != -1){
10.        return mat[indice][capacidad];
11.    }
12.    if(indice == valor.size()){
13.        mat[indice][capacidad] = 0;
14.    }
15.    else if(tam[indice] > capacidad){
16.        mat[indice][capacidad] = knapsack (indice+1,
   capacidad);
```

```
17.     }
18.     else {
19.         int val1, val2;
20.         val1 = valor[indice] + knapsack(indice+1,
    capacidad - tam[indice]);
21.         val2 = knapsack(indice+1, capacidad);
22.         mat[indice][capacidad] = max(val1, val2);
23.     }
24.     return mat[indice][capacidad];
25. }
26. int main(){
27.     int capacidad, n, res, size, value;
28.     cin >> capacidad >> n;
29.     for(int i = 0; i < n; i++){
30.         cin >> size>> value;
31.         tam.push_back(size);
32.         valor.push_back(value);
33.     }
34.     res = knapsack(0,capacidad);
35.     cout << res << endl;
```

```
36.     return 0;
37.}
```

Explicación de la solución:

Para la solución de este problema se tuvieron en cuenta tres aspectos. Cuando nuestra capacidad es igual a 0, para la cual regresamos un 0, representando que no se agregó ningún objeto de valor. El segundo es cuando la capacidad de la mochila es menor a la del objeto, entonces regresamos lo que sea calculado para el siguiente objeto, mandando así el indice+1.
Finalmente, si ninguna de las otras condiciones se cumplió entonces obtenemos el resultado de mandar llamar a nuestra función tomando ese objeto, modificando la capacidad y aumentando el indice+1, y lo comparamos con el resultado de mandar a llamar la función sin tomar en cuenta el objeto, aumentando sólo el indice+1.

Para modificar nuestra solución e implementarla con DP, únicamente tenemos que preguntar al inicio de la función si ya existe un valor diferente a -1 en la matriz en los índices que manejamos, si es así entonces sólo regresamos ese valor, en caso contrario calculamos todo lo que ya anteriormente mencionamos, guardamos el valor en la matriz y lo regresamos.
La complejidad es O(NS).

Modelo de implementación: **Top-Down**

Bibliografía: http://www.eafranco.com/docencia/analisisdealgoritmos/files/08/Tema08.pdf