



INFORME DE LABORATORIO

INFORMACIÓN BÁSICA

ASIGNATURA:	ANÁLISIS Y DISEÑO DE ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	PROGRAMACIÓN DINÁMICA				
NÚMERO DE PRÁCTICA:	P2	AÑO LECTIVO:	2024	SEMESTRE:	PAR
ESTUDIANTES: 20173377, ALMANZA MAMANI EDGAR RAUL					
DOCENTES: Marcela Quispe Cruz, Manuel Loaiza, Alexander J. Benavides					

RESULTADOS Y PRUEBAS

El informe se presenta con un formato de artículo.
Revise la sección de *Resultados Experimentales*.

CONCLUSIONES

El informe se presenta con un formato de artículo.
Revise la sección de *Conclusiones*.

METODOLOGÍA DE TRABAJO

El informe se presenta con un formato de artículo.
Revise la sección de *Diseño Experimental*.

REFERENCIAS Y BIBLIOGRAFÍA

El informe se presenta con un formato de artículo.
Revise la sección de *Referencias Bibliográficas*.

Programación Dinámica – Ejemplos de Aplicación

Resumen

Este artículo presenta la resolución de tres problemas mediante programación dinámica. Cada sección detalla la implementación teórica y práctica, destacando la metodología SRTBOT para diseñar soluciones recursivas eficientes. Se concluye que las soluciones desarrolladas mejoran significativamente la eficiencia computacional en comparación con enfoques ingenuos.

1. Introducción

La programación competitiva es un campo fundamental para desarrollar habilidades de resolución de problemas, aplicando conocimientos de algoritmos y estructuras de datos. Este informe analiza tres problemas seleccionados de la plataforma Vjudge.net: Pay the Price, Determine it, y Murcia's Skyline, los cuales abarcan técnicas avanzadas de programación dinámica y estrategias de optimización. El objetivo es aplicar y contrastar dichas técnicas, evaluando su eficiencia y aplicabilidad en contextos prácticos.

- La Sección 2 presenta el Marco Teórico Conceptual
- La Sección 3 describe el Diseño Experimental
- La Sección 4 muestra los Resultados
- La Sección 5 expresa las Conclusiones,
- La seccion 6 muestra las Referencias Bibliográficas
- La seccion 7 muestra los Anexos e informacion relevante.

2. Marco Teórico Conceptual

La programación dinámica es una técnica clave para resolver problemas que pueden ser descompuestos en subproblemas más pequeños. En este contexto, se han seleccionado tres problemas específicos de vjudge.net con el fin de aplicar y reforzar el aprendizaje en este tipo de metodologías. Estos problemas presentan características que permiten explorar la optimización mediante el almacenamiento de resultados parciales para reducir la complejidad computacional. El presente informe detalla el análisis de cada problema, los métodos empleados en su resolución, y los resultados obtenidos. Para resolver los problemas seleccionados, es fundamental el uso de programación dinámica, técnica introducida por Bellman (1952), que optimiza problemas dividiéndolos en subproblemas. En este informe, se utilizan los conceptos de:

- Subproblemas: División del problema en partes más pequeñas para abordarlos de forma incremental.
- Relaciones Recursivas: Fórmulas que permiten definir la solución de un problema en función de soluciones anteriores.
- Memoización: Técnica para almacenar los resultados de subproblemas y evitar cálculos redundantes.
- Complejidad Temporal y Espacial: Evaluación del tiempo y espacio requerido por cada algoritmo, lo cual es crucial en la programación competitiva.

Para cada problema, se empleará el método SRTBOT, que guía la implementación de programación dinámica mediante la identificación de subproblemas, la formulación de relaciones recursivas, la definición de casos base, y la aplicación de memoización.

3. Diseño Experimental

3.1. Objetivos

Los objetivos de este trabajo son:

- Reforzar el conocimiento en programación dinámica.
- Aplicar métodos de programación dinámica para resolver problemas de vjudge.net.

3.2. Actividades

Virtual Judge

Virtual Judge is not a conventional online judge. It retrieves problems from other standard online judges and simulates submissions to them. Its goal is to facilitate hosting contests when test data is not available.

Currently, Virtual Judge supports the following online judges:

POJ	ZOJ	UVa 2010-05-26 ~	SGU
URAL	HUST	SPOJ	HDU
HYSBZ	UVA	CodeForces	Z-Trening
Aizu	LightOJ	UESTC	NBUT
FZU	CSU	SCU	ACdream
CodeChef	OpenJudge	Kattis	HihoCoder
HIT	HRBUST	acm hipt	AtCoder
HackerRank	51Nod	TopCoder	EOlymp
计蒜客	LibreOJ	UniversalOJ	黑暗爆炸
CSG	DMOJ	Toph	洛谷
Baekjoon	QOJ	CSES	USACO
oj.uz	Yosupo	yukicoder	VNOJ
TLX	BZOJ	Kilonova	Szkopuł
CSAcademy	牛客		

● Service status

El

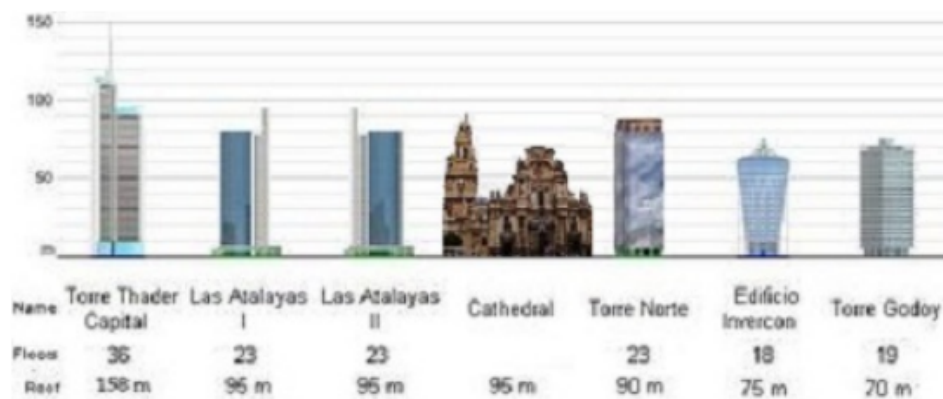
estudiante deberá realizar las siguientes acciones.

1. Crear un usuario en <http://vjudge.net> e indicar en este paso el nombre de usuario utilizado.
2. Seleccionar aleatoriamente tres problemas de la lista disponible en <http://bit.ly/3UxdCVL>. Note que debe loguearse en la plataforma para poder ver los problemas.
3. Deberá diseñar una solución utilizando la técnica SRTBOT para cada problema.
4. Deberá mostrar el pseudocódigo recursivo resultante sin y con memoización.
5. Deberá incluir el código en C++ que resulta del modelo SRTBOT.
6. Deberá anexar el PDF del código aceptado por la plataforma <http://vjudge.net> al final del artículo.

4. Resultados

4.1. Problema 11790 – Murcia's Skyline

Murcia's skyline is growing up very fast. Since the 15th century, it used to be dominated by the profile of its Baroque Cathedral. But nowadays, new skyscrapers are rising in Murcian *huerta*.

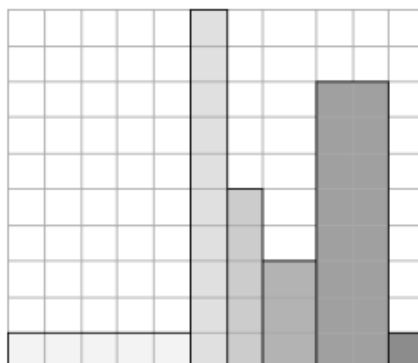


Some people say that if look at the skyline from left to right, you can observe an increasing profile; but other people say the profile is decreasing.

Looking at Murcia's skyline from left to right, we have a series of N buildings. Each building has its own height and width. You have to discover if the skyline is increasing or decreasing.

We say the skyline is **increasing** if the longest increasing subsequence of buildings is bigger or equal than the longest decreasing subsequence of buildings; in other case, we say it is **decreasing**. A **subsequence** is a subset of the original sequence, in the same order. The **length** of a subsequence of buildings is the sum of the widths of its elements.

For example, assuming we have six buildings of heights: 10, 100, 50, 30, 80, 10; and widths: 50, 10, 10, 15, 20, 10; then we have an increasing subsequence of 3 buildings and total length 85, and a decreasing subsequence of 1 building and total length 50 (also, there is a decreasing subsequence of 4 buildings and length 45). So, in this case, we say that the skyline is increasing. You can see this example below.



Input

The first line of the input contains an integer indicating the number of test cases.

For each test case, the first line contains a single integer, N , indicating the number of buildings of the skyline. Then, there are two lines, each with N integers separated by blank spaces. The first line indicates the heights of the buildings, from left to right. The second line indicates the widths of the buildings, also from left to right.

Output

For each test case, the output should contain a line. If the skyline is increasing, the format will be:

Case i . Increasing (A). Decreasing (B).

If the skyline is decreasing, the format will be:

Case i . Decreasing (B). Increasing (A).

where i is the number of the corresponding test case (starting with 1), A is the length of the longest increasing subsequence, and B is the length of the longest decreasing subsequence.

Sample Input

```
3
6
10 100 50 30 80 10
50 10 10 15 20 10
4
30 20 20 10
20 30 40 50
3
80 80 80
15 25 20
```

Sample Output

```
Case 1. Increasing (85). Decreasing (50).
Case 2. Decreasing (110). Increasing (50).
Case 3. Increasing (25). Decreasing (25).
```

Explicación: El problema "Murcia's Skyline" plantea determinar si el perfil de edificios de Murcia, observado de izquierda a derecha, tiene una tendencia creciente o decreciente. Dado un conjunto de edificios con alturas y anchos específicos, el perfil se considera creciente si la subsecuencia creciente de mayor longitud (en términos de suma de anchos) es igual o mayor a la subsecuencia decreciente de mayor longitud; de lo contrario, se considera decreciente. La entrada del problema incluye múltiples casos, donde cada caso especifica el número de edificios, seguido por listas de alturas y anchos de cada edificio. La salida indica si el perfil es creciente o decreciente, junto con las longitudes de ambas subsecuencias.

Algoritmo sin memoizacion:

Java ▼

Leer el número total de pruebas.

Para cada prueba:

Leer el número de elementos `N` y las listas de `alturas` y `pesos`.

Crear dos arreglos `lis` y `lds` de longitud `N`:

Inicializar cada `lis[i]` y `lds[i]` con `pesos[i]`.

Para cada elemento `i` en el rango `0` hasta `N-1`:

Para cada elemento `j` en el rango `0` hasta `i-1`:

Si `alturas[i] > alturas[j]`:

Calcular `lis[i]` como el mayor valor entre `lis[i]` y
`lis[j] + pesos[i]`.

Si `alturas[i] < alturas[j]`:

Calcular `lds[i]` como el mayor valor entre `lds[i]` y
`lds[j] + pesos[i]`.

Encontrar el máximo valor en `lis` y guardarlo en `creciente`.

Encontrar el máximo valor en `lds` y guardarlo en `decreciente`.

Si `creciente` es mayor o igual a `decreciente`:

Imprimir "Increasing (creciente). Decreasing (decreciente)"

Si no:

Imprimir "Decreasing (decreciente). Increasing (creciente)"

Algoritmo con memoización:

```

C/C++
INICIO
// Definir variables globales para el número máximo de casos y las listas de
alturas y pesos.
MAX = 100000 + 7
ALTURAS[MAX]
PESOS[MAX]
MEM_LIS[MAX] // Memorización para subsecuencia creciente
MEM_LDS[MAX] // Memorización para subsecuencia decreciente

// Función recursiva para encontrar la subsecuencia creciente más larga
(LIS)
FUNCION LIS(i):
    SI MEM_LIS[i] NO ES -1: // Si el valor ya fue calculado, devolver el
valor memorizado
        DEVOLVER MEM_LIS[i]

    MEJOR = PESOS[i] // La mejor subsecuencia incluye al menos el peso de i
    PARA cada j DESDE 0 HASTA i-1: // Comparar el índice i con los
anteriores
        SI ALTURAS[j] < ALTURAS[i]: // Si la altura en j es menor que la de
i, es una subsecuencia creciente
            MEJOR = MAX(MEJOR, LIS(j) + PESOS[i]) // Tomar la mejor
subsecuencia sumando el peso actual
    MEM_LIS[i] = MEJOR // Almacenar el resultado para evitar cálculos
repetidos
    DEVOLVER MEJOR

// Función recursiva para encontrar la subsecuencia decreciente más larga
(LDS)
FUNCION LDS(i):
    SI MEM_LDS[i] NO ES -1: // Si el valor ya fue calculado, devolver el
valor memorizado
        DEVOLVER MEM_LDS[i]

    MEJOR = PESOS[i] // La mejor subsecuencia incluye al menos el peso de i
    PARA cada j DESDE 0 HASTA i-1: // Comparar el índice i con los
anteriores
        SI ALTURAS[j] > ALTURAS[i]: // Si la altura en j es mayor que la de
i, es una subsecuencia decreciente

```

```

        MEJOR = MAX(MEJOR, LDS(j) + PESOS[i]) // Tomar la mejor
subsecuencia sumando el peso actual
        MEM_LDS[i] = MEJOR // Almacenar el resultado para evitar cálculos
repetidos
        DEVOLVER MEJOR

// Función principal que procesa múltiples casos de prueba
FUNCION MAIN():
    LEER el número de pruebas
    PARA cada caso DE 1 HASTA el número de pruebas:
        LEER el número de elementos (numCasos)

        LEER la lista de ALTURAS[numCasos] // Leer las alturas de las
personas
        LEER la lista de PESOS[numCasos] // Leer los pesos correspondientes

        INICIALIZAR MEM_LIS con -1 // Inicializar la memorización de LIS con
-1
        INICIALIZAR MEM_LDS con -1 // Inicializar la memorización de LDS con
-1

        CRECER = 0 // Variable para almacenar la longitud de la subsecuencia
creciente más larga
        DECRESCER = 0 // Variable para almacenar la longitud de la
subsecuencia decreciente más larga

        PARA cada i DESDE 0 HASTA numCasos-1:
            CRECER = MAX(CRECER, LIS(i)) // Obtener el máximo valor de LIS
            DECRESCER = MAX(DECRESCER, LDS(i)) // Obtener el máximo valor de
LDS

        SI CRECER ES MAYOR O IGUAL A DECRESCER:
            IMPRIMIR "Caso X. Increasing (CRECER). Decreasing (DECRESCER).".
        SINO:
            IMPRIMIR "Caso X. Decreasing (DECRESCER). Increasing (CRECER).".

FIN

```


Código:

```

1 #include <bits/stdc++.h>
2
3 #define todos(x) x.begin(), x.end()
4 #define llenar(a, v) memset(a, v, sizeof a)
5 using namespace std;
6
7 typedef long long Long;
8 typedef vector<int> vi;
9
10 const int MAX = 1e5 + 7;
11 int alturas[MAX], pesos[MAX];
12 int memLIS[MAX], memLDS[MAX];
13
14 // Función recursiva con memoización para LIS
15 int LIS(int i) {
16     if (memLIS[i] != -1) return memLIS[i]; // Si ya se calculó, devolver el valor almacenado
17     int mejor = pesos[i]; // Base: la suma incluye al menos el peso actual
18     for (int j = 0; j < i; ++j) {
19         if (alturas[j] < alturas[i]) { // Subsecuencia creciente
20             mejor = max(mejor, LIS(j) + pesos[i]);
21         }
22     }
23     return memLIS[i] = mejor;
24 }
25
26 // Función recursiva con memoización para LDS
27 int LDS(int i) {
28     if (memLDS[i] != -1) return memLDS[i]; // Si ya se calculó, devolver el valor almacenado
29     int mejor = pesos[i]; // Base: la suma incluye al menos el peso actual
30     for (int j = 0; j < i; ++j) {
31         if (alturas[j] > alturas[i]) { // Subsecuencia decreciente
32             mejor = max(mejor, LDS(j) + pesos[i]);
33         }
34     }
35     return memLDS[i] = mejor;
36 }
37
38 int main() {
39     int pruebas;
40     cin >> pruebas;
41     for (int caso = 1; caso <= pruebas; ++caso) {
42         int numCasos;
43         cin >> numCasos;
44
45         for (int i = 0; i < numCasos; ++i) cin >> alturas[i];
46         for (int i = 0; i < numCasos; ++i) cin >> pesos[i];
47
48         llenar(memLIS, -1); // Inicializamos la memoización para LIS
49         llenar(memLDS, -1); // Inicializamos la memoización para LDS
50
51         int creciente = 0, decreciente = 0;
52         for (int i = 0; i < numCasos; ++i) {
53             creciente = max(creciente, LIS(i));
54             decreciente = max(decreciente, LDS(i));
55         }
56
57         if (creciente >= decreciente)
58             printf("Case_%d: Increasing_%d Decreasing_%d\n", caso, creciente, decreciente);
59         else
60             printf("Case_%d: Decreasing_%d Increasing_%d\n", caso, decreciente, creciente);
61     }
62     return 0;
63 }

```

4.2. Problema 10520 – Determine it

Consider that $a_{i,j}$ is defined as:

$$a_{i,j} = \begin{cases} \begin{cases} \max_{i < k \leq n} (a_{k,1} + a_{k,j}) & , i < n \\ 0 & , i = n \end{cases} + \begin{cases} \max_{1 \leq k < j} (a_{i,k} + a_{n,k}) & , j > 0 \\ 0 & , j = 0 \end{cases} & , i \geq j \\ \max_{i \leq k < j} (a_{i,k} + a_{k+1,j}) & , i < j \end{cases}$$

You are to calculate the value of $a_{1,n}$ on the basis of the values of n and $a_{n,1}$.

Input

The input consists of several test cases. Each Test case consists of two integers n ($0 < n < 20$) and $a_{n,1}$ ($0 < a_{n,1} < 500$).

Output

For each test case your correct program should print the value of $a_{1,n}$ in a separate line.

Sample Input

```
5 10
4 1
6 13
```

Sample Output

```
1140
42
3770
```

Explicacion: El problema consiste en calcular un valor óptimo basado en una serie de condiciones y reglas de maximización entre elementos de una estructura. A partir de dos parámetros de entrada, se aplican reglas que combinan valores previos en distintos rangos para obtener el resultado final. Este enfoque requiere optimización y reutilización de cálculos anteriores, utilizando programación dinámica para encontrar la mejor solución en cada caso de prueba.

Algoritmo sin memoizacion:

C/C++ ▼

```
funcion resolver(i, j):
    si i == tamano y j == 1:
        retornar valor

    resultado = 0

    si i < j:
        para cada k desde i hasta j-1:
            resultado = max(resultado, resolver(i, k) + resolver(k + 1, j))
    sino:
        r1 = 0
        r2 = 0
        si i < tamano:
            para cada k desde i + 1 hasta tamano:
                r1 = max(r1, resolver(k, 1) + resolver(k, j))
        si j > 0:
            para cada k desde 1 hasta j-1:
                r2 = max(r2, resolver(i, k) + resolver(tamano, k))
        resultado = r1 + r2

    retornar resultado

funcion main():
    mientras haya entrada (tamano, valor):
        imprimir resolver(1, tamano)
```

Algoritmo con memoización:

```
C/C++
funcion resolver(i, j):
    si i == tamano y j == 1:
        retornar valor

    si Visitado[i][j]:
        retornar DP[i][j]

    Visitado[i][j] = verdadero
    DP[i][j] = 0

    si i < j:
        para cada k desde i hasta j-1:
            DP[i][j] = max(DP[i][j], resolver(i, k) + resolver(k + 1, j))
    sino:
        r1 = 0
        r2 = 0
        si i < tamano:
            para cada k desde i + 1 hasta tamano:
                r1 = max(r1, resolver(k, 1) + resolver(k, j))
        si j > 0:
            para cada k desde 1 hasta j-1:
                r2 = max(r2, resolver(i, k) + resolver(tamano, k))
        DP[i][j] = r1 + r2

    retornar DP[i][j]

funcion main():
    mientras haya entrada (tamano, valor):
        limpiar matriz Visitado a falso
        imprimir resolver(1, tamano)
```

Código:

```

1 #include <bits/stdc++.h>
2 #define pb push_back
3 #define all(x) x.begin(),x.end()
4 #define ms(a,v) memset(a,v,sizeof a)
5 #define II ({int a; scanf("%d", &a); a;})
6 #define LL ({Longo a; scanf("%lld", &a); a;})
7 #define DD ({double a; scanf("%lf", &a); a;})
8 #define EPS 1e-10
9 #define pi 3.1415926535897932384626433832795
10 using namespace std;
11
12 typedef long long Longo;
13 typedef unsigned long long ull;
14 typedef vector<int> vi ;
15 typedef set<int> si;
16 typedef vector<Longo> vl;
17 typedef pair<int,int> pii;
18 typedef pair<Longo,Longo> pll;
19 typedef pair<double,double> pdd;
20
21 #define forab(i, a, b) for (__typeof (b) i = (a) ; i <= b ; ++i)
22 #define rep(i, n) forab (i, 0, (n) - 1)
23 #define For(i, n) forab (i, 1, n)
24 #define rofba(i, a, b) for (__typeof (b)i = (b) ; i >= a ; --i)
25 #define per(i, n) rofba (i, 0, (n) - 1)
26 #define rof(i, n) rofba (i, 1, n)
27 #define forstl(i, s) for (__typeof ((s).end ()) i = (s).begin (); i != (s).end (); ++i)
28
29 const int MX = 507;
30 const int INF = 1e8 + 7;
31
32 Longo dp[MX][27];
33 bool Visitado[MX][27];
34 int tamano, valor;
35
36 Longo resolver(int i, int j) {
37     if (i == tamano && j == 1) return valor;
38     Longo &retorno = dp[i][j];
39     if (Visitado[i][j]) return retorno;
40     Visitado[i][j] = true;
41     retorno = 0;
42     if (i < j) {
43         for (int k = i; k < j; k++)
44             retorno = max(retorno, resolver(i, k) + resolver(k + 1, j));
45     } else {
46         Longo r1 = 0, r2 = 0;
47         if (i < tamano) {
48             for (int k = i + 1; k <= tamano; k++)
49                 r1 = max(r1, resolver(k, 1) + resolver(k, j));
50         }
51
52         if (j > 0) {
53             for (int k = 1; k < j; k++)
54                 r2 = max(r2, resolver(i, k) + resolver(tamano, k));
55         }
56         retorno = r1 + r2;
57     }
58     return retorno;
59 }
60 }
61
62 int main() {
63     #ifdef LOCAL
64         freopen("in.txt", "r", stdin);
65     #endif
66     while (scanf("%d%d", &tamano, &valor) != EOF) {
67         ms(Visitado, false);
68         printf("%lld\n", resolver(1, tamano));
69     }
70 }

```

4.3. Problema 10313 – Pay the Price

In ancient days there was a country whose people had very interesting habits. Some of them were lazy, some were very rich, some were very poor and some were miser. Obviously, some of the rich were miser (A poor was never miser as he had little to spend) and lazy but the poor were lazy as well (As the poor were lazy they remained poor forever). The following things were true for that country

- a) As the rich were miser, no things price was more than 300 dollars (Yes! their currency was dollar).
- b) As all people were lazy, the price of everything was integer (There were no cents and so beggars always earned at least one dollar)
- c) The values of the coins were from 1 to 300 dollars, so that the rich (who were idle) could pay any price with a single coin.

Your job is to find out in how many ways one could pay a certain price using a limited number of coins (Note that the number of coins paid is limited but not the value or source. I mean there was infinite number of coins of all values). For example, by using three coins one can pay six dollars in 3 ways, $1+1+4$, $1+2+3$, and $2+2+2$. Similarly, one can pay 6 dollars using 6 coins or less in 11 ways.

Input

The input file contains several lines of input. Each line of input may contain 1, 2 or 3 integers. The first integer is always N ($0 \leq N \leq 300$), the dollar amount to be paid. All other integers are less than 1001 and non-negative.

Output

For each line of input you should output a single integer.

When there is only one integer N as input, you should output in how many ways N dollars can be paid.

When there are two integers N and $L1$ as input, then you should output in how many ways N dollars can be paid using $L1$ or less coins.

When there are three integers N , $L1$ and $L2$ as input, then you should output in how many ways N dollars can be paid using $L1, L1 + 1, \dots, L2$ coins (summing all together). Remember that $L1$ is not greater than $L2$.

Sample Input

```
6
6 3
6 2 5
6 1 6
```

Sample Output

```
11
7
9
11
```

Explicación: El problema trata de encontrar la cantidad de formas en que se puede pagar una cantidad específica de dólares usando monedas de distintos valores. En un país antiguo, la moneda tenía valores entre 1 y 300 dólares, y la gente podía usar cualquier cantidad de monedas para alcanzar un precio exacto, limitado solo por el número de monedas disponibles. La entrada varía: puede ser solo la cantidad a pagar, la cantidad y el número máximo de monedas a utilizar, o la cantidad y un rango de monedas permitido. El objetivo es calcular el número de combinaciones posibles que sumen el valor deseado usando las restricciones indicadas, empleando programación dinámica para optimizar el cálculo de combinaciones posibles.

Algoritmo sin memoizacion:

```
C/C++
// Función para contar las maneras de pagar una cantidad
contarManeras(cantidad, maxMoneda):
    crear un arreglo dp de tamaño (cantidad + 1) inicializado en 0
    dp[0] = 1 // Una forma de hacer 0: no usar monedas

    para moneda desde 1 hasta maxMoneda:
        para i desde moneda hasta cantidad:

            dp[i] = dp[i] + dp[i - moneda] // Actualizar maneras de hacer 'i'
    retornar dp[cantidad] // Número total de maneras de hacer la cantidad
// Función principal para procesar la entrada y calcular el resultado
función principal: W
    leer la entrada s (una línea con números)
    dividir la entrada en una lista de números, guardarlos en numerosEntrada
    si longitud de numerosEntrada es 1:
        p = 0
        q = numerosEntrada[0]
    si longitud de numerosEntrada es 2:
        p = 0 q = numerosEntrada[1]
    si longitud de numerosEntrada es 3:
        p = numerosEntrada[1]
        q = numerosEntrada[2]
        n = numerosEntrada[0]
    si n == 0: imprimir 1 // Si la cantidad es 0, imprimir 1
    sino: // Calcular el número de maneras de pagar entre p y q
        resultado = contarManeras(n, min(300, q)) - contarManeras(n, min(300, max(0,
        p - 1))) imprimir resultado // Imprimir el resultado fin de la función
principal
```

Algoritmo con memoización:

```

C/C++
| // Verificar si el subproblema ya fue calculado y almacenado en memo
  si clave está en memo:
      retornar memo[clave] // Retornar el resultado ya memorizado

  // Recursión para contar las maneras:
  resultado = contarManeras(cantidad, maxMoneda - 1) +
              (si cantidad >= maxMoneda: contarManeras(cantidad -
maxMoneda, maxMoneda) : 0)

  // Almacenar el resultado en memo
  memo[clave] = resultado

  retornar resultado

// Función principal para procesar la entrada y calcular el resultado
función principal:
  leer la entrada s (una línea con números)
  dividir la entrada en una lista de números, guardarlos en numerosEntrada

  si longitud de numerosEntrada es 1:
      p = 0
      q = numerosEntrada[0]
  si longitud de numerosEntrada es 2:
      p = 0
      q = numerosEntrada[1]
  si longitud de numerosEntrada es 3:
      p = numerosEntrada[1]
      q = numerosEntrada[2]

  n = numerosEntrada[0]

  si n == 0:
      imprimir 1 // Si la cantidad es 0, imprimir 1
  sino:
      // Calcular el número de maneras de pagar entre p y q
      resultado = contarManeras(n, min(300, q)) - contarManeras(n,
min(300, max(0, p - 1)))
      imprimir resultado // Imprimir el resultado

```

Código:

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4 #include <vector>
5 #include <unordered_map>
6 using namespace std;
7
8 static const int MAXIMO_VALOR = 300;
9
10 typedef unsigned long long ull_t;
11 unordered_map<string, ull_t> memo; // Memorización para evitar recalculer subproblemas
12
13 // Función recursiva con memorización para contar las maneras de pagar
14 ull_t contarManeras(int cantidad, int maxMoneda) {
15     if (cantidad == 0) return 1; // Caso base: hay una sola forma de pagar 0, no usar ninguna moneda
16     if (maxMoneda == 0) return 0; // No hay formas si no tenemos monedas
17     string clave = to_string(cantidad) + "," + to_string(maxMoneda); // Clave única para cada subproblema
18
19     // Si el subproblema ya fue calculado, devolver el resultado memorizado
20     if (memo.find(clave) != memo.end()) {
21         return memo[clave];
22     }
23
24     // Recursión: contar las maneras de pagar la cantidad con monedas hasta maxMoneda
25     ull_t resultado = contarManeras(cantidad, maxMoneda - 1) + (cantidad >= maxMoneda ? contarManeras(cantidad - maxMoneda, maxMoneda) : 0);
26
27     // Memorizamos el resultado antes de devolverlo
28     memo[clave] = resultado;
29     return resultado;
30 }
31
32 int main()
33 {
34     string s;
35     while (getline(cin, s))
36     {
37         vector<int> numerosEntrada;
38         stringstream ss(s);
39         int valor;
40         while (ss >> valor)
41             numerosEntrada.push_back(valor);
42
43         int limiteInferior, limiteSuperior;
44         switch(numerosEntrada.size())
45         {
46             case 1:
47                 limiteInferior = 0;
48                 limiteSuperior = numerosEntrada[0];
49                 break;
50             case 2:
51                 limiteInferior = 0;
52                 limiteSuperior = numerosEntrada[1];
53                 break;
54             default:
55                 limiteInferior = numerosEntrada[1];
56                 limiteSuperior = numerosEntrada[2];
57         }
58
59         // Casos especiales cuando la cantidad es 0
60         if (numerosEntrada[0] == 0)
61             cout << "1" << endl;
62         else
63             cout << contarManeras(numerosEntrada[0], min(300, limiteSuperior)) - contarManeras(numerosEntrada[0], min(300, max(0, limiteInferior))) << endl;
64     }
65     return 0;
66 }

```

5. Conclusiones

La resolución de los tres problemas con programación dinámica se basa en descomponer el problema en subproblemas más pequeños y almacenar resultados parciales para evitar cálculos redundantes. Para cada problema, se identificaron patrones que permiten reutilizar combinaciones previas, ya sea en la

suma de valores específicos, en secuencias crecientes o decrecientes, o en combinaciones de monedas. La programación dinámica influyó en la eficiencia de las soluciones, reduciendo significativamente la complejidad temporal al evitar cálculos repetitivos y permitiendo manejar restricciones complejas con mayor rapidez y precisión. Esta técnica no solo optimiza el rendimiento sino que facilita la resolución de problemas que implican maximización o conteo de combinaciones, lo que sería inviable con enfoques puramente recursivos o iterativos sin almacenamiento intermedio.










6. Referencias bibliograficas

- <https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/resources/lecture-15-dynamic-programming-part-1-srtbot-fib-dags-bowling/>
- <https://www.youtube.com/watch?v=e5zaZEfHsIst=79s>
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

7. Anexos

Capturas de los problemas:

- Pay the Price UVA - 10313
- Murcia's Skyline UVA - 11790
- Determine it UVA - 10520

Korkbook User Group Forum Help +					
Previous 1 2 3 4 5 ... Next Filter Reset					
	OJ	Prob	Title	Source	Solved
 	UVA	10313	Pay the Price		 x 2095
 	UVA	11790	Murcia's Skyline		 x 3775
 	UVA	10520	Determine it		 x 1102

#56090501 | Raul_Almanza's solution for [UVA-10313]



Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	360ms	2175	C++11 5.3.0	2024-11-16 12:51:04	<input checked="" type="checkbox"/>	<input type="checkbox"/>	29967942

```
1  #include <iostream>
2  #include <string>
3  #include <sstream>
4  #include <vector>
5  #include <unordered_map>
6  using namespace std;
7
8  static const int MAXIMO_VALOR = 300;
9
10 typedef unsigned long long ull_t;
11 unordered_map<string, ull_t> memo; // Memorización para evitar recalcular subproblemas
12
13 // Función recursiva con memorización para contar las maneras de pagar
14 ull_t contarManeras(int cantidad, int maxMoneda) {
15     if (cantidad == 0) return 1; // Caso base: hay una sola forma de pagar 0, no usar ninguna moneda
16     if (maxMoneda == 0) return 0; // No hay formas si no tenemos monedas
17     string clave = to_string(cantidad) + "," + to_string(maxMoneda); // Clave única para cada
18     subproblema
19
20     // Si el subproblema ya fue calculado, devolver el resultado memorizado
21     if (memo.find(clave) != memo.end()) {
22         return memo[clave];
23     }
24
25     // Recursión: contar las maneras de pagar la cantidad con monedas hasta maxMoneda
26     ull_t resultado = contarManeras(cantidad, maxMoneda - 1) + (cantidad >= maxMoneda ?
27     contarManeras(cantidad - maxMoneda, maxMoneda) : 0);
28
29     // Memorizamos el resultado antes de devolverlo
30     memo[clave] = resultado;
31     return resultado;
32 }
33
34 int main()
35 {
36     string s;
37     while (getline(cin, s))
38     {
39         vector<int> numerosEntrada;
40         stringstream ss(s);
41         int valor;
42         while (ss >> valor)
43             numerosEntrada.push_back(valor);
44
45         int limiteInferior, limiteSuperior;
46         switch(numerosEntrada.size())
47         {
48             case 1:
49                 limiteInferior = 0;
50                 limiteSuperior = numerosEntrada[0];
51                 break;
52             case 2:
53                 limiteInferior = 0;
54                 limiteSuperior = numerosEntrada[1];
55                 break;
56             default:
57                 limiteInferior = numerosEntrada[1];
58                 limiteSuperior = numerosEntrada[2];
59         }
60     }
```

Copy C++

#56060944 | Raul_Almanza's solution for [UVA-11790]



Status	Time	Length	Lang	Submitted	Open	Share text ?	RemoteRunId
Accepted	180ms	2020	C++ 5.3.0	2024-11-15 14:39:17	<input checked="" type="checkbox"/>	<input type="checkbox"/>	29966305

Copy C++

```

1  #include <bits/stdc++.h>
2
3  #define todos(x) x.begin(), x.end()
4  #define llenar(a, v) memset(a, v, sizeof a)
5  using namespace std;
6
7  typedef long long Long;
8  typedef vector<int> vi;
9
10 const int MAX = 1e5 + 7;
11 int alturas[MAX], pesos[MAX];
12 int memLIS[MAX], memLDS[MAX];
13
14 // Función recursiva con memoización para LIS
15 int LIS(int i) {
16     if (memLIS[i] != -1) return memLIS[i]; // Si ya se calculó, devolver el valor almacenado
17     int mejor = pesos[i]; // Base: la suma incluye al menos el peso actual
18     for (int j = 0; j < i; ++j) {
19         if (alturas[j] < alturas[i]) { // Subsecuencia creciente
20             mejor = max(mejor, LIS(j) + pesos[i]);
21         }
22     }
23     return memLIS[i] = mejor;
24 }
25
26 // Función recursiva con memoización para LDS
27 int LDS(int i) {
28     if (memLDS[i] != -1) return memLDS[i]; // Si ya se calculó, devolver el valor almacenado
29     int mejor = pesos[i]; // Base: la suma incluye al menos el peso actual
30     for (int j = 0; j < i; ++j) {
31         if (alturas[j] > alturas[i]) { // Subsecuencia decreciente
32             mejor = max(mejor, LDS(j) + pesos[i]);
33         }
34     }
35     return memLDS[i] = mejor;
36 }
37
38 int main() {
39     int pruebas;
40     cin >> pruebas;
41     for (int caso = 1; caso <= pruebas; ++caso) {
42         int numCasos;
43         cin >> numCasos;
44
45         for (int i = 0; i < numCasos; ++i) cin >> alturas[i];
46         for (int i = 0; i < numCasos; ++i) cin >> pesos[i];
47
48         llenar(memLIS, -1); // Inicializamos la memoización para LIS
49         llenar(memLDS, -1); // Inicializamos la memoización para LDS
50
51         int creciente = 0, decreciente = 0;
52         for (int i = 0; i < numCasos; ++i) {
53             creciente = max(creciente, LIS(i));
54             decreciente = max(decreciente, LDS(i));
55         }
56
57         if (creciente >= decreciente)
58             printf("Case %d. Increasing (%d). Decreasing (%d).\n", caso, creciente, decreciente);
59         else
60             printf("Case %d. Decreasing (%d). Increasing (%d).\n", caso, decreciente, creciente);
61     }
62 }
```

#55941532 | Raul_Almanza's solution for [UVA-10520]



Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	100ms	1972	C++11 5.3.0	2024-11-12 02:26:36	<input checked="" type="checkbox"/>	Link	29956166

Copy C++

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  #define all(x) x.begin(),x.end()
4  #define ms(a,v) memset(a,v,sizeof a)
5  #define II ({int a; scanf("%d", &a); a;})
6  #define LL ({longo a; scanf("%lld", &a); a;})
7  #define DD ({double a; scanf("%lf", &a); a;})
8  #define EPS 1e-10
9  #define pi 3.1415926535897932384626433832795
10 using namespace std;
11
12 typedef long long Longo;
13 typedef unsigned long long ull;
14 typedef vector<int> vi;
15 typedef set<int> si;
16 typedef vector<Longo> vl;
17 typedef pair<int,int> pii;
18 typedef pair<Longo,Longo> pll;
19 typedef pair<double,double> pdd;
20
21 #define forab(i, a, b) for (__typeof (b) i = (a) ; i <= b ; ++i)
22 #define rep(i, n) forab (i, 0, (n) - 1)
23 #define For(i, n) forab (i, 1, n)
24 #define rofba(i, a, b) for (__typeof (b) i = (b) ; i >= a ; --i)
25 #define per(i, n) rofba (i, 0, (n) - 1)
26 #define rof(i, n) rofba (i, 1, n)
27 #define forstl(i, s) for (__typeof ((s).end ()) i = (s).begin (); i != (s).end (); ++i)
28
29 const int MX = 507;
30 const int INF = 1e8 + 7;
31
32 Longo dp[MX][27];
33 bool Visitado[MX][27];
34 int tamano, valor;
35
36 Longo resolver(int i, int j) {
37     if (i == tamano && j == 1) return valor;
38     Longo &retorno = dp[i][j];
39     if (Visitado[i][j]) return retorno;
40     Visitado[i][j] = true;
41     retorno = 0;
42     if (i < j) {
43         for (int k = i; k < j; k++)
44             retorno = max(retorno, resolver(i, k) + resolver(k + 1, j));
45     } else {
46         Longo r1 = 0, r2 = 0;
47         if (i < tamano) {
48             for (int k = i + 1; k <= tamano; k++)
49                 r1 = max(r1, resolver(k, 1) + resolver(k, j));
50         }
51         if (j > 0) {
52             for (int k = 1; k < j; k++)
53                 r2 = max(r2, resolver(i, k) + resolver(tamano, k));
54         }
55         retorno = r1 + r2;
56     }
57     return retorno;
58 }
59
60 int main() {

```