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

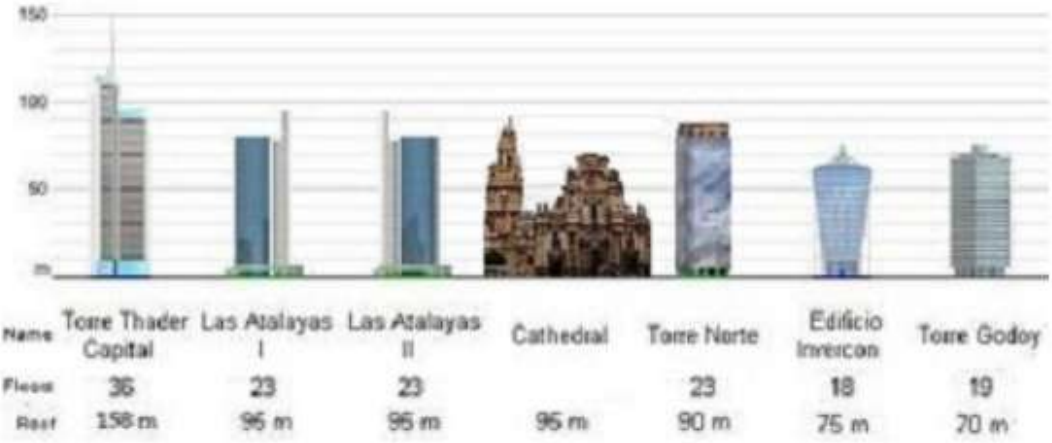
El estudiante deberá realizar las siguientes acciones.

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*.



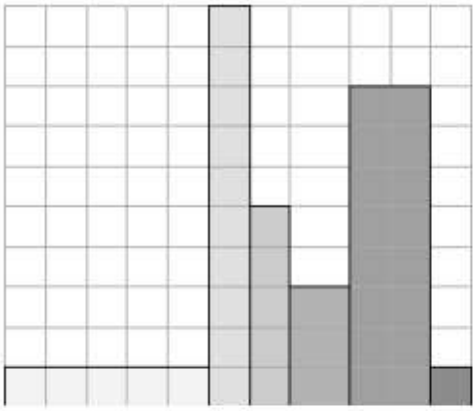
Name	Torre Thader Capital	Las Atalayas I	Las Atalayas II	Cathedral	Torre Norte	Edificio Inveicon	Torre Godoy
Floors	36	23	23		23	18	19
Roof	158 m	96 m	96 m	96 m	90 m	75 m	70 m

Some people say that if look at the skyline from left to right, you can observe and 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).
```

Explicacion: 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 memoizacion:

```
C/C++
Definir función `calcularLIS(i)` para calcular la subsecuencia creciente
desde el elemento `i`:
    Si `lis[i]` ya ha sido calculado, devolver el valor de `lis[i]`.
    Inicializar `lis[i]` con `pesos[i]`.

    Para cada `j` en el rango `0` hasta `i-1`:
        Si `alturas[i] > alturas[j]`:
            Actualizar `lis[i]` como el máximo entre `lis[i]` y
`calcularLIS(j) + pesos[i]`.

    Devolver `lis[i]`.

Definir función `calcularLDS(i)` para calcular la subsecuencia decreciente
desde el elemento `i`:
    Si `lds[i]` ya ha sido calculado, devolver el valor de `lds[i]`.
    Inicializar `lds[i]` con `pesos[i]`.

    Para cada `j` en el rango `0` hasta `i-1`:
        Si `alturas[i] < alturas[j]`:
            Actualizar `lds[i]` como el máximo entre `lds[i]` y
`calcularLDS(j) + pesos[i]`.

    Devolver `lds[i]`.

Para cada prueba:
    Leer los datos de entrada (`N`, `alturas` y `pesos`).

    Inicializar `creciente` y `decreciente` en 0.

    Para cada `i` en el rango `0` hasta `N-1`:
        Calcular `creciente` como el máximo entre `creciente` y
`calcularLIS(i)`.
        Calcular `decreciente` como el máximo entre `decreciente` y
`calcularLDS(i)`.

    Si `creciente` es mayor o igual a `decreciente`:
        Imprimir "Increasing (creciente). Decreasing (decreciente)"
    Si no:
        Imprimir "Decreasing (decreciente). Increasing (creciente)"
```

Código:

```cpp
#include <bits/stdc++.h>

#define agregar push_back
#define todos(x) x.begin(), x.end()
#define llenar(a, v) memset(a, v, sizeof a)
#define primero first
#define segundo second
#define emparejar make_pair
#define leerCaracter getchar
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Long;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef set<int> si;
typedef vector<Long> vl;
typedef pair<int, int> parEntero;
typedef pair<string, int> parStringEntero;
typedef pair<Long, Long> parLong;
typedef pair<double, double> parDouble;

#define paraDesdeHasta(i, desde, hasta) for (__typeof(hasta) i = (desde); i <= hasta; ++i)
#define repetir(i, n) paraDesdeHasta(i, 0, (n) - 1)
#define desdeUno(i, n) paraDesdeHasta(i, 1, n)
#define desdeHastaReverso(i, desde, hasta) for (__typeof(hasta)i = (hasta); i >= desde; --i)
#define repetirReverso(i, n) desdeHastaReverso(i, 0, (n) - 1)
#define desdeUnoReverso(i, n) desdeHastaReverso(i, 1, n)
#define paraCadaElemento(i, s) for (__typeof((s).end()) i = (s).begin(); i != (s).end(); ++i)

// Funciones para leer entrada
int leerInt() { int a; scanf("%d", &a); return a; }
Long leerLong() { Long a; scanf("%lld", &a); return a; }
double leerDouble() { double a; scanf("%lf", &a); return a; }

const int MAX = 1e5 + 7;
int numCasos, alturas[MAX], pesos[MAX], lis[MAX], lds[MAX];

int main() {
    int pruebas = leerInt();
    desdeUno(caso, pruebas) {
        numCasos = leerInt();
        repetir(i, numCasos) alturas[i] = leerInt();
        repetir(i, numCasos) pesos[i] = leerInt();
        repetir(i, numCasos) {
            lis[i] = lds[i] = pesos[i];
            repetir(j, i) {
                if (alturas[i] > alturas[j]) lis[i] = max(lis[i], lis[j] + pesos[i]);
                if (alturas[i] < alturas[j]) lds[i] = max(lds[i], lds[j] + pesos[i]);
            }
        }
        int creciente = 0, decreciente = 0;
        repetir(i, numCasos) {
            creciente = max(creciente, lis[i]);
            decreciente = max(decreciente, lds[i]);
        }
        if (creciente >= decreciente)
            printf("Case␣%d.␣Increasing␣(%d).␣Decreasing␣(%d).\n", caso, creciente, decreciente);
        else
            printf("Case␣%d.␣Decreasing␣(%d).␣Increasing␣(%d).\n", caso, decreciente, creciente);
    }
}
```

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 memoizacion:

```
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:

```cpp
#include <bits/stdc++.h>
#define pb push_back
#define all(x) x.begin(),x.end()
#define ms(a,v) memset(a,v,sizeof a)
#define II ({int a; scanf("%d", &a); a;})
#define LL ({Longo a; scanf("%lld", &a); a;})
#define DD ({double a; scanf("%lf", &a); a;})
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Longo;
typedef unsigned long long ull;
typedef vector<int> vi ;
typedef set<int> si;
typedef vector<Longo> vl;
typedef pair<int,int> pii;
typedef pair<Longo,Longo> pll;
typedef pair<double,double> pdd;

#define forab(i, a, b) for (__typeof (b) i = (a) ; i <= b ; ++i)
#define rep(i, n)      forab (i, 0, (n) - 1)
#define For(i, n)      forab (i, 1, n)
#define rofba(i, a, b) for (__typeof (b)i = (b) ; i >= a ; --i)
#define per(i, n)      rofba (i, 0, (n) - 1)
#define rof(i, n)      rofba (i, 1, n)
#define forstl(i, s)   ↩
for (__typeof ((s).end ()) i = (s).begin (); i != (s).end (); ++i)

const int MX = 507;
const int INF = 1e8 + 7;

Longo dp[MX][27];
bool Visitado[MX][27];
int tamano, valor;

Longo resolver(int i, int j) {
    if (i == tamano && j == 1) return valor;
    Longo &retorno = dp[i][j];
    if (Visitado[i][j]) return retorno;
    Visitado[i][j] = true;
    retorno = 0;
    if (i < j) {
        for (int k = i; k < j; k++)
            retorno = max(retorno, resolver(i, k) + resolver(k + 1, j));
    } else {
        Longo r1 = 0, r2 = 0;
        if (i < tamano) {
            for (int k = i + 1; k <= tamano; k++)
                r1 = max(r1, resolver(k, 1) + resolver(k, j));
        }


        if (j > 0) {
            for (int k = 1; k < j; k++)
                r2 = max(r2, resolver(i, k) + resolver(tamano, k));
        }
        retorno = r1 + r2;
    }
    return retorno;
}

int main() {
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
    #endif
    while (scanf("%d␣%d", &tamano, &valor) != EOF) {
        ms(Visitado, false);
        printf("%lld\n", resolver(1, tamano));
    }
}
```

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 $L1, L1+1, \ldots, L2$ 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
```

Explicacion: 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 memoización

```
C/C++
funcion contar_formas_sin_memorizacion(suma, limite):
    si suma == 0:
        retornar 1   // una forma de obtener la suma 0: no usar ninguna
moneda
    si suma < 0 o limite == 0:
        retornar 0   // no hay formas de obtener una suma negativa o sin
monedas

    // Opciones: incluir Moneda[limite] o no incluirla
    incluir_moneda = contar_formas_sin_memorizacion(suma - Monedas[limite],
limite)
    no_incluir_moneda = contar_formas_sin_memorizacion(suma, limite - 1)

    retornar incluir_moneda + no_incluir_moneda

funcion main():
    leer entrada
    inicializar_monedas()
    si entrada tiene un solo valor:
        imprimir contar_formas_sin_memorizacion(suma, MAX - 7)
    sino si entrada tiene dos valores:
        limite = minimo(limite, MAX - 7)
        imprimir contar_formas_sin_memorizacion(suma, limite)
    sino:
        limite_inferior = minimo(limite_inferior, MAX - 7)
        limite_superior = minimo(limite_superior, MAX - 7)
        imprimir contar_formas_sin_memorizacion(suma, limite_superior) -
contar_formas_sin_memorizacion(suma, limite_inferior - 1)
```

Algoritmo con memoizacion:

```
C/C++
funcion inicializar_monedas():
    para i desde 1 hasta MAX - 7:
        Monedas[i] = i

funcion contar_formas_con_memorizacion(suma, limite):
    crear matriz DP de tamaño [suma + 1][limite + 1]
    inicializar DP[0][0] = 1

    para s desde 0 hasta suma:
        para j desde 1 hasta limite:
            DP[s][j] = DP[s][j-1]   // formas sin incluir Monedas[j]
            si s - Monedas[j] >= 0:
                DP[s][j] += DP[s - Monedas[j]][j]  // formas incluyendo
Monedas[j]

    retornar DP[suma][limite]

funcion main():
    leer entrada
    inicializar_monedas()
    si entrada tiene un solo valor:
        imprimir contar_formas_con_memorizacion(suma, MAX - 7)
    sino si entrada tiene dos valores:
        limite = minimo(limite, MAX - 7)
        imprimir contar_formas_con_memorizacion(suma, limite)
    sino:
        limite_inferior = minimo(limite_inferior, MAX - 7)
        limite_superior = minimo(limite_superior, MAX - 7)
        imprimir contar_formas_con_memorizacion(suma, limite_superior) -
contar_formas_con_memorizacion(suma, limite_inferior - 1)
```

Código:

```cpp
#include <bits/stdc++.h>
#define agregar push_back
#define todos(x) x.begin(), x.end()
#define inicializar(a, v) memset(a, v, sizeof a)
#define leerInt ({int a; scanf("%d", &a); a;})
#define leerLong ({Long a; scanf("%lld", &a); a;})
#define leerDouble ({double a; scanf("%lf", &a); a;})
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Long;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef set<int> si;
typedef vector<Long> vl;
typedef pair<int, int> pii;
typedef pair<Long, Long> pll;
typedef pair<double, double> pdd;

#define forab(i, a, b)  for (__typeof(b) i = (a); i <= b; ++i)
#define rep(i, n)   forab(i, 0, (n) - 1)
#define For(i, n)   forab(i, 1, n)
#define rofba(i, a, b)  for (__typeof(b)i = (b); i >= a; --i)
#define per(i, n)   rofba(i, 0, (n) - 1)
#define rof(i, n)   rofba(i, 1, n)
#define forstl(i, s)  for (__typeof((s).end()) i = (s).begin(); i != (s).end(); ++i)

const int MAX = 307;
const int INF = 1e8 + 7;
int Suma, Limite1, Limite2;
Long dp1[MAX];
Long dp2[MAX][MAX];
int Monedas[MAX];
int Indice;

Long aEntero(string s) {
    Long r = 0;
    istringstream sin(s); sin >> r;
    return r;
}

vector<string> dividir(string s) {
    vector<string> resultado;
    istringstream ss(s);
    string token;
    while (ss >> token) resultado.agregar(token);
    return resultado;
}

char Cadena[MAX];

void inicializarValores() {
    vector<string> resultado = dividir(Cadena);
    Indice = resultado.size();
    if (Indice == 1) {
        Suma = aEntero(resultado[0]);
    }
    else if (Indice == 2) {
        Suma = aEntero(resultado[0]);
        Limite1 = aEntero(resultado[1]);
    } else {
        Suma = aEntero(resultado[0]);
        Limite1 = aEntero(resultado[1]);
        Limite2 = aEntero(resultado[2]);
    }
}

void preparar() {
    For(i, MAX - 7) Monedas[i] = i;
    inicializar(dp1, 0);
    dp1[0] = 1;
    for (int i = 1; i <= MAX - 7; i++)
        for (int j = 0; j + Monedas[i] < MAX; j++)
            if (dp1[j]) dp1[j + Monedas[i]] += dp1[j];

    inicializar(dp2, 0);
    dp2[0][0] = 1;
    for (int s = 0; s <= MAX - 7; s++) {
        for (int j = 1; j <= MAX - 7; j++) {
            dp2[s][j] += dp2[s][j - 1];
            if (s - Monedas[j] >= 0)
                dp2[s][j] += dp2[s - Monedas[j]][j];
        }
    }
```

Código:

```
int main() {
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
    #endif
    preparar();
    while (gets(Cadena)) {
        inicializarValores();
        if (Indice == 1) printf("%lld\n", dp1[Suma]);
        else if (Indice == 2) {
            Limite1 = min(Limite1, 300);
            printf("%lld\n", dp2[Suma][Limite1]);
        } else {
            Limite1 = min(Limite1, 300);
            Limite2 = min(Limite2, 300);
            printf("%lld\n", dp2[Suma][Limite2] - dp2[Suma][Limite1 - 1]);
        }
    }
}
```

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.

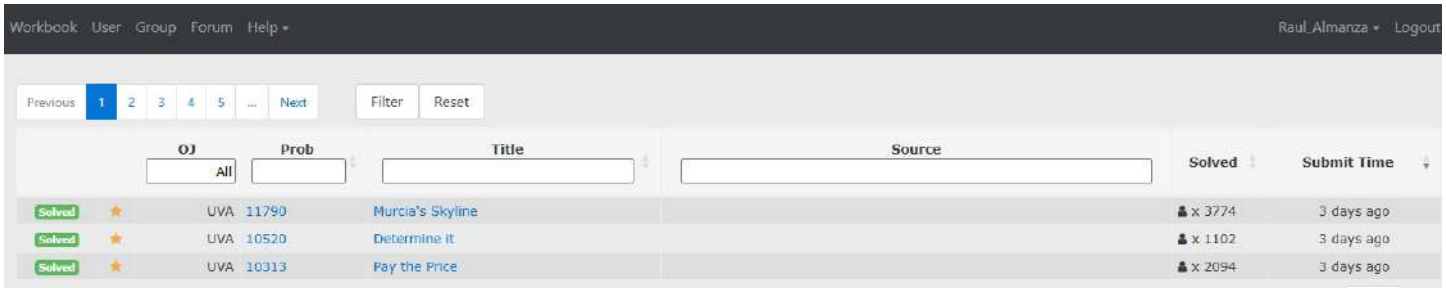
6. Referencias bibliograficas

- https://ocw.mit.edu/courses/6-006-introduction-to-algorithms-spring-2020/resources/lecture-15-dynamic-progra 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:

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



#55942094 | Raul_Almanza's solution for [UVA-11790]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	70ms	2324	C++11 5.3.0	2024-11-12 02:43:40	☑	Link	29956210

Copy C++

```cpp
#include <bits/stdc++.h>

#define agregar push_back
#define todos(x) x.begin(), x.end()
#define llenar(a, v) memset(a, v, sizeof a)
#define primero first
#define segundo second
#define emparejar make_pair
#define leerCaracter getchar
#define EPS 1e-10
#define pi 3.14159265358979323846264338327950
using namespace std;

typedef long long Long;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef set<int> si;
typedef vector<Long> vl;
typedef pair<int, int> parEntero;
typedef pair<string, int> parStringEntero;
typedef pair<Long, Long> parLong;
typedef pair<double, double> parDouble;

#define paraDesdeHasta(i, desde, hasta) for (__typeof(hasta) i = (desde); i <= hasta; ++i)
#define repetir(i, n) paraDesdeHasta(i, 0, (n) - 1)
#define desdeUno(i, n) paraDesdeHasta(i, 1, n)
#define desdeHastaReverso(i, desde, hasta) for (__typeof(hasta)i = (hasta); i >= desde; --i)
#define repetirReverso(i, n) desdeHastaReverso(i, 0, (n) - 1)
#define desdeUnoReverso(i, n) desdeHastaReverso(i, 1, n)
#define paraCadaElemento(i, s) for (__typeof((s).end()) i = (s).begin(); i != (s).end(); ++i)

// Funciones para leer entrada
int leerInt() { int a; scanf("%d", &a); return a; }
Long leerLong() { Long a; scanf("%lld", &a); return a; }
double leerDouble() { double a; scanf("%lf", &a); return a; }

const int MAX = 1e5 + 7;
int numCasos, alturas[MAX], pesos[MAX], lis[MAX], lds[MAX];

int main() {
    int pruebas = leerInt();
    desdeUno(caso, pruebas) {
        numCasos = leerInt();
        repetir(i, numCasos) alturas[i] = leerInt();
        repetir(i, numCasos) pesos[i] = leerInt();
        repetir(i, numCasos) {
            lis[i] = lds[i] = pesos[i];
            repetir(j, i) {
                if (alturas[i] > alturas[j]) lis[i] = max(lis[i], lis[j] + pesos[i]);
                if (alturas[i] < alturas[j]) lds[i] = max(lds[i], lds[j] + pesos[i]);
            }
        }
        int creciente = 0, decreciente = 0;
        repetir(i, numCasos) {
            creciente = max(creciente, lis[i]);
            decreciente = max(decreciente, lds[i]);
        }
        if (creciente >= decreciente)
            printf("Case %d. Increasing (%d). Decreasing (%d).\n", caso, creciente, decreciente);
        else
```

#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	☑	Link	29956166

Copy C++

```cpp
#include <bits/stdc++.h>
#define pb push_back
#define all(x) x.begin(),x.end()
#define ms(a,v) memset(a,v,sizeof a)
#define II ({int a; scanf("%d", &a); a;})
#define LL ({Longo a; scanf("%lld", &a); a;})
#define DD ({double a; scanf("%lf", &a); a;})
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Longo;
typedef unsigned long long ull;
typedef vector<int> vi ;
typedef set<int> si;
typedef vector<Longo> vl;
typedef pair<int,int> pii;
typedef pair<Longo,Longo> pll;
typedef pair<double,double> pdd;

#define forab(i, a, b) for (__typeof (b) i = (a) ; i <= b ; ++i)
#define rep(i, n)      forab (i, 0, (n) - 1)
#define For(i, n)      forab (i, 1, n)
#define rofba(i, a, b) for (__typeof (b)i = (b) ; i >= a ; --i)
#define per(i, n)      rofba (i, 0, (n) - 1)
#define rof(i, n)      rofba (i, 1, n)
#define forstl(i, s)   for (__typeof ((s).end ()) i = (s).begin (); i != (s).end (); ++i)

const int MX = 507;
const int INF = 1e8 + 7;

Longo dp[MX][27];
bool Visitado[MX][27];
int tamano, valor;

Longo resolver(int i, int j) {
    if (i == tamano && j == 1) return valor;
    Longo &retorno = dp[i][j];
    if (Visitado[i][j]) return retorno;
    Visitado[i][j] = true;
    retorno = 0;
    if (i < j) {
        for (int k = i; k < j; k++)
            retorno = max(retorno, resolver(i, k) + resolver(k + 1, j));
    } else {
        Longo r1 = 0, r2 = 0;
        if (i < tamano) {
            for (int k = i + 1; k <= tamano; k++)
                r1 = max(r1, resolver(k, 1) + resolver(k, j));
        }
        if (j > 0) {
            for (int k = 1; k < j; k++)
                r2 = max(r2, resolver(i, k) + resolver(tamano, k));
        }
        retorno = r1 + r2;
    }
    return retorno;
}

```

```cpp
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Longo;
typedef unsigned long long ull;
typedef vector<int> vi ;
typedef set<int> si;
typedef vector<Longo> vl;
typedef pair<int,int> pii;
typedef pair<Longo,Longo> pll;
typedef pair<double,double> pdd;

#define forab(i, a, b) for (__typeof (b) i = (a) ; i <= b ; ++i)
#define rep(i, n)      forab (i, 0, (n) - 1)
#define For(i, n)      forab (i, 1, n)
#define rofba(i, a, b) for (__typeof (b)i = (b) ; i >= a ; --i)
#define per(i, n)      rofba (i, 0, (n) - 1)
#define rof(i, n)      rofba (i, 1, n)
#define forstl(i, s)   for (__typeof ((s).end ()) i = (s).begin (); i != (s).end (); ++i)

const int MX = 507;
const int INF = 1e8 + 7;

Longo dp[MX][27];
bool Visitado[MX][27];
int tamano, valor;

Longo resolver(int i, int j) {
    if (i == tamano && j == 1) return valor;
    Longo &retorno = dp[i][j];
    if (Visitado[i][j]) return retorno;
    Visitado[i][j] = true;
    retorno = 0;
    if (i < j) {
        for (int k = i; k < j; k++)
            retorno = max(retorno, resolver(i, k) + resolver(k + 1, j));
    } else {
        Longo r1 = 0, r2 = 0;
        if (i < tamano) {
            for (int k = i + 1; k <= tamano; k++)
                r1 = max(r1, resolver(k, 1) + resolver(k, j));
        }
        if (j > 0) {
            for (int k = 1; k < j; k++)
                r2 = max(r2, resolver(i, k) + resolver(tamano, k));
        }
        retorno = r1 + r2;
    }
    return retorno;
}

int main() {
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
    #endif
    while (scanf("%d %d", &tamano, &valor) != EOF) {
        ms(Visitado, false);
        printf("%lld\n", resolver(1, tamano));
    }
}
```

Leave a comment

#55941158 | Raul_Almanza's solution for [UVA-10313]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	120ms	2849	C++11 5.3.0	2024-11-12 02:14:40	☑	Link	29956152

Copy C++

```cpp
#include <bits/stdc++.h>
#define agregar push_back
#define todos(x) x.begin(), x.end()
#define inicializar(a, v) memset(a, v, sizeof a)
#define leerInt ({int a; scanf("%d", &a); a;})
#define leerLong ({Long a; scanf("%lld", &a); a;})
#define leerDouble ({double a; scanf("%lf", &a); a;})
#define EPS 1e-10
#define pi 3.14159265358979323846264338327795
using namespace std;

typedef long long Long;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef set<int> si;
typedef vector<Long> vl;
typedef pair<int, int> pii;
typedef pair<Long, Long> pll;
typedef pair<double, double> pdd;

#define forab(i, a, b)  for (__typeof(b) i = (a); i <= b; ++i)
#define rep(i, n)       forab(i, 0, (n) - 1)
#define For(i, n)       forab(i, 1, n)
#define rofba(i, a, b)  for (__typeof(b)i = (b); i >= a; --i)
#define per(i, n)       rofba(i, 0, (n) - 1)
#define rof(i, n)       rofba(i, 1, n)
#define forstl(i, s)    for (__typeof((s).end()) i = (s).begin(); i != (s).end(); ++i)

const int MAX = 307;
const int INF = 1e8 + 7;
int Suma, Limite1, Limite2;
Long dp1[MAX];
Long dp2[MAX][MAX];
int Monedas[MAX];
int Indice;

Long aEntero(string s) {
    Long r = 0;
    istringstream sin(s); sin >> r;
    return r;
}

vector<string> dividir(string s) {
    vector<string> resultado;
    istringstream ss(s);
    string token;
    while (ss >> token) resultado.agregar(token);
    return resultado;
}

char Cadena[MAX];

void inicializarValores() {
    vector<string> resultado = dividir(Cadena);
    Indice = resultado.size();
    if (Indice == 1) {
        Suma = aEntero(resultado[0]);
    }
    else if (Indice == 2) {
```

```cpp
    istringstream ss(s);
    string token;
    while (ss >> token) resultado.agregar(token);
    return resultado;
}

char Cadena[MAX];

void inicializarValores() {
    vector<string> resultado = dividir(Cadena);
    Indice = resultado.size();
    if (Indice == 1) {
        Suma = aEntero(resultado[0]);
    }
    else if (Indice == 2) {
        Suma = aEntero(resultado[0]);
        Limite1 = aEntero(resultado[1]);
    } else {
        Suma = aEntero(resultado[0]);
        Limite1 = aEntero(resultado[1]);
        Limite2 = aEntero(resultado[2]);
    }
}

void preparar() {
    For(i, MAX - 7) Monedas[i] = i;
    inicializar(dp1, 0);
    dp1[0] = 1;
    for (int i = 1; i <= MAX - 7; i++)
        for (int j = 0; j + Monedas[i] < MAX; j++)
            if (dp1[j]) dp1[j + Monedas[i]] += dp1[j];

    inicializar(dp2, 0);
    dp2[0][0] = 1;
    for (int s = 0; s <= MAX - 7; s++) {
        for (int j = 1; j <= MAX - 7; j++) {
            dp2[s][j] += dp2[s][j - 1];
            if (s - Monedas[j] >= 0)
                dp2[s][j] += dp2[s - Monedas[j]][j];
        }
    }
}

int main() {
    #ifdef LOCAL
        freopen("in.txt", "r", stdin);
    #endif
    preparar();
    while (gets(Cadena)) {
        inicializarValores();
        if (Indice == 1) printf("%lld\n", dp1[Suma]);
        else if (Indice == 2) {
            Limite1 = min(Limite1, 300);
            printf("%lld\n", dp2[Suma][Limite1]);
        } else {
            Limite1 = min(Limite1, 300);
            Limite2 = min(Limite2, 300);
            printf("%lld\n", dp2[Suma][Limite2] - dp2[Suma][Limite1 - 1]);
        }
    }
}
```

Leave a comment