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: 20230482 - Mestas Zegarra Christian 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 informe explora la aplicación de programación dinámica con memoización para resolver tres problemas de la plataforma UVa Virtual Judge, empleando el método SRTBOT. El proceso de resolución consistió en descomponer cada problema en subproblemas identificables, con el método SRTBOT como guía para estructurar el proceso de resolución. Se implementaron las soluciones en C++, siendo enviadas a la plataforma para su evaluación y validación. Los resultados muestran cómo SRTBOT ayuda a mejorar la estructuración de los algoritmos implementados. Además, el uso de memoización permitió evitar cálculos redundantes en problemas con subestructuras superpuestas, reduciendo el tiempo de ejecución. Las conclusiones destacan el aprendizaje obtenido en cuanto al planeamiento y optimización de algoritmos y confirman que el uso de técnicas como SRTBOT, facilita la implementación y comprensión de soluciones basadas en programación dinámica.

1. Introducción

La resolución de problemas de manera eficiente es un reto en el desarrollo de algoritmos. Entre las diversas técnicas disponibles para mejorar la eficiencia de las soluciones, la programación dinámica con memoización ha probado ser útil para problemas que poseen subestructuras repetitivas, reduciendo el tiempo de cálculo al guardar resultados de subproblemas ya resueltos. En este contexto, el uso del método SRTBOT propuesto por Demaine (2021), facilita la estructuración de soluciones basadas en programación dinámica, al dar una guía para resolver este tipo de problemas de manera organizada.

El presente informe describe el proceso de resolución de tres ejercicios extraídos de la plataforma UVa Virtual Judge, utilizando la técnica de programación dinámica con memoización y aplicando el método SRTBOT. Los objetivos de este trabajo incluyen reforzar el conocimiento sobre la programación dinámica y aplicar el método para la resolución de problemas específicos.

A lo largo del informe, se desarrollan las siguientes secciones: el Marco Teórico Conceptual (Sección 2), donde se introducen los conceptos básicos de la programación dinámica, la memoización y el método SRTBOT. El Diseño Experimental (Sección 3), que detalla el proceso de selección de los problemas y el enfoque utilizado para resolverlos, junto con los objetivos y las actividades desarrolladas. Los Resultados (Sección 4), en los que se presenta la estructura SRTBOT aplicada a cada problema junto con su implementación en C++. Por último, la sección de Conclusiones (Sección 5) resume los hallazgos y aprendizajes obtenidos a través del trabajo realizado.

2. Marco Teórico Conceptual

En esta sección, se realizará la introducción a los conceptos teóricos necesarios para entender el trabajo realizado. Se desarrollarán los conceptos de programación dinámica, memoización y el método SRTBOT.

La programación dinámica fue propuesta por Bellman (1952). Esta es una técnica que permite resolver problemas de optimización dividiendo el problema en subproblemas más simples, resolviendo cada uno de estos subproblemas una sola vez y almacenando sus soluciones para evitar recalculos. Esta técnica permite reducir el tiempo de resolución de problemas donde la solución depende de cálculos repetitivos.

El término memoización fue utilizado por primera vez por Michie (1968), es el nombre de la técnica que se utiliza como parte de la programación dinámica para guardar los resultados de los subproblemas

ya calculados. La memoización es una manera de estandarizar el proceso de almacenamiento de soluciones de la programación dinámica, permitiendo llegar a una solución de forma más consistente.

El nemotécnico SRTBOT, propuesto por Demaine (2021), ayuda a diseñar algoritmos recursivos para resolver problemas de una forma estructurada mediante los seis conceptos fundamentales: Subproblemas, Relaciones Recursivas, Topología, Bases, Original, y Tiempo. A continuación, se detalla cada uno de estos conceptos.

Subproblemas: El primer paso consiste en dividir el problema en subproblemas más pequeños e identificar cual es la forma de estos subproblemas. Esto implica identificar que partes del problema pueden resolverse de forma independiente y que juntas forman la solución del problema original.

Relaciones Recursivas: El segundo paso consiste en establecer las relaciones entre los subproblemas, generando una función recursiva que permita resolver cada subproblema a partir de los otros subproblemas. Esta relación recursiva es la base del algoritmo, pues define cómo se construyen las soluciones de los subproblemas para llegar a la solución final.

Topología: El tercer paso consiste en dibujar la topología, esto ayuda a visualizar cómo se interconectan los subproblemas y a entender mejor el orden de resolución. De esta manera, se puede identificar si el problema tiene una estructura de grafo acíclico dirigido, y, por lo tanto, hace posible la implementación del algoritmo de programación dinámica.

Bases: El cuarto paso consisten en formalizar los casos base. Un caso base se define como el subproblema mínimo en el que el problema se puede resolver sin usar otras soluciones. Es importante identificar los casos base para no caer en un escenario en el que el algoritmo nunca se detenga.

Original: El quinto paso consiste en resolver el problema original con la información de los cuatro puntos anteriores. Se debe diseñar el algoritmo recursivo que incorpora los subproblemas, las relaciones recursivas, la topología y los casos base. Posteriormente, se debe agregar **memoización** para guardar los resultados de los subproblemas ya calculados y evitar cálculos repetidos, para mejorar la eficiencia de la solución.

Tiempo: El último paso consiste en analizar el tiempo de ejecución de la solución implementada. Esto incluye evaluar el número de subproblemas que se generan y el tiempo requerido para resolver cada uno de ellos. De esta manera, se puede determinar si la solución es óptima y adecuada para el problema.

3. Diseño Experimental

En esta sección se describirá el proceso seguido para la selección y resolución de los problemas propuestos.

Para seleccionar los problemas, se utilizó la lista de ejercicios proporcionada. La elección de los problemas específicos se realizó de manera aleatoria mediante el uso de Random Number Generator.

Una vez seleccionados los problemas, se aplicó el método SRTBOT para estructurar el proceso de resolución de cada ejercicio. Este método facilitó la división de cada problema en subproblemas, la definición de relaciones recursivas, la visualización de la topología del problema, la identificación de los casos base y la implementación de memoización.

Para cada problema, se implementó una solución en un lenguaje de alto nivel (`Python`) y se hizo uso de los casos de prueba de uDebug para validar el funcionamiento correcto del algoritmo, posteriormente, se realizó la traducción respectiva a `C++`.

Finalmente, se realizó el envio de la implementación a la plataforma Virtual Judge para medir el tiempo de ejecución y obtener evidencia de la correcta implementación del algoritmo.

3.1. Objetivos

Los objetivos de este trabajo fueron los siguientes:

- Reforzar los conocimientos sobre el método de programación dinámica.
- Mejorar la eficiencia de las soluciones obtenidas mediante el uso de memoización.
- Comprender y aplicar el método SRTBOT para resolver los tres problemas propuestos.

3.2. Actividades

Para cumplir los objetivos, se realizaron las siguientes actividades:

1. Se creó un usuario en la plataforma *Virtual Judge* para acceder y resolver los problemas asignados, usando como nombre de usuario `Cricro`.
2. Se seleccionaron aleatoriamente tres problemas de la *Lista de Ejercicios* utilizando la herramienta *Random Number Generator*.
3. Para cada problema, se diseñó una solución aplicando los pasos del método SRTBOT: identificación de subproblemas, definición de relaciones recursivas, visualización de la topología, determinación de casos base y formulación del problema original.
4. Se redactó el pseudocódigo para cada problema, creando la versión básica y la versión optimizada mediante memoización.
5. Se implementó el pseudocódigo en `C++`, generando una solución de acuerdo con el modelo estructurado por SRTBOT.
6. Se compiló y ejecutó el código en *Virtual Judge* para verificar su aceptación, y se documentaron los resultados.
7. Se evaluó la eficiencia de cada solución en términos de tiempo de ejecución para la sección de tiempo de SRTBOT.
8. Se capturaron imágenes que contienen el código aceptado por la plataforma y se las anexaron al final del informe.

4. Resultados

A continuación, se presentan los resultados obtenidos para cada uno de los problemas seleccionados.

4.1. Problema 825 – Walking on the Safe Side

Enunciado:

Dados:

w = Cantidad de calles este-oeste

n = Cantidad de calles norte-sur

m = Intersecciones bloqueadas

Hallar:

Cantidad de caminos mínimos no bloqueados desde la posición $(0, 0)$ hasta la posición (w, n)

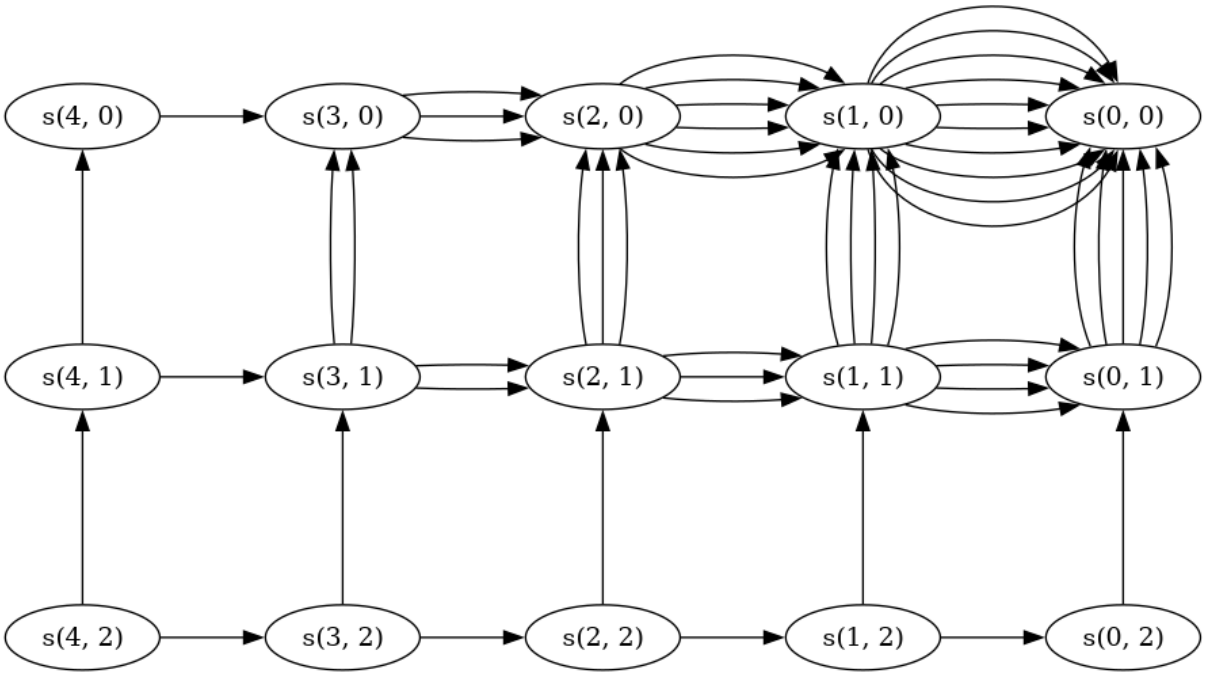
Subproblema:

Encuentre $S(i, j)$ para la calle en (i, j), $0 \leq i \leq w$ y $0 \leq j \leq n$

Relaciones Recursivas:

$S(i, j) = S(i-1, j) + S(i, j-1)$

Topología: (Para $w = 5$ y $n = 3$)



Básico:

$S(0,0) = 1$

$S(i,j) = 0, i < 0 \vee j < 0 \vee M[i][j] = 1$

Original:

Algorithm S(m, i, j) // sin memoización
Input: matriz m, índices i, j
Output: número de caminos válidos hasta (i, j)
1: **if** $i = 0$ **and** $j = 0$ **then**
2: **return** 1
3: **if** $i < 0$ **or** $j < 0$ **or** $m[i][j] = 1$ **then**
4: **return** 0
5: **return** S$(m, i, j-1)$+ S$(m, i-1, j)$

Algorithm SM(m, i, j) // con memoización
Input: matriz m, índices i, j
Output: número de caminos válidos hasta (i, j)
1: **if** $i = 0$ **and** $j = 0$ **then**
2: **return** 1
3: **if** $i < 0$ **or** $j < 0$ **or** $m[i][j] = 1$ **then**
4: **return** 0
5: **if** $M[i, j]$ is undefined **then**
6: $M[i, j]$ = SM$(m, i, j-1)$+ SM$(m, i-1, j)$
7: **return** $M[i, j]$

Tiempo: SM$(m, i, j) \in O(i * j)$

Código:

```
#include <iostream>
#include <sstream>

int s(int **memo, int **m, int i, int j)
{
    if (i == 0 && j == 0)
        return 1;
    if (i < 0 || j < 0 || m[i][j] == 1)
        return 0;
    if (memo[i][j] == -1)
        memo[i][j] = s(memo, m, i, j - 1) + s(memo, m, i - 1, j);
    return memo[i][j];
}

int split(std::string s, int a[])
{
    int idx = 0;
    std::stringstream ss(s);
```

```
19      std::string t;
20      while (std::getline(ss, t, '␣'))
21      {
22          a[idx] = std::stoi(t);
23          idx++;
24      }
25      return idx;
26  }
27
28  int main()
29  {
30      int n, ew, ns;
31      std::string l;
32      std::cin >> n;
33      for (int c = 0; c < n; c++)
34      {
35          std::cin >> ew >> ns;
36          std::getline(std::cin, l);
37          int **m = new int *[ew];
38          int **memo = new int *[ew];
39          int t[ns];
40          for (int r = 0; r < ew; r++)
41          {
42              m[r] = new int[ns];
43              memo[r] = new int[ns];
44              for (int i = 0; i < ns; i++)
45              {
46                  m[r][i] = 0;
47                  memo[r][i] = -1;
48              }
49              std::getline(std::cin, l);
50              int z = split(l, t);
51              for (int i = 1; i < z; i++)
52                  m[r][t[i] - 1] = 1;
53          }
54          std::cout << s(memo, m, ew - 1, ns - 1) << "\n";
55          if (c < n - 1)
56              std::cout << "\n";
57          for (int i = 0; i < ew; i++)
58          {
59              delete[] m[i];
60              delete[] memo[i];
61          }
62          delete[] m;
63          delete[] memo;
64      }
65  }
```

4.2. Problema 10313 – Pay the Price

Enunciado:

Dados:

min = Cantidad mínima de monedas

max = Cantidad máxima de monedas

val = Valor a obtener

Hallar:

Cantidad de maneras de obtener el valor val con cantidades de monedas i entre min y max (Hay monedas de valor $1, 2, 3, \ldots, 300$)

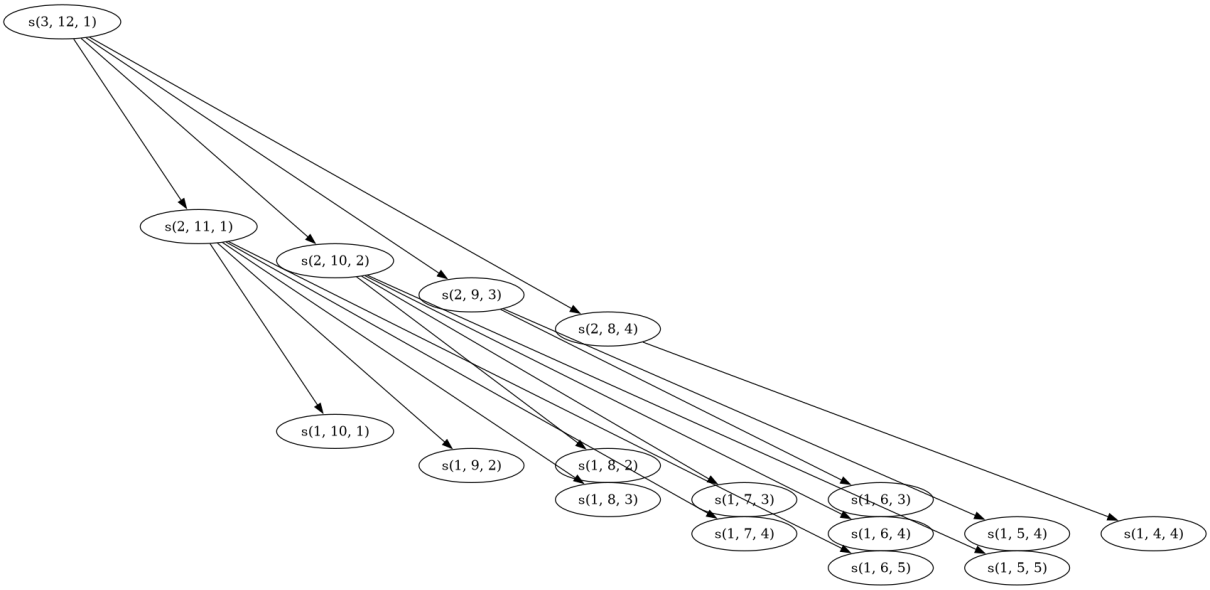
Subproblema:

Encuentre $S(c, t, v)$ para c monedas, t valor deseado y v valor mínimo, $0 \leq c \leq i$, $0 \leq t \leq val$ y $1 \leq v \leq val$

Relaciones Recursivas:

$S(c, t, v) = \sum_{i=v}^{t/c} S(c-1, t-i, i)$

Topología: (Para $c = 3$ y $t = 12$)



Básico:

$S(0, 0, v) = 1$

$S(0, t, v) = 0, t \neq 0$

$S(c, t, v) = 0, c \neq 0 \wedge t < 0$

Original:

Algorithm S(c, t, v) // sin memoización
Input: cantidad de monedas c, total t, valor mínimo v
Output: número de maneras de formar el total t con c monedas

```
1:  if c = 0 then
2:      if t = 0 then
3:          return 1
4:      else
5:          return 0
6:  if t < 0 then
7:      return 0
8:  a = 0
9:  for i = v to t/c do
10:     a = a + S(c−1, t−i, i)
11: return a
```

Algorithm SM(c, t, v) // con memoización
Input: cantidad de monedas c, total t, valor mínimo v
Output: número de maneras de formar el total t con c monedas

```
1:  if c = 0 then
2:      if t = 0 then
3:          return 1
4:      else
5:          return 0
6:  if t < 0 then
7:      return 0
8:  if M[c, t, v] is undefined then
9:      M[c, t, v] = 0
10:     for i = v to t/c do
11:         M[c, t, v] = M[c, t, v]+ SM(c−1, t−i, i)
12: return M[c, t, v]
```

Tiempo:

SM$(c, t, v) \in O(c * t * v)$

Código:

```
1 #include <iostream>
2 #include <sstream>
3
4 long long ss(long long ***memo, int c, int t, int v)
5 {
6     if (c == 0)
7     {
8         if (t == 0)
```

```
 9            return 1;
10        return 0;
11    }
12    if (t < 0)
13        return 0;
14    if (memo[c][t][v] == -1)
15    {
16        memo[c][t][v] = 0;
17        for (int i = v; i <= t / c; i++)
18            memo[c][t][v] += ss(memo, c - 1, t - i, i);
19    }
20    return memo[c][t][v];
21 }
22
23 long long s(long long ***memo, int min, int max, int v)
24 {
25    long long a = 0;
26    for (int i = min; i <= max; i++)
27    {
28        if (i > 300)
29            break;
30        if (i == 0 && v == 0)
31            a += 1;
32        else
33            a += ss(memo, i, v, 1);
34    }
35    return a;
36 }
37
38 int split(std::string s, int a[3])
39 {
40    int idx = 0;
41    std::stringstream ss(s);
42    std::string t;
43    while (std::getline(ss, t, '␣'))
44    {
45        a[idx] = std::stoi(t);
46        idx++;
47    }
48    return idx;
49 }
50
51 int main()
52 {
53    long long ***memo = new long long **[301];
54    for (int i = 0; i <= 300; i++)
55    {
56        memo[i] = new long long *[301];
57        for (int j = 0; j <= 300; j++)
58        {
59            memo[i][j] = new long long[301];
60            for (int k = 0; k <= 300; k++)
61                memo[i][j][k] = -1;
62        }
63    }
64    std::string line;
65    std::getline(std::cin, line);
66    int a[3];
67    while (!line.empty())
68    {
69        int c = split(line, a);
70        switch (c)
71        {
72        case 1:
73            std::cout << s(memo, 0, a[0], a[0]) << std::endl;
74            break;
75        case 2:
76            std::cout << s(memo, 0, a[1], a[0]) << std::endl;
77            break;
78        case 3:
79            std::cout << s(memo, a[1], a[2], a[0]) << std::endl;
80            break;
81        }
82        std::getline(std::cin, line);
83    }
84    for (int i = 0; i <= 300; i++)
85    {
86        for (int j = 0; j <= 300; j++)
```

```
87            delete[] memo[i][j];
88        delete[] memo[i];
89    }
90    delete[] memo;
91 }
```

4.3. Problema 10337 – Flight Planner

Enunciado:

Dados:

$winds$ = Velocidad del viento en cada altura y distancia

$dist$ = Distancia total a recorrer

Hallar:

Consumo mínimo de combustible para ir de la posición $(0, 0)$ a la posición $(dist, 0)$

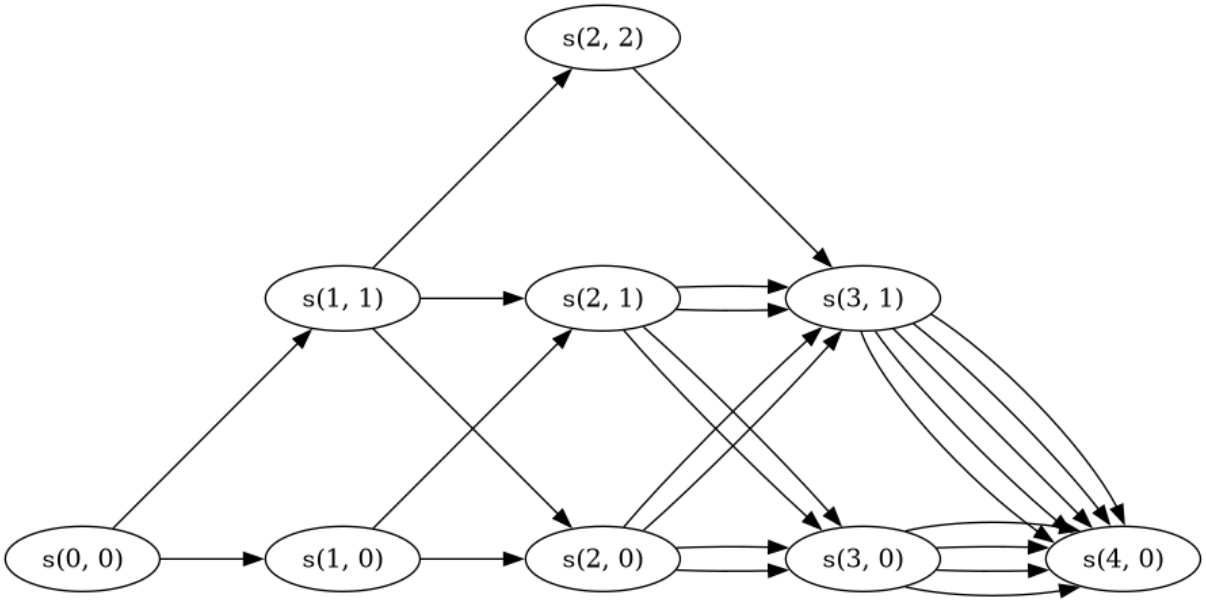
Subproblema:

Encuentre $S(h, c)$ para la altura h y la distancia recorrida c, $0 \leq h \leq 9$ y $0 \leq c \leq$ dist.

Relaciones Recursivas:

$$S(h,c) = \text{mín}\begin{pmatrix} 60 - \text{winds}[h][c] + S(h{+}1, c{+}1), \\ 30 - \text{winds}[h][c] + S(h, c{+}1), \\ 20 - \text{winds}[h][c] + S(h{-}1, c{+}1) \end{pmatrix}$$

Topología: (Para $dist = 4$)



Básico:

$S(c, 0) = 0, c = dist$

$S(c, h) = \infty, (c = dist \wedge h \neq 0) \vee [c \neq dist \wedge (h < 0 \vee h > dist - c)]$

Original:

Algorithm S(h, c) // sin memoización
Input: altura h, distancia recorrida c
Output: consumo mínimo de combustible para alcanzar el destino

if c = distancia total **then**
  if $h = 0$ **then**
    return 0
  else
    return ∞
if $h < 0$ **or** $h > dist - c$ **then**
  return ∞
$m = 60 - \text{winds}[h][c] + S(h{+}1, c{+}1)$
$m = \text{mín}(m, 30 - \text{winds}[h][c] + S(h, c{+}1))$
$m = \text{mín}(m, 20 - \text{winds}[h][c] + S(h{-}1, c{+}1))$
return m

Algorithm SM(h, c) // con memoización
Input: altura h, distancia recorrida c
Output: consumo mínimo de combustible para alcanzar el destino

if c = distancia total **then**
  if $h = 0$ **then**
    return 0
  else
    return ∞
if $h < 0$ **or** $h > dist - c$ **then**
  return ∞
if $M[h][c]$ is undefined **then**
  $M[h][c] = 60 - \text{winds}[h][c] + \text{SM}(h{+}1, c{+}1)$
  $M[h][c] = \text{mín}(m, 30 - \text{winds}[h][c] + \text{SM}(h, c{+}1))$
  $M[h][c] = \text{mín}(m, 20 - \text{winds}[h][c] + \text{SM}(h{-}1, c{+}1))$
return $M[h][c]$

Tiempo: $\text{SM}(h, c) \in O(maxh * dist)$

Código:

```
#include <iostream>
const int INF = 7000001;

int s(int **memo, int c, int d, int **w, int h)
{
    if (c == d)
    {
        if (h == 0)
            return 0;
        return INF;
    }
    if (h < 0 || h > d - c || h > 9)
    {
        return INF;
    }
    if (memo[h][c] == -1)
    {
        memo[h][c] = 60 - w[h][c] + s(memo, c + 1, d, w, h + 1);
        int t = 30 - w[h][c] + s(memo, c + 1, d, w, h);
        memo[h][c] = std::min(memo[h][c], t);
        t = 20 - w[h][c] + s(memo, c + 1, d, w, h - 1);
        memo[h][c] = std::min(memo[h][c], t);
    }
    return memo[h][c];
}

int main()
{
    int n;
    std::cin >> n;
    int **w = new int *[10];
    int **memo = new int *[10];
    for (int z = 0; z < n; z++)
    {
        int d;
        std::cin >> d;
        d /= 100;
        for (int h = 9; h >= 0; h--)
        {
            w[h] = new int[d];
            memo[h] = new int[d];
            for (int i = 0; i < d; i++)
            {
```

```
44                 std::cin >> w[h][i];
45                 memo[h][i] = -1;
46             }
47         }
48         std::cout << s(memo, 0, d, w, 0) << "\n\n";
49         for (int i = 0; i < 10; i++)
50         {
51             delete[] w[i];
52             delete[] memo[i];
53         }
54     }
55     delete[] w;
56     delete[] memo;
57 }
```

5. Conclusiones

En este informe, se ha explorado el uso de algoritmos recursivos para resolver problemas complejos mediante la técnica de programación dinámica con memoización. Un aspecto fundamental en el diseño de estas soluciones ha sido la utilización de SRTBOT, una herramienta que permite modelar el proceso de construcción de soluciones recursivas de forma más sencilla.

En particular, se resolvieron tres problemas aplicando estas técnicas:

- Problema 825 - Walking on the Safe Side: A través de una solución recursiva, se resolvió el problema de encontrar el número de rutas que se pueden tomar para cruzar una ciudad rectangular, evitando intersecciones bloqueadas.
- Problema 10313 - Pay the Price: Se utilizó programación dinámica con memoización para optimizar el cálculo de formas de alcanzar un valor dado utilizando monedas de todos los valores.
- Problema 10337 - Flight Planner: Se diseñó una solución recursiva que resolvió el problema de planificación de vuelos minimizando el consumo de combustible bajo diversas condiciones de viento.

No se encontraron problemas mayores durante el desarrollo de los ejercicios.

6. Referencias Bibliográficas

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the national Academy of Sciences*, *38*(8), 716-719.

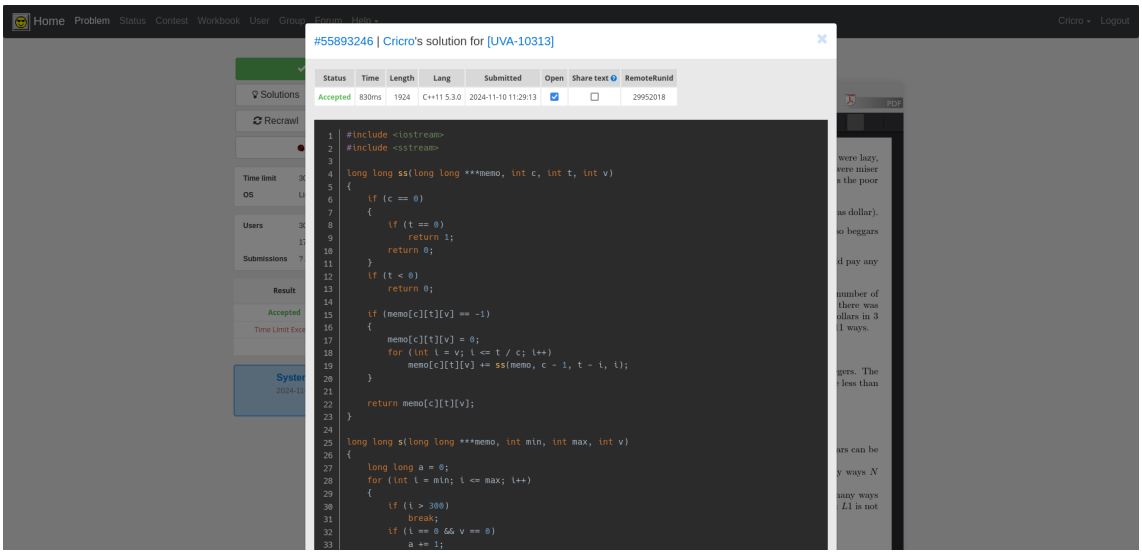
Demaine, E. (2021). *Dynamic Programming, Part 1: SRTBOT, Fib, DAGs, Bowling*. MIT OpenCourseWare. http://youtu.be/r4-cftqTcdI

Michie, D. (1968). “Memo” functions and machine learning. *Nature*, *218*(5136), 19-22.

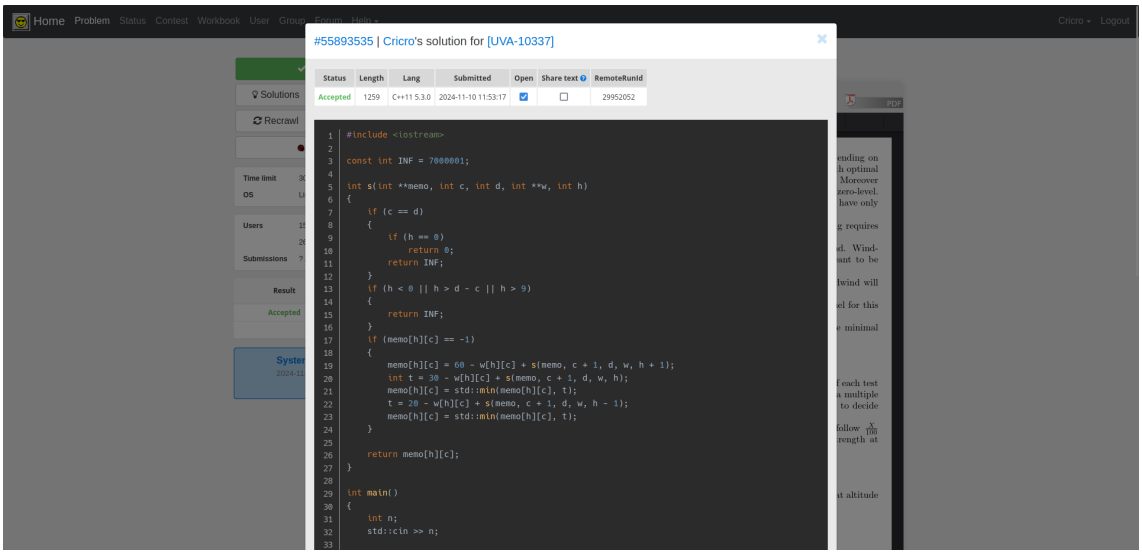
7. Anexos

En las siguientes páginas se anexa el resultado de la plataforma *Virtual Judge* al evaluar el código propuesto.

825 – Walking on the Safe Side



10313 – Pay the Price



10337 – Flight Planner