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: 20232208 Ajra Huacso Jeans Anthony					
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

El resumen presenta de manera breve, parte por parte, el contenido de todo el artículo. Debe contener entre 100 y 300 palabras. Es más fácil escribir el resumen luego de completar todo el artículo.

1. Introducción

En la actualidad, el estudio de algoritmos y técnicas de programación es fundamental para la resolución eficiente de problemas complejos en diversas áreas. Uno de los enfoques más poderosos para la resolución de problemas es la programación dinámica. Esta técnica, introducida por Richard Bellman en la década de 1950, ha sido aplicada con éxito en una amplia variedad de problemas, desde la optimización de recursos hasta el análisis de secuencias y patrones. La programación dinámica permite descomponer problemas complejos en subproblemas más simples, los cuales se resuelven de forma recursiva Y CON ciertas técnicas como el SRTBOT pueden ser almacenadas, mejorando significativamente la eficiencia.

Este trabajo tiene como objetivo profundizar en la metodología de la programación dinámica, enfocándose en su aplicación a través del nemotécnico SRTBOT. Este enfoque facilita la creación de algoritmos recursivos eficientes mediante la identificación y resolución de subproblemas. Además, se explora cómo la técnica de memoización puede ser utilizada para optimizar aún más el rendimiento de los algoritmos al almacenar resultados intermedios.

El artículo se organiza de la siguiente manera: en la Sección 2 se presenta el marco teórico que sustenta la programación dinámica y el algoritmo SRTBOT. En la Sección 3 se describe el diseño experimental y los objetivos del trabajo. La Sección 4 muestra los resultados obtenidos a partir de la resolución de tres problemas de la página VJudge.net considerando diferentes factores y especialmente la métodologia SRTBOt para entender de mejora manera el tema planteado en esta prtunidad, y en la Sección 5 se discuten las conclusiones a las que see llego a partir del desarrollo de este trabajo.

2. Marco Teórico Conceptual

La programación dinámica fue propuesta por Bellman (1952) como una técnica para resolver problemas complejos dividiéndolos en subproblemas más simples, lo cual facilita la búsqueda de soluciones óptimas. Este enfoque es particularmente útil cuando el problema presenta una estructura de subproblemas superpuestos, es decir, cuando los mismos subproblemas se resuelven varias veces. Al almacenar los resultados de estos subproblemas, la programación dinámica evita cálculos redundantes, mejorando significativamente la eficiencia en comparación con métodos recursivos directos.

El nemotécnico SRTBOT, propuesto por Demaine (2021), nos proporciona un marco sistemático para diseñar algoritmos recursivos mediante la identificación y resolución de subproblemas. A continuación, se detallan los seis conceptos fundamentales de SRTBOT:

Subproblemas: El primer paso consiste en dividir el problema original en subproblemas más pequeños y manejables. Para esto, se deben identificar los componentes del problema que pueden ser resueltos de manera independiente y luego combinar sus soluciones para resolver el problema general. La identificación de los subproblemas es crucial, ya que debe existir una estructura que permita la reutilización de los resultados de estos subproblemas.

Relaciones Recursivas: En este paso, se define cómo los subproblemas se interrelacionan entre sí. Es necesario expresar la solución al problema original en términos de las soluciones de los subproblemas. Las relaciones recursivas proporcionan la fórmula que permite calcular las soluciones a partir de los

subproblemas previamente resueltos. Esto puede implicar decisiones basadas en condiciones que dependen de los valores de los subproblemas.

Topología: En esta sección se dibuja la topología del problema en formato de recrsión, esto ayuda a visualizar las dependencias entre los subproblemas. La topología es un diagrama o representación gráfica que muestra cómo los subproblemas se conectan y se influencian mutuamente. Este paso es esencial para comprender la estructura global del algoritmo recursivo y asegurarse de que todos los subproblemas sean tratados de manera correcta.

Bases: Aquí se formalizan los casos base. Un caso base se define como un subproblema cuya solución es trivial o directamente conocida. Los casos base son fundamentales para evitar ciclos infinitos en la recursión y para proporcionar los valores iniciales que permiten construir las soluciones para los subproblemas más complejos.

Original: Ahora debemos resolver el problema original utilizando la información obtenida de los subproblemas, las relaciones recursivas, la topología y los casos base. En esta etapa, se diseña el algoritmo recursivo que utilizará estos elementos para resolver el problema completo. Para mejorar la eficiencia, es recomendable agregar la técnica de **memoización**, que consiste en almacenar los resultados de los subproblemas ya resueltos para evitar cálculos redundantes en futuras llamadas recursivas, la ejecución de ello, se era en secciones más adelnate.

Tiempo: Finalmente, debemos analizar el tiempo de ejecución de la solución obtenida. El análisis de la complejidad temporal es esencial para evaluar la eficiencia del algoritmo propuesto. En programación dinámica, la memoización juega un papel crucial en la reducción del tiempo de ejecución al evitar la recomputación de subproblemas, lo que mejora la eficiencia del algoritmo en comparación con métodos recursivos que directamente no la implemntan.

3. Diseño Experimental

En esta sección se presenta los objetivos que se siguieron a lo largo de la resolución de los problemas que se discutirán más adelante, así como también las actividades propuestas para cumplir con los objetivos planteados. De forma general, se describe el proceso seguido durante la realización de la investigación para seleccionar, analizar y resolver los problemas propuestos. También se especificarán los logros alcanzados y las dificultades que se encontraron al momento de realizar las actividades.

3.1. Objetivos

Los objetivos de este trabajo son:

- Reforzar los conocimientos del método de programación dinámica.
- Aplicar el método de programación dinámica para resolver algunos problemas propuestos.

3.2. Actividades

A continuación se detallan las acciones que se han llevado a cabo durante el proceso:

1. Se ha creado un usuario en http://vjudge.net y se ha indicado el nombre de usuario utilizado.
2. Se han seleccionado aleatoriamente tres problemas de la lista disponible en http://bit.ly/3UxdCVL. Cabe señalar que fue necesario iniciar sesión en la plataforma para poder acceder a los problemas.
3. Se ha diseñado una solución utilizando la técnica SRTBOT para cada uno de los problemas seleccionados.
4. Se ha presentado el pseudocódigo recursivo resultante, tanto sin como con memoización.
5. Se ha incluido el código en C++ derivado del modelo SRTBOT.
6. Se ha anexado el PDF del código aceptado por la plataforma http://vjudge.net al final de este artículo.

3.3. Logros Alcanzados

A continuación se describen los logros alcanzados durante el proceso:

- Se concretaron los requerimientos planteados en la sección de actividades, ingresando a la plataforma http://vjudge.net con una cuenta de usuario creada previamente, lo que permitió acceder a la lista de problemas disponible. Tras ello, se seleccionaron tres problemas de la lista proporcionada siguiendo un criterio que aseguiraba resolver problemas que sean lo más acertados posibles para aplicar la metodología explicada. Para cada uno de los problemas, se diseñó una solución utilizando la técnica SRTBOT, se generó el pseudocódigo recursivo correspondiente, y se incluyó el código en C++ derivado de dicho modelo. Finalmente, se anexó el PDF del código aceptado por la plataforma, conforme a los requerimientos establecidos.
- Se ha diseñado una solución utilizando la técnica SRTBOT para cada uno de los problemas seleccionados, lo que permitió mejorar el pensamiento sobre la programación dinámica y la forma de abordar problemas complejos mediante la descomposición en subproblemas.
- Se logró resolver los tres problemas seleccionados de la lista planteada, aplicando con éxito los principios de la técnica SRTBOT para cada caso.
- Se presentó el pseudocódigo recursivo resultante, tanto con y sin memoización, con la intención de facilitar la comprensión entre ambos enfoques y optimización del rendimiento en cada uno de los casos planteados.

3.4. Dificultades Encontradas

A continuación se detallan las dificultades encontradas durante el proceso:

- Al seleccionar aleatoriamente tres problemas de la lista disponible en http://bit.ly/3UxdCVL, se encontró que cada problema pedía soluciones distintas, necesitando con ello un enfoque diferente para cada caso, y existiendo la posibilidad de fallar en cosas como la recolección de datos de entrada, e impresión de datos de salida.
- En algunos casos, al intentar aplicar la técnica recursiva, se descubrió que el uso de la recursión no era necesarimanete el más optimo, lo que hacía el problema más complejo de lo esperado. En este tipo de casos, se optaron por problemas que puedan apoyar de mejor manera los objetivos planteados en esta investigación .
- Se encontraron dificultades al tratar de adaptar la técnica SRTBOT a ciertos problemas que no se ajustaban bien a la estructura de subproblemas típicos, lo que llevó a buscar problemas con un enfoque algo distinto, para garantizar la aplicabilidad de la técnica.
- Hubo momentos en los que la implementación de la memoización no resultó en una mejora significativa del tiempo de ejecución, lo que obligó a reevaluar el uso de esta optimización en algunos de los problemas planteados.

4. Resultados

A continuación se presentan los resultados obtenidos luego de la realización de tres problemas seleccionados de la lista otorgada de problemas de la página de VJudge.net, en este caso, los problemas que fueron seleccionados para resolver fueron: Luggage, testing the CATCHER y Tiling

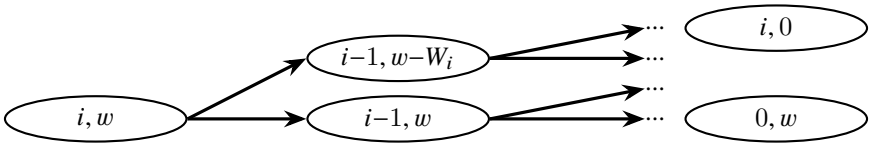
4.1. Problema 10664 – Luggage

Subproblema: Encuentre $F(i,w)$ donde $0 \le i \le n$ y $0 \le w \le \text{maxW}$.

Relaciones Recursivas:

$$F(i,w) = \begin{cases} F(i-1,w), & w < W_i \\ \text{máx}\ (\ F(i-1,w),\ F(i-1,w-W_i)\ ), & w \ge W_i \end{cases}$$

Topología:



Básico:

$$F(0,w) = \text{true}, \quad \text{si} \quad w = 0$$

$$F(n,w) = \text{false}, \quad \text{si} \quad n = 0 \text{ o } w < 0$$

Original:

Algorithm $F(i,w)$ // sin memoización
Input: objeto i, peso restante w
Output: máxima ganancia

```
1: if i = 0 or w = 0 then
2:     return 0
3: else
4:     if w < W[i] then
5:         return F(i−1, w)
6:     else
7:         return máx(F(i−1, w),
8:                    F(i−1, w−W[i]))
```

Algorithm $FM(i,w)$ // con memoización
Input: objeto i, peso restante w
Output: máxima ganancia

```
1: if i = 0 or w = 0 then
2:     return 0
3: else
4:     if M[i, w] is undefined then
5:         if w < W[i] then
6:             M[i, w] = FM(i−1, w)
7:         else
8:             M[i, w] = máx(FM(i−1, w),
9:                           FM(i−1, w−W[i]))
10:    return M[i, w]
```

Tiempo: con memoización $\text{F}(i,w) \in O(i,w)$

Código:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

bool maxLuggageMEMO(const vector<int>& weights,
int n, int maxW, unordered_map<int,
unordered_map<int, bool>>& memo) {
    if (maxW == 0) return true;
    if (n == 0 || maxW < 0) return false;

    if (memo[n].find(maxW) != memo[n].end()) {
        return memo[n][maxW];
    }

    bool result =
    maxLuggageMEMO(weights, n - 1, maxW, memo);

    if (!result) {
        result = maxLuggageMEMO(weights,
        n - 1, maxW - weights[n - 1], memo);
    }

    memo[n][maxW] = result;
    return result;
}
```

```cpp
int main() {
    int testCases;
    cin >> testCases;
    while (testCases--) {
        vector<int> weights;
        int weight, totalWeight = 0;
        while (cin >> weight) {
            weights.push_back(weight);
            totalWeight += weight;
            if (cin.peek() == '\n') break;
        }
        if (totalWeight % 2 != 0) {
            cout << "NO" << endl;
        } else {
            int maxW = totalWeight / 2;
            unordered_map<int,
            unordered_map<int, bool>> memo;
            if (maxLuggageMEMO(weights,
            weights.size(), maxW, memo))
                cout << "YES" << endl;
            else
                cout << "NO" << endl;
        }
    }
    return 0;
}
```

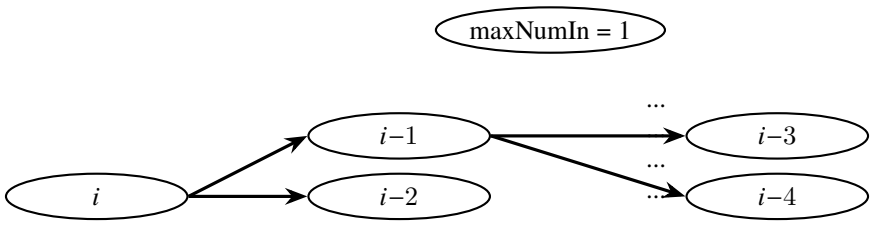
4.2. Problema 231 – Testing the CATCHER

Subproblema: Encuentra $F(i)$ donde $0 \leq i \leq n$.

Relaciones Recursivas:

$$F(i) = \begin{cases} 1 & \text{si } i = 1 \\ \max\,(1 + F(j)) & \text{si } misiles[j] \geq misiles[i] \text{ para } j < i \end{cases}$$

Topología:



Básico: $F(i) = \text{memo}[i], \quad \text{si} \quad \text{memo}[i] \neq -1$

Original:

Algorithm F(i) // sin memoización
Input: misil i
Output: número máximo de intercepciones
1: maxNumIn = 1
2: **for** $j = 0$ **to** $i - 1$ **do**
3: **if** $misiles[j] \geq misiles[i]$ **then**
4: maxNumIn = $\max(maxNumIn, \text{F}(j) + 1)$
5: **return** maxNumIn

Algorithm F(i) // con memoización
Input: misil i
Output: número máximo de intercepciones
1: **if** $memo[i]$ is undefined **then**
2: memo[i] = 1
3: **for** $j = 0$ **to** $i - 1$ **do**
4: **if** $misiles[j] \geq misiles[i]$ **then**
5: memo[i] = $\max(memo[i], \text{F}(j) + 1)$
6: **return** memo[i]

Tiempo: con memmoización $\text{F}(i) \in O(i^2)$

Código:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int maxInterc(const vector<int>& missiles,
int index, vector<int>& memo) {
    if (memo[index] != -1)
        return memo[index];

    int maxNumIn = 1;
    for (int j = 0; j < index; ++j) {
        if (missiles[j] >= missiles[index])
            maxNumIn = max(maxNumIn,
            maxInterc(missiles, j, memo) + 1);
    }

    memo[index] = maxNumIn;
    return memo[index];
}

int main() {
    vector<vector<int>> testCases;
    int height;

    while (cin >> height) {
        vector<int> missiles;

        while (height != -1) {
            missiles.push_back(height);
            cin >> height;
        }
        if (missiles.empty() && height == -1)
            break;
        if (!missiles.empty())
            testCases.push_back(missiles);
    }

    int testCase = 1;
    int totalCases = testCases.size();
    for (const auto& missiles : testCases) {
        int n = missiles.size();
        vector<int> memo(n, -1);
        int numMmaxInterc = 0;

        for (int i = 0; i < n; ++i)
            numMmaxInterc =
            max(numMmaxInterc,
            maxInterc(missiles, i, memo));

        cout << "Test #" << testCase << ":\n";
        cout << "  maximum
        possible interceptions: "
        << numMmaxInterc << "\n";

        if (testCase != totalCases)
            cout << "\n";
        testCase++;
    }
    return 0;
}
```

```cpp
int main () {
  vector<vector<int>> testCases;
  int height;

  while (cin >> height) {
    vector<int> missiles;

    while (height != -1) {
      missiles.push_back(height);
      cin >> height;
    }
    if (missiles.empty() && height == -1)
    break;
    if (!missiles.empty())
    testCases.push_back(missiles);
  }

  int testCase = 1;
  int totalCases = testCases.size();
  for (const auto& missiles : testCases) {
    int maxNumIn = 0;
    int n = missiles.size();

    vector<int> memo(n, -1);

    for (int i = 0; i < n; ++i) {
      maxNumIn = max(maxNumIn,
      maxInterc(missiles, i, memo));
    }
```

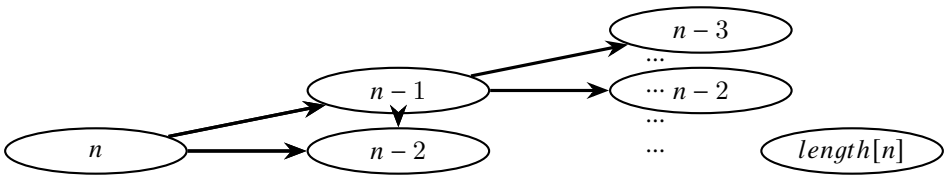
4.3. Problema – Max Tiling

Subproblema: Encuentra $F(n)$ para $0 \leq n \leq N$, donde N = valor de entrada.

Relaciones Recursivas:

$$F(n) = \begin{cases} 1 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F(n-1) + 2 \times F(n-2) & \text{si } n > 1 \end{cases}$$

Topología:



Básico: memo[0] ≠ −1, memo[1] ≠ −1

Original:

Algorithm F(n) // sin memoización
Input: n
Output: número de formas en que se puede tilizar un rectángulo 2 × n

```
1:  if n ≤ 1 then
2:      return
3:  F((n - 1))
4:  F((n - 2))
5:  for j = 1 to 300 do
6:      a[n][j] += a[n−1][j]+a[n−2][j]×2
7:      if a[n][j] ≥ 10 then
8:          a[n][j + 1] += a[n][j] ÷ 10
9:          a[n][j] %= 10
10: for k = 300 to 1 do
11:     if a[n][k] > 0 then
12:         length[n] = k
```

Algorithm F(n) // con memoización
Input: n
Output: número de formas en que se puede tilizar un rectángulo 2 × n

```
1:  if length[n] > 0 then //caso memoización
2:      return
3:  F((n - 1))
4:  F((n - 2))
5:  for j = 1 to 300 do
6:      dp[n][j] += dp[n−1][j]+dp[n−2][j]×2
7:      if dp[n][j] ≥ 10 then
8:          dp[n][j + 1] += dp[n][j] ÷ 10
9:          dp[n][j] %= 10
10: for k = 300 to 1 do
11:     if dp[n][k] > 0 then
12:         length[n] = k
```

Tiempo: con memoización O(N)

Código:

```cpp
#include <cstring>
#include <cstdio>
using namespace std;

int dp[255][305], length[255];

void MaxTilingMEMO(int n) {
    if (length[n] > 0) return;
    MaxTilingMEMO(n - 1);
    MaxTilingMEMO(n - 2);
    for (int j = 1; j <= 300; j++) {
        dp[n][j] += dp[n - 1][j] + dp[n - 2][j] * 2;
        if (dp[n][j] >= 10) {
            dp[n][j + 1] += dp[n][j] / 10;
            dp[n][j] %= 10;
        }
    }
    for (int k = 300; k >= 1; k--) {
        if (dp[n][k] > 0) {
            length[n] = k;
            break;
        }
    }
}

int main() {
    memset(dp, 0, sizeof(dp));
    memset(length, -1, sizeof(length));
    dp[0][1] = 1;
    dp[1][1] = 1;
    length[0] = 1;
    length[1] = 1;

    int n;
    while (scanf("%d", &n) != EOF) {
        if (n > 1) {
            MaxTilingMEMO(n);
            for (int i = length[n]; i >= 1; i--) {
                printf("%d", dp[n][i]);
            }
            printf("\n");
        } else
            printf("1\n");
    }
    return 0;
}
```

5. Conclusiones

La programación dinámica es una técnica poderosa para la resolución de problemas complejos, ya que permite dividir un problema grande en subproblemas más manejables. En este trabajo, se profundizó en el uso de la programación dinámica mediante el nemotécnico SRTBOT, el cual facilita el diseño de algoritmos recursivos eficientes. La metodología SRTBOT permitió identificar y resolver

subproblemas de manera estructurada, optimizando la solución de los problemas seleccionados a través de la técnica de memoización.

El uso de memoización mostró su efectividad al reducir el tiempo de ejecución en comparación con enfoques recursivos tradicionales. Sin embargo, se presentaron desafíos al aplicar la técnica a problemas que no seguían la estructura típica de subproblemas superpuestos. En algunos casos, la memoización no generó mejoras significativas en el rendimiento, lo que llevó a reevaluar su implementación en ciertos problemas.

En cuanto a los logros obtenidos, se logró resolver con éxito los tres problemas seleccionados, aplicando de manera adecuada los principios de la programación dinámica y utilizando el enfoque SRTBOT para optimizar las soluciones. A través de esta experiencia, se mejoró la comprensión de cómo abordar problemas complejos mediante la descomposición en subproblemas y cómo evaluar la eficiencia de los algoritmos mediante el análisis de su complejidad temporal.

En resumen, este trabajo demuestra que la programación dinámica, especialmente con la ayuda de la técnica SRTBOT y la memoización, es una herramienta esencial para resolver problemas complejos de manera eficiente. Sin embargo, es crucial seleccionar problemas que se ajusten bien a esta metodología y evaluar cuidadosamente las mejoras en el tiempo de ejecución al implementar optimizaciones como la memoización.

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

7. Anexos

En las siguientes páginas anexamos el resultado de la plataforma http://vjudge.net al evaluar el código propuesto.

Estas impresiones fueron hechas con Chromium Browser con la impresora destino “Guardar como PDF” y con un tamaño de página “A3” para que entre en una sola.

All
Mine
Follow

#55556583 | alumnoUNSA's solution for [UVA-10130]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	30ms	815	C++11 5.3.0	2024-10-28 10:41:36	☑	☐	29924612

```cpp
#include <bits/stdc++.h>
using namespace std;

const int mxN = 1000+1;
const int mxW = 30+1;
int M[mxN][mxW], P[mxN], W[mxN];

int SM ( int i, int w ) {
  if ( i == 0 or w == 0)
    return 0;
  else {
    if ( M[i][w] < 0) {
      if ( w < W[i] )
      M[i][w] = SM ( i-1, w );
      else {
        int p1 = SM ( i-1, w );
        int p2 = P[i] + SM ( i-1, w-W[i] );
        M[i][w] = p1 > p2 ? p1 : p2;
      }
    }
    return M[i][w];
  }
}

int main () {
  int T, N, G;
  cin >> T;
  while ( T-- ) {
    int N;
    memset ( M, -1, sizeof ( M ) );
    cin >> N;
    int *pp = P, *pw = W;
    for ( int n = N; n; --n )
      cin >> *(++pp) >> *(++pw);
    cin >> G;
    int total = 0;
    while ( G-- ) {
      int w;
      cin >> w;
      total += SM ( N, w );
    }
    cout << total << '\n';
  }
  return 0;
}
```

Leave a comment

Loading comments...

All
Mine
Follov

#55492470 | alumnoUNSA's solution for [UVA-147]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	20ms	789	C++11 5.3.0	2024-10-28 11:19:45	☑	☐	29918885

```cpp
#include <bits/stdc++.h>
using namespace std;

const int mxN = 11+1;
const int mxT = 30000+1;
long long M[mxN][mxT];
int V[mxT] = {0, 5, 10, 20, 50, 100, 200,
              500, 1000, 2000, 5000, 10000};

long long CM ( int i, int t ) {
  if ( i <= 1 )
    return 1;
  else {
    if ( M[i][t] < 0) {
      if ( t < V[i] )
        M[i][t] = CM ( i-1, t );
      else
        M[i][t] = CM ( i-1, t )
                + CM ( i, t-V[i] );
    }
    return M[i][t];
  }
}

int main () {
  memset ( M, -1, sizeof ( M ) );
  double tot;
  cout.setf(ios_base::fixed);
  cout.precision(2);
  cin >> tot;
  while ( tot > 0.00001 ) {
    int t = int(tot * 20); t *= 5; // FIX!
    cout << right << setw(6) << tot << right
        << setw(17) << CM ( 11, t ) << '\n';
    cin >> tot;
  }
  return 0;
}
```

Leave a comment

Loading comments...

All
Mine
Follov

#55530590 | alumnoUNSA's solution for [UVA-10651]

Status	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	857	C++11 5.3.0	2024-10-29 15:46:57	☑	☐	29922455

```cpp
#include <bits/stdc++.h>
using namespace std;

const int nP = 12, mxT = ( 1 << nP );
long long M[mxT];

int cntP ( int t ) {
  int cnt = 0;
  while ( t /* > 0 */) {
    if ( t & 1 ) cnt++;
    t >>= 1;
  }
  return cnt;
}

int CM ( int t ) {
  if ( M[t] < 0 ) {
    M[t] = cntP(t);
    int pre = 2048; //100000000000
    int cnt = 1024; //010000000000
    int pos =  512; //001000000000
    while ( pos /* > 0 */ ) {
      if ( t & cnt and
          ((t&pre)>>2) ^ t&pos ) {
        int c = CM(t^pre^cnt^pos);
        if ( c < M[t] ) M[t] = c;
      }
      pre>>=1; cnt>>=1; pos>>=1;
    }
  }
  return M[t];
}

int main () {
  memset ( M, -1, sizeof ( M ) );
  int N;
  cin >> N;
  while ( N-- ) {
    string T; int t = 0;
    cin >> T;
    for ( auto & l : T ) {
      t<<=1;
      if ( l == 'o' ) ++t;
    }
    cout << CM(t) << '\n';
  }
  return 0;
}
```

Leave a comment

Loading comments...