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: 20230464, SEQUEIROS CONDORI LUIS GUSTAVO					
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 explora la aplicación del método SRTBOT y la programación dinámica en la resolución de problemas de programación competitiva. A partir de una selección aleatoria de problemas, se empleó una metodología de diseño que permite descomponer cada problema en subproblemas. Luego, se aplicó memoización para optimizar el uso de recursos y mejorar la eficiencia de las soluciones. La implementación inicial en Python facilitó la validación de la lógica y la depuración de errores, mientras que la versión final en C++ optimizó el rendimiento en términos de tiempo de ejecución.

El objetivo de este trabajo fue fortalecer el conocimiento en programación dinámica y familiarizarse con el uso del método SRTBOT. Entre los principales logros se encuentran el desarrollo de soluciones eficientes y el aprendizaje de técnicas avanzadas de modelado y análisis algorítmico. Las dificultades encontradas incluyeron la correcta identificación de casos base y la optimización de la estructura recursiva en problemas con restricciones complejas.

Este artículo se organiza de la siguiente forma: se presenta la teoría y el contexto de las técnicas utilizadas, el diseño experimental para la selección de problemas, la implementación de las soluciones y, finalmente, las conclusiones donde se destacan los logros y las áreas de mejora.

1. Introducción

Una forma de entrenar la resolución de problemas y la construcción de algoritmos es mediante la resolución de problemas de programación competitiva. Este artículo se enfoca en el uso de técnicas avanzadas, como el método SRTBOT y la programación dinámica, para abordar problemas complejos de manera estructurada y eficiente. Estas técnicas permiten descomponer problemas en subproblemas más pequeños y reutilizar soluciones previas. De esta manera, se optimizan los recursos computacionales necesarios para la ejecución.

El objetivo principal de este trabajo es reforzar los conocimientos en programación dinámica y aplicar el método SRTBOT para la resolución de problemas. A través de esta experiencia, se pretende consolidar habilidades en el diseño de algoritmos recursivos optimizados. Los logros alcanzados incluyen el análisis sistemático de cada problema y la implementación de soluciones eficientes C++.

El resto del artículo está organizado de la siguiente manera. La Sección 2 describe los fundamentos teóricos de las técnicas utilizadas. La Sección 3 presenta el diseño experimental y el proceso de selección de problemas. La Sección 4 muestra los resultados obtenidos en la implementación de las soluciones. Finalmente, la Sección 5 expresa las conclusiones del trabajo, resaltando los logros alcanzados y las dificultades encontradas.

2. Marco Teórico Conceptual

Programación dinámica fue propuesta por Bellman (1952). Esta es una técnica que busca la resolución de problemas que pueden ser divididos en subproblemas superpuestos, cada uno de ellos tiene una solución independiente y considerada óptima. Por ello, estas subsoluciones pueden ser almacenadas y recordadas cuando sea necesario; de esta manera se mejora la eficiencia de la solución. Asimismo, la solución del problema general viene dada por la unión de las soluciones óptimas de los subproblemas.

El nemotécnico SRTBOT propuesto por Demaine (2021) ayuda en el diseño de algoritmos recursivos para resolver un problema. A continuación, se explican los seis conceptos de SRTBOT.

Subproblemas El primer paso consiste en dividir el problema original en subproblemas. Para esto es necesario definir etapas y estados, así como las transiciones entre ellos. Se deben considerar las precondiciones y postcondiciones del problema. De este modo, se puede modelar la signatura de una función que aplique programación dinámica.

Relaciones Recursivas En este paso se deben relacionar, unir o combinar las subsoluciones de manera que se obtenga una solución óptima dependiendo del tipo de problema. En esta etapa se modela la implicación matemática de la función.

Topología Dibujar la topología ayuda a comprender cómo se relacionan las etapas, estados y sus transiciones. Asimismo, ayuda a visualizar el flujo del problema. De esta forma, se corrobora que el problema va a tener una solución finita.

Bases Aquí se formalizan los casos base. Un caso base se define como la situación o estado en el que el problema tiene una solución ya conocida y simple. Por lo tanto, no es necesario calcularla con subsoluciones.

Original Ahora se debe resolver el problema original; es decir, con la información de los cuatro puntos anteriores se diseña el algoritmo recursivo en pseudocódigo. Agregar **memoización** ayudará a mejorar la eficiencia de la solución gracias al almacenamiento de soluciones a subproblemas superpuestos.

Tiempo Finalmente se debe analizar el tiempo de ejecución de la solución para confirmar su eficiencia. En caso no sea eficiente, es recomendable analizar nuevamente la solución y optimizarla o utilizar otro enfoque diferente de programación dinámica.

3. Diseño Experimental

A continuación, se describen los pasos que se siguieron en esta investigación para seleccionar y dar solución a los problemas escogidos.

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

El estudiante deberá realizar las siguientes acciones.

1. Se creó un usuario en http://vjudge.net dientificado como **gustadev**. El registro permitió acceder a una variedad de problemas de programación competitiva en un entorno unificado. Asimismo, facilitó el seguimiento de los resultados y la evaluación del progreso.
2. Se seleccionaron aleatoriamente tres problemas de la lista disponible en http://bit.ly/3UxdCVL. Al cargar la hoja de cálculo con la lista de problemas, se combinó su orden utilizando una función de mezcla aleatoria para asegurar una distribución imparcial. A partir de esta lista mezclada, se generaron tres números aleatorios utilizando el generador de números aleatorios de Google, lo que permitió determinar las posiciones de los problemas seleccionados.
3. Se diseñaron soluciones para cada problema utilizando la técnica SRTBOT. Esta técnica facilitó el modelado de las soluciones gracias al análisis estructurado del problema antes de la implementación. Durante este proceso, se encontraron algunas dificultades en la identificación de subproblemas y en la optimización del enfoque recursivo. Para este último, en los tres problemas se tuvo dificultades, ya que no se identificaron condiciones de quiebre al comienzo el diseño. Esto se traducía en cálculos de subsoluciones innecesarios.

4. Se elaboraron dos versiones del pseudocódigo recursivo para cada problema: una versión inicial sin memoización y una versión optimizada con memoización. Esto permitió comparar el impacto de la optimización en términos de eficiencia y comprender mejor los beneficios del almacenamiento de resultados previos.
5. Se implementaron las soluciones en C++ siguiendo el modelo SRTBOT. Esta implementación en C++ se realizó tras verificar la funcionalidad en Python. Se hicieron notorias mejoras significativas en términos de tiempo de ejecución y aprovechamiento de memoria.
6. Se generó un PDF de cada solución aceptada en la plataforma http://vjudge.net y se anexó al final del artículo como evidencia de la correcta resolución de los problemas. Esto sirvió para demostrar la efectividad de las soluciones y el cumplimiento de los objetivos del trabajo.

4. Resultados

En esta sección se muestra el proceso de resolución para cada uno de los problemas escogidos.

4.1. Problema 10074 – Take the Land

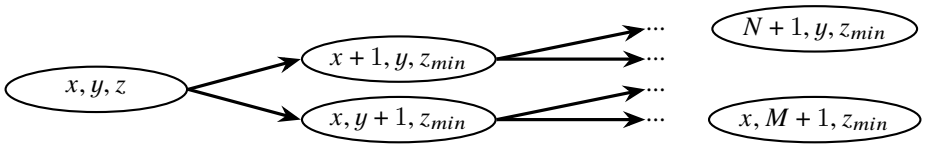
Subproblema: Encuentre $A(x, y, k, z)$ para las coordenadas de matriz $0 \leq x < N$, $0 \leq y < M$. Asimismo, para la base del mayor rectángulo k y su altura permitida z, se cumple: $0 < k \leq x$, $0 < z \leq y$ si se trabaja horizontalmente y $0 < k \leq y$, $0 < z \leq x$ si es verticalmente. Para hallar la altura z que cumple con las condiciones de un rectángulo $z = Z(x, y)$, donde la altura Z se halla dependiendo la dirección: horizontal o vertical.

Relaciones Recursivas:

$$A(x,y,k,z) = \begin{cases} A(x+1, y, 1, Z(x,y)), & Horizontal \wedge z = 0 \\ A(x, y+1, 1, Z(x,y)), & Vertical \wedge z = 0 \\ \text{máx}(z * k, A(x+1, y, k+1, min(Z(x,y)))), & Horizontal \wedge z > 0 \\ \text{máx}(z * k, A(x, y+1, k+1, min(Z(x,y)))), & Vertical \wedge z > 0 \end{cases}$$

$$Z(x,y,k,z) = \begin{cases} M_{x,y} + Z(x+1, y), & Horizontal \\ M_{x,y} + Z(x, y+1), & Vertical \end{cases}$$

Topología:



Básico: $A(x, y, k, z) = 0$, $Z(x, y) = 0$ si $x \geq N \vee y \geq M$

Original:

Algorithm A(x, y, k, z, s) // sin memoización
Input: posición x, posición y, orientación s, valor k, mínimo de ceros z
Output: área máxima

```
1: if y ≥ M or x ≥ N then
2:     return 0
3: zc ← Z(x, y, ¬s)
4: minZ ← if z ≠ 0 then mín(z, zc) else zc
5: maxA ← 0
6: vx ← if s es horizontal then x + 1 else x
7: vy ← if s es horizontal then y else y + 1
8: if minZ = 0 then
9:     maxA ← A(vx, vy, s, 1, minZ)
10: else
11:     maxA ← máx(minZ · k,
12:                A(vx, vy, s, k + 1, minZ))
13: return maxA
```

Algorithm AM(x, y, k, z, s) // con memoización
Input: posición x, posición y, orientación s, valor k, mínimo de ceros z
Output: área máxima

```
1: if y ≥ M or x ≥ N then
2:     return 0
3: Inicializar MZ, memoria para Z
4: zc ← ZM(x, y, ¬s)
5: minZ ← if z ≠ 0 then mín(z, zc) else zc
6: if M[x, y, s, minZ] is defined then
7:     return M[x, y, s, minZ]
8: maxA ← 0
9: vx ← if s es horizontal then x + 1 else x
10: vy ← if s es horizontal then y else y + 1
11: if minZ = 0 then
12:     maxA ← AM(vx, vy, s, 1, minZ)
13: else
14:     maxA ← máx(minZ · k,
15:                AM(vx, vy, s, k + 1, minZ))
16: M[x, y, s, minZ] ← maxA
17: return maxA
```

Algorithm Z(x, y, s) // sin memoización
Input: posición x, posición y, orientación s
Output: número de ceros consecutivos

```
1: if y ≥ M or x ≥ N then
2:     return 0
3: r ← 0
4: if s es horizontal then
5:     r ← 1 + Z(x + 1, y, s, m)
6: else
7:     r ← 1 + Z(x, y + 1, s, m)
8: return r
```

Algorithm ZM(x, y, s) // con memoización
Input: posición x, posición y, orientación s
Output: número de ceros consecutivos

```
1: if y ≥ M or x ≥ N then
2:     return 0
3: if MZ[x, y, s] then
4:     return MZ[x, y, s]
5: r ← 0
6: if s es horizontal then
7:     r ← 1 + ZM(x + 1, y, s, m)
8: else
9:     r ← 1 + ZM(x, y + 1, s, m)
10: MZ[x, y, s] ← r
11: return r
```

Tiempo: AM$(x, y, k, z) \in O(x * y)$, ZM$(x, y) \in O(x,\ y)$

Código:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <tuple>
#include <algorithm>

using namespace std;

struct MemoKey
{
  int x, y, minZ;
  bool s;
  bool operator==(const MemoKey &o) const
  {
    return tie(x, y, s) == tie(o.x, o.y, o.s);
  }
};

struct MemoKeyHash
{
  size_t operator()(const MemoKey &key) const
  {
    return hash<int>()(key.x)^hash<int>()(key.y)
    ^ hash<int>()(key.minZ)^hash<bool>()(key.s);
  }
};

int zeros(int x, int y, bool s,
const vector<vector<int>> &m,
unordered_map<MemoKey, int, MemoKeyHash> &mZ)
{
  if (y >= m.size() || x >= m[y].size())
    return 0;
  if (m[y][x] == 1)
    return 0;

  MemoKey key = {x, y, s};
  if (mZ.find(key) != mZ.end())
  {
    return mZ[key];
  }

  int result;
  if (s)
  {
    result = 1 + zeros(x + 1, y, s, m, mZ);
  }
  else
  {
    result = 1 + zeros(x, y + 1, s, m, mZ);
  }
  mZ[key] = result;
  return result;
}

int maxArea(int x, int y, bool s,
int k, int minZeros,
const vector<vector<int>> &m,
unordered_map<MemoKey, int, MemoKeyHash> &memo)
{
  if (y >= m.size() || x >= m[y].size())
    return 0;
  unordered_map<MemoKey, int, MemoKeyHash> mZ;
  int zerosC = zeros(x, y, !s, m, mZ);
  int minZ = (minZeros != 0)
  ? min(minZeros, zerosC) : zerosC;
  MemoKey key = {x, y, minZ, s};
  if (memo.find(key) != memo.end())
  {
    return memo[key];
  }
  int maxA = 0;
  if (minZ == 0)
  {
    maxA = maxArea((s ? x + 1 : x),
    (s ? y : y + 1), s, 1, minZ, m, memo);
  }
  else
  {
    maxA = max(minZ * k, maxArea((s ? x + 1 : x),
    (s ? y : y + 1), s, k + 1, minZ, m, memo));
  }
  memo[key] = maxA;
  return maxA;
}

int max_land(int x, int y, bool s,
const vector<vector<int>> &m)
{
  int r = 0;
  unordered_map<MemoKey, int, MemoKeyHash> memo;

  if (s && x >= 0 && y >= 0)
  {
    for (int i = x; i < m[y].size(); ++i)
    {
      r = max(r,maxArea(i, y, s, 1, 0, m, memo));
    }
    if (x == 0)
    {
      r = max(r, max_land(x, y - 1, s, m));
    }
    else
    {
      r = max(r, max_land(x - 1, y, !s, m));
    }
  }
  else if (!s && y >= 0 && x >= 0)
  {
    for (int i = y; i < m.size(); ++i)
    {
      r = max(r,maxArea(x, i, s, 1, 0, m, memo));
    }
    if (y == 0)
    {
      r = max(r, max_land(x - 1, y, s, m));
    }
    else
    {
      r = max(r, max_land(x, y - 1, !s, m));
    }
  }
  return r;
}
int main()
{
  int m, n;
  cin >> m >> n;
  while (m != 0 || n != 0)
  {
    vector<vector<int>> grid(m, vector<int>(n));

    for (int i = 0; i < m; ++i)
    {
      for (int j = 0; j < n; ++j)
      {
        cin >> grid[i][j];
      }
    }

    cout <<
    max_land(n - 1, m - 1, true, grid) << '\n';
    cin >> m >> n;
  }
  return 0;
}
```

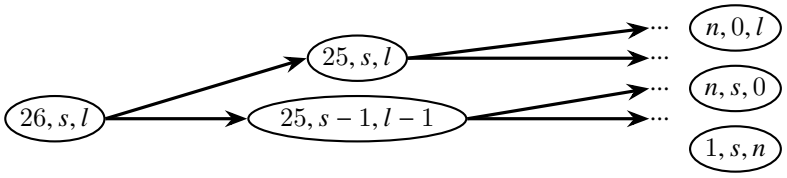
4.2. Problema 10912 – Simple Minded Hashing

Subproblema: Encuentre $H(n, s, l)$ para las letras en miñuscula $0 < n < 27$, la suma de hashing $0 < s < 10000$ y la cantidad de letras a utilizar $0 < l < 10000$

Relaciones Recursivas:

$$H(n, s, l) = \begin{cases} H(s, s, l), & n > s \\ H(n-1, s, l) + H(n-1, s-n, l-1), & n \leq s \end{cases}$$

Topología:



Básico: $H(n, 0, 0) = 1$, $H(n, s, l) = 0$ si $n > 27 \vee l < 0 \vee s < 0 \vee s < l \vee s > l * n$

Original:

Algorithm H(n, s, l) // sin memoización
Input: letra n, cantidad restante para hashing s, cantidad de letras restantes l
Output: cuántas maneras

```
1:  if s = 0 and l = 0 then
2:      return 1
3:  if n ≤ 0 or l < 0 or s < 0 then
4:      return 0
5:  if s < l or s > l * n then
6:      return 0
7:  if n > s then
8:      return H(s, s, l − 1)
9:  else
10:     return H(n−1, s−n, l−1)+H(n−1, s, l)
```

Algorithm HM(n, s, l) // con memoización
Input: letra n, cantidad restante para hashing s, cantidad de letras restantes l
Output: cuántas maneras

```
1:  if n > 27 or l < 0 or s < 0 then
2:      return 0
3:  if s = 0 and l = 0 then
4:      return 1
5:  if M[n, s, l] is defined then
6:      return M[n, s, l]
7:  if s < l or s > l * n then
8:      return 0
9:  if n > s then
10:     M[n, s, l] = HM(s, s, l)
11: else
12:     M[n, s, l] = HM(n − 1, s − n, l − 1) + HM(n − 1, s, l)
13: return M[n, s, l]
```

Tiempo: HM$(n, s, l) \in O(s * l)$

Código:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int hashNumber(int n, int s, int l,
vector<vector<vector<int>>> &memo)
{
  if (s == 0 && l == 0)
    return 1;
  if (l <= 0 || s < 0 || n <= 0)
    return 0;
  if (memo[n][s][l] != -1)
    return memo[n][s][l];

  if (s < l || s > l * n)
    return 0;
  if (n > s)
    memo[n][s][l] = hashNumber(s, s, l, memo);
  else
    memo[n][s][l] =
    hashNumber(n - 1, s - n, l - 1, memo)
    + hashNumber(n - 1, s, l, memo);
  return memo[n][s][l];
}
```

```cpp
int main()
{
  int l, s;
  int i = 1;

  vector<vector<vector<int>>> memo(27,
  vector<vector<int>>(10001,
  vector<int>(101, -1)));

  cin >> l >> s;
  while (l != 0 && s != 0)
  {
    cout << "Case␣" << i << ":␣"
    << hashNumber(min(s, 26), s, l, memo)
    << '\n';
    cin >> l >> s;
    i++;
  }

  return 0;
}
```

4.3. Problema 497 – Strategic Defense Initiative

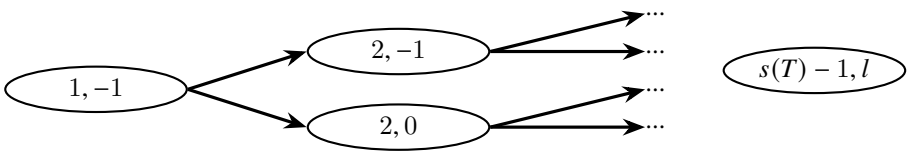
Subproblema: Encuentre $D(i, l)$ para el objetivo a destruir i y para el último objetivo destruido l

Relaciones Recursivas:

$$D(i,l) = \begin{cases} D(i+1,l), & T[i] \leq T[l] \\ max\{D(i+1,l), 1+D(i+1,i)\}, & T[i] > T[l] \end{cases}$$

donde T son los objetivos mapeados por índices y $s(T)$ es la cantidad de objetivos.

Topología:



Básico: $D(s(T), l) = 0$

Original:

Algorithm D(i, l) // sin memoización
Input: posición del objetivo actual i, posición del último objetivo destruido l
Output: máximo de objetivos destruídos

1: **if** $i \geq s(T)$ **then**
2: **return** 0
3: **if** $T[i] > T[l]$ **then**
4: **return** $max\{D(i+1,l), 1+D(i+1,i)\}$
5: **else**
6: **return** $D(i+1,l)$

Algorithm DM(i, l) // con memoización
Input: posición del objetivo actual i, posición del último objetivo destruido l
Output: máximo de objetivos destruídos

1: **if** $i \geq s(T)$ **then**
2: **return** 0
3: **if** $M[i][l]$ *is defined* **then**
4: **return** $M[i][l]$
5: **if** $T[i] > T[l]$ **then**
6: $M[i][l] = max\{DM(i+1,l), 1+DM(i+1,i)\}$
7: **else**
8: $M[i][l] = DM(i+1,l)$
9: **return** $M[i][l]$

Tiempo: DM$(i, l) \in O(s(T)^2)$

Código:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <cstring>

using namespace std;

long long maxHits(int i, int l,
const vector<long long> &a,
vector<vector<long long>> &m)
{
  if (i >= a.size())
    return 0;
  if (m[i][l] != -1)
    return m[i][l];
  long long t = 0;
  if (a[i] > a[l])
    t = 1 + maxHits(i + 1, i, a, m);
  long long nt = maxHits(i + 1, l, a, m);
  m[i][l] = max(t, nt);
  return m[i][l];
}
void rec(const vector<long long> &a,
const vector<vector<long long>> &m)
{
  if (a.size() == 1)
  {
    cout << a[0] << '\n';
    return;
  }
  long long l = 0;
  long long i{0};
  while (i < a.size() - 1)
  {
    if (a[i] > a[l] && m[i][l] == 1 + m[i+1][i])
    {
      cout << a[i] << '\n';
      l = i;
    }
    ++i;
  }
  if (a[i] > a[l])
    cout << a[i] << '\n';
}

int main()
{
  int n;
  cin >> n;
  cin.ignore().ignore();

  for (int i = 0; i < n; i++)
  {
    string line;
    vector<long long> a;
    a.push_back(-1);
    while (getline(cin, line) && !line.empty())
    {
      long long j;
      istringstream ss(line);
      ss >> j;
      a.push_back(j);
    }

    if (a.size() == 1)
      continue;

    vector<vector<long long>> m(a.size(),
    vector<long long>(a.size() + 1, -1));
    m[0][0] = -1;
    long long maxLength = maxHits(1, 0, a, m);
    cout << "Max hits: " << maxLength << '\n';

    rec(a, m);
    if (i != n - 1)
      cout << '\n';
  }

  return 0;
}
```

5. Conclusiones

En este artículo se aplicó el método SRTBOT para el diseño de soluciones recursivas. Esta técnica facilitó el análisis y la estructuración de cada problema en subproblemas más manejables. Asimismo, permitió comprender la lógica subyacente de cada problema y diseñar soluciones eficientes debido a su enfoque sistemático.

Gracias al uso de programación dinámica y memoización, las soluciones implementadas optimizaron el uso de recursos al evitar la recalculación de subproblemas ya resueltos. La implementación inicial en Python permitió una rápida validación y corrección de errores de lógica, mientras que la versión final en C++ mejoró el rendimiento.

A pesar de los logros, surgieron algunas dificultades al identificar correctamente los casos base y optimizar la estructura recursiva en problemas con condiciones de borde no expícitas. No obstante, la experiencia adquirida en el uso de SRTBOT y programación dinámica refuerza el aprendizaje de técnicas avanzadas de resolución de problemas.

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 se anexó el resultado de la plataforma http://vjudge.net al evaluar el código propuesto.

#55830946 | gustadev's solution for [UVA-10074]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	20ms	2809	C++11 5.3.0	2024-11-08 13:04:38	☑	☐	29948401

Copy C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <tuple>
#include <algorithm>

using namespace std;

struct MemoKey
{
  int x, y, minZ;
  bool s;
  bool operator==(const MemoKey &o) const
  {
    return tie(x, y, s) == tie(o.x, o.y, o.s);
  }
};

struct MemoKeyHash
{
  size_t operator()(const MemoKey &key) const
  {
    return hash<int>()(key.x) ^ hash<int>()(key.y) ^ hash<int>()(key.minZ) ^ hash<bool>()
(key.s);
  }
};

int zeros(int x, int y, bool s,
          const vector<vector<int>> &m,
          unordered_map<MemoKey, int, MemoKeyHash> &mZ)
{
  if (y >= m.size() || x >= m[y].size())
    return 0;
  if (m[y][x] == 1)
    return 0;

  MemoKey key = {x, y, s};
  if (mZ.find(key) != mZ.end())
  {
    return mZ[key];
  }

  int result;
  if (s)
  {
    result = 1 + zeros(x + 1, y, s, m, mZ);
  }
  else
  {
    result = 1 + zeros(x, y + 1, s, m, mZ);
  }
  mZ[key] = result;
  return result;
}

int maxArea(int x, int y, bool s,
            int k, int minZeros,
            const vector<vector<int>> &m,
            unordered_map<MemoKey, int, MemoKeyHash> &memo)
{
  if (y >= m.size() || x >= m[y].size())
    return 0;
  unordered_map<MemoKey, int, MemoKeyHash> mZ;
  int zerosC = zeros(x, y, !s, m, mZ);
```

```
  unordered_map<MemoKey, int, MemoKeyHash> mZ;
  int zerosC = zeros(x, y, !s, m, mZ);
  int minZ = (minZeros != 0)
                ? min(minZeros, zerosC)
                : zerosC;
  MemoKey key = {x, y, minZ, s};
  if (memo.find(key) != memo.end())
  {
    return memo[key];
  }
  int maxA = 0;
  if (minZ == 0)
  {
    maxA = maxArea((s ? x + 1 : x),
                   (s ? y : y + 1), s, 1, minZ, m, memo);
  }
  else
  {
    maxA = max(minZ * k, maxArea((s ? x + 1 : x),
                                 (s ? y : y + 1), s, k + 1, minZ, m, memo));
  }
  memo[key] = maxA;
  return maxA;
}

int max_land(int x, int y, bool s,
             const vector<vector<int>> &m)
{
  int r = 0;
  unordered_map<MemoKey, int, MemoKeyHash> memo;

  if (s && x >= 0 && y >= 0)
  {
    for (int i = x; i < m[y].size(); ++i)
    {
      r = max(r, maxArea(i, y, s, 1, 0, m, memo));
    }
    if (x == 0)
    {
      r = max(r, max_land(x, y - 1, s, m));
    }
    else
    {
      r = max(r, max_land(x - 1, y, !s, m));
    }
  }
  else if (!s && y >= 0 && x >= 0)
  {
    for (int i = y; i < m.size(); ++i)
    {
      r = max(r, maxArea(x, i, s, 1, 0, m, memo));
    }
    if (y == 0)
    {
      r = max(r, max_land(x - 1, y, s, m));
    }
    else
    {
      r = max(r, max_land(x, y - 1, !s, m));
    }
  }
  return r;
}
int main()
{
  int m, n;
  cin >> m >> n;
  while (m != 0 || n != 0)
  {
    vector<vector<int>> grid(m, vector<int>(n));

    for (int i = 0; i < m; ++i)
```

```
  }
  memo[key] = maxA;
  return maxA;
}

int max_land(int x, int y, bool s,
             const vector<vector<int>> &m)
{
  int r = 0;
  unordered_map<MemoKey, int, MemoKeyHash> memo;

  if (s && x >= 0 && y >= 0)
  {
    for (int i = x; i < m[y].size(); ++i)
    {
      r = max(r, maxArea(i, y, s, 1, 0, m, memo));
    }
    if (x == 0)
    {
      r = max(r, max_land(x, y - 1, s, m));
    }
    else
    {
      r = max(r, max_land(x - 1, y, !s, m));
    }
  }
  else if (!s && y >= 0 && x >= 0)
  {
    for (int i = y; i < m.size(); ++i)
    {
      r = max(r, maxArea(x, i, s, 1, 0, m, memo));
    }
    if (y == 0)
    {
      r = max(r, max_land(x - 1, y, s, m));
    }
    else
    {
      r = max(r, max_land(x, y - 1, !s, m));
    }
  }
  return r;
}
int main()
{
  int m, n;
  cin >> m >> n;
  while (m != 0 || n != 0)
  {
    vector<vector<int>> grid(m, vector<int>(n));

    for (int i = 0; i < m; ++i)
    {
      for (int j = 0; j < n; ++j)
      {
        cin >> grid[i][j];
      }
    }

    cout << max_land(n - 1, m - 1, true, grid) << '\n';
    cin >> m >> n;
  }
  return 0;
}
```

Leave a comment

#55833505 | gustadev's solution for [UVA-10912]

Status	Time	Length	Lang	Submitted	RemoteRunId
Accepted	120ms	1101	C++11 5.3.0	2024-11-08 16:18:00	29948555

Copy C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int hashNumber(int n, int s, int l, vector<vector<vector<int>>> &memo)
{
  if (s == 0 && l == 0)
    return 1;
  if (l <= 0 || s < 0 || n <= 0)
    return 0;
  if (memo[n][s][l] != -1)
    return memo[n][s][l];

  // Si la suma es menor que el límite, no podré cubrirla ya que como mínimo se tiene 1 de valor para las l letras restantes.
  // Por otro lado, si la suma es mayor que el máximo valor que puedo sacar entre las letras que me quedan, la suma no es alcanzable
  if (s < l || s > l * n)
    return 0;

  if (n > s)
  {
    memo[n][s][l] = hashNumber(s, s, l, memo);
  }
  else
  {
    memo[n][s][l] = hashNumber(n - 1, s - n, l - 1, memo) + hashNumber(n - 1, s, l, memo);
  }

  return memo[n][s][l];
}

int main()
{
  int l, s;
  int i = 1;

  vector<vector<vector<int>>> memo(27, vector<vector<int>>(10001, vector<int>(101, -1)));

  cin >> l >> s;
  while (l != 0 && s != 0)
  {
    cout << "Case " << i << ": " << hashNumber(min(s, 26), s, l, memo) << '\n';
    cin >> l >> s;
    i++;
  }

  return 0;
}
```

Leave a comment

#55897847 | gustadev's solution for [UVA-497]

Status	Time	Length	Lang	Submitted	Open	Share text	RemoteRunId
Accepted	10ms	1446	C++11 5.3.0	2024-11-10 21:19:10	☑	☐	29952772

Copy C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>
#include <cstring>

using namespace std;

long long maxHits(int i, int l, const vector<long long> &a, vector<vector<long long>> &m)
{
  if (i >= a.size())
    return 0;

  if (m[i][l] != -1)
    return m[i][l];

  long long t = 0;
  if (a[i] > a[l])
  {
    t = 1 + maxHits(i + 1, i, a, m);
  }
  long long nt = maxHits(i + 1, l, a, m);
  m[i][l] = max(t, nt);
  return m[i][l];
}

void rec(const vector<long long> &a, const vector<vector<long long>> &m)
{
  if (a.size() == 1)
  {
    cout << a[0] << '\n';
    return;
  }
  long long l = 0;
  long long i{0};
  while (i < a.size() - 1)
  {
    if (a[i] > a[l] && m[i][l] == 1 + m[i + 1][i])
    {
      cout << a[i] << '\n';
      l = i;
    }
    ++i;
  }
  if (a[i] > a[l])
    cout << a[i] << '\n';
}

int main()
{
  int n;
  cin >> n;
  cin.ignore().ignore();

  for (int i = 0; i < n; i++)
  {
    string line;
    vector<long long> a;
    a.push_back(-1);
    while (getline(cin, line) && !line.empty())
    {
      long long j;
      istringstream ss(line);
      ss >> j;
```

```
  {
    t = 1 + maxHits(i + 1, i, a, m);
  }
  long long nt = maxHits(i + 1, l, a, m);
  m[i][l] = max(t, nt);
  return m[i][l];
}

void rec(const vector<long long> &a, const vector<vector<long long>> &m)
{
  if (a.size() == 1)
  {
    cout << a[0] << '\n';
    return;
  }
  long long l = 0;
  long long i{0};
  while (i < a.size() - 1)
  {
    if (a[i] > a[l] && m[i][l] == 1 + m[i + 1][i])
    {
      cout << a[i] << '\n';
      l = i;
    }
    ++i;
  }
  if (a[i] > a[l])
    cout << a[i] << '\n';
}

int main()
{
  int n;
  cin >> n;
  cin.ignore().ignore();

  for (int i = 0; i < n; i++)
  {
    string line;
    vector<long long> a;
    a.push_back(-1);
    while (getline(cin, line) && !line.empty())
    {
      long long j;
      istringstream ss(line);
      ss >> j;
      a.push_back(j);
    }

    if (a.size() == 1)
      continue;

    vector<vector<long long>> m(a.size(), vector<long long>(a.size() + 1, -1));
    m[0][0] = -1;
    long long maxLength = maxHits(1, 0, a, m);
    cout << "Max hits: " << maxLength << '\n';

    rec(a, m);
    if (i != n - 1)
      cout << '\n';
  }

  return 0;
}
```