Proyecto Segundo Parcial

1) TCP unidireccional

Descripción\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

El problema del agente viajero, uno de los problemas más complejos estudiados en el campo computacional, consiste en encontrar la ruta más corta que pase entre un conjunto de ciudades por las que debe seguir un agente vendedor ante de llegar a la ciudad de la que partió originalmente, a todo esto se restringe la condición de que dicho agente solo puede pasar por una ciudad a la vez.

A más del campo computacional, este problema puede verse involucrado en otros campos de las ciencias, involucrándose en problemas que están relacionados con las cadenas de ADN,

Se ha encontrado que este

Análisis de un problema particular\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Método Voraz:

Si queremos resolver el problema por este algoritmo debemos tener ciertas consideraciones de por medio:

* Una vez que tomemos a una ciudad en consideración, no volverá a ser reconsiderada de nuevo.
* La elección de ciudades debe hacerse desde la columna izquierda más externa a la columna derecha más externa.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| W | 8 | 8 | 1 | 8 | 1 | 6 |
| X | 5 | 7 | 7 | 3 | 4 | 5 |
| Y | 4 | 6 | 4 | 2 | 1 | 5 |
| Z | 3 | 2 | 3 | 1 | 1 | 1 |

**Propiedad voraz:**

El costo de las celdas

Se analiza el costo de las celdas adyacentes a la actual que cumplen con ciertos requisitos. Únicamente podemos escoger entre las celdas superior, inferior y adyacente derecha a la celda actual.

**Inicialización:**

Empezaremos con un apila vacía R que consecuentemente se irá llenando de valores en la tabla, hasta que este contenga un camino que conecte una celda de la columna A con una celda de la columna F.

**Secuencia del algoritmo:**

1. Anexamos a nuestra pila R la celda con el menor costo de todas en la columna A, esto es porque debemos de partir desde A hacia F.
2. Anexamos a R la próxima celda que cumple con la propiedad voraz
3. Repetimos el paso 2 hasta haber llegado a un elemento de la columna F.

**Resultado final:**

Si continuamos los pasos mencionados podremos observar el siguiente resultado.

R = {3, 2, 3, 1, 1, 1}

**Tiempo de ejecución:**

Podemos observar que para tomar una decisión de entre todas las celdas posibles, la propiedad voraz nos permite escoger inmediatamente cual será la celda menos costosa.

Si tuviésemos una tabla de dimensiones el tiempo de ejecución del algoritmo en el peor de los casos:

La creación de la pila tomaría un tiempo de ejecución de O(m).

Podría ocurrir que los elementos en la tabla estén dispuesto de tal forma que cuando deseamos anexar uno a R, el siguiente elemento será el inmediato superior o inferior, hasta terminar la columna, lo que hace que tengamos que recorrer entera la tabla exceptuando la primera columna de la cual partimos con el menor valor en sí.

El tiempo de ejecución para anexar cada valor a R será .

Sumando estos dos tiempos de ejecución tendremos que:

2) Algoritmo de Strassen:

Descripción\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

El algoritmo de Strassen es un método por división y conquista utilizado para multiplicar matrices del orden , llenando filas y columnas con ceros a las matrices que no cumplen con el requisito de dimensión.

Es un método por división y conquista ya que cumple con los siguientes puntos:

1. El problema general puede reducirse en problemas más pequeños (subproblemas) que son del mismo tipo que el problema original.
2. Cada subproblema puede resolverse de forma independiente al resto de subproblemas.
3. Finalmente, podremos expresar la solución del problema principal al combinar las soluciones de cada subproblema.

Análisis de un problema particular\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Obtener el resultado de multiplicar dos matrices A y B de dimensiones 4x4 cada una mediante el algoritmo de Stranssen.

A = , B =

Las matrices A y B tienen dimensión 4x4, por lo tanto no es necesario completar filas y columnas de las mismas con ceros ya que pertenecen al orden .

Para obtener los elementos que corresponden de la nueva matriz C producto de la multiplicación entre las matrices A y B se requiere realizar 7 multiplicaciones, que se mostrarán a continuación:

= 10

= 8

= 15

= 6

= -14

= -16

= 40

Como paso final, debes estructurar la matriz producto de A por B con los resultados previamente obtenidos, de la siguiente forma:

C = AxB =

Tiempo de ejecución\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Para obtener el tiempo de ejecución del algoritmo de Strassen debemos hallar el tiempo que el método por división y conquista necesita para resolver el problema, estos son 3 tiempos:

* Tiempo de división.
* Tiempo de conquista.
* Tiempo de combinación.

**Tiempo de división:**

El orden de cada matriz a multiplicar es , por lo que el tiempo de división para multiplicar todas las matrices es:

**Tiempo de conquista:**

Al analizar el algoritmo encontramos que se deben efectuar 7 multiplicaciones de matrices de orden , en cada multiplicación se invierte , como resultado de esto tendremos 7 multiplicaciones de matrices, y el tiempo empleado será:

**Tiempo de combinación:**

No se requiere esfuerzo alguno para combinar los tiempos de división y conquista, por lo que podemos asegurar que este tiempo es cero.

La ecuación de recurrencia para el presente ejercicio se expresa como:

Aplicando el teorema maestro tenemos que:

Analicemos en que caso del teorema maestro podemos estar:

**Caso 1:  Si ∃ε>o fn∈O(nlgba-ε)**

Si n ∈ O(n), RO es reflexiva, por lo que queda demostrado que:

)

Comparación con el método tradicional\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

El método tradicional de multiplicación de matrices n3 multiplicaciones y n3 sumas, por lo que tenemos que este método tiene un tiempo de ejecución de:

Al comparar el método tradicional con el método Strassen, encontramos una serie de ventajas y desventajas que citaremos a continuación:

* El método Strassen requiere menos multiplicaciones que el método tradicional, lo que supone menos carga para el sistema de cómputo que realiza los cálculos debido a que una multiplicación es mucho más laboriosa que una suma.
* El método strassen puede emplearse para matrices que no cumpla con la restricción de dimensión que presenta el método tradicional
* La multiplicación de matrices por medio del método de Strassen es mucho más rápido que el método de multiplicación de matrices tradicional.

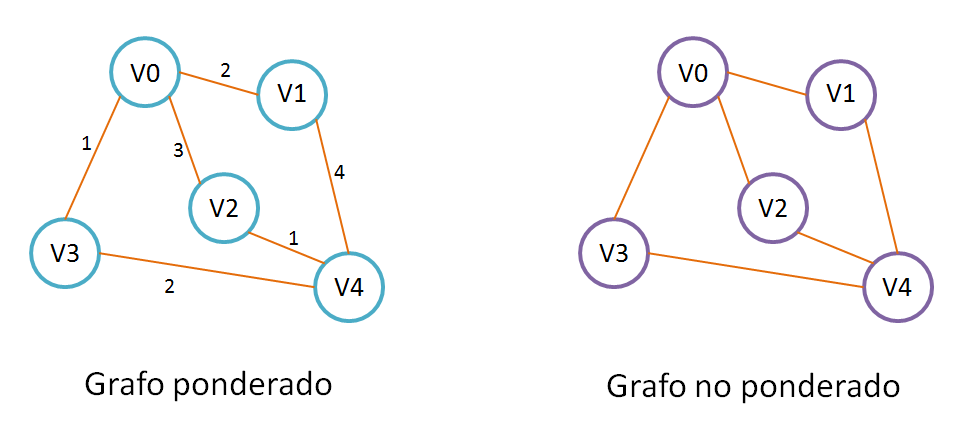
3) Problema de distancias mínimas

Descripción\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Se requiere buscar en un grafo sencillo, conectado y ponderado, las distancias mínimas de un vértice señalado a los demás vértices por dijktra.

Análisis de un problema particular\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Supongamos que debemos analizar el problema para el siguiente grafo



La implementación de dijktra para el presente grafo presenta el siguiente método:

1. Partir de un vértice inicial llamado a
2. Recorremos cada nodo adyacente al nodo inicial a
3. Marcamos cada nodo recorrido
4. Utilizamos la formula , actualizamos el vector con la distancia tentativa, si
5. Repetimos los pasos 1 al 4 hasta obtener la distancia mínima

La propiedad voraz viene a ser los pesos de los arcos.

Cada vez que se presenta una distancia prometedora (distancia menor a la antes preestablecida) intercambia la distancia que actualmente tiene almacenada por aquella, e internamente lo que hace dicha acción es escoger la distancia más prometedora, en esto se refleja cómo se utiliza la propiedad recursiva

Finalmente se puede verificar que es una respuesta correcta o eficiente ya que puede ser demostrada por fuerza bruta.

Conclusión\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

En base al previo análisis podemos afirmar que Dijktra es un algoritmo voraz ya que su funcionamiento es idéntico al funcionamiento de los algoritmos voraces.

4) Antenna Placement

Código\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

El código que permitirá la colocación de las antenas en el correcto lugar de tal forma que se utilice el mínimo espacio es:

#include<bitset>

#include<algorithm>

#include<functional>

#include<deque>

#include<cstdio>

#include<cstring>

#include<ctime>

#include<cctype>

#include<cmath>

#include<iostream>

#include<vector>

#include<queue>

#include<map>

#include<set>

// namespace

using namespace std;

// typedef

typedef long long lli;

typedef unsigned long long ull;

// const variable

#define MAX\_N 50

#define MAX\_M 1000

#define INF 2e9

#define NONE -1

#define MAX(x, y, z) {  return max(max(x,y),z); }

#define MIN(x, y, z) {  return min(min(x,y),z); }

#define EPS 1e-6

const double PI = 2.0 \* acos(0.0);

const int mx[] = { 0, 1, 0, -1 };

const int my[] = { -1, 0, 1, 0 };

// main

int n, m;

vector<int> E[MAX\_N \* MAX\_N];

bool V[MAX\_N \* MAX\_N];

int match[MAX\_N \* MAX\_N];

char T[MAX\_N][MAX\_N];

bool DFS(int source);

int convert(int a, int b);

int main(int argc, char\*\* argv) {

int t = 1;

   //scanf("%d", &t);

   while (t--){

printf("ingrese:\n");

       scanf("%d%d", &n, &m);

       for (int i = 0; i < n; i++)

           for (int j = 0; j < m; j++)

               E[convert(i,j)].clear();

       for (int i = 0; i < n; i++)

           scanf("%s", T[i]);

       for (int i = 0; i < n; i++){

           for (int j = 0; j < m; j++){

               if (T[i][j] == 'o') continue;

               for (int k = 0; k < 4; k++){

                   int nextX = j + mx[k], nextY = i + my[k];

                   if (nextX < 0 || nextX >= m || nextY < 0 || nextY >= n || T[nextY][nextX] == 'o')

                       continue;

                   E[convert(i, j)].push\_back(convert(nextY, nextX));

               }

           }

       }

       int ans = 0;

       memset(match, NONE, sizeof(match));

       for (int i = 0; i < n; i++){

           for (int j = 0; j < m; j++){

               if (T[i][j] == '\*' && match[convert(i, j)] == NONE){

                   memset(V, false, sizeof(V));

                   if (DFS(convert(i, j)))

                       ans++;

               }

           }

       }

       for (int i = 0; i < n; i++)

           for (int j = 0; j < m; j++)

               if (T[i][j] == '\*' && match[convert(i, j)] == NONE)

                   ans++;

       printf("%d\n", ans);

   }

return 0;

}

int convert(int a, int b){

   return a \* m + b;

}

bool DFS(int source){

   for (int i = 0; i < E[source].size(); i++){

       int target = E[source][i];

       if (!V[target]){

           V[target] = true;

           if (match[target] == NONE || DFS(match[target])){

               match[target] = source, match[source] = target;

               return true;

           }

       }

   }

   return false;

}

Analisis

El presente algoritmo pregunta la cantidad de veces que queremos ejecutar la evaluación del campo lleno con las antenas.

Una vez hecho esto recorrerá el campo buscando la forma de posicionar la antenas de tal forma que ocupen todo el área del campo.