ITC-ADA-C1-2023: Assignment #4

Luis Ballado
luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — February 19, 2023

1 Un grafo es bipartita si todos sus vértices pueden ser divididos en dos subconjuntos disjuntos X y Y de forma tal que cada arco conecte un vértice en X con uno en Y.

Pregunta 1

Diseñe e implemente un algoritmo basado en DFS para verificar si un grafo es bipartita.

Visto en clase que la implementación de un algoritmo DFS en su versión recursiva no hace uso de una implementación de tipo stack, ya que hace uso del call to stack que la recursión está creando.
Un algoritmo DFS con recursión puede quedar como a continuación:

Algorithm 1: Algoritmo DFS version recursiva

```
entrada: G: Un grafo representado como lista de adyacencia
entrada: nodo: El nodo de inicio
visitado ← falso;
DFS(nodo);
Function DFS(u):
    if visitado[u] = true then
        return;
    end
    print(u);
    visitado[u] ← true;
    for v ∈ G[u].vecinos() do
        DFS(v);
    end
end
```

Recordando que un grafo es bipartita sí y sólo sí, éste puede ser dividido en dos conjuntos U & V tales que no existe una arista entre ambos.

Para saber si hablamos de un grafo bipartita podemos comenzar en cualquier nodo, marcarlo como visitado.

Basandonos en ello buscaremos tomar la idea de exploración DFS para un grafo dado en texto.txt encontrar si éste es bipartito o no, proponemos el siguiente pseudo-código:

Algorithm 2: Algoritmo DFS Bipartita

entrada: G: Un grafo representado como lista de adyacencia
Function *DFS(vertice,vector_visitado,lista_adyacencia(G)):*
    marcar vertice como visitado en vector_visitado
    for *v ∈ G[vertice]* **do**
        if *v no ha sido visitado* **then**
            marcar v como visitado siempre y cuando el vertice fue visitado
            if *DFS(v,vector_visitado,lista_adyacencia) == false* **then**
                return false
            end
        end
        else if *visitado[v] == visitado[vector]* **then**
            return false
        end
    end
    return true
end
vector_visitado de la cardinalidad del núm de aristas
for *v ∈ G[u].vecinos()* **do**
    if *v no ha sido visitado* **then**
        marcar v como visitado
        if *DFS(v,vector_visitado,lista_adyacencia) == false* **then**
            bipartita = false
            break
        end
    end
end
print(bipartita)

1.1 Implementación y corridas

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ tarea4_bipartito.cpp -o bip
$ ./bip < grafo.txt
El grafo de entrada no es bipartito.
```

Pregunta 2

Analice matemáticamente la complejidad temporal de su algoritmo. Presente ejemplos de sus corridas.

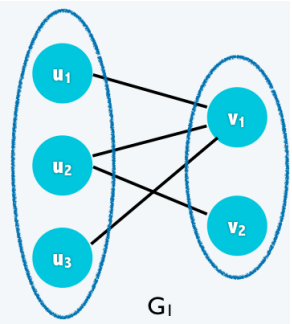
La complejidad temporal del algoritmo DFS que usa recursividad es O(|V| + |E|), donde |V| es el número de vértices en el gráfico, y |E| es el número de aristas.

En el peor de los casos, el algoritmo visita cada vértice y borde del gráfico exactamente una vez. Específicamente, cada vértice se agrega al conjunto visitado exactamente una vez y cada borde se explora exactamente una vez. El ciclo for que itera a través de los vecinos de cada vértice toma un tiempo O(grado(v)), donde grado(v) es el grado (es decir, el número de aristas que inciden en) el vértice v. Por lo tanto, la complejidad temporal total es O (|V| + |E|).

Tenga en cuenta que esta complejidad de tiempo supone que el gráfico se representa como una lista de adyacencia. Si el gráfico se representa como una matriz de adyacencia, la complejidad temporal de comprobar si existe una arista entre dos vértices requiere un tiempo O(1), por lo que la complejidad temporal total se convierte en $O(|V|^2)$.

Ejemplos:

Dado el siguiente grafo:

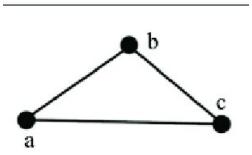


se representa como un archivo de texto como:

```
5 4
0 1
1 2
1 4
2 3
```

Command Line

```
$ cat grafo.txt
5 4
0 1
1 2
1 4
2 3
$ ./bip < grafo.txt
El grafo de entrada es bipartito.
```



se representa como un archivo de texto como:

3 3
0 1
1 2
2 0

Command Line

```
$ cat grafo.txt
3 3
0 1
1 2
2 0
$ ./bip < grafo_1.txt
El grafo de entrada no es bipartito.
```

Se incluyen varios grafos ya representados como archivos de entrada:
núm vertices - num aristas
vertice origen - vertice destino
vertice origen - vertice destino
.. etc

2 Diseñe e implemente un algoritmo basado en DFS que permita resolver eficientemente el problema 3.8 de la pág 102 del libro de Dasgupta. Analice matemáticamente la complejidad temporal de su algoritmo

3.8. Vaciando agua. Tenemos tres recipientes cuyos tamaños son de 10 lts, 7 lts y 4 lts, respectivamente. Los recipientes de 7 lts y 4 lts comienzan llenos de agua, pero el recipiente de 10 lts está inicialmente vacío. Tenemos permitido un tipo de operación: verter el contenido de un recipiente en otro, deteniéndose sólo cuando el contenedor de origen está vacío o el contenedor de destino está lleno. Queremos saber si hay una secuencia de vertidos que deja exactamente 2 lts en el recipiente de 7 o 4 lts.

Pregunta 3

Modele esto como un problema de grafos: proporcione una definición precisa del gráfico involucrado y plantee la pregunta específica sobre este grafo que necesita ser respondida.

Para modelar este problema como un problema de grafos, podemos representar los posibles estados de los tres contenedores como nodos en un grafo, donde cada nodo representa una combinación particular de niveles de agua en los tres contenedores. Luego, podemos agregar bordes entre nodos para representar las operaciones de vertido, donde un borde del nodo A al nodo B representa el vertido de agua de un recipiente a otro, lo que da como resultado un nuevo estado representado por el nodo B. La pregunta específica que debe responderse es si existe una ruta desde un nodo de inicio hasta un nodo de destino que deja exactamente 2 pintas en el contenedor de 7 o 4 pintas.

Cada nodo representa una configuración particular de los contenedores, y un borde entre dos nodos representa una operación de vertido válida que se puede realizar para pasar de una configuración a la otra. En concreto, cada nodo del gráfico corresponde a una tupla de la forma (a, b, c) que representa la cantidad actual de agua en cada recipiente, y existe una arista entre los nodos (a, b, c) y (a', b', c') si y solo si es posible verter agua de un recipiente a otro para pasar de (a, b, c) a (a', b', c').

Podemos considerar partir de la condición inicial (0, 7, 4) y llegar a la solución bajo los siguientes pasos 7 pasos:

1. 0, **7**, **4** - INICIO –> Vaciar 4 litros del tanque3 al tanque1
2. 4, **7**, 0 - Vaciar 6 litros del tanque2 al tanque1
3. **10**, 1, 0 - Vaciar 4 litros del tanque1 al tanque3
4. 6, 1, **4** - Vaciar 4 litros del tanque3 al tanque2
5. **6**, 5, 0 - Vaciar 4 litros del tanque1 al tanque3
6. 2, 5, **4** - Vaciar 2 litros del tanque3 al tanque2
7. 2, 7, 2 - Tanque1 queda con 2 litros, Tanque2 queda con 7 litros y Tanque3 queda con 2 litros

También existe una solución para llegar con 8 pasos:

1. 0, **7**, **4** - INICIO –> Vaciar 7 litros del tanque2 al tanque1
2. 7, 0, **4** - Vaciar 3 litros del tanque3 al tanque1
3. 10, 0, **1** - Vaciar 1 litro del tanque3 al tanque2
4. **10**, 1, 0 - Vaciar 4 litros del tanque1 al tanque3
5. 6, 1, **4** - Vaciar 4 litros del tanque3 al tanque2
6. **6**, 5, 0 - Vaciar 4 litros del tanque1 al tanque3
7. 2, 5, **4** - Vaciar 2 litros del tanque3 al tanque2
8. 2, **7**, 2 - Tanque1 queda con 2 litros, Tanque2 queda con 7 litros y Tanque3 queda con 2 litros

2.1 Implementación

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ tarea4_contenedores.cpp -o contenedores
$ ./contenedores
Si existe un camino donde quedan 2 litros en los contenedores
```

Pregunta 4

¿Qué algoritmo puede ser aplicado para resolver el problema?

Un algoritmo que se puede aplicar para resolver este problema es una búsqueda en profundidad (DFS) apartir del grafo de estados. Comenzando desde el estado inicial del sistema (0, 7, 4), podemos realizar un DFS del gráfico, explorando todas las secuencias posibles de operaciones de vertido hasta que encontremos una secuencia que deje exactamente 2 lts en el contenedor de 7 o 4 lts, o hasta que hayamos explorado todas las secuencias posibles sin encontrar una solución.

Podemos comenzar desde el estado inicial (nodo) y explorar recursivamente todos los caminos posibles siguiendo los bordes del gráfico, hasta alcanzar un estado objetivo. Si se encuentra un estado objetivo, podemos devolver el camino que condujo a él. También podemos usar un conjunto visitado para evitar volver a visitar los nodos y evitar bucles infinitos en los casos en que hay ciclos en el gráfico.

Pregunta 5

Encuentre la respuesta aplicando el algoritmo

Si existe un camino donde quedan 2 litros en los contenedores

3 Referencias