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 grafor es bipartita.

Partimos proponiendo el siguiente algoritmo basado en el Algoritmo DFS

Algorithm 1: Algoritmo para saber si es Bipartita

entrada: Un arreglo $A[0..n-1]$ de números reales
if *n==1* **then**
  return A[0];
else
  $temp \leftarrow Misterio(A[0..n-2])$
  if *temp <= A[n-1]* **then**
    return temp;
  else
    return A[n-1];

1.1 Implementación y corridas

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ -o ./tarea1 ./tarea1.cpp
$ ./tarea1
```

tarea4_bipartito.cpp

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

// Funcion booleana para indicar si el grafor es bipartito o no
// hacemos uso del callstack para almacenar nuestro stack para la version recursiva
bool esBipartita(int u, std::vector<int> adj[], std::vector<int>& visitado) {
  for (int v : adj[u]) {
    if (visitado.at(v) == -1) {
      visitado.at(v) = 1 - visitado.at(u); //si se invalidan dara 0
      if (!esBipartita(v, adj, visitado)) {
  return false;
      }
    } else if (visitado.at(v) == visitado.at(u)) {
      return false;
    }
  }
  return true;
}

int main() {

  int vertice; //numero de vertices
  int arista;  //numero de aristas

  //Crear el grafo con lista de adjacencia con los valores leidos
  std::cin >> vertice >> arista;

  //se crea un array de vectores con la cardinalidad de vertices(nodos)
  std::vector<int> adj[vertice];

  //ir poblando con la lectura de consola respecto
  //al num de aristas, este indica el num de conexiones en el grafo decimos que existe una conexion entre el nodo 0 y nodo 1 de la siguiente forma: 0 1
  for (int i = 0; i < arista; i++) {
    int v1, v2;  //auxiliares que representan los vertices
    std::cin >> v1 >> v2;
    adj[v1].push_back(v2); //agregar en dos vias las
    adj[v2].push_back(v1); //conexiones del grafo
  }

  //vector visitado - se inicializa con la cardinalidad de vertices(nodos) y todos en -1
  std::vector<int> visitado(vertice, -1);

  //default verdadero - hasta que la funcion diga lo contrario
  bool es_bipartita = true;

  //iterar hasta encontrar resultado
  for (int i = 0; i < vertice; i++) {
    if (visitado.at(i) == -1) {
      visitado.at(i) = 0; //marcar como visitado
      if (!esBipartita(i, adj, visitado)) {
  es_bipartita = false;
  break;
      }
    }
  }

  if (es_bipartita) {
    std::cout << "El grafo de entrada es bipartito." << std::endl;
  } else {
    std::cout << "El grafo de entrada no es bipartito." << std::endl;
  }
  return 0;
}
```

Pregunta 2

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

La complejidad shala la

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

tarea1.cpp

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

//Complejidad funcion principal O(n^2)
//por el doble for que recorre los arreglos
int main(){

  std::vector<int> a;
  a.push_back(2);
  a.push_back(5);
  a.push_back(5);
  a.push_back(5);

  std::vector<int> b;
  b.push_back(2);
  b.push_back(2);
  b.push_back(3);
  b.push_back(5);
  b.push_back(5);
  b.push_back(7);

  //vector de resultados
  std::vector<int>arr;

  int last_index = 0;

  //Usando fuerza bruta                    // O(n^2)
  for (int i = 0; i < a.size(); i++){      // O(n)
    for (int j = i; j < b.size(); j++){    // O(n)

      //std::cout << a[i] << "<-a comparacion b->"<< b[
    last_index] << "\n";

      if (a[i] == b[last_index]){
        arr.push_back(a[i]);
        break; // romper ciclo cuando sean iguales
      }

      last_index = i+1; //indice auxiliar para avanzar

    }
  }

  //---------------------
  // Imprimir resultado
  //---------------------

  std::cout << "El resultado es: \n";       // 1
  for(int i = 0; i<arr.size(); i++){        // n
    std::cout << arr[i] << "\n";
  }

}
```

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ -o ./tarea1 ./tarea1.cpp
$ ./tarea1
```

Pregunta 4

¿Qué algoritmo puede ser aplicado para resolver el problema?

(b) Un algoritmo que se puede aplicar para resolver este problema es una búsqueda en profundidad (DFS) del gráfico 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.

Un algoritmo que se puede aplicar para resolver el problema es la búsqueda primero en profundidad (DFS). DFS se puede utilizar para explorar el gráfico a partir del estado inicial, buscando un camino hacia un estado objetivo. Cuando el algoritmo DFS visita un nodo, genera todos los siguientes estados posibles aplicando todas las operaciones de vertido posibles. Si no se ha visitado antes un siguiente estado generado, el algoritmo visita recursivamente ese estado. El algoritmo retrocede si no se encuentra un estado objetivo y se han explorado todas las rutas posibles.

Pregunta 5

Encuentre la respuesta aplicando el algoritmo

Si existe un camino donde queden 2 litros en los contenedores de 7 u 4 litros

3 Referencias