ITC-ADA-C1-2023: Assignment #5

Luis Ballado
luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — March 14, 2023

1 Diseñe e implemente los algoritmos necesarios para resolver eficientemente los dos incisos del problema 4.13 de la página 127 del libro de Dasgupta, Papadimitriou y Vazirani. Analice matemáticamente la complejidad temporal de sus algoritmos.

Pregunta 1

Dado un conjunto de ciudades, junto con el patrón de carreteras entre ellas, en forma de un gráfico no dirigido G=(V,E). Cada tramo de la carretera $e \in E$ conecta dos de las ciudades, y usted conoce su longitud en millas. Suponga que quiere ir de la ciudad s a la ciudad t . Pero hay un problema, su coche solo puede contener suficiente gasolina para cubrir L millas. Hay gasolineras en cada ciudad, pero no entre ciudades. Por lo tanto, solo se puede tomar una ruta si cada una de sus aristas tiene una longitud de $l_e \neq L$

a) Dada la limitación en la capacidad del tanque de combustible de su coche, muestre cómo determinar en tiempo lineal si existe una ruta factible de s a t
b) Ahora planea comprar un coche nuevo y desea saber cuál es el tanque mínimo de combustible que se necesita para viajar de s a t. Proponga un algoritmo O((|V| + |E|) log|V|) para determinar esto.

a) Para determinar si existe un camino de s $\implies$ t con las limitantes del tanque de gasolina del carro, podemos hacer uso del algoritmo de BFS; manteniendo un rastro de las ciudades visitadas y revisando cuando el costo de cada arista es menor o igual que la capacidad del tanque.

Si la corrida del algoritmo BFS logra alcanzar la ciudad destino, podemos decir que existe una ruta con la capacidad del tanque.

Comenzando en la ciudad s y visitamos todas las ciudades adjacentes a ella que se puedan llegar con L millas. Marcamos la ciudad como visitada y la agregamos a la cola, continuaremos este proceso hasta llegar a la ciudad destino o hasta donde se acabe la gasolina.

*La complejidad del algoritmo es **O(|V|+|E|)** ya que visitamos todos los vertices y aristas almenos una vez*

b) Para determinar la capacidad mínima, que necesitamos para viajar entre las ciudades de interés, podemos hacer uso de una búsqueda binaria comenzando con una mínimo de cero y una máxima distancia entre las dos ciudades en el grafo. Para así haciendo uso de la búsqueda con las capacidades del tanque, se busca una ruta entre ciudades con la capacidad del tanque.

Para conocer si existe una ruta, dada la capacidad del tanque se hace uso del algoritmo de Dijkstra, para buscar el camino más corto.
Si existe, repetiremos la búsqueda binaria hasta encontrar la capacidad mínima que nos permita llegar de la ciudad s $implies$ t.

*Dado que la búsqueda binaria corre en **O(log(v))** para cada iteración y cada iteración necesita correr el algoritmo BFS en el grafo cuyo costo es **O(|V|+|E|)**, teniendo una complejidad total de **O(|V|+|E|)log|V|)***

1.1 Implementación y corridas

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ -std=c++11 tarea4_bipartito.cpp -o bip
$ ./bip < grafo.txt
El grafo de entrada no es bipartito.
```

Algorithm 1: Algoritmo DFS version recursiva

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

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* $\in$ *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* $\in$ *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)

2 Diseñe e implemente un algoritmo que permita resolver eficientemente el inciso (a) del problema 4.21 de la página 130 del libro de Dasgupta, Papadimitriou y Vazirani. Analice matemáticamente la complejidad temporal de su algoritmo.

Pregunta 2

Los algoritmos de ruta más corta se pueden aplicar en el comercio de divisas. Sea c_1 , c_2 , ... , c_n ser varias monedas; por ejemplo, c_1 podría ser dólares, c_2 libras y c_3 liras. Para dos monedas cualesquiera c_i y c_j , hay un tipo de cambio $r_{i,j}$; esto significa que puede comprar $r_{i,j}$ unidades de moneda c_j en cambiar por una unidad de c_i. Estos tipos de cambio satisfacen la condición de que $r_{i,j} * r_{j,i} < 1$, de modo que si comienza con una unidad de moneda c_i , la cambia a moneda c_j y luego vuelve a convertirla a moneda c_i , terminas con menos de una unidad de moneda c_i (la diferencia es el costo de la transacción).

a) Realice un algoritmo eficiente: dado un conjunto de tipos de cambio $r_{i,j}$, y dos monedas s y t, encuentre la secuencia más ventajosa de cambios de moneda para convertir la moneda s en la moneda t. Represente las monedas y tasas en un grafo cuyas longitudes son números reales.

Podemos resolver el problema haciendo uso del algoritmo de Dijkstra, encontrando el camino más corto en un grafo con pesos.
Representando las divisas como los nodos del grafo y las tasas de cambio como aristas con pesos. Especificamente para cada par de divisas c_i y c_j, podemos crear una arista de c_i a c_j con un peso de $-log(r_i, j)$, su hace uso del algoritmo negativo ya que la tasa de intercambio satisface la condición de $r_{i,j} * r_{j,i} < 1$

La idea es transformar las tasas de intercambio en retornos logarítmicos. Esta propiedad es de ayuda, ya que nos permite hacer uso de algoritmos para encontrar el camino más corto, como el de Dijkstra y así encontrar el camino con mejores ganancias.
Con los pesos de las aristas representados como logaritmos negativos convirtiendo el problema con pesos no negativos, de esta forma es posible hacer uso del algoritmo de Dijkstra, de otra forma se usaria el algoritmo de Bellman Ford para aristas negativas.

2.1 Implementación

ver código en github

Ejecutar desde una terminal

Command Line

```
$ g++ -std=c++11 tarea4_contenedores.cpp -o contenedores
$ ./contenedores
Si existe un camino donde quedan 2 litros en los contenedores
```

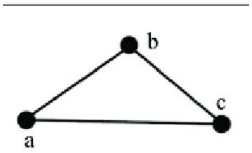
Ejemplos:

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

3 Referencias

Introduction to Algorithms - 4th Edition Cormen Leiserson - Pág 563-565

Introduction to the Design & Analysis of Algorithms - 3rd Edition - Levitin - Pág 29

Cápitulo 3 - Decomposición de grafos - Algorithms - S. Dasgupta

Intro DFS - https://www.baeldung.com/cs/depth-first-search-intro

The Vertex coloring problem and bipartite graphs
https://tylermoore.ens.utulsa.edu/courses/cse3353/slides/l08-handout.pdf

Bipartite Graphs - https://www.geeksforgeeks.org/bipartite-graph/

Learn DFS from scratch
https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm#pseudocode_of_depthfirst_search_algorithm

Bipartite graph Pág 3-7
https://speakerdeck.com/gustavoatt/bipartite-graph-matching-and-vertex-cover?slide=6

Temas de C++ - https://www.fing.edu.uy/tecnoinf/mvd/cursos/eda/material/teo/EDA-teorico14.pdf

Array de vectores en C++ - https://www.geeksforgeeks.org/array-of-vectors-in-c-stl/

Vectores C++ - https://www.programiz.com/cpp-programming/vectors

Implementacion de un grafo para programacion competitiva
https://www.geeksforgeeks.org/graph-implementation-using-stl-for-competitive-programming-set-1-dfs-c