ITC-ADA-C1-2023: Assignment #5

Luis Ballado
luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — March 16, 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.

Comenzamos 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.

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

*Para conocer si existe una ruta dada la capacidad del tanque, se hace uso nuevamente del algoritmo BFS.
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 Análisis Matemático

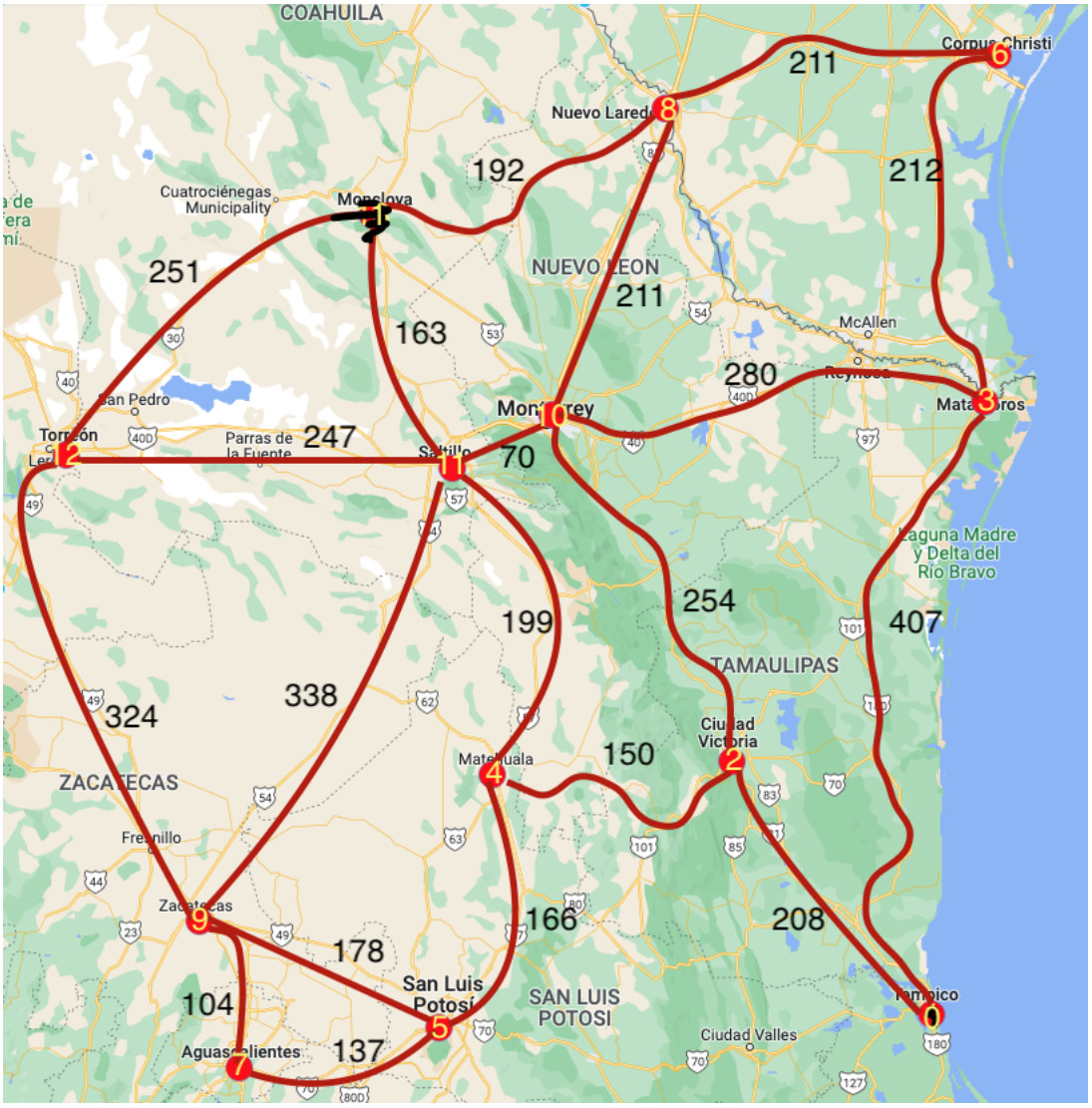
Análisis de entrada para cada iteración de la búsqueda binaria:
Primera iteración : n
Segunda iteración : $\frac{n}{2}$
Tercera iteración : $\frac{\frac{n}{2}}{2} = \frac{n}{2^2}$ Iteración n : $\frac{n}{2^k}$,
sabemos que después de n iteraciones no se necesita hacer alguna comparación $\frac{n}{2^k} = 1 \implies n = 2^k$
Aplicando log en ambos lados:
$log_2(n) = log_2(2^k)$
$log_2(n) = k * log_2(2) \implies k = log_2(n)$

Análisis de BFS:

Cada vértice vecino se inserta a la cola si no ha sido visitado. Esto se hace con las aristas adjacentes al vértice. Cada vértice visitado se marca como visitado, por lo tanto, cada vértice se visita una sola vez y se verifican todas las aristas adjacentes al vértice. Haciendo que su complejidad sea: $O(|V| + |E|)$

1.2 Implementación y corridas

ver código en github



Partiendo del siguiente grafo propuesto, dónde nuestra ciudad de origen es Tampico (soy de Tampico) y nuestra ciudad destino es Monclova. Cada arista representa aproximadamente la distancia en kilometros de ciudad a ciudad.
Nuestro grafo cuenta con 13 vértices, y 20 aristas y queremos saber si existe una ruta de Tampico (Vértice 0) a Monclova (Vértice 1).

La representación del grafo se presenta en el archivo data.txt

Ejecutar desde una terminal

El archivo data.txt contiene los datos de

- **número de ciudades, número de caminos**
- **Seguido de las aristas (ciudad origen-ciudad destino) y sus respectivos pesos**
- **ciudad origen, ciudad destino**

Command Line

```
$ g++ -std=c++11 cars_highways.cpp -o cars_highways
$ ./cars_highways < data.txt
Existe una ruta de la ciudad 0 a la ciudad 1
Considerar un tanque que cubra 208 km para viajar de la ciudad 0 a la ciudad 1
```

Si forzamos los datos desconectando a Cd. Victoria del grafo en la arista Tampico-Cd.Victoria, forzaremos que la ruta sea vía Matamoros convirtiendose ese tramo en el mas costoso que debe ser considerado para la adquisición del nuevo auto. data2.txt

Command Line

```
$ g++ -std=c++11 cars_highways.cpp -o cars_highways
$ ./cars_highways < data2.txt
Existe una ruta de la ciudad 0 a la ciudad 1
Considerar un tanque que cubra 407 km para viajar de la ciudad 0 a la ciudad 1
```

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 $-log(r_i, j)$, se 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 (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.

*La complejidad es **O(|E|+|V|log|V|)**, hay que señalar que a raíz de que hacemos uso de logaritmos, necesitamos tener cuidado con aristas que sean cercanas a cero.*

2.1 Análisis matemático

Dado el grafo representado como una lista de adjacencia y una cola de prioridad, se calcula su complejidad:

1. Tomará $O(|V|)$ inicializar la cola de prioridad dados $|V|$ vértices.
2. Con la representación del grafo como lista de adjacencia, todos los vértices del grafo se pueden recorrer usando BFS. Ya que se itera sobre todos los vértices vecinos, actualizando su distancia. Esto toma $O(|E|)$
3. El tiempo para cada iteración es $O(|V|)$, ya que los vértices son removidos de la cola en cada ciclo.
4. La cola de prioridad nos permite extraer el nodo con minima distancia. Esto toma $O(log(|V|))$
5. Su complejidad está dada por $O(|V|)+O(|E|*log|V|)+O(|V|*log|V|) \implies O((|E|+|V|)*log|V|) = O(|E| * log|V|)$, dado que $|E| >= |V| - 1$.

2.2 Implementación

ver código en github

Ejecutar desde una terminal

El archivo data.txt contiene los datos de

- **monedas, tasas de cambio**
- **moneda origen, moneda destino, tasa de cambio**
- **moneda origen, moneda destino**

Command Line

```
$ g++ -std=c++11 currency_trading.cpp -o currency_trading
$ ./currency_trading < data.txt
El intercambio de 0 a 3 es 0.4
```

3 Referencias

Dijkstra Complexity - https://www.baeldung.com/cs/dijkstra-time-complexity

Bellman-Ford to Find Arbitrage Condition in Forex Trading - https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2019-2020/Makalah2019/13518058.pdf

C++ Currency Arbitrage Bellman-Ford- https://cppsecrets.com/users/1663197115115991051059955641031099710510
C00-Currency-Arbitrage.php

Búsqueda Binaria - https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm

Búsqueda Binaria - https://medium.com/@Emmitta/b%C3%BAsqueda-binaria-c6187323cd72

Rate of return - https://en.wikipedia.org/wiki/Rate_of_return

Returns and Log Returns - https://gregorygundersen.com/blog/2022/02/06/log-returns

Analysis of breadth-first search algorithm - https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/analysis-of-breadth-first-search

COMPSCI 311: Introduction to Algorithms - https://people.cs.umass.edu/~marius/class/cs311-fa18/lec15-nup.pdf

Arbitrage as a Shortest-Path Problem - https://hackernoon.com/arbitrage-as-a-shortest-path-problem-u2l34ow

Currency Arbitrage using Bellman Ford Algorithm - https://anilpai.medium.com/currency-arbitrage-using-bellman-ford-algorithm-8938dcea56ea

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-o