Universidad de Buenos Aires

Facultad de Ingeniería

Trabajo Practico 3

[75.29/95.06] Teoría de Algoritmos
Segundo cuatrimestre 2022

- Alumno: Aguirre Juan - 102227 - jaguirreb@fi.uba.ar
- Alumno: Mosquera Sebastian - 106744 - smosquera@fi.uba.ar
- Alumno: Ramos Federico Andres - 101640 - faramos@fi.uba.ar

Índice

1. Parte1: Un proyecto distribuido en dos

1.1. Enunciado

Contamos con un proyecto que tiene un conjunto de tareas a realizar. Entre algunas tareas existe cierta interdependencia, es decir que para realizar una se requiere los resultados que otorgan una o más tareas previas. Se cuenta con dos equipos de trabajo con sus propios recursos. Cualquier tarea la puede realizar cualquiera de los equipos con un costo asociado. Llamaremos $c_{i,e}$ al costo que el equipo e realice la tarea i. Existe un costo de transferencia de resultados que se aplica si el resultado de una tarea elaborada por un equipo se la debe trasladar al otro para resolver otra tarea. Llamaremos $t_{i,j}$ al costo de la transferencia del resultado de la tarea i a la tarea j. Si dos tareas son realizadas por el mismo equipo no hay costo asociado al intercambio de resultados. Consideraremos que todos los costos son enteros positivos. Se desea minimizar el costo total del desarrollo del problema.

1.2. Solución propuesta

1.2.1. Introducción

Se propone el siguiente algoritmo:

Siendo e_{1i} el costo de que la tarea i sea desarrollada por el equipo 1, identificando al equipo 1 como $E1$ y siendo e_{2j} el costo de que la tarea j sea desarrollada por el equipo 2, identificando al equipo 2 como $E2$.

Definimos S como nodo fuente y sea T como nodo sumidero y creamos un nuevo grafo $G = (V, E)$ siendo v_i cada una de las tareas a realizar. Empezamos creando ejes $S - i$ por cada tarea con capacidad e_{1i} y se crean ejes $i - T$ por cada tarea con capacidad e_{2i}. Finalmente, se unen ejes $i - j$ por cada tarea que tiene dependencia con otra, y la primer tarea contenga un costo de transición p_{ij}.

Una vez creado el grafo, se busca el corte mínimo. Este corte mínimo divide el grafo en el segmento A y B, dándonos como resultado que los nodos de A están asignados al equipo $E2$ y los nodos de B están asignados al equipo $E1$.

1.2.2. Pseudocódigo

Algorithm 1 Network Flow Algorithm

```
L1 array de tamaño V                          ▷ Lista de costos de cada tarea para grupo 1
L2 array de tamaño V                          ▷ Lista de costos de cada tarea para grupo 2
P array de arrays de tupla de tamaño V        ▷ Lista de costos de transición de cada nodo,
conteniendo un array de tuplas relacionando nodo y costo
G Graph
S node added to Graph G                       ▷ Nodo Fuente
T node added to Graph G                       ▷ Nodo sumidero
for v in P do
    Add node v to G if not added already
    Edge S-> v with capacity L1[vindex] added to G
    Edge v-> T with capacity L2[vindex] added to G
    for e in P[v index] do                    ▷ Itero por cada costo de transición
        Edge v-> e with capacity evalue added to G
    end for
end for
residualGraph = FordFulkerson(S)
resultingGroups = separeteGroups(residualGraph) ▷ Obtengo que nodos son accesibles desde
la fuente para obtener los distintos subsets del corte minimo y asignar equipo
```

1.3. Complejidad

1.3.1. Complejidad Temporal

Primero se itera por cada uno de los nodos y luego por cada uno de los vértices para crear el nuevo grafo, resultando en $O(|V||E|)$. Luego se aplica Ford-Fulkerson, con complejidad $O(|E|C)$. Finalmente se itera cada uno de los nodos para segmentarlos, validando que este en la lista de ejes de salida de $minCutEdges$, obteniendo $O(|V|^2)$.

La complejidad temporal del problema es $O(|V||E|) + O(|E|C)$.

1.3.2. Complejidad Espacial

La operación que utiliza más memoria es la creación de un nuevo grafo, resultando en $O(V)$. Además se utilizan otras estructuras para almacenar los costos por equipo $O(V)$ y costos de transición $O(V + E)$. Finalmente, la complejidad espacial es $O(V + E)$

1.4. Implementación del Algoritmo en Python

1.4.1. Como ejecutar el script

El **main.py** se puede ejecutar con cualquier versión de Python igual o superior a **3.9.10**. Para ejecutar el programa se debe pasar por argumento:

- −−f: Nombre del archivo.

Ejemplos:

python3 main.py −−f example.txt

1.4.2. Complejidad del problema

Debido a que solo utilizamos arrays para representar el grafo como una matriz de adyacencia y para datos complementarios, solo debemos analizar una operación a la hora de calcular la complejidad temporal en Python. Esta operación es el indexado en una lista, que es una operación $O(1)$ según la documentación de Python. Con este análisis se puede concluir que la complejidad temporal de la solución escrita en Python se mantiene.

2. Parte 2: Más allá de Bellman-Ford

2.1. Reducción Polinómica a SSPGNC

Se busca demostrar que el problema de *Longest Path Problem* (LSP) se puede reducir polinomialmente a el problema de *Shortest Simple Path in a Graph with Negative Cycles* ($SSPGNC$).

Expresado mas formalmente, se busca demostrar

$$LSP \leq_p SSPGNC$$

Demostrarlo implica mostrar que las instancias del problema de LSP se pueden transformar, en tiempo polinómico, en instancias de $SSPGNC$. Antes de realizar la demostración vamos a explicar brevemente en que consiste el problema de LSP y $SSPGNC$ respectivamente

Longest Path Problem Dado un grafo $G = (V, E)$, encontrar el camino simple de mayor longitud en G. Buscar un camino simple implica que no se repitan vértices; y con mayor longitud nos referimos a maximizar los pesos de las aristas atravesadas, en un grafo ponderado.

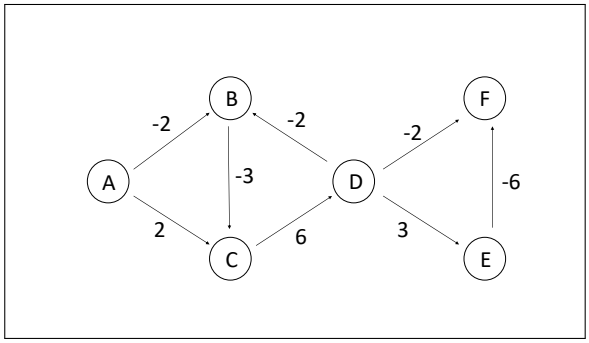
Shortest Simple Path in a Graph with Negative Cycles Dado un grafo $G = (V, E)$ dirigido, ponderado y que puede tener ciclos, encontrar un camino simple a través del cual la suma de los pesos es mínima.

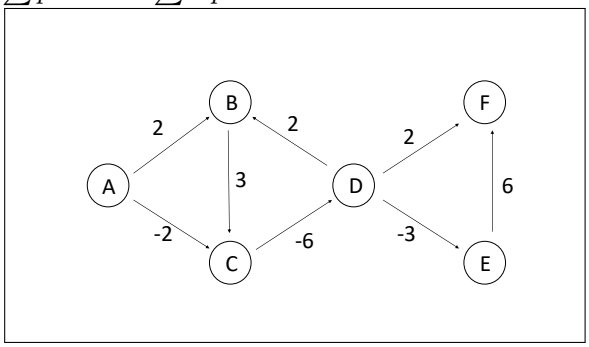
Entonces, supongamos que existe un algoritmo S que resuelva el problema de $SSPGNC$. Este algoritmo nos daría el camino simple mas corto de nuestro grafo. Por lo tanto si lo que buscamos es el camino simple de mayor longitud (mayor peso), entonces bastaría con invertir los pesos de las aristas de nuestra instancia de LSP para transformarla en una instancia de $SSPGNC$.

Luego, dado $G = (V, E)$ una instancia de LSP, se aplica la transformación $T(e) = -e$ con $e \in E$, para todas las aristas; resultando un nuevo grafo $G' = (V', E')$, instancia de $SSPGNC$. Notamos que esta transformación se puede resolver en un tiempo polinomial de $O(|E|)$ (se aplica T a cada una de las aristas de G). Por lo tanto LSP es reducible polinomialmente a $SSPGNC$.

$$\therefore \quad LSP \leq_p SSPGNC$$

Matemáticamente lo que se busca es minimizar la función L, siendo $L = \sum pesos$ y en el segundo caso se busca maximizar la función $L' = \sum pesos' = \sum -pesos = -L$





Para el primer caso, la solucion al camino mas corto es: A-B-C, con un L=-5
Para el segundo caso, luego de la transformación, la solución al camino mas largo es el mismo camino: A-B-C, con un L'=5

2.2. Longest Path Problem es NP-C

Para que el "problema del camino mas largo"(LSP) sea $NP - C$ se tienen que cumplir las siguientes dos condiciones

- $LSP \in NP$: Esto implica que exista un algoritmo que lo certifica de manera eficiente
- $LSP \in NP - HARD$: Para todo $Y \in NP$ se da que $Y \leq_p LSP$

Cumpliéndose ambas podríamos afirmar entonces que $LSP \in NP - C$.

2.2.1. Longest Path Problem pertence a NP

Como se vio antes, hay que encontrar un algoritmo que lo certifique de manera eficiente, esto quiere decir en tiempo polinomial. Entonces, planteamos el problema de LSP como el siguiente problema de decisión

```
Problema: Longest Path Problem
Instancia: Grafo G=(V,E), k un entero
Pregunta:¿Existe un camino simple que recorra el conjunto de
vértices dado con una "longitud"  "suma de pesos" igual a C?
```

Sea $G = (V, E)$ un grafo y $V' = \{v_1, v_2, \ldots, v_n\}$ un subconjunto ordenado de V que forma un camino simple. Dado el certificado (V', k), buscamos validar que V' sea un camino simple de k

elementos. Vemos que este problema se puede resolver en tiempo polinomial mediante el siguiente algoritmo

Algorithm 2 Longest Path Problem

procedure LSP(G, V', k)
Si |V'| = k, continuar, sino devolver No
Verificar que V' sea un camino simple, recorriendo el camino, si lo es devolver Si
En otro caso devolver No
end procedure

Se puede ver que este algoritmo resuelve en tiempo polinomial la certificación, mas precisamente $O(|V|)$. Por lo tanto es correcto afirmar que $LSP \in NP$.

2.2.2. Longest Path Problem pertenece a NP-HARD

Para demostrarlo tenemos que probar que para todo $Y \in NP$ se da que $Y \leq_p LSP$. Esto lo podemos probar encontrando un problema $Y \in NP - C$ que sea reducible polinomialmente a LSP

$$Y \leq_p LSP\,, \quad Y \in NP - C$$

Entonces, vamos a probar que $LSP \in NP - HARD$, probando que el problema del camino Hamiltoniano ($HPATH$) es polinomialmente reducible a LSP.

Hamiltonian Path Problem Dado un grafo $G = (V, E)$, dirigido o no, se busca un camino en el cual se visiten todos los vértices exactamente una vez

Ahora, necesitamos encontrar una transformación eficiente que nos lleve de una instancia de $HPATH$ a una de LSP. Sea un grafo $G = (V, E)$, el mismo va a tener un camino Hamiltoniano si y solo si el camino simple mas largo del mismo tiene longitud n, donde $n = |V| - 1$. Entonces, para transformar una instancia del camino Hamiltoniano en una del problema del camino mas largo, basta con reformular el certificado para que lo se certifique sea si existe un camino simple de longitud $n = |V| - 1$ en G. Por lo tanto queda demostrado que $LSP \in NP - HARD$.
Como $LSP \in NP$ y $LSP \in NP - HARD$, entonces $LSP \in NP - C$.

2.3. ¿Es SSPGNC NP-C?

En base a las dos secciones previas, podemos afirmar solamente que el problema de $SSPGNC$ es $NP - HARD$. Esto se debe a que en la primera sección probamos que

$$LSP \leq_p SSPGNC$$

y en la segunda sección probamos que $LSP \in NP - C$, por lo tanto $SSPGNC \in NP - HARD$. Faltaría demostrar que pertenece a NP para demostrar que es $NP - C$. Entonces, hay que encontrar un algoritmo eficiente que certifique el siguiente problema de decisión

```
Problema: Shortest Simple Path in a Graph with Negative Cycles
Instancia: Grafo G=(V,E), (o,d) par de vértices de G, y C un entero
Pregunta:¿Existe un camino simple de costo menor a C que los una?
```

Sea un grafo $G = (V, E)$ y $V' = \{v_1, v_2, \ldots, v_k\}$, un subconjunto ordenado de k vértices de V que forma un camino simple de peso W, nuestro certificado. Queremos certificar que V' es un camino simple de K elementos

Algorithm 3 Shortest Simple Path in a Graph with Negative Cycles

procedure SSPGNC(G, V', k, W)
Verificar que |V'| = k, si se cumple continuar, sino devolver No
Seguir el camino en V' y acumular el peso.
Si el camino no existe devolver No. Si existe pero el peso acumulado es distinto a W devolver No.
Si el camino existe y el peso acumulado es W devolver Si.
end procedure

Este algoritmo tiene complejidad temporal polinomial, $O(|V|)$. Por lo tanto $SSPGNC \in NP$. Como antes vimos que $SSPGNC \in NP - HARD$, entonces $SSPGNC \in NP - C$

2.4. Concepto de Transitividad y definición de NP-C

Dado un problema Z que es reducible polinomialmente a otro problema Y, y este ultimo a su vez es reducible polinomialmente a un problema X, entonces por propiedad de transitividad, Z es reducible polinomialmente a X.

$$\begin{aligned} & Z \leq_p Y \\ & Y \leq_p X \\ \implies & Z \leq_p X \end{aligned}$$

Todo problema $NP - C$ se puede reducir entre si polinomialmente y todo problema NP se puede reducir polinomialmente a otro $NP - C$.

Si se encontrara un algoritmo que resuelva eficientemente X ($SSPGNC$) entonces, por propiedad de transitividad, se podría resolver cualquier problema $NP - C$ mediante una reducción polinómica, y, cualquier problema NP mediante otra reducción polinómica, esto es equivalente a decir que cualquier problema NP es resoluble en tiempo polinómico o bien que $P = NP$.

2.5. ¿Que podemos decir de la complejidad de problemas reducibles polinomialmente?

Si un problema X se puede reducir polinomialmente a $SSPGNC$, el cual se ha demostrado que es $NP - C$. Entonces tenemos

$$X \leq_p SSPGNC$$

Con la información que tenemos solamente podemos afirmar que X pertenece a $NP - HARD$. Esto significa que X es al menos igual de difícil que cualquier problema de NP. Si además fuera NP entonces, $X \in NP - C$; sabemos que todo problema $NP - C$ es reducible entre si con igual complejidad.

2.6. Clases de Complejidad: P, NP, NP-C

La clase P es el conjunto de problemas de decisión para los que existe un algoritmo que los resuelve en tiempo polinómico. Mientras que la clase NP es el conjunto de problemas de decisión para los que existe un algoritmo que los certifica en tiempo polinómico (de forma eficiente) El conjunto P esta incluido en NP

$$P \subseteq NP$$

pero no esta demostrado si $NP \subseteq P$. Luego esta la clase $NP - HARD$, que son aquellos problemas tales que todos los problemas de NP son reducibles a ellos, por lo que son por lo menos igualmente de difíciles que cualquier problema de NP. Por ultimo la clase $NP - C$ es el conjunto intersección entre NP y $NP - HARD$, es decir, se trata de los problemas mas difíciles dentro de NP.

$$NP - C = NP \cap NP - HARD$$