

Tarea 4

1) a) Algoritmo para cuando la capacidad de la arista e incrementa en 1

El algoritmo sera, primero incrementamos la capacidad de la arista c y luego hacemos una iteración del algoritmo de flujo-refljo

(Ford-Fulkerson), que es buscar un camino aumentante de s a t

(con s el origen y t el destino del flujo), para encontrar el camino

usamos un algoritmo para recorrer graficos como BFS o DFS,

si no existe el camino terminamos, pero si existe entonces vemos cual

es la capacidad minima de las aristas del camino y la llamamos c_f

Luego a las aristas del camino les sumamos c_f al flujo actual y les

sumamos c_f a su refljo, por ultimo le sumamos c_f al flujo optimo

y terminamos

El algoritmo funciona ya que, al aumentar la capacidad en 1 de

solo una arista es posible que se pueda formar un solo camino

nuevo, por lo tanto si solo hacemos una iteración del algoritmo

de flujo-refljo entonces podemos considerar ese posible nuevo

camino y actualizar el flujo optimo

La complejidad es $O(n+E)$, veamos porque, primero encontramos si hay un nuevo camino, como lo hacemos con BFS o DFS, entonces nos toma $O(n+E)$, luego actualizar las aristas del posible camino nos toma a lo más $O(E)$, por lo tanto todo toma $O(n+E)$

o) Algoritmo para cuantos la capacidad de la arista e decremente en 1

El algoritmo sera, primero revisamos si la capacidad de la arista

e es mayor que su flujo actual (b que pasa por ella), si es mayor entonces solo le restamos 1 a la capacidad y terminamos, pero si es

menor o igual haremos cambios, los cambios serán estos:

digamos que e es $e=(u,v)$ (e conecta de u a v) , entonces

usamos BFS para encontrar un camino de s a u y otro de

v a t (pasando por aristas que tengan flujo utilizado mayor a 0,

y el camino existe ya que hay flujo actual en e), luego unimos los

caminos y obtenemos un camino de s a t que pasa por e , lo llamaremos

p , ahora cada arista del camino p la actualizaremos así

les restamos 1 al flujo que pasa por la arista y al reflujo,

despues a la arista e le restamos 1 a la capacidad y por ultimo le restamos 1 al flujo optimo, por ultimo paso del algoritmo hacemos una iteracion de flujo-reflujo (Ford-Fulkerson) como en el algoritmo anterior y terminamos

El algoritmo funciona ya que primero vemos si e tiene espacio para mas flujo (lo que nos dice que disminuir capacidad no nos afecta), si no tiene entonces buscamos un camino de sat que pase por e para disminuir en 1 el flujo que pasa por el camino y asi disminuir la capacidad no cause problema, pero usemos una iteracion de flujo-reflujo por si se genera un nuevo camino y cambie el flujo optimo

La complejidad es $O(n+E)$, debido a que encontrar el camino nos toma $O(n+E)$ (dos BFS), modificar aristas del camino es $O(E)$, una iteracion de flujo-reflujo es $O(n+E)$ (usando el algoritmo anterior), entonces todo toma $O(n+E)$

2) Usaremos una gráfica para representar el lugar (o la ciudad o la colonia), ahora veamos como sera la gráfica cada vértice representa una esquina, y si un vértice u y v representan las esquinas a y b respectivamente entonces si existe una calle entre las esquinas a y b entonces ponemos una arista bidireccional entre u y v , con capacidad 1 la arista / por ultimo la casa sera el vértice s y la escuela el vértice t , ahora con la gráfica aplicamos el algoritmo de flujo-refljo (Ford-Fulkerson) y si el flujo maximo obtenido es igual o mayor a 2, significa que si pueden ir a la misma escuela.

Esto funciona debido a que modelamos la ciudad como una gráfica entonces buscamos dos caminos que son iguales en aristas para llegar de s a t , esto lo logramos con flujo-refljo y capacidad 1 en las aristas, ademas supongamos que solo 1 niño puede pasar por una calle, de otra manera (si camina uno puede pasar por una

acera de la calle) entonces el problema se trivializa
a encontrar un camino de sat y ya no usaremos flujos

3) Primero representaremos a los postes como una gráfica, cada poste sera un vértice de la gráfica, y dos postes o vértices p_i y p_j tendrán una arista si los uni si la distancia entre los postes estuviera entre d y $2d$ (esto lo calculamos sabiendo sus coordenadas), luego de hacer la gráfica (o tenerla si nos la dan) vamos hacer una búsqueda BFS empezamos, usamos una cola y metemos al poste p_s y mientras la pila no este vacía hacemos esto, sacamos a un poste de la cola, si el que sacamos es p_t , regresamos que si hay camino, en otro caso marcamos al sacado como visitado y para cada vecino no visitado del que sacamos lo metemos a la cola, si la cola se queda vacía regresamos que no hay camino

El algoritmo Encierra ya que modelamos el problema como si fuera una gráfica y solo encontramos el camino entre dos vértices

La complejidad del algoritmo es $O(n^2)$, esto debido a q

que modelen la grafica $O(n^2)$, cada nodo conoce la distancia hacia los demás nodos \rightarrow la parte de ver si existe un camino (BFS) nos toma $O(|E| + |N|)$, ya que revisa todos los nodos y posibles aristas (que al más se n^2), y es algo eficiente y más si no tenemos que modelar la grafica (Si nos da o reditizamos para multiples buscadores)

4) Recordemos un resultado de graficas "Una grafica dirigida es aciclica si un recorrido DFS no genera una arista e , tal que e conecta el nodo a hacia el nodo b , con b siendo un ancestro de a (back edge)", por lo tanto podemos usar DFS, entonces el algoritmo es , primero creamos un arreglo para decir si un nodo esta visitado y un arreglo para decir si un nodo esta en la pila, marcamos la todos como no visitados y fuera del stack, ahora iteramos en todos los vertices de G , y llamamos a una funcion auxiliar que recibe el vertice, los visitados y la pila , si esta funcion auxiliar regresa True entonces hay un ciclo y terminamos pero si llego de iterar todos los vertices regresamos que no hay ciclos, veamos la funcion auxiliar $\text{aux}(v, \text{visitados}, \text{pila})$, primero revisamos si v esta en la pila , si esta regresamos True , en caso que no seguimos, ahora revisamos si ya fue visitado , si ya fue visitado regresamos False , en otro caso continuamos , marcamos a v como visitado y que esta en la pila , ahora

por cada vértice u adyacente a v , llamamos a la función auxiliar con u , los visitados y la pila, si algún vecino u regresa true con la función auxiliar regresamos true, pero si terminamos de iterar los vecinos sin ningún true, marcamos a v como fiera de la pila y regresamos false, si en algún momento obtenemos true detenemos el algoritmo.

El algoritmo funciona ya que usamos el resultado del nudo, y checamos si algún nodo se encuentra a si mismo en la pila durante la función auxiliar para poder simular que encontramos un back edge, ya que mientras procesamos a los vecinos del nodo llegamos a él (un ciclo)

La complejidad del algoritmo es $O(|V| + |E|)$, ya que procesamos todos los vértices y checamos todas las aristas (al hacer recursión en b: vecinos), por lo tanto los vértices son $O(|V|)$ y los aristas son $O(|E|)$, no se como hacerlo en solo $O(|V|)$ debido a que al ser una gráfica dirigida, si un vértice v es vecino de u

puede ser que u no sea vecino de v , por lo tanto en el peor caso tenemos que revisar todas las aristas de la grafica para revisar que no existan ciclos y ademas una grafica sin ciclos tiene a lo mas $(n-1) + (n-2) + \dots + 1$ aristas y eso es mas que los n vertices, entonces no podemos entrar la E de la complejidad

5) Supongamos, para generar una contradicción, que T' no es un árbol de peso mínimo de G' , entonces existe un árbol A que es un árbol de peso mínimo de G' y tiene peso total menor que T' .

Ahora, sea D el conjunto de aristas, tales que la arista e está en D si e está en T pero no en T' , ahora si le agregamos las aristas de D a A , obtenemos un árbol de peso mínimo de G , llamémoslo B , pero aquí pasa algo como A es un árbol de peso mínimo y tiene menor peso total que T' entonces B debe tener menor peso total que T , por lo tanto B sería árbol de peso mínimo de G y T no!, pero esto es una contradicción, y la contradicción surge de suponer que T' no es un árbol de peso mínimo de G' , por lo tanto concluimos que T' debe ser un árbol de peso mínimo de G' \square

6) Primero recordemos que los árboles generados (spanning trees) T_0 y T_1 de G , son diferentes si al menos existen las aristas a, b en G tales que $a \in E(T_0)$, $a \notin E(T_1)$ y $b \in E(T_0)$, $b \in E(T_1)$

entonces veamos como obtener la secuencia.

empezamos con $T_0 = T$, ahora iteramos una variable i de 0 a d (donde d es el numero de árboles generados de G vértices), entonces en la iteración $i=j$, revisamos si el árbol T_j es igual a R si pasa esto entonces ya terminamos, en otro caso le agregamos una arista que este en R pero no en T_j (como son distintos podemos hacerlo), ahora llamemos a la nueva grafica A , notemos que A tiene un ciclo, y como R es un árbol (\wedge los árboles no tienen ciclos) entonces el ciclo de A al menos tiene una arista que no esta en R , y la eliminamos de A obteniendo el árbol B al cual agregamos al la secuencia como T_{j+1} , y aumentamos a i , repetimos este proceso hasta tener a R

t

El código funciona ya que al inicio a T le agregamos una arista que esté en R pero no en T , y aunque una arista que forme un ciclo, por

mantener que sea arbol) que no estaba en R , por lo tanto poco a poco volvemos T a R , y como agregamos los que estan en R pero no en el arbol actual y eliminamos los que estan en el actual pero no en R , es la secuencia mas corta

Todo el algoritmo toma tiempo, que depende de cual sea la complejidad de agregar, eliminar y encontrar las aristas que pedimos, entonces digamos que encontrar una arista nos toma $O(|E|)$ y buscar un ciclo nos toma $O(|V| + |E|)$ usando BFS, por lo tanto cada iteracion nos toma $O(|V| + |E|)$ y esto lo repetimos a lo más el numero de aristas de R que son $|V|-1$, por lo tanto todo nos toma $O(|V| * (|V| + |E|)) = O(|V|^2 + |V|*|E|)$

7) a) Algoritmo lista de adyacencias

Creamos una nueva lista de adyacencias G^2 y copiamos la lista de adyacencias de G

ahora hacemos un ciclo, para cada arista v de G , y hacemos

otro ciclo interno, para cada vértice u en la lista de adyacencias

de v , hace otro ciclo, para cada vértice w en la lista de adyacencias

de u hacemos esto agregamos w a la lista de adyacencias

de v (si ya está en las adyacencias no agregamos) de G^2 , terminamos

cuando los ciclos terminen

El algoritmo funciona ya que usamos los tres ciclos anidados

para encontrar caminos de longitud 2 de v a w y luego

si encontramos uno agregamos la arista de v a w

La complejidad del algoritmo es $O(|V| \times |E|)$, ya que copiar la

lista de adyacencias nos toma $O(|V|)$ (solo copiamos listas, pero

depende algo de la complejidad de como se haga)) luego

terminemos un ciclo sobre todos los vértices, que toma $O(|V|)$,

ahora veamos los dos ciclos internos, notemos que en ambos ciclos usamos las aristas para llegar a los vértices (ya que son vecinos) por lo tanto a lo más en ambos ciclos recorremos todas las aristas por lo tanto toma $O(|E|)$, por lo tanto nuestros ciclos toman $O(|V| \times |E|)$, y todo toma $O(|V| + (|V| \times |E|)) \in O(|V| \times |E|)$ que es algo eficiente

o) Algoritmo matriz

Creamos una matriz nueva G^2 como una copia de G , sea $|n| = |V|$, ahora iteramos i de $1 \text{ a } n$, luego dentro del ciclo iteramos k de $1 \text{ a } n$, dentro de este ciclo hacemos otro ciclo , iterando j de $1 \text{ a } n$, y dentro de este ciclo revisamos si $G[i][k] == 1$ (existe la arista de i a k) y $G[k][j] == 1$ (existe la arista de k a j) , si se cumple lo anterior marcamos a $G^2[i][j] == 1$ y rompemos el ciclo de k (terminamos de iterar) , si no se cumplió para ninguna k (el ciclo se terminó) entonces marcamos $G^2[i][j] == 0$ al final de los ciclos terminamos

El algoritmo funciona ya que igual que en el primer algoritmo buscamos los caminos de la j caj de longitud 2, pero lo hacemos mas eficiente, ya que revisamos todos los posibles aristas (incluyendo las que no existen), por lo tanto si funciona

La complejidad es $O(|V|^3)$, esto debido a nuestros ciclos considerados son los tres son de la n, por lo tanto todo en conjunto es $O(n^3) = O(|V|^3)$ y es algo eficiente pero no tanto como el algoritmo anterior, ya que en el anterior cambiaron $O(|V|^2)$ por $O(|E|)$, pero como aqui revisamos todos los posibles aristas (que son $n(n-1)$) entonces nos toma mas tiempo, el copiar la matriz G toma $O(|V|^2)$ por lo tanto no "afecta" la complejidad total, y todo toma $O(|V|^3)$, se es algo eficiente

8) o) Esto es posible si hacemos las siguientes modificaciones:

crearemos una grafica H usando a la red de flujos G como base, para cada vértice v de G , crearemos dos nuevos vértices v_1 y v_2 , y unimos a v_1 y v_2 con una arista e que tendrá la capacidad de v (los vértices v_1 y v_2 no tienen capacidad), de esta forma "partimos" todos los vértices de G , para las aristas de G hacemos esto, pero la arista e de G si e une a los vértices u_1 y v de G , entonces la arista e en H unirá a los vértices u_2 y v_1 , y tendrá la misma capacidad de e , entonces al final H tendrá $2V$ vértices y $E+V$ aristas (con V el num de vértices de G y E el num de aristas de G), de esta manera podemos sacar el flujo maximo de H usando el algoritmo flujo-reflujo (Ford-Fulkerson) y este flujo maximo es el mismo de G , debido tratamos los vértices de G como aristas en H , por lo tanto podemos usar flujo-reflujo y respetar las capacidades.

de los vértices de G , solo los simularía como aristas pero respetando todas las "conexiones" de G

e) Algoritmo

basado en la malla creamos una gráfica G , donde cada punto de la malla va a ser un vértice de G con capacidad 1 y tendremos aristas bidireccionales con capacidad 1 que van a los vértices tales que los puntos que representan en la malla son adyacentes, ahora agregaremos dos vértices nuevos, el vértice origen que tendrá aristas uniendo al origen hacia a los m vértices de arriba, y el otro vértice final que tendrá aristas uniendo a los vértices en la frontera hacia el vértice destino, por lo tanto G tiene $n^2 + 2$ vértices y $2n^2 - 2n + m + 4n - 4$ aristas ($2n^2 + m + 2n - 4$ aristas)

Entonces ahora creamos la gráfica H , usando el método descrito en el punto anterior (solo ignorando a los vértices origen y destino, no los partimos), entonces ya con la gráfica H usamos

el algoritmo de flujo-reflujo (Ford-Fulkerson) para calcular el flujo maximo de origen a destino, ahora llamens el flujo maximo como f , si f es menor a m entonces no existen las m trayectorias, en otro caso si existen

El algoritmo funciona ya que usamos flujos para ver si podemos "llegar" de los m puntos hacia las fronteras, usando vertices auxiliares para unir a todos los m puntos y para unir a las fronteras, ademas usamos que cada vertice tiene capacidad 1, entonces con la modificación del ejercicio anterior podemos usar flujo-reflujo y obtener la respuesta.

Veamos la complejidad, primero construir a G nos toma $O(n^2 + m)$, ya que ponemos cada vertice y luego ponemos los arcos (ante, de uno, cuantos son, $O(n^2 - 2n + m + 4n - 4) \in O(n^2 + m)$ arcos y $O(n^2 + 2) \in O(n^2)$ vertices), ahora transformar G a H nos toma $O(n^2 + m)$ (solo con constante mucho mas grande), ya que

hacemos nuevos vértices y aristas ($O(2(n^2+2))$ (G) $O(n^2)$ vértices)

y $O(n^2+2+2n^2-2n+m+4n-4) \in O(n^2+m)$ aristas) por lo tanto

generar a H nos toma en total $O(n^2+m)$ (muy grande)

y usar flujo-reflido (Ford-Fulkerson) nos toma $O(F * (|V| + |E|))$

(con F el flujo máximo) como vimos el 13/11/23, entonces nos toma

toma $O(F * (n^2+m+n^2))$ y en total todo toma

$O([F * (n^2+m+n^2)] * n^2+m)$ en total y es algo eficiente, espero

$$\in O((F+1)(n^2+m))$$

Explicación de BFS y DFS

• BFS

empezando en un nodo, lo revisamos y marcamos como visitado luego revisamos a sus vecinos y despues a los vecinos de los vecinos, pero primero revisamos a todos los vecinos antes de los vecinos de los vecinos
esto mediante el uso de una cola donde vamos poniendo nodos a revisar, para asi ir revisando por niveles (ignorando a los ya visitados)
para poder terminar)

• DFS

empezando en un nodo, luego vamos explorando los vecinos y repitiendo este proceso, al iniciar DFS en un nodo lo metemos a una pila
y cuando terminamos con sus vecinos y de revisar al nodo, lo sacamos de la pila