Técnicas de Diseño de Algoritmos - Algoritmos III

1er Cuatrimestre 2024 - 2do Parcial

Nombre y Apellido	LU	#Ord
.....	.....	.....

Tema	Ej A / B	#hojas
A	.....	.....

*Duración: 4 horas. Este examen es a **libro cerrado**. Para aprobar se debe alcanzar 50 puntos.*

Contents

1 Preguntas opción múltiple (60 ptos.)

*Marcar con una cruz (X) **todas** las respuestas correctas.*
*Un ejercicio con todas sus opciones correctas marcadas y todas sus incorrectas **no** marcadas suma 6 puntos.*

Pregunta 1 Sea $k \in \mathbb{R}$ un número real y G un grafo pesado con m aristas y n vértices tal que toda arista tiene peso k. ¿Qué algoritmo/s puede/n utilizarse para encontrar un AGM de G en menor complejidad?

- ☐ Kruskal
- ☐ Djikstra
- ■ DFS
- ☐ Prim
- ■ BFS

1.0.1 Solución 1:

Como todas las aristas tienen mismo peso cualquier AG es AGM con peso total $k(n-1)$, por lo que cualquiera de los 5 algoritmos sirve. Los de menor complejidad son BFS y DFS que son lineales.

Pregunta 2 Lean quiere armar un grafo muy simple para el segundo parcial de Algo III, por lo que crea un grafo G conexo con n vértices y $\frac{n(n-1)}{2}$ aristas. Define la longitud de cada arista según su índice de la siguiente manera: para todo i tal que $1 \leq i \leq \frac{n(n-1)}{2}$, define la longitud l de e_i como $l(e_i) = i$. Pero tiene un problema: el grafo le quedó muy grande para las diapositivas debido a las aristas tan largas, por lo que quiere construir un subgrafo generador H de G que minimice la longitud total de las aristas que se muestran en pantalla. Naturalmente, también quiere que el grafo resultante sea conexo. ¿Qué algoritmo/s puede usar Lean para generar H?

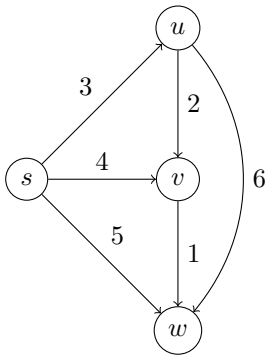
- [x] Prim
- [ ] DFS
- [x] Kruskal
- [ ] Alcanza con devolver las $n-1$ aristas más livianas.
- [ ] Dijkstra
- [ ] BFS

Aparte, Lean se pregunta ¿Cuántos AGM distintos puede puede tener este grafo?

- [ ] Tiene exactamente 2 AGMs.
- [x] Solo tiene un AGM.
- [ ] Tiene más de 2 AGMs.

1.0.2 Solución 2:

En este caso como los pesos son distintos no podemos usar ni DFS ni BFS. Prim y Kruskal van a funcionar, queda ver que dijkstra y devolver las aristas $n-1$ más livianas tampoco. Si ponemos las $n-1$ aristas más livianas pero que no conectan los n vértices entonces nos va a devolver un grafo disconexo. Para dijkstra es fácil ver con el siguiente ejemplo:



Prim o Kruskal nos van a devolver un AGM con peso $3+1+2$. Mientras que Dijkstra devuelve un AG con peso $3+4+5$.

Pregunta 3 Decidir cuáles de las siguientes afirmaciones son verdaderas para todo grafo pesado conexo $G = (V, E)$:

- [x] Si $e \in E$ es una de las aristas con peso mínimo de G entonces pertenece a por lo menos un AGM.
- [x] Si $e \in E$ es una arista puente de G entonces pertenece a todos los AGM.
- [x] Si $e \in E$ tiene peso menor estricto a todas las otras aristas de G entonces e pertenece a todos los AGMs de G.
- [ ] Si $e \in E$ tiene peso mayor estricto a todas las otras aristas de G entonces e no pertenece a ningún AGM.
- [ ] Si $e \in E$ pertenece a un ciclo C, entonces hay un AGM de G que no contiene a e.
- [ ] Si $|E| > n - 1$ entonces hay alguna arista que no pertenece a ningún AGM.

1.0.3 Solución 3:

1. Supongamos que no y sea un AGM cualquiera que no contenga a e, si agregamos e vamos a formar un ciclo que incluye a e. Si miramos el otro camino que no pasa por e entre los vértices que unió podemos sacar cualquier eje del mismo y nos vuelve a quedar un árbol, como e era de peso mínimo o sacamos una arista con mayor peso o sacamos una de peso mínimo también, en cualquiera de los dos casos conseguimos un AGM que contiene a e.
2. Como es una arista puente conecta dos componentes conexas, si no la incluimos nunca podemos tener un árbol.
3. Mismo argumento que el primer item solo que acá llegamos a un absurdo.
4. Elegir una arista que es un puente con peso mayor que el resto.
5. Basta tomar un ciclo donde la arista que pertenece al ciclo menor estricto que el resto.
6. Tomar AGM con todas las aristas del mismo peso y usar item 1.

Pregunta 4 Facundo tiene un grafo conexo y quiere armar un AGM, pero no quiere usar ninguno de los algoritmos conocidos. Luego, se le ocurre el siguiente algoritmo: va a encontrar ciclos iterativamente, y cada vez que encuentre uno va a quitar el eje más pesado del mismo. Para buscar cada ciclo va a usar DFS, explotando la estructura que devuelve el algoritmo para encontrar un ciclo cualquiera en el menor tiempo posible. Este proceso continuará hasta que el grafo no tenga más ciclos. Decidir cuáles de las siguientes afirmaciones son verdaderas:

- [ ] El algoritmo propuesto por Facundo tiene complejidad $\Theta(nm)$ en peor caso (usando los algoritmos vistos en la materia).
- [x] El algoritmo propuesto por Facundo tiene complejidad $\Theta(m^2)$ en peor caso (usando los algoritmos vistos en la materia).
- [x] El algoritmo propuesto por Facundo es correcto (es decir, al final del procedimiento tendrá un AGM).
- [ ] El algoritmo propuesto por Facundo es incorrecto (es decir, al final del procedimiento no tendrá un AGM).

1.0.4 Solución 4:

1. No vale por item siguiente.
2. Como un AGM tiene $n-1$ aristas y tenemos que sacar las restantes, si hay, y si $m > 2n$ vamos a tener que eliminar $O(m)$ aristas es decir iterar $O(m)$ veces y cada iteración toma $O(m)$ por ser conexo, lo que da que nuestro algoritmo es $O(m^2)$.
3. G es conexo y tiene ciclos, eliminar el eje más pesado de un ciclo no deja que deje de ser conexo, luego de sacar todos los ciclos tiene un grafo sin ciclos y conexo, es decir un árbol.
4. Invalido por item anterior.

Pregunta 5 Sea $G = (V, E)$ un grafo completo y pesado, con función de peso $w : E \to \mathbb{R}_{>0}$. Sean s y t dos vértices de G, y sea D un DAG de caminos mínimos desde s a t de G. Decidir cuáles de las siguientes afirmaciones son verdaderas.

- [x] Una arista de D puede pertenecer a más de un camino mínimo de G entre s y t.
- [ ] El camino máximo entre s y t no puede pertencer a D.
- [ ] Toda arista de G aparece con exactamente una dirección definida en D.
- [ ] Toda arista $vz \in E$ que pertence al DAG de caminos mínimos satisface $d(s, v)+w(vz)+w(zt) = d(s, t)$.
- [ ] D tiene a lo sumo $|V| - 1$ aristas.

1.0.5 Solución 5:

1. Pensar en un grafo G que tiene n vertices (incluyendo a s y t). Ponemos todas las aristas que salen de s al resto de los vértices con el mismo peso (sin usar a t) y todas las aristas que salen de los vértices, también con el mismo peso a t, entonces tenemos $n - 2$ caminos.
2. Falso, considerar todas las aristas con el mismo peso
3. Una arista podría no aparecer, por ejemplo en el grafo original podríamos tener una arista puente que vaya a una componente que no llegue a t.
4. Esto es falso, porque nada te garantiza que estas a un salto de distancia de t.
5. Mismo ejemplo que el primero pero contando $2(n - 2)$ aristas.

Pregunta 6 Tuki tiene un grafo pesado G con n nodos, m_1 aristas con peso positivo y m_2 aristas con peso negativo. En su modelo los pesos representan dinero: un eje con peso positivo indica que se debe pagar al recorrerlo (*ejes de pago*), mientras que uno con peso negativo indica que se cobra (*eje de cobro*). Naturalmente, los ejes de cobro no forman ciclos.
Dados dos nodos v y w, está interesado en conocer el mayor monto final que se puede obtener yendo de v a w y luego volviendo a v, asumiendo que a la ida solo se pueden usar *ejes de pago*, y a la vuelta solo *ejes de cobro* ¿Cuál es la complejidad más ajustada en la que puede resolver el problema, usando los algoritmos vistos?

- [x] $O(n + m_1 \log n + m_2)$
- [ ] $O(m_1 \log n + m_2 n)$
- [ ] $O((m_1 + m_2)n)$
- [ ] $O((m_1 + m_2) \log n)$
- [ ] $O(n + m_1 + m_2)$

1.0.6 Solución 6:

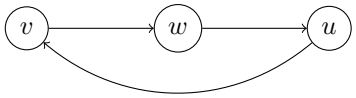
La idea era minimizar la ida y maximizar la vuelta, para la ida se corría Dijkstra ya que los pesos son positivos esto nos daba $O(m_1 log(n))$ y a la vuelta podemos simplemente usar camino máximo en DAGs, esto nos da $O(n + m_2)$, sumados dan la complejidad marcada.

Pregunta 7 Rocío tiene un grafo dirigido sin pesos $G = (V, E)$ con $|V| = n$ y $|E| = m$, y dados dos nodos v y w quiere encontrar el camino simple mínimo de longitud par que los une (es decir, si C es el conjunto de caminos simples de v a w de longitud par en G, entonces Rocío quiere un camino simple de longitud mínima de C). Para eso plantea el siguiente modelo: como conjunto de nodos usa los pares (v, b) con $v \in V$ y $b \in \{0, 1\}$, donde b va a ser usado para indicar la paridad del camino, y por cada eje $xy \in E$ agrega los ejes $(x, 0) \to (y, 1)$ y $(x, 1) \to (y, 0)$. Luego, Rocío propone ejecutar BFS desde el nodo $(v, 0)$ y ver la distancia al nodo $(w, 0)$. Decidir cuáles de las siguientes afirmaciones son verdaderas:

- ☐ Rocío podría ejecutar BFS desde $(w, 0)$ y ver la distancia a $(v, 0)$ y la respuesta no cambiaría.
- ■ Rocío podría ejecutar BFS desde $(v, 1)$ y ver la distancia a $(w, 1)$ y la respuesta no cambiaría.
- ■ Su modelo es incorrecto, no le va a devolver la longitud de un camino simple de long. par mínima.
- ☐ La complejidad del algoritmo propuesto por Rocío es $\Theta((n + m)^2)$ en peor caso.
- ☐ Su modelo es correcto, y le va a indicar la longitud de un camino simple de long. par mínima.

1.0.7 Solución 7:

1. Como es dirigido puede ser que de w no salgamos a ningun lado asi que no va a encontrar un camino $v \to w$.
2. Como todas las conexiones son las mismas es lo mismo buscar un camino por $(v, 0) \to (w, 0)$ que por $(v, 1) \to (w, 1)$.
3. Considerar el siguiente ejemplo y construir el modelo propuesto por Rocío.



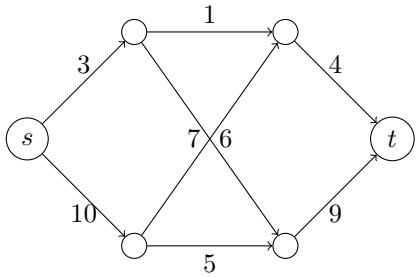
4. la cantidad de aristas y vértice se multiplicó por una constante asi que sigue siendo lineal.
5. Inválido por contraejemplo del tercer item.

Pregunta 8 Dada G una red de flujo con capacidades enteras y nodos distinguidos s y t. Decidir cuáles de las siguientes afirmaciones son verdaderas.

- ☐ Si todos los arcos de G tienen capacidad distinta entonces G tiene un único corte mínimo.
- ■ Si G tiene flujo máximo impar entonces al menos un arco tiene capacidad impar.
- ☐ Si G es acíclico entonces para todos los flujos posibles el grafo residual G_f también es acíclico.
- ☐ Hay un flujo de valor $k \iff$ los arcos que salen de la fuente s tienen capacidades que suman al menos k y los arcos que entran al sumidero t tienen capacidades que suman al menos k.
- ☐ Si todos los arcos de G tienen capacidad impar entonces existe un flujo máximo f tal que $f(e)$ es impar para todo arco e.

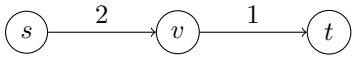
1.0.8 Solución 8:

1. La siguiente red tiene al menos dos cortes de capacidad mínima.

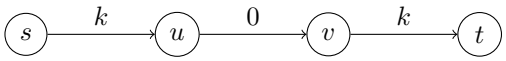


2. Esto es el contrarrecíproco de si todos los arcos tienen capacidad par entonces el flujo tiene que ser par de la práctica.

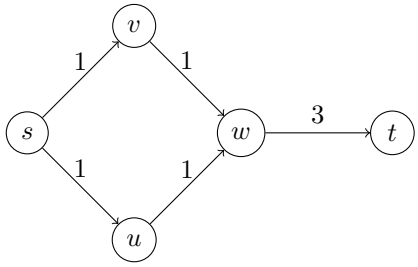
3. Calcular la red residual para la siguiente red con capacidades y función de flujo $f(sv) = 1, f(vt) = 1$



4. Considerar la siguiente red con capacidades



5. Considerar



Pregunta 9 Sea G una red con nodos distinguidos s y t. Decidir cuáles de las siguientes afirmaciones son verdaderas:

☐ Sea C un corte mínimo de G y e una arista de G que cruza el corte, yendo de la componente de s a la de t. Es cierto que si aumentamos su capacidad entonces el flujo máximo aumenta.

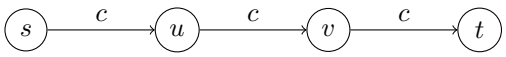
☐ Si se le suma una constante $\delta > 0$ a todas las capacidades de los arcos de G entonces el flujo máximo no cambia.

■ Para todo $n \in \mathbb{N}$ existe una red de $n + 2$ vértices con 2^n cortes s-t distintos, todos con capacidad mínima.

☐ En cualquier flujo máximo no hay ciclos que lleven flujo mayor que 0.

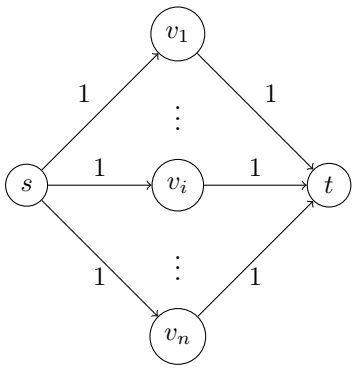
1.0.9 Solución 9:

1. Considerar la siguiente red.

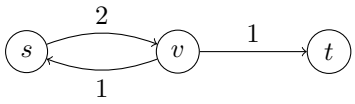


2.

3. La forma más fácil de ver esto es poner $n + 2$ vértices dónde conectamos la fuente a todos los vértices menos el sumidero con la misma capacidad (podía ser 0, y luego de todos los vértices al sumidero (menos la fuente) con la misma capacidad que usamos antes. Cualquier corte va a tener la misma capacidad constante y tenemos 2^n formas de elegir los cortes (o esta el vértice i-ésimo o no.



4. Considerar la siguiente red con capacidades.



Pregunta 10 Gracias a la carrera de computación Sasha consiguió empleo en una fábrica que construye herramientas. En la fábrica tienen un conjunto P de $p = |P|$ piezas de tipo 1 que se pueden acoplar a un conjunto Q de $q = |Q|$ piezas de tipo 2. Lamentablemente, no toda pieza de tipo 1 puede acoplarse a una de tipo 2.
Más puntualmente, la fábrica conoce, para cada pieza $x \in P$ el conjunto $N_x \subseteq Q$ de piezas de tipo 2 a la que puede acoplarse x. Aparte, se sabe que N_x siempre tiene a lo sumo 15 piezas.
Su jefe le pide que consiga con *algún algoritmo de flujo* la cantidad máxima de piezas acoplables. Sasha nos pide ayuda para modelar este problema y que le demos su complejidad como pista para que elle lo pueda resolver. ¿Cuál es la complejidad más ajustada para resolver este problema con un modelado de flujo, en términos de p y q?

- ■ $O(pq)$
- ■ $O(\min\{p, q\}(p + q))$
- □ $O(\min\{p, q\}pq)$
- □ $O((p + q)^3)$
- □ $O(p(p + q))$

1.0.10 Solución 10

Lo que nos piden es un matching máximo, para esto podemos definir la siguiente red de flujo.

Hay p piezas tipo 1 que podemos acoplar a q piezas tipo 2, salimos de la fuente a con arcos a vértices que representan una pieza de tipo 1 con capacidad 1 y de cada pieza tipo 1 salimos hacia sus posibles piezas tipo 2[§§]ambién con capacidad 1 finalmente de cada pieza tipo 2 salimos al sumidero con la cantidad que tenemos de cada una. La red queda de la siguiente manera:

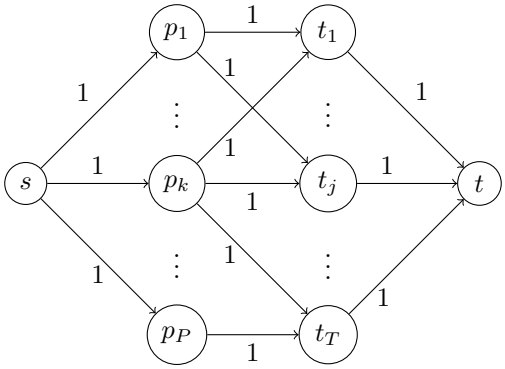


Figure 1: Modelo de Sasha.

Analizando la complejidad queremos estimar arcos y vértices. Esto nos da como mucho $O(pq)$ arcos, además hay $p+q+2$ vértices y sabemos que el flujo máximo con FF normal es $O(|E|f)$, por nuestro modelo que f está acotado por el mínimo de los cortes $\{s, V - \{s\}\}$ y por el corte $\{V - \{t\}, t\}$ y esto es $\min(p, q)$.[‡‡]

2 Ejercicio a desarrollar (40 ptos.)

Marcar con una cruz el cuadrado □ *del ejercicio que vas a entregar.*
Solo se entrega 1 de los 2 ejercicios.

[§§] t

[‡‡] Como $F_{\max} \leq \min\{p, q\} \leq |V||E|$ podemos ignorar la cota de EK.

2.1 ☐ Problema A

La empresa *Tuki logística* está planificando rutas para sus repartidores, quienes deben entregar paquetes en diferentes puntos de una ciudad. Los repartidores tienen un límite de tiempo que pueden caminar sin descansar.

La empresa conoce el mapa de la ciudad, el cual se representa con un conjunto de nodos V (ubicaciones) y un conjunto de aristas E (calles) uniendo pares de nodos. También conocen un subconjunto $B \subseteq V$ de nodos donde los repartidores pueden tomar descansos y recargar energía. Para cada calle $e \in E$, se sabe el tiempo $t(e)$ que toma recorrerla, y cada repartidor es capaz de caminar k minutos antes de necesitar un descanso.

El día de hoy, *Tuki logística* quiere descubrir el menor tiempo en el cual un repartidor puede llegar desde el club Luna de Avellaneda (LA) hasta Ciudad Universitaria (CU), asegurándose que no se agote en el camino.

a) Modele el problema como un problema de recorrido en un grafo ponderado. Justifique cómo un camino en su modelo representa la solución al problema original.

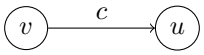
b) Describa un algoritmo eficiente para resolver el problema, con una complejidad temporal de $O(k|E|\log(k|V|))$. Incluya pseudocódigo si es necesario para mayor claridad. Describa las estructuras de datos utilizadas y justifique la correctitud y la complejidad del algoritmo.

2.2 Solución A

2.2.1 a)

La idea acá es modelar en cada vértice cuanto cansancio tiene el repartidor hasta llegar ahi. Entonces por cada vértice v lo vamos a convertir en la tupla (v, i) con $0 \leq i \leq k$ que representa si llego con i cansancio total. Es decir si teníamos vw conectados con peso c en nuestro remodelado vamos a tener los vértices $(v,0),(v,1),\cdots,(v,k)$ y vamos a conectar cada uno al vértice (si hay) $(w,c),(w,c+1),\cdots(w,i+c)$ y además $i + c \leq k$.

Es decir, si teníamos la siguiente conexión dentro del grafo



La vamos a transformar en

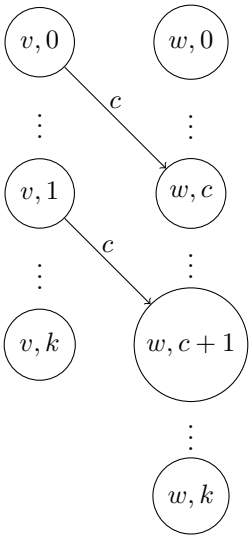


Figure 2: Modelando el tiempo en los vértices.

El dibujo anterior es el caso en el que los dos vértices no son de descanso o si salimos de un vértice que no es de descanso a uno de descanso. En el tercer caso, de descanso a cualquier otro vértice vamos a volver a 0 el tiempo de salida es decir, vamos a ir siempre al vértice con valor de su tupla $0+c$, sin importar con cuanto cansancio llegamos ahi. En el caso de que nos pasemos de k cansancio vamos a omitir la arista, también la vamos a omitir si ya de entrada $c(e) > k$ ya que es un camino que no nos importa porque siempre se va a

cansar de más pase por donde pase antes o después. Además para avellaneda solo vamos a agregar el vértice $(A, 0)$ y sus conexiones correspondientes. Una última cosa que vamos a hacer y por comodidad es conectar todos los vértices de CU a un vértice $final$ con arcos de peso 0, así solo tenemos que fijarnos la distancia a ese vértice luego de correr el algoritmo.[‡‡]

"Justificación": Veamos dos cosas, una es que nunca se puede pasar de k cansancio y que no aparecieron caminos nuevos, ni sacamos caminos preexistentes. Lo primero es porque cualquier camino va del cansancio actual a un vértice donde su segundo elemento de la tupla es igual al cansancio actual más el peso de la arista, y esto nunca se pasa de k así que cualquier camino respeta cansancio. Lo segundo se ve de que solo conectamos vértices que ya estaban conectados entre sí, por lo cual no agregamos caminos nuevos y además incluimos todos los vértices que ya estaban menos los de $w(e) > k$ y donde nos pasábamos de cansancio (que eran caminos inválidos para nuestra solución).

Poniendo todo junto, por cada vértice creamos k vértices nuevos y por cada arista creamos a lo sumo k aristas nuevas, entonces en nuestro modelo vamos a tener $k|V|$ vértices y a lo sumo $k|E|$ aristas.

Luego nuestro camino lo vamos a encontrar corriendo Dijkstra desde $(LA, 0)$ a $final$.

2.2.2 b)

Como lo único que hicimos fue construir otro grafo que posee k vértices por cada vértice del original y k arcos por cada arco del original tenemos un grafo de $k|V|$ vértices y $k|E|$ arcos, si usamos Dijkstra para buscar camino mínimo sabemos que podemos conseguir la complejidad

$$O((k|V|+k|E|)\log(k|V|)) = O(k|E|\log(k|V|))$$

2.3 ☐ Problema B

En el torneo de voley TYVA el organizador Ramiro se cansó de armar el fixture a mano, por lo que contrató a Sasha para que desarrolle un algoritmo que arme los fixtures automáticamente.

La liga consiste de un conjunto E de equipos y un conjunto P de partidos que deben disputarse, siendo un partido un par $p = (e, e')$ con $e, e' \in E$ y $e \neq e'$. El torneo además tiene un conjunto T de turnos disponibles, siendo un turno $t \in T$, una tupla $(cancha, horario)$. En cada turno se puede jugar a lo sumo 1 partido. Cada equipo $e \in E$ tienen una disponibilidad $D_e \subseteq T$ de turnos en los que puede jugar (donde nunca informa que puede jugar en dos turnos que sean en el mismo horario). Aparte, para cada equipo e hay un turno $f_e \in T$ que es su **turno favorito**.

Para cada partido $p = (e, e')$, se busca encontrar un turno $t \in T$ en el que pueda realizarse. Para que el partido se pueda jugar, ambos equipos deben estar disponibles en ese turno, o bien ese turno debe ser uno de los favoritos de alguno de los dos equipos. Aparte, para evitar favoritismos se desea evitar que se jueguen más de K partidos en una misma cancha.

En base a estos datos se busca armar un fixture, que consiste en encontrar turnos disponibles para todos los partidos. En caso de no ser posible, se debe devolver la máxima cantidad de partidos que pueden organizarse siguiendo las restricciones.

Se pide:

1. Modelar el problema como un problema de flujo
2. Dar una interpretación a cada unidad de flujo y cada restricción de capacidad.
3. Justificar que el modelo es correcto. En particular, mostrar cómo se construye el fixture a partir del flujo.
4. Determinar la complejidad temporal de resolver el problema en base al tamaño de la entrada. Asumir que para resolver el modelo de flujo se emplea el algoritmo de Edmonds y Karp, y dar una cota ajustada. La entrada del algoritmo consiste en los conjuntos E, P, T y D_e para cada $e \in E$; los turnos f_e para cada $e \in E$ y el número natural K.

2.4 Solución B

2.4.1

Para el modelo tenemos que describir una red de flujo, para esto vamos a salir de una fuente s a todos vértices que van a representar los partidos con capacidad 1, de cada partido tenemos que salir de la intersección de las disponibilidades hacia un turno y también del mismo hacia los favoritos, todo esto también con capacidad

[‡‡]Esto no es ncesario, uno también puede iterar en $O(k)$ para conseguir la distancia a los k vértices que representan CU.

1, luego de cada turno tenemos que salir a las canchas con capacidad 1 y de cada cancha al sumidero con capacidad k.

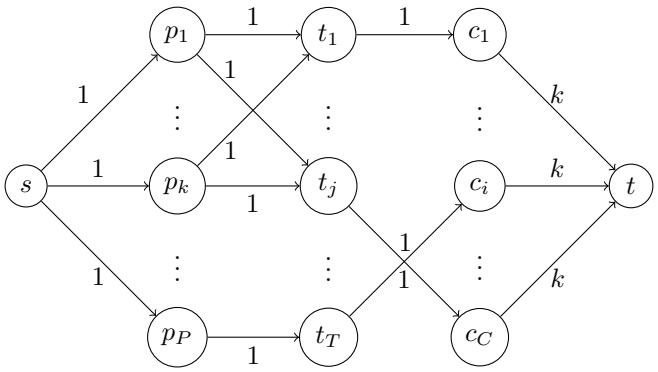


Figure 3: Nuestro modelo solo con capacidades.

2.4.2

1. Las capacidades $s-p$ representan la cantidad máxima de veces que se puede jugar ese partido, un flujo de 1 significa que en el fixture final va a estar ese partido.
2. De la misma manera las capacidades $p-turno$ representan la máxima cantidad de veces que se puede usar ese turno que es 1, si hay un flujo 1 significa que vamos a usar ese turno
3. Mismo que el anterior pero entre turnos y canchas
4. Entre canchas y t tenemos capacidad k porque queríamos evitar que se use de más una cancha, esto nos dice que a lo sumo van a jugar k veces en esa cancha y un flujo f en alguna de esas aristas representa que se juegan f veces en esa cancha.

2.4.3

Queremos probar que el modelo es correcto. Esto lo hacemos haciendo dos cosas

1. $\rightarrow$) Mostramos que cualquier fixture se puede transformar en un flujo válido para nuestro modelo.
2. $\leftarrow$) Mostramos que cualquier flujo válido de nuestro modelo nos da un fixture válido.
3. Concluimos que el flujo máximo nos da la cantidad máxima de partidos jugables en el torneo.

$\Rightarrow$) Veamos que un fixture se puede transformar en un flujo válido. Un fixture es una serie de partidos p_{ij} en distintos turnos (c_i, h_i), si se juega el partido p_{ij} podemos asignar flujo 1 a la arista sp_{ij}, luego como tenemos que conservar flujo tenemos que salir de p_{ij} con flujo 1 esto lo hacemos con $f(p_{ij}(c_i, h_i)) = 1$ y de vuelta al tener que conservar sobre (c_i, h_i) ponemos $f((c_i, h_i)c_i) = 1$, por último para $f(c_i t)$ lo ponemos igual a la suma de todos los flujos que salimos de algún (c_i, h_k)

Además como un fixture válido a lo sumo hace jugar un partido una sola vez y usa un turno una sola vez y a lo sumo usa k veces cada cancha nunca tenemos un fixture que nos genere un flujo inválido (que nos pasemos de las capacidades).

$\Leftarrow$) Supongamos que tenemos una función de flujo válida (es decir que respeta capacidades y conservación), queremos construir un fixture válido, podemos ir del sumidero a la fuente o viceversa.[††] Vayamos desde el sumidero, si al sumidero llega un flujo f significa que alguna arista $c_i t$ tiene flujo positivo, digamos $f_i \leq k$, por conservación de flujo sabemos que le tiene que entrar f_i de los turnos, como cada turno puede usarse una sola vez tiene que haber f_i turnos, de vuelta por cada uno de estos turnos y por conservación de flujo tiene que haber algún partido que mande flujo igual a 1 a cada uno de estos f_i turnos, y finalmente por cada uno de estos partidos, de vuelta por conservación tenemos que sale del sumidero 1 de flujo a cada uno. Estos partidos con sus respectivos turnos representan el fixture que buscábamos.

[††]Hacerlo desde el sumidero podría ser útil para comprenderlo mejor.

2.4.4

Veamos la complejidad, primero veamos cuantos vértices y aristas, sean P los partidos, T los turnos y C las canchas entonces tenemos[**]

$$V = P + T + C = O(P + T)$$

Como cada partido se puede jugar potencialmente en cualquier turno tenemos PT aristas, cada turno a lo sumo se juega en una sola cancha por lo cual hay T aristas de turnos a canchas y como las canchas están acotadas por los turnos tenemos que $C \leq T$ y, por lo tanto, hay a lo sumo T aristas al sumidero, en total tenemos.

$$E = P + PT + T + T = O(PT)$$

Además, el flujo máximo f está acotado por P (basta tomar ese corte)[‡‡].

Si corremos FFEK tenemos las siguientes complejidades

$$O(\min(Ef, E^2V)) = O(\min(P^2T, (PT)^2(P + T + C)) = O(P^2T)$$

[**]Escribo E y V sin las barritas para comodidad.

[‡‡]Aunque hay casos particulares donde hay otros cortes más chicos, por ejemplo si hay una sola cancha con muchos partidos y turnos, el peor caso es cuando tenemos la misma cantidad de partidos, la misma cantidad de turnos y todos los turnos van a canchas diferentes.