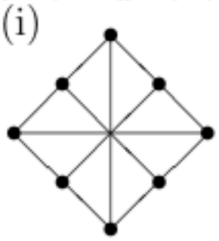
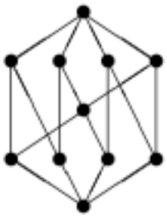
Contents

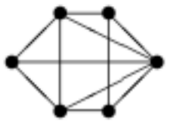
1 Practico 1

1.1 2)

(2) Hallar el número cromático de los siguientes grafos.

(i) 

(ii) 

(iii) 

1. Encontrar algun subgrafo para el cual se sepa su número cromatico
2. Plantear $\chi(H) \leq \chi(G)$ #. Dar un coloreo para el grafo que usa C colores, buscando que $C = \chi(H)$
3. Plantear $\chi(G) \leq C$
4. Pleantear que como $\chi(H) = C \Rightarrow \chi(G) = C$

1.1.1 iii)

Para buscar grafos completos tener en cuenta el grado que deberia tener cada nodo.

Por ejemplo, si buscas un K_4 buscar 4 nodos con grado 3

1.2 3)

Para cada caso (r impar, $r = 2$ y $r > 2$ y r par > 2))

1. Visualizar como se veria el grafo
2. Encontrar algun subgrafo para el cual se sepa su número cromatico
3. Plantear $\chi(H) \leq \chi(G)$
4. Dar un coloreo para el grafo que usa C colores, buscando que $C = \chi(H)$ Este coloreo puede ser definido como una funcion definida por casos que toma argumento un i que es el i-esimo vertice Ver los casos para los cuales la funcion C no puede ser igual. Por ejemplo: $C(i) \neq C(i+1)$
5. Plantear $\chi(G) \leq C$
6. Pleantear que como $\chi(H) = C \Rightarrow \chi(G) = C$

1.3 6)

- Con los ciclos se puede decir que son simetricos por rotacion

1. Visualizar grafo
2. Demostrar que no existe un coloreo propio con 3 colores
    - Tomar un subgrafo y plantear un lema en base a el
    - Demostrar lema
    - Buscar llegar a un absurdo al aplicar el lema en el grafo entero
3. Dar un coloreo propio con 4 colores
    - Ver casos segun cada tipo de vertice
4. Plantear $4 \leq \chi(G) \leq 4 \Rightarrow \chi(G) = 4$

1.4 7)

1. Visualizar grafo con un ejemplo mas chico de G_n
    - Pensar en que significa que $mcd(i, j) = 1$
2. Encontrar algun subgrafo para el cual se sepa su número cromatico
3. Plantear $\chi(H) \leq \chi(G)$
4. Dar un coloreo para el grafo que usa C colores, buscando que $C = \chi(H)$
    - Este coloreo puede ser definido como una funcion definida por casos que toma argumento un i que es el i-esimo vertice
    - Ver los casos para los cuales la funcion C no puede ser igual.
    - Por ejemplo: $C(i) \neq C(i + 1)$
5. Demostrar que el coloreo es propio viendo los distintos casos
6. Plantear $\chi(G) \leq C$
7. Pleantear que como $\chi(H) = C \Rightarrow \chi(G) = C$

1.5 8)

1. Visualizar grafo con n chicos
    - Ver cuales son los coprimos de 6
    - Usar coordenadas para definir los vertices
2. Encontrar algun subgrafo H para el cual se sepa su número cromatico
3. Plantear $\chi(H) \leq \chi(G)$
4. Dar un coloreo para el grafo que usa C colores, buscando que $C = \chi(H)$

- Ver casos en los que dos vertices estan conectados, usando dos vertices (a,b) y (c,d)

5. Demostrar que el coloreo es propio viendo los distintos casos
   - Evaluar C(i,j) en cada caso y probar que $C(a,b) \neq C(c,d)$ para demostrar que se cumple
6. Plantear $\chi(G) \leq C$
7. Pleantear que como $\chi(H) = C \Rightarrow \chi(G) = C$

1.6 10)

1. Dar un coloreo propio con 4 colores
2. Demostrar que no existe un coloreo propio con 3 colores
   - Dar un coloreo general para algun subgrafo
   - Deducir un coloreo general para el resto de vertices
   - Ver si se llega a un absurdo
3. Plantear $4 \leq \chi(G) \leq 4 \Rightarrow \chi(G) = 4$

1.7 12)

1.7.1 Clase

Tomar subgrafo con todos los vertices menos el vertice cuyo lado es delta Como por propiedad p < k, al agregar x se aumenta el NC, los vecinos de x deben tener todos los colores por lo cual: delta >= p como el maximo de aumento del NC es 1: p+1 = k Y hay que llegar a que delta+1 >= k

Ideas: 1) Tomar x con grado delta 2) Para que x use nuevo color los vecinos de x tienen que incluir todos los colores 3) k = p+1

2 Practico 2

2.1 1)

- Pensar en algun flujo donde las capacidades no tengan interseccion

2.2 2)

2.2.1 Notas

Un flujo maximal es cuando se esta mandando todo lo que se puede y ser recibe todo lo que se puede

2.2.2 Howto

1. Partir de un network super basico
2. Ver si sirve de contraejemplo para alguna de las preguntas.
3. Añadir mas cosas al network y volver al paso anterior

2.3 3)

- Prestar mucha atencion a los dominios de las “funciones”

1. Graficar todas las transformaciones que hay que hacer para usar la caja negra
    - P1 -T1-> P2 -> CN -> S2 -T2> S1
2. Ver como adaptar el problema a la caja negra
    - Hacer un ejemplo de un network con un loop y ver de que manera transformarlo para que no afecte.
    - Hacer un ejemplo de un network con un lado paralelo y ver de que manera transformarlo para que no afecte
3. Ver como adaptar la solucion de la caja negrea a la solucion que buscamos
4. Demostrar que el flujo obtenido es flujo del problema original

2.4 7)

- Tener en cuenta teorema de la integralidad
- Es posible aplicar FF para demostrar

1. Hacer un ejemplo sencillo. Arrancar con al menos 4 vertices
2. Ver si se llega a alguna conclusion.
3. Complicar el ejemplo y repetir el paso anterior

- Para llegar a un flujo maximal impar en un flujo par no hay que usar todas las capacidades.
    - Y similar para el caso par -> impar

2.5 8)

- Pensar en que la **capacidad** por vuelo es de 1 agente
  - Por ende, no podemos mandar medio agente
  - Esto que caracteristica le da al flujo?
- Que algoritmo va muy bien para flujos maximales enteros?

3 FF

- Armar network residual con los flujos que se pueden deshacer y los flujos que aun se pueden tomar

4 Practico 3

4.1 2)

- Tener en cuenta el orden en el que se agregan los vertices y en el que se exploran
- Hay que buscar armar un network con la forma de una S que tiene un agujero en el medio

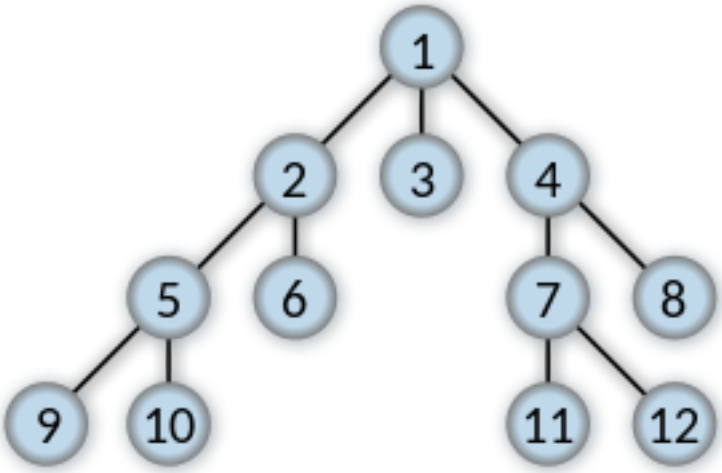
1. Partir de un network super basico
2. Ver si sirve
3. Añadir mas cosas al network y volver al paso anterior

5 Algoritmos

5.1 BFS

5.1.1 Que hace?

BFS arma un arbol de la forma:



5.1.2 Que tener en cuenta al usarlo?

- Solo añadir nodos que esten al mismo nivel
- No volver añadir nodos ya recorridos

5.2 EK

5.2.1 Idea

EK puede ser pensado como un algoritmo para encontrar el menor camino aumentante de s hasta t

5.2.2 Notacion

Para resolver EK usamos la siguiente notacion:

```
sB C A  D t
 s B C- A D
 9 9 7  5 5
```

1) La primer fila son los vertices que vamos agregando
2) La segunda fila es quien los agrego, si es un lado backward lleva exponente "-"
3) La tercer fila es el flujo que puede ser mandado

5.2.3 Armar camino aumentante

Primer nodo descubierto: s

Por cada nodo descubierto:

1. Añadir vecinos del nodo al camino aumentante
    - Añadir si:
        - Es vecino forward
        - Es vecino backward
        - El lado aun tiene capacidad disponible
        - El nodo aun no fue descubierto
    - Se escribe: Nombre, padre y capacidad real
2. Tachar nodo
3. Avanzar con el siguiente nodo en la lista de nodos descubiertos y volver a 1

5.2.3.1 A tener en cuenta

- No olvidar agregar nodos backward!

5.2.4 Pasos

1) Armar primer camino aumentante (BFS) con notacion adecuada
2) Reconstruir camino empezando desde t (leyendo segunda fila)
3) Actualizar tabla con las capacidades nuevas
    - Solo cambia si pertenece al camino aumentante formado
4) Armar siguiente camino aumentante

5.2.5 Verficar

Para verificar se puede calcular si el valor del flujo es igual a la capacidad del corte minimal

Una vez llegado al punto que no se pueda armar otro camino:

1. Definir este camino incompleto como un corte S
2. Calcular valor de f ($v(f)$)
    - Ver cuanto flujo sale de s y sumarlo
3. Calcular capacidad de S

- Sumar capacidades de los lados que salen del corte
- Osea, todos los lados xy donde x pertenezca a S, pero y no

4. Chequear que $v(f) = C(S)$
5. Concluir que f es flujo maximal y S corte minimal

5.2.6 Cosas a tener en cuenta

- El BFS de cada camino aumentante es independiente del anterior
  - Tampoco toma caminos saturados
  - Si está saturado es como que se elimina ese lado
- En la notacion de camino aumentante solo poner la cantidad que de verdad puede ser mandada, no la total
- Los lados backward tambien agregan vecinos!

5.3 Dinic

5.3.1 Armar network auxiliar

- Solo se añaden si estan a mayor distancia de s que el padre
- Se pueden añadir lados a nodos ya descubiertos
- Apenas se llega a t se para de añadir nodos
- Tambien se añaden lados backward al network auxiliar

5.3.2 Pasos

1. Armar network auxiliar por niveles (BFS)

- A partir de las segunda iteracion se toman solo los que tienen capacidad mayor a 0

1. Hacer DFS desde s hasta t hasta que se llegue a un flujo bloqueante

- Si se llega a un camino sin salida hacer backtrack y seguir hasta llegar a t

1. Repetir pasos anteriores

- Flujo bloqueante: cuando no se pueden encontrar mas caminos desde s hasta t porque todos los vertices han sido saturados
- Cuando hay un flujo backward se toma la capacidad que se está usando al calcular el valor del camino

5.4 Wave

1. Hacer lista de vertices ordenados por niveles y alfabeticamente
    - Se va armando una tablita (2D) con lo que manda cada vertice

Ola forward:

1. Cada vertice manda todo lo que puede a sus vecinos
2. Marcar los vertices que quedan bloqueados (no pueden mandar el flujo que tienen)
3. Una vez se termina de mandar todo se hace una linea al final de la tablita y se hace una flechita hacia la derecha en algun lado de la parte de arriba, como para denotar que es forward

Ola backward:

1. Arrancar desde los vertices bloqueados
2. Devuelve todo lo que puede
    - Para devolver usar orden antialfabetico (solo se puede devolver a alguien que manda algo)

Se hace ola forward de nuevo

1. Se manda hacia adelante en los vertices desbalanceados, pero no se manda a los bloqueados

Se hace de nuevo lado backward

5.4.1 Pasos

1. Armar tablita de nodos
2. s manda a sus hijos todo el flujo que tiene
3. Los hijos de s mandan todo lo que pueden a sus hijos
    - Sin sobrepasar lo que les mandó s
    - Se actualiza en la tabla el flujo disponible en el padre
    - Si no se manda todo lo que se pudo mandar se encierra en un cuadrado el numero
    - Una vez se bloquean no se desbloquean más

Ola backward:

1. Toma solo los nodos bloqueados
2. Cada nodo bloqueado devuelve al que le mandó

- Devolver en orden antialfabetico

Ola forward:

- No se manda a los bloqueados

6 Matchings

6.1 Como interpretar matrices de adjacencia?

6.1.1 Que representan?

Un network, cuyos vertices pertenecen a uno de dos conjuntos, llamemoslos X e Y.

6.1.1.1 A quien estan conectados los vertices de X?

Estan conectados a s

6.1.1.2 A quien estan conectados los vertices de Y?

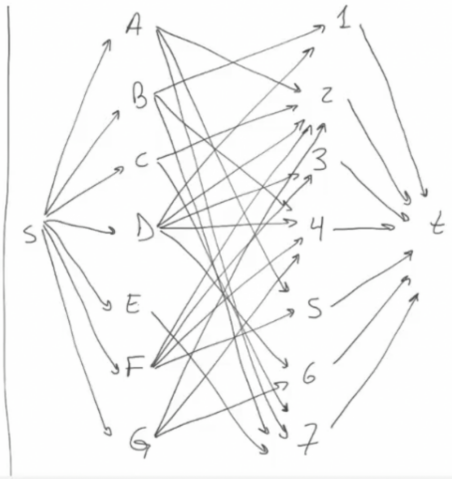
Estan conectados a t

6.1.1.3 Que capacidad tienen los lados?

Siempre tienen capacidad 1

f)

	1	2	3	4	5	6	7
A	0	1	0	0	1	0	1
B	1	0	0	1	0	0	1
C	0	1	0	0	0	0	1
D	1	1	1	1	0	1	0
E	0	0	0	0	0	0	1
F	0	1	1	1	1	0	0
G	0	1	0	1	0	1	0



6.1.1.4 Ejemplo grafico

6.1.2 Como es la estructura de la matriz?

Las filas son nodos que perteneces a X y las columnas nodos que pertenecen a Y.

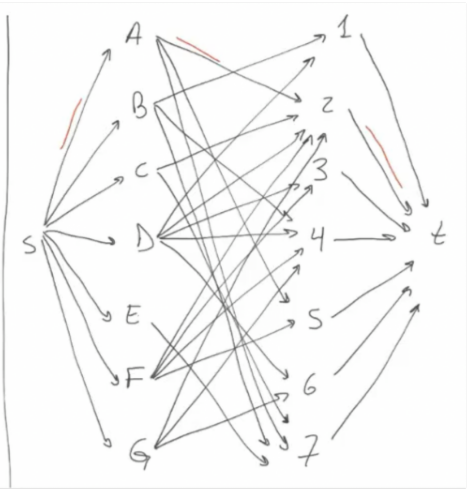
6.2 Como se arman los caminos usando solo la matriz?

Se buscan los 1s en cada fila, ya que sabemos que los vertices de las filas estan conectados a s y los de las columnas estan conectados a t

6.2.1 Ejemplo

f)

	1	2	3	4	5	6	7
A	0	1	0	0	1	0	1
B	1	0	0	1	0	0	1
C	0	1	0	0	0	0	1
D	1	1	1	1	0	1	0
E	0	0	0	0	0	0	1
F	0	1	1	1	1	0	0
G	0	1	0	1	0	1	0



Para marcar el camino `sA2t` basta con encerrar en un circulo el 1 de la fila A en la columna 2

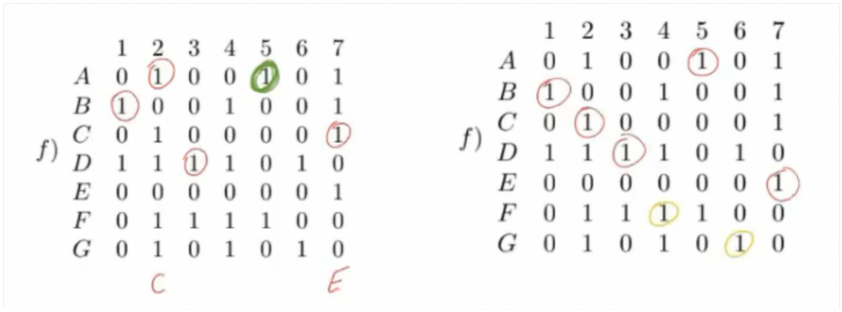
6.2.2 Algoritmo

Por cada fila de la matriz: 1. Marcar el primer 1 de la fila que no posea otro 1 marcado en la misma columna 2. Si todas los 1s disponibles estan en una columna ocupada: 1. Marcar la columna bloqueada con el nombre del vertice de la fila 2. Ver si el nodo que bloqueo la columna puede mandar flujo por otro lado 3. Repetir (1) y (2) hasta que se encuentre una forma de mandar el flujo 4. Una vez hallada la forma de mandar flujo, hacer una copia de la matriz que use ese camino para que quede más prolijo

3. Seguir con la siguiente fila

6.2.2.1 Tips

- Al encontrarse con una columna bloqueada pensar el proceso como un camino backward de EK o Dinic



$f)$		1	2	3	4	5	6	7
	A	0	1	0	0	1	0	1
	B	1	0	0	1	0	0	1
	C	0	1	0	0	0	0	1
	D	1	1	1	1	0	1	0
	E	0	0	0	0	0	0	1
	F	0	1	1	1	1	0	0
	G	0	1	0	1	0	1	0

C E

$f)$		1	2	3	4	5	6	7
	A	0	1	0	0	1	0	1
	B	1	0	0	1	0	0	1
	C	0	1	0	0	0	0	1
	D	1	1	1	1	0	1	0
	E	0	0	0	0	0	0	1
	F	0	1	1	1	1	0	0
	G	0	1	0	1	0	1	0

6.2.2.2 Ejemplo

7 Codigos correccion de errores

7.1 Notacion

7.1.1 Peso de hamming

$|v| \rightarrow$ peso de hamming de v

7.1.2 Cardinalidad

#

7.1.3 Distancia de hamming

7.1.4 Dimension codigo lineal

k

7.1.5 Codigo lineal

Dimension k, longitud n y $\delta(C) = \delta\ (n, k, \delta)$

7.2 Matriz generadora G

- Va a contener una matriz identidad
  - Despues a la derecha puede tener cualquier cosa
- Cantidad de columnas es la longitud
- Cantidad de filas es la dimension
- Tiene que ser LI
- En Z_2 a y b son LD $\Leftrightarrow a + b = (0, 0)$
- No puede haber filas 0?
- Sea u un vector, uG es palabra del codigo

7.2.1 Forma

$$\left[I_{(k\times k)}|A\right]$$

7.3 Matriz chequadora H

- Nucleo H es el conjunto de vectores u tal que $Hu^t = 0$
- Llega una palabra, la mutliplico por H y si no da cero hubo un error
- Cantidad de columnas de H = longitud de u

7.3.1 Forma

$$\left[A^t | I_{(n-k)}\right]$$