TAREA SEMANAL 2 – 3

GREEDY

Integrantes:
Andrés Monetta
Alejandro Clara
Sebastián Daloia

Contenido

3. Preguntas

1. ¿En qué consiste la técnica Greedy?

Dado un espacio de soluciones D, teniéndose un conjunto de candidatos a solución, la técnica Greedy se aplica para elegir un subconjunto de aquellas soluciones que cumplan con ciertas restricciones o criterios de optimalidad.

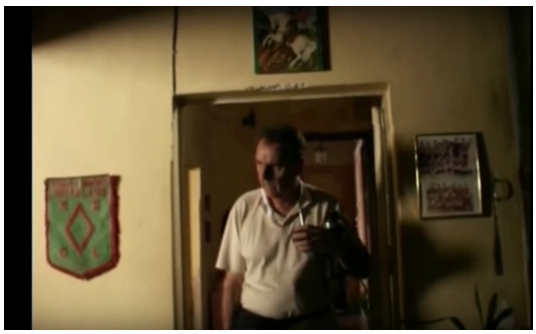
La técnica Greedy analiza un conjunto de candidatos, en cada etapa, y toma una decisión óptima. La desición es tomada localmente, sin considerar etapas anteriores o futuras.

2. Dé un ejemplo de problema y un algoritmo basado en la técnica Greedy aplicada a él que no siempre devuelva la solución buscada.

En la Ruta 1, km 17, entrenan las formativas del club Rampla Juniors.

Al momento de comenzar las prácticas los distintos entrenadores le piden camisetas para practicar al equipier (y kinesiologo-masajista) del club.
El equipier, Zelmar(ver figura abajo), es dueño de organizar las camisetas en el depósito a gusto. Zelmar ha elegido organizarlas de la siguiente manera.

Camisetas
Modelo 2015 1 camiseta en 1 bolsa
Modelo 2013 3 camiseta en 1 bolsa
Modelo 2012 4 camisetas en1 bolsa
Modelo 2009 7 camisetas en1 bolsa
Modelo 2008 8 camisetas en1 bolsa



ZELMAR

En cualquier momento un entrenador va y le pide camisetas para los **m** gurises de su plantel.
Se asume que siempre hay disponibilidad de bolsas de camisetas, cada vez que se hace un pedido.

Se desea generar una solución que le haga a Zelmar desembolzar la mínima cantidad de camisetas por pedido de entrenador, logrando que cada gurí tenga su camiseta para el entrenamiento.

Se plantea el siguiente algoritmo:
mientras jugadores > 0 hacer
sea b la bolsa del modelo de camiseta más vieja que cumple camisetas(b) <= jugadores entonces
k:= jugadores div camisetas(b)
jugadores=jugadores – kjugadores

Ejemplo 1

jugadores = 23
Zelmar(la solución) devuelve
2 bolsas con camisetas modelo 2008
1 bolsas con camisetas modelo 2009

La solución es la buscada.

Ejemplo 2
jugadores = 30
Zelmar(la solución) devuelve

3 bolsas con camisetas modelo 2008
1 bolsa con camisetas modelo 2012
2 bolsas con camisetas modelo 2015

Un total de 6 bolsas con camisetas

La cual es solución al problema pero no es la más óptima ya que Zelmar podría haber devuelto el siguiente resultado
3 bolsas con camisetas modelo 2008
2 bolsas con camisetas modelo 2013

Un total de 5 bolsas, solución óptima para este caso.

3. ¿En qué propiedad basan su correctitud los algoritmos Prim y Kruskal?

Propiedad **MST, Minimum Spanning Tree**
Sea G=(V, E) un grafo conexo con una función de costo definido sobre las aristas.
Sea U un subconjunto de los vértices V, si (u,v) es una arista de costo mímino tal que $u \in U \; v \in V - U$ entonces existe un árbol de cubrimiento de costo mínimo que incluye a (u,v) como una de sus aristas.

4. ¿Cómo aplica el algoritmo de Prim esa propiedad?

Construcción:
Sea G=(V, E) grafo conexo, con una función de costo asociada a sus aristas
A partir de un vértice arbitrario v, que inicialmente es el único elemento del subconjunto U, en cada paso se tiene un único árbol que va creciendo al ir agregándose aristas, se agegra desde luego el nuevo vértice a U, se termina el proceso cuando U=V.
La forma de asignar una nueva a arista es

$$.<i,j>.\cdot i \in U, j \in V - u \text{ y } c(<i,j>) = min\{c(<k,l>).\cdot <k,l> ¿ E, k \in U, l \in U - V\} \quad \textbf{(1)}$$

como en cada paso el árbol agrega una arista de costo mínimo entre U y U-V y no agrega aristas entre elementos de U, entonces se asegura la correctitud de algoritmo por la proposición MST.

5. ¿Cómo aplica el algoritmo Kruskal esta propiedad?

Construcción a partir no de un solo árbol sino a través de un bosque.
Inicialmente el bosque está conformado por todos los vértices de V y luego se van generando uniones entre árboles que tengan aristas de costo mínimo.

Para asegurar la correctitud, Kruskal exige que las aristas sean consideradas en orden creciente de costos.

La arista considerada si es entre vértices de diferentes árboles entonces se agrega pues cumple con la proposición MST, es una arista de coste mínimo entre el subconjuntos de vértices de cualquiera de los dos árboles y el resto de los vértices de V.
Si la arista considerada es entre vértices de un mismo árbol del bosque entonces no se agrega, pues se generaría un ciclo.

6. Describa semejanzas y diferencias entre los algoritmos de Prim y Dijkstra.

Sea G=(V,E) Grafo conexo

Para cada paso los algoritmos mantienen un conjunto S de vértices, y en cada paso se agrega un nuevo vértice a S.
La cantidad de pasos para estos métodos es #V -1

Ambos comienzan desde un nodo arbitrario u origen, el cual es el primer elemento de S.

Difieren en la forma de agregar elementos al conjunto.
Prim agrega vértices como se indica en (1) evaluando las aristas de los nodos de S-V adyacentes a los vértices de S.
Para Dijkstra, los valores ponderados para la desición del vértice a elegir en cualquier paso varian. Ya que los vértices v que no pertenecen a S si existe un camino desde el origen pasando por S y llegando a v de menor costo que el asignado a ese v en ese paso, entonces se actualiza el valor ponderado por ese nuevo valor de menor costo.
Teniendo en cuenta esto los vértices agregados en cada paso pueden diferir en uno u otro algoritmo.

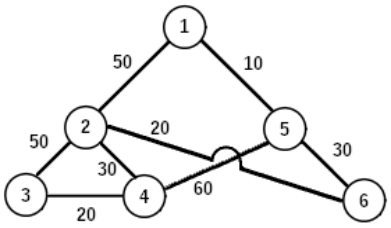
Estructuralmente se puede observar que Prim trabaja solamente en árboles no dirigidos, ya que se basa en MST, y este último está definido solo para no dirigidos.
En cambio Dijkstra permite trabajar tanto en árboles dirigidos, como no dirigidos, dado que se puede construir un árbol de caminos de costo mínimo para ambos.

El algoritmo de Prim genera un árbol MST, el cual conecta a todos los nodos siendo su costo total el mínimo posible. Un ejemplo de aplicación de este algoritmo podría ser, si se quisieran conectar distintos puntos para proveerles electricidad mediante cables energéticos. Prim va a asegurar que el costo total en cables sea mínimo. Sin embargo, no se garantiza que el costo del camino entre dos puntos cualesquiera, sea el menor posible.
Dijkstra genera un árbol con los caminos menos costosos desde el nodo raíz, a todos los demás. Una buena aplicación para este algoritmo sería por ejemplo, si se quisieran construir vías de tren y se pretendiera que los caminos fueran los más rápidos posibles (tomando el costo como el tiempo) desde una ciudad “raíz”. Una desventaja de este algoritmo es que puede generar un costo total mayor que el de un MST.

Otra diferencia a tomar en cuenta es que Dijkstra no permite trabajar con valores de costo negativos, ya que así, no garantiza encontrar el camino de costo mínimo.

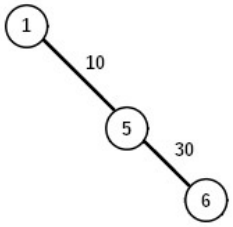
Ejemplo

Se toma el siguiente grafo

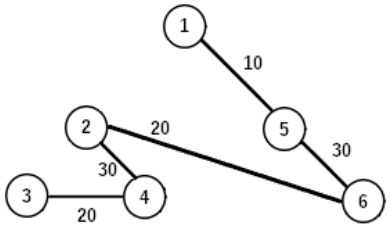


Prim

El algoritmo comienza desde el vértice 1. Las aristas que cumplen que uno de sus vértices está en el árbol y el otro no son: <1,2> y <1,5> cuyos costos son 50 y 10 respectivamente. Se agrega al árbol la arista de mínimo costo de las anteriormente mencionadas, o sea la arista <1,5> y se repite el proceso. Las aristas que cumplen que uno de sus vértices está en el árbol y el otro no son: <1,2>, <5,4> y <5,6> cuyos costos son 100, 60 y 30 respectivamente. Por lo tanto en este paso se elige la arista <5,6> y el árbol resultante es el siguiente:



Luego se sigue repitiendo el proceso y se van eligiendo sucesivamente las aristas: <6,2>, <2,4> y <4,3>. Una vez finalizado el algoritmo el ACCM queda de la siguiente manera:

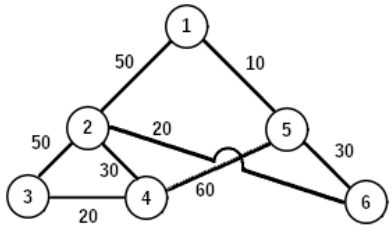


Observaciones:

> Costo total de 110

> Para llegar a 3 desde 1, se toma el camino 1-5-6-2-4-3. El camino más corto en el grafo original es 1-2-3. En este ejemplo se forma un árbol degenerado, lo cual ilustra de buena manera la desventaja de Prim respecto de Dijkstra en cuanto a caminos menos costosos.

Dijkstra



C:

	1	2	3	4	5	6
1	∞	50	∞	∞	10	∞
2	50	∞	50	30	∞	20
3	∞	50	∞	20	∞	∞
4	∞	30	20	∞	60	∞
5	10	∞	∞	60	∞	30
6	∞	20	∞	∞	30	∞

S={1}

La primera iteración inicializa el arreglo D

D:

∞	50	∞	∞	10	∞

El camino de menor costo a un vértice de V-S es al vértice 5. Se agrega el vértice 5 a S y se actualiza D:

S = {1,5}

D:

∞	50	∞	70	10	40

El camino de menor costo a un vértice de V-S es al vértice 6. Se agrega el vértice 6 a S y se actualiza D:

S = {1,5,6}

D:

∞	50	∞	70	10	40

El camino de menor costo a un vértice de V-S es al vértice 2. Se agrega el vértice 2 a S y se actualiza D:

S = {1,5,6,2}

D:

∞	50	100	70	10	40

El camino de menor costo a un vértice de V-S es al vértice 4. Se agrega el vértice 4 a S y se actualiza D:

S = {1,5,6,2,4}

D:

∞	50	90	70	10	40

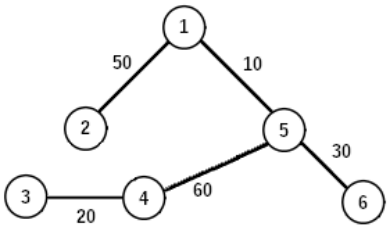
El camino de menor costo a un vértice de V-S es al vértice 3. Se agrega el vértice 3 a S y se actualiza D:

S = {1,5,6,2,4,3}

D:

∞	50	90	70	10	40

Se genera el siguiente árbol final



> Costo total: 170
> Los caminos tomados son los menos costosos tomando en cuenta el grafo original
> El grafo original podría haber sido dirigido. Si todas las flechas fueran “hacia abajo”, la solución sería la misma.

4. Problema

a) ¿Es realmente Greedy?

Para cada etapa se toma la decisión más óptima localmente, que es comparar la charla ch_k con las k-1 charlas que le preceden en el arreglo ordenado y ver si es compatible con aquellas que ya han sido incorporados a la solucion, entonces de ser así se agrega a la solución.
Como se mencionó la decisión se toma sin tener en cuenta etapas anteriores o futuras.
Entonces es un algoritmo Greedy.

b) Orden en función de n

Considerando el **for** y las **i-1** iteraciones por cada pasada se tiene la siguiente expresión:

$$\sum_{i=2}^{i=n}(i-1)$$

Sea $h=i-1$

Entonces $\sum_{h=1}^{h=n-1}(h)=\frac{(n-1)n}{2}\equiv O(n^2)$

La propiedad Fecha de Fin puede mejorar el orden
Algoritmo:

```
charla[1...n] ordenada por Fecha de Fin

S={1}
actual=1

for i=2 to n do
{
        if (compatibles(actual, i))
        {
                S=S∪{i}
            actual = i
        }
}
retornar S
```

Correctitud:
charla está ordenada según Fecha de Fin en forma creciente.
actual es el último elemento solución agregado al algoritmo, es decir no hay otra charla siguiente que tenga hora de finalización menor a actual.
Entonces si i no es compatible con cualquier otro elemento de S que actual entonces no es

compatible con actual por lo expresado en el párrafo anterior.
Luego solo es necesario compara la compatibilidad con el último elemento agregado a la solución S.

El orden según n para este caso es $\sum_{i=2}^{n} 1 = n-2+1 = n-1 \equiv O(n)$

c) Correctitud de algoritmo según atributo

Sea el algoritmo propuesto como solución.
Sea el arreglo **charlas[1..n]** ordenado según el atributo considerado

Sea atributo *Duración*

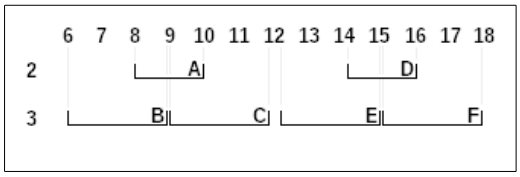
Juego de datos

Charla A inicio: 8 fin: 10 duración: 2	Charla B inicio: 6 fin: 9 duración: 3	Charla C inicio: 9 fin: 12 duración: 3
Charla D inicio: 14 fin: 16 duración: 2	Charla E inicio: 12 fin: 15 duración: 3	Charla F inicio: 15 fin: 18 duración: 3

Si se toma como atributo la duración, la charla A será la primera en ser asignada a la solución. Luego se asignará la charla D, y hasta aquí no habrán problemas de compatibilidad.

Después se procederá a comprobar la compatibilidad de la charla B con la charla A, y se verá que no son compatibles. Lo mismo sucederá con la charla C. Este proceso se repite al comprobar la compatibilidad de la charla E y la F con la D. Se generaría la solución t_1 = <A,D>.Sin embargo las charlas B, C, E y F son compatibles entre sí, por lo tanto t_1 no es solución óptima. La solución óptima (que garantiza mayor cantidad de charlas) es t = <B,C,E,F>.

Diagrama:



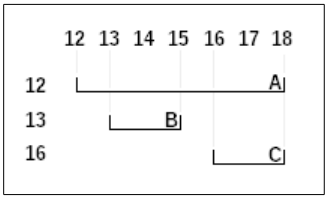
Sea atributo *Fecha de inicio*

Juego de datos

Charla A inicio: 12 fin: 18	Charla B inicio: 13 fin: 15	Charla C inicio: 16 fin: 18

Si se toma como atributo la fecha inicial, la charla A será la primera en ser asignada a la solución. Luego se procederá a comprobar la compatibilidad de la charla B con la charla A, y se verá que no son compatibles. Lo mismo sucederá con la charla C. Esto generaría la solución t_1 = <A>.
Sin embargo tanto la charla B como la charla C son compatibles entre sí, por lo tanto t_1 no es solución. La solución óptima (que garantiza mayor cantidad de charlas) es t = <B,C>.

Diagrama:



Sea atributo *Fecha de fin*

Demostración por inducción de que este algoritmo puede resolver de forma óptima el problema *max_charlas*
Sean

$t=<t_1, t_2, \ldots, t_p>$ tupla solución

$s=<s_1, s_2, \ldots, s_p>$ tupla genérica en la cual cada par de elementos son compatibles entre sí.

Se asume que la tupla genérica s está ordenada según el mismo atributo t.

Observación: como t es solución ordenada según el atributo Fecha de Fin la primera solución tendrá la fecha de fin más temprana y la última solución de la tupla t tendrá la fecha de fin más tardía.

El proceso de inducción se realizará en k

Sea k=1
Sea $t=<t_1>$ solución del algoritmo

Sea $s=<s_1, .., s_r>$ solución tal que s está ordenada según el mismo atributo t

Sí 1<r entonces charlas[t_1].Fecha de Fin $.\leq.$ charlas[s_1].Fecha de fin
Pues por la observación el algoritmo agrega desde la primera charla que termine más temprano en forma creciente.

Luego que compatibles(s_1, s_{1+1}) sea verdadero implica que
charlas[s_{1+1}].Fecha de Inicio $.\geq.$ charlas[s_1].Fecha de Fin
Entonces charlas[s_{1+1}] es compatible con las soluciones de t.

Lo cual es absurdo pues sino debería se incluida por el algoritmo.
Luego $1 \geq r$
Entonces t es solución óptima.

Supongamos que se cumple $t=<t_1, t_2, \ldots, t_{p-1}>$ es solución

Sea k=p

$t=<t_1, t_2, \ldots, t_p>$ tupla solución del algoritmo

Sea $s=<s_1, .., s_r>$ solución tal que s está ordenada según el mismo atributo t

Sí p<r entonces charlas[t_p].Fecha de Fin $.\leq.$ charlas[s_p].Fecha de fin
Pues por la observación el algoritmo agrega desde la primera charla que termine más temprano en forma creciente.

Luego que compatibles(s_p, s_{p+1}) sea verdadero implica que
charlas[s_{p+1}].Fecha de Inicio $.\geq.$ charlas[s_p].Fecha de Fin
Entonces charlas[s_{p+1}] es compatible con las soluciones de t.

Lo cual es absurdo pues sino debería se incluida por el algoritmo.
Luego $t \geq r$
Entonces t es solución óptima.

d) Contraejemplo de atributo superposición de charlas

Juego de datos

Charla A
inicio: 8
fin: 9
duración: 1
superposición: 1

Charla B
inicio: 8
fin: 11
duración: 3
superposición: 2

Charla C
inicio: 10
fin: 12
duración:2
superposición: 2

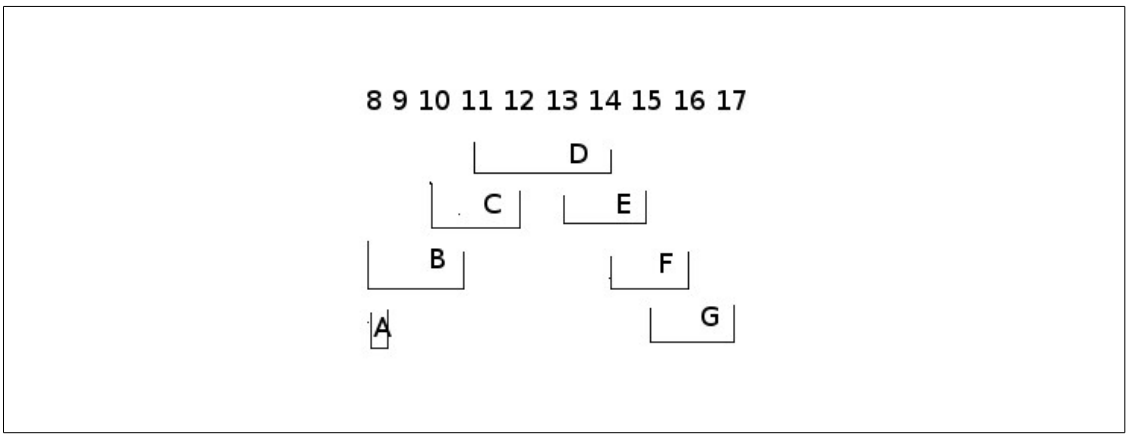
Charla D
inicio: 11
fin: 14
duración: 3
superposición: 2

Charla E
inicio: 13
fin: 15
duración: 2
superposición: 2

Charla F
inicio: 14
fin: 16
duración: 2
superposición: 2

Charla G
inicio: 15
fin: 17
duración: 2
superposición: 1

Esquema de superposiciones:



Para este caso hay varias charlas con igual cantidad de superopiciones:
Observación: En el caso de que haya varias salas con igual cantidad de superposiciones, no se sigue ningún criterio para acomodar en el arreglo el juego de datos.

Si el arreglo está ordenado de la siguiente manera: A-G-D-C-E-F-B
El algoritmo devolvería como tupla solución: <A,G,D>
Pero existe una tupla compatible con más elementos: <A, G, C, E>