Universidad de Concepción

Ingeniería Civil Informática

Análisis de Algoritmos

Tarea II : Análisis de Algoritmos

Integrantes:

Javier Arriagada.
Ignacio Sanhueza.

Docente:

Cecilia Hernández

Concepción - Julio, 2020

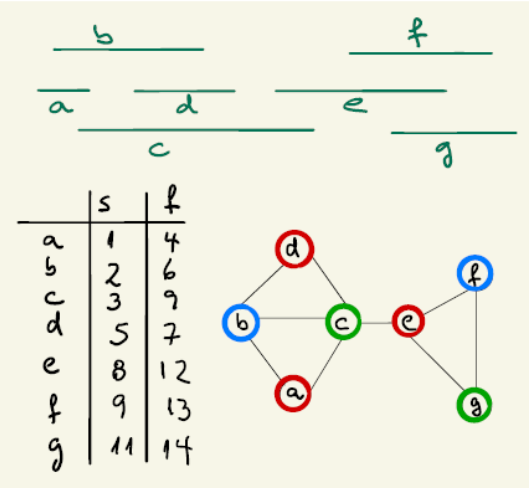
Índice

1. Introducción.

Este proyecto consiste en aplicar la técnica de diseño greedy para resolver el problema de colorear los vértices de un grafo de intervalos usando el mínimo número de colores. Este problema es análogo al asignar un conjunto de actividades al mínimo número de personas de manera que todas las actividades se realicen.
Sea un grafo G(V,E) y C : V $\rightarrow$ {1..K} una función de coloración de vértices tal que para toda arista (u, v) $\in$ E, se tiene que C(u) $\neq$ C(v). El objetivo es minimizar K, cuyo valor representa el máximo número de colores.
El problema general de colorear los vértices de un grafo es un problema NP-Completo, pero existen algunos tipos de grafos para los cuales se tienen algoritmos polinomiales. En particular este proyecto se focaliza en los grafos de intervalos, el cual se puede colorear con un algoritmo greedy.

Grafo de intervalos

Dado un conjunto de intervalos, se define un grafo donde los vértices corresponden a los intervalos y existe una arista entre dos vértices cuando existe una sobreposición entre dos intervalos. Podemos pensar que cada intervalo representa una actividad con tiempo de inicio y término.
Dado un conjunto de n actividades A = { a_1, a_2, ...a_i, ..., a_n}, donde cada a_i = (s_i, f_i). El grafo de intervalos correspondiente se define como G = (V,E) donde V = A y (a_i, a_j) $\in$ E, si $a_i \cap a_j \neq 0$.
A continuación se presenta un ejemplo para el set de actividades A = {a, b, c, d, e, f, g} con el correspondiente grafo de intervalos coloreado.



2. Requerimientos del proyecto.

1. Muestre que usando el algoritmo greedy para resolver el problema de selección de actividades repetidamente no proporciona una solución óptima al problema de asignar todas las actividades al mínimo número de personas. Es decir, aplicar el algoritmo de selección de actividades maximizando el número de actividades para una persona, luego repetir el algoritmo para la siguiente persona y asi sucesivamente.
   Solución
   Se ordenan las actividades en orden creciente de su tiempo de finalización

   Primero definimos la estrategia principal, esta es elegir la actividad que termina más pronto

   $f(j_1) \leq f(j_2) \leq f(j_3) \cdots \leq f(j_n)$

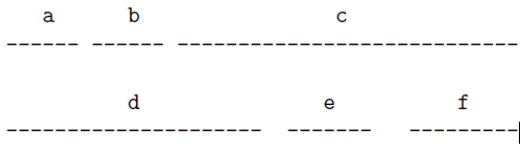
Algoritmo 1 Greedy_maximo_actividades(A, s, f)

Entrada: A: lista actividades ordenada, s: lista t inicial, f: lista t final
Salida: Aproximacion solucion optima

```
1: A = {1}
2: j = 1
3: para i = 2 hasta n hacer
4:    si s[i] ≥ f[j] entonces
5:       A = A + {i}
6:       j = i
7:    fin si
8: fin para
9: devolver A
```

Contraejemplo

	s	f
a	0	6
b	7	12
c	13	34
d	0	21
e	24	28
f	29	34



Es claro ver que estas actividades necesitan sólo 2 personas, pero greedy necesita a 3, ya que d y c, no pueden ser ejecutados por la misma persona.

Greedy maximizando numero de actividades

Persona 1: a b e
Persona 2: d f
Persona 3: c

Optimo

Persona 1 : a,b,c
Persona 2: d,e,f

2. Proporcione un algoritmo greedy que resuelva el problema de asignación de todas las actividades usando el mínimo de personas. Para ello describa el algoritmo en palabras y luego proporcione un pseudocódigo y su complejidad asintótica. Notar que acá se pide pseudocódigo, y no el código fuente del programa que implemente.

   Estrategia: Primero tenemos un vector de actividades ordenado por tiempo de termino y una cola vacía.

   Iteramos sobre dicho vector, y dentro de la iteración se hacen las siguientes comparaciones:

Vamos revisando si en la iteración actual i, dicho vector tiene asociado un color, si no lo tiene, se le asigna uno y el número de colores ocupados aumenta en uno.

Luego se compara el vector actual i, con el siguiente i+1, si el tiempo de término de la actividad i es menor o igual al tiempo de inicio de la actividad i+1, i+1 toma el mismo color de i, o dicho de otra forma, una misma persona puede hacer las 2 actividades

Si ninguna de las comparaciones anteriores se se ejecutó entonces significa que hora de inicio de actividad i+1 es menor a la hora de inicio de la actividad i y se añade la actividad i a la cola. Dentro de este condicional, se ve si la cola tiene un tamaño mayor a cero, si es el caso, se compara el valor de hora de termino que tiene asociado una actividad al frente de la cola (cola.front().horaTermino) y a comparar con la hora de inicio de i+1, si i+1 es mayor o igual a dicho elemento de la cola, entonces i+1 toma el mismo color que tiene la actividad que se encuentra en la cola, y se procede a remover al actividad de la cola. En caso que la cola este vacía o no se cumpla la comparación anterior, se le asigna un nuevo color a i+1 y el número de colores (o personas) aumenta en uno.

Algoritmo 2 repartir_actividad(A[], n)

Entrada: A: vector actividades ordenado, n: cantidad actividades vector
Salida: Aproximación solución óptima

```
queue Cola;
num_personas = 0
para i = 0 hasta n hacer
  si A[i].conColor==false
    Asignar Color a A[i]
    num_personas++;
  si A[i].fin ≥ A[i+1].inicio entonces
    Asignar color de i a i+1
  si no
    si Cola.size()>0 y cola.front().fin≤A[i+1].inicio
      Asignar color de la cola a i+1
      Cola.pop();
    fin si
    Cola.push(i);
    si Cola.size()==0 ó cola.front().fin>A[i+1].inicio
      Asignar Nuevo Color a i+1
      num_personas++
  fin si
fin para
devolver num_personas
```

Ordenar tiene una complejidad de $O(n^2)$(bubble sort), y recorrer el arreglo de actividades tiene un tiempo de $O(n)$, por lo tanto nuestro algoritmo tiene un complejidad asintótica de $O(n^2)$

3. Demuestre la correctitud de su algoritmo. En este caso, se usa un argumento que no incluye propiedad greedy y subestructura óptima, sino mas bien un argumento que muestra una cota mínima o usando contradicción.

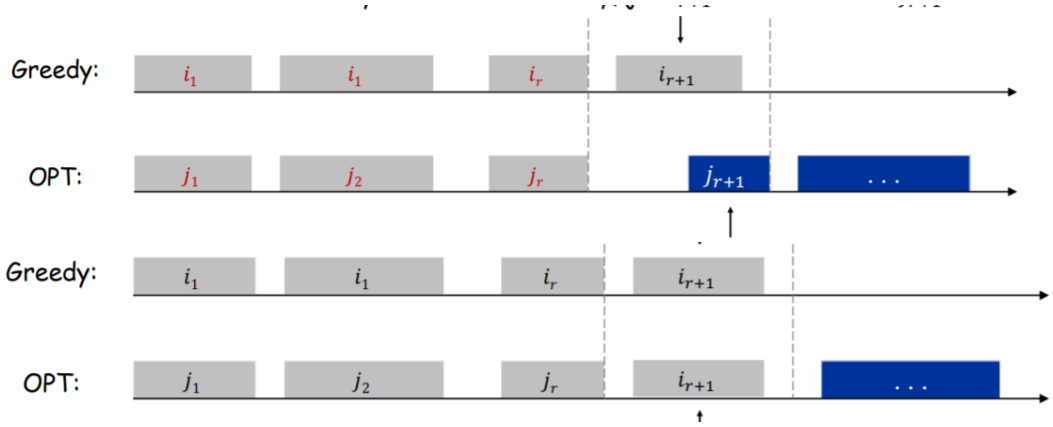
   Se quiere demostrar que el algoritmo greedy es óptimo

   Prueba

   - Asumir que greedy es diferente a OPT
   - Sea i_1 , i_2 , ... i_k actividades seleccionadas por greedy

- Sea j_1 , j_2 , . . . j_k actividades solución óptima. Encontrar el valor más grande posible para r tal que $i_1 = j_1$, $i_2 = j_2$, ..., $i_r = j_r$

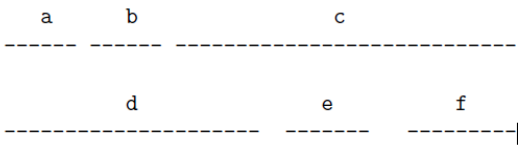
Por definición de greedy, actividad i_{r+1}, termina antes que j_{r+1}



En OPT, reemplazamos la actividad j_{r+1} con i_{r+1}, manteniendo lo que queda del OPT igual. Al tener los mismos números de actividades sigue siendo óptimo Esto crea una nueva solución óptima que comparte sus primeros r+1 actividades con greedy.
Hacer este procedimiento hasta que OPT sea el mismo que greedy.
Ya que el costo se mantiene igual, la solución final creada la cual es greedy es ÓPTIMA

4. Proporcione un ejemplo con la ejecución de su algoritmo donde muestre en cada paso las actividades que han realizado y están actualmente realizando cada una de las personas. Para mayor claridad, construya una tabla con dicha información.

	s	f
a	0	6
b	7	12
c	13	34
d	0	21
e	24	28
f	29	34



Primero se ordena la lista de actividades según su tiempo de término(f)

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

iteracion 1
z=0 y i=1,tamañoPila1=6, tamañoPila2=0 Primero vemos si actividad[0].conColor=false, como sabemos que no tiene ningún color asociado, le asignamos un color que esta en el tope de la pila, lo apilamos en un segundo stack y nos queda tamañoPila1=5, tamañoPila2=1 y actividad[0].conColor=true

Luego sabemos que actividad[1].conColor=false y comparando $actividad[0].fin \leq actividad[1].inicio => 6 \leq 7$, se irán destacando los números para mayor claridad en la tabla. Como sabemos que la comparación es verdadera, entonces el color que toma $actividad[1].color = actividad[0].color$

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

iteracion 2
z=1 y i=2,tamañoPila1=5, tamañoPila2=1 Primero vemos si activi-

dad[1].conColor=true, en la iteración 1 se le asigno un color a actividad[1].color, por lo tanto vamos al siguiente paso.

Luego sabemos que actividad[2].conColor=false y comparando $actividad[1].fin \leq actividad[2].inicio$ => $12 \leq 0$, lo cual es falso, por lo tanto el nodo(actividad) B pasa a una cola de actividades Cola=[B]

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

iteracion 3
z=2 y i=3,tamañoPila1=5, tamañoPila2=1 Cola=[B].Primero vemos si actividad[2].conColor=falso, en la iteración 2 no se le asigno un color a actividad[2].color, por lo tanto le asignamos un color que este al tope de pila1, y lo apilamos en pila2, lo que nos queda: tamañoPila1=4, tamañoPila2=2.

Luego sabemos que actividad[3].conColor=false y comparando $actividad[2].fin \leq actividad[3].inicio$ => $21 \leq 24$, lo cual es verdadero, por lo tanto $actividad[3].color = actividad[2].color$

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

iteracion 4
z=3 y i=4,tamañoPila1=4, tamañoPila2=2 Cola=[B].Primero vemos si actividad[3].conColor=verdadero, dado que en la iteración 3 se le

asigno un color.

Luego sabemos que actividad[4].conColor=false y comparando $actividad[3].fin \leq actividad[4].inicio$ => $28 \leq 13$, lo cual es falso, por lo tanto insertamos la actividad D en la cola,pero como tenemos la actividad B al frente de la cola, y todos sus valores asociados, comparamos $B.fin \leq actividad[4].inicio$ => $12 \leq 13$ lo cual es verdadero, por lo tanto, actividad[4].color=a.color, nos queda Cola=[D]

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

iteracion 5
z=4 y i=5,tamañoPila1=4, tamañoPila2=2 Cola=[D].Primero vemos si actividad[4].conColor=verdadero, dado que en la iteración 4 se le asigno un color.

Luego sabemos que actividad[5].conColor=false y comparando $actividad[4].fin \leq actividad[5].inicio$ => $31 \leq 29$, lo cual es falso, por lo tanto insertamos la actividad E en la cola,pero como tenemos la actividad D al frente de la cola, y todos sus valores asociados, comparamos $D.fin \leq actividad[5].inicio$ => $28 \leq 29$ lo cual es verdadero, por lo tanto, actividad[5].color=D.color, nos queda Cola=[E], como i=n-1 iteramos completamente, y asígnanos colores a todas las actividades como tamañoPila2=2, entonces se necesitan 2 colores(personas), para realizar las actividades

	s	f
a	0	6
b	7	12
c	0	21
d	24	28
e	13	34
f	29	34

5. Implemente su algoritmo para resolver el problema de número de personas mínimo utilizando las estructuras de datos que estime conveniente, y proporcione las razones por las cuales decide usar tales estructuras

   La implementación de pregunta 5, 6, 7 esta en un código c++

   Struct : Definir nodo actividad

   Vector: Usados porque necesitamos arreglos dinámicos para las actividades.

   Stack: Almacena colores en forma de un numero entero. Cada numero se asocia a un color distinto. Además se prefiere al tener todos sus métodos con complejidad O(1). Si tenemos por ejemplo 5 actividades, vamos a tener 5 colores, colores=0,1,2,3,4. Cada vez que una actividad ocupe un color por primera vez, este se desempila, y se apila en otra pila, osea en total ocupamos 2 stacks.

   Cola: Siempre que iteremos y Actvidad[i].termino>Actividad[i+1].inicio, haremos un Cola.push(i). Esto lo hacemos porque como sabemos que las actividades ya están ordenadas, en las siguientes iteraciones puede que haya una actividad que haya iniciado cuando i ya haya terminado, entonces en ese caso, una misma persona puede hacer ambas actividades.

6. Implemente un generador que permita generar aleatoriamente un conjunto de actividades donde se pueda especificar el total de actividades, n; el tiempo de inicio y término aleatorias dentro de un rango con parámetros [si, fi]. Note que las actividades generadas deben ser válidas, es decir si ¡fi

7. Implemente una función que permita leer desde un archivo un conjunto de actividades donde cada línea tiene el formato si, fi de cada actividad.

8. Modifique su algoritmo para resolver ahora el problema de coloreado de grafo de intervalos y proporcione la salida del grafo con los vértices

con su respectivo color y las aristas.

9. Use el paquete networkx de python para escribir un script que permita desplegar los grafos de intervalo coloreados por su implementación. Para ello puede usar como base el siguiente código python o hacer uno mas sofisticado.