FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

[75.29] Teoría de Algoritmos 1

Primer cuatrimestre 2022

TRABAJO PRACTICO 1

Alumno: Manuel Longo Elia
Padrón: 102425
Mail: mlongoe@fi.uba.ar
Docentes: Víctor Podberezski
Lucas Ludueño
Kevin Untrojb
Ernesto Alvarez
Martín Suarez

Fecha de entrega: 6 de Abril de 2022

Enunciado

Para una ruta recién inaugurada de “k” kilómetros de longitud se debe construir un conjunto de antenas celulares para cubrir su recorrido. Se recibieron un conjunto de “n” propuestas. Cada propuesta corresponde a la instalación de una antena en una ubicación determinada. Las características de las antenas pueden variar. Sabemos por la información de cada contrato dónde se ubicará la antena y la cantidad de kilómetros que cubrirá de la ruta expresado en un radio de una cantidad de kilómetros desde donde está ubicada.

Nos solicitan seleccionar el menor subconjunto de contratos de forma que toda la ruta quede totalmente cubierta o que informe que esto no es posible.

Resuelva:

1.Proponga al menos 2 estrategias greedy que no sean óptimas para resolver el problema. Ejemplifique. ¿Por qué considera que son del tipo greedy?

2.Proponga una estrategia greedy que sea óptima. Justifique su optimalidad.

3.Explique cómo implementar algorítmicamente esa estrategia. Brinde pseudocódigo y estructuras de datos a utilizar.

4.Analice complejidad temporal y espacial de su propuesta

5.Programe su algoritmo.

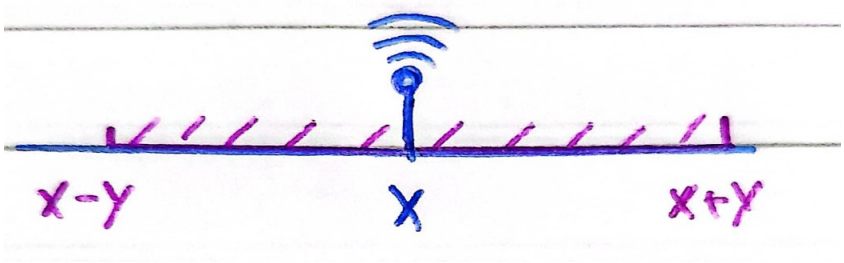
6.Analice el resultado que obtiene mediante su algoritmo. ¿Puede encontrar alguna característica que beneficie algunos contratos frente a otros? ¿Existe alguna otra solución óptima alternativa?

7.Su programa mantiene la complejidad espacial y temporal de su algoritmo?

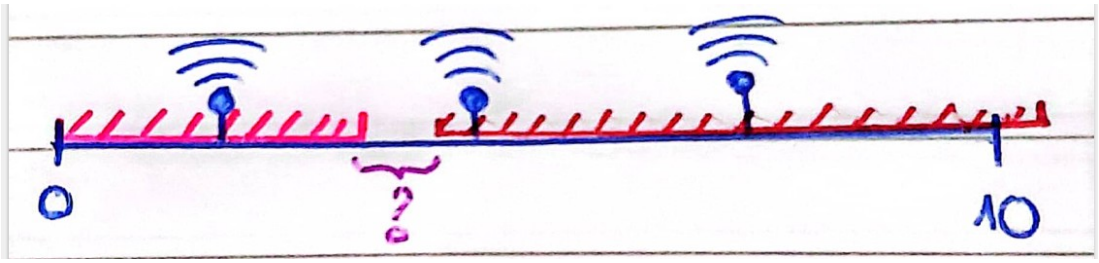
Resolución

1) Las dos estrategias de tipo greedy a utilizar son variantes del Interval Scheduling. Este es un algoritmo greedy ya que su resolución divide el problema en pequeños subproblemas con una jerarquía entre ellos, también se resuelve de manera iterativa en función de una solución heurisitica. También, esta estrategia contienen las dos propiedades básicas de los algoritmos de tipo greedy, la elección greedy y la subestructura optima. La elección greedy busca seleccionar una solución optima local esperando que esta se acerque a la solución optima global, analizando los subproblemas en el estado que llegaron al mismo y eligiendo de manera heurisitica la mejor solución local posible condicionando los posteriores estados del conjunto de elementos. La otra propiedad, subestructura optima, que dice que si la solución optima global del problema contiene en su interior las soluciones optimas de sus subproblemas la elección greedy resolverá los subproblemas óptimamente, llevando a la solución optima global.
La estrategia greedy seleccionada es Interval Scheduling, la elección esta fundamentada en que las antenas pueden verse como un evento con horario de inicio y de finalización, dado que se puede calcular el “horario de inicio” del evento como la diferencia entre la posición del poste y el radio de alcance de la señal emitida, y el “horario de finalización del evento” se puede calcular como la

suma entre la posición del poste y el radio de alcance de la señal emitida. En la imagen a continuación se muestra gráficamente como se pensó la situación anteriormente descripta.

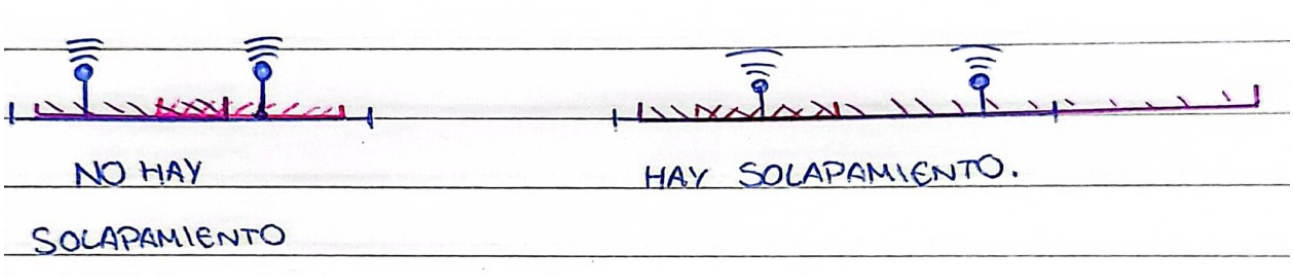


Lo que busca este algoritmo greedy es seleccionar un subconjunto de elementos del conjunto inicial de mayor tamaño posible de modo que todas las tareas seleccionadas sean compatibles entre si, es decir que no haya superposición de “eventos”.
La respuesta a ¿Por qué esta algoritmo no es óptimo? Es muy clara, el algoritmo no permite solapamientos, es decir no puede haber dos antenas conectadas al mismo tiempo que compartan una cierta área, por lo que lleva a que solo pueda conseguirse una solución optima en el caso de que las antenas estén perfectamente distribuidas consiguiendo así una conexión completa. En el siguiente imagen se muestra un ejemplo en el cual se cumple lo anteriormente explicado:



Aquí se muestra de que si se hubiese encendido la segunda antena se habría encontrado la solución optima y se lograría una cobertura celular total en el recorrido de la ruta. El resultado que se obtendría en este caso es “no es posible la cobertura total” lo cual es falso.
Por lo tanto, si se comienza el algoritmo eligiendo la antena que ocupa mayor área o la antena que este mas cerca del origen de la ruta va a ser muy difícil que se llegue a la solución optima, como se explico anteriormente. Estas dos estrategias no son las mejores.

2) A diferencia del algoritmo de Interval Scheduling puro, en nuestro caso lo que se busca es la menor cantidad posibles de “eventos” en el conjunto buscando ocupar el total de la ruta, entonces deberían hacerse unos pequeños cambios al algoritmo.
Lo primero a realizarse es una redefinición del concepto de solapamiento. En nuestra estrategia greedy dos eventos estarán solapados si el área de cobertura de la antena esta incluida en su totalidad en el área de cobertura del otro evento. En la próxima imagen se podrá visualizar esto.



Lo primero que se haría es ordenar las antenas por lugar de inicio de la cobertura. Poner el primer "evento" en la lista y eliminar de la lista de opciones los que se solapen con este. Luego se repetiría el proceso hasta que se complete, teniendo en cuenta que cada vez que se ingrese un nuevo "evento" el punto de inicio de la cobertura debe ser igual o menor al ultimo punto de la ruta cubierto, ya que si esto no sucede se estaría dejando lugares de la ruta sin cobertura.
Esta estrategia es optima ya que sacara de análisis a las antenas que no abarquen un área que no pueda ser abarcada por otra antena, dejando así las antenas mas importantes y que solo dependa la cobertura total de la ruta si las antenas que quedaron pueden abarcar el área o no.

3) El pseudocodigo de la estrategia seria el siguiente:

Sea P un set de datos
Sea A el conjunto final

Ordenar P por lugar de inicio de cobertura

Mientras P no este vacío o no se haya cubierto todo el trayecto
Sea i el pedido en P con menor lugar de inicio de cobertura
Agrego i a la lista A
Quito de P todas las antenas solapadas con i
Elimino el primer elemento del set de datos
Si el primer elemento de P tiene un lugar de inicio mayor al ultimo punto cubierto por el ultimo elemento de la lista A
Retornar que no se puede cubrir toda la ruta
Calculo los kilómetros que cubrí en esta iteración
Si toda la ruta fue cubierta
Retornar que se puede conectar toda la ruta y las antenas que deben conectarse
Si no
Retornar que no se puede cubrir la ruta entera

4) El algoritmo tendrá un proceso de ordenamiento que tiene complejidad O(n log n) y la iteración por toda la lista que es O(n). Por lo que tendrá una complejidad temporal total de O(n log n) y una complejidad espacial de O(n) en el caso de que todas las antenas sean compatibles entre si.

5) Se adjuntan imágenes del código escrito en el archivo 'código.py'

4

```python
import sys

def criterioOrdenamiento(setDeDatos):
    return (int(setDeDatos[2]) - int(setDeDatos[1]))
```

```
def quitarSolapadas(setDeDatos):
    inicioPrimeroLista = int(setDeDatos[0][1]) - int(setDeDatos[0][2])
    if inicioPrimeroLista < 0:
        inicioPrimeroLista = 0
    finalPrimeroLista  = int(setDeDatos[0][1]) + int(setDeDatos[0][2])
    numerosAEliminar = []
    for i in range(1,len(setDeDatos)):
        inicioEvaluadoActual = int(setDeDatos[i][1]) - int(setDeDatos[i][2])
        if inicioEvaluadoActual < 0:
            inicioEvaluadoActual = 0
        finalEvaluadoActual  = int(setDeDatos[i][1]) + int(setDeDatos[i][2])
        if ((inicioPrimeroLista <= inicioEvaluadoActual) & (finalPrimeroLista >= finalEvaluadoActual)):
          numerosAEliminar.append(i)

    numerosAEliminar.sort(reverse=True)

    for i in range(0,len(numerosAEliminar)):
        setDeDatos.pop(numerosAEliminar[i])

    return setDeDatos
def intervalSchedulingAdaptado(file, k):
    setDeDatos = []
    conjuntoFinal = []
    kmCubiertos = 0

    with open(file) as archivo:
        for linea in archivo:
            setDeDatos.append(linea.split()[0].split(','))

    setDeDatos.sort(reverse=True, key=criterioOrdenamiento)

    while ( (len(setDeDatos) != 0) and (kmCubiertos != int(k)) ) :
        conjuntoFinal.append(setDeDatos[0])
        setDeDatos = quitarSolapadas(setDeDatos)
        setDeDatos.pop(0)

        inicioUltimoLista = int(conjuntoFinal[-1][1]) - int(conjuntoFinal[-1][2])
        finalUltimoLista  = int(conjuntoFinal[-1][1]) + int(conjuntoFinal[-1][2])
        if finalUltimoLista > int(k):
            finalUltimoLista = int(k)
        if inicioUltimoLista > kmCubiertos:
            print('No es posible cubrir toda la ruta con las propuestas existentes ')
            return -1
        kmCubiertos += (finalUltimoLista - kmCubiertos)

    if kmCubiertos < int(k):
        print('No es posible cubrir toda la ruta, solamente se pudieron cubrir ', kmCubiertos,' kilometros.')
        return -1
    else:
        print('Se puede cubrir la ruta completa. Las propuestas seleccionadas son:', end=" ")
        for i in range(0,len(conjuntoFinal)):
            print(conjuntoFinal[i][0], end=" ")
        print('\nSe cubriran ', kmCubiertos, ' kilometros.')
        return 0

intervalSchedulingAdaptado("contratos.txt", sys.argv[1])
```

6) Con el set de datos que proporciona el enunciado se puede obtener lo siguiente:

```
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 100
Se puede cubrir la ruta completa. Las propuestas seleccionadas son: 1 5
Se cubriran  100  kilometros.
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 300
Se puede cubrir la ruta completa. Las propuestas seleccionadas son: 1 5 4 6
Se cubriran  300  kilometros.
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 480
Se puede cubrir la ruta completa. Las propuestas seleccionadas son: 1 5 4 6 2
Se cubriran  480  kilometros.
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 552
Se puede cubrir la ruta completa. Las propuestas seleccionadas son: 1 5 4 6 2
Se cubriran  552  kilometros.
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 600
No es posible cubrir toda la ruta, solamente se pudieron cubrir  552  kilometros.
```

Los cuales son los resultados que se esperan si se analizan los datos provistos. Lo máximo que puede llegar a cubrirse por las antenas dadas es 552 kilómetros.
Ahora, si se modifican los datos para ver si teniendo un espacio sin cobertura se obtiene el resultado esperado el programa muestra lo siguiente:

```
(base) manulon@manulongo:~/Escritorio/TEORIA DE ALGORITMOS I/TPS/TP1$ python3 codigo.py 480
No es posible cubrir toda la ruta con las propuestas existentes
```

El set de datos utilizado para obtener este resultado es:

1,44,50
2,402,150
3,219,35

La característica clara que se puede ver es que las antenas con mayor cobertura tienen muchas mas chances de ser contratada que las que cubran poco, ya que hay mucha mas chances de que estas se solapen con otras. La solución encontrada es la optima ya que encuentra la menor cantidad de antenas para cubrir el espacio requerido.

7) Si, el algoritmo mantiene la complejidad espacial y temporal deseada. Ya que el código y pseudocodigo siguen al pie de la letra lo explicado en los puntos anteriores. Lo único que puede generar dudas es el algoritmo de la librería estándar de python. Según documentación oficial el algoritmo utilizado es ‘TimSort’ que tiene una complejidad de O(n log n) tal como se describió anteriormente.

INSTRUCCIONES DE COMPILACIÓN Y CÓDIGO FUENTE

Para compilar simplemente debe ejecutarse en la terminal el siguiente comando
python3 codigo.py k
Siendo k el largo de la ruta a cubrir

El código fuente se mostrara a continuación. Es el mismo que esta mostrado en las imágenes de las respuestas anteriores. El código, el set de datos de ejemplo y este informe esta subido en el repositorio de Github público *https://github.com/manulon/TeoriaDeAlgoritmos1*

```
import sys

def criterioOrdenamiento(setDeDatos):
    return (int(setDeDatos[2]) - int(setDeDatos[1]))

def quitarSolapadas(setDeDatos):
    inicioPrimeroLista = int(setDeDatos[0][1]) - int(setDeDatos[0][2])
    if inicioPrimeroLista < 0:
        inicioPrimeroLista = 0
    finalPrimeroLista  = int(setDeDatos[0][1]) + int(setDeDatos[0][2])
    numerosAEliminar = []
    for i in range(1,len(setDeDatos)):
        inicioEvaluadoActual = int(setDeDatos[i][1]) - int(setDeDatos[i][2])
        if inicioEvaluadoActual < 0:
            inicioEvaluadoActual = 0
        finalEvaluadoActual  = int(setDeDatos[i][1]) + int(setDeDatos[i][2])
        if ((inicioPrimeroLista <= inicioEvaluadoActual) & (finalPrimeroLista >=
finalEvaluadoActual)):
            numerosAEliminar.append(i)

    numerosAEliminar.sort(reverse=True)

    for i in range(0,len(numerosAEliminar)):
        setDeDatos.pop(numerosAEliminar[i])

    return setDeDatos

def intervalSchedulingAdaptado(file, k):
    setDeDatos = []
    conjuntoFinal = []
    kmCubiertos = 0

    with open(file) as archivo:
        for linea in archivo:
            setDeDatos.append(linea.split()[0].split(','))
```

```
    setDeDatos.sort(reverse=True, key=criterioOrdenamiento)

    while ( (len(setDeDatos) != 0) and (kmCubiertos != int(k)) ) :
        conjuntoFinal.append(setDeDatos[0])
        setDeDatos = quitarSolapadas(setDeDatos)
        setDeDatos.pop(0)

        inicioUltimoLista = int(conjuntoFinal[-1][1]) - int(conjuntoFinal[-1][2])
        finalUltimoLista  = int(conjuntoFinal[-1][1]) + int(conjuntoFinal[-1][2])
        if finalUltimoLista > int(k):
            finalUltimoLista = int(k)
        if inicioUltimoLista > kmCubiertos:
            print('No es posible cubrir toda la ruta con las propuestas existentes ')
            return -1
        kmCubiertos += (finalUltimoLista - kmCubiertos)

    if kmCubiertos < int(k):
        print('No es posible cubrir toda la ruta, solamente se pudieron cubrir ',
kmCubiertos,' kilometros.')
        return -1
    else:
        print('Se puede cubrir la ruta completa. Las propuestas seleccionadas son:',
end=" ")
        for i in range(0,len(conjuntoFinal)):
            print(conjuntoFinal[i][0], end=" ")
        print('\nSe cubriran ', kmCubiertos, ' kilometros.')
        return 0

intervalSchedulingAdaptado("contratos.txt", sys.argv[1])
```