Algorítmica

Capitulo 3. Algoritmos Greedy

Ejercicios prácticos

Objetivos de las prácticas

- Con estas prácticas se persigue:
  1. Apreciar la utilidad de los algoritmos greedy para resolver problemas de forma muy eficiente, en algunos casos obteniendo soluciones óptimas y en otros soluciones cercanas a las óptimas.
  2. Constatar la utilidad del enfoque greedy en problemas que se planteen sobre grafos.
  3. Comprobar la utilidad de las heurísticas en Algorítmica
  4. Trabajar comprometidamente en equipo
  5. Aprender a expresar en público las ventajas, inconvenientes y alternativas empleadas, para lograr la solución alcanzada

Contenedores en un barco

- Se tiene un buque mercante cuya capacidad de carga es de K toneladas y un conjunto de contenedores $c_1, \ldots, c_n$ cuyos pesos respectivos son $p_1, \ldots, p_n$ (expresados también en toneladas).
- Teniendo en cuenta que la capacidad del buque es menor que la suma total de los pesos de los contenedores:
  - Diseñar un algoritmo que maximice el número de contenedores cargados. Demostrar su optimalidad.
  - Diseñar un algoritmo que intente maximizar el número de toneladas cargadas.

Contenedores en un barco

- Contemplamos dos algoritmos:
- Uno, maximizando el número de contenedores cargados
- Otro, maximizando el número de toneladas cargadas
- Algoritmo maximizando el número de contenedores cargados
- El algoritmo greedy para el primer caso simplemente escoge en cada momento el contenedor todavía no cargado en el barco que tenga el menor peso. O dicho de otra forma, selecciona los contenedores en orden no decreciente de peso.
- El algoritmo es optimal

Contenedores en un barco

- Veamos ahora el algoritmo maximizando el número de toneladas cargadas.
- En este caso parece intuitivo que el algoritmo greedy debe de intentar cargar en cada momento el contenedor de mayor peso de los que queden aun sin cargar. En otras palabras, cargar los contenedores en orden no creciente de peso.
- El algoritmo no es da siempre el óptimo

Minimizar el número de visitas al proveedor

- Un granjero necesita disponer siempre de un determinado fertilizante. La cantidad máxima que puede almacenar la consume en *r* días, y antes de que eso ocurra necesita acudir a una tienda del pueblo para abastecerse.
- El problema es que dicha tienda tiene un horario de apertura muy irregular (solo abre determinados días). El granjero conoce los días en que abre la tienda, y desea minimizar el número de desplazamientos al pueblo para abastecerse.
  - Diseñar un algoritmo greedy que determine en qué días debe acudir al pueblo a comprar fertilizante durante un periodo de tiempo determinado (por ejemplo durante el siguiente mes).
  - Demostrar que el algoritmo encuentra siempre la solución óptima.

Minimizar el número de visitas al proveedor

- La idea del algoritmo (greedy) es muy simple: aguantar lo máximo posible sin ir al pueblo, es decir acudir al pueblo en un día de apertura de la tienda, de modo que si no fuese ese día y acudiese en un día de apertura posterior, se quedaría sin fertilizante.
- Para formalizar el problema, sean $d_1, d_2, \ldots, d_n$ los días en que abre la tienda durante el periodo de interés a partir del día de inicio ($d_0 = 0$), y d_{n+1} el día final del periodo de interés, de modo que si $i < j$ entonces $d_i < d_j$
- Para que el problema tenga solución, el número de días entre aperturas sucesivas de la tienda no puede ser mayor el número de días que tardamos en consumir el fertilizante (en caso contrario siempre nos quedaríamos sin fertilizante antes de la siguiente apertura), o sea $d_{i+1} - d_i \leq r \ \forall i = 0,\ldots,n$.
- Una solución factible es una lista ordenada (de los días en que iremos al pueblo), $d_{j_0}, d_{j_1}, \ldots, d_{j_m}, d_{j_{m+1}}$, tal que

$$d_{j_0} = d_0,\ d_{j_{m+1}} = d_{n+1},\ \text{y}\ d_{j_{i+1}} - d_{j_i} \leq r$$

Minimizar el número de visitas al proveedor

- La estrategia greedy es la siguiente: si nos encontramos en el día de visita d_i, entonces la siguiente visita será en el día d_{j^*} tal que $j^* = \max_{j>i}\{j/d_j - d_i \leq r\}$ eso equivale a decir que j^* es el día tal que $d_{j^*} - d_i \leq r$ pero $d_{j^*+1} - d_i > r$
- Así puede probarse que la solución que se obtiene es optimal.
- Construir el algoritmo y demostrar que siempre da la solución óptima

Minimizar el tiempo medio de acceso

- Sean n programas P_1, P_2, . . . , P_n que hay que almacenar en una cinta. El programa P_i requiere s_i kilobytes de espacio y la cinta es suficientemente larga para almacenar todos los programas.
- Se sabe con qué frecuencia se utiliza cada programa: una fracción π_i de las solicitudes afecta al programa P_i (y por tanto $\sum_{i=1}^{n} \pi_i = 1$).
- Los datos se almacenan en la cinta con densidad constante y la velocidad de la cinta también es constante.
- Una vez que se carga el programa, la cinta se rebobina hasta el principio.

Minimizar el tiempo medio de acceso

- Si los programas se almacenan por orden $i_1, i_2, \ldots, i_n$ el tiempo medio requerido para cargar un programa es, por tanto:

$$\hat{T} = c \sum_{j=1}^{n} \left[\pi_{i_j} \sum_{k=1}^{j} s_{i_k} \right]$$

- donde la constante c depende de la densidad de grabación y de la velocidad de la cinta.
- Se desea minimizar T^ empleando un algoritmo greedy. Demostrar la optimalidad del algoritmo o encontrar un contraejemplo que demuestre que el algoritmo no es óptimo para los siguientes criterios de selección:
  - Programas en orden no decreciente de s_i.
  - Programas en orden no creciente de π_i.
  - Programas en orden no creciente de π_i/s_i.

Minimizar el tiempo medio de acceso

- Por tanto los criterios (a) y (b) no producen un algoritmo greedy óptimo.
- Para demostrar que las dos primeras opciones para seleccionar de forma greedy el orden de almacenamiento de los programas no son óptimas, basta con encontrar un ejemplo en el que los tiempos de acceso obtenidos sean mayores que los obtenidos por alguna ordenación de los programas.
- Demostrar que el criterio de selección c) si produce el óptimo

Recubrimiento de un grafo no dirigido

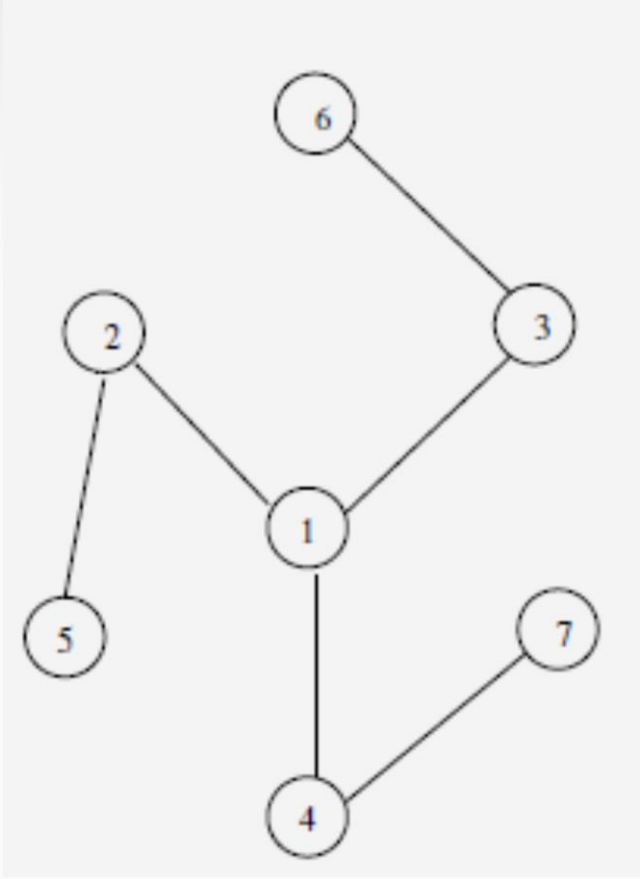
- Sea G = (V,E) un grafo no dirigido. Un conjunto U se dice que es un recubrimiento de G si $U \subseteq V$ y cada arista en E incide en, al menos, un vértice o nodo de U, es decir $\forall(x, y) \in E$, o bien $x \in U$ o $y \in U$. Un conjunto de nodos es un recubrimiento minimal de G si es un recubrimiento con el menor número posible de nodos.
- Diseñar un algoritmo greedy para intentar obtener un recubrimiento minimal de G. Demostrar que el algoritmo es correcto, o dar un contraejemplo.
- Diseñar un algoritmo greedy que obtenga un recubrimiento minimal para el caso particular de grafos que sean árboles y opcionalmente, realizar un estudio experimental de las diferencias entre los dos algoritmos anteriores cuando ambos se aplican a árboles.

Recubrimiento de un grafo no dirigido

- La estrategia greedy consistirá en ir seleccionando de uno en uno los nodos que formarán el recubrimiento.
- Una función de selección razonable sería escoger en cada paso el nodo con mayor grado de incidencia de entre los nodos aún no seleccionados (el grado de incidencia de un vértice es el número de aristas que inciden en él). De este modo se intenta recubrir el mayor número posible de aristas con el menor número de nodos.

Recubrimiento de un grafo no dirigido

- Este algoritmo no garantiza encontrar un recubrimiento minimal.
- Por ejemplo, para el grafo G = (V, E) de la figura: V = {1, 2, 3, 4, 5, 6, 7} y E = {(1, 2), (1, 3), (1, 4), (2, 5), (3, 6), (4, 7)}, el algoritmo anterior da como resultado U = {1, 2, 3, 4} o también, cambiando 5 por 2, 6 por 3 y 7 por 4. El tamaño del conjunto es 4.
- Sin embargo, la solución óptima es U* = {2, 3, 4}, de tamaño 3.



Recubrimiento de un grafo no dirigido

- En un grafo que sea un árbol (o un bosque de arboles) siempre podemos encontrar al menos un nodo hoja (un nodo con una única arista incidente en el).
- Para poder recubrir esa arista es necesario que alguno de sus nodos extremos esté en el recubrimiento. Seleccionaremos el único nodo adyacente al nodo hoja (que tiene la posibilidad de recubrir otras aristas, mientras que el nodo hoja solo puede recubrir a su arista incidente).
- Si eliminamos del árbol todas las aristas incidentes en el nodo seleccionado, el resultado sigue siendo un bosque. Por tanto sigue existiendo al menos un nodo hoja en ese grafo.
- Repetimos el proceso hasta que hayamos eliminado todas las aristas del árbol (cuando tengamos solo una arista aislada, podemos escoger cualquiera de sus nodos extremos). El resultado es necesariamente un recubrimiento minimal, porque todos los nodos seleccionados son imprescindibles para poder recubrir al menos una de las aristas.

Reparaciones

- Un electricista necesita hacer n reparaciones urgentes, y sabe de antemano el tiempo que le va a llevar cada una de ellas: en la tarea i-ésima tardará t_i minutos.
- Como en su empresa le pagan dependiendo de la satisfacción del cliente y esta es inversamente proporcional al tiempo que tardan en atenderles, necesita decidir el orden en el que atenderá los avisos para minimizar el tiempo medio de atención de los clientes (desde el inicio hasta que termine la reparación)
- Diseñar un algoritmo greedy para resolver esta tarea. Demostrar que el algoritmo obtiene la solución óptima.
- Modificar el algoritmo anterior para el caso de una empresa en la que se disponga de los servicios de más de un electricista.

Reparaciones

- Minimizar el tiempo medio de atención es lo mismo que minimizar el tiempo total de atención.
- Como un cliente j espera mientras atienden a todos los clientes anteriores a él $i_1, i_2, \ldots, i_m$, el incremento en tiempo que se obtiene al añadir al cliente j es igual a la suma de los tiempos de servicio de esos m clientes,

$$\sum_{k=1}^{m} t_{i_k}$$

- (porque eso es lo que el cliente j tiene que esperar antes de recibir servicio), mas t_j (el tiempo necesario para servir a ese cliente). Por tanto si se quiere minimizar el tiempo total, parece que lo mejor es atender a continuación al cliente con menor tiempo de servicio.
- Así la estrategia greedy es atender en cada momento al cliente que requiera menor tiempo de servicio de entre los pendientes. Dicho de otra manera, atender a los cliente en orden no decreciente de tiempos de servicio.

Reparaciones

- Si suponemos que hay M electricistas, el criterio no cambia esencialmente:
- Se ordenan las tareas de forma no decreciente de tiempo de servicio y vamos asignando las tareas al primer electricista que quede desocupado.
- En definitiva, al electricista k-ésimo le asignamos las tareas (ordenadas) k, k + M, k + 2M,...