Diseño de Algoritmos - Algoritmos II

Nazareno Aguirre, Sonia Permigiani, Gastón Scilingo, Simón Gutiérrez
Departamento de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto

Clase 7: Algoritmos Greedy

Estrategias de Diseño de Algoritmos

Las estrategias de diseño de algoritmos nos brindan herramientas para atacar el problema de construir soluciones algorítmicas para problemas. Hemos visto ya algunas de estas estrategias, tales como Fuerza Bruta, Divide & Conquer y Programación Dinámica.

Hasta el momento, hemos utilizado estas técnicas para construir soluciones algorítmicas exactas para diferentes problemas. Veremos ahora una nueva técnica, denominada Técnica Greedy, que si bien puede usarse para resolver problemas de forma exacta, tiene una enorme aplicación en la construcción de soluciones algorítmicas aproximadas para problemas.

Técnica Greedy

Una forma general de describir la técnica Greedy es la siguiente:

Dado un problema P, construya una solución (no necesariamente exacta) para P basada en la siguiente estrategia:

> Para resolver el problema P, se realizarán una secuencia de pasos. En cada paso se "expandirá" una solución parcial construida hasta ese momento que sea localmente optimal, en el sentido de que, con la información disponible en el momento, sea la mejor solución entre todas las posibilidades localmente visibles.
>
> Más precisamente, cada paso de "expansión" de la solución parcial construida debe dar:
>
> - una solución localmente optimal,
> - un paso hacia la solución global al problema que sea irrevocable (i.e., no puede ser revisado más adelante en el cómputo de la solución global)

Ejemplo: Dar Cambio

Consideremos el problema de dar cambio por C centavos, usando monedas de valores $d_1 > d_2 > \ldots > d_k$, de manera tal que el número de monedas dadas sea minimal.

Supongamos por ejemplo que tenemos la siguiente instancia de este problema:

> C = 63
>
> $d_1 = 25$, $d_2 = 10$, $d_3 = 5$, $d_4 = 1$.

La solución (greedy) que uno aplicaría sería simplemente la siguiente:

> Elegir la moneda d_i más grande posible que no exceda el valor C, e incluirla en la solución. Reducir el problema a dar cambio por $C - d_i$.

En este caso tendríamos la siguiente sucesión de pasos:

> C = 63, Cambio = {}; C = 38, Cambio = {25}; C = 13, Cambio = {25, 25};
>
> C = 3, Cambio = {25, 25, 10}; C = 2, Cambio = {25, 25, 10, 1};
>
> C = 1, Cambio = {25, 25, 10, 1, 1}; C = 0, Cambio = {25, 25, 10, 1, 1, 1}

Ejemplo: Dar Cambio (cont.)

La solución greedy descripta anteriormente es óptima. Esto se debe a una propiedad particular de los valores asociados con las monedas, pero no se cumple en todos los casos. Consideremos por ejemplo la siguiente instancia del problema de dar cambio:

C = 25

$d_1 = 7$, $d_2 = 5$, $d_3 = 1$.

La solución greedy daría por resultado lo siguiente:

C = 25, Cambio = {}; C = 18, Cambio = {7}; C = 11, Cambio = {7, 7};

C = 4, Cambio = {7, 7, 7}; C = 3, Cambio = {7, 7, 7, 1}; C = 2, Cambio = {7, 7, 7, 1, 1};

C = 1, Cambio = {7, 7, 7, 1, 1, 1}; C = 0, Cambio = {7, 7, 7, 1, 1, 1, 1}.

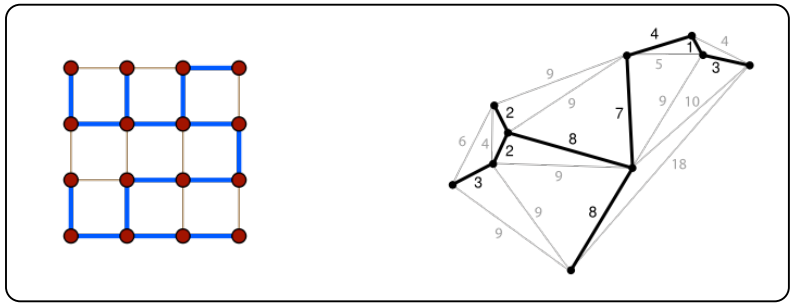
Esta solución no es óptima: {5, 5, 5, 5, 5} es un cambio válido que requiere menos monedas.

Ejemplo: Árbol Abarcador de Costo Mínimo

Consideremos ahora el siguiente problema. Dado un grafo conexo no dirigido G, con costos en los arcos, se desea obtener el árbol abarcador de costo mínimo (minimum spanning tree).

Un árbol abarcador de un grafo conexo G es un subgrafo G' de G que es acíclico, conexo y contiene todos los nodos de G.

Ej.:



Ejemplo: Árbol Abarcador de Costo Mínimo (cont.)

Un algoritmo para resolver el problema de computar el árbol abarcador de costo mínimo es el Algoritmo de Prim. Este es otro ejemplo de un algoritmo Greedy, que consiste en lo siguiente:

1. Tomar cualquier vértice del grafo G. Esta es la primera solución parcial T1.

2. En cada iteración, construir el árbol T(i+1) agregando a Ti el vértice más cercano a Ti que no pertenece ya a Ti.

3. Terminar cuando ya no queden vértices por agregar.

Claramente, el segundo paso toma decisiones de manera "greedy".

Ejemplo: Árbol Abarcador de Costo Mínimo (cont.)

Otro algoritmo Greedy para computar el árbol abarcador de costo mínimo es el Algoritmo de Kruskal. Éste consiste en lo siguiente:

1. Ordenar los arcos según sus costos (de forma no decreciente).

2. Comenzar con el arco (y los vértices involucrados) de mínimo costo. Esto constituye la solución parcial inicial T1.

3. En cada iteración, construir T(i+1) agregando el arco de menor costo que no pertenezca a Ti y que no genere un ciclo.

4. Terminar cuando Ti sea conexo e involucre todos los nodos de G.

Nuevamente, el paso 3 corresponde a una decisión Greedy.

Ejemplo: El Problema del Agente Viajero

Consideremos un nuevo problema. Este es el problema conocido como El Problema del Agente Viajero, y consiste en lo siguiente

> Dado un grafo no dirigido G con costos en los arcos, encontrar un ciclo Hamiltoniano (un ciclo simple que involucre todos los vértices de G) cuyo costo sea mínimo.

Este problema es un problema clásico de optimización, con numerosas aplicaciones prácticas (en particular, en encontrar rutas óptimas para visitar un número de lugares y volver a la posición inicial).

Ejemplo: El Problema del Agente Viajero (cont.)

Un algoritmo Greedy para resolver este problema es el basado en una variante del algoritmo de Kruskal. Este algoritmo funciona de la siguiente manera:

1. Ordenar los arcos según sus costos (de forma no decreciente).

2. Comenzar con el arco (y los vértices involucrados) de mínimo costo. Esto constituye la solución parcial inicial T1.

3. En cada iteración, construir T(i+1) agregando el arco de menor costo que no pertenezca a Ti, que no genere un ciclo (excepto cuando se hayan considerado todos los nodos), y que no genere vértices con grafo mayor a 2.

4. Terminar cuando Ti sea conexo e involucre todos los nodos de G.

Nuevamente, el paso 3 corresponde a una decisión Greedy.

Ejemplo: Coloreo de Grafos

El problema de coloreo de grafos consiste en lo siguiente:

> Dado un grafo G, asignar la mínima cantidad de colores a los nodos de manera tal que, si dos nodos x e y son adyacentes en G, reciban colores diferentes.

Al igual que el Problema del Agente Viajero, este problema pertenece a la clase de problemas NP-completos. Para estos problemas, como veremos más adelante en la materia, las únicas soluciones óptimas conocidas son del tipo "probar todas las posibilidades".

En nuestro caso, probar todas las posibilidades consiste en probar con todas las posibles asignaciones de colores a nodos, comenzando con un color, luego probando con dos, y así sucesivamente hasta conseguir una coloración válida.

Una Alternativa Greedy para el Coloreo de Grafos

Una heurística razonable para el problema de coloreo mínimo de grafos es la siguiente:

> Comenzar a visitar los nodos en orden, coloreando todos los nodos que se pueda con un color, sin causar conflictos entre nodos adyacentes. Si quedan nodos sin colorear, tomar un nuevo color y repetir el coloreo de tantos nodos no coloreados como se pueda, nuevamente sin causar conflictos. Continuar el proceso con nuevos colores hasta que ya no queden nodos sin colorear.

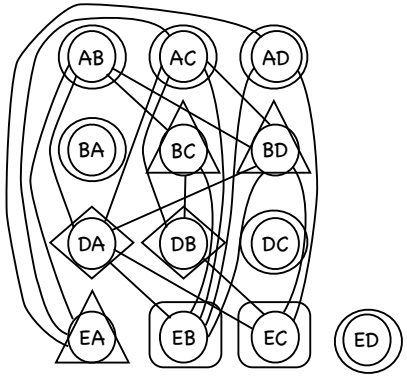
Proceso Greedy para el Coloreo de Grafos

Veamos un ejemplos de cómo funciona la heurística descripta:

Comenzar a visitar los nodos en orden, coloreando todos los nodos que se pueda con un color, sin causar conflictos entre nodos adyacentes.

Si quedan nodos sin colorear, tomar un nuevo color y repetir el coloreo de tantos nodos no coloreados como se pueda, nuevamente sin causar conflictos.

Continuar el proceso con nuevos colores hasta que ya no queden nodos sin colorear.



Otros Ejemplos de Algoritmos Greedy

Algunos ejemplos más de algoritmos Greedy son los siguientes:

1. El algoritmo de Dijkstra para computar caminos de costo mínimo desde un nodo determinado.

3. El algoritmo de construcción de la codificación de Huffman.

4. Algoritmos Greedy para el Problema de la Mochila

Matroides

Muchos problemas de optimización se pueden formular como la búsqueda de subconjuntos de cierto dominio que maximizan una función objetivo. Los matroides son estructuras algebraicas que capturan este tipo de problemas, y que bajo ciertas condiciones admiten soluciones greedy optimales.

Un matroide es un par (S, l), donde

- S es un conjunto finito
- l es una familia no vacía de subconjuntos de S, denominados subconjuntos independientes de S, que cumplen las siguientes propiedades:

$\forall A, B \cdot B \in l \wedge A \subseteq B \Rightarrow A \in l$ (hereditariedad)

$\forall A, B \in l \cdot |A| < |B| \Rightarrow \exists\, x \in B - A \cdot A \cup \{x\} \in l$ (intercambio)

Propiedad: todos los subconjuntos independientes maximales de un matroide tienen el mismo tamaño.

Matroides con Costos

Un matroide M = (S, l) tiene costos si a cada elemento x de S se puede asignar un costo no negativo w(s). Cada elemento de l tendrá igualmente un costo asociado, definido por:

$$w(A) = \sum_{x \in A} w(x)$$

Cómputo Greedy de Subconjuntos Optimales en Matroides con Costos

Sea M = (S, I) un matroide con costo w. El siguiente algoritmo retorna un subconjunto optimal A, con respecto a w.

```
GREEDY(M, w)
    A <- emptyset
    ordenar S[M] en orden decreciente por peso w
    para cada x en S[M] (tomados en orden decreciente)
        if A u {x} pertenece a I[M] entonces A <- A u {x}
    retornar A
```

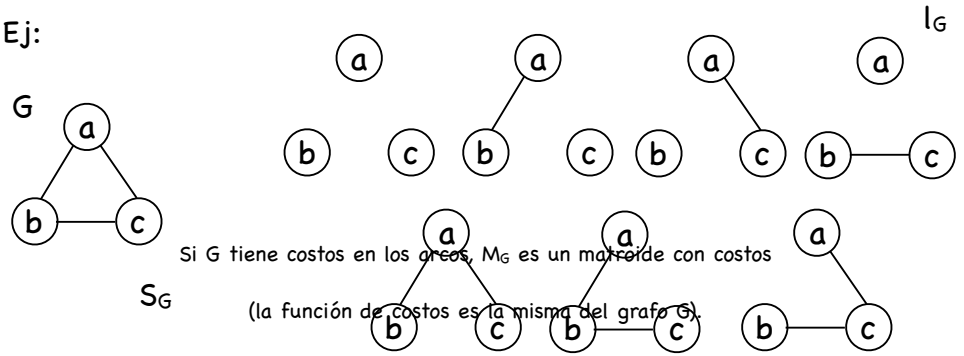
Teorema: Si M = (S, I) es un matroide con costo w, entonces GREEDY(M, w) retorna un subconjunto optimal (con respecto a w).

Ejemplo: Matroides a partir de Grafos para Árbol Abarcador de Costo Mínimo

Dado un grafo no dirigido G = (V, E), podemos construir un matroide MG = (S_G, I_G) para G, de la siguiente manera:

- S_G = E
- Si A es un subconjunto E, A pertenece a I_G si y sólo si A es acíclico.

Ej:



Si G tiene costos en los arcos, M_G es un matroide con costos (la función de costos es la misma del grafo G).

Matroides a partir de Grafos (cont.)

Teorema: Dado un grafo (finito) no dirigido G = (V, E), la estructura M_G = (S_G, I_G) es un matroide.

Demostración: La primera condición para que M_G sea un matroide es trivialmente satisfecha: S_G = E es obviamente finito. Demostremos hereditariedad en I_G. Sea A un subconjunto de E, acíclico, y B subconjunto de A. Claramente, A es un subconjunto acíclico de E, con lo cual A pertenece a I_G.

Finalmente, demostremos la propiedad de intercambio. Sean A y B subconjuntos acíclicos de E, tales que |A| < |B|. Dado que A tiene menos arcos que B, A está compuesto por menos árboles que B. Luego, B debe contener un árbol T y un par de vértices (adyacentes) x e y en él, tales que x e y se encuentran en árboles diferentes en A. Dado que x e y están en árboles diferentes en A, A u {(x,y)} es acíclico (conecta dos componentes acíclicas no conexas), y por lo tanto A u {(x,y)} pertenece a I_G.