Algoritmos Greedy

Dante Zanarini

12 de abril de 2010

La estrategia a seguir

- Un algoritmo greedy elige, en cada paso, una solución local óptima
- En general, son bastante sencillos de programar
- *¿desafortunadamente?*, no siempre conducen al óptimo

Algunos ejemplos

- Si tenemos que dar un vuelto, elegir la moneda (o billete) de mayor denominación que no supere el monto a devolver

Algunos ejemplos

- Si tenemos que dar un vuelto, elegir la moneda (o billete) de mayor denominación que no supere el monto a devolver
- En el problema del árbol recubridor minimal, elegir la arista de menor peso que no cierre ciclos

Algunos ejemplos

- Si tenemos que dar un vuelto, elegir la moneda (o billete) de mayor denominación que no supere el monto a devolver
- En el problema del árbol recubridor minimal, elegir la arista de menor peso que no cierre ciclos
- En el problema del viajante, elegir la ciudad más cercana a la actual

Algunos ejemplos

- Si tenemos que dar un vuelto, elegir la moneda (o billete) de mayor denominación que no supere el monto a devolver
- En el problema del árbol recubridor minimal, elegir la arista de menor peso que no cierre ciclos
- En el problema del viajante, elegir la ciudad más cercana a la actual
- Al armar el código de Huffman, elegir los dos subárboles de menor frecuencia y "unirlos"

¿Puedo siempre aplicar una estrategia greedy?

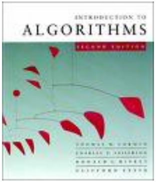
- Para un problema de optimización, es bastante fácil encontrar una estrategia greedy
- Las estrategias greedy encuentran una solución maximal al problema.
- La solución que tenemos no se puede mejorar, a menos, claro está, que volvamos atrás y revisemos las elecciones que hicimos.
- La cuestión está en ver si esa solución **maximal** es realmente **máxima**

Ejemplo, el problema de la mochila fraccionario y 0-1

capacidad: 5 kilos

1 kilo, \$60

2 kilos, \$100



3kilos, \$120

Problemas que encuentran el óptimo siguiendo alguna estrategia greedy

- Árbol recubridor minimal
- Algoritmo del camino más corto
- Problema de la mochila fraccionaria
- Construcción del código de Huffman
- Problemas de asignación de tareas (algunos)
- Problema del cambio (con algunas denominaciones)

Problemas que no encuentran el óptimo siguiendo una estrategia greedy

- Matching
- Problema del viajante
- SAT
- Coloreo
- Paginación
- Problema del cambio (con algunas denominaciones)

La pregunta del día

- **¿Cómo me puedo dar cuenta si una estrategia greedy encuentra el óptimo?**
- Veremos dos enfoques sobre este asunto

Diseñando un algoritmo greedy óptimo

1. Pensemos el problema P como uno en el cual, en cada paso, hay que tomar una decisión (greedy) y luego resolver un subproblema P' de P.

Diseñando un algoritmo greedy óptimo

1. Pensemos el problema P como uno en el cual, en cada paso, hay que tomar una decisión (greedy) y luego resolver un subproblema P' de P.
2. Demostramos que siempre hay una solución óptima del problema que "toma" la elección greedy. Por lo tanto, esta elección es "segura"

Diseñando un algoritmo greedy óptimo

1. Pensemos el problema P como uno en el cual, en cada paso, hay que tomar una decisión (greedy) y luego resolver un subproblema P' de P.
2. Demostramos que siempre hay una solución óptima del problema que "toma" la elección greedy. Por lo tanto, esta elección es "segura"
3. Demostramos que, una vez que realizamos la elección greedy, lo que queda es un subproblema tal que, si combinamos su solución con la elección greedy, entonces obtenemos una solución del problema original (subestructura óptima)

Un ejemplo

- Tenemos n tareas $S = \{a_1, \ldots, a_n\}$ que requieren acceso exclusivo a un recurso.
- Cada tarea tiene un tiempo de comienzo s_i y un tiempo de finalización f_i.
- Dos tareas a_i, a_j son compatibles si $s_i \geq f_j$ o $s_j \geq f_i$
- Hay que elegir el máximo número de tareas compatibles dos a dos

Una instancia del problema:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Algunas soluciones maximales:

Una instancia del problema:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
			$\uparrow$						$\uparrow$		$\uparrow$

Algunas soluciones maximales:

- $\{a_3, a_9, a_{11}\}$

Una instancia del problema:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$			$\uparrow$				$\uparrow$			$\uparrow$

Algunas soluciones maximales:

- $\{a_3, a_9, a_{11}\}$
- $\{a_1, a_4, a_8, a_{11}\}$

Una instancia del problema:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
		$\uparrow$		$\uparrow$					$\uparrow$		$\uparrow$

Algunas soluciones maximales:

- $\{a_3, a_9, a_{11}\}$
- $\{a_1, a_4, a_8, a_{11}\}$
- $\{a_2, a_4, a_9, a_{11}\}$

Tratemos de pensar una estrategia greedy...

1. Sea S_k el conjunto de tareas que comienzan después de la tarea k. Es decir, si elegí la tarea k, S_k es el conjunto de tareas compatibles con k (que comienzan después)
2. Inicialmente, el problema es sobre el conjunto $S_0 = S$

Tratemos de pensar una estrategia greedy...

1. Sea S_k el conjunto de tareas que comienzan después de la tarea k. Es decir, si elegí la tarea k, S_k es el conjunto de tareas compatibles con k (que comienzan después)
2. Inicialmente, el problema es sobre el conjunto $S_0 = S$
3. Para resolver S_j, elijo la tarea que termine lo antes posible (la idea es dejar la mayor cantidad de tiempo para las que quedan). Es decir, elijo i tal que $f_i = \min \{f_k \mid a_k \in S_j\}$

Elección greedy

1. Sea S_k el conjunto de tareas que comienzan después de la tarea k. Es decir, si elegí la tarea k, S_k es el conjunto de tareas compatibles con k (que comienzan después)
2. Inicialmente, el problema es sobre el conjunto $S_0 = S$
3. Para resolver S_j, elijo la tarea que termine lo antes posible (la idea es dejar la mayor cantidad de tiempo para las que quedan). Es decir, elijo i tal que $f_i = \min \{f_k \mid a_k \in S_j\}$
4. Supongamos que es la tarea i, entonces, el subproblema que me queda es elegir el número máximo de tareas entre $S_i = \{a_k \in S_j \mid f_i \leq s_k\}$

Subestructura

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$										

- Elijo a_1

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$		$\times$					$\times$	

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$					$\times$	

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\times$			$\times$	

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4
- Puedo descartar a_6, a_7

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\times$	$\uparrow$		$\times$	

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4
- Puedo descartar a_6, a_7
- Elijo a_8

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4
- Puedo descartar a_6, a_7
- Elijo a_8
- Puedo descartar a_9

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\uparrow$

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4
- Puedo descartar a_6, a_7
- Elijo a_8
- Puedo descartar a_9
- Elijo a_{11}

Veamos cómo trabaja el algoritmo en el ejemplo

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14
	$\uparrow$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\times$	$\uparrow$	$\times$	$\times$	$\uparrow$

- Elijo a_1
- Puedo descartar a_2, a_3, a_5, a_{10}
- Elijo a_4
- Puedo descartar a_6, a_7
- Elijo a_8
- Puedo descartar a_9
- Elijo a_{11}
- Listo!

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Demostración

- *Sea A_j un conjunto de tareas máximo para S_j.*

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Demostración

- *Sea A_j un conjunto de tareas máximo para S_j.*
- *Ordenemos A_j según el tiempo de finalización (creciente)*

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j *un subproblema,* $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m *pertenece a una solución óptima de* S_j

Demostración

- *Sea* A_j *un conjunto de tareas máximo para* S_j.
- *Ordenemos* A_j *según el tiempo de finalización (creciente)*
- *Sea* a_i *la primer tarea de* A_j

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Demostración

- *Sea A_j un conjunto de tareas máximo para S_j.*
- *Ordenemos A_j según el tiempo de finalización (creciente)*
- *Sea a_i la primer tarea de A_j*
- *Si $a_m = a_i$, listo!*

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Demostración

- *Sea A_j un conjunto de tareas máximo para S_j.*
- *Ordenemos A_j según el tiempo de finalización (creciente)*
- *Sea a_i la primer tarea de A_j*
- *Si $a_m = a_i$, listo!*
- *Si no, sea $A'_j = A_j \setminus \{a_i\} \cup \{a_m\}$.*

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j *un subproblema,* $a_m = \min\{f_k \mid a_k \in S_j\}$*.*
Entonces a_m *pertenece a una solución óptima de* S_j

Demostración

- *Sea* A_j *un conjunto de tareas máximo para* S_j*.*
- *Ordenemos* A_j *según el tiempo de finalización (creciente)*
- *Sea* a_i *la primer tarea de* A_j
- *Si* $a_m = a_i$*, listo!*
- *Si no, sea* $A'_j = A_j \setminus \{a_i\} \cup \{a_m\}$*.*
- A'_j *tiene el mismo número de tareas que* A_j*, y como* $f_m \leq f_i$*, las tareas en* A'_j *son disjuntas*

Verificando que la estrategia es óptima (I)

Teorema 1 (Seguridad de la elección greedy)

Sean S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$.
Entonces a_m pertenece a una solución óptima de S_j

Demostración

- *Sea A_j un conjunto de tareas máximo para S_j.*
- *Ordenemos A_j según el tiempo de finalización (creciente)*
- *Sea a_i la primer tarea de A_j*
- *Si $a_m = a_i$, listo!*
- *Si no, sea $A'_j = A_j \setminus \{a_i\} \cup \{a_m\}$.*
- *A'_j tiene el mismo número de tareas que A_j, y como $f_m \leq f_i$, las tareas en A'_j son disjuntas*
- *Por lo tanto, A'_j es óptima para S_j, y $a_m \in A'_j$*

Verificando que la estrategia es óptima (II)

Teorema 2 (Subestructura óptima)

Sea S_j *un subproblema,* $a_m = \min\{f_k \mid a_k \in S_j\}$,
$S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ *el subproblema resultante al elegir* a_m, *y*
A_m *una solución óptima para* S_m.
Entonces $A_j = \{a_m\} \cup A_m$ *es una solución óptima para* S_j

Teorema 2 (Subestructura óptima)

Sea S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$,
$S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ el subproblema resultante al elegir a_m, y A_m una solución óptima para S_m.
Entonces $A_j = \{a_m\} \cup A_m$ es una solución óptima para S_j

Demostración (Por el absurdo)

Verificando que la estrategia es óptima (II)

Teorema 2 (Subestructura óptima)

Sea S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$, $S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ el subproblema resultante al elegir a_m, y A_m una solución óptima para S_m.
Entonces $A_j = \{a_m\} \cup A_m$ es una solución óptima para S_j

Demostración (Por el absurdo)

- *Si no lo es, exite B_j que es solución de S_j y $|A_j| < |B_j|$*

Verificando que la estrategia es óptima (II)

Teorema 2 (Subestructura óptima)

*Sea S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$,
$S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ el subproblema resultante al elegir a_m, y
A_m una solución óptima para S_m.
Entonces $A_j = \{a_m\} \cup A_m$ es una solución óptima para S_j*

Demostración (Por el absurdo)

- *Si no lo es, exite B_j que es solución de S_j y $|A_j| < |B_j|$*
- *Tomamos $B_j' = B_j \setminus \{a_k\}$, donde a_k es la primer tarea en terminar de B_j*

Verificando que la estrategia es óptima (II)

Teorema 2 (Subestructura óptima)

Sea S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$,
$S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ el subproblema resultante al elegir a_m, y A_m una solución óptima para S_m.
Entonces $A_j = \{a_m\} \cup A_m$ es una solución óptima para S_j

Demostración (Por el absurdo)

- *Si no lo es, exite B_j que es solución de S_j y $|A_j| < |B_j|$*
- *Tomamos $B_j' = B_j \setminus \{a_k\}$, donde a_k es la primer tarea en terminar de B_j*
- *Se puede ver que B_j' es solución de S_m (porque a_k termina después que a_m)*

Verificando que la estrategia es óptima (II)

Teorema 2 (Subestructura óptima)

Sea S_j un subproblema, $a_m = \min\{f_k \mid a_k \in S_j\}$,
$S_m = \{a_k \in S_j \mid s_k \geq f_m\}$ el subproblema resultante al elegir a_m, y A_m una solución óptima para S_m.
Entonces $A_j = \{a_m\} \cup A_m$ es una solución óptima para S_j

Demostración (Por el absurdo)

- *Si no lo es, exite B_j que es solución de S_j y $|A_j| < |B_j|$*
- *Tomamos $B'_j = B_j \setminus \{a_k\}$, donde a_k es la primer tarea en terminar de B_j*
- *Se puede ver que B'_j es solución de S_m (porque a_k termina después que a_m)*
- *Como $|A_m| < |B'_j|$, A_m no es óptima para S_m.* **Absurdo!**

```
TASK–SELECTOR(s, f, j, n)
    m ← j + 1
    while m ≤ n ∧ s_m < f_j do    ▷ Buscamos el primer
        m ← m + 1                 ▷ elemento en S_j
    if m ≤ n then
        return {a_m} ∪ TASK–SELECTOR(s, f, m, n)
    else
        return ∅
```

No todas las estrategias greedy llevan al óptimo

- Para el caso de las tareas, consideremos elegir la tarea de menor duración
- Nos quedan (posiblemente) dos subproblemas, pero al unir las soluciones de éstos con la tarea elegida no necesariamente obtenemos el óptimo
- Es fácil encontrar un contraejemplo...

Sobre la subestructura óptima

- Decimos que un problema tiene subestructura óptima si una solución óptima del problema contiene soluciones óptimas de sus subproblemas

Sobre la subestructura óptima

- Decimos que un problema tiene subestructura óptima si una solución óptima del problema contiene soluciones óptimas de sus subproblemas
- Esta característica es lo que nos permite hacer inducción al demostrar que el algoritmo greedy es óptimo (y recursión al resolver el problema)

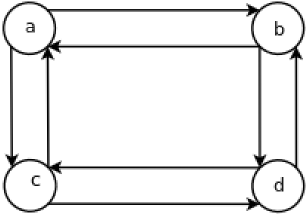
Sobre la subestructura óptima

- Decimos que un problema tiene subestructura óptima si una solución óptima del problema contiene soluciones óptimas de sus subproblemas
- Esta característica es lo que nos permite hacer inducción al demostrar que el algoritmo greedy es óptimo (y recursión al resolver el problema)
- No todos los problemas que exhiben subestructura óptima pueden resolverse mediante un algortimo greedy (quizás podamos aplicar programación dinámica)

No todos los problemas exhiben subestructura óptima

Veamos un par de problemas...

1. Camino de longitud mínima en un grafo sin pesos
2. Camino simple de longitud máxima en un grafo sin pesos



Otro enfoque...

- Veamos otra forma de encarar el problema de optimalidad de un algoritmo greedy

Otro enfoque...

- Veamos otra forma de encarar el problema de optimalidad de un algoritmo greedy
- Utilizaremos una estructura algebraica llamada matroide, que caracteriza (algunos de los) problemas para los cuales los algoritmos greedy encuentran el óptimo

Otro enfoque...

- Veamos otra forma de encarar el problema de optimalidad de un algoritmo greedy
- Utilizaremos una estructura algebraica llamada matroide, que caracteriza (algunos de los) problemas para los cuales los algoritmos greedy encuentran el óptimo
- Escribiremos un algoritmo greedy para matroides, y demostraremos que es óptimo

Otro enfoque...

- Veamos otra forma de encarar el problema de optimalidad de un algoritmo greedy
- Utilizaremos una estructura algebraica llamada matroide, que caracteriza (algunos de los) problemas para los cuales los algoritmos greedy encuentran el óptimo
- Escribiremos un algoritmo greedy para matroides, y demostraremos que es óptimo
- Por lo tanto, dado un problema, si logramos representarlo como un matroide
    1. sabemos que se puede seguir una estrategia greedy

Otro enfoque...

- Veamos otra forma de encarar el problema de optimalidad de un algoritmo greedy
- Utilizaremos una estructura algebraica llamada matroide, que caracteriza (algunos de los) problemas para los cuales los algoritmos greedy encuentran el óptimo
- Escribiremos un algoritmo greedy para matroides, y demostraremos que es óptimo
- Por lo tanto, dado un problema, si logramos representarlo como un matroide
  1. sabemos que se puede seguir una estrategia greedy
  2. ya tenemos escrito el algoritmo que encuentra el óptimo

Definición 1 (Matroide)

Un **matroide** *M es un par (S, I) tal que*

1. *S es un conjunto finito*
2. *$I \subseteq \mathbb{P}(S)$ es una familia de subconjuntos de S, llamados* **conjuntos independientes**, *tales que, si $B \in I$ y $A \subseteq B$, entonces $A \in I$. A esta propiedad la llamamos* **propiedad hereditaria**.
3. *Si $A \in I$, $B \in I$, y $|A| < |B|$, entonces existe algún elemento $x \in B \setminus A$ tal que $A \cup \{x\} \in I$. A esta propiedad la llamamos* **propiedad de intercambio**

Ejemplos de matroides

Ejemplo 1 (Grafos)

- *S: conjunto de arcos de un grafo*
- *I: conjuntos de arcos acíclicos*

Ejemplo 2 (Matrices)

- *S: conjunto de filas de una matriz de tamaño $m \times n$*
- *I: conjunto de filas linealmente independientes*

Ejemplo 3 (Subconjuntos)

- *S: cualquier conjunto finito*
- *I: conjuntos con a lo sumo k elementos, para algún $k \leq |S|$*

Ejemplos de cosas que no son matroides

Ejemplo 4 (Matching)

- *S: conjunto de arcos de un grafo*
- *I: conjuntos de arcos que no comparten vértices*

Ejemplo 5 (Mochila 0-1 de capacidad W)

- $S = \{w_1, w_2, \ldots, w_n\}$
- *I: conjuntos $\{w_{i_1}, \ldots, w_{i_m}\}$ tales que $w_{i_1} + \cdots + w_{i_m} \leq W$*

Elementos maximales en matroides

Definición 2

Sean $M = (S, I)$ un matroide, $A \in I$.

- *Un elemento $x \notin A$ es una* **extensión** *de A si x puede agregarse a A preservando independencia; es decir, si $A \cup \{x\} \in I$*
- *$A \in I$ es* **maximal** *si no tiene extensiones. Es decir, si no existe $B \in I$ tal que $A \subset B$*

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Demostración (por el absurdo)

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Demostración (por el absurdo)

- *Supongamos que no.*

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Demostración (por el absurdo)

- *Supongamos que no.*
- *Entonces existen $A, B \in I$ maximales y tales que $|A| < |B|$*

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Demostración (por el absurdo)

- *Supongamos que no.*
- *Entonces existen* $A, B \in I$ *maximales y tales que* $|A| < |B|$
- *Por la propiedad de intercambio, existe* $x \in B \setminus A$ *tal que* $A \cup \{x\} \in I$

Elementos maximales en matroides (II)

Teorema 3

Todos los elementos maximales de un matroide tienen el mismo cardinal

Demostración (por el absurdo)

- *Supongamos que no.*
- *Entonces existen* $A, B \in I$ *maximales y tales que* $|A| < |B|$
- *Por la propiedad de intercambio, existe* $x \in B \setminus A$ *tal que* $A \cup \{x\} \in I$
- A *es extensible. Absurdo, pues es maximal!*

Definición 3 (Función peso en matroides)

Sea $M = (S, I)$. Una función $w : S \to \mathbb{R}^+$ es una función de pesos para M. Para $A \in I$, el peso de A es

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

- Un problema de optimización en un matriode se reduce a buscar un conjunto independiente $A \in I$ tal que $w(A)$ es máximo.
- Tal A siempre será maximal, dado que los pesos son positivos

```
GREEDY((S, I), w)
  A ← ∅
  Ordenamos S en forma decreciente por pesos
  para cada x ∈ S, tomados en orden decreciente de pesos
    if A ∪ {x} ∈ I then
      A ← A ∪ {x}
  return A
```

Un algortimo greedy para matroides

```
GREEDY((S, I), w)
  A ← ∅
  Ordenamos S en forma decreciente por pesos
  para cada x ∈ S, tomados en orden decreciente de pesos
    if A ∪ {x} ∈ I then
      A ← A ∪ {x}
  return A
```

- Dado que $\emptyset$ es independiente, y que cada x se agrega sólo si preserva independencia, el conjunto A devuelto es independiente

Un algortimo greedy para matroides

```
GREEDY((S, I), w)
  A ← ∅
  Ordenamos S en forma decreciente por pesos
  para cada x ∈ S, tomados en orden decreciente de pesos
    if A ∪ {x} ∈ I then
      A ← A ∪ {x}
  return A
```

- Dado que $\emptyset$ es independiente, y que cada x se agrega sólo si preserva independencia, el conjunto A devuelto es independiente
- Si $n = |S|$, ordenar nos lleva $O(n \lg n)$. La complejidad de GREEDY es entonces

$$O(n \lg n + nf(n))$$

donde $f(n)$ es el tiempo requerido para verificar la condición de extensión

Lema 4 (greedy-choice property)

Sea $M = (S, I)$ un matroide con función de pesos w, tal que S está ordenado de forma decreciente.
Si existe un elemento x tal que $\{x\} \in I$ y $w(x)$ es máximo, entonces existe un conjunto óptimo A de S que contiene a x.

Lema 5 (Subestructura óptima)

Sea x el primer elemento elegido por GREEDY. El problema de encontrar un subconjunto independiente de peso máximo que contiene a x (que existe, por el teorema anterior), se reduce a buscar un subconjunto independiente máximo en el matroide

- $S' = \{y \in S \mid \{x, y\} \in I\}$
- $I' = \{B \subseteq S \setminus \{x\} \mid B \cup \{x\} \in I\}$

Correctitud de GREEDY

Teorema 6

El algoritmo GREEDY para matroides devuelve un subconjunto óptimo de S.

Demostración

Usando los lemas anteriores

Ejemplo 6 (Algoritmo de Kruskal)

Sea $G = (V, E)$ *un grafo con costos* $c : E \to \mathbb{R}^+$.
Definimos $M = (S, I)$ *y* w *donde*

- $S = E$
- I *son todos los conjunto de aristas acíclicos*
- $w(e) = w_0 - c(e)$, *donde* $w_0 > \max\{c(e) \mid e \in E\}$
- *Ordenar las aristas ya sabemos, lo hacemos en tiempo* $O(|E| \lg |E|)$
- *Para ver si* $A \cup \{x\}$ *es independiente (no forma un ciclo), puedo ir guardando las componentes conexas de* A, *resultando* $f \in O(|V|)$ *(con una estructura que veremos más adelante)*
- *Por lo tanto, GREEDY encuentra un árbol recubridor minimal en tiempo* $O(|E| \lg |E|)$

Conclusiones

- Hemos encontrado una estructura algebraica que subyace a una técnica de diseño de algoritmos
- Pudimos probar propiedades generales sobre esta técnica
- Escribimos un algoritmo general y calculamos su complejidad
- Dado un problema, si lo representamos como un problema de optimización en un matroide, tenemos todo resuelto

Conclusiones

- Hemos encontrado una estructura algebraica que subyace a una técnica de diseño de algoritmos
- Pudimos probar propiedades generales sobre esta técnica
- Escribimos un algoritmo general y calculamos su complejidad
- Dado un problema, si lo representamos como un problema de optimización en un matroide, tenemos todo resuelto

Hay mucho más sobre álgebras y algoritmos...