MA3705. Algoritmos Combinatoriales. 2014.
Profesor: José Soto
Escriba(s): Manuel Cáceres, Camilo Gómez y Sebastián Muñoz.
Fecha: 11 de Agosto 2014 .

Cátedra 5

1. Recuerdo del algoritmo de KRUSKAL

La clase anterior se describió el algoritmo de Prim, el cual a través de la formación de una única componente conexa, encuentra un árbol cobertor de peso mínimo de un grafo conexo. Se describieron además diferentes implementaciones y se analizaron las complejidades de las mismas.

Por último, se describió una forma alternativa de enfrentar el problema del árbol cobertor de peso mínimo que consiste en ir agregando en cada iteración la arista de menor coste mientras esta no forme un ciclo[1].Esta forma corresponde al algoritmo de Kruskal, que se recuerda a continuación :

Algoritmo 1: Algoritmo de Kruskal

Entrada: $G = (V, E)$ conexo
$T \leftarrow \varnothing$
while T *no es generador* **do**
  Sea $e \in E \setminus T$ de peso mínimo, tal que $T + e$ no tiene ciclos
  $T \leftarrow T + e$
end
return (V, T)

1.1. Correctitud del algoritmo

Veamos primero que Kruskal encuentra un árbol cubridor.

Por un lado, es simple notar que el resultado del algoritmo corresponde a un grafo acícliclo, dado que al agregar aristas al resultado se exige que estas no formen ciclos. Por otro lado, supongamos por contradicción que existen dos vértices u y v que no están contectados a través de T, no obstante como G es conexo, existe un camino entre u y v. De este camino consideremos la primera arista e que tiene como incidente un vértice x que no está en la misma componente conexa de u. Con respecto a esta arista (dado que fue procesada por el algoritmo en su momento) se puede razonar que $T + e$ forma un ciclo, pero esto es contradictorio con el hecho de que u y v no están conectado a través de T.

Veamos ahora que el árbol corresponde a uno de peso mínimo.

Para esto tomemos T el árbol cubridor entregado por el algoritmo de Kruskal y T' algún otro árbol cubridor del grafo G, consideremos que sus aristas son $\{e_1, \ldots, e_{n-1}\}$ y $\{e'_1, \ldots, e'_{n-1}\}$ respectivamente y que están ordenadas en orden creciente según sus pesos.
Como se quiere mostrar que T es de peso mínimo, supongamos por contradicción que existe un par e_i, e'_i, tales que el peso del primero es estrictamente mayor que el peso del segundo y supongamos sin pérdida de generalidad que este i es el mínimo índice que cumple la condición anterior.

Consideremos por último los conjuntos de aristas $E' = \{e_1, \ldots, e_{i-1}\}$, $E'' = \{e'_1, \ldots, e'_i\}$ y los conjuntos de vértices $V_1 \ldots V_s$ correspondientes a las componentes conexas del grafo (V, E').

Veamos ahora que dado que las s componentes conexas cumplen que

$$|E' \cap \binom{V_j}{2}| \geq |V_j| - 1$$

[1]Es importante notar que ha diferencia del algoritmo de Prim, Kruskal no mantiene una gran componente conexa durante su aplicación.

y por lo tanto se tiene que $|E'| \geq n - s$. Por otro lado, como E'' no tiene ciclos, se tiene que :

$$|E'' \cap \binom{V_j}{2}| \leq |V_j| - 1$$

y por lo tanto las aristas de E'' contenidas en las componentes V_j son a lo mas $n - s$, y finalmente recordando que $|E'| = |E''| - 1$, podemos afirmar que existe por lo menos una arista $e \in E''$ tal que conecta dos componentes conexas de E'. Finalmente como las aristas de E" están enumeradas en orden creciente de pesos, el peso de e debe ser menor o igual que el peso de e_i, pero, recordando nuestra hipótesis de contradicción se tiene que el peso de e_i es estrictamente mayor que el de e'_i y por lo tanto, mayor que el de e.
Resumiendo lo anterior, tenemos una arista e que conecta componentes conexas de $E' = \{e_1, \ldots, e_{i-1}\}$(y por lo tanto no formaría ciclo el agregarla) y tal que su peso es estrictamente menor que el de e_i, pero el algoritmo opta por e_i, lo que es una contradicción.[2]

1.2. Análisis de complejidad

El análisis de complejidad de este algoritmo se hará en base una implementación mejor en términos de complejidad descrita a continuación :

Algoritmo 2: Algoritmo de Kruskal(segunda implementación)

```
Entrada: G = (V, E)
Ordenar E = {e_1, ..., e_m} de menor a mayor peso
T ← ∅
for i = 1 ... m do
    if T + e_i es acíclico then
        T ← T + e_i
    end
end
return (V, T)
```

Se nota que esta implementación factoriza el trabajo de estar buscando cada vez la arista de menor peso, y en vez de eso primero ordena las arista en orden ascendente lo que toma $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$, para luego hacer $\mathcal{O}(n)$ repeticiones en las que comprueba si al añadir la siguiente arista se forma algún ciclo.

El trabajo de verificar si al agregar una arista se forma algún ciclo, se puede hacer comprobando si los vértices incidentes a la arista a agregar se encuentran en la misma componente conexa, si es así, la arista forma un ciclo y esta no debe ser agregada. Este trabajo se puede realizar con alguno de los algoritmos de búsqueda en grafos vistos anteriormente (BFS o DFS modificados para tal fin) y por lo tanto toma $\mathcal{O}(|V| + |T|) = \mathcal{O}(n)$.

Finalmente se concluye que esta implementación de Kruskal toma $\mathcal{O}(m \log n + n^2)$.

1.3. Observaciones

- El problema de averiguar cuando dos vértices pertenecen a la misma componente conexa de manera eficiente es conocido como "Union-Find". Las mejores implementaciones de "Union-Find" se han hecho en $\mathcal{O}(\alpha(n))$, donde α corresponde a la inversa de la función de Ackermann, la cual crece extremadamente lento, esta es incluso $\mathcal{O}(\log^*(n))$[3]. Por lo tanto la mejor implementación de Kruskal tarda $\mathcal{O}(m\alpha(n))$.
- El algoritmo de Kruskal se puede aplicar en un grafo no conexo, en el cual retorna un bosque generador de peso mínimo, conservando las complejidades antes mencionadas.

[2] Veremos una demostración alternativa de la correctitud de Kruskal cuando veamos algoritmos glotones en matroides.
[3] $\log^*(n)$ corresponde al número de veces que hay que aplicar logarítmo a n para que el resultado sea 1.

2. Algoritmos Glotones

2.1. Introducción a Algoritmos Glotones

El algoritmo de Kruskal pertenece a una familia de algoritmos llamados algoritmos glotones o codiciosos. En general, este tipo de algoritmos siguen una estrategia que intentan ganar tanto como sea posible en cada momento y no contempla las desventajas que estas elecciones podrían acarrear en el futuro.
Para el estudio de este tipo de algoritmos se presentarán algunas definiciones y luego se introducirá el concepto de "Matroides".

Definición 1 (Par Hereditario)**.** Sea E un conjunto finito e $\mathcal{I} \subseteq 2^E$. El par ordenado $(E, \mathcal{I})$ se dice hereditario si:

- $\mathcal{I} \neq \varnothing$.
- Si $J \in \mathcal{I}$ e $I \subseteq J \implies I \in \mathcal{I}$.

Observación 1. *Los elementos (conjuntos) de $\mathcal{I}$ se llaman conjuntos independientes de $(E, \mathcal{I})$.*

Ejemplo 1 (Pares Hereditarios)**.**

(i) Sea E un conjunto finito cualquiera, si consideramos $\mathcal{I} = 2^E$, se tiene que $(E, \mathcal{I})$ es hereditario.

(ii) Sea $G = (V, E)$ un grafo, si consideramos $\mathcal{I} = \{F \subseteq E : \ F$ acíclico$\}$, se sique que $(E, \mathcal{I})$ es hereditario.

(iii) Sea E conj. finito, entonces $(E, \mathcal{I}) = (E, \{X \subseteq E : |X| \leq 7\})$ es par hereditario.

(iv) Dado $w : E \to \mathbb{R}^+$, $(E, \{X \subseteq E : w(X) \leq c\})$ es par hereditario, donde $c \in \mathbb{R}$.

Definición 2 (Bases)**.** Sea $(E, \mathcal{I})$ un par hereditario. Los conjuntos independientes de $(E, \mathcal{I})$ que son maximales para la inclusión se llaman Bases.

Ejemplo 2 (Importante)**.** *Sea el siguiente par hereditario $(E, \mathcal{I})$, donde E es un conjunto de vectores de $\mathbb{R}^n$ ($\mathbb{F}^n$, con $\mathbb{F}$ cuerpo[4]) e $\mathcal{I}$ contiene los conjuntos de E que son linealmente independiente.*

Veamos el siguiente ejemplo particular en $\mathbb{R}^2$:
Si consideramos el siguiente conjunto de vectores en $\mathbb{R}^2$, $E = \left\{ \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}0\\0\end{pmatrix} \right\}$,
entonces $\mathcal{I} = \left\{ \left\{ \begin{pmatrix}1\\0\end{pmatrix} \right\}, \left\{ \begin{pmatrix}0\\1\end{pmatrix} \right\}, \left\{ \begin{pmatrix}1\\1\end{pmatrix} \right\}, \left\{ \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix} \right\}, \left\{ \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix} \right\}, \left\{ \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix} \right\} \right\}$.

2.2. Resolución de Problemas

Dados $(E, \mathcal{I})$ hereditario, $w : E \to \mathbb{R}$, queremos estudiar los siguientes problemas:

1. Encontrar un conjunto independiente de cardinalidad máxima.
2. Encontrar conjunto independiente de peso máximo.
3. Encontrar una base de peso máximo.

Para cada uno de estos problemas propondremos un algoritmo glotón que lo resuelve bajo ciertas condiciones.

[4] En general, se puede en cualquier cuerpo $\mathbb{F}$.

2.2.1. Problema de Cardinalidad

Para el primer problema se propondrá un algoritmo que va agregando elementos $e \in E$ a un conjunto I, en la medida que I continúe siendo independiente. Esta idea se concretiza en el algoritmo **Glotón-Cardinalidad**.

Algoritmo 3: Glotón-Cardinalidad

Entrada: $(E = \{e_1, ..., e_m\}, \mathcal{I})$ hereditario.
$I \leftarrow \varnothing$
for $i = 1 \dots m$ **do**
  if $I + e_i \in \mathcal{I}$ **then**
    $I \leftarrow I + e_i$
  end
end
return I

Proposición 1. *Si todas las bases tienen el mismo cardinal, el algoritmo* ***Glotón-Cardinalidad*** *resuelve el problema de cardinalidad.*

Sin embargo, como veremos más adelante, esto no es cierto para el algoritmo **Glotón-Base(max)**.

2.2.2. Problema del Independiente de peso Máximo

Para resolver este problema la idea es ir agregando en cada paso elementos de peso "positivo" que tengan el peso máximo entre los elementos candidatos a ser agregados.

Algoritmo 4: Glotón-Independiente(max)

Entrada: $(E, \mathcal{I})$ hereditario, $w : E \to \mathbb{R}$.
Tomar $E^+ \subseteq E$, los objetos de peso positivo de E.
Ordenar $E^+ = \{e_1, ..., e_m\}$ de mayor a menor peso.
$I \leftarrow \varnothing$
for $i = 1 : m$ **do**
  if $I + e_i \in \mathcal{I}$ **then**
    $I \leftarrow I + e_i$
  end
end
return I

2.2.3. Problema de la Base de peso Máximo

Para este problema, la idea es análoga a la del algoritmo **Glotón-Independiente(max)**, pero considerando todos los elementos de E (los que tienen peso positivo como los que no).

Algoritmo 5: Glotón-Base(max)

Entrada: $G = (E, \mathcal{I})$ hereditario, $w : E \to \mathbb{R}$.
Ordenar $E = \{e_1, ..., e_m\}$ de mayor a menor peso.
$I \leftarrow \varnothing$
for $i = 1 \dots m$ **do**
  if $I + e_i \in \mathcal{I}$ **then**
    $I \leftarrow I + e_i$
  end
end
return I

Para estos dos algoritmos se podría pensar que una condición suficiente para que resuelvan el problema es que las bases tengan igual cardinal, sin embargo, el siguiente ejemplo ilustra una complicación adicional.

Ejemplo 3 (Falla algoritmo **Glotón-Base(max)**). *Notemos que las bases tengan igual cardinal, se cumple en la figura 1. Además* $(A \cup B, \mathcal{I})$ *hereditario para* $\mathcal{I} = 2^A \cup 2^B$. *Glotón devuelve* $\{a, b\}$ *de peso 3, pero la mejor base es* $\{c, d\}$ *de peso 4. Observemos que* $(A \cup B, \mathcal{I})$ *tiene solo 2 bases.*

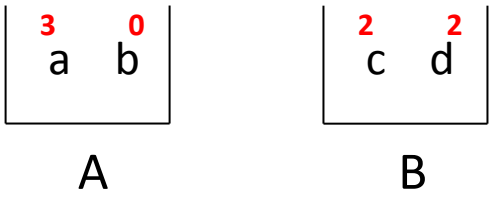

Figura 1

Antes de dar la condición suficiente para que ***Glotón-Base(max)*** *funcione, veamos la siguiente definición:*

Definición 3 (Base de un conjunto). Sea $(E, \mathcal{I})$ un par hereditario. Dado $X \subseteq E$,
decimos que B es base de X ssi $B \in \mathcal{I} \wedge (\forall Y \in \mathcal{I})\ (B \subseteq Y \wedge Y \subseteq X \implies B = Y)$

Luego una condición suficiente para que el algoritmo ***Glotón-Base(max)*** *funcione es:*

$$\forall X \subseteq E \text{ las bases de } X \text{ tienen el mismo cardinal.}$$

Para establecer condiciones para que estos algoritmos funcionen, se introducirá el concepto de **"Matroide"**.

2.3. Matroides

Definición 4 (Matroide). Es un par hereditario $M = (E, \mathcal{I})$ que satisface el siguiente axioma:

(Aumento) Si $I, J \in \mathcal{I}$ con $|I| < |J|$, entonces existe $e \in J \setminus I$ tal que $I + e \in \mathcal{I}$.

Lema 1 (Equivalencias con el axioma de **(Aumento)**). *El axioma de* ***(Aumento)*** *es equivalente a cualquiera de los siguientes axiomas:*

- ***(Aumento débil)*** *Si* $I, J \in \mathcal{I}$, $|J \setminus I| = 2$, $|I \setminus J| = 1$, *entonces existe* $e \in J \setminus I$ *tal que* $I + e \in \mathcal{I}$.
- ***(Cardinalidad de las Bases)*** $\forall F \subseteq E$ *las bases de* F *tienen el mismo cardinal.*

Demostración (Pendiente) ■

2.4. Ejemplos de Matroides

2.4.1. Matroides Lineales/Representables

Sea E un conjunto de vectores en un espacio vectorial ($\mathbb{F}^n$, con $\mathbb{F}$ un cuerpo).

$$X \in \mathcal{I} \iff X \text{ es linealmente independiente.}$$

Típicamente las matroides lineales se representan mediante una matriz A cuyas columnas son los vectores de E. Decimos en ese caso que la matroide está representada por A, o que es representable en el cuerpo $\mathbb{F}$.

2.4.2. Matroides Gráficas

$M = (E, \mathcal{I})$[5] donde existe un grafo $G = (V, E)$ tal que $F \in \mathcal{I} \iff F$ no tiene ciclos en G.

[5] Se dice que $M = (E, \mathcal{I})$ es una Matroide Gráfica.

Observación 2. *Grafos distintos pueden tener igual matroide gráfica, como en el siguiente ejemplo:*

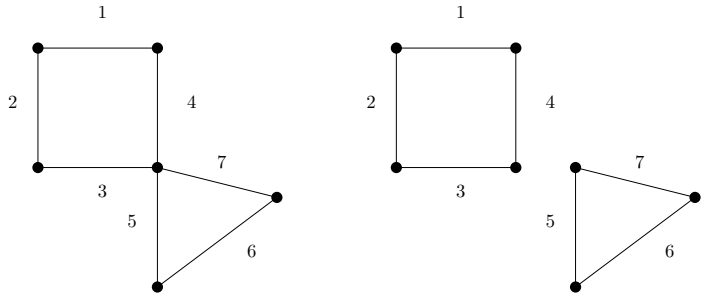

Figura 2

2.4.3. Matroide de Partición

Sea $E = \bigcup_{i=1}^{k} E_i$ donde $\{E_i\}_{i=1}^{k}$ es disjunta a pares, tal que $|E_i| \geq c_i \in \mathbb{N}$, $\forall\, i = 1, \ldots, k$.
La Matroide de Partición asociada $(E, \mathcal{I})$ es tal que

$$X \in \mathcal{I} \iff \forall\, i \in \{1, \ldots, k\} : |X \cap E_i| \leq c_i$$

Verifiquemos que las Matroides de Partición son matroides.
Demostración Sea $(E, \mathcal{I})$ una matroide de partición, tal que $E = \bigcup_{i=1}^{k} E_i$ disjunta. Sean $I, J \in \mathcal{I}$, $|I| < |J|$. Como $(E, \mathcal{I})$ es matroide de partición, $\forall\, i \in \{1, \ldots, k\} : |I \cap E_i| \leq c_i \wedge |J \cap E_i| \leq c_i$. Dado que $|J| > |I|$, se tiene que existe $j \in \{1, \ldots, k\}$ tal que $c_j \geq |J \cap E_j| > |I \cap E_j|$. Luego, existe $x \in (J \cap E_j) \setminus I$, luego $I + x$ es independiente. Por lo tanto, $(E, \mathcal{I})$ satisface el axioma de *Aumento*, es decir, es Matroide. ■

Teorema Algoritmo Glotón-Base

Teorema 1. *Si $M = (E, \mathcal{I})$ es matroide, entonces el algoritmo **Glotón-Base(max)** devuelve una base de peso máximo.*

Demostración

Sea S el conjunto devuelto por Glotón-Base y sea T una base de peso máximo. Notemos que S es una base (esto es consecuencia de que M en particular es hereditario), luego por **axioma de *Cardinalidad de la Bases***, se sique que S y T por ser bases, tienen el mismo cardinal, digamos $|S| = |T| = l$. Sean

$$S = \{s_1, s_2, ..., s_l\},$$
$$T = \{t_1, t_2, ..., t_l\},$$

ordenados de mayor a menor peso, es decir, $w(t_1) \geq w(t_2) \geq \ldots \geq w(t_l)$, lo mismo para S.

Queremos demostrar que S es base de peso máximo, procedamos por contradicción, supongamos que S no tiene peso máximo, es decir, $w(T) > w(S)$. Luego $\exists\, k \in \{1, \ldots, l\}$ tal que $w(t_k) > w(s_k)$, dado que en particular existe al menos un índice con esta propiedad, lo podemos tomar como el más pequeño. Sea S' el conjunto S en el momento en que el algoritmo revisa a t_k. Notemos que el algoritmo revisa a t_k antes que s_k. Sean $S' = \{s_1, s_2, \ldots, s_{k-1}\}$ y $T' = \{t_1, t_2, \ldots, t_k\}$.

Dado que $S', T' \in \mathcal{I}$ (pues, $\mathcal{I}$ es hereditario), se tiene por **axioma de *Aumento***, como $|S'| < |T'|$, $\exists\, t_i \in T' \setminus S'$ tal que $S' + t_i \in \mathcal{I}$.

Además, como $t_i \in T', w(t_i) \geq w(t_k) > w(s_k)$. Luego, el algoritmo revisó a t_i antes que s_k, en un momento en que tiene a algún $S'' \subseteq S'$. Como $S'' + t_i \subseteq S' + t_i$ y $S' + t_i \in \mathcal{I}$, se sigue que $S'' + t_i \in \mathcal{I}$, por lo que el algoritmo debió haber agregado a t_i, lo cual no ocurrió, ya que $t_i \notin S'$ (lo cual nos lleva a una contradicción). ■