Algoritmos Voraces

Problema 4 - Recubrimiento de grafos

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

13 de mayo de 2015

Introducción

- En esta presentación se proporciona una solución para el ejercicio 4.
- El código, los resultados de las ejecuciones, las gráficas y los pdf asociados se puede encontrar en GitHub.

Explicación del problema

Definición de recubrimiento

Consideremos un grafo no dirigido $G = (V, E)$. Un conjunto U se dice que es un recubrimiento de G si $U \subset V$ y cada arista en E incide en, al menos, un vértice o nodo de U ,es decir,

$$\forall (x, y) \in E : x \in U \text{ o } y \in U$$

Explicación del problema

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Definición de recubrimiento minimal

Un conjunto U, recubrimiento de G, se dice minimal si cuenta con el menor número de nodos posibles.

Explicación del problema

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Enunciado del problema

Encontrar un algoritmo que calcule el recubrimiento minimal de cualquier grafo arbitrario.

Explicación del problema

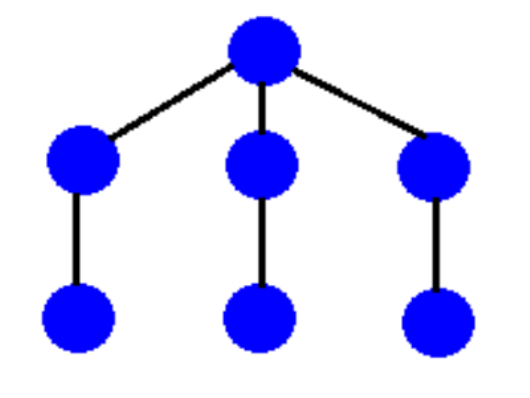
A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Organización de la exposición

- Algoritmos voraces (no óptimos) para grafos arbitrarios.
- Algoritmo voraces (óptimo) para árboles.
- Análisis empírico de los algoritmos.

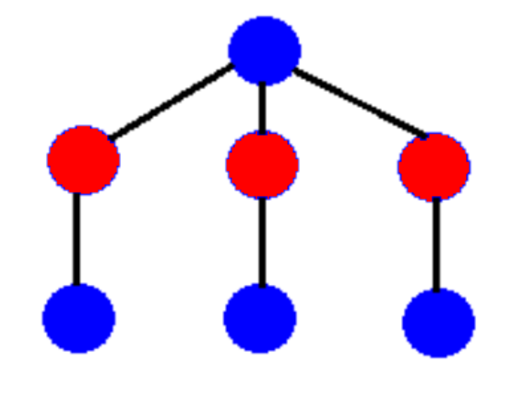
Explicación del problema

Grafo sobre el que ejemplificaremos los algoritmos



Explicación del problema

Solución al problema para este grafo



Algoritmos voraces (no óptimos) para grafos

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

- Múltiples intentos de aloritmos voraces sin éxito.
  - Algoritmo aleatorio
  - Algoritmo voraz aleatorizado
  - Algoritmo voraz basado en grados
- **El problema en su versión de decisión es NP-Completo.**

Algoritmos voraces (no óptimos) para grafos

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Solución trivial: $U = G$

Algoritmo aleatorio

1. $U = \emptyset$
2. Para cada arista $(x, y) \in E$ elegimos un nodo aleatoriamente entre x e y y lo añadimos a U.

Algoritmos voraces (no óptimos) para grafos

Algoritmo aleatorio

- **Eficiencia:** $\theta(|E|)$
- **Problema:** Añadimos para cada arista uno de los nodos, sin tener en cuenta si uno de ellos ya está en U.
- **Solución:** Añadir una estrategia voraz.

Algoritmos voraces (no óptimos) para grafos

Algoritmo voraz aleatorizado

1. $U = \emptyset$
2. Para cada arista $(x, y) \in E$ tomamos un nodo v y lo añadimos a U donde v es:

- x si $x \in U$.
- y si $y \in U$.
- Uno de los dos, elegido aleatoriamente, si $x, y \in U$.

Algoritmos voraces (no óptimos) para grafos

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

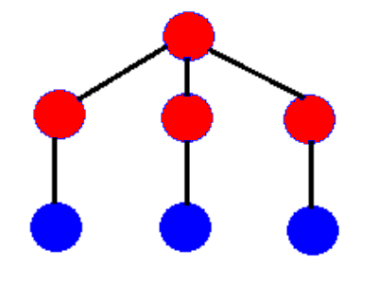
Algoritmo voraz aleatorizado

- Algoritmo aleatorio + Estrategia voraz
- **Eficiencia:** $\theta(|E|)$ usando tablas de Hash.

Algoritmos voraces (no óptimos) para grafos

Algoritmo voraz aleatorizado

- No es óptimo (aunque puede que conseguirlo en alguna iteración):



Algoritmos voraces (no óptimos) para grafos

Definición

El **grado** de un nodo del grafo G es el número de aristas que inciden sobre él.

Algoritmo voraz basado en grados

- **Idea:** Los nodos con grado pequeño son malos.
- **Algoritmo:**

1. Añadir el nodo de mayor grado a U.
2. Eliminarlo de V así como las aristas que inciden en él.
3. Volver a 1 hasta que U sea un recubrimiento.

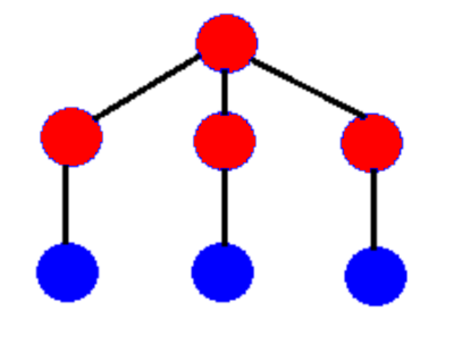
A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Algoritmo voraz basado en grados

```
# Pseudocódigo del algoritmo.
# G = (V, E)
U = []
while not E.isEmpty:
  v = V.nodeMaximumDegree()
  U.add(v)
  for edge in E:
    E.delete(edge) if edge[0] == v or edge[1] == v
  V.delete(v)
```

Algoritmo voraz basado en grados

- No es óptimo:



Algoritmos voraces para árboles

- ¿Será el problema resoluble sobre árboles?
- Los anteriores no calculan la solución óptima, hace falta una nueva idea.

Proposición 1

Sea $T = (V, E)$ un árbol de raíz r. Entonces, existe un recubrimiento minimal del mismo que no contiene a ninguna hoja del árbol pero sí a todo padre de una hoja.

Demostración. Sea $U \subset V$ un recubrimiento minimal de T.

- Si ninguna hoja del árbol está en U se tiene el resultado.
- En caso contrario, para cada hoja del árbol en U añadimos su padre y la eliminamos obteniendo así el recubrimiento U'.
- U' es un recubrimiento minimal que no contiene hojas.
- U' contiene a todo padre de una hoja ya que la arista que los une tiene al menos un nodo en U'.

Algoritmos voraces para árboles

Algoritmo óptimo

1. Se calculan las hojas del árbol en el último nivel y se añaden los padres al futuro recubrimiento, siguiendo la filosofía de la proposición 1.
2. Se eliminan las hojas del último nivel, sus padres y las aristas que inciden en estos de T.
3. Se repite el proceso mientras queden nodos en el árbol.

Algoritmos voraces para árboles

Algoritmo óptimo (Pseudocódigo)

```
# T es el árbol sobre el que se ejecuta el algoritmo.
# Se asume que se ha tomado una raíz para T.
# hojasUltimoNivel() calcula las hojas del árbol
# que se encuentran en el último nivel de este.
U = []
while not T.isEmpty:
    hojas_ultimo_nivel = T.hojasUltimoNivel()
    for hoja in hojas_ultimo_nivel:
        U.append(hoja.parent)
        T.delete([hoja, parent])
```

Proposición 2

El algoritmo calcula el recubrimiento minimal del árbol.

Demostración.

Tras la última iteración, U es un recubrimiento de T. Veamos que es minimal. Basta ver que existe un recubrimiento minimal de T que contiene a U.

Este hecho se prueba por inducción sobre las iteraciones del algoritmo. Denotamos T_i al grafo al inicio de cada iteración.

- Iteración 1. Usar la proposición 1.

Proposición 2

Supongamos el resultado cierto para la iteración $i-1$. Esto es, existe W, recubrimiento minimal de T que contiene a U.

- $W - U$ es un recubrimiento minimal de T_i.
- W_i recubrimiento minimal de T_i dado por la proposición 1.
- $W' = W_i \cup U$ es un recubrimiento. $|W'| = |W| \Rightarrow W'$ es minimal y $U \subset W'$.
- Además, W_i contiene a los padres de las hojas de T_i. Entonces, W' contiene los padres de las hojas de T_i.
- **Conclusión:** W' contiene a U tras realizar la i-ésima iteración.

Algoritmos voraces para árboles

Algoritmo óptimo para árboles con recorrido post-orden

Una mejor implementación del algoritmo se puede lograr usando el recorrido en post-orden del árbol.

Se recorre el árbol en post-orden. Para cada nodo v:

- Si tiene un hijo que no está en U se añade v a U.
- Si es hoja o todos sus hijos están en U entonces no se añade.

Algoritmos voraces para árboles

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

```python
# Llamar como algoritmoOptimo(T.raiz).
# Parámetros: v es un nodo del árbol
def algoritmoOptimo(v):
  U = []
  # Se calcula primero U para el nivel inferior.
  for hijo in v.hijos:
    U = U.union(algoritmoOptimo(hijo))
    # Si algún hijo no está en U, se añade v.
  for hijo in v.hijos:
    if hijo not in U:
      U.append(hijo); break
  return U
```

Proposición 3

El algoritmo anterior obtiene el mismo recubrimiento que el algoritmo óptimo.

Demostración. Este hecho se puede probar por inducción sobre los niveles del árbol.

Si v es una hoja, entonces no tiene hijos luego no se añade a U como ocurre en el algoritmo óptimo.

Si v tiene hijos, supongamos como hipótesis de inducción que para cada subárbol que cuelga de estos el resultado es el mismo que el que daría el algoritmo óptimo.

- U se define como la unión de los resultados sobre los subárboles.

Algoritmos voraces para árboles

A. Herrera, A. Moya, I. Sevillano, J.L. Suarez

Proposición 3

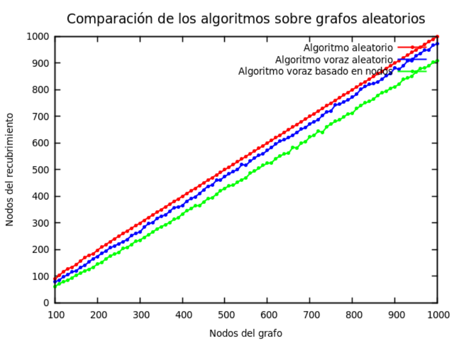
Veamos que el nuevo algoritmo tiene el mismo comportamiento sobre v que el algoritmo óptimo.

- Si todos los hijos de v están en U, el algoritmo óptimo no escogería a v pues sería una hoja en determinada iteración.
- En caso contrario, un hijo de v sería una hoja en alguna iteración al no estar nunca en U y v se añade a U .

Análisis empírico

Tamaño de los recubrimientos sobre grafos aleatorios

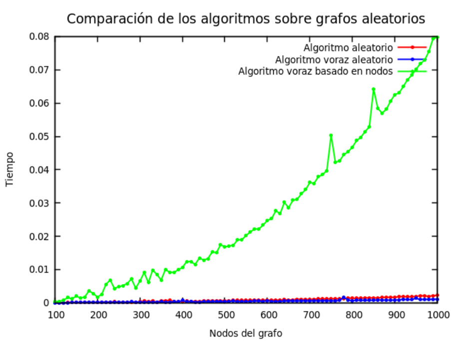
- El algoritmo aleatorio es muy malo. Los algoritmos greedy no obtienen resultados mucho mejores.



Análisis empírico

Tiempos obtenidos sobre grafos aleatorios

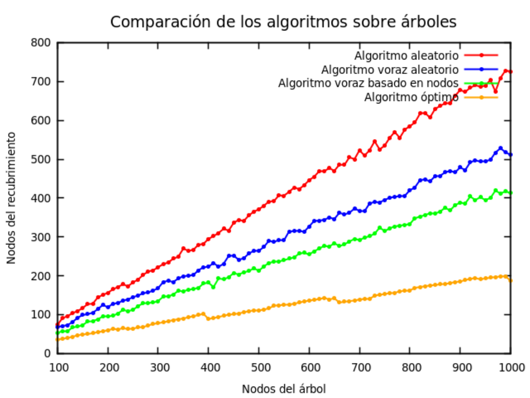
- Diferencia entre los algoritmos lineales y el algoritmo cuadrático.



Análisis empírico

Tamaño de los recubrimientos sobre árboles aleatorios

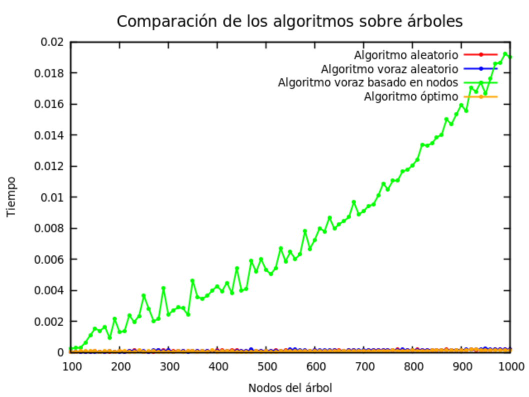
- Diferencia abrumadora con respecto al óptimo.



Análisis empírico

Tiempos obtenidos sobre árboles aleatorios

- ¡El algoritmo óptimo es lineal!



Fin de la presentación

¡Gracias por su atención!