CC4102 - Diseño y Análisis de Algoritmos
Auxiliar 4

Prof. Gonzalo Navarro; Aux. Mauricio Quezada

23 de Noviembre, 2011

1 Union-Find

1. Sea $G = (V, E)$ un grafo no dirigido. Muestre un algoritmo que permita encontrar componentes conexas usando una estructura de tipo Union-Find.

   Para ello, dé un algoritmo de inicialización Componentes-Conexas(G) que identifique las componentes de G, y luego un algoritmo Misma-Componente(u, v) para consultar si dos nodos u, v pertenecen a la misma componente. ¿Cuál es la complejidad de encontrar todas las componentes conexas de G?

2. Usando lo anterior, dé un algoritmo y su complejidad para encontrar el árbol cobertor mínimo de un grafo G no dirigido con pesos en sus aristas.

3. Muestre que cualquier secuencia de m operaciones Make-Set, Find y Union, donde todos los Union aparecen antes de cualquier Find, toma tiempo $O(m)$ si se usa Compresión de caminos y Unión por rank. ¿Qué pasa en la misma situación si sólo se usa Compresión de caminos?

1.1 Union-Find

1.1.1 Componentes conexas

Los algoritmos:

- Componentes-Conexas(G)
  - `for each vertex` $v \in V[G]$
    - `do` Crear-Conjunto(v)
  - `for each edge` $(u, v) \in E[G]$
    - `do if` Find(u) Find(v)
      - `then` Union(u, v)
- Misma-Componente(u, v)
  - `if` Find$(u) =$ Find(v)
    - `then return true`

* `else return false`

La complejidad de Componentes-Conexas(G) se determina observando la cantidad de operaciones:

1. $|V|$ operaciones Crear-Conjunto
2. $2|E|$ operaciones Find

Luego, el tiempo en que corre este algoritmo es $\mathcal{O}(|V| + 2|E| \log^* |E|)$.

1.1.2 Minimum Spanning Tree

El algoritmo de Kruskal es como sigue:

- Para cada vértice $v \in V$
  - Crear-Conjunto(v)
- $A \leftarrow \emptyset$
- Sea H una cola de prioridad donde las llaves son los pesos del grafo G
- Mientras $|A| < |V| - 1$
  - $(u, v) \leftarrow$ ExtractMin(H)
  - Si Find(u) Find(v)
    * Union(u, v)
    * Agregar (u, v) a A
- retornar A

Para ilustrar la correctitud (formalmente la prueba es por inducción y usando el Lema del corte), vemos que si $|A| < |V| - 1$, y si A' es un MST (Minimal Spanning Tree) que contiene a A y e una arista de mínimo tamaño en $A' - A$, entonces e será agregado a A en algún momento.

La complejidad del algoritmo: construir la estructura toma $O(|V|)$. Construir el heap, $O(|E|)$. En el peor caso, el loop principal tomará $|E|$ iteraciones. El costo de cada iteración es $O(\log |E|)$ para la cola de prioridad, y $O(\log^* |V|)$ para las operaciones Union-Find. Asumiendo sin problemas que $\log^* |V|$ es $O(\log |V|)$ y como $|E| \leq |V|^2$, $\log |E|$ es $O(\log |V|)$. Por lo tanto, el tiempo total es $O(|E| \log |V|)$.

1.1.3 Secuencia de operaciones Union-Find

La observación clave es que una vez que un nodo x aparece en un camino de Find, entonces x será la raíz o bien un hijo de la raíz todo el tiempo a partir de entonces.

Usando el método de contabilidad de costos obtendremos la cota de $O(m)$:

- Se cobra \$2 a Crear-Conjunto. \$1 por la operación en sí, y el resto se mantiene en el nodo creado para cuando pase a ser hijo de la raíz cuando sea consultado por primera vez en un Find

- Se cobra \$1 para un FIND. Se paga por visitar la raíz y su hijo, y para la compresión de caminos de esos dos nodos. Todos los nodos que aparezcan en el camino usarán su valor guardado para pagar su visita y la compresión.

- Se cobra \$1 por un UNION, el cual paga la unión real entre dos nodos.

Como se cobra entre \$1 y \$2 por cada operación, la secuencia cuesta a lo más $2m$, por lo que el tiempo total es $O(m)$.

2 Splay Trees

1. Pruebe que el costo amortizado de la operación SPLAY sobre un árbol de n nodos es $O(\log n)$

2.1 Solucion

Ver para splay trees y más ejemplos: http://www.cs.princeton.edu/˜fiebrink/423/AmortizedAnalysisExplained_Fieb