STAT 534

Lecture 6

Disjoint Sets and Connected Component

23 Apr 2019

mmp@stat.washington.edu

Scribes: Fengjie Chen, Yiwei Zhang

1 Notation of Graph

A *graph* G consists of a set of *nodes* V, and a set of *edges* E, where each *edge* is a pair of *nodes*. In an *undirected* graph an edge is denoted as $\{u, v\}$ (order does not matter), whereas in a *directed* graph an edge is an ordered pair (u, v) with the edge pointing from u to v.

Some common concepts of *graph* are defined as follows:

- $G =< V, E >$, $n := |V|$, $m := |E|$
- A *path* $(x, z_1, z_2, ..., z_k, y)$ from x to y in $G =< V, E >$ is valid, if $(x, z_1), (z_1, z_2), ..., (z_{k-1}, z_k), (z_k, y) \in E$
- $S \subseteq V$ is called *connected set* $\Leftrightarrow \forall x, y \in S$, there exists a *path* in E between x and y. (Not only x to y, but also y to x in *directed* graph)
- $S \subseteq V$ is called *connected component* $\Leftrightarrow S$ is *connected* and *maximal* $\Leftrightarrow \nexists S^{'} \subseteq V$, $S \subset S^{'}$, and $S^{'}$ *connected*
- A graph G is *connected* $\Leftrightarrow$ its set of nodes V is *connected*
- (Exercise) Suppose $S, S^{'} \subset V$ are connected components, proof: $S \neq S^{'} \Rightarrow S \cap S^{'} = \emptyset$. Also, $V = S_1 \cup S_2 \cup .. \cup S_k$, this decomposition is unique if $S_i (i = 1, .., k)$ is *connected components*. And *connectedness* is an *equivalence relation*.

2 Connected Components

Given $G =< V, E >$ an *undirected* graph, how to find all connected components in an efficient way? A sketch of algorithm is presented in **Algorithm 1**.

Algorithm 1 Find all Connected Components(C.C.)

Input: $G =< V, E >$
Output: $\{S_1, S_2, ..., S_K\}$ C.C., such that $V = S_1 \cup S_2 \cup ... \cup S_K$
Init: $k \leftarrow n$, $S_i \leftarrow \{i\}$ for $\forall i \in V$
for $(x, y) \in E$ **do**
  if $S_x = \mathit{FIND\text{-}SET}(x) \neq \mathit{FIND\text{-}SET}(y) = S_y$ **then**
    $S \leftarrow S_x \cup S_y$
    delete S_x, S_y, $k \leftarrow k - 1$
  end if
end for
Return $\{S_1, S_2, ..., S_K\}$

Now we need efficient realizations of *FIND-SET*(x) and *UNION-SET*(x).

2.1 Realization using doubly-linked list

One way is to use *doubly-linked list*. Suppose we realize a *class* named *NODE* with four attributes:

- *representative(R)*: point to the head of list, which is representative of S_x
- *data*: a place to cache data attached to the node
- *prev*: point to nearest previous node
  next: point to next node

Let *FIND-SET*(x) simply returns $x.R$, and $S_x \cup S_y$ makes S_y append to the head of S_x. *tail*(S_y).next=S_x, and S_x.prev=*tail*(S_y). Also, update *representative* of nodes in *list*(S_x) to point to S_y.

An example shown in class is presented below. In the example, $V = \{A, B, C, D, E, F, G\}$, $E = \{AB, AD, BC, FG, EG\}$. The following six tables show how those attributes vary during the execution.

step0	A	B	C	D	E	F	G
R	A	B	C	D	E	F	G
prev	-	-	-	-	-	-	-
next	-	-	-	-	-	-	-

step1	A	B	C	D	E	F	G
R	B	B	C	D	E	F	G
prev	B	-	-	-	-	-	-
next	-	A	-	-	-	-	-

step2	A	B	C	D	E	F	G
R	D	D	C	D	E	F	G
prev	B	D	-	-	-	-	-
next	-	A	-	B	-	-	-

step3	A	B	C	D	E	F	G
R	C	C	C	C	E	F	G
prev	B	D	-	C	-	-	-
next	-	A	D	B	-	-	-

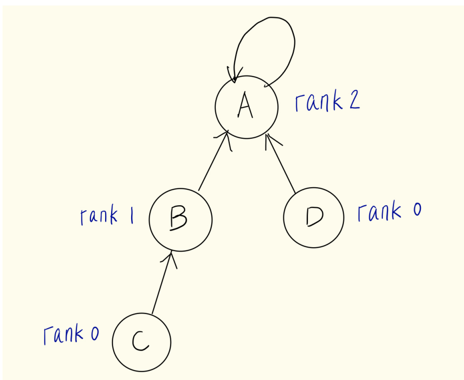
step4	A	B	C	D	E	F	G
R	C	C	C	C	E	G	G
prev	B	D	-	C	-	G	-
next	-	A	D	B	-	-	F

step5	A	B	C	D	E	F	G
R	C	C	C	C	G	G	G
prev	B	D	-	C	F	G	-
next	-	A	D	B	-	E	F

In this realization, *FIND-SET* takes $O(1)$ time, while *UNION-SET* takes $O(n)$ time in worst case.

2.2 Realization using tree structure

Using doubly linked list for a graph with $|S| = v$ takes $O(l^2)$ runtime for updating representatives. To avoid a long time, use tree structure. Below is an example of a tree structure.



The root of the tree T is the ***representative***, and points to itself.
The ***rank*** of each node is the longest length from it to the leaf in its subtree, hence for leaves, rank is 0.