600.363/463 Algorithms - Fall 2013
Solution to Assignment 7

(40 points)

23.2-2 Suppose that we represent the graph $G = (V, E)$ as an adjacency matrix. Give a simple implementation of Prims algorithm for this case that runs in $O(V^2)$ time.

Solution. If Graph $G = (V, E)$ is represented as an adjacency matrix, for an vertex u, to find its adjacent vertices, instead of searching the adjacency list, we search the row of u in the adjacency matrix. We assume that the adjacency matrix stores the edge weights, and those unconnected edges have weights 0. The Prim's algorithms is modified as:

Algorithm 1: MST-PRIM2(G, r)

```
for each $u \in V[G]$ do
    $key[u] = \infty$;
    $\pi[u] = NIL$;
end
$key[r] = 0$;
Q=V[G];
while $Q \neq \emptyset$ do
    u=EXTRACT-MIN(Q);
    for each $v \in V[G]$ do
        if $A[u,v] \neq 0$ and $v \in Q$ and $A[u,v] < key[v]$ then
            $\pi[v] = u$;
            $key[v] = A[u, v]$;
        end
    end
end
```

The outer loop (while) has $|V|$ variables and the inner loop (for) has $|V|$ variables. Hence the algorithm runs in $O(V^2)$.
Remarks There are several ways to implement Prim's algorithm in $O(V^2)$ algorithm:

(a) Using the priority queue as above;
(b) Using an array so each time extracting the minimum by one-by-one comparison, which takes $O(V)$ time;
(c) Converting the adjacency matrix into adjacency list representation in $O(V^2)$ time, then using the implementation in textbook.

All above methods run in $O(V^2)$ time. ■

23.2-8 Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut V_1, V_2, and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.
Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

Solution. We claim that the algorithm will fail. A simple counter example is shown in Figure 1. Graph $G = (V, E)$ has four vertices: $\{v_1, v_2, v_3, v_4\}$, and is partitioned into subsets $G1$ with $V_1 = \{v_1, v_2\}$ and G_2 with $V_2 = \{v_3, v_4\}$. The minimum-spanning-tree(MST) of G_1 has weight 4, and the MST of G_2 has weight 5, and the minimum-weight edge crossing the cut (V_1, V_2) has weight 1, in sum the spanning tree forming by the proposed algorithm is $v_2 - v_1 - v_4 - v_3$ which has weight 10. On the contrary, it is obvious that the MST of G is $v_4 - v_1 - v_2 - v_3$ with weight 7. Hence the proposed algorithm fails to obtain an MST. ■

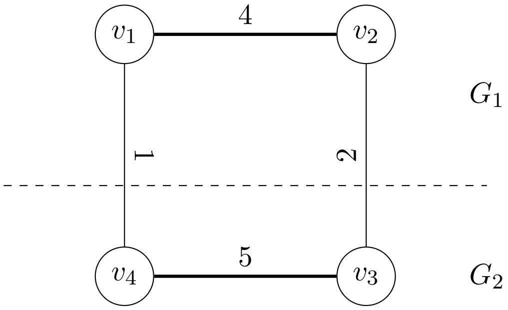

Figure 1: An counter example.

22.5-1 How can the number of strongly connected components of a graph change if a new edge is added?

Solution. The number of strongly connected components (SCCs) may remain the same or reduced to any number no less than 1, i.e. let m be the number of SCCs in the original graph, and m' be the number of SCCs of the new graph after adding the edge, then

$$m' \leq m \text{ and } m' \geq 1.$$

An explanatory example is shown in Figure 2. The left figure shows the original graph in which each node is an SCC, thus total n SCCs. If the new added edge is a self-loop of any node, or if the new added edge is pointing down, then then number of SCCs will not change. If the new added edge is a pointing up, it forms an SCC, and it may reduce the number of SCC to any number between 1 and n.

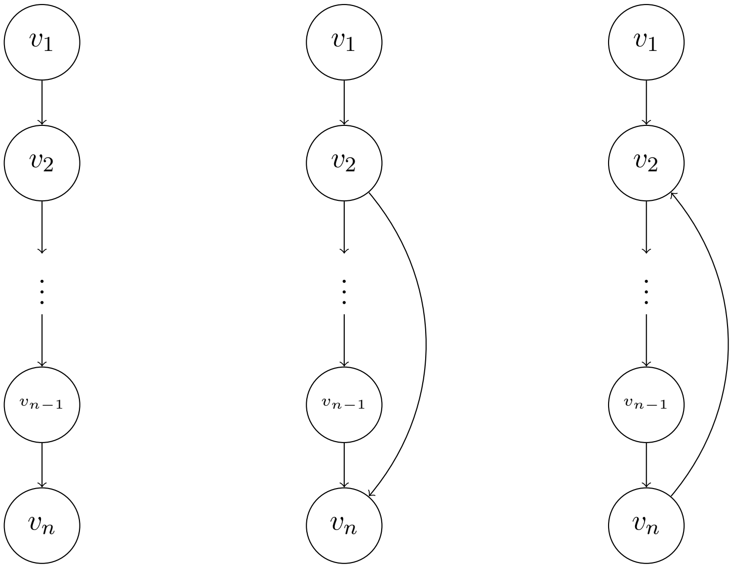

Figure 2: Examples for changing of the strongly connected component by adding an edge.

■

22.5-3 Professor Bacon claims that the algorithm for strongly connected components would be simpler if it used the original (instead of the transpose) graph in the second depth-first search and scanned the vertices in order of increasing finishing times. Does this simpler algorithm always produce correct results?

Solution. This simpler algorithm cannot always produce correct results. Figure 3 shows an example that will leads to a incorrect result. Assuming that we start DFS from v_1, then after the first DFS the order of increasing finishing time is v_2, v_1, v_3. In the second DFS, if using the original graph and scanning the vertices in order of increasing finishing time, that is, starting from v_2, will lead to one strongly connected component(SCC) of $\{v_1, v_2, v_3\}$. In fact there are two SCCs in the graph: $\{v_1, v_2\}$ and $\{v_3\}$.

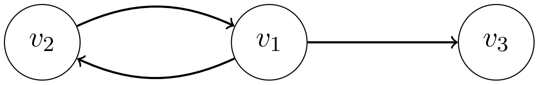

Figure 3: An example disproving the proposed algorithm.

■