CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #4, due Thursday, February 16^{th}, 2012 at 11:00

[8] 1. Kruskal's algorithm is not the only existing simple, greedy algorithm to find a minimum spanning tree of an undirected graph G. Another such algorithm is the Prim-Jarník algorithm. It is very similar to Dijkstra's algorithm, but instead of storing in `Cost(v)` the cost of the least costly path from s to v, we instead store the cost of the cheapest edge that connects s to an element of the tree T we have constructed so far. Here is most of the pseudo-code of this algorithm.

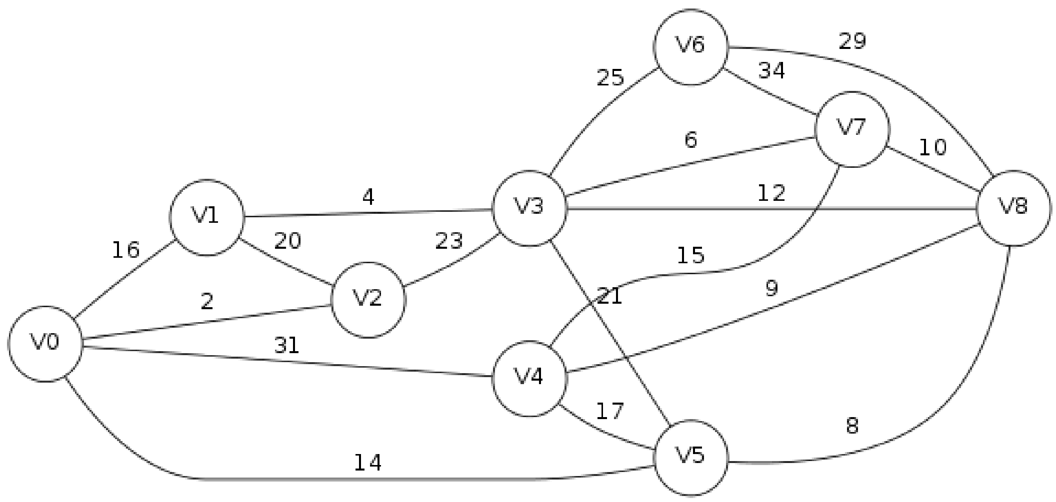
Note: You can find the pseudocode of Kruskal's and Dijkstra's algorithm in the appendix.

```
Algorithm Prim-Jarník(V, E, cost)

T ← ∅
Cost(V[0]) ← 0
Prev(V[0]) ← none

for i ← 1 to length[V] - 1 do
    Cost(V[i]) ← +∞
    Prev(V[i]) ← none

Build heap NotInTree from V using costs as keys

for i ← 1 to length[V] do
    u ← DeleteMin(NotInTree)
    add (u, Prev(u)) to T
    for each neighbor v of u do
        if (∗∗∗∗∗∗∗∗∗∗) then
            ∗∗∗∗∗∗∗∗∗∗
            Prev(v) ← u

return T
```

[2] (a) What code should replace the $**********$?

[2] (b) What is the worst-case running time of the Prim-Jarník algorithm, as a function of the number of nodes and edges of the graph? Justify your answer.

[4] (c) Execute the Prim-Jarník algorithm on the following graph, starting from node v_3, and draw the final tree.



[15] 2. Consider an undirected graph $G = (V, E)$ with positive edge weights defined by the function cost : $E \rightarrow \mathbf{R}^+$. Assume furthermore than no two edges have the same weight. For each of the following statements about G, either prove that the statement is true, or give a counter-example that shows that it is false (hint: think of the algorithms we discussed in class).

[5] a. Given a node s of G, the tree of shortest paths from s and a minimum spanning tree of G must share at least one edge.

[5] b. For every connected subgraph H of G, and minimum spanning tree T of G, $T \cap H$ is contained in a minimum spanning tree of H.

[5] c. There is a minimum spanning tree of G that contains, for every node v of G, the least-cost edge incident upon v.

Minimum Spanning Trees

```
Algorithm Kruskal(V, E, cost)

T ← ∅
H ← heap with elements of E using costs as keys
for each vertex v ∈ V do
    set C(v) to { v }

while T has fewer than |V| - 1 edges do
  (u,v) ← deleteMin(H)
  if C(u) ≠ C(v) then
    add (u,v) to T
    merge C(u) and C(v) into one cluster

return T
```

Shortest Paths

```
Algorithm Dijkstra(V, E, cost, s)

T ← ∅
Cost(V[s]) ← 0
Prev(V[s]) ← none

for i ← 0 to length[V] - 1 do
  if (i ≠ s) then
    Cost(V[i]) ← +∞
    Prev(V[i]) ← none

Build heap NotInTree from V

for i ← 1 to length[V] do
  u ← DeleteMin(NotInTree)
  add (u, Prev(u)) to T
  for each neighbor v of u do
    if (Cost(v) > Cost(u) + cost(u,v)) then
        Cost(v) ← Cost(u) + cost(u,v)
        Prev(v) ← u

return T
```