TCSS 343 - Week 8

Jake McKenzie

November 28, 2018

Graph Algorithms

"If you've never missed a flight, you're spending too much time in airports."

...

Umesh Vazirani

"Programming has things called "threads" and things called "strings" and they somehow have **** all to do with each other."

...

Ramsey Nasser

"Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less."

...

Marie Curie

0. Here we shall show how to use MSTs to get an approximation algorithm to the famous traveling salesman problem. You are given an undirected graph, where every edge {u, v} has a nonnegative weight $c_{u,v}$. The goal is to find the lowest weight tour of the graph. A tour is a sequence of vertices v_1, ... , v_r such that $v_1 = v_r$, every vertex of the graph is visited at least once, and for every i, $\{v_i, v_{i+1}\}$ is an edge of the graph. The cost of the tour is:

$$\sum_{i=1}^{r-1} c_i, v_{i+1}$$

In this problem we will give a polynomial time algorithm to find a tour that is at most twice the optimum tour, i.e., you want to design a polynomial time algorithm with approximation ratio 2.

a) Show that the weight of the optimum tour is at least the weight of the MST.

```
dfs(G)
{
list L = empty
tree T = empty
choose a starting vertex x
search(x)
while(L nonempty)
    remove edge (v,w) from end of L
    if w not yet visited
    {
    add (v,w) to T
    search(w)
    }
}

search(vertex v)
{
visit(v);
for each edge (v,w)
    add edge (v,w) to end of L
}
```

b) Show that the cost of the tour that you find in the previous part is twice the cost of the MST. By performing DFS on a MST of a graph we can obtain a tour. You might want to think about how this would obtain a tour.

1. Consider Kruska's algorithm for the graph G = (V,E) with edges {A,B}, {B,C} , {A,D}, {B,D}, {C,E}, {B,E}, {D,E}, {D,F}, {F,E}, {F,G}, {E,G}. If the edge weights are:

w({A,B}) = 7, w({B,C}) = 8, w({A,D}) = 5, w({B,D}) = 9,
w({C,E}) = 5, w({B,E}) = 7, w({D,E}) = 15, w({D,F}) = 6,
w({F,E}) = 8, w({F,G}) = 11, w({E,G}) = 9

2. Now use Prim's algorithm on the same graph G. Did you obtain the same results?

For the next few problems state whether they are true or false:

3. If $P \neq_p NP$ then every problem in NP requires exponential time.

4. If a problem σ is in P then $\sigma \leqslant_p \psi$ for every problem ψ in NP.

5. If a new problem is in NP and we can reduce a known NP-Complete problem to it, then the new problem is NP-Complete.

6. If $\Sigma_1 \in$ NP and for every problem $\Sigma_2 \in$ NP, $\Sigma_1 \leqslant_p \Sigma_2$, then Σ_1 is NP-Complete.

7. If $\Sigma_1 \in$ NP and $\Sigma_2 \leqslant_p \Sigma_1$ for some NP-Complete problem Σ_2, then Σ_1 is NP-Complete

8. A divide and conquer algorithm for the selection problem that reduced the problem on lists of size n to one selection problem of size $\frac{3n}{20}$ and another selection problem of size $\frac{9n}{10}$ plus a linear amount of extra work would yield a linear time algorithm.

9. Suppose we define a different kind of graph where we have weights on the vertices and not the edges. Does the shortest-paths problem make sense for this kind of graph? If so, give a precise and formal description of the problem. If not, explain why not. Note we are not asking for an algorithm, just what the problem is or that it makes no sense.

A. Consider the following loop, in which i is, so far, undeclared:

```
while( i == i + 1) {}
```

Find the definition of i, that precedes this loop, *such that the while loop continues for ever.*

B. Consider the following loop, in which i is, so far, undeclared:

```
while( i != i ) {}
```

Find the definition of i, that precedes this loop, *such that the while loop continues for ever.*