INTRO TO ALGORITHM, CSE 241

PROFESSOR ZHANG

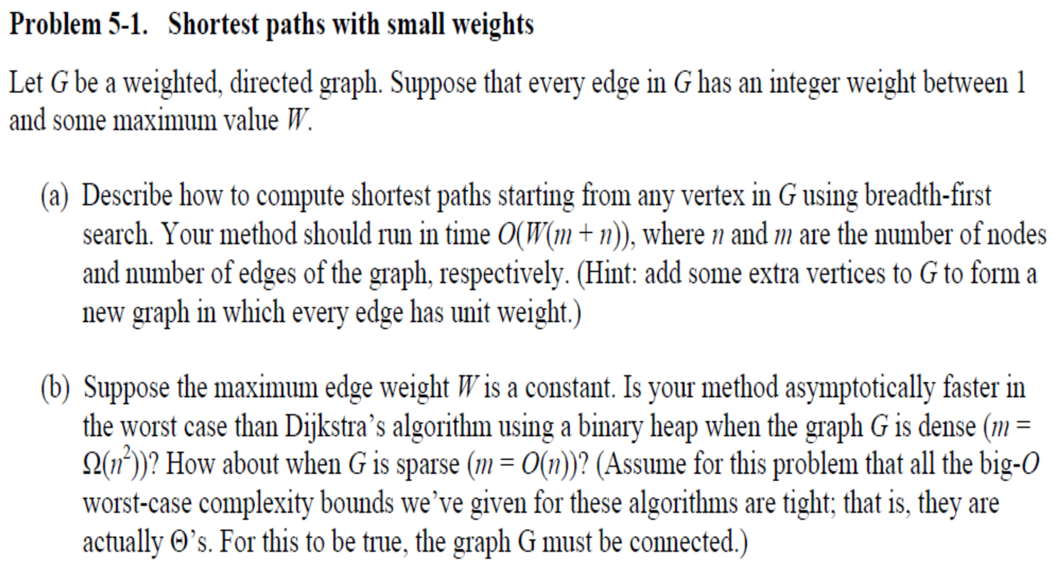
Homework 5

Peiyun(Seed) Zeng

EN:437801

Due date: Soon, 2014

PROBLEM 1



Problem 5-1. Shortest paths with small weights

Let G be a weighted, directed graph. Suppose that every edge in G has an integer weight between 1 and some maximum value W.

(a) Describe how to compute shortest paths starting from any vertex in G using breadth-first search. Your method should run in time $O(W(m+n))$, where n and m are the number of nodes and number of edges of the graph, respectively. (Hint: add some extra vertices to G to form a new graph in which every edge has unit weight.)

(b) Suppose the maximum edge weight W is a constant. Is your method asymptotically faster in the worst case than Dijkstra's algorithm using a binary heap when the graph G is dense ($m = \Omega(n^2)$)? How about when G is sparse ($m = O(n)$)? (Assume for this problem that all the big-O worst-case complexity bounds we've given for these algorithms are tight; that is, they are actually Θ's. For this to be true, the graph G must be connected.)

Solution:

(a):

Introduction

In the textbook 596-600,we can learn that BFS works for unweighted graph to find the shortest path. The correctness is proved and the time complexity is analyzed to be O(V+E) where V is the number of vertices and E is the number of edges

We are gonna take for granted that this base algorithm where BFS for shortest path of unweighted graph works (Let's call it BFSU). Then for a weighted graph, all we need to do is to find a way to **transfer** graph G (V,E) with weighted edges to a new graph G'(V',E') where E is unweighted (so each edge is with 1/unit weight). To do this, we can just put k-1 virtual vertices with equal separation along any edge with weight $k(0 < k < w)$. When introduce the virtual vertices, we label it as "vv", which allows us to distinguish them from the original vertices from original graph G

After this transformation, we will be able to find the shortest path of the new graph us using BFSS.

High-Level Design

Before we talking about the algorithm, let us be more clear about what we are assuming about BFFS.

We assume BFFS described by the textbook can correctly find the shortest path for a unweighted, directed graph in ***O(E+V)***, where V is number of vertices and E is the number of edges. The algorithm, besides giving us the distance, will also be able to return the vertices of the shortest path (using predecessor method).

The algorithm

WBFFS (V,E)

1. transform G to G'
   - start with the same vertices V
   - we will replace each edge $e = (u, v) \in E$ with e new edges. Thus $e - 1$ vertices are introduced (each with a label vv)
   - now G' becomes a unweighted graph with all edge has length unit 1.
2. we will be able to apply BFFS to find the shortest path(SP') for G'
3. We can find the SP for original graph G by taking out the virtual vertices, which are labeled with vv

Proof of correctness

1. step 1 will be able to transfer G to G', which is a unweighted, directed graph
2. the BFFS introduced by textbook then will be able to find the shortest path
3. step 3 will be able to find the original shortest path

Time Complexity

1. step 1 will at worst case take $O(W * E)$ to add all the vertices. So it is every edge is at its maximum length W.
2. step 2 is the core step. As our assumption, BFFS will take $O(E + V)$. So we need to find what is the E' and V' for the new graph G'. assuming there are E edges and V vertices for original graph. At worst case, we add W edges to each edge. Also $W - 1$ vertices will be added at each step. Thus the new graph G' has $W * E$ edges and $(W-1)V + V = W * V$ vertices. $O(W(E + V))$
3. at worst case we need to go through each vertices to take out the virtual vertices. So it at worst costs $O(W * V)$

Total Complexity: So $T(n) = O(W * E) + O(W(E + V)) + O(W * V) = O(W * (E + V))$. we say $E = m$ and $V = n$, then $T(n) = O(W * (m + n))$

(b):

Since the bound is tight we regard all upper and lower bound as tight bound We know Dijkstra's algorithm costs $T_d(n) = \Theta(E + V * \log(V)) = \Theta([m + n] * \log(n))$ while our WBFFS costs us $T(n) = \Theta(W * (m + n))$

1. when the graph is dense $m = n^2$
   - WBFFS: $T(n) = \Theta(W * n^2 + W * n)$
   - Dijkstra:$T_d(n) = \Theta(n^2 * \log(n) + n * \log(n))$

   apparently since we are talking about asymptotic efficiency, $n >> W$ so WBFFS is better
2. when the graph is sparse $m = n$
   - WBFFS: $T(n) = \Theta(W * n + W * n)$
   - Dijkstra:$T_d(n) = \Theta(n * \log(n) + n * \log(n))$

   apparently since we are talking about asymptotic efficiency, $n >> W$ so WBFFS is better

INTRO TO ALGORITHM, CSE 241

PROFESSOR ZHANG

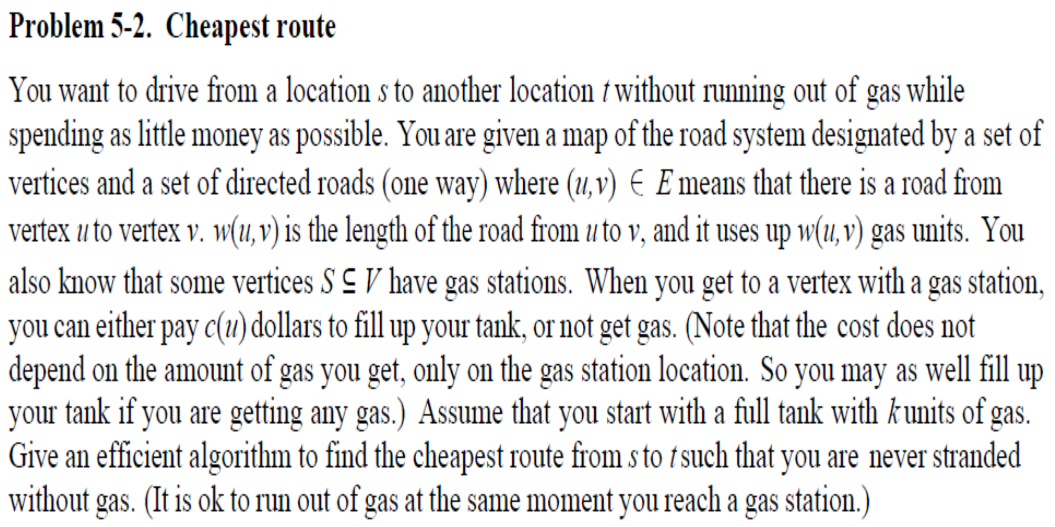
Homework 5

Peiyun(Seed) Zeng

EN:437801

Due date: Soon, 2014

PROBLEM 2



Problem 5-2. Cheapest route

You want to drive from a location s to another location t without running out of gas while spending as little money as possible. You are given a map of the road system designated by a set of vertices and a set of directed roads (one way) where $(u,v) \in E$ means that there is a road from vertex u to vertex v. $w(u,v)$ is the length of the road from u to v, and it uses up $w(u,v)$ gas units. You also know that some vertices $S \subseteq V$ have gas stations. When you get to a vertex with a gas station, you can either pay $c(u)$ dollars to fill up your tank, or not get gas. (Note that the cost does not depend on the amount of gas you get, only on the gas station location. So you may as well fill up your tank if you are getting any gas.) Assume that you start with a full tank with k units of gas. Give an efficient algorithm to find the cheapest route from s to t such that you are never stranded without gas. (It is ok to run out of gas at the same moment you reach a gas station.)

Solution:

Introduction

We are gonna use greedy algorithm paradigm to design the algorithm. In this problem we also assume we use a Dijkstra algorithm works property and cost $O([E+V]\log(V))$ let's call it DJ(G,u,v)

High-Level Design

1. apply $DJ(G,s,t)$ to each vertex, thus we find the shortest path from each vertex to the destination t,d

2. find the vertex u_0 that is with the largest d while $d<k$. the cheapest route then will be the cheapest route from s to u_0 plus the shortest path from u_0 to t (with no stop)

3. Find the cheapest route from s to u_0 by recursively calling $DJ(G,s,u_0)$ (so we update u_0 each time)

4. each u_0 we find is the place we have to stop to refill, so apply Dijistra to find the closest gas station near each u_0

5. calculate the total cost

Proof of Correctness

we will be able to find the shortest path using Dijkstra algorithm and each iteration the problem size is reduced.

so the known cheapest route is expanding each iteration. in the end we will be able combine all information and get the total cheapest route

Time complexity

1. we pay $n*(n*\log(n)+m*\log(n))$ to calculate the distance to s for each node

2. $T(n)=T(n-1)+O(1)$. This is solved to be n^2

So the Total complexity is :$T(n)=O(n^2+[m+n]\log(n))$