Homework 5 due Friday 2/11/2015

Huimin He , section 1

February 11, 2015

1. 4.4 weighted DAG linear single-source min-cost path

   Input: adjacency list representation of graph $G(V, E, w)$.

   Output: min cost path tree and min cost from s to each vertex.

   (a) Algorithm

   Phase 1

   Topological sort the graph and label the sorted vertices with index i from 1 to $|V| - 1$. Adjust the adjacency list of the graph s.t. vertices s u_1 u_2 u_3 ... to $u_{|V|-1}$ in the list are in the sorted order.

   Phase 2 modification of dijsktra algorithm.

   Define variable $u.cost$ to be the cost the vertex. $u.p$ to be the parent of vertex v. i is the loop counter and also the index of the vertices except for the source s. i is from 1 to $|V| - 1$.

   Pseudocode

   Initialization

   ```
   for each vertex u in G
       u.p = NIL
       u.cost = ∞
   s.p = s
   s.cost = −∞ End Initialiazation

   for i = 1 to |V| − 1
       for each vertex v in G.adj[u_i] (note u_i are sorted)
           Relax(u,v,w)
   ```

   Define Relax(u,v,w)

   ```
   if v.cost < u.cost + w(u, v)
       v.cost = u.cost + w(u, v)
       v.p = u
   ```

   (b) analysis of time complexity

   Phase 1 topological sort runs in linear time (taken from previous studies). Phase 2 runs also in linear time. The initialization takes $2|V|$ operations. The for loop runs $O|E|$ times as each edge is relaxed once. So the total time is

   $$2|V| + O|E| + O(|V| + |E|) = O(|V| + |E|)$$

linear in the size of an andjacency-list representation.

(c) proof of correctness Loop invariants reference to the book. page 650, 655, and 671-674.

Upper-bound property

For all vertices in V, when $v.cost$ is at optimal, it does not change.

Convergence property

s to u to v is a shortest path from s to v. If a sequence of relaxation steps including Relax(u,v,w) is executed on the edges of the G, and $u.cost$ is optimal at any time prior to the relaxation, then $v.cost$is optimal at all timess after the call.

Path-relaxation property

If there is a shortest path $(v_0, v_1, ..., v_k)$ and we relax the edges in the order (v_0, v_1),(v_2, v_3), ...,(v_{k-1}, v_k), then $v_k.cost$ is optimal.

Shortest path result

The pseudocode above will assign optimal cost to each of its vertex it terminates. And the predecessor subgraph $G.p$ is the shortest path tree.

Proof of shortest path result:

For unreachable vertices from s, $v.cost$ is ∞. For any reachable vertices v_k, so there is a shortest path $p = (s, v_1, ..., v_k)$. Because all the vertices are in topologically sorted order, we relax the edges (v_0, v_1),(v_2, v_3),...,(v_{k-1}, v_k) in sorted order. According to path -relaxation property, $v_i.cost$ is optimal for $i = 0, 1, ...k$. And by predecessor-subgraph property, $G.p$ is a shortest path-tree.

Proof of Path relaxation property: Induction hypothesis: after ith edge of path p is relaxed, v_i is optimal. Base case, $i = 0$, before relaxing any edge in p, $v_0(s)$ has cost $-\infty$. This does not change after any relaxing. so $v_0.cost$ is optimal after edge 0 is relaxead by upper bound property.

Inductive step, assume $v_{i-1}.cost$ is optimal. By convergence property, after relaxing the edge (v_{i-1}, v_i),v_i would be optimal and v_i does not change afterwards. So Path relaxation property is proved.

Proof of convergence property

Relaxing edge(u, v) will result in :

$$v.d \leq u.d + w(u, v)$$

$$= u.cost(optimal) + w(u, v)$$

$$= v.cost(optimal)$$

from the theorem that subgraph is also a shortest path. By the upper-bound property, $v.cost \geq v.cost(optimal)$. So $v.cost = v.cost(optimal$ is concluded and is maintained afterwards.

Proof of upper bound property

$v.cost \geq v.cost(optimal)$ for all vertices v in V by induction over the number of relaxation steps i. For $i = 0$, this is true since $v.cost = \infty \geq optimalcost$ for all vertices other than source. By inductive hypothesis, $v.cost \geq v.cost(optimal)$ prior to relaxation for all vertices. If cost of v changes, then

$$v.cost = u.cost + w(u, v)$$

$$\geq u.cost(optimal) + w(u, v)$$

by inductive hypothesis

$$\geq v.cost(optimal)$$

by trangle inequality.

$v.cost$ never changes after it reaches optimal because any relaxation will not increase the cost.

2. 5.3 Bellman-Ford algorithm single-source min-cost

   part (b) Let i denote the i^{th} iteration of the loop. $z = False$ if and only if non of the edge is relaxed. Also, there is no negative cycle if $z = False$, since relax along a negative cycle would never end, making $z = False$ impossible. So the configuration does not change under this condition. Let m be the iteration that $z = False$. Let C_{m-1} be the configuration before the iteration that $z = False$. C_m be the configuration after the iteration that $z = False$. So C_{m-1}'s vertices have min cost of combinatorial length less than $m-1$. C_m's vertices have min cost of combinatorial length less than m. But $C_m = C_{m-1}$ since $C_m = T(C_{m-1}) = C_{m-1}$. And $(C_m) = (C_{m+1})$ (C_{m+1} is the configuration after two iterations that $z = False$). When $i > m$, $C_i = C_m$. so once $z = False$, the configuration does not change. After $z = False$,the cost of all vertices remain same and they are all optimal cost of combinotorial length less than N ($N > |V|$) (by part(a)). Since there is no negative cycle, maximum combinatorial length for optimal path is $|V| - 1$. Having a larger combinatorial legnth requires a cycle. Non-negative cycles can always be dropped since dropping them does not increase the cost of the vertices. So configuration $C_{|V|-1} = C_{m-1}$ has the optimal costs for all vertices.

   part (d) an accessible negative cycle exists if and only if iteration $i = |V|$ of the main for loop sets the variable z to $TRUE$.

   From left to right: if there is a negative cycle, at iteration $i = |V|$, the relaxation still occurs since relaxing along the negative cycle never ends ($c(v) > c(u) + w(u, v)$ always true for some vertices along the negative cycle). So z is set $TRUE$. Left to right proved.

   From right to left: if the main loop sets the variable z to $TRUE$ at iteration $i = |V|$, there must be a negative cycle.

   Proof by contradiction: Assume no negative cycle. We know at iteration $i = |V| - 1$
   By **part(a)** for all vertices, $v.cost$ is optimal of all walks of combinatorial $length \leq i$ from s to v ($i = |V| - 1$ here). (i)

   At iteration i, the max combinatorial length is i. When $z = TRUE$, $c(v) > c(u) + w(u, v)$. so we have a path to a vertex of combinatorial length i (for $length \leq i - 1$ all vertices are at optimal from last iteration from fact (i)) that gives lower cost. Since there are $|V|$ vertices including source ($|V| - 1$ edges), there must be a cycle to have combinatorial length $i = |V|$. From assumption there is no negative cycle, the cycle must be positive or zero. If the cycle is non-negative, the condition $c(v) > c(u) + w(u, v)$ would be evaluated false and giving $z = False$ because non negative cycles does not reduce the cost for any vertex. This contradicts with the premise that $z = TRUE$. By contradiction, there must be a negative cycle. Right to left is proved.

3. 5.4 Uphill - downhill Dijkstra problem

   Note: HW 4.4 claimed and concluded that we can solve the single-source min cost problem for a DAG in linear time. Denote this algorithm as DAG_{min}.

   Divide the graph described in HW5.4 G into two subgraphs, G_{uphill} and $G_{downhill}$ (G_{uphill} contains only uphill edges and $G_{downhill}$ contains only downhill edges). Given the fact that $v.elevation$ is different for all vertices in G, claim that G_{uphill} and $G_{downhill}$ are both DAG. This is easy to see because cycles are not possible. Take G_{uphill} for example: a cycles requires an edge from v to u such that there is a path from u to v. Since we are only taking uphill edges, $u.elevation < v.elevation$. Then the edge vu would be downhill, which contradicts with the uphill graph. Thus DAG is proved. The same is true for $G_{downhill}$. Then solve the single source two subgraphs $G_{downhill}$ with source s and G^T_{uphill} (transpose) with source t. The rest of the algorithm follows the Dijkstra time method.(search the two arrays generated by the two graphs and then choose the min cost. This cost $O|V|$). The DAG part should be solved in linear $O(|V|+|E|)$ as the Note claims. So the overall run time would be $O(|V|+|E|)$.

4. 5.6 Divide and conquer:order statistics We want to show that $T(n) = O(n)$ given the following recurrence relatioship,

$$T(n) \leq T(\frac{1}{5}n) + T(\frac{7}{10}n) + O(n)$$

We need to pick a $g(n) = O(n)$ and show that,

$$g(n) \geq g(\frac{n}{5}) + g(\frac{7n}{10}) + Cn(C > 0) \tag{1}$$

$$g(1) \geq T(1) \tag{2}$$

Then

$$T(n) \leq g(n)$$

would follow and we will eventually have

$$T(n) = O(n)$$

Pick $g(n) = An(A > 0)$ and plug into equation 1 We get

$$An \geq (\frac{1}{5} + \frac{7}{10})(An) + Cn$$

$$\frac{1}{10}(An) \geq Cn$$

Pick $A = 10C + T(1)$ and the above condition can be satisfied since $T(1) \geq 0$.

Plug $g(n) = An$ to equation 2 and we get, We get

$$g(1) = A + T(1) \geq T(1)$$

Thus equaton 1 and 2 are both satisfied with the pick $A = 10C + T(1)$. It can be concluded that

$$T(n) \leq g(n)$$

so

$$T(n) = O(n)$$

.

5. XC credit **Claim**: The procedure does not work with groups of size 3 but it works with groups of size 7 with $T(n) = O(n)$. To get recurrence relationship for size is 7, divide n number to group of size 7. Choose the median M of the medians of all the groups. The number of elements that are smaller than M is

$$\frac{\frac{n}{7} - 1}{2} + 3$$

This equals

$$\frac{2n}{7} - 2 + 3 = \frac{2n}{7} + 1$$

So $\frac{2n}{7} < rank(M) < \frac{5n}{7}$ The recurrence relationship can be written as

$$T(n) \leq T(\frac{5n}{7}) + T(\frac{n}{7}) + O(n)$$

Here $T(n) = O(n)$. Proof: We want to find a $g(n)$ s.t.

$$g(n) \geq g(\frac{5n}{7}) + g(\frac{n}{7}) + Cn(C > 0)$$

$$g(1) \geq T(1)$$

let $g(n) = An$ with $A = 7C + T(1)$.

$$An \leq \frac{5n}{7}A + \frac{n}{7}A + Cn$$

$$\frac{1}{7}An \leq Cn$$

$$Cn + \frac{T(1)}{7}n \geq Cn$$

This is true as $T(1)$ is positive, so the first condition is saistified.

$$g(1) = A = 7C + T(1) \geq T(1)$$

because $T(1) \geq 0$. So the second condition is also satisfied. Thus $T(n) \leq g(n)$ follows. $T(n) = O(n)$ proved.

For blocks are size of 3 case, use the same reasoning and get the recurrence relationship.

$$T(n) \leq T(\frac{n}{3}) + T(\frac{2n}{3}) + O(n)$$

let $g(n) = An + B$. We need $g(n)$ to satisfy

$$g(n) \geq g(\frac{n}{3}) + g(\frac{2n}{3}) + Cn(C > 0)$$

$$An + B \geq \frac{An}{3} + B + \frac{2An}{3} + B + Cn$$

$$An + B - An - 2B \geq Cn$$

$$-B \geq Cn$$

But Cn goes to infinity when n is large. There does not exist a B to satisfy the above the equation. So $T(n)$ is not $O(n)$. Higher order on n will satisfy the the recurrence inequality in this case.

So it is concluded that block of 3 does not run in linear time but block of 7 runs in linear time.