CS 3510 Homework 2

Wenqi He, whe47

October 9, 2017

1

(a)

1. State: $LOS[i], LES[i]$: length of the longest odd/even sequence ending at i
2. Base case: $LOS[i] \geq 1, LES[i] \geq 0$
3. Transition:

$$LES[i] = \max_{1 \leq j < i, A[j] < A[i]} \{LOS[j] + 1\}$$

$$LOS[i] = \max_{1 \leq j < i, A[j] > A[i]} \{LES[j] + 1\}$$

$$LAS[i] = \max\{LES[i], LOS[i]\}$$

(b)

Suppose there are two lists of vertices A_{even} and A_{odd}, and each element $A[i]$ is represented by vertices $A_{even}[i]$ and $A_{odd}[i]$. Every path must start from some vertex in A_{odd}, and each edge $u : i \to j$ must satisfy $i < j$ and either:
(i) the i-th vertex is in A_{odd}, the j-th vertex is in A_{even}, and

$$A_{odd}[i] < A_{even}[j]$$

(ii) the i-th vertex is in A_{even}, the j-th vertex is in A_{odd}, and

$$A_{even}[i] > A_{odd}[j]$$

This graph is a directed acyclic graph, so the longest path problem can be solved using dynamic programming in the same way as (a)

2

(a)

The cut given by $\{A, B\}$,

$$E(\{A, B\}, V \setminus \{A, B\})$$

(b)

If an edge e is in some MST in G, then removing the edge from the MST partitions the vertices in G into two sets, S and $V(G) - S$. By the cut rule, e must be the smallest edge on the cut S. If the weight of e is unchanged, and other edges either remain the same weight or gains more weight, e must still be the smallest edge on the cut S in H, and by the cut rule it must still be in some MST in H

3

(a)

Graph: A string of vertices, with each vertex of at most degree 2.
Order: From the further end to the closer end.

(b)

For such graphs there exists a shortest path tree. Suppose such a tree is already known, then we can order the edges in the following way: First update the edges that are directly connected to the root on the shortest path tree, then update the edges that are one edge away from the root, and so on. This way, when $dist[t]$ is being updated, $dist[s]$ of it's 'parent vertex' s on the shortest path tree has already been updated to its true distance, so $dist[t]$ will be updated to the true distance in the same iteration.

(c)

Using the ordering given in the problem (top to bottom, left to right), for each iteration, the true distances can propagate all the way to the right and to the bottom along some shortest path, but only 1 block leftwards or upwards, therefore the number of iterations is bounded by the length of the longest subpath in the shortest path tree that goes leftwards or upwards. To construct a graph that contains such a shortest path tree, we can:

(1) Choose path

$$\big((1,1)\cdots(1,k)\cdots(k,k)\cdots(k,k-\lfloor k/10\rfloor)\big)$$

as the shortest path to $(k,k-\lfloor k/10\rfloor))$. The subpath

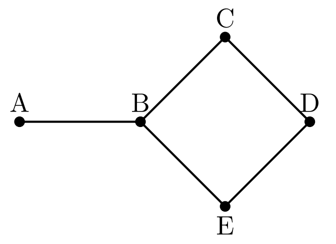
$$((1,1)\cdots(1,k)\cdots(k,k),(k,k-1))$$

takes only one iteration to update, and the rest of the path takes $\lfloor k/10\rfloor - 1$ iterations.
(2) Randomly select other branches for the shortest path tree.
(3) Set the weight of all edges on the tree to 1.
(4) Set the weight of all other edges to some sufficiently large number so that the tree constructed is indeed a shortest path tree.

Since it takes $\lfloor k/10\rfloor$ iterations to ensure that this particular shortest path is fully updated, the entire algorithm takes at least $\lfloor k/10\rfloor$ iterations.

4

(a)



Using the given algorithm, one might first color $\{A, D\}$ with color 1, then color $\{C, E\}$ with color 2, and finally color B with color 3. However, there is a way to color the graph using only 2 colors: color $\{B, D\}$ with color 1 and $\{A, C, E\}$ with color 2.

(b)

1. State: $Min(S)$: The minimum number of colors needed to color a subset S of vertices in graph G. For a graph that has n vertices, there are 2^n subsets of vertices, and therefore 2^n states.
2. Base case: $Min(\{v_i\}) = 1$
3. Transition:
Iterate through all the subsets S_i of S, if the subset is an independent set, i.e. if $Min(S_i) = 1$, compare and update $Min(S)$ by:

$$Min(S) = \min\{Min(S), 1 + Min(S - S_i)\}$$

In other words, the transition is:

$$Min(S) = \min_{1 \leq i \leq 2^{\|S\|}} \{1 + Min(S - S_i)\}$$

where S_i is a independent subset of S.

There are $2^{\|S\|} \leq \mathcal{O}(2^n)$ subsets of S, and therefore $\mathcal{O}(2^n)$ operations are needed for each state transition. Since both the inner loop and the outer loop are $\mathcal{O}(2^n)$, the entire algorithm is $\mathcal{O}(2^n) \cdot \mathcal{O}(2^n) = \mathcal{O}(4^n)$

(c)

Since the number of operations performed on each subset S is $2^{\|S\|}$ rather than 2^n, the upper bound can actually be tighter. There are $\begin{pmatrix} i \\ n \end{pmatrix}$ subsets of size i, and size i ranges from 0 to n, so the total runtime is:

$$T(n) = \sum_{0 \leq i \leq n} \begin{pmatrix} i \\ n \end{pmatrix} \cdot 2^i = \sum_{0 \leq i \leq n} \begin{pmatrix} i \\ n \end{pmatrix} \cdot 2^i \cdot 1^{n-i} = (2+1)^n = 3^n \leq \mathcal{O}(3^n)$$