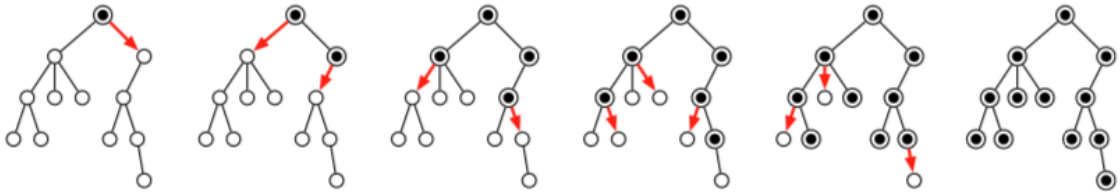
Fall 2019 B503 Homework 4 Solutions

Due date: TBD Lecturer: Qin Zhang

Problem 1 (10 points). Suppose we need to distribute a message to all the nodes in a rooted tree. Initially, only the root node knows the message. In a single round, any node that knows the message can forward it to at most one of its children. Describe and analyze an efficient algorithm to compute the minimum number of rounds required for the message to be delivered to every node.



A message being distributed through a tree in five rounds.

Solution: Define $OPT(v)$ to be the minimum number of rounds required for the message to be delivered to every node in the sub tree rooted in v. Our desired output is $OPT(root)$.

To pass information with minimum round say we are at node v, we need to first pass info to the child node u with maximum $OPT(u)$. If we don't do in such a greedy way, it is easy to see that we will definitely not get a better result.

Boundary condition: if v is a leaf, we make $OPT(v) = 0$.

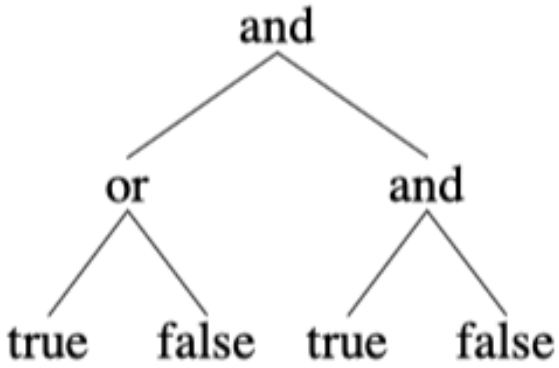
Define $Index(u, v)$ to be index in a sorted array based on $OPT(u)$ (from large to small) for all v's children u. The index begins with 1. Then we have the following relationship:

$$OPT(v) = \max_{u \text{ as } v\text{'s children}} \{OPT(u) + Index(u, v)\}$$

The running time of the algorithm is $O(n \log n)$ due to the sort process.

Problem 2 (10 points). At Aperture Bakeries, every cake comes with a binary boolean-valued tree indicating whether or not it is available. Each leaf in the tree has either a *true* or a *false* value. Each of the remaining nodes has exactly two children and is labeled either *and* or *or*; the value is the result of recursively applying the operator to the values of the children. One example is the following tree:

If the root of a tree evaluates to *false*, like the one above, the cake is a lie and you cannot have it. Any *true* cake is free for the taking. You may modify a tree to make it *true*; the only thing you can do to change a tree is to turn a *false* leaf into a *true* leaf, or vice versa.



This costs \$1 for each leaf you change. You can't alter the operators or the structure of the tree.

Describe an efficient algorithm to determine the minimum cost of a cake whose tree has n nodes, and analyze its running time.

Solution: We can use dynamic programming to determine the cost. Define $OPT(v)$ as the minimum cost to make node v to be evaluated *true*. Our desired output is $OPT(root)$.

Boundary condition: If v is a leaf with value *true*, we make $OPT(v) = 0$. If v is a leaf with value *false*, we make $OPT(v) = 1$.

We have the following relationship:

$$OPT(v) = \begin{cases} \sum_{u \text{ as } v\text{'s children}} OPT(u) & \text{if v has operator } \textit{and} \\ \min_{u \text{ as } v\text{'s children}} OPT(u) & \text{if v has operator } \textit{or} \end{cases}$$

because both children need to be *true* in an *and* operator node and only one child to be *true* in an *or* operator node.

Since there are $O(n)$ subproblems, and the cost of each subproblem is constant, the total running time is $O(n)$.

Problem 3 (10 points). Consider two vertices, s and t, in a *directed acyclic graph* $G = (V, E)$. Give an efficient algorithm to determine whether the number of paths in G from s to t is odd or even. Analyze its running time in terms of number of vertices n and number of edges m.

Solution: Define $OPT(v)$ as the number of paths in G from v to t. Our desired output is the evenness or oddness of $OPT(s)$.

As the graph is directed acyclic, we could produce a topological ordering $v_1, v_2, \ldots, v_n$ of the vertices in G. If t is in front of s, then there is no path between them. Otherwise, without loss of generality, let $v_1 = s$ and $v_n = t$ as we only care about nodes in $s \to t$ range.

Boundary condition: $OPT(t) = 0$.

We have the following relationship:

$$OPT(v) = \sum_{(u,v)\in E} OPT(u)$$

We compute the value of $OPT(v)$ from t to s as in the reversed topological order.

We need to solve $O(n)$ problem and each take O(degree of node) time, making the total running time $O(m)$. The topological sort takes $O(m+n)$ time. So the total running time is $O(m+n)$.

Problem 4 (10 points). Recall that in the standard sequence alignment problem, you are given two strings X and Y , of length n and m, respectively, over a common alphabet Σ. The input also includes the penalty α_{gap} of inserting a gap and the penalty α_{ab} of (mis)matching each pair of characters a, b.

It often makes sense that the penalty for a gap of length 10, say, should be strictly smaller than 10 times the penalty of a single gap. (This reflects the possibility that a large chunk of a genome may get added or deleted via a single "mutation".)

To model this, consider a generalization of the sequence alignment problem in which, instead of a single gap penalty α_{gap} , the input includes parameters α_0, α_1 , and where the penalty of inserting exactly $k \geq 1$ gaps in a row is $\alpha_0+k\alpha_1$. (α_0 is larger than α_1) For example, the string AGT can be matched with ACCCGCCT with total penalty $(\alpha_0+3\alpha_1)+(\alpha_0+2\alpha_1) = 2\alpha_0 + 5\alpha_1$.

Design a dynamic programming algorithm that correctly solves this generalized sequence alignment problem and runs in $O(mn)$ time. Your algorithm should compute the value (i.e., minimum-possible total penalty) of an optimal alignment.

Solution: In this generalization of sequence alignment problem, we need some more information about our alignment. Three possible states can occur in the alignment: 1) continuing from a match; 2) extending an existed gap in x; 3) extending an existed gap in y;

To cover all possible states, we define the following three matrices. Define $M_{xy}(i,j)$ the maximum alignment score of $X[1\dots i]$ and $Y[1\dots j]$ where x_i is match with y_j. Define $M_x(i,j)$ the maximum alignment score of $X[1\dots i]$ and $Y[1\dots j]$ where x_i is match with a gap. Define $M_y(i,j)$ the maximum alignment score of $X[1\dots i]$ and $Y[1\dots j]$ where y_j is match with a gap. Our desired output is the maximum value in $\{M_{xy}(n,m), M_x(n,m), M_y(n,m)\}$.

Boundary conditions: we only need to fill $M(0,j)$ and $M(i,0)$ for all three matrices based on the new gap penalty. For $M(0,j)$, $M(0,1) = -\alpha_0 - \alpha_1$ and $M(0,j) = M(0,j-1) - \alpha_1$ for $1 < i \leq m$. $M(i,0)$ can be filled in the similar rule.

Then we will have the following relationships:

$$\begin{cases} M_{xy}(i,j) = \alpha_{x_i y_j} + \max \begin{cases} M_{xy}(i-1,j-1) \\ M_x(i-1,j-1) \\ M_y(i-1,j-1) \end{cases} \\ M_x(i,j) = \begin{cases} M_{xy}(i-1,j) - \alpha_0 - \alpha_1 & \text{start a gap in Y} \\ M_x(i-1,j) - \alpha_1 & \text{continue a gap in Y} \\ M_y(i-1,j) - \alpha_0 - \alpha_1 & \text{stop a gap in X and start a gap in Y} \end{cases} \\ M_y(i,j) = \begin{cases} M_{xy}(i,j-1) - \alpha_0 - \alpha_1 & \text{start a gap in X} \\ M_y(i,j-1) - \alpha_1 & \text{continue a gap in X} \\ M_x(i,j-1) - \alpha_0 - \alpha_1 & \text{stop a gap in Y and start a gap in X} \end{cases} \end{cases}$$

We have $O(nm)$ entries to fill in and each takes $O(1)$ time. The total running time is $O(nm)$. The trace back can be done in the same way as the original sequence alignment problem.

Problem 5 (10 points). In this problem, we consider a problem about clustering n points on the number line into k clusters. The input are distinct points $x_1, x_2 \ldots, x_n \in R$ in sorted order, and a parameter $k \leq n$. We would like to divide the n points into k disjoint non-empty clusters $S_1, \ldots, S_k$such that $\bigcup_{i=1}^{k} S_i = \{x_1, x_2 \ldots, x_n\}$ and all points in S_i are to the left of all points in S_{i+1} for $1 \leq i < k$, i.e., for any $y \in S_i, z \in S_{i+1}, y < z$.

The goal is to minimize the following objective function:

$$\sum_{i=1}^{k} [\max(S_i) - \min(S_i)]^2$$

Note that $\min(S_i)$ is the minimum element of S_i , i.e., the leftmost point of S_i. Similarly, $\max(S_i)$ is the maximum element of S_i, i.e., the rightmost point of S_i.

Design an $O(n^2 k)$ time algorithm to find the optimal clustering using dynamic programming. Prove the correctness and analyze running time of your algorithm.

Solution: Define $OPT(i,j)$ as the minimum cost for clustering $x_1, x_2, \ldots, x_i$ into j clusters. The desired output is $OPT(n,k)$.

Boundary conditions: $OPT(i,i) = 0$ $(0 \leq i \leq k)$ as we could have i clusters each with size 1.

For the new coming point x_j, we only need to explore all it clustering possibilities. So we have the following relationship if $j \leq \min(k,i)$:

$$OPT(i,j) = \min_{j \leq l \leq i} \{OPT(l-1, j-1) + (x_i - x_l)^2\}$$

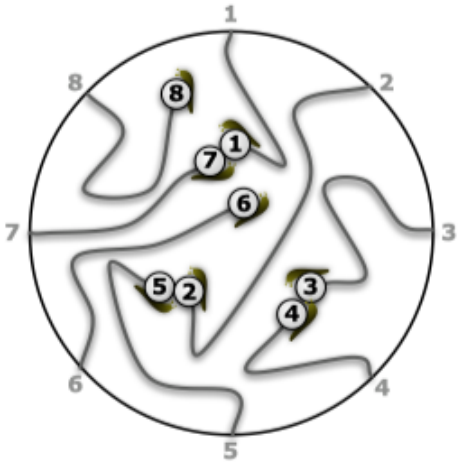
We apply this for i increasing from 1 to n.

Each entry of the table takes $O(n)$ time to compute and there are $O(nk)$ entries, so overall running time is $O(n^2 k)$.

Problem 6 (10 points). Every year, as part of its annual meeting, the Antarctican Snail Lovers of Upper Glacierville hold a Round Table Mating Race. Several high-quality breeding snails are placed at the edge of a round table. The snails are numbered in order around the

table from 1 to n. During the race, each snail wanders around the table, leaving a trail of slime behind it. The snails have been specially trained never to fall off the edge of the table or to cross a slime trail, even their own. If two snails meet, they are declared a breeding pair, removed from the table, and whisked away to a romantic hole in the ground to make little baby snails. Note that some snails may never find a mate, even if the race goes on forever.

For every pair of snails, the Antarctican SLUG race organizers have posted a monetary reward, to be paid to the owners if that pair of snails meets during the Mating Race. Specifically, there is a two-dimensional array $M[1..n, 1..n]$ posted on the wall behind the Round Table, where $M[i, j] = M[j, i]$ is the reward to be paid if snails i and j meet.



The end of a typical Antarctican SLUG race. Snails 6 and 8 never find mates.
The organizers must pay $M[3,4] + M[2,5] + M[1,7]$.

Describe and analyze an algorithm to compute the maximum total reward that the organizers could be forced to pay, given the array M as input.

Solution: As the snails cannot cross a slime trail, if snail i and snail j meet during the race, all snails within the (i, j) range can only meet snails that are also within the range.

So we could define our sub problems in the following way. Define $OPT(i, j)$ the maximum total reward for snails $(i, i + 1, \ldots, j)$. Our desired output is $OPT(1, n)$.

Boundary condition: Self pair is set to be 0 $OPT(i, i) = 0$ and neighboring pair is set to be $OPT(i, i + 1) = M(i, i + 1)$.

Then we have the following relationship:

$$OPT(i, j) = \begin{cases} \max_{i \le k < j} \{OPT(i, k - 1) + OPT(k + 1, j - 1) + M(k, j)\} \\ OPT(i, j - 1) \end{cases}$$

The first relationship corresponds to the case where j is matched with some k and the second relationship corresponds to the case j is not matched with any other snail.

We fill the matrix by the increasing order of l which represents the interval length. The boundary condition solves the case where $l = 0$ and $l = 1$. Then for $l = \{2, 3 \ldots\}$, we solve for all possible $OPT(i, i + l)$ until we reach $OPT(1, n)$.

We have $O(n^2)$ subproblems and it takes $O(n)$ time to solve each of them. The total running time is $O(n^3)$.

Problem 7 (10 points). Arthur Dent has \$500 and 1000 hours to go from Cambridge, MA, to Berkeley, CA. He has a map of the States represented as a directed graph $G = (V, E)$. The vertices of the graph represent towns, and there is a directed edge $e = (A, B)$ from town A to town B if there is some means of public transportation connecting the two towns. Moreover, the edge is labeled with a pair (m_e, t_e), representing the cost $m_e \in \{0, 1, \dots\}$ in dollars of transportation from A to B and the time $t_e \in \{0, 1, \dots\}$ in hours that it takes to go from A to B. But m_e and t_e cannot both be 0.

Arthur is interested in finding a path from Cambridge to Berkeley that does not cost more than \$500 and does not take more than 1000 hours, while also minimizing the objective $5M^2 + 2T^2$, where M is the cost of the trip in dollars and T is the duration of the trip in hours.

(a) Due to his lack of knowledge in algorithms, he gave up on the idea of respecting the budget and time constraints. At least he thought he could efficiently find the path minimizing the objective $5M^2 + 2T^2$. He tried modifying Dijkstra's algorithm as follows: If an edge e was labeled (m_e, t_e), he assigned it a weight $w_e = 5m_e^2 + 2t_e^2$, and ran Dijkstra's algorithm on the resulting weighted directed graph. Show that Arthur's algorithm may return incorrect results,

(b) Now provide an algorithm that solves Arthur's original problem in time polynomial in number of vertices n and number of edges m. Your algorithm should find the path that minimizes the objective $5M^2 + 2T^2$, while at the same time respecting the constraints $M \leq 500$ and $T \leq 1000$. Please describe a linear time algorithm, and justify its correctness and running time. (Hint: M and T are upper bounded by a constant number.)

Solution:

(a) Suppose there is a directed triangle graph of three nodes s, v, t with the following weights

$$e_{sv} = (0, 2), e_{vt} = (0, 2), e_{st} = (0, 3)$$

The modified Dijkstra's algorithm will return path $s - v - t$ but the optimal path should be $s - t$.

(b) For each vertex v, we construct a 501×1001 matrix M_v such that the (i, j)th entry of M_v is 1 if there is a path from Cambridge to v with total cost i and time j and 0 otherwise.

We can define $OPT(v, i, j)$ as the (i, j)th entry of matrix M_v.

Boundary condition: $OPT[v, i, j] = 1$ if $v =$ Cambridge, $i = 0, j = 0$.

We have the following relationship:

$$OPT(v, i, j) = \max_{(u,v)\in E} \{OPT(u, i - m_{(u,v)}, j - t_{(u,v)})\}$$

The relationship can be view as a propagation process in which u will update all its neighbors v when u get updated. We begin from Cambridge and work in a BFS fashion to fill in the matrices. Initially we only put Cambridge in a queue and as we propagate using the above relationship, we will add nodes whose matrix has been change into the queue and process until the queue is empty. Then we check for $M_{Berkeley}$ all entries with value 1 to find the minimum $5i^2 + 2j^2$.

There are n constant size matrices to fill. A node will propagate at most 501×1001 times which is $O(1)$. Each propagation takes $O(\text{degree of node})$ time. So the total filling time is $O(n + m)$. The final search for result takes constant time on a constant size matrix.