Homework 9

Nicholas Amoscato
naa46@pitt.edu

Josh Frey
jtf15@pitt.edu

September 18, 2013
CS 1510 Algorithm Design

1. **Dynamic Programming Problem 6:**

   Given an n-sided polygon $P = \langle v_1, v_2, \ldots, v_n \rangle$ where the vertices have been numbered sequentially in a clockwise order, let edge $e = \langle v_1, v_n \rangle$.

   Note that by definition of triangulation, the minimal triangulation of P must include a triangle with edge e. However, there are $n-2$ triangles $\{\langle v_1, v_2, v_n \rangle, \langle v_1, v_3, v_n \rangle, \ldots, \langle v_1, v_{n-1}, v_n \rangle\}$ that can be constructed with e as an edge. These possibilities are described by the inner for loop in the pseudocode below.

   Also note that any of these triangles $\langle v_1, j, v_n \rangle$ (where $v_1 < j < v_n$) split the original polygon into two smaller polygons $P_1 = \langle v_1, v_2, \ldots, v_j \rangle$ and $P_2 = \langle v_j, v_{j+1}, \ldots, v_k \rangle$. These polygons each have their own minimal triangulation that will be computed in a bottom-up fashion.

   A dynamic programming algorithm will construct an $n \times n$ table t where the entry $t[i][k]$ is the cost of the minimal triangulation of polygon $\langle v_i, v_{i+1}, \ldots, v_{k-1}, v_k \rangle$ constructed by the entries below and to the left of it. This is derived from the fact that $j > i$ and $j < k$. Specifically, each entry contains the sum of the perimeters of the triangles included in the minimal triangulation.

   Thus, the top right entry in the table $t[1][n]$ contains the cost of the minimal triangulation of our initial polygon.

	1	$\cdots$	j	$\cdots$	k	$\cdots$	n
1							
$\vdots$							
i				$\rightarrow$	$\boxed{t[i][j]}$		
$\vdots$					$\uparrow$		
j							
$\vdots$							
n							

Table 1: Table constructed by the dynamic programming algorithm

Note that when $j == i + 1$, the "triangle" constructed is simply a line. Thus, the perimeter of this is 0. This is our base case defined in the first for loop in the pseudocode below.

Let $c(x, y, z)$ be the perimeter of the triangle $\langle x, y, z\rangle$.

```
for $i = 1$ to $n - 1$ do
    $t[i][i + 1] = 0$
end for
for $i = n$ down to 1 do
    for $k = i + 2$ to $n$ do
        $t[i][k] = \infty$
        for $j = i + 1$ to $k - 1$ do
            $w = t[i][j] + t[j][k] + c(i, j, k)$
            if $w < t[i][k]$ then
                $t[i][k] = w$
                $v[i][k] = j$
            end if
        end for
    end for
end for
```

Note that in addition to storing the actual cost of each subproblem in table t, we also store the j value that produced the minimal triangulation in an additional table v. This is used to determine the actual triangles involved in the minimal triangulation after t has been constructed.

For example, with $v[1][n] = j$, we know that the triangle $\langle 1, j, n\rangle$ is a part of the minimal triangulation. Additionally, we are able to construct the two polygons split by this choice $P_1 = \langle v_1, v_2, \ldots, v_j\rangle$ and $P_2 = \langle v_j, v_{j+1}, \ldots, v_n\rangle$. Each of these polygons contains a triangle and two subsequently split polygons that are determined by the value of $v[1][j]$ and $v[j][n]$ respectively. This process would continue until all $n-2$ triangles have been defined.

2. **Dynamic Programming Problem 8:**

   Let T be a tree with n vertices $V = \{v_1, v_2, \ldots, v_n\}$ and integer weights on the edges. Let $e_{i,j} = e_{j,i}$ be the edge that connects v_i to v_j.

   The shortest unconstrained simple path can be found by placing each vertex as the root of a tree. If the root vetex knew the weight of the minimal path for all of its k child subtrees, the problem would be relatively straightforward:

   - Let $e_m = \min(e_1, e_2, \ldots, e_k)$ where e_i is the edge that connects the root vertex with the ith subtree.
   - Let $w_m = \min(w_1, w_2, w_k)$ where $w_i = w(i) + e_i$ and $w(i)$ is the weight of the ith subtree's minimal path.
   - If $e_m < w_m$, the smallest path would simply be e_m. Otherwise, the smallest path would be the smallest path of the mth subtree $+e_m$.

   In order to recover the minimal path in the mth subtree, we must strengthen the inductive hypothesis. Specifically, in addition to returning the weight of the minimal path, each subtree must also return the edge connecting the root node to that subtree.

   The algorithm described above provides a decent overview of a top-down (recursive) implementation; however, this problem should be implemented as a bottom-up dynamic programming algorithm in which the minimal path of the leaf nodes are computed before traversing up the tree in a reversed breadth-first search fashion.

   Let $bfs(V)$ return an array of the vertices in V ordered by level via a breadth-first search.

```
for $r = 1$ to $n$ do
    $bfs = bfs(V)$
    for $i = n$ down to 1 do
        if $i \neq r$ and $bfs[i]$ is only connected to one other vertex then
            $ssp[i] = 0$
            $sspe[i] =$ weight of edge connecting $v_i$
        else
            $ssp[i] = \infty, sspe[i] = \infty$
            for each child $v_c$ of $bfs[i]$ connected by $e_c$ do
                $ssp[i] = \min(ssp[i], e_c)$
                $sspe[i] = \min(sspe[i], bfs[v_c] + e_c)$
            end for
            **end if**
        **end for**
    **end for**
```

To construct the minimal path, the root r that has the minimum $ssp[1]$ should be used as a starting point. The value of $sspe[1]$ will then determine the appropriate edge to traverse next. This process continues until $sspe[i]$ equals an edge that has already been traveled (it is no longer adventageous to continue).