Homework 18

Nicholas Amoscato
naa46@pitt.edu

Josh Frey
jtf15@pitt.edu

October 11, 2013
CS 1510 Algorithm Design

1. **Dynamic Programming Problem 24:**

   The objective of this problem is to develop an algorithm that determines if one can partition a collection of n boxes $B_1, \ldots, B_n$ into two disjoint sub-collections S_1 and S_2 of equal weight where the weight of each box x_k is an integer from 1 to some constant L inclusive. Let $|S_1|$ and $|S_2$ be the cumulative weight of S_1 and S_2 respectively.

   We derive a dynamic programming algorithm by constructing a binary tree that enumerates all possible sub-collections S_1. For the purpose of this construction, we assume that all boxes exist in S_2 at the root of the tree; thus S_1 is empty. The tree is constructed level-by-level, iterating through the boxes $B_1, \ldots, B_n$ sequentially. Each node at level k of the tree has two children: the left child adds B_{k+1} to S_1 and removes it from S_2 while the right child does not add B_{k+1} to S_1 (and it stays in S_2).

   In its current form, the tree will have an exponential number of leaf nodes; thus, we introduce pruning rules as follows.

   (a) First, it is important to note that as we construct the tree, $|S_1|$ will never decrease from level k to level $k+1$. This comes from the fact that we are either (1) adding a box to S_1 or (2) not adding a box to S_1. Thus, $|S_1|$ will either increase or stay the same.

   With this said, if $|S_1|$ ever exceeds $|S_2|$ in a given node, the sub-collections enumerated in the node's subtrees will never be of equal weight. Therefore, we prune this node.

   (b) If there are two nodes at the same level k in which the weights of S_1 are equivalent, arbitrarily prune one node. This comes from the fact that all remaining

boxes $B_{k+1}, \ldots, B_n$ considered after level k will enumerate idential solutions.

With these pruning rules, we ensure that there are no more than $\frac{n \times L}{2}$ nodes on each of the n levels of the tree. That is, if all n boxes had the maximum weight of L, the cummulative sum of all of these boxes would be $n \times L$. However, by our first pruning rule, $|S_1|$ will never exceed $\frac{n \times L}{2}$. The second pruning rule ensures that there is at most one node for each unique cummulative sum $1, \ldots, \frac{n \times L}{2}$.

Let $M[k, w]$ be the sub-collection that contains at most k of the first k boxes with a cummulative sum of w. We assume each entry of M is initially undefined. (Note that if we only need to decide whether or not a solution is feasible, $|S_2|$ could be stored in each table entry as opposed to the entire sub-collection.)

The dynamic programming algorithm is described in pseudocode below:

```
|S_2| = Σ_{i=1}^{n} x_i                          ▷ initialize weight of S_2
M[1, x_1] = B_1                                  ▷ there must be at least one box in S_1
for k = 1 to n do                                ▷ for each level and box B_k
    |S_2| = |S_2| − x_k                          ▷ update the weight of S_2
    for w = 1 to (n×L)/2 do                      ▷ for each possible cummulative weight
        if M[k, w] is defined then               ▷ if there is a sub-collection
            if w + x_k ≤ |S_2| then              ▷ first pruning rule
                M[k + 1, w + x_k] = M[k, w] + B_k ▷ move B_k from S_2 to S_1
            end if
            M[k + 1, w] = M[k, w]                ▷ keep B_k in S_2
        end if
    end for
end for
```

Clearly the algorithm runs in $O(n^2 L)$ time.

If a solution to this problem exists, it will be in the last level of the matrix $M[n, w]$.

```
for w = 1 to (n×L)/2 do                          ▷ for each possible weight of S_1
    if w == Σ_{B_i ∉ M[n,w]} x_i then            ▷ if |S_1| == |S_2|
        solution exists with S_1 = M[n, w]
    end if
end for
solution does not exist
```

2. **Reduction Problem 1:**

Theorem: If there is an $O(n^2)$ algorithm for multiplying two n by n lower triangle matrices then there is an $O(n^2)$ time algorithm for multiplying two arbitrary n by n matrices.

Proof: Let MM be the problem of multiplying two arbitrary n by n matrices. Let $LTMM$ be the problem of multiplying two $n \times n$ lower triangle matrices. In order to prove the theorem we must show that $MM \leq LTMM$. If we prove this reduction is true, we will prove that there is an algorithm to solve MM that lacks only code for $LTMM$.

Thus, we define an $O(n^2)$ algorithm for MM where $n \times n$ input matrices A and B are transformed to $3n \times 3n$ lower triangle matrices C and D which will be used as input to $LTMM$. Note that the placement of A and B in C and D ensure that $A \times B$ will be in the bottom left of the resultling matrix.

problem $MM(A, B)$
$C = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & A & 0 \end{bmatrix}$ ▷ create LTM from A in $O(n^2)$
$D = \begin{bmatrix} 0 & 0 & 0 \\ B & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ ▷ create LTM from B in $O(n^2)$
$P = LTMM(C, D)$ ▷ compute LTMM of $C \times D$ in $O(n^2)$
for $i = 1$ to n **do** ▷ for each row
  for $j = 1$ to n **do** ▷ for each column
    $Q[i, j] = P[2n + i, j]$ ▷ store bottom left of P in $O(n^2)$
  end for
end for
return Q

As we can see, based on the transformation of input and transformation of outputs being from arbitrary matrecies to lower triangle matrecies being $O(n^2)$, if we are able to solve lower triangle matrix multiplication in $O(n^2)$ time, then matrix multiplication can be solved in $O(n^2)$ times.

3. **Reduction Problem 4:**

Theorem: If you can solve the minimum Steiner tree problem in linear time, then you can sort n numbers in linear time.

Proof: Let $Sort$ be the algorithm for sorting n numbers $x_1, \ldots, x_n$ by calling MST, which is the minimum Steiner tree problem. We show that $Sort \leq MST$ by tranforming our inputs and outputs as follows.

Let the n numbers used as input to $Sort$ be transformed into n points on the Cartesian plane where each point has an x-value of 0 and a y-value of x_i.

Assuming MST returns n adjacency lists for each of the n points, we start at the adjacency list for the point a with the smallest y coordinate. This point is obviously

the lowest point on the Cartesian plane, and it will only be connected to one point b, the point directly above it (which has a greater y-value). Point b will be connected to two points: obviously point a and a second point c that is directly above b.

Thus, traversing the adjacency lists from the point with the smallest y coordinate will output the points in order of increasing y coordinate.

problem Sort$(x_1, \ldots, x_n)$
for $i = 1$ to n **do**
    v_i=$(0, x_i)$ ▷ transform numbers to points in $O(n)$
end for
$T = MST(v_1, \ldots, v_n)$ ▷ find minimum Steiner tree in $O(n)$
(a, b) = adjacency list in T where point a has smallest y coordinate ▷ $O(n)$
$x_1 = a_y$ ▷ x_1 = y coordinate of point a
for $i = 2$ to n **do** ▷ $O(n)$
    $x_i = b_y$ ▷ x_i = y coordinate of point b above a
    $(a, b) = (b, c) \in T \mid c \neq a$ ▷ find point c above b
end for
return $x_1, \ldots, x_n$

Clearly the above algorithm shows that $Sort \leq MST$ in linear time.