Homework #9
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2016

Due on: Friday, Apr 29, 11:59pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: On blackboard, under student assessment
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

May 6, 2016

Problem 1 (20 points)

Let $G = (V, E)$ be a flow network with the capacities of all edges being zero or one. Recall that we consider networks where $(a, b) \in E$ implies $(b, a) \notin E$. Provide a formal proof for the claim that, (e.g., by induction on the number of iterations), that every iteration, except the last iteration, of the Ford-Fulkerson algorithm will increase the flow by exactly one unit and prove that the number of iterations is at most $|V|$.

We prove the following claim by induction on the number of the iterations M:

1. During the M-th iteration the flow will increase from $M - 1$ to M.

2. For every edge $(a, b) \in E$ it is true that $f(a, b) \in \{0, 1\}$.

$M = 1$: Initially the flow is 0. If there is a path from s to t in the residual network then all residual capacities are 1 on that path. Thus, the flow will be increased by exactly one unit and $|f| = 1$. Also, since we increase the flow by one on a single path then for every edge $(a, b) \in E$ it is true that $f(a, b) \in \{0, 1\}$.

$M \to M+1$: Assume that this is not the last iteration and thus the flow can be increased. Thus, there must be a path from s to t in the residual network. By induction, the current value of the flow is M. By the theorem that we proved in class, the flow will increase by $c_f(P)$ where $c_f(P)$ is the residual capacity of the path P that we found during the $(M+1)$-th iteration. By the second claim, $f(a,b) \in \{0,1\}$ and thus $c_f(a,b) \in \{0,1\}$. Thus, $c_f(P) = 1$. Thus, the flow will be increased by exactly one unit and $|f| = M+1$.

To prove the second part, consider $(a,b) \in P$. If $(a,b) \in E$ then it must be the case that $f(a,b) = 0$ before augmenting the flow with P. (otherwise by induction $f(a,b) = 1$ and $(a,b) \notin E_f$, i.e., not in the residual network) Thus, $f(b,a) = 0$ after augmenting the path. Other values did not change. Thus, the claim is correct.

Proof that there are at most $|V|$ iterations: Each edge has capacity at most 1, so each edge that is connected to t can carry a flow of at most 1 into t. Therefore, the maximum flow into t in no greater than the number of vertices with an edge to t. There are $|V|-1$ edges, besides t itself, that could be connected to t. Therefore, the maximum possible flow in this network is $|V|-1$. Because the flow increases by one with each iteration of the algorithm, it takes at most $|V|-1$ iterations to find the max flow, plus an additional iteration to verify it has found the maximum flow. Therefore, there are at most $|V|$ iterations.

Problem 2 (20 points)

Given a set of integers $S = \{x_1, \ldots, x_n\}$ and integer A such that $A \le n^k$, where $k = O(1)$, design a dynamic programming algorithm that checks if there exists a subset T of S such that $\sum_{x \in T} x^k = A$. Your algorithm should have a running time that is polynomial with respect to n.

The following algorithm assumes that all inputs are positive integers.

ALGORITHM

```
1. $subset$ = new zero-indexed **boolean** matrix of size $(A+1)$ x $(n+1)$
   /* $subset[i][j]$ is True if and only if some subset of $\{x_1, \ldots, x_j\}$ sums to $i$
   */

2. **for** $j = 0 \to n$

        $subset[0][j]$ = True
        /* A subset of any set can sum to 0 */

3. **for** $i = 1 \to A$
```

```
        $subset[i][0]$ = False
        /* A subset of an empty set cannot sum to a positive integer */

4. for $i = 1 \to A$
        for $j = 1 \to n$
            $subset[i][j] = subset[i][j-1]$
            /* If a subset of $\{x_1, \ldots, x_{j-1}\}$ sums to $i$, then the same subset of
            $\{x_1, \ldots, x_j\}$ sums to $i$ */
            if $i \geq x_j$ and $subset[i - x_j][j-1]$ is True
                $subset[i][j]$ = True
                /* If a subset of $\{x_1, \ldots, x_{j-1}\}$ sums to $i - x_j$, then a subset
                of $\{x_1, \ldots, x_j\}$ sums to $i$. */

5. return $subset[A][n]$
```

CORRECTNESS This version of the subset-sum problem exhibits optimal substructure, because some subset of the positive numbers $\{x_1, \ldots, x_n\}$ sums to A, than one of two cases must be true.

1. x_n is not used in the subset, and some subset of $\{x_1, \ldots, x_{n-1}\}$ sums to A
2. x_n is used in the subset, and some subset of $\{x_1, \ldots, x_{n-1}\}$ sums to $A - x_n$

Because of this, we can use a dynamic programming table to calculate the answer to the problem from its subproblems:

Induction Hypothesis: For values of i and j previously examined, $subset[i][j]$ is True if and only if some subset of $\{x_1, \ldots, x_j\}$ sums to i

Base Case: The top row of the matrix must all be True, because a subset of any set can sum to 0. The leftmost column of the matrix (except for $subset[0][0]$) must all be False, because a subset of an empty set cannot sum to a positive integer.

Induction Step: For any given i and j, some subset of $\{x_1, \ldots, x_j\}$ sums to i if and only if one of the following cases holds:

1. some subset of $\{x_1, \ldots, x_{j-1}\}$ sums to i
2. some subset of $\{x_1, \ldots, x_{j-1}\}$ sums to $i - x_j$

Our algorithm marks $subset[i][j]$ as True if and only if one of the above cases holds, therefore the induction hypothesis remains True.

RUNTIME Assuming that the initialization of the matrix (and the return statement) are constant time, the runtime of the algorithm is equal to the time spent filling each value in the matrix. Clearly, each value in the matrix is filled only once, and it take constant time to decide what value goes in each position of the matrix. Therefore our total runtime is

$$T(n) = O\left((A+1)(n+1)\right) \tag{1}$$

$$T(n) = O(An) \tag{2}$$

$$T(n) = O\left(n^k n\right) \tag{3}$$

$$T(n) = O\left(n^{k+1}\right) \tag{4}$$

which is polynomial in n.

Problem 3 (5 points)

Let T be a red-black tree. Let x and y be two nodes in T such that $bh(x) \leq bh(y) \leq 5bh(x)$. Let P_1 be a longest simple path from x to a descendant leaf and let P_2 be a shortest simple path from y to a descendant leaf. What is the relation between the lengths of P_1 and P_2? Prove your answer.

In a red-black tree, the shortest possible path from a node to a leaf will include only black nodes, and the longest possible path from a node to a leaf will alternate between red and black nodes. Therefore, a path from a node x to a leaf will have a length between $bh(x)$ and $2bh(x)$. Therefore, from the problem definition and the definition of red-black trees, we have the following three equations:

$$bh(x) \leq bh(y) \leq 5bh(x) \tag{5}$$

$$bh(x) \leq length(P_1) \leq 2bh(x) \tag{6}$$

$$bh(y) \leq length(P_2) \leq 2bh(y) \tag{7}$$

From equation (5) we know that $2bh(y) \leq 10bh(x)$ and $bh(x) \leq bh(y)$, and we can combine this with equation (7) to get:

$$bh(x) \leq length(P_2) \leq 10bh(x) \tag{8}$$

It follows from equation (6) that:

$$\frac{bh(x)}{2} \leq \frac{length(P_1)}{2} \leq bh(x) \tag{9}$$

$$10bh(x) \leq 10length(P_1) \leq 20bh(x) \tag{10}$$

We can combine (8) with (9) and (10) to get:

$$\frac{length(P_1)}{2} \leq length(P_2) \leq 10length(P_1) \tag{11}$$

Problem 4 (5 points)

Let D be a data structure that operates on integers. D supports two operations $\alpha(i)$ and $\beta(i)$, where i is an integer. Every $\alpha(i)$ operation costs $O(i)$. Any $\beta(i)$ operation takes time $O(1)$. Assume that every $\alpha(i)$ is followed by at least i operations of type $\beta(i)$. That is, if S is a sequence of m operations then every operation $\alpha(i)$ from S is followed by at least i operations $\beta(i)$. Use direct computations to prove that the amortized time of both operations is $O(1)$.

Assume that the m operations in question use the $\alpha(i)$ operation K times, to process the K integers, $i_1, i_2, \ldots, i_{K-1}, i_K$. In this case, our m operations must include exactly K calls to $\alpha(i)$ and at least $\sum_{k=1}^{K} i_k$ calls to $\beta(i)$, as required by the problem definition. We may also have x additional calls to $\beta(i)$, beyond what is required by the problem definition, where x may or may not be zero. This means that

$$m = K + \sum_{k=1}^{K} i_k + x \tag{12}$$

or,

$$m - K - x = \sum_{k=1}^{K} i_k \tag{13}$$

Now, assume without loss of generality that the running time of the α and β operations is:

$$T(\alpha(i)) \leq c_\alpha i \tag{14}$$

$$T(\beta(i)) \leq c_\beta \tag{15}$$

for some constants c_α and c_β. I will refer to the total running time of the m operations as $T(m)$. We can calculate $T(m)$ as the sum of the costs of K calls to α and $\sum_{k=1}^{K} i_k$ calls to β, plus an additional x calls to β:

$$T(m) = \sum_{k=1}^{K} (T(\alpha(i_k))) + \sum_{k=1}^{K} (i_k T(\beta(i_k))) + \sum_{i=1}^{x} (T(\beta(i))) \tag{16}$$

$$T(m) \leq \sum_{k=1}^{K} (T(\alpha(i_k)) + i_k T(\beta(i_k))) + \sum_{i=1}^{x} (T(\beta(i)) \tag{17}$$

$$T(m) \leq \sum_{k=1}^{K} (c_\alpha i_k + c_\beta i_k) + \sum_{i=1}^{x} c_\beta \tag{18}$$

$$T(m) \leq (c_\alpha + c_\beta) \sum_{k=1}^{K} (i_k) + c_\beta \, (x) \tag{19}$$

Using equation (13), this simplifies to

$$T(m) \leq (c_\alpha + c_\beta) \, (m - K - x) + c_\beta \, (x) \tag{20}$$

$$T(m) \leq m \, (c_\alpha + c_\beta) \tag{21}$$

Let T' be the amortized cost of one of our m operations. Then

$$T' = \frac{T(m)}{m} \leq \frac{m \, (c_\alpha + c_\beta)}{m} = c_\alpha + c_\beta \tag{22}$$

Thus, the amortized cost of a single operation is no more than $c_\alpha + c_\beta$, a constant as desired.