Homework 4 SOLUTIONS
Fundamental Algorithms, Fall 2004, Professor Yap

Due: Thu Nov 18, in class
SOLUTION PREPARED BY Instructor and T.A.s Ariel Cohen and Vikram Sharma

INSTRUCTIONS:

- Please read carefully.
- General remark: we do encourage students to work in groups for discussion. However, when you finally write up the solutions, they *must* represent your individual work.

---

1. (30 Points) DFS
   Note that the graph in part (a) is a bigraph, but parts (b) and (c) it is a digraph.
   (a) Suppose T is the DFS Tree for a connected bigraph G. Let $u{-}v$ be a non-tree edge. Prove that either u is an ancestor of v or vice-versa.
   (b) Let $G = (V, E)$ be a digraph. Suppose we call $DFS((V, E; s_0))$ in which INIT$((V, E; s_0))$ colors s_0 as `seen`, but every node as `unseen`. Prove every vertex that is reachable from s_0 will be seen. NOTE: vertex u is **reachable** from s_0 iff there is a path from s_0 to u.
   (c) Let $G = (V, E; s_0)$ be a digraph. Fill in the shell subroutines of the DFS Algorithm so that it correctly classifies every edge of G into tree, back, forward, cross and unseen edges. You must briefly argue why your algorithm is correct.

   SOLUTION:

   (a) This is a consequence of the Unseen Path Lemma: suppose that u is visited before v. Then at the time we first see u, there is an unseen path from u to v (since $u{-}v$ is an edge). The Lemma tells us that v would become a descendent of u. Hence the edge $u{-}v$, if it is not a tree-edge, will connect a vertex to its ancestor.

   (b) Again, this is an immediate consequence of the Unseen Path Lemma.

   (c) Let us slightly change the notation in the Lecture Notes so that the PREVISIT subroutine takes a second argument, PREVISIT(v, u) means we are "previsiting vertex v from u". Now, we can use this subroutine to classify all our edges. In the GLOBAL_INIT, we mark all edges as "unseen edge". We also initialize `lastTime`$[v]$ to ∞ for all $v \in V$. In PREVISIT(v, u), we perform the following tests:

   > if v is unseen, mark $u{-}v$ as "tree edge"
   > else if `firstTime`$[v]$ $>$ `firstTime`$[u]$, mark $u{-}v$ as "forward edge"
   > else if `lastTime`$[v] = \infty$, mark $u{-}v$ as "back edge"
   > else mark $u{-}v$ as "cross edge"

   These decisions are justified by the properties we prove in Lemma 7 in the Lecture IV.

2. (25 Points) Radius and Diameter
   (a) Let $G = (V, E)$ be a connected bigraph. For any vertex $v \in V$ define

   $$\text{radius}(v, G) := \max_{u \in V} \delta(v, u)$$

   where $\delta(v, u)$ is the link distance from u to v. The *center* of G is the vertex v_0 such that radius(v_0, G) is minimized. We call radius(v_0, G) the *radius* of G and denote it by radius(G). Define the *diameter* diameter(G) of G to be the maximum value of $\delta(u, v)$ where $u, v \in V$. Prove that

   $$\text{diameter}(G) \geq \cdot\text{radius}(G) \geq \lceil \text{diameter}(G)/2 \rceil .$$

(b) Show that for every natural number n, there are graphs G_n and H_n such that $n = \text{radius}(G_n) = \text{diameter}(G_n)$ and $n = \text{radius}(H_n) = \lceil \text{diameter}(H_n)/2 \rceil$.
(c) Give an efficient algorithm to compute the diameter of a undirected tree (i.e., connected acyclic undirected graph). Please use the "shell" subroutines for BFS or DFS. What is the complexity of your algorithm?

SOLUTION:

(a) Suppose $\delta(u, v) = \text{diameter}(G)$ and c is a center. Then clearly, $\delta(u, v) \geq \delta(c, w)$ for all w. Hence $\delta(u, v) \geq \text{radius}(c, G) = \text{radius}(G)$.

To see the other inequality, notice that $\text{radius}(G) \geq \lceil \text{diameter}(G)/2 \rceil$ is equivalent to $\text{radius}(G) \geq \text{diameter}(G)/2$ (because $\text{radius}(G)$ must be integer). This is in turn equivalent to

$$2\text{radius}(G) \geq \text{diameter}(G). \tag{1}$$

To see Equation (1), notice that if $\text{diameter}(G) = \delta(u, v)$ and c is a center, then $\delta(u, v) \leq \delta(c, u) + \delta(c, v)$ (triangular inequality), and so $\delta(u, v) \leq 2\text{radius}(c, G) = \text{radius}(G)$.

(b) For any natural number n, let G_n be the "ring graph" on vertices $V = \{1, 2, \ldots, 2n\}$ and the edge set is $E = \{i{-}i + 1 : i = 1, \ldots, 2n - 1\} \cup \{2n{-}1\}$. Clearly,

$$\text{diameter}(G_n) = \text{radius}(G_n) = n.$$

Let H_n be the "line graph" on vertices $V = \{1, 2, \ldots, n + 1\}$ and edges $E = \{i{-}i + 1 : i = 1, \ldots, n\}$. Clearly,

$$\text{diameter}(H_n) = n, \text{radius}(H_n) = \lceil n/2 \rceil .$$

Please note that there are other valid soultions.

(c) Suppose we run the DFS algorithm on the tree starting from any node s_0. Let inductively, each node u of the DFS Tree compute the values $u.d$ and $u.h$ where $u.d$ is the diameter of the subtree rooted at u and $u.h$ is the height of the subtree rooted at u. Then we can compute $u.d$ and $u.h$ recursively as follows:

$$u.h = 1 + \max\{v.h : v \text{ is a child of } u$$

and

$$u.d = \max\{M_1, M_2, M_3\}$$

where

$$\begin{aligned} M_1 &= \max\{v.d : v \text{ is a child of } u\}, \\ M_2 &= \max\{2 + v.h + v'.h : v, v' \text{ are distinct children of } u\}, \\ M_3 &= \max\{1 + v.h : v \text{ is a child of } u\}\} \end{aligned}$$

To implement this, we define the POSTVISIT(u) subroutine as follows: first, we need to know who is the parent of each node. Let $Parent[v]$ be a new array to record this information. In VISIT(v, u), we will set $Parent[v] = u$. In POSTVISIT(u), we do:

```
Initialize M1, M2, M3 to 0 and H to −1.
For each v adjacent to u
    If v ≠ Parent[u],
        u.h = max{u.h, v.h + 1}
        M1 = max{M1, v.d}
        M2 = max{M2, 2 + H + v.h}
        M3 = max{M3, 1 + v.h}
        H = max{H, v.h}
u.d = max{M1, M2, M3}
```

Almost none of the students got a full credit for this question. Please note that you had to non-trivial steps in your algorithms.

3. (20 Points) Greedy
(a) Give an algorithm that computes the optimal bin packing for an input (M, w) where $w = (w_1, \ldots, w_n)$ are positive weights. You must reduce this to linear bin packing, by considering all $n!$ permutations of $w_1, \ldots, w_n$.
(b) What is the complexity of your algorithm?
(c) Improve your complexity by algorithm by considering only $(n-1)!$ permutations. What is the new complexity of your algorithm?
(d) Further improve upon (c) by using $2(n-2)!$ permutations only.

SOLUTION:

Most students did this question correctly.

(a) Compute all the possible permutations for w. Go over the permutations and execute the linear algorithm for bin packing as seen in class. Pick the packing with the minimum number of bins.

```
optimalBin(M,w)
    p[n!]=calculatePermutations(w)
    bestBinPacking := NULL
    minNumberOfBins := |w|
    for(i = 1; i <= n!; i++)
        localBinPacking := linearBinPackingAlg(p[i]) ◁ the algorithm seen in class
    localMinNumberOfBins := numberOfBins(localBinPacking)
    if(localMinNumberOfBins ¡ minNumberOfBins)
        bestBinPacking := localBinPacking
        minNumberOfBins := localMinNumberOfBins
    return bestBinPacking
```

(b) Complexity: $n!$ times the number of executing linearBinPackingAlg which is $n! * n$

(c) Fix the first weight such that it will always be in the first bin and only then calculate the permutations. Continue with the same algorithm from (a). There are $(n-1)!$ permutations and the linearBinPackingAlg is only executed $n-1$ times, therefore the complexity is $(n-1)! * (n-1)$

(d) Fix the first two weights, considering the following: 1. They are fixed at different bins, or 2. They are fixed at the same bin Hence, there are two cases when computing the $(n-2)!$ permutations whereas the linearBinPackingAlg is executed only $n-2$ times. Therefore the complexity is $2(n-2)! * (n-2)$.

4. (25 Points) Huffman
In our Huffman tree algorithm, we represented the Huffman tree as a binary tree. Now consider a more compact representation of a Huffman tree T by exploiting its Sibling property: suppose T has $k \geq 1$ leaves. Each of its $2k-1$ nodes is identified by its rank, *i.e.*, a number from 0 to $2k-2$. Hence node i has rank i. We use two arrays

$$W[0..2k-2], \qquad L[0..2k-2]$$

of length $2k-1$ where $W[i]$ is the weight of node i, and $L[i]$ is the left child of node i. So $L[i]+1$ is the right child of node i (by the Sibling Property). In case node i is a leaf, we let $L[i]$ denote the **canonical code** for the letter that it represents in Σ. I.e., view Σ as a subset of a universal set U where $U \subseteq \{0,1\}^N$. In reality, U might be the set of ASCII characters and $N = 7$. The transmitter and receiver both know this global parameter N and the set U.

(a) Please implement the RESTORE(u) subroutine in detail, using the above representation. In order to ensure sufficient details, we ask you to use either `C`, `C++` or `Java`.
(b) Illustrate your algorithm above by showing how to transmit our familiar string `hello world!`.

5. (0 Points)
   The problems below carry no credit. Please do not hand such problems; they are for your own practice. We strongly recommend that you do them.

6. (0 Points) Greedy
   Do Exercises 1.1, 1.5.

7. (0 Points) Huffman
   Do Exercises 2.1, 2.2 and 2.3 in Lecture V.