Homework 4 Solutions
Fundamental Algorithms, Fall 2005, Professor Yap

Due: Wed Nov 30, in class.
SOLUTION PREPARED BY Instructor and T.A.s

INSTRUCTIONS:

- Please try to give clear but brief answers. Usually, the length of your answer will go down in proportion to the amount of thought given to the problem.

Question 1

(15 Points)
In this question, we are asking for two numbers. BUT you must briefly indicate how your computations are done to get credit:
(a) What is the length of the Huffman code of the string "`Hello, world!`"?
(b) How many bits would be transmitted by the Dynamic Huffman code algorithm in sending the string "`Hello, world!`"? NOTE: assume that we transmit an 8-bit ASCII code with each newly discovered character.

SOLUTION (a) 42. One possible Huffman tree for the string is Figure 1. The length of the Huffman code for the string is just the sum of the values in the internal nodes of the tree.

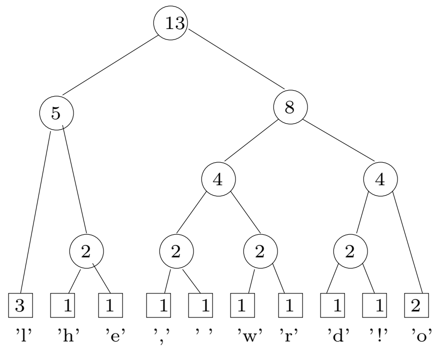

Figure 1: Huffman code for "`Hello, world!`"

(b) Table 1 shows the breakup of the bits sent; the first column shows the string sent and the second shows the corresponding bits. Thus we send 120 bits in total. The final tree looks like Figure 2. .

♠

Question 2

(15 Points)
Exercise 1.6 in Lecture 5. Improve the bin packing upper bound by another factor of n. MORE HINT: You would like to consider only $(n-2)!$ permutations. But how can you fix any two of the inputs weights?

SOLUTION Fix any two arbitrary weights w_1, w_2. The algorithm will do the following:
1) Find the optimal solution for the weights $w_1 + w_2, w_3, \ldots, w_n$. This can be done in time $O(n(n-2)!) = O((n/e)^{n-0.5})$, using the previous trick.
2) Find the optimal solution for the weights $w_1, w_3, \ldots, w_n, w_2$, where we only permute the weights $w_3, \ldots, w_n$. This take the same time as (1).
3) Output the better of the above two solution.

Thus the overall algorithm takes $O((n/e)^{n-0.5})$ time.

♠

Empty string + 'h'	0+8
Code for * + 'e'	1+8
Code for * + 'l'	2+8
Code for 'l'	3
Code for * + 'o'	3+8
Code for * + ','	4+8
Code for * + ' '	4+8
Code for * + 'w'	4+8
Code for 'o'	3
Code for * + 'r'	4+8
Code for 'l'	2
Code for * + 'd'	5+8
Code for * + '!'	5+8

Table 1: Breakup of the bits sent

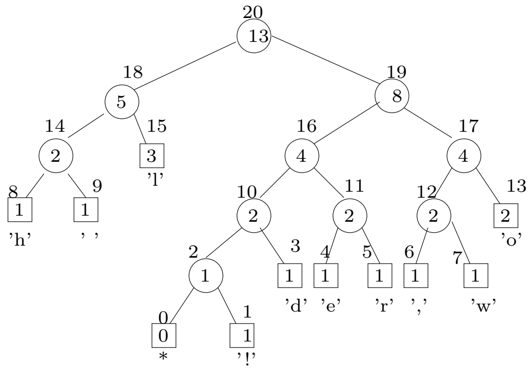

Figure 2: Dynamic Huffman tree corresponding to "`Hello, world!`"

Question 3

(10 Points)
In the text, we prove that for any frequency function f, there is an optimal Huffman code tree in which there is a deepest pair of leaves whose frequencies are the least frequent and the next to least frequent. Is the converse true? The converse says that if T is an optimum code tree for f, then there must be a deepest pair whose frequencies are least frequent and next to least frequent. Prove it or show a counter example.

SOLUTION False. For the input frequencies $(1, 1, 1, 1, 2, 2, 2)$, Figure 3 shows two optimal trees; however, unlike (a), which is the Huffman tree for the given frequencies, (b) does not satisy the above mentioned property. .

♠

Question 4

(35 Points)
We develop Boruvka's approach to MST: for each vertex v, pick the edge $(v{-}u)$ that has the least cost among all the nodes u that are adjacent to v. Let P be the set of edges so picked.
(a) Show that $n/2 \leq |P| \leq n$. Give general examples to show that these two extreme bounds are achieved for each n.
(b) Show that if the costs are unique, P cannot contain a cycle. What kinds of cycles can form if weights are not unique?
(c) Assume edges in P are picked with the tie breaking rule: among the edges $v{-}u_i$ $(i = 1, 2, \ldots)$ adjacent to

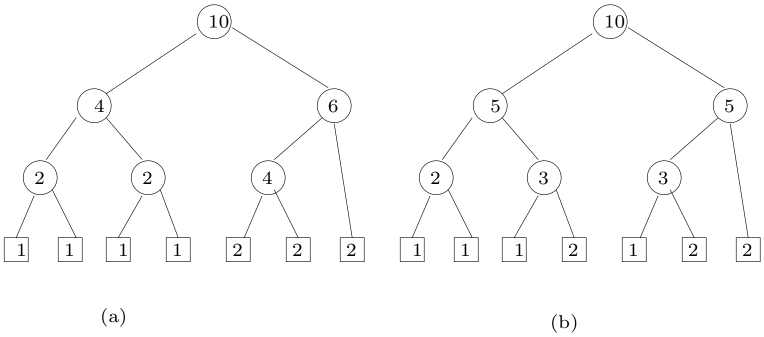

Figure 3: Two optimal trees for the frequencies $(1, 1, 1, 1, 2, 2, 2)$

v that have minimum cost, pick the u_i that is the smallest numbered vertex (assume vertices are numbered from 1 to n). Prove that P is acyclic and has the following property: if adding an edge e to P creates a cycle Z in $P + e$, then e has the maximum cost among the edges in Z.
(d) For any costed bigraph $G = (V, E; C)$, and $P \subseteq E$, define a new costed bigraph denoted G/P as follows. First, two vertices of V are said to be **equivalent modulo** P if they are connected by a sequence of edges in P. For $v \in V$, let $[v]$ denote the equivalence class of v. The vertex set of G/P is $\{[v] : v \in V\}$. Next, $([u]{-}[v])$ is an edge of G/P iff there exists an edge $(u'{-}v') \in E$ where $u' \in [u]$ and $v' \in [v]$. Moreover, the cost of $([u]{-}[v])$ is defined as $\min\{C(u', v') : u' \in [u], v' \in [v], (u'{-}v') \in E\}$. Moreover, we can pick another set P' of edges in G/P using the same rules as before. This gives us another graph $(G/P)/P'$, etc. We can continue this until the reduced graph has only 1 vertex. Briefly say how this leads to an MST algorithm.
(e) How many phases does the algorithm in (d) take, where one "phase" corresponds to computing of the graph G/P from G?

SOLUTION (a) $|P| \le n$ follows since we make n choices for edges. If P is acyclic, then we must get $|P| \le n - 1$. This bound is achieved if G is a linear graph whose edge weights are distinct and in sorted order. At most two choices might result in the same edge; hence $|P| \ge n/2$. This bounds is achieved using the linear graph again, where the weights are alternatively positive and negative.

(b) The cycles can only be formed by edges with the same weight.

(d)

The following are the steps for computing G/P:
(1) Let $NN[u]$ be the vertex v such that $(u, v) \in P$. Also, let $CN[u] = C(u, v)$. Thus, NN is a representation of the set P. NOTE: we can compute this following the tie breaker rule to ensure P is acyclic. THIS TAKES $O(m + n)$ time using DFS.
(2) Compute an adjacency list representation of the graph (V, P) from information in (1) THIS TAKES $O(n)$ time using DFS.
(3) Compute the connected components of (V, P), where $CC[u]$ is the component number of u. THIS TAKES $O(n)$ time using DFS.
(4) Compute the edges of G/P in $O(n^2)$ time, using the adjacency matrix, $M(CC[u], CC[v]) = 1$ iff there is an edge.
(5) At the same time, we can compute the edge cost of $(CC[u], CC[v])$.

♠

Question 5

(30 Points)
A **vertex cover** for a bigraph $G = (V, E)$ is a subset $C \subseteq V$ such that for each edge $e \in E$, at least one of its two vertices is contained in C. A **minimum vertex cover** is one of minimum size. Here is a greedy algorithm to finds a vertex cover C:

1. Initialize C to the empty set.
2. Choose from the graph a vertex v with the largest out-degree. Add vertex v to the set C, and remove vertex v and all edges that are incident on it from the graph.
3. Repeat step 2 until the edge set is empty.
4. The final set C is a vertex cover of the original graph.

(a) Show a graph G, for which this greedy algorithm fails to give a minimum vertex cover. HINT: An example with 7 vertices exists.
(b) Let $\mathbf{x} = (x_1, \ldots, x_n)$ where each x_i is associated with vertex $i \in V = \{1, \ldots, n\}$. Consider the following set of inequalities:

- For each $i \in V$, introduce the inequality
$$0 \leq x_i \leq 1.$$
- For each edge $(i, j) \in E$, introduce the inequality
$$x_i + x_j \geq 1.$$

If $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ satisfies these inequalities, we call $\mathbf{a}$ a **feasible solution**. If each a_i is either 0 or 1, we call $\mathbf{a}$ a 0-1 feasible solution. Show a bijective correspondence between the set of vertex covers and the set of 0-1 feasible solutions. If C is a vertex cover, let $\mathbf{a}^C$ denote the corresponding 0-1 feasible solution.
(c) Suppose $\mathbf{x}^* = (x_1^*, \ldots, x_n^*) \in \mathbb{R}^n$ is a feasible solution that minimizes the function $f(\mathbf{x}) = x_1 + x_2 + \cdots + x_n$, i.e., for all feasible $\mathbf{x}$,
$$f(\mathbf{x}^*) \leq f(\mathbf{x}).$$
Call $\mathbf{x}^*$ an **optimum vector**. Construct a graph $G = (V, E)$ where $\mathbf{x}^*$ is not a 0-1 feasible solution. HINT: you do not need many vertices ($n \leq 4$ suffices).
(d) Given an optimum vector $\mathbf{x}^*$, define set $\overline{C} \subseteq V$ as follows: $i \in \overline{C}$ iff $x_i \geq 0.5$. Show that $\overline{C}$ is a vertex cover.
(e) Suppose C^* is a minimum vertex cover. Show that $|\overline{C}| \leq 2|C^*|$. HINT: what is the relation between $|\overline{C}|$ and $f(\mathbf{x}^*)$? Between $f(\mathbf{x}^*)$ and $|C^*|$? REMARK: using Linear Programming, we can efficiently find a optimum vector $\mathbf{x}^*$.

SOLUTION

(a) Consider the graph with vertex set $\{a, a', b, b', c, c', d\}$ and edge set $\{a'{-}a, b'{-}b, c'{-}c, a{-}d, b{-}d, c{-}d\}$. Our algorithm will first choose the vertex d of degree 3. The final vertex cover has size 4, but the optimum has size 3.

(b) For any 0-1 feasible solution, $\mathbf{a} = (a_1, \ldots, a_n)$ we have a corresponding set $C = \{i \in V : a_i = 1\}$. This is easily seen to be a vertex cover. Conversely, for any vertex cover C, we have a corresponding $\mathbf{a} = (a_1, \ldots, a_n)$ which is a 0-1 feasible solution.

(c) Consider the graph with $V = \{a, b, c\}$ and $E = \{a{-}b, b{-}c, c{-}a\}$. Then we have the inequalities $x_a + x_b \geq 1$, $x_b + x_c \geq 1$ and $x_c + x_a \geq 1$. Then $f(x_a, x_b, x_c) = x_a + x_b + x_c$. If $\mathbf{x} = (x_a, x_b, x_c)$ is a 0-1 vector that arise from a vertex cover, then $f(\mathbf{x}) \geq 2$. On the other hand, if $\mathbf{x}^* = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ then $f(\mathbf{x}^*) = 1.5$. It is not hard to see that $\mathbf{x}^*$ is optimum: if $x_a < \frac{1}{2}$, then $x_b > \frac{1}{2}$ and $x_c > \frac{1}{2}$, and $f(\mathbf{x}^*) > 1.5$.

(d) Note that for each edge (i, j), either $i \in \overline{C}$ or $j \in \overline{C}$; since $0 \leq x_i, x_j \leq 1$ and $x_i + x_j \geq 1$ implies that one of x_i or x_j is greater than 0.5. Hence C^* is a vertex cover.

(e) Let C^* be a minimum vertex cover. Then we know from (b) that $|C^*| = f(\mathbf{a}^{C^*}) \geq f(\mathbf{x}^*)$. On the other hand, $2f(\mathbf{x}^*) = \sum_i 2x_i^* \geq \sum_i \overline{x}_i = |\overline{C}|$. The last inequality follows from $2x_i^* \geq \overline{x}_i$. To see this: if $x_i^* \geq 0.5$, then clearly $2x_i^* \geq 1 = \overline{x}_i$, otherwise, $2x_i^* \geq 0 = \overline{x}_i$.

♠

Question 6

(0 Points) DO NOT HAND THIS IN.
Exercise 3.6. Construct the Huffman Tree in linear time when the frequencies are already sorted.

SOLUTION Assume that the frequencies are sorted as $(f_1 \leq f_2 \leq \cdots \leq f_n)$. Inductively, let the frequencies $F = (f_i \leq f_{i+1} \leq \cdots f_n)$ be the remaining unprocessed frequencies. Among the frequencies re-introduced into the queue, assume that they are already sorted as $G = (g_1 \leq g_2 \leq \cdots \leq g_k)$. Then our algorithm works as follows: assume the subroutine $RemoveMin(F, G)$ that removes the minimum of f_i and g_1 from F or G. We do $RemoveMin(F, G)$ twice, and create a new node with weight g_{k+1}, whose children are the two removed vertices. The new node is placed at teh end of G. Thus, G becomes $G = (g_1 \leq \cdots \leq g_k \leq g_{k+1})$. We can prove that the sorted order is maintained. Each of these steps takes constant time, and there are linear number of steps. Hence $O(n)$ suffices.

♠

Question 7

(0 Points) DO NOT HAND THIS IN.
Exercise 3.8. 3-ary Huffman code.

SOLUTION

♠