Basic Algorithms (V22.0310); Fall 2005; Yap

HOMEWORK 4

Date Due: Nov 7

Instructions

- Please write CLEARLY. If you have bad handwriting, please consider printing your solutions.
- Be concise, but not obscure.

Question 1

(20+5+5 Points)

Question 5, page 190. Bucolic setting for greedy algorithm. The input is a sequence $x_1 < x_2 < \cdots < x_n$ of numbers where x_i (in miles) indicates the location of the ith house on the road.

We want you to (1) describe your algorithm, (2) prove its optimality, and (3) give a complexity analysis.

To describe your algorithm, you must first give a high level description, and then put your ideas in Java (as a reality check). If this Java code does not compile, you get zero for part (1). Here is a check list:
(a) Specify the input and output of your program.
(b) Explain any variables and data structures.
(c) Initialize your variables and cover the base cases.
(d) Add comments as appropriate.
(e) You must hand in a Makefile. The default target of your Makefile is to compile the program. The target "run" should run the program on some sample data that you provide.

Question 2

(15 Points)

Recall that in the Cache Problem, you have a set U of pages and a cache size k. You are given a sequence $D = (d_1, d_2, \ldots, d_m)$ of pages ($d_i \in U$) and the initial contents of the cache, $C_0 \subseteq U$ where $|C_0| = k$. Your goal is to design a **schedule**

$$S = (e_1, e_2, \ldots, e_m)$$

where $e_i \subseteq U$. The idea is that if C_i is the cached pages just before d_i, then the new cached pages will be

$$C_{i+1} = (C_i \setminus e_i) \cup \{d_i\}.$$

That is, we **evict** the pages in e_i and bring the new page d_i. Without loss of generality, we will assume $|e_i| \leq 1$ (so e_i is empty or a singleton). Suppose $k = 3, C_0 = \{a, b, c\}$ and $D = (d, a, d, b, a, c, a, e, d, b, c, e, f, a, d, b, e, a)$.

(a) Give the Farthest-in-Future schedule for this input. How many evictions are necessary?

(b) Suppose we maintain the cache as a FIFO queue (initially, the queue is (a, b, c) where a is at the front of the queue and c is at the tail of the queue. We always evict the front of the queue and append to the tail of the queue. How many evictions are used in this case?

(c) What if you maintain the cache as a FIFO queue, but each time you reference a cached page, that page is brought to the tail of the queue again. How many evictions are used then?

Question 3

(15 Points)

Let G_n be the bigraph whose vertices are $V = \{1, 2, \ldots, n\}$. The edges are defined as follows: for each $i \in V$, if i is prime, then $(1, i) \in E$ with weight i. [Recall that 1 is not considered prime, so 2 is the smallest prime.] For $1 < i < j$, if i divides j then we add (i, j) to E with weight j/i.

(a) Draw the graph G_{13}.

(b) Compute the MST of G_{18} using Prim's algorithm, using node 1 as the source vertex. Please use the organization described in the appendix below.

Question 4

(15 Points) (a) Dijkstra's algorithm assumes that the weights of edges are positive. Show a counter example where Dijkstra's algorithm fails if the weights are negative. HINT: use as few vertices and edges as you can in your counter example.
(b) What does negative weight do to Prim's algorithm?
(c) We know that Dijkstra's algorithm is very similar to Prim's algorithm. Indeed, we can do hand-simulations of Dijkstra's algorithm in a very similar way to Prim's simulation. Again, we maintain an array $d[1..n]$ where $V = \{1, \dots, n\}$. With source vertex 1, we initialize the array with $d[1] = 0$ and $d[i] = \infty$ for $i = 2, \dots, n$. We maintain the set $S \subseteq V$ with $S = \emptyset$ (empty set). In stage $i \geq 1$, we pick a vertex $v_i \in V \setminus S$ such that $d[v_i]$ is minimum, and we update the array $d[1..n]$: for each u adjacent to v_i, we let

$$d[u] = \min\{d[u], d[v_i] + C(v_i, u)\}$$

where $C(u, v)$ is the cost of the edge (u, v). We write the updated arrow out as a row of a matrix M, as in Prim's simulation.

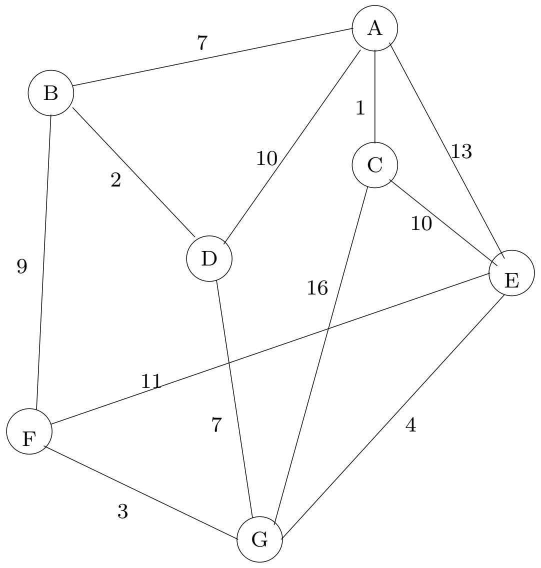

Figure 1: Graph for Dijkstra's algorithm

Please do a hand simulation of Dijkstra's algorithm on the graph in Figure 1. You must draw the array $d[1..n]$ after each phase of the algorithm. HINT: underline the entry corresponding to v_i in each phase, and do not write the value of $d[v]$ unless it is changed.

Question 5

(10 Points)

Give the optimum Huffman code for the following string: `hello!  this is my little world!`

Note that we have space and exclamation marks but no caps. Here is how you must organize your solution into PHASES: In the initial phase, show a set of trivial trees (each tree has only a single node, representing a character of the string). In subsequent phases, show the pairs of trees that are being merged. In the final phase, we must see only one tree.

These trees are called "Huffman trees" or "code trees". Use the following CONVENTION for Huffman trees: Leaves are denoted by squares, while non-leaves are denoted by circles. Write the weight of each node inside the node, but the letter (symbol) associated to a leaf is written beside it.

APPENDIX: Hand Simulation of Prim's Algorithm.

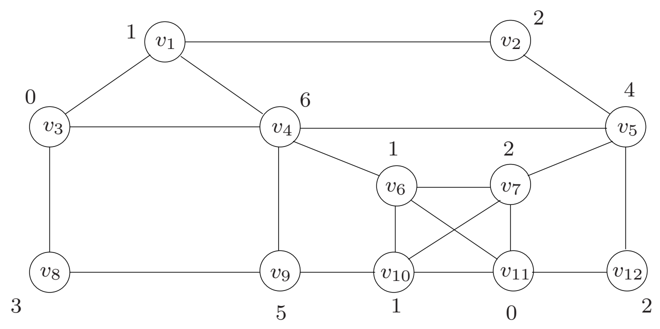

Figure 2: The house graph.

We want you to use the following method to do hand simulation of Prim's algorithm. We maintain an array $d[1..n]$ assuming the vertex set is $V = \{1, \ldots, n\}$. We also maintain a subset $S \subseteq V$ representing the set of vertices which we know how to connect to the source node 1 in a MST. The set S is "Prim safe". Initially, let $S = \emptyset$ and $d[1] = 0$ and $d[v] = \infty$ for $v = 2, \ldots, n$.

In general, the entry $d[v]$ $(v \in V \setminus S)$ represents the "cheapest" cost to connect vertex v to the MST on the set S. Our simulation consists in building up a matrix M which is a $n \times n$ matrix, where the 0th row represents the initial array d. In the ith stage $(i = 1, \ldots, n-1)$, the array d is updated, and its value is written as the ith row of M.

In stage i, we pick a node $v_i \in V \setminus S$ where $d[v_i] = \min\{d[j] : j \in V \setminus S\}$. We add v_i to S, and update all the values $d[u]$ for each $u \in V \setminus S$ that is adjacent to v_i. The update rule is this:

$$d[u] = \min\{d[u], COST[v_i, u]\}.$$

The resulting array is written as row i in our matrix.

Let us illustrate this process on the graph of Figure 2. The vertex set is $V = \{v_1, v_2, \ldots, v_{11}, v_{12}\}$. We specify the weights of each edge in a slightly unconventional way: we first specify a weight for each vertex (e.g., weight of vertex v_1 is 1 and weight of v_4 is 6). The weight of edge (v_i, v_j) is taken to be the sum of the weights of v_i and v_j. E.g. the weight of $(v_1, v_4) = 1 + 6 = 7$. The final matrix is the following:

Stage	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	X	3	1	7	∞	∞	∞	∞	∞	∞	∞	∞
2			X	6				3				
3		X			6							
4								X	8			
5				X		7						
6					X		6					6
7						3	X			3	2	
8						1				1	X	2
9						X						
10									6	X		
11												X
12									X			

Some conventions: We mark the newly picked node in each stage with an 'X'. (Instead of 'X', you can also write the weight, and mark it with an underscore, as we did in class.) Also, any value that is unchanged from the previous row may be left blank. Thus, in stage 2, the node v_3 is picked and we update $d[v_4]$ using $d[v_4] = \min\{d[v_4], COST[v_3, v_4]\} = \min\{7, 6\} = 6$.

There is one other thing you need to learn how to glean from the matrix: how to determine the actual MST and its cost. In the above example, the cost of the MST is 37. To see this, each X corresponds to a vertex v that was picked, and the last value of $d[v]$ contributes to the cost of the MST. E.g., the X corresponding to vertex 1 has cost 0, the X corresponding to vertex 2 has cost 3, etc. Summing up over all X's, we get 37. Moreover, if the last value of $d[v]$ is updated as a result of picking vertex v_i (in stage i), then the edge (v_i, v) is in our MST.