University of Nevada, Las Vegas Las Vegas Computer Science 477/677 Fall 2019

Assignment 8 Due Monday November 18, 2019

Name:____

You are permitted to work in groups, get help from others, read books, and use the internet. But the handwriting on this document must be your own. Print out the document, staple, and fill in the answers. You may attach extra sheets, but only by stapling. Turn in the pages to the graduate assistant at the beginning of class, November 4.

1. (Cuckoo hash table problem.) We define two hash functions, h_1 and h_2, on 4-letter words as follows. Define [letter] to be the place, from one to 26, of that letter in the Roman alphabet, that is:

   [A] = 1, [B] = 2, ... [M] = 13 ... [Z] = 26.

   Let $h_1(\text{word}) = ([\text{first letter}] * 2 + [\text{second letter}] * 4)\,\%\,7$
   and $h_2(\text{word}) = (h_1(\text{word}) + ([\text{third letter}] + [\text{fourth letter}] * 5)\,\%\,6 + 1)\,\%\,7$

   For example, if the work is "John," we have:

   $h_1(\text{John}) = ([\text{J}] * 2 + [\text{O}] * 4)\,\%\,7 = (10 * 2 + 15 * 4)\,\%\,7 = 3$
   $h_2(\text{John}) = (h_1(\text{John}) + ([\text{H}] + [\text{N}] * 5)\,\%\,6 + 1)\,\%\,7 = (3 + (8 + 70)\,\%\,6 + 1)\,\%\,7 = 4$

   (a) Find a word such that h_1 and h_2 of that word are equal.

   (b) Enter the following 4-letter words into a cuckoo hash table of size 7, using the two hash functions h_1 and h_2 defined above. Show the steps. That is, don't erase ejected words, simply cross them out.

   Faye
   Trip
   Thom
   Suzy
   Bess
   Leah
   Ping

2. You are given 12 pennies, all of which are exactly the same weight, except for one “bad” penny, which is either slightly too heavy or slightly too light. You have a balance scale; if you put pennies on each side, the scales gives you one of three answers: the left side is heavier, the right side is heavier, or the two sides have equal weight.

   (a) Give an algorithm which deternies which penny is bad, and whether it is too heavy or too light, using only three weighings.

   There are 24 possible outcomes. Three weighings can, in theory, distinguish $3^3 = 27$ outcomes, so you should be able to do it, and in fact, you can. The first thing to do is to put k pennies on each side, but what is k? There must be at most 9 outcomes after the first weighing. If $k = 3$ and the scale balances, the bad penny is one of the 6 remaining, so there are 12 remaining outcomes. If $k = 5$ and the scale does not balance, there are 10 remaining outcomes. So k can be neither 3 nor 5. If $k = 4$ and the scale balances, there are 8 remaining outcomes, while if the scale does not balance, there are also 8 remaining outcomes. Thus the correct first weighing is to put 4 on each side.

   But what do you do next? The harder case is when the scales do not balance. Suppose the left side goes down and the right side up. Then either one of the 4 ”left” pennies is too heavy, or one of the 4 ”right” pennies is too light. The other 4 pennies are known to be ”good.”

   It took me three days to figure this out when I first worked the problem in 1953. You put two ”left” pennies and one ”right” penny on the left, and put one ”left” penny, one ”right” penny, and one ”good” penny on the right. What then?

   (b) Considier the same problem with 13 pennies. There are 26 possible outcomes, so you should be able to do it in three weighings, but you can’t. Use information theory to prove that it’s impossible. (Hint: What is the number of remaining possibilities after the first weighing?)

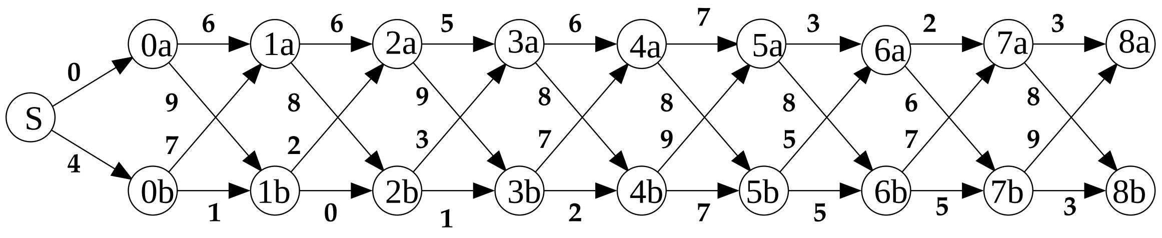
   (c) Suppose there are 13 pennies, one of which is too heavy or too light, and you also have a 14th penny which you know is good. Show how you can find the bad penny and determine whether it’s too heavy or too light in three weighings.

Use extra paper for this problem, and staple that paper to your homework.

3. It is rather easy to compute the maximum (or minimum, or sum) of a list of n numbers in $O(\log n)$ time using n processors, by the "tournament" method. Can you show how to do it in $O(\log n)$ time using $\dfrac{n}{\log n}$ processors?

   Problems 4 and 6 can also be worked with $\dfrac{n}{\log n}$ processors in $O(\log n)$ time.

4. A *layered graph* is a graph whose vertices are partitioned into consecutive layers, and where each edge only connects adjacent layers. For example, the graph G given below is a weighted direected layered graph, with nine layers.



(a) Use dynamic programming to work the single source shortest path problem for G, where S is the source. Show minimum lengths and backpointers.

You know how to multiply matrices. For example: $\begin{pmatrix} 2 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 5 \\ 1 & 2 \end{pmatrix}$ This is the usual kind of matrix multiplication, called "$(+,*)$" matrix multiplication, because each entry in the product matrix is the sum $(+)$ of products $(*)$. That is

$$\begin{array}{rr} 2*1+(-1)*0=2 & 2*2+(-1)*(-1)=5 \\ 1*1+0*0=1 & 1*2+0*(-1)=2 \end{array}$$

But there is another very useful kind, called $(\min,+)$ matrix multiplication. Each entry of the product matrix is the minimum of sums.[1] For example: $\begin{pmatrix} 2 & -1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix}$ Since

$$\begin{array}{rr} \min\{2+1,(-1)+0\}=-1 & \min\{2+2,(-1)+(-1)\}=1 \\ \min\{1+1,0+0\}=0 & \min\{1+2,0+(-1)\}=-1 \end{array}$$

Relating $(\min,+)$ **Multipication to the Layered Graph Problem.** We can define the *initial vector* of our problem to be the vector of minimum costs to the 0th layer, namely $\text{Vec}[0]=(0,4)$. After working the dynamic program, we know that $\text{Vec}[1]=(6,5)$.

Let $M[0,1]=\begin{pmatrix} 6 & 9 \\ 7 & 1 \end{pmatrix}$, the matrix representing the arcs from Layer 0 to Layer 1. Then $\text{Vec}[0]M[0,1]=(0,4)\begin{pmatrix} 6 & 9 \\ 7 & 1 \end{pmatrix}=(6,5)=\text{Vec}[1]$. Similarly, $\text{Vec}[0]M[0,1]M[1,2]M[2,3]M[3,4]=(13,8)=\text{Vec}[4]$.

This formulation does not save any time. However, we can shortcut the computation by using n parallel processors to compute the product $M[0,i]=M[0,1]M[1,2]\cdots M[i-1,i]$ for each $i\le n$, in $O(\log n)$ time, where $n=8$ in our example. Then $Vec[0]M[0,i]=\text{Vec}[i]$ For example

$$M[0,2]=M[1]M[2]=\begin{pmatrix} 6 & 9 \\ 7 & 1 \end{pmatrix}\begin{pmatrix} 6 & 8 \\ 2 & 0 \end{pmatrix}=\begin{pmatrix} 11 & 9 \\ 3 & 1 \end{pmatrix}$$

$$\text{Vec}[0]M[0,2]=(0,4)\begin{pmatrix} 11 & 9 \\ 3 & 1 \end{pmatrix}=(7,5)=\text{Vec}[2]$$

(b) Using the example layered graph G, Compute the following $(\min,+)$ matrix products. (We have already computed $M[0,2]$.)
$M[2,4]=M[2,3]M[3,4]$
$M[4,6]=M[4,5]M[5,6]$
$M[6,8]=M[6,7]M[7,8]$
$M[0,4]=M[0,2]M[2,4]$
$M[4,8]=M[4,6]M[7,8]$
$M[0,8]=M[0,4]M[4,8]$
Finally, compute $\text{Vec}[0]M[0,8]=\text{Vec}[8]$.

(c) For an instance of the problem of arbitrary size, can you compute $\text{Vec}[i]$ for all i in $O(\log n)$ time with n processors? (Hint: Yes.)

[1] We can use any two operators, provided the second operator distributes over the first operator.

5. This is an "extra difficulty" problem. The Levenstein edit distance between two strings of lengths m and n, respectively, can be found in $O(nm)$ time using the dynamic program we went over in class. However, filling out that huge matrix seems awfully wasteful if the edit distance is small.

   Let h be the actual Levenstein distance between two words of lengths n and m. Use the A^* algorithm to find the edit of minimum Levenstein cost in $O(n(h+1)\log n)$ time.

6. This is an “extra difficulty” problem. In the last homework, one problem was to design a dynamic programming algorithm that inputs a sequence of integers, and finds that subsequence which has the largest total, subject to the condition that no two consecutive terms of the original sequence are in the subsequence. For example, if the input sequence is 3,1,4,1,5,9,2,6,5,3,5, your algorithm will find the subsequence 3,4,9,6,5, whose terms add up to 27.

   As we have shown in class, the problem can be worked in $O(n)$ sequential time. The problem can be reduced to the problem of finding the shortest path through a weighted layered directed graph, as in problem 4 of this homework, but each layer has three nodes; using the method given in that problem, we can find the best subsequence in $O(\log n)$ time using n processors.

   The reduction is not easy to find, but once it’s found, it’s easy to understand.