COMP 3711 – Design and Analysis of Algorithms
2019 Spring Semester – Written Assignment # 3
Distributed: April 8, 2019 – Updated April 18, 2019
Due: April 24, 2019

Your solutions should contain (i) your name, (ii) your student ID #, and (iii) your email address

Notes:

- Follow the guidelines on doing your own work and avoiding plagiarism given on the class home page.
  In particular ***don't forget to acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.
- Write clearly and follow the submission guidelines on the class web page. Use white, unwatermarked paper, e.g., no student society stationary.
  - If handwritten, solutions should be single sided, start a new page for every problem, be single column and leave space between consecutive lines and more space between paragraphs.
  - If typed, try to use an equation editor, e.g., latex , the equation editor in MS-WORD or whatever the equivalent is in whatever typesetting system you are using
- This assignment is due by 23:59 on April 24, 2019 in BOTH hard AND soft copy formats. A hard copy should be deposited in one of the two COMP3711 assignment collection boxes outside of room 4210. A soft copy for our records in PDF format should also be submitted via the online CASS system. See the Assignment 1 page in Canvas for information on how to submit online.
- The default base for logarithms will be 2, i.e., $\log n$ will mean $\log_2 n$. If another base is intended, it will be explicitly stated, e.g., $\log_3 n$.
- Fixed typos: There were two typographical errors that were fixed in this update.
  - Problem 3: In original version, the example adjacency list of $(1, 1)$ contained $(0, 2)$. This was corrected to $(1, 2)$.
  - Problem 2, page 4, condition b. Was originally stated as $Gap(s_i, e_i) \leq M$. This was corrected to $Gap(s_i, e_i) \geq 0$.
    Equation on last line. Was originally stated as $\min\{OPT(i) : Gap(i + 1, n) \leq M\}$.
    This was corrected to $\min\{OPT(i) : Gap(i + 1, n) \geq 0\}$.

Problem 1: Optimal Binary Search Trees [15 pts]

Consider the following input to the Optimal Binary Search Tree problem (it is presented in the same format as the example powerpoint file posted on the lecture page):

i	1	2	3	4	5	6	7	8
a_i	A	B	C	D	E	F	G	H
$f(a_i)$	5	7	1	8	4	6	2	3

a) **Fill in the two tables below. As in the example powerpoint only the entries with $i \leq j$ need to be filled in. We have started you off by filling in the $[i, i]$ entries.**

i/j	1	2	3	4	5	6	7	8
1	5							
2		7						
3			1					
4				8				
5					4			
6						6		
7							2	
8								3

Table 1: Left matrix is $e[i, j]$.

i/j	1	2	3	4	5	6	7	8
1	1							
2		2						
3			3					
4				4				
5					5			
6						6		
7							7	
8								8

Right matrix is $root[i, j]$.

b) Draw the optimal Binary Search Tree (with 8 nodes) and give its cost.

Problem 2: Dynamic programming for pretty printing [25 pts]

Consider the problem of neatly formatting a paragraph on a page for printing or displaying. The input text is a sequence of n words of lengths $\ell_1, \ell_2, \ldots, \ell_n$, measured in characters. We want to print this paragraph neatly on lines that hold a maximum of M characters each. As an example, consider the problem of laying out the nonsense phrase, *It is the time for all wise good people and very exceptional friends to hasten quickly and silently home* on a line of length 29. Here are two possible layouts.

```
12345678901234567890123456789
It is the time for all wise
good people and very
exceptional friends to hasten
quickly and silently home
```

```
12345678901234567890123456789
It is the time for all
wise good people and very
exceptional friends to hasten
quickly and silently home
```

Note that the first layout has $2, 9, 0, 4$ spaces remaining at the end of, respectively, its 1st, 2nd, 3rd and 4th lines, while the second layout has $7, 4, 0, 4$ spaces remaining. The second one 'looks" neater (less ragged) than the first.

The criterion we use for measuring "neatness" is as follows: The input will be a sequence $\ell_1, \ell_2, \ldots, \ell_n$ where ℓ_i is the length of word i.

Note that if a given line contains words i through j and we leave exactly one space between words, the number of extra space characters remaining at the end of the line is

$$Gap(i, j) = M - j + i - L(i, j) \quad \text{where} \quad L(i, j) = \sum_{k=i}^{j} \ell_k.$$

The penalty associated with using this line will be the cost

$$C(i, j) = (Gap(i, j))^2 .$$

The cost of a layout will be the sum, over all lines except the last, of the cost of the line. The reason for not charging for the last line is that, in printing, the last line is allowed to be ragged.

Consider the example that started this problem. $M = 29$ and the remainder of the input is the list of the lengths of the 19 words in the paragraph:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ℓ_i	2	2	3	4	3	3	4	4	6	3	4	11	7	2	6	7	3	8	4

Two possible ways (there are others) of placing these words on four lines are (see below for definitions of s_i, e_i)

	Layout 1					Layout 2			
	s_i, e_i	$L(s_i, e_i)$	$Gap(s_i, e_i)$	$C(s_i, e_i)$		s_i, e_i	$L(s_i, e_i)$	$Gap(s_i, e_i)$	$C(s_i, e_i)$
Line $i = 1$	$1, 7$	21	2	4		$1, 6$	17	7	49
Line $i = 2$	$8, 11$	17	9	81		$7, 11$	21	4	16
Line $i = 3$	$12, 15$	26	0	0		$12, 15$	26	0	0
Line $i = 4$	$16, 19$	22	4			$16, 19$	22	4	

The cost of Layout 1 is $2^2 + 9^2 + 0^2 = 85$ while the cost of Layout 2 is $7^2 + 4^2 + 0^2 = 65$ so Layout 2 is much better (less ragged). Note that the costs of their last lines are *not* included in the costs of the paragraphs.

A *Layout* will be defined as a partition of the words into lines. Lines will not be allowed to overflow (be longer than M). A layout will be formally given by a sequence s_i, e_i, for $i = 1 \ldots j$. These are the starting and ending word indices of each line and satisfy:

(a) $s_1 = 1$, $e_j = n$

(b) $\forall i = 1, \ldots, j, \quad s_i \leq e_i$ and $Gap(s_i, e_i) \geq 0$.
(Line i contains a continuous set of words and can not overflow)

(c) $\forall i = 1, \ldots, j-1, \quad s_{i+1} = e_i + 1$
(line $i + 1$ starts immediately after the last word of line i.)

Note that j, the number of lines, is NOT fixed in advance and may range anywhere between 1 and n.

The problem is, given M and the ℓ_i as input, to find a layout that minimizes the cost, $\sum_{i=1}^{j-1} C(s_i, e_i)$ of the paragraph. For example, for the input above, Layout 2 is the best possible layout.

Give a dynamic-programming algorithm to solve this problem for n words. The input will be the value M and the list $\ell_1, \ldots, \ell_n$ of word lengths. Your algorithm should report the optimum cost and the layout (list out the s_i, e_i) that achieves this.

Analyze the running time and space requirements of your algorithm. Full credit will only be given for an $O(n^2)$ time algorithm. Anything that runs worse than that will only receive partial credit.

Design Hint: Modify the problem so that the cost also includes the cost of the last line, define $OPT(i)$ to be the minimum cost of solving this modified problem restricted to just the first i words and find a DP recurrence for $OPT(i)$. Then show that the solution to the original problem satisfies $\min\{OPT(i) : Gap(i + 1, n) \geq 0\}$.

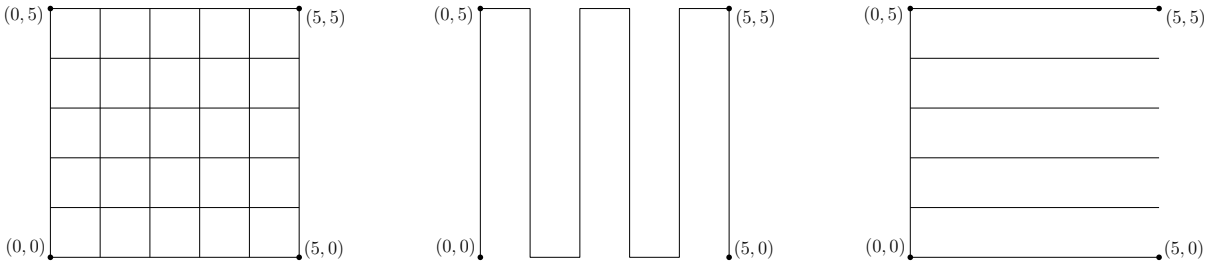
Problem 3: BFS/DFS [20 pts]

In this problem you will have to describe the depth and breadth first search trees calculated for particular graphs. Let graph G_n be the $n \times n$ grid containing the n^2 points (i, j), $i = 0, \ldots, n-1$, $j = 0, \ldots, n-1$. Figure (A) illustrates G_6. Each point is connected to four neighbors: The one below it, the one above it, the one to its left and the one to its right. Note that some points only have two or three neighbors, e.g., the four corner points only have 2 neighbors and that the edge points (aside from the corners) each have three neighbors. The adjacency list representation used is

$$(i, j) : (i, j-1) \to (i, j+1) \to (i-1, j) \to (i+1, j).$$

For example, the adjacency list representation for $(1, 1)$ in G_6 is

$$(1, 1) :\to (1, 0) \to (1, 2) \to (0, 1) \to (2, 1).$$



Some nodes will only have two or three neighbors; their adjacency lists should be adjusted appropriately. For example

$$(0, 0) :\to (0, 1) \to (1, 0) \quad \text{and} \quad (i, 0) :\to (i, 1) \to (i-1, 0) \to (i+1, 0)$$

for $i = 1, \ldots, n-2$.

In what follows *Describe the tree* means (i) list the edges in the tree and (ii) sketch a diagram that illustrates how the tree looks.

(a) Describe the tree produced by Depth First Search run on G_n starting at root $s = (0, 0)$. Figure (B) illustrates the tree produced for G_6.

(b) Describe the tree produced by Breadth First Search run on G_n starting at root $s = (0, 0)$. Figure (C) illustrates the tree produced for G_6.

For the rest of this problem keep G_n the same but change its adjacency list representation to

$$(i, j) :\to (i+1, j) \to (i, j-1) \to (i-1, j) \to (i, j+1).$$

(c) Now, show the tree produced by Depth First Search run on G_6 starting at root $s = (0, 0)$ using this new adjacency list representation.

(d) Now, do the same for the tree produced by Breadth First Search run on G_6.

(e) Finally, describe the tree produced by Depth First Search and Breadth First Search run on G_n using this new adjacency list representation.

Problem 4: Cycle Finding [15 pts]

In class, we learned how to check if an undirected graph has a cycle in $O(|V|)$ time. That algorithm either returned a cycle or told us that the graph is a tree. For this problem the input is an undirected graph $G = (V, E)$ *and* a specified vertex $v \in V$. Design an $O(|E| + |V|)$ time algorithm that returns

- NO: if the graph does not contain a cycle containing v.
- YES: if the graph does contain a cycle containing v. In this case, it also prints out any simple cycle containing v.

Your algorithm should be given using well-documented pseudocode. You need to prove correctness and the fact that it has a $O(|E| + |V|)$ running time.

Your algorithm and proof may explicitly use any statement or algorithm we taught in class or the tutorial as long as you explicitly reference what you are using.

Problem 5: MST Checking [25 pts]

Let G be a connected undirected graph with distinct weights on the edges. Recall that such a graph has a *unique* MST.

Given an edge e of G, can you decide whether e belongs to the MST in $O(|E|)$ time? If you compute the MST and then check whether e belongs to the MST, this would take $O(|E| \log |V|)$ time. To design a faster algorithm, you will need the following theorem:

Edge $e = (u, v)$ does not belong to the MST if and only if there is a path from u to v that consists of only edges cheaper than e.

1. Prove this theorem.
   Caveat: your proof may explicitly use any statement or algorithm we taught in class or the tutorial as long as you explicitly reference what you are using.
2. Describe and prove the correctness and running time of the $O(|E|)$-time algorithm. Hint. Use BFS or DFS.
   Caveat: again, you may use any statement or algorithm we taught in class or the tutorial as long as you explicitly reference what you are using.