#

HKUST – Department of Computer Science and Engineering

COMP3711H: Design and Analysis of Algorithms – Fall 2016

Final Examination

Date: Friday December 16, 2016 Time: 16:30-19:30 Venue: LG3 Multi-purpose Room

Name: ______________________ Student ID: __________________

Email: ______________________

Instructions

- This is a closed book exam. It consists of 26 pages and 10 questions .
- Please write your name, student ID and ITSC email at the top of this page.
- For each subsequent page that you write on, please write your student ID at the top of the page in the space provided.
- Please sign the honor code statement on page 2.
- Answer all the questions within the space provided on the examination paper. You may use the back of the pages for your rough work. The last 2 pages are scrap paper and may also be used for rough work. Each question is on a separate page and has at least one extra page for writing answers This is for clarity and is not meant to imply that each question requires all of the blank pages. Many can be answered using much less space.

Questions	1	2	3	4	5	6	7	8	9	Total
Points	8	13	11	6	13	15	13	15	6	100
Score										

Student ID: ____________

As part of HKUST's introduction of an honor code, the HKUST Senate has recommended that all students be asked to sign a brief declaration printed on examination answer books that their answers are their own work, and that they are aware of the regulations relating to academic integrity. Following this, please read and sign the declaration below.

```
I declare that the answers submitted for
this examination are my own work.

I understand that sanctions will be
imposed, if I am found to have violated the
University regulations governing academic
integrity.

Student's Name:      ____________________________

Student's Signature:   _______________________
```

Student ID: ___________

1. **Sorting** [8 pts]

The leftmost array gives 8 4-digit numbers, where each digit is in the range [0..9]. Run Radix-sort on these numbers. Show the result after sorting the array on each digit. The rightmost array should be your final result.

9 1 6 2				
8 2 3 4				
1 2 3 4				
4 3 2 1				
3 5 2 1				
3 6 2 1				
7 1 8 2				
2 9 8 5				

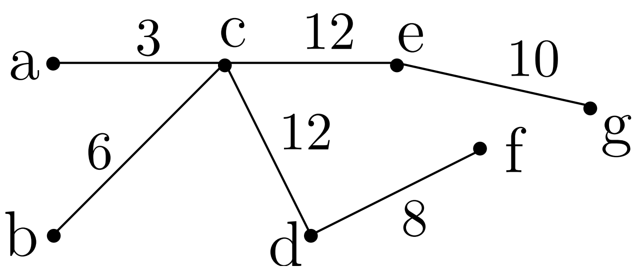
2. **Greedy Algorithms** [13 pts]
Given two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ design an algorithm that determines if X is a *subsequence* of Y. You should assume $m \leq n$.

For example, if $X = BCBA$ and $Y = ABCBDAB$ then the answer is "yes". But, if $X = BDBA$, then the answer is "no". For full credit, your algorithm should run in $O(n)$ time.

You should clearly describe how your algorithm works, either with documented pseudocode or in words. You should also prove that your algorithm is correct and give its running time analysis.

3. **Graphs** [11 pts]
   Recall that a spanning tree of a graph G is a connected acyclic subgraph containing all of the vertices of G.

   Let T be a spanning tree of an undirected weighted graph G. The *bottleneck weight* of T is the largest weight of any edge in T. As an example, in



   the edges (c, e) and (c, d) have the bottleneck weight 12.

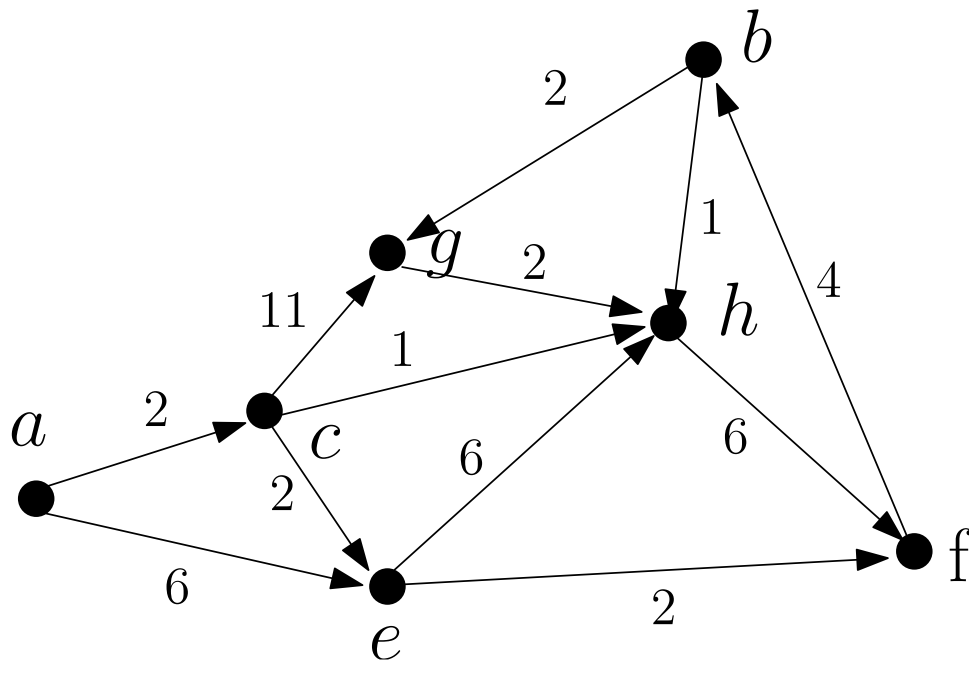
   Design an $O(E)$ algorithm that, given connected, undirected, weighted graph G and value B, either returns a spanning tree with bottleneck weight at most B or reports back that no such spanning tree exists.

   You must explain why your algorithm is correct and justify its running time.

   You may use any algorithm given in class as a subroutine (without having to describe the internal details of that algorithm). If you do use such an algorithm, you must explicitly state the algorithm you are using and its running time.

Student ID: ________

4. **Dijkstra's Algorithm** [6 pts]
Run Dijskstra's algorithm on the graph below, starting from vertex a.



List the vertices by row in the order that Dijkstra's algorithm takes them out of the heap. Recall that $v.d$ is the current value that Dijkstra's algorithm stores with that vertex. Fill in the table below. We have filled in the first two rows for you.

	$a.d$	$b.d$	$c.d$	$e.d$	$f.d$	$g.d$	$h.d$
$-$	0	∞	∞	∞	∞	∞	∞
a	0	∞	2	6	∞	∞	∞

5. **Dynamic programming** [13 pts]
Suppose you are given three strings of characters: $X = x_1x_2\cdots x_n$, $Y = y_1y_2\cdots y_m$, $Z = z_1z_2\cdots z_p$, where $p = n + m$. Z is said to be a *shuffle* of X and Y iff Z can be formed by interleaving the characters from X and Y in a way that maintains the left-to-right ordering of the characters from each string. The goal of this problem is to design a dynamic-programming algorithm that determines whether Z is a shuffle of X and Y.

   (a) Show that *cchocohilaptes* is a shuffle of *chocolate* and *chips*, but that *chocochilatspe* is not.

   (b) For any string $A = a_1a_2\cdots a_r$, let $A_i = a_1a_2\cdots a_i$ be the substring of A consisting of the first i characters of A. For example, if A is *chocolate*, then A_3 is *cho* and A_6 is *chocol*.

   Define $f(i, j)$ to be 1 if Z_{i+j} is a shuffle of X_i and Y_j, and 0 otherwise. Derive a recurrence relation for $f(i, j)$. Remember to include the basis cases. Briefly explain your derivation.

   (c) Give pseudocode for an algorithm for determining whether Z is a shuffle of X and Y. Analyze the running time of your algorithm. For full marks in this part and part (b) your algorithm should run in $O(nm)$ time.

Student ID: ________

Student ID: ________

6. **Dynamic Programming** [15pts]
A string of parentheses is said to be *balanced* if the left-parentheses and right-parentheses in the string can be paired off properly. For example

$$()(),\quad (()),\quad \text{and}\quad ((())())$$

are balanced but ((), and (()))(are not.

Given a string S of length n consisting only of parentheses, design an algorithm to find the *length* of the longest subsequence of S that is balanced.

As an example, the length of the longest balanced subsequence in

$$)(()()(()($$

is 6. One such balanced subsequence of length 6 - there are others- is) (()() (( ) (

- *Hint: Let $s_1, \ldots, s_n$ be the input string. One workable solution is to create a dynamic programming table $OPT[i, j]$ which stores the length of the longest subsequence of $s_i, \ldots, s_j$ that is balanced. Your final solution would then be in $OPT[1, n]$.*
- You must explicitly give the recurrence upon which your algorithm is based.
- You can present the algorithm either in words or via pseudocode
- It is not necessary to explain why your recurrence and algorithms are correct but we recommend that you briefly do so. Without explanations we can not give partial credit for a wrong solution.
- Give the running time of your algorithm (no explanation necessary). In order to gain full credit for this problem your algorithm must run in at most $O(n^3)$ time.

Student ID: ________

7. **Minimum Spanning Trees** [13 pts]
Let G be a weighted undirected connected graph with distinct edge weights. Recall that we proved that such a graph has a unique Minimum Spanning tree. Let T be this unique MST.

   (a) Prove the *Cut Lemma* Below. This is a simplified version of the lemma on safe edges that we proved in class.

   <u>Cut Lemma:</u> Let S be any subset of nodes in G. Let e be the minimum-cost edge among all edges with exactly one endpoint in S and the other endpoint not in S. Then the MST of G must contain e.

   Your proof should be from first principals.

   (b) Either prove or disprove the following statement:
   The edge with the second smallest weight in G must always be in T

   (c) Now either prove or disprove the following statement:

   The edge with the third smallest weight in G must always be in T

For parts (b) and (c) you should assume that G has more than 3 vertices. A proof of correctness may use the Cut-Lemma as a subroutine. Any other lemma or theorem used must be proven from scratch.

A proof that one of the statements is not correct requires a counter-example, e.g., find a graph whose MST does not contain the edge with the second or third smallest weight.

Student ID: ________

Student ID: __________

8. **Max-Flow** [15 pts]

A community of n people is trying to set up a neighborhood patrol. They will send out m patrols each week and each patrol needs three participants. Every community member has agreed to participate in at least 2 patrols a week if he or she is available and $2n = 3m$. The availability information is provided as a set of variables $A_{i,j}$ with, for $1 \leq i \leq n$, $1 \leq j \leq m$,

$$A_{i,j} = \begin{cases} \text{TRUE} & \text{if person } i \text{ is available to participate in patrol } j \\ \text{FALSE} & \text{Otherwise} \end{cases}$$

The *Roster Problem* is to decide whether it is possible to create a feasible schedule. A feasible schedule is one in which:

- Exactly 3 people are assigned to each patrol
- Each person is assigned to exactly two patrols, and
- Person i can only be assigned to patrol j if $A_{i,j} = \text{TRUE}$

(a) Describe how to solve the roster problem by creating an associated max-flow problem and solving that problem using the Ford-Fulkerson algorithm. You must describe the max-flow problem you create (listing the edges and capacities) and show how to transform the solution to the max-flow problem into a solution to the roster problem. It is not necessary to prove correctness of your algorithm

(b) What is the running time of your algorithm as a function of n?

(c) In part (a) you used the Ford-Fulkerson algorithm as a subroutine. As explained in class, if edge capacities are all integers, the Ford-Fulkerson algorithm returns a flow in which every edge value is also an integer. Suppose that your computer system provides another Max-Flow solver, not Ford-Fulkerson, that will always return a Max-Flow, but does not provide the same guarantee, i.e., even if the capacities are all integers it might return a Max-Flow with some non-integer values.

If you replace the Ford-Fulkerson algorithm in part (a) with your system's Max-Flow solver will your algorithm still always be able to find a correct solution to the roster problem? Provide a brief explanation as to why it will or will not be able to find a correct solution. If the answer is that it will not, give a brief example as part of your explanation, showing where your algorithm breaks down.

Student ID: ________

Student ID: ________

9. **Hashing** [6 pts]
In hashing you know the set $U = \{0, 1, \ldots, u-1\}$ of the universe of possible keys that can exist and want to store them in an array of size m (indexed from 0 to $m-1$) using a *hash function* $h : U \to \{0, 1, \ldots, m-1\}$.

   (a) Briefly explain what is meant by *chaining* and by *open addressing*.

   (a) Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps $h : U \to \{0, 1, \ldots, m-1\}$.
   What does it mean to say that "$\mathcal{H}$ **is Universal**"?

Student ID: ________

Scrap Paper

Student ID: ________

Scrap Paper