CITY UNIVERSITY OF HONG KONG

Course code & title	:	CS4335 Design and Analysis of Algorithms
Session	:	Semester A 2020/21
Time allowed	:	Two hours

This paper has Five pages (including this cover page).

1. This paper consists of SIX questions.
2. Answer <u>ALL</u> questions.

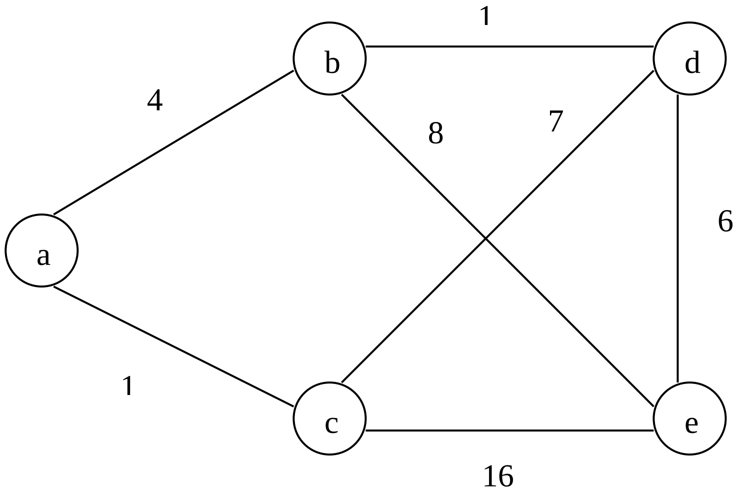
This is a ***closed-book*** *examination.*
This question paper should NOT be taken away.

No materials or aids are allowed during the whole examination. If any unauthorized materials or aids are found on a candidate during the examination, the candidate will be subject to disciplinary action.

\

Question 1. (15 points)

(a) (5 points) Use Kruskal's algorithm to find the minimum spanning tree for the following graph. (Intermediate steps are required. You should indicate the order that the edges are selected.)



(b) (10 points) Given 8 jobs with the following *(v, s, f)*-values (*v*=value, *s*= start time, and *f*= finish time):
a=(3.5,0,6), b=(2,1,4), c=(3,3,5), d=(3,3,8), e=(6.5,4,7), f=(2.5,5,9), g=(12,6,10), h=(8,8,11).
Find a set of mutually compatible jobs with the maximal total value. (Intermediate steps are required.)

Question 2. (15 points)

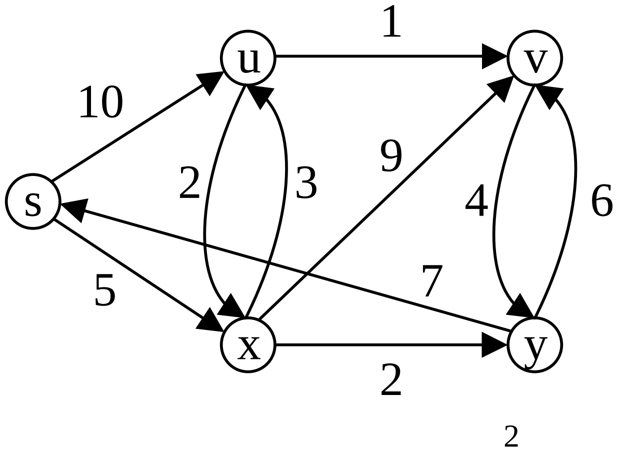
(a) (5 points) Use the dynamic programming algorithm to compute a longest common subsequence of ***ABCABBACA*** and ***BCBABABCA***. Show the backtracking process.

(b) (5 points) Use the dynamic programming algorithm to compute a shortest common supersequence of ***ABCABBACA*** and ***BCBABABCA***. Show the backtracking process.

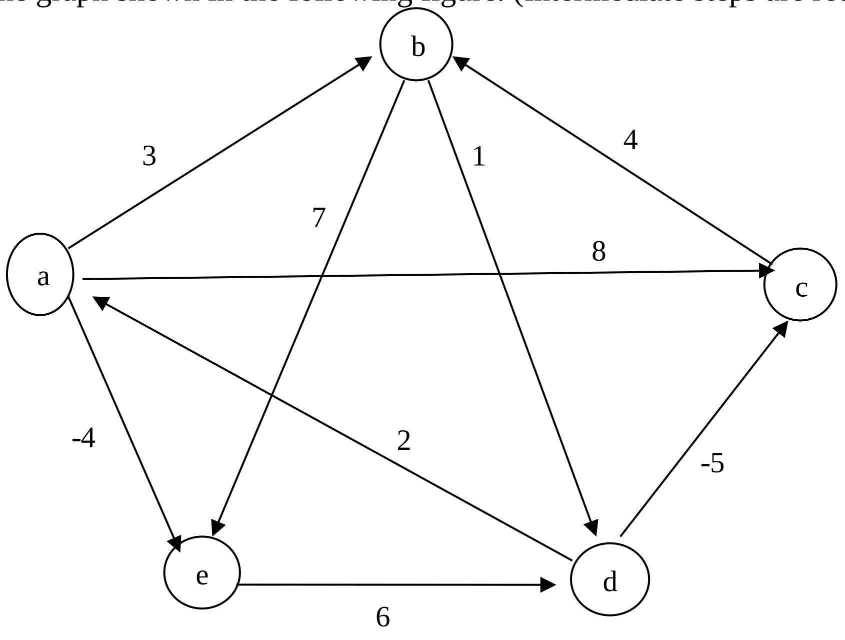
(c) (5 points) Write the pseudo-code for the dynamic programming algorithm (backing process is also included)) for the longest common subsequence problem.

Question 3. (15 points)

(a) (6 points) Use Dijkstra's algorithm to compute the shortest path from *s* to *v* in the graph shown in the following figure. (Intermediate steps are required.)



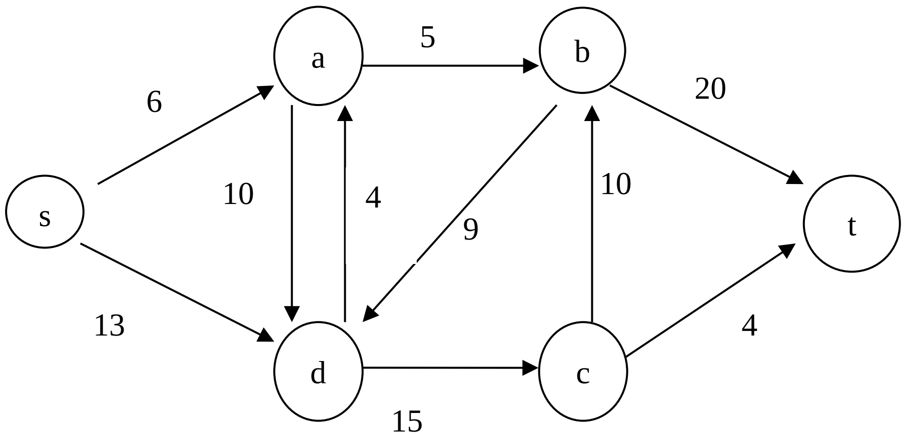
(b) (7 points) Use Bellman-Ford algorithm to compute the shortest path from node *a* to node *c* in the graph shown in the following figure. (Intermediate steps are required.)



(c) (2 points) Suppose that in the input graph there are some negative weight edges, but there is no negative weight circuit. Does Dijkstra's algorithm always give the correct solution? Why?

Question 4. (10 points)

(a) (8 points) Compute the maximum flow for the following graph (source *s* and sink *t*) using Edmonds-Karp's algorithm. (Intermediate steps are required.)



(b) (2 points) What is the running time of the Edmonds-Karp's algorithm?

Question 5. (20 points)

(a) (8 points) **Given the following 0-1 knapsack problem:**

Item	**Value**	**Weight**
1	**1**	**1**
2	**3**	**2**
3	**2**	**1**
4	**3**	**1**
5	**6**	**6**
6	**4**	**3**

The capacity of the knapsack is 8. Use the DP algorithm to solve it.

(b) (2 points) Name two NP-complete problems.

(c) (8 points) The maximum total interval problem is as follows:

Given an array *A[1…n]* of *n* integers (positive or negative), the task is to find a non-empty interval *[i, j]* such that *A[i]+A[i+1]+...+A[j]* is maximized.

Let *d(i)* be the total value of the max sum interval ending at position *i*. For example:

$$d(3) = \max\{A[1] + A[2] + A[3], A[2] + A[3], A[3]\}$$

To compute *d(i),* we can use the recursive equation:

$$d(i) = \begin{cases} A[1] & if\ i = 1 \\ A[i] & if\ i > 1\ and\ \ d(i-1) \le 0 \\ d(i-1) + A[i] & if\ i > 1\ and\ d(i-1) > 0 \end{cases}$$

The final solution is the interval ending at *k*-th position, where *d(k)* is the largest among *d(1), d(2), …, d(n).*

(c1) Compute all the *d(i)*'s for the input array *A=[-1, 2, -1, 3, 4, -5]*.

(c2) Let *b(i)* be the corresponding information for *d(i)* used for backtracking. Design an equation for computing *b(i)* and give the pseudo codes for backtracking process using *b(i)*'s. Compute all the *b(i)*'s for the input array *A=[-1, 2, -1, 3, 4, -5]*. Do backtracking to get the interval with the maximum total value (intermediate steps are required).

(d)(2 points) Suppose that *T(1)=1,* and *T(n)=T(n/5)+1* for n=2, 3, 4 ,…. What is *T(n)* in terms of Big *O* notation?

Question 6. (25 points)

(a) (15 points) A palindrome is a sequence, which reads the same backward and forward. For example, *a*, *bab*, *ababa, bb, abba,* and *caabbaac* are palindromes. Note that a palindrome can have both even and odd numbers of characters.

Given a sequence *S* of *n* letters, the longest palindromic subsequence problem is to find the longest subsequence of *S* that is a palindrome. For example, if the given sequence *S* is *b**babcb**b**cab**c*, then ***babcbab*** is the longest palindromic subsequence in it, where the bold letters appear in the longest palindromic subsequence. Here the term "subsequence" is the same as defined in lectures for LSC and SCS.

Design a dynamic programming algorithm to solve the problem. The running time of the algorithm should be polynomial in terms of *n*. The pseudo codes for backtracking is also required.

(b) (10 points) Suppose you are given a set $S=\{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time. Let w_i be the waiting time of task a_i, that is, the time at which task a_i starts. Your goal is to minimize the average waiting time, that is, to minimize $\frac{1}{n}\sum_{i=1}^{n} w_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1=3$ and $p_2=5$, and consider the schedule in which

a_2 runs first, followed by a_1. Then $w_2=0$, $w_1=5$, and the average waiting time is (0+5)/2=2.5.
Give an algorithm that schedules the tasks so as to minimize the average *waiting* time. Each task must run non-preemptively, that is, once task a_i is started, it must run continuously for p_i units of time. What is the running time of your algorithm. Prove that the algorithm is correct.

- END -