

YORK UNIVERSITY – CSE 3101 – JULY 4, 2006  
**TEST 2**

Student Number: \_\_\_\_\_

Email Address: \_\_\_\_\_

Last (Family) Name(s): \_\_\_\_\_

First (Given) Name(s): \_\_\_\_\_

- This test is worth 20% of your final mark.
- Answer each question directly on the test paper, in the space provided. Use the reverse side of the pages for rough work. If you need more space for one of your solutions, use the reverse side of a page and *indicate clearly the part of your work that should be marked*.
- **20% rule:** If you are unable to answer (part of) a question you will get 20% of the marks for the (part of the) question if you write “I don’t know” and *nothing else* for that part/question.

Question 1 (out of 20)	
Question 2 (out of 10)	
Question 3 (out of 20)	
Question 4 (out of 10)	
<b>Total (out of 50)</b>	

### Question 1 [20]

Recall the Activity Scheduling (or Activity Selection) problem. We are given a set of activities  $\{A_1, \dots, A_n\}$  that wish to use a single resource which can be used by only one activity at a time. Each activity  $A_i$  has a starting time  $s(i)$  and finishing time  $f(i)$ , where  $0 \leq s(i) < f(i)$ , for  $1 \leq i \leq n$ . The Activity Scheduling problem is to find a compatible set of activities of maximum cardinality.

In this question we consider two greedy approaches to solve this problem.

The first approach is to *always select the compatible activity that overlaps with the smallest number of the remaining activities*. Parts **a)** and **b)** below are about this approach.

**a** [5] Write a pseudo-code for a greedy algorithm using the above greedy choice.

**b** [5] Does this approach always produce an optimal solution ? Give a counter-example if answer “No”, and give a proof if answer “Yes”.

(Question 1 part **b** continues here)

The second approach is to *always select the compatible activity that starts the latest among all remaining activities*. Parts **c**) and **d**) below are about this approach.

**c** [5] Write a pseudo-code for a greedy algorithm using the second greedy approach above.

**d [5]** Does this approach always produce an optimal solution ? Give a counter-example if answer “No”, and give a proof if answer “Yes”.

## Question 2 [10]

This question is about Kruskal's algorithm for the Minimum Spanning Tree (MST) problem. First, recall the algorithm GenericMST for the MST problem:

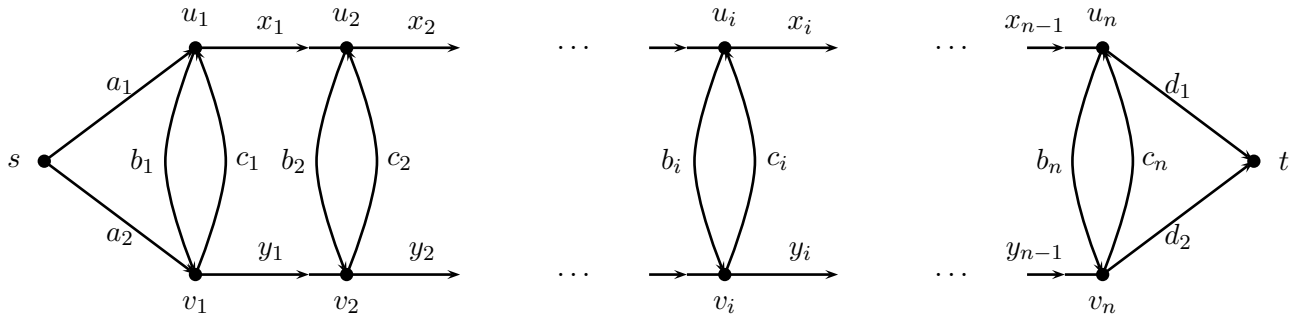
1.  $A \leftarrow \emptyset$  %The set of solution
2. While  $A$  does not form a spanning tree
3. Find a cut  $(X, V - X)$  with no edges from  $A$  crossing the cut
4. Let  $e$  be an edge crossing the cut with smallest weight
5.  $A \leftarrow A \cup \{e\}$
6. End While
7. Output  $A$ .

**a** [5] Describe (or give pseudo-code for) Kruskal's algorithm.

**b [5]** Show that Kruskal's algorithm is a special instance of GenericMST.

### Question 3 [20]

Consider the problem of finding a shortest path from  $s$  to  $t$  (i.e.,  $st$ -path) in the following *directed* graph.



A detailed description of the graph is as follows:

- There are  $(2n + 2)$  vertices:  $s, t$  and  $u_1, v_1, u_2, v_2, \dots, u_n, v_n$ .
- There are edges from  $s$  to  $u_1$  with weight  $a_1$ , and from  $s$  to  $v_1$  with weight  $a_2$ .
- There are edges from  $u_n$  to  $t$  with weight  $d_1$ , and from  $v_n$  to  $t$  with weight  $d_2$ .
- For  $1 \leq i < n$ , there are edges from  $u_i$  to  $u_{i+1}$  with weight  $x_i$ , and from  $v_i$  to  $v_{i+1}$  with weight  $y_i$ .
- For  $1 \leq i \leq n$ , there are edges from  $u_i$  to  $v_i$  with weight  $b_i$ , and from  $v_i$  to  $u_i$  with weight  $c_i$ .

Notice in particular that the directed edges  $(u_i, v_i)$  and  $(v_i, u_i)$  may have different weights.

The conditions on the weights are that  $a_1, a_2, d_1, d_2$  and all the  $x_i, y_i$  ( $1 \leq i < n$ ) are positive.  $b_i$  and  $c_i$  can be negative, although  $b_i + c_i > 0$  for all  $i$ .

Here Dijkstra's algorithm may not work, since the weights  $b_i, c_i$  can be negative. This question asks for a *dynamic programming algorithm* to solve the problem.

**a [5]** Describe an array that can be used to solve this problem. (Explain the meaning of each element of the array.)

**b [5]** Give the initialization and a recurrence for computing the elements of the array. (Hint: Any optimal path from  $s$  to  $t$  must select either  $u_i$  or  $v_i$ , for each  $i$ .)

**c** [5] Give a program to compute the array described in **a**).

**d [5]** Give a program that computes an optimal  $st$  path using the array you described in part **a**). Give the output in the form of an  $n$ -element array  $B$ , where  $B[i] = 1$  if  $u_i$  is in the path, and  $B[i] = 2$  if  $v_i$  is in the path.

#### Question 4 [10]

Given a set of  $n$  items  $\{1, \dots, n\}$  where item  $i$  has nonnegative integer weight  $w(i)$ , and a bound  $W$ , the Subset-Sum (SSS) problem is to find a subset  $S$  of the items with maximum total weight, which is  $\leq W$ , i.e.  $S \subseteq \{1, \dots, n\}$  so that

$$\sum_{i \in S} w(i) \leq W \quad \text{and} \quad \sum_{i \in S} w(i) \text{ is maximum}$$

The Scheduling Jobs with Deadlines, Durations and Profits (SJDDP) problem is specified as follows. The input consists of a set of  $n$  jobs  $\{J_1, \dots, J_n\}$ , where job  $J_i$  has deadline  $d_i$ , duration  $t_i$  and profit  $w_i$  (here  $d_i$ ,  $t_i$  and  $w_i$  are positive integers). There is one machine that can process only one job at a time, and job  $J_i$  must run uninterruptedly for a period of length  $t_i$ . If job  $J_i$  is completed by its deadline  $d_i$ , then you receive a profit  $w_i$ , but if it is completed after its deadline, then you receive a profit of 0. The SJDDP problem is to find a schedule with maximum total profit.

Show that the SSS problem is a special instance of the SJDDP problem. (You are *not* asked to solve these two problems.)