University of Toronto
Scarborough Campus
December 20, 2021

CSC C73 Final Examination
Instructor: Vassos Hadzilacos

Aids allowed: One 8.5 × 11 handwritten, non-photocopied 'cheat sheet' (may be written on both sides)

Duration: Three hours

READ THE INSTRUCTIONS CAREFULLY

- There should be 11 pages in this exam booklet, including this cover page.
- Answer all questions.
- If possible, print the exam booklet, and put all answers in this booklet, in the spaces provided. If you do not have access to a printer, ***write the answer to each question on a different sheet of paper***.
- At the end of the exam period scan or take a photograph of your answers, and upload them on Crowdmark in PDF, JPG, or PGN form. Make sure that you upload the correct file for each answer, and that the images are clear and oriented correctly.
- In your answers you may use any result discussed in this course or its prerequisites merely by stating it.
- Good luck!

Problem	Marks Rec'ved	Marks Worth
1.		20
2.		20
3.		14
4.		20
5.		16
6.		20
7.		30
8.		40
TOTAL		180

QUESTION 1. (20 marks)

For each of the statements below, indicate whether it is true or false by ***circling*** the appropriate response. Do not justify your answers. No penalty for wrong answers, but do not rush to guess.

a. An algorithm whose running time $T(n)$ satisfies the recurrence $T(n) = 2T(n/4) + cn$ is asymptotically faster than an algorithm whose running time $S(n)$ satisfies the recurrence $S(n) = 4S(n/2) + c$. ..... **True** | **False**

b. If all symbols have distinct probabilities, Huffman's algorithm ensures that no symbol has a shorter codeword than the most frequent symbol. ..... **True** | **False**

c. If a dynamic programming algorithm solves n subproblems, where n is the size of its input, then its running time is $O(n)$. ..... **True** | **False**

d. Johnson's algorithm adjusts the weights of edges by adding the same amount to the weight of every edge. ..... **True** | **False**

e. Dijkstra's algorithm works correctly on any graph that has no negative-weight cycles. .... **True** | **False**

f. The $O(n \log n)$ algorithm to find a pair of closest points on the plane that we saw in the course is an example of a greedy algorithm. ..... **True** | **False**

g. If in a flow graph we double the capacity of every edge, then the value of the maximum flow also doubles. ..... **True** | **False**

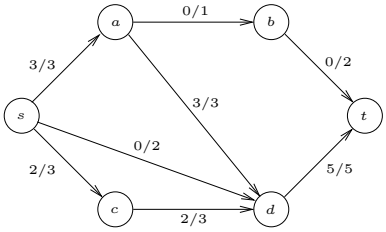
h. The Ford-Fulkerson algorithm runs in polynomial time when we choose the augmenting paths arbitrarily. ..... **True** | **False**

i. The set of nodes of every undirected graph can be partitioned into sets X and Y so that at least half of the graph's edges join a node in X to a node in Y. ..... **True** | **False**

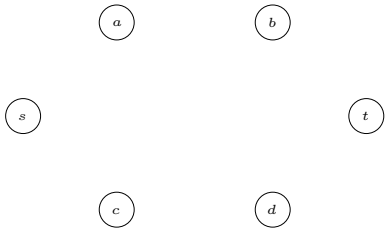
j. For any $\epsilon > 1$, there is a polynomial time algorithm for the 0-1 knapsack problem that finds a feasible solution whose value is at most a factor of ϵ smaller than the value of an optimal solution. ..... **True** | **False**

QUESTION 2. (20 marks)

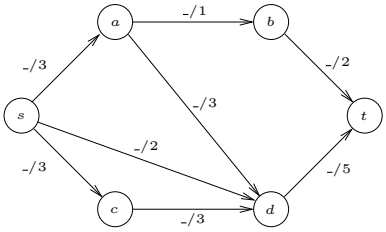
Shown below in Figure (a) is a flow graph with source s and sink t, and an associated flow f. The label of the form k/c next to each edge indicates that in flow f there are k units of traffic going through the edge of capacity c. For example, there are 2 units of traffic in edge (c, d), whose capacity is 3.



(a) Flow f



(b) Residual graph wrt f



(c) Maximum flow

a. (7 marks) Draw the residual graph with respect to f by completing Figure (b) above.

b. (7 marks) Flow f is not a maximum flow. In Figure (c) above, show a maximum flow of the flow graph by adding appropriate values in the spaces that contain "_".

c. (6 marks) Let $\mathcal{F} = (G, s, t, c)$ be a flow network, f be a flow in $\mathcal{F}$, and (S, T) be a cut in $\mathcal{F}$ such that $s \in S$ and $t \in T$. Prove that if there is an edge (u, v) such that $u \in T$, $v \in S$, and $f(u, v) > 0$, then either f is ***not*** a maximum flow or (S, T) is ***not*** a minimum cut. Make sure to cite the relevant facts about flows and cuts that we have proved in the course.

Proof:

QUESTION 3. (14 marks)

Recall the following terminology about linear programs (LPs) and integer linear programs (ILPs):

- A program is ***feasible*** if it has a feasible solution, i.e., a solution that satisfies all the constraints.
- A program is ***unbounded*** if it has feasible solutions that give its objective function arbitrarily large values (for maximization LPs) or arbitrarily small values (for minimization LPs).
- A program ***has an optimal solution*** if it is feasible but is not unbounded — so there is a feasible solution that maximizes (or minimizes) its objective function.

Let P be a ***maximization*** ILP, and P' be its ***relaxation***; i.e., P' is the LP obtained by removing the integer constraints on the variables of P, thus allowing the variables to take real values. For each part (a)-(d) below, circle the unique choice (i), (ii), or (iii) that completes the phrase to a true statement. Do not justify your answer.

a. If P is feasible, then

(i) P' ***must*** be feasible.

(ii) P' ***must*** be infeasible.

(iii) P' ***may or may not*** be feasible, depending on its constraints.

b. If P' is feasible, then

(i) P ***must*** be feasible.

(ii) P ***must*** be infeasible.

(iii) P ***may or may not*** be feasible, depending on its constraints.

c. If P is unbounded, then

(i) P' ***must*** be unbounded.

(ii) P' ***may or may not*** be unbounded, depending on its ***constraints***.

(iii) P' ***may or may not*** be unbounded, depending on the ***objective function***.

d. If the vector $\mathbf{x}$ is an optimal solution of P, the vector $\mathbf{x}'$ is an optimal solution of P', and f is the (common) objective function of P and P', then

(i) it ***must*** be the case that $f(\mathbf{x}) \leq f(\mathbf{x}')$.

(ii) it ***must*** the case that $f(\mathbf{x}) \geq f(\mathbf{x}')$.

(iii) it ***may*** be that $f(\mathbf{x}) \leq f(\mathbf{x}')$ or it ***may be*** that $f(\mathbf{x}) \geq f(\mathbf{x}')$, depending on the objective function.

QUESTION 4. (20 marks)

The incomplete algorithm SQUARE(x) shown below takes as input a binary string x representing a non-negative integer a and returns the binary string representating a^2. For example, given as input the string 101, the algorithm returns the string 11001. The algorithm is very similar in spirit to Karatsuba's multiplication algorithm.

a. Complete the missing parts of SQUARE. Do not justify your answer. You may assume that you are given the following algorithms to use in your answer (two of them appear in the given parts of the algorithm):

- SUM(x, y) takes as input binary strings x and y representing two non-negative integers a and b, and returns the binary string representing $a + b$. It runs in time $O(\max(|x|, |y|)$, where $|x|$ is the length of x.
- DIFF(x, y) takes as input binary strings x and y representing two non-negative integers a and b such that $a \geq b$, and returns the binary string representing $a - b$. It runs in time $O(\max(|x|, |y|)$.
- APPENDZ(x, k) takes as input a binary string x and a positive integer k and returns the string x with k 0's appended. (That is, it returns the binary representation of the integer represented by x multiplied by 2^k.) It runs in time $O(|x| + k)$.

```
SQUARE(x[1..n])
1  if n = 1 then return x
2  else
3      x_1 := x[1..⌈n/2⌉]          ▶  x_1 is the first half (most significant bits) of x
4      x_0 := x[⌈n/2⌉ + 1..n]      ▶  x_0 is the second half (least significant bits) of x
5      a := SQUARE(_____)
6      b := SQUARE(_____)
7      c := SQUARE(_____)
8      d := DIFF(c, SUM(a, b))
9      return _____
```

b. Use the Master Theorem to analyze the running time of the algorithm as a function of n, the length of the input binary string x.

Running time analysis:

QUESTION 5. (16 marks)

Shown below is the pseudocode for the Floyd-Warshall algorithm, where $G = (V, E)$ is a directed graph with $V = \{1, 2, \ldots, n\}$ and $\mathbf{wt} : E \to \mathbb{R}$ is an edge-weight function such that G has no negative-weight cycle. The algorithm returns the 2-dimensional array $C[-, -, n]$, where $C[i, j, n]$, $1 \leq i, j \leq n$, contains the weight of a shortest $i \to j$ path in G.

```
    FloydWarshall(G, wt)
1   for i := 1 to n do
2       for j := 1 to n do
3           if i = j then C[i, j, 0] := 0
4           elsif (i, j) ∈ E then C[i, j, 0] := wt(i, j)
5           else C[i, j, 0] := ∞
6   for k := 1 to n do
7       for i := 1 to n do
8           for j := 1 to n do
9               C[i, j, k] := C[i, j, k − 1]
10              if C[i, j, k] > C[i, k, k − 1] + C[k, j, k − 1] then
11                  C[i, j, k] := C[i, k, k − 1] + C[k, j, k − 1]
12  return C[−, −, n]
```

Modify the algorithm so that it returns a 2-dimensional array $N[-, -, n]$ where $N[i, j, n]$ is the ***number*** of distinct shortest $i \to j$ paths in G (0, if there is no $i \to j$ path in G).

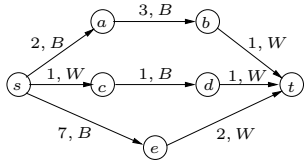
You do not need to copy the entire algorithm; you can describe the changes by saying "replace line(s) ... by ..." for every place that you need to change.

Answer:

QUESTION 6. (20 marks)

We are given a directed graph $G = (V, E)$ and two distinguished nodes $s, t \in V$. Each edge e has a positive weight $\mathbf{wt}(e)$ and a colour $\mathbf{col}(e) \in \{\text{black}, \text{white}\}$. The ***coloured-length*** of a path in G is the sum of the weights of the edges plus the fixed amount 5 ***each time*** the path changes colour — i.e., it leaves a node with an edge of the opposite colour than the edge with which it enters the node.

For example, the colour-length of the three $s \to t$ paths in the graph shown to the right (where the label 'x, y' of an edge indicates its weight x and color y) are: $2 + 3 + 5 + 1 = 11$ (for s, a, b, t), $1 + 5 + 1 + 5 + 1 = 13$ (for s, c, d, t), and $7 + 5 + 2 = 14$ (for s, e, t).



Given as input the graph G, the nodes s, t, and the edge weight and colour functions **wt** and **col**, we want to find the minimum coloured-length of an $s \to t$ path in G. Give an efficient algorithm to solve this problem ***by reducing it to a shortest-path problem***. Describe your algorithm in clear point-form English, not in pseudocode, and analyze its running time. Do not justify its correctness.

Algorithm:

Running time analysis:

QUESTION 7. (30 marks)

Let $a_1\, b_1\, a_2\, b_2\, a_3\, \ldots\, a_{n-1}\, b_{n-1}\, a_n$ be a sequence where each a_i, $1 \le i \le n$, is an integer, and each b_i, $1 \le i \le n-1$, is either $+$ or $-$. Thus, this sequence represents an unparenthesized arithmetic expression such as

$$3 - 5 - 7 + 9.$$

The actual value of such an expression depends on the order in which the operations $+$ and $-$ are applied, i.e., on how the expression is parenthesized. For instance the above example can be parenthesized in five ways:

$$\begin{aligned}
&3 - \big(5 - (7 + 9)\big) = 14\\
&3 - \big((5 - 7) + 9\big) = -4\\
&(3 - 5) - (7 + 9) = -18\\
&\big(3 - (5 - 7)\big) + 9 = 14\\
&\big((3 - 5) - 7\big) + 9 = 0
\end{aligned}$$

In this question you are asked to develop a ***polynomial-time dynamic programming algorithm*** to find the largest and smallest possible values of a given unparenthesized expression, over all possible orders of performing the operations — i.e., over all possible parenthesizations. For instance in the above example your algorithm should return the pair $(14, -18)$. More precisely, your algorithm should solve the following problem:

INPUT: Sequences $a_1, \ldots, a_n$ and $b_1, \ldots, b_{n-1}$, where $a_i \in \mathbb{Z}$ for each $1 \le i \le n$ and $b_i \in \{+, -\}$ for each $1 \le i \le n-1$.

OUTPUT: The pair $(v_{\max}, v_{\min})$, where $v_{\max}$ is the maximum and $v_{\min}$ is the minimum value that can be achieved by parenthesizing the expression $a_1\, b_1\, a_2\, b_2\, a_3\, \ldots\, a_{n-1}\, b_{n-1}\, a_n$.

Hint. A parenthesization of the expression can be viewed as a tree, where the leaves are the a_i's (integers) and the internal nodes are the b_i's ($+$ or $-$).

a. Define clearly the subproblems that your dynamic programming algorithm will solve.

Answer:

[continued on next page]

b. Give a recursive formula to compute the solution to the subproblems in part (a). Do not explain why your formula is correct.

Answer:

c. Describe your dynamic programming algorithm in pseudocode. Do not explain why your algorithm is correct.

Answer:

d. What is the running time of your algorithm? Do not justify your answer.

Answer:

QUESTION 8. (40 marks)

A temporary employment agency has a set W of ***workers*** and provides a set S of ***services*** (e.g., answering phones, filing, typing, Excel, etc). Each worker $w \in W$ is qualified for a subset S_w of these services and is available for h_w of hours per day, an integer in the range between 1 and 8. Every day, the agency receives a set R of ***requests*** to be carried out. Each request $r \in R$ requires exactly one service $s_r \in S$ and a number of hours t_r, a positive integer. For example, if the request r is "5 hours of typing", then $s_r =$ 'typing' and $t_r = 5$.

An ***assignment*** (of workers to requests) is a set A of triples of the form $(w, r, d) \in W \times R \times \mathbb{R}^+$, indicating that worker w is assigned to request r for $d > 0$ hours (note that d is a positive real, not necessarily an integer). The assignment A may contain several triples with the same first component (when a worker is assigned to multiple requests), or with the same second component (when several workers are assigned to the same request). Also note that an assignment is ***not*** required to fill all of a worker's available hours or to provide all the hours needed to complete a request. An assignment is ***feasible*** if it satisfies the following constraints:

(a) If a worker w is assigned to request r, then w is qualified for the service s_r that r requires, i.e., $s_r \in S_w$.

(b) The total number of hours that worker w is assigned to requests does not exceed w's availability h_w.

(c) The total number of hours that workers are assigned to request r does not exceed r's time requirement t_r.

You are hired to design algorithms to determine an assignment. Your algorithms take as input

- the set of workers W and the set of services S;
- for each $w \in W$, the services S_w for which w is qualified, and the number of hours h_w for which w is available;
- the set of requests R; and
- for each $r \in R$, the service s_r and the number of hours t_r that r requires.

They output a feasible assignment with the optimality requirement specified in each part below.

a. (25 marks) We want a feasible assignment that ***maximizes the total number of hours assigned to the workers***. Explain how to solve this problem ***by reducing it to the maximum flow problem***, and analyze the running time of your solution. Do not justify its correctness. Your explanation may include (but should not be limited to) a useful diagram.

Algorithm:

[Continued (and additional space) on the next page]

Running time analysis:

b. (15 marks) Suppose that for every hour (or fraction thereof) that a worker is assigned to request r the agency makes a profit p_r, which is also given as part of the input. The agency wants to find a feasible assignment that ***maximizes its total profit***. Write a linear program that solves this problem. Clearly state your variables, including their intended meaning, the objective function, and the constraints. Also explain how to obtain the desired assignment (i.e., set of triples of the form (w, r, d)) from the solution to your linear program.

Variables:

Objective function:

Constraints:

Assignment obtained from solution:

THE END