University of Toronto
Scarborough Campus
December 15, 2020

**CSC C73 Final Examination**
**Instructor:** Vassos Hadzilacos

**Aids allowed:** One $8.5 \times 11$ 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. | | 15 |
| 3. | | 20 |
| 4. | | 15 |
| 5. | | 20 |
| 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.** If $T(n) = 4T(n/2) + n^2$ then $T(n) = O(n^2)$. ..... **True** | **False**

**b.** The maximum length of any codeword produced by Huffman's algorithm when the input is an alphabet with $n$ symbols is $\lceil \log_2 n \rceil$. ..... **True** | **False**

**c.** The Fast Fourier Transform is an example of a divide-and-conquer algorithm. ..... **True** | **False**

**d.** There is an algorithm to compute the product of two $n$-bit numbers in $O(n^{\log_2 3})$ time. .... **True** | **False**

**e.** If for some iteration $k > 0$ of the Bellman-Ford algorithm, $L(u, k) = L(u, k-1)$ for every node $u$, then for all iterations $\ell > k$ and all nodes $u$, $L(u, \ell) = L(u, k)$. ..... **True** | **False**

**f.** If for some iteration $k > 0$ of the Floyd-Warshall algorithm, $C(i, j, k) = C(i, j, k-1)$ for every pair of nodes $i$ and $j$, then for all iterations $\ell > k$ and all nodes $i, j$, $C(i, j, \ell) = C(i, j, k)$. ..... **True** | **False**

**g.** Let $G$ be a directed graph with $n$ nodes, so that every node of $G$ has $O(\log n)$ outgoing edges. The running time of Johnson's algorithm on $G$ is $O\big((n \log n)^2\big)$. ..... **True** | **False**

**h.** The Floyd-Warshall algorithm can be used to compute the transitive closure of a graph with $n$ nodes in $O(n^3)$ time. ..... **True** | **False**

**i.** The simplex algorithm solves any linear program in polynomial time. ..... **True** | **False**

**j.** If a linear program is feasible and bounded, then it has a unique optimal solution. ..... **True** | **False**

**QUESTION 2.** (15 marks)

For each of the following algorithms:

- State what it does, by describing its input, including any restrictions required for its correctness, and its output.
- State its worst-case running time in asymptotic terms. Make sure to define any parameters used in the running time function. If the running time depends on any special assumptions, state them.

You may use terms and concepts defined in the course without redefining them.

**a.** The Fast Fourier Transform.

**Input:**

**Output:**

**Running time:**

**b.** The Bellman-Ford algorithm.

**Input:**

**Output:**

**Running time:**

**c.** The Ford-Fulkerson algorithm.

**Input:**

**Output:**

**Running time:**

**QUESTION 3.** (20 marks)

For each statement (a)-(d) below indicate whether it is true for ***every*** flow network $\mathcal{F}$ and justify your answer.

**a.** If $f$ is a flow of $\mathcal{F}$ with value 0, then $f(e) = 0$ for every edge $e$ of $\mathcal{F}$.

**Circle the correct answer:** True | Not true

**Justification:**

**b.** If $\mathcal{F}$ has a unique maximum flow then it has a unique minimum cut.

**Circle the correct answer:** True | Not true

**Justification:**

**c.** Let $\mathcal{F}^{+\alpha}$ be the flow network obtained from $\mathcal{F}$ by adding $\alpha$ to the capacity of every edge. If $(S, T)$ is a minimum cut of $\mathcal{F}$, then $(S, T)$ is also a minimum cut of $\mathcal{F}^{+\alpha}$ for every $\alpha > 0$.

**Circle the correct answer:** True | Not true

**Justification:**

**d.** Let $\mathcal{F}^{\times\alpha}$ be the flow network obtained from $\mathcal{F}$ by multiplying the capacity of every edge by $\alpha$. If $(S, T)$ is a minimum cut of $\mathcal{F}$, then $(S, T)$ is also a minimum cut of $\mathcal{F}^{\times\alpha}$ for every $\alpha > 1$.

**Circle the correct answer:** True | Not true

**Justification:**

**QUESTION 4.** (15 marks)

A butcher has received orders from $n$ customers, each requesting a turkey for the holidays. Each customer specifies a ***minimum*** weight for the turkey he or she ordered. The butcher has $n$ turkeys in stock and wants to know whether these can satisfy the minimum weight requirement of every customer's order. Give an ***efficient greedy algorithm*** to help the butcher determine this.

More precisely, your algorithm takes as input two arrays $M[1..n]$ and $W[1..n]$ of positive integers, where $M[i]$ is the minimum weight of the turkey ordered by customer $i$, and $W[i]$ is the actual weight of the $i$-th turkey in stock. Your algorithm should return "true" if it is possible to assign one turkey to each customer so that the turkey's weight is at least the minimum weight requested by the customer, and no turkey is assigned to different customers; it should return "false" if no such assignment exists. (Your algorithm need not return the assignment; all it needs to do is to determine whether one is possible.)

***Describe your algorithm*** in clear high-level pseudocode or point-form English, and ***analyze its running time***. You need not justify its correctness.

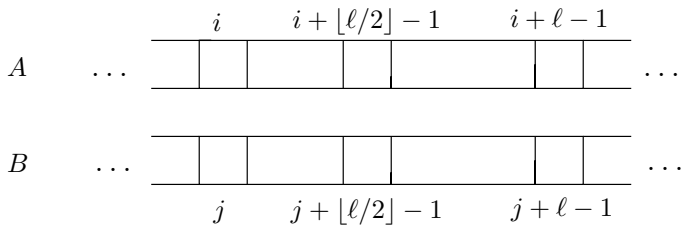
**Answer:**

**QUESTION 5.** (20 marks)

Let $A[1..n]$ and $B[1..n]$ be two arrays that contain $2n$ ***distinct*** integers; each array is ***sorted*** in increasing order. We want to find the median of the $2n$ integers contained in the union of the two arrays. Recall that the median of $k$ distinct numbers is the number in position $\lceil k/2 \rceil$, if the $k$ numbers are arranged in increasing order.

**a.** Shown below is an incomplete divide-and-conquer algorithm MEDIANOFTWO$(i, j, \ell)$ that returns the median of the union of the two subarrays of $A$ and $B$ that start at index $i$ and $j$, respectively, and have length $\ell$; that is, the subarrays $A[i..i+\ell-1]$ and $B[j..j+\ell-1]$. The algorithm assumes that $i, j$, and $\ell$ are positive integers such that $i+\ell < n$ and $j+\ell < n$, so that the subarrays are within the range of $A[1..n]$ and $B[1..n]$. (Thus, the call MEDIANOFTWO$(1, 1, n)$ returns the median of the union of numbers in $A$ and $B$.) Complete the missing parts of MEDIANOFTWO, so that the algorithm runs in $O(\log \ell)$ time. Do not justify your answer.

The figure below may help visualize the algorithm.




MEDIANOFTWO$(i, j, \ell)$

**if** $\ell = 1$ **then return** _____

**else**

**if** $A[i + \lfloor \ell/2 \rfloor - 1] < B[j + \lfloor \ell/2 \rfloor - 1]$ **then return** _____

**else return** _____

**b.** Justify the running time of the algorithm using the Master Theorem. You may assume that $\ell$ is a power of 2.

**Running time justification:**

**QUESTION 6.** (20 marks)

We are given a directed graph $G = (V, E)$ and two distinguished nodes $s, t \in V$. A set of $s$-to-$t$ paths in $G$ is ***node-disjoint*** if no two paths in the set share a node other than $s$ and $t$. We wish to find a maximum-size set of node-disjoint $s$-to-$t$ paths in $G$.

In high-level pseudocode or point-form English describe an algorithm that solves this problem, and analyze its running time. Do not justify the correctness of your algorithm.

**Hint.** Solve this problem by reducing it to the problem of finding a maximum cardinality set of edge-disjoint paths.

**Algorithm:**

**Running time analysis:**

**QUESTION 7.** (30 marks)

The UTSC landscaping department is building a new walkway connecting two buildings. The walkway is $n$ meters long and 1 meter wide, and will be covered using tiles of three types: $1m \times 1m$ black tiles (B), $2m \times 1m$ black-white (BW), and $3m \times 1m$ black-white-black tiles (BWB), as illustrated below.

B BW BWB

Thus, each $1 \times 1$ square of the walkway will be coloured black or white. The landscaping department has a pattern $P[1..n]$ describing the sequence of colours that should appear on the walkway. For example, if the walkway is seven meters long ($n = 7$), and the pattern is BBWWBBB then the first two square meters of the walkway should be black, the next two should be white, and the last three should be black. There is a limited number of tiles of each type: $k_1$ B-tiles, $k_2$ BW-tiles, and $k_3$ BWB-tiles. Note that a BW-tile can be used to form the pattern BW or (by turning it 180°) WB.

In this question you will develop a dynamic programming algorithm to determine whether the available tiles can be used to make the pattern $P$. More precisely, your algorithm should solve the following problem:

INPUT: A pattern $P[1..n]$, where $P[i] \in \{\text{B}, \text{W}\}$ for each $1 \leq i \leq n$, and integers $k_1, k_2, k_3 \geq 0$.

OUTPUT: Return "yes", if there is a way of making the pattern $P$ using no more than $k_1$ B-tiles, $k_2$ BW-tiles, and $k_3$ BWB-tiles; and return "no", otherwise.

For example, if $P$ = BWWBBWBBBBWB and $k_1 = 4$, $k_2 = 2$, $k_3 = 3$, your algorithm should return "yes" since pattern $P$ can be made with two tiles of each type as follows: BW|WB|BWB|B|B|BWB (the vertical lines indicate boundaries between tiles). For the same input except that $k_2 = 1$, the algorithm should return "no", since we need at least two BW-tiles for a pattern that contains two consecutive W. The algorithm should return "no" if $P$ = WWBB no matter what $k_1, k_2$ and $k_3$ are, since there is no way of making a pattern that starts with WW.

**Hint.** Supposing there is a way to create the desired pattern with the available tiles, what are the possibilities for the last tile used? How many tiles of each type are available for the remaining pattern (i.e., the pattern excluding the last tile)?

**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? Is it a polynomial time algorithm?

**Answer:**

**QUESTION 8.** (40 marks)

**a.** (25 marks) A school district has a set $T$ of $n$ teachers and a set $S$ of $m$ schools. There are three possibly overlapping subsets of $T$: the set $M$ of teachers who are qualified to teach mathematics, the set $H$ of teachers who are qualified to teach humanities, and the set $P$ of teachers who are qualified to teach physical education. Every teacher belongs to at least one (but possibly more) of these subsets.

Each teacher can be assigned to at most one school. The school district rules require every school to have 15 teachers, ten of whom must have designated qualifications: four of them must be qualified to teach mathematics, four ***others*** must be qualified to teach humanities, and two ***others*** must be qualified to teach physical education; the remaining five teachers assigned to the school can have any qualification.

Explain how to use the Ford-Fulkerson algorithm to obtain a polynomial-time algorithm that takes as input the sets $T$, $S$, $M$, $H$, and $P$, and determines whether it is possible to assign teachers to schools so as to satisfy the school district rules for ***all*** $m$ schools, and if so it produces an assignment of teachers to schools that does. Do not justify the correctness of your algorithm. Analyze the running time of your algorithm in terms of the number $n$ of teachers and the number $m$ of schools.

**Answer:**

**b.** (15 marks) Now suppose that in addition to the sets $T$, $S$, $M$, $H$, and $P$, we are given the salary $a_i$ of every teacher $i \in T$. The school district wants an assignment of teachers to schools that satisfies the requirements stated in part (a) (if possible) and minimizes the sum of the salaries of the teachers assigned to schools.

Write a 0-1 linear program that solves this problem. (Your 0-1 LP should be infeasible if the given teachers and their qualifications cannot satisfy the school district's rules.) Clearly state your variables, including their intended meaning, the objective function, and the constraints.

**Hint:** You may find it useful to use different variables to indicate whether a teacher is assigned to a school to teach mathematics, humanities, physical education, or as an additional teacher.

**Variables:**

**Objective function:**

**Constraints:**

**THE END**