National University of Singapore

School of Computing

CS4234 - Optimisation Algorithms

(Semester 1: AY2017/18)

Wednesday, 29 November 2017, PM (2 hours)

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises TEN (10) printed pages, including this page.
3. This is an **Open Book Assessment**.
4. Answer **ALL** questions within the **boxed space** or **on free page 10**.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard *) question.
   Read all the questions first! Some questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name *and its time complexity*, e.g. run $O(n^2m)$ Dinic's algorithm on flow graph G, run $O(n^3)$ Push-Relabel on flow graph G', run $O(n^3)$ Edmonds Matching on unweighted graph G'', run $O(n^3)$ Hungarian algorithm on weighted bipartite graph G''', etc.
7. Write your Student Number in the box below:

A	0							

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Remarks
A	45		
B	55		
Total	100		

A (P?-)easy Questions (45 marks)

A.1 Push-Relabel (10 marks)

Please run the (n^3) Push-Relabel algorithm on the flow graph shown in Figure 1 with the following RELABEL-TO-FRONT strategy (after the usual initialization steps): "At each step, choose the vertex with excess at maximum height and if ties, the vertex with the smallest vertex number", "if we can push excess flow of a vertex u through more than one edge, we push that excess flow from u to neighboring vertex v that has the smallest vertex number", "otherwise, we relabel u by setting the height of u, i.e. $h(u)$, to be 1+min$(h(v) : (u, v)$ in the residual graph)".

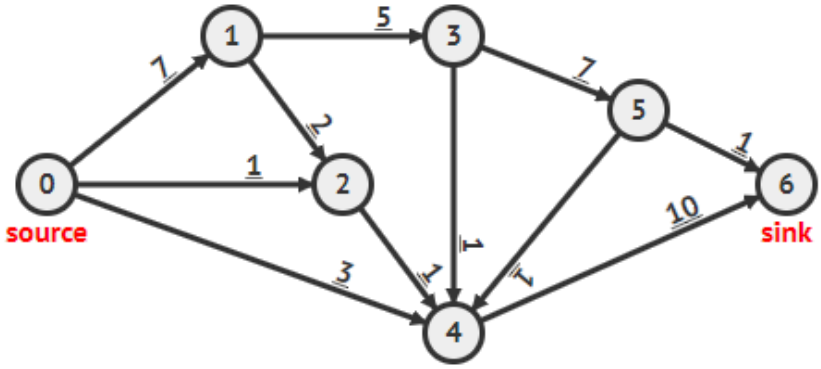

Figure 1: Perform Push-Relabel algorithm until the required terminating condition.

Now to check that you really use Push-Relabel algorithm instead of another Max Flow algorithm, please **stop** the algorithm after you perform the **first** relabel operation that makes a vertex u have height $h(u) > 7$, write down the **label** (the **height/h**) + the **excess/x** of each vertex and the residual capacities of each edge at this point of time in the empty Figure 2 below (up to 7 marks only if the next sub-question is correct).

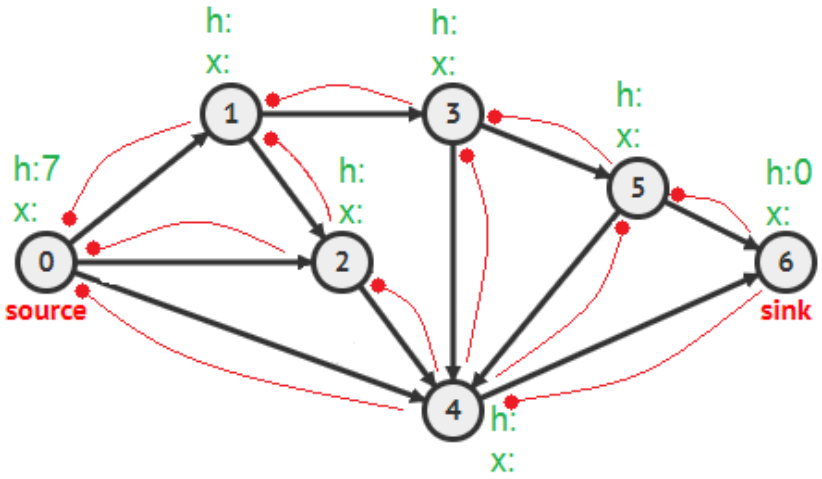

Figure 2: Write down your answer here.

To facilitate quicker grading, please mention the *unique* vertex number u where the **first** relabel operation that causes that $h(u) > 7$ occurs (and you stop the Push-Relabel afterwards) (3 marks):

A.2 Graph Matching (10 marks)

Solve the Max-Cardinality-Matching (MCM) problem on the unweighted undirected graph G with $n = 22$ vertices, $m = 28$ edges, and several connected components shown in Figure 3. Please highlight the edges that form the MCM by circling those edges directly in Figure 3 (up to 5 marks only if the next sub-question is correct).

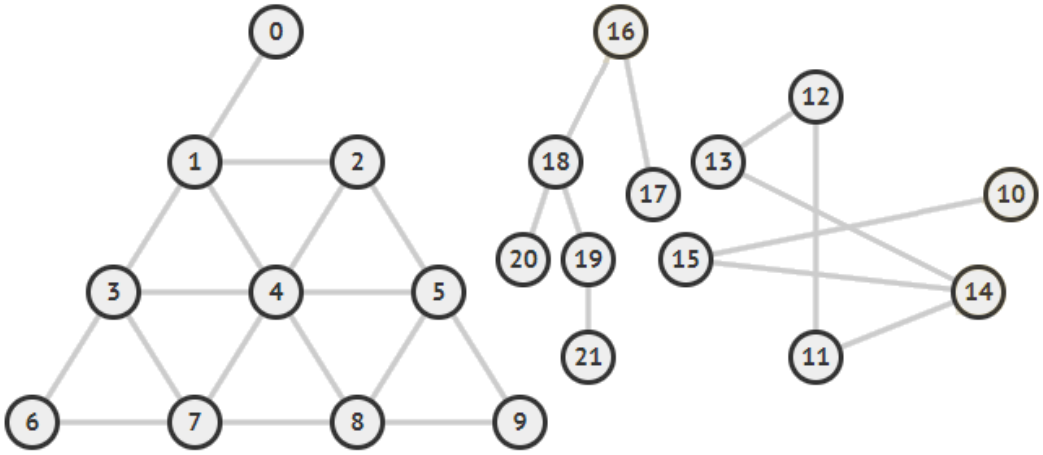

Figure 3: The input graph G; please circle the edges that form the MCM

To facilitate quicker grading, please write down the size (cardinality) of the MCM (i.e. the number of edges that you circled in Figure 3) (3 marks):

Additionally, is the MCM above a perfect matching? (1 mark)
Follow up sub-question (1 mark):
If yes, is it a **unique** perfect matching? Please justify your answer!
If no, can you **add just one edge to** G so that we are able to find a perfect matching on G.

A.3 Which Algorithm is the Best? (10 Marks)

For this section, you will be given five optimization problem scenarios (2 marks each). For each one of them, please mention the best algorithm and its time complexity so that the optimization problem is solved *correctly* with the lowest possible worst case time complexity.

1. Solve Max Cardinality Matching, the input graph is an *unweighted tree* with $n = 10\,000$.

2. Solve Weighted Max Cardinality Matching, the input graph is a *weighted tree* with $n = 10\,000$.

3. Solve Max Flow, the input flow graph is a weighted Directed Acyclic Graph (DAG) with $n = 100$ and $m = 4\,950$.

4. Solve Min Cut, the input flow graph only has k *unit-capacity* edges that goes out from source vertex s ($n = 100\,000, m = 200\,000, k = 7$).

5. Solve Metric Steiner Tree, the input graph is an *undirected unweighted tree* with $n = 100\,000$ and *all* vertices in the input graph are the required (terminal) vertices.

A.4 Statements About SLS (15 marks)

For each statement below about Stochastic Local Search (SLS) algorithm, determine if it is More towards True/It depends/More towards False and give a short explanation (3 marks per question).

1. We can make *any SLS algorithm* for Metric No-Repeat TSP to have a 2-approximation ratio.

2. In Tabu Search algorithm, setting high Tabu Tenure value/setting encourages diversification search strategy.

3. In ILS, it is better to use Acceptance Criteria that *always accept* newly found Local Optima to encourages exploration of Local Optima space.

4. Quadratic Assignment Problem (QAP) is quite close to Traveling Salesman Problem (TSP). In fact, most NP-hardness proof of QAP is derived from reducing NP-hard TSP into QAP. Thus, *any SLS algorithm* that empirically works well for TSP should also works well for QAP.

5. If we use Tabu Search for TSP, the best parameter setting for Tabu Tenure is a fixed constant 7, i.e. that is, forbid the last 7 local moves that Tabu Search has just performed.

Section A Marks = ___ + ___ + ___ = ___

B (NP?-)hard Questions (55 Marks)

B.1 Avoid Weekend? (15 marks)

Disclaimer: The following question is modified from a Kattis problem.

You (currently a single young student) have N study tasks ($1 \leq N \leq 100$). For simplicity, each task requires you to work one full day to complete, not longer and not shorter. You have up to 91 days (13 weeks, or about 3+ months) starting from tomorrow (day 1, a Monday), ... day 6 (a Saturday), day 7 (a Sunday), ..., day 13 (a Saturday), day 14 (a Sunday), ... until tomorrow+91 more days (the last day is day 91, a Sunday). Task i can be done as early as day e_i and as late as day l_i. The timings of these tasks may overlap and thus you may end up choosing to do only some of the given tasks.

If you can complete all N tasks without working on your study task on weekends (Saturdays and Sundays), output "STUDY-LIFE BALANCE". Otherwise if completing all N tasks is only possible if you also work on weekends, output "RULE 4 VIOLATION". Finally, if you cannot complete all N tasks even by working on weekends, output "OVERWORKED!!!".

Example 1: If you have $N = 2$ tasks and task 1 can be done between day [7..8] and task 2 can be done between day [6..9], then output "STUDY-LIFE BALANCE" as you can do task 1 on day 8 (a Monday) and task 2 on day 9 (a Tuesday), i.e. you avoid doing task 1 or 2 on day [6..7] (weekend).

Example 2: If you have $N = 2$ tasks and task 1 can be done between day [7..7] and task 2 can be done between day [6..9], then output "RULE 4 VIOLATION" as you can only do task 1 on day 7 (a Sunday) and task 2 on either day 6, 8 or 9 (doesn't matter due to weekend work for task 1).

Example 3: If you have $N = 2$ tasks and task 1 can be done between day [7..7] and task 2 can be done between day [7..7] too, then output "OVERWORKED" as you can only do either task 1 or task 2 on day 7 (a Sunday) but not both...

What is this problem? Design your best algorithm to decide this and **analyze its time complexity**!

B.2 Company Man (20 Marks)

Disclaimer: The following question is modified from a Kattis problem.

You work in an interesting software company that works in a peculiar way. In this company, all software projects must be done in a *team of two*. Moreover, the team must consists of *members from two divisions*: The programmers and the testers. Your boss wants to hold a grand meeting (in an exotic location, to be disclosed after this final assessment) and he wants to hear the latest update from each team. However, your boss is also a very cost-conscious businessman, thus he wants this meeting to be attended by as few relevant employees as possible.

You have access to the list of all the m teams ($1 \leq m \leq 10\,000$) in your company listed in form of distinct pairs of two integers (programmer-id, tester-id). You, the company's top programmer, are asked by the boss to determine the *minimum* size x of this grand meeting and to list down at least one possible candidate set of meeting attendees (of size x, of course). The ids of programmers range from [0..999] and the ids of testers range from [1 000..1 999]. As a personal reward for doing this, you will alter the list of candidates to always include yourself (programmer with id = 7), whenever possible.

Example 1: If $m = 2$ and the teams are $\{(7, 1\,000), (8, 1\,001)\}$, then the minimum $x = 2$ and you suggests to your boss to invite {programmer 7 (you), programmer 8} (or alternatively $\{7, 1\,001\}$).

Example 2: If $m = 2$ and the teams are $\{(7, 1\,000), (8, 1\,000)\}$, then the minimum $x = 1$ and you have no choice but report to your boss to invite only {tester 1 000} to update him on both teams.

What is this problem? Design your best algorithm to decide this and **analyze its time complexity**!

B.3 Min-Vertex-Cover, Finale (13 Marks)

Suppose that you are given several *general* graphs with n up to 1 000 vertices and m ranging from 0 up to $n \times (n-1)/2$ (a complete graph). You are tasked to find the sizes of the Min-Vertex-Cover of those graphs. If the optimal answer for one of the graph G is x and your answer for that graph G is y, your score is 0 (for this entire question) if $y \geq 2x$. Otherwise, your score is $13.0 - (y-x)/x * 13.0$. Using *any* algorithm(s) or technique(s) that you have learned in this module, propose how you are going to deal with this scenario. You will be graded by the detail and soundness of your proposal.

B.4 New Game* (7 Marks)

Disclaimer: The following question is modified from a Kattis problem.

You are currently standing at origin (0, 0) and have a stack of N ($1 \leq N \leq 20$) heavy stone bricks located at origin. You need to move these N heavy stone bricks to N different target locations. The integer (x, y) coordinates of these N locations are given (all coordinate values are between -100 to 100). No target location is at origin. Each location can only be used to store **one** heavy stone brick. As the bricks are heavy, you can only carry up to **two** bricks at a time. Your task is to minimize the **minimum walking distance** to complete this game.

Example 1: $N = 1$, location 1 = (4, 0). Output: 4.0 (note: the output may not be integer).
Example 2: $N = 2$, location 1 = (4, 0), location 2 = (4, 3). Output: 7.0.
Example 3: $N = 3$, location 1 = (4, 0), location 2 = (4, 3), location 3 = (1, 1). Output: 9.828...
Example 4: $N = 3$, location 1 = (4, 0), location 2 = (4, 3), location 3 = (-6, 0). Output: 18.0

Section B Marks = ___ + ___ + ___ + ___ = ___

– End of this Paper, All the Best, You can use this Page 10 for extra writing space –