National University of Singapore

School of Computing

CS4234 - Optimisation Algorithms

(Semester 1: AY2022/23)

Date and Time: Thursday, 06 October 2022, 12.02-13.32 (90m)

INSTRUCTIONS TO CANDIDATES:

1. You can start immediately after you are given the password to open this file.
2. This assessment paper contains FOUR (4) sections.
   It comprises FOURTEEN (14) printed pages, including this page.
3. This is an **Open Book Assessment**.
   Additionally, you can also use your laptop too (but in airplane mode).
4. Answer **ALL** questions within the **boxed space** of the answer sheet (page 9-14).
   You will only need to hand over page 9-14 after this quiz.
   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 (subtask) 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, e.g. run Dijkstra's on graph G, run Kruskal's on graph G', etc.
7. All the best :)

A The Simpler Questions ($6 \times 5 = 30$ marks)

A.1 Constructive Feedback about VisuAlgo Online Quiz System (5 marks)

On 04 October 2022, 9-11pm, most CS4234 students were stress testing 'almost the end product' of one of Steven's FYP student who is tasked to improve VisuAlgo Online Quiz system, including adding new questions on 5 modules that are used in the first half of CS4234 lectures: /mvc, (only missing set cover), /steinertree, /tsp, /maxflow, and /matching. At the start of this paper, please quickly write down your short (but must be constructive) feedback (a few sentences) in the answer sheet. Your feedback will be anonymised in the FYP report and enjoyed by future CS4234 students.

A.2 Special cases of `Min-Weight-Vertex-Cover` (5 marks)

In class, we discussed the `Min(-Weight)-Vertex-Cover (M(W)VC)` problem in details. There are a few special cases (special input graph type) of this NP-hard optimization problem that admits polynomial solutions. One such special case is as follows: *"If the input graph is a weighted Tree, we can use $O(V)$ Dynamic Programming (DP) to get the optimal answer"*. Your job is to list down *at least five more* special cases (1 mark each) of `MWVC` (the weighted version), the polynomial solutions for that special cases, and their polynomial time complexities.

A.3 Create a `Min-Vertex-Cover` test case with a special property (5 marks)

In Lecture 02, we learned about the deterministic `Approx-Vertex-Cover-2` algorithm where each edges in E' that are considered by this algorithm is a matching M. We see that $|\texttt{MVC}| \geq |M|$ and we use that bound to show that `Approx-Vertex-Cover-2` is 2-approx.

Your task: Draw or describe any simple and small undirected unweighted graph with exactly $|V| = 10$ vertices (the number of edges $|E|$ is up to you) so that $|\texttt{MVC}| = |M| = |V|/2$. Give a short explanation of why your answer satisfies the requirements.

A.4 Create a `Min-Vertex-Cover` test case with another special property (5 marks)

Your task (next level): Draw or describe any simple and small undirected unweighted graph with exactly $|V| = 10$ vertices $14 \leq |E| \leq 45$ so that $|\texttt{MVC}| = 2$. Give a short explanation of why your answer satisfies the requirements.

A.5 Can we do `Max-Flow` $\leq_p$ `MCBM`? (5 marks)

In Tut 04 and in Lecture 06, we have seen a simple $O(V)$ polynomial time reduction: Reduce any instance of an optimization problem `MCBM` (from Lecture 06) into an instance of another optimization problem `Max-Flow` (from Lecture 05) by adding a super source vertex s that connects to all vertices on the left set and then connect all vertices on the right set to a super sink vertex t. Set all edges to be directed from s to t and all such directed edges have capacity 1. The max flow value $|\texttt{MF}|$ of this flow graph will be equal to the $|\texttt{MCBM}|$. This `MCBM` $\leq_p$ `Max-Flow` reduction establishes that both problems have polynomial solutions.

Now, can we do similar reduction in the other direction, i.e., `Max-Flow` $\leq_p$ `MCBM`? If we can, just give a simple argument. If we cannot, also just give a simple argument.

A.6 Two Augmenting Path Algorithm Variants (5 marks)

Part 1 (this is independent from Part 2 below, i.e., nothing changed in the `int main()`, 2 marks): Starting from the original `reference.cpp` code zipped together with this e-question paper (it has been simplified: not using the randomized greedy pre-processing and we process free vertices on the left set one by one from lower vertex number), decide if the code `int Aug(int L)` remains correct if the first two lines are commented (`if (vis[L]) return 0;` and `vis[L] = 1;`)?

```
int Aug(int L) {
  if (vis[L]) return 0;                          // Part 1, can this be commented?
  vis[L] = 1;                                    // Part 1, can this be commented?
  for (auto& R : AL[L])
    if ((match[R] == -1) || Aug(match[R])) {
      match[R] = L;
      return 1;
    }
  return 0;
}
```

Part 2 (this is independent from Part 1 above, i.e., the first two lines of `int Aug(int L)` remain **NOT** commented as with the original implementation, 3 marks): Starting from the original `reference.cpp` code zipped together with this e-question paper (it has been simplified: not using the randomized greedy pre-processing and we process free vertices on the left set one by one from lower vertex number), decide if the code portion in `int main()` that controls the calls of `int Aug(int L)` remains correct (and become faster?) if we move the (`vis.assign(VLeft, 0);`) outside the loop that tries all free vertices on the Left set?

```
  match.assign(V, -1);                           // this one is only initialized once
  int MCBM = 0;
  // we don't use randomized greedy pre-processing
  vis.assign(Vleft, 0);                          // for Part 2, uncomment this
  for (int L = 0; L < Vleft; ++L) {              // try sequentially
    // vis.assign(Vleft, 0);                     // for Part 2, comment this
    MCBM += Aug(L);
  }
```

B `Graph-Coloring` on Planar Graphs (20 marks)

In Tut01 Q5, you were given this Graph-Coloring problem: "Given a graph $G = (V, E)$ and an integer k, can we assign a color (an integer in $[1..k]$) to each vertex such that no two adjacent vertices (that share an edge) are of the same color? The Graph-Coloring optimization variant that will be implemented in VisuAlgo in the future seeks for the *actual coloring* with the smallest possible k.

Steven asked what is the best way to do this if VisuAlgo has *one additional graph drawing constraint*, i.e., the drawn graph must be planar.

The 2022 model answer says "For planar graph, the four-color theorem says that we can color the planar graph with at most four colors. A simple backtracking algorithm that tries all color per vertex has a worst case time complexity $O(4^{|V|})$ but early pruning for any infeasible solution may help avoid this worst case behavior. We probably can do faster with DP with subset but so far, there is no known specific planar graph property that can be used to solve GRAPH-COLORING in polynomial time."

Your task is to further improve upon this model answer by using the following sub-questions.

B.1 Partition any Planar Graph into two Bipartite subgraphs (7 marks)

There is another property of planar graph that can lead to a faster solution for this special case. First, give a proof that *any* planar graph can be decomposed into two disjoint Bipartite Graphs.

B.2 Pre-process Two Special Cases First (3 marks)

For planar graph, the four-color theorem says that it needs *at most* four colors.
What if the planar graph is actually 1-colorable (1 mark) or 2-colorable (2 marks)?
What simple pre-processing routine that we can do to detect and process these two special cases?

B.3 Use the Idea Above (10 marks)

Now design an algorithm that is still exponential in the time complexity, but uses the fact that any planar graph can be partitioned into two Bipartite subgraphs, to find any one valid 3-colorable or 4-colorable solution faster than Tut01 Q5 2022 model answer. What is the faster time complexity?

C `Partition-Into-Triangles?` (25 marks)

Steven is the coach of NUS International Collegiate Programming Contest (ICPC) teams. This year, NUS is registering several teams to participate in ICPC Asia Ho Chi Minh City Regionals programming contest 2022. The ICPC is participated by teams of three contestants.

In NUS, there are n ($1 \leq n$ and n does not have to be a multiple of 3) eligible contestants, numbered from 0 to n-1. Each contestant $i \in [0..n\text{-}1]$ has a CodeForces rating of `cf[i]` (each rating is a non-negative integer up to 10^8 — in reality, the current highest CodeForces rating at the time of this paper is 'only' 3757). Steven defines the skill level of a *team* consisting of contestant i, j, and k to be $min(cf[i], cf[j], cf[k]) + max(cf[i], cf[j], cf[k])$. PS: This is **not** how Steven selects NUS ICPC teams 2022 and this definition is only applicable for this problem.

Steven want to only register teams with a skill level of *strictly more* than S, to avoid embarrassment in the actual contest. Each programmer may only be assigned to at most one registered team. Steven wants to know the *maximum* number of teams that he can register.

Example 1: If Steven wants a few strong teams, e.g., $S = 5127$ and there are $n = 8$ contestants with these 8 CodeForces ratings of $\{2759, 2681, 2622, 2863, 2390, 2436, 2733, 2447\}$, then Steven can registers at most 2 teams, e.g., team 1 consists of contestants $\{3, 0, 6\}$ with skill level $min(2863, 2759, 2733) + max(2863, 2759, 2733) = 2733 + 2863 = 5596$ and team 2 consists of contestants $\{1, 2, 7\}$ with skill level $min(2681, 2622, 2447) + max(2681, 2622, 2447) = 2447 + 2681 = 5128$. Note that there are other possible solutions with 2 teams.

Example 2: For the same $n = 8$ contestants with the same 8 CodeForces ratings, if Steven sets $S = 5595$, then only 1 team can be registered (only the 3 contestants with the highest ratings).

Example 3: For the same $n = 8$ contestants with the same 8 CodeForces ratings, if Steven sets $S = 5596$, then 0 team can be registered and he has to lower his expectation.

C.1 Two Manual Test Cases ($2 \times 2 = 4$ marks)

If you have understood this question, what is the answer (the maximum number of teams to be registered) if you are given:

1. $n = 9$, $S = 4099$, and the 9 CodeForces ratings are
   $\{1600, 2400, 1700, 2100, 1900, 2000, 2200, 1800, 2300\}$?
2. $n = 10$, $S = 4299$, and the 10 CodeForces ratings are
   $\{1900, 2225, 2050, 1950, 2100, 2350, 1800, 2200, 2150, 2400\}$?

To convince the grader that your answer is not a random guess, show the members of each teams as with Example 1 above.

C.2 Subtask 1 (3 marks)

For easy 3 marks, design an $O(1)$ algorithm that can solve this problem when $n \leq 3$.

C.3 Subtask 2 (3 marks)

For another 3 marks, design an $O(n^3)$ algorithm which is technically also $O(1)$ that can solve this problem when $n = 6$ (a fixed constant).

C.4 Is this an NP-hard optimization problem? (1 mark)

If you say it is, give a short justification, and you are allowed to use an exponential algorithm for Subsection C.5. If you say it is not, you must use a polynomial solution for Subsection C.5.

C.5 Solve This Problem (14 marks)

One of the route in Subsection C.4 is correct.
The correct route worth the full (14) marks.
The other one worth only 7 marks (2-opt of the optimal marks).

D `Bipartite-Edge-Set`: General → Bipartite Graphs (25 marks)

In Week 06, we discuss the importance of Graph Matching on Bipartite Graphs (MCBM). Steven wants to create a new MCBM problem for an upcoming programming contest problem and he needs a lot of Bipartite Graph test cases. He currently has a bunch of *unweighted undirected* graph test cases. For test case that is already bipartite, he doesn't need to do anything. But for test cases involving non-Bipartite graphs, deleting *some* of their edges may turn them into bipartite graphs[1] that he can use for his new MCBM problem. Steven wants to keep as many edges (i.e., delete as few edges) as possible in those non-Bipartite graphs when doing this non-Bipartite to Bipartite conversion.

Formally, `Bipartite-Edge-Set` can be stated as a problem of finding the largest subgraph $G' = (V, E') \subset G = (V, E)$ that is bipartite.

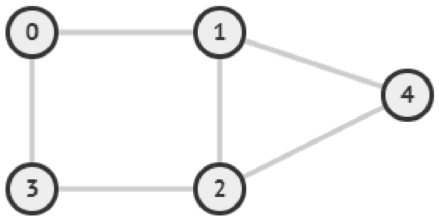

Figure 1: Example of a non-Bipartite Graph

For example, see Figure 1 which is a non-Bipartite graph (notice odd length cycle $1-2-4-1$). If we delete just 1 edge $(1,4)$ (or edge $(2,4)$), we will then have a Bipartite graph (verify it yourself). Note that deleting edge $(1,2)$ does not help as we still have another odd length cycle $0-1-4-2-3-0$. As we can already convert Figure 1 from a non-Bipartite graph into a Bipartite graph by just deleting 1 edge (and keeping the other 5 edges), then this is the optimal answer (we do not need to delete any other edge).

[1]This is somewhat similar to the `Min-Feedback-Arc-Set` problem in Tut02 Q4.

D.1 Two Manual Test Cases ($2 \times 2 = 4$ marks)

If you have understood this question, what are the answers (which edges to delete) if you are given Figure 2: Graph A and Graph B? To convince the grader that your answer is not a random guess, list down the edges that you delete (this implies that you keep the other non-deleted edges).

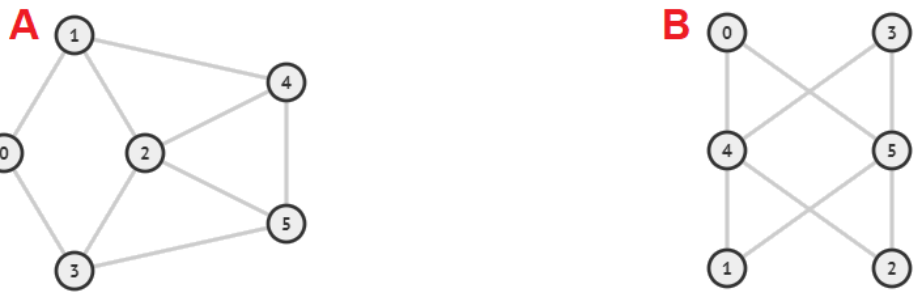

Figure 2: Two Manual Test Cases: Graph A (left) and Graph B (right)

D.2 Do you know the actual problem name? (1 mark)

The name `Bipartite-Edge-Set` in this section is not the real problem name.
If you know the real name of this problem, mention it.

D.3 An Algorithm

Without giving you the option to choose, Steven says that this optimization problem is really NP-hard (proof omitted). For this question, we will describe an iterative[2] algorithm:

```
1. Split V into 2 partitions arbitrarily, e.g., A = V (all vertices), B = {} (empty).
2. For ANY vertex v in V which has < 1/2 of its edges crossing the two partitions,
   move that vertex v from one of the partition to the other partition
3. If no such vertex exists then go to step 4, else go back to step 2.
4. Keep edges that crosses the two partitions (bipartite edges)
   and delete edges within partition A only and partition B only.
```

An example run of this iterative algorithm using Figure 1 (in the previous page) is as follows:

- Initially, $A = \{0, 1, 2, 3, 4\}$ and $B = \{\}$.
- Perhaps, we pick vertex 1 (notice the keyword ANY) that has 0 edge crossing the two partitions and 3 edges all within set A; we move vertex 1 across, so $A = \{0, 2, 3, 4\}$ and $B = \{1\}$.
- Next, we pick vertex 3 (again, remember the keyword ANY) that has 0 edge crossing the two partitions and 2 edges all within set A; we move vertex 3 across, so $A = \{0, 2, 4\}$ and $B = \{1, 3\}$.
- At this stage, no other vertex has $< 1/2$ of its edges crossing the two partitions, so we stop.
- We delete edge $(2, 4)$ as this edge is within partition A only and keep the 4 other edges. We now have a bipartite graph after only deleting 1 edge (optimal, for this run).

[2]The concept of 'iterative algorithm' is is similar to Max Flow and Matching algorithms discussed in class.

D.4 Analyze that Algorithm (5 marks)

Prove that the algorithm in Subsection D.3 will eventually terminates.
Then, analyze its time complexity! Is it polynomial or exponential?

D.5 The algorithm in Subsection D.3 can give sub-optimal solution (5 marks)

Create a simple/small test case (do not overthink) and do another run of the algorithm in Subsection D.3 on your test case and clearly show that the algorithm *doesn't always* give the optimal solution for this `Bipartite-Edge-Set` problem.

D.6 The algorithm in Subsection D.3 is 2-approx (10 marks)

Prove that the algorithm in Subsection D.3 is actually a 2-approx algorithm!
That is, it will keep at least half of the edges in $G = (V, E)$ (which may be non-bipartite) when it turns G into $G' = (V, E')$ after deleting some (possibly none) edges.

The Answer Sheet

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	30		
B	20		
C	25		
D	25		
Total	100		

Box A.1. The giveaway marks: give your constructive feedback for VA OQ system.

Box A.2. Special cases of `MWVC`.

Box A.3. `MVC` with a special property.

Box A.4. `MVC` with another special property.

Box A.5. `Max-Flow` $\leq_p$ `MCBM`?

Box A.6. Two Augmenting Path Algorithm Variants.

Box B.1. Any planar graph can be partition into two Bipartite subgraphs

Box B.2. Two special cases

Box B.3. Use the idea above

Box C.1. Two manual test cases

Box C.2. Subtask 1, $1 \leq n \leq 3$

Box C.3. Subtask 2, $n = 6$

Box C.4. NP-hard?

Box C.5. Solve this!

Box D.1. Two manual test cases

Box D.2. Say the actual problem name

Box D.4. Analyze that algorithm (D.3 is not a question).

Box D.5. That algorithm does not always give optimal solution

Box D.6. That algorithm is 2-approx

End of this Paper, All the Best