CS 180: Introduction to Algorithms and Complexity
Final Exam

March 17, 2021

Name	
UID	
Section	

1	2	3	4	5	6	Total

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

★ **Your Exams need to be uploaded in Gradescope. Print your answers, or use a dark pen or pencil. Handwriting should be clear and legible.**

- Do not write code using C or some programming language. Use English in bullet points or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
- Your answers are supposed to be in a simple and understandable manner. Sloppy answers are expected to receive fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.

1. **(a.)** Consider the problem of "Approx-3SAT": The input is the same as 3-SAT, which is a boolean expression $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where each C_i is an "or" of three literals (each literal can be one of $x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m$). Note that we assume a clause cannot contain duplicate literals (e.g., $(x_1 \vee x_1 \vee x_2)$ is not allowed). Instead of determining whether there's a truth assignment on $\{x_i\}_{i=1}^m$ that satisfies this boolean expression (which means it satisfies all the clauses), now we want to determine whether there's an assignment that satisfies at least $n-1$ clauses.
Prove that Approx-3SAT is NP-Complete.

   (b.) Give poly time reduction of Hamiltonian Cycle in an undirected graph to a Hamiltonian path in an undirected graph. I.e. if HAM path is poly then HAM cycle is poly. (Hint: Replicate some node twice, and now add "something" so that a HAM path in the new graph will not be in the "middle" of the path, but toward the "end." Alternatively, use cook reduction which allows to invoke HAM path |E| times.)

   (c.) Although Vertex Cover (V.C.) is NP-Complete we want to show we can get a vertex cover that is at most greater by a factor of 2 than the size of the optimal V.C. in poly time.

   - A Maximal size (number of edges) Matching in an undirected graph G is a Matching M such that no edge e can be added to it such that $M \cup \{e\}$ is a matching in G. Give a linear time algorithm for for Maximal size matching assuming G is given by adjacency list.
   - Argue that the vertices which are adjacent on the edges in a Maximal size Matching constitute a vertex cover.
   - Argue that the optimal size V.C. is larger equal to the size of a maximal size matching.

Solution 1

(a) It's easy to show this is NP. Given a certificate, simply iterate over the clauses and see they are all T. This takes linear time.

For NP-completeness, we'll show that 3-SAT can be reduced to this problem. Given a 3-SAT, add a set of dummy clauses that there's always exact one clause not satisfiable. For example, we can add $(z_1 \vee z_2 \vee z_3)$, $(z_1 \vee z_2 \vee \bar{z}_3)$, $(z_1 \vee \bar{z}_2 \vee z_3)$, $(z_1 \vee \bar{z}_2 \vee \bar{z}_3)$, $(\bar{z}_1 \vee z_2 \vee z_3)$, $(\bar{z}_1 \vee z_2 \vee \bar{z}_3)$, $(z_1 \vee \bar{z}_2 \vee z_3)$, $(z_1 \vee \bar{z}_2 \vee \bar{z}_3)$, so there will be exactly one violation and the 7 of them are satisfied for any truth assignment. Therefore, this newly constructed instance (by adding those dummy cluases) will be yes instance of Approx-3SAT if and only if the original instance is a yes instance for 3-SAT.

(b) `HamiltonianCycle` $\le_p$ `HamiltonianPath` Given an instance of `HamiltonianCycle` G, we create a instance of `HamiltonianPath` G': Let G' be copy of G; add a new vertex t and choose a arbitrary node in G as the source node s; for every node v in G that has an edge (v,s), add an edge (v,t) in G'.
Again it can easily be shown that there is a Hamiltonian cycle in G if and only if there is a Hamiltonian path from s to t in G'.

(c) - $M \leftarrow \emptyset$ - While $E \neq \emptyset$ do - select $(u,v) \in E$ - $M \leftarrow M \cup \{(u,v)\}$ - delete all edges incident to u and v
- Return M
The above algorithm runs in $O(m+n)$ time if we store $G=(V,E)$ using adjacency list.

- A matching or independent edge set in an undirected graph is a set of edges without common vertices. As we have seen in class, given that a set X of vertices contained in our matching (vertices on one side of all chosen edges) there will exist a vertex cover of the set V/X, or the set of vertices neighboring all other vertices.

- Now we prove this algorithm is 2-approximation. Let OPT be the maximum matching. For each edge $e \in$ OPT, there must exist $e' \in M$ s.t. $|e \cap e'| \in \{1,2\}$, i.e. e and e' share one or two common vertex. Moreover, at most two different edges of OPT are incident to the same edge of M.
Now we divide OPT into two disjoint subsets OPT$_1$ and OPT$_2$. In OPT$_1$, no two edges are incident to the same edge in M. In other words, M has a subset M_1 s.t. OPT$_1$ and M_1 are one-to-one, which implies $|OPT_1| = |M_1|$. We can pair all edges in OPT$_2$ s.t. different pair of edges in OPT_2 are mapped to different edges in $M_2 \subseteq M$. It's easy to show that $M_1 \cap M_2 = \emptyset$ and $M_1 \cup M_2 = M$.

We have $|M| = |M_1| + |M_2|$ and $|OPT| = |OPT_1| + |OPT_2| = |M_1| + 2|M_2|$. Therefore, $2|M| \geq |OPT|$.

2. There is an array with n integers, but the values are hidden to us. Our goal is to partition the elements into groups based on their values — elements in the same group should have the same value, while elements in different groups have different values. The values are hidden to us, but we can probe the array in the following way: we can query a subset of these n elements, and get the number of unique integers in this subset. Design an algorithm to partition these n elements in $O(n \log n)$ queries. (Hint: Assume inductively that you have solved the problem over the elements observed so far. Consider how to "observe" another element.)

Solution 2

Assume we have partitioned the first m elements into K groups such that elements in the same group have the same value and elements in different groups have different values. Now we consider the $(m+1)$-th element, we first query $\{1, \ldots, m, m+1\}$. If this outputs $K+1$, then we know that the $(m+1)$-th element is in a new group different from existing K groups. Otherwise, if the output of the query is K, then we know the $(m+1)$-th element must belong to one of the K groups.

In the latter case, we need to determine which group the $(m+1)$-th element belongs to. To this end, we denote these groups as $G_1, \ldots, G_K$. We can first query the subset $\{m+1\} \cup G_1 \cup \ldots G_{K/2}$ which contains elements $m+1$ and all the elements from the first $K/2$ groups. Based on the output, we know whether element $m+1$ is in the first $K/2$ groups or the other half of groups. Once we decide which half it belongs to, we can again divide these groups into two halves and repeat the above process. Note that there are K groups in total, which means we will repeat this recursion at most $O(\log K)$ times before we know which of the K groups the $(m+1)$-th element belongs to.

Iterate the whole process from $m = 0$ to $m = n-1$, we finish the grouping of these n elements. The time complexity (total number of queries) is $O(\sum_{m=0}^{m=n-1} \log K) = O(n \log n)$. The pseudo code is displayed as follows.

```
arr with $n$ elements
Group[1]={1}; Group[i]={} for all $i \neq 1$
K=1
**for** $m = 1, \ldots, n$
  **if** query(arr[m+1], Group[1], ..., Group[K]) = K+1
    Group[K+1]={m+1}
    K ++
  **else if** query(arr[m+1], Group[1], ..., Group[K]) = K
    $l = 1$, $r = K$, $mid = (r - l + 1)/2$
    **while**($r - l > 0$)
      $mid = (r - l + 1)/2$
      **if** query(arr[m+1], Group[$l$], ..., Group[$mid$]) = $mid - l + 1$
        $r = mid$
      **else if** query(arr[m+1], Group[$l$], ..., Group[$mid$]) = $mid - l + 2$
        $l = mid$
    Group[$l$]={Group[$l$], m+1}
```

3. **(a.)** Prove that Euclidean version of the Traveling Salesmen Problem (TSP) is NP-Complete. Euclidean TSP is a special case of general TSP, where distances in the complete graph obey the triangle inequality, i.e for each three edges that constitute a triangle the sum the weights of any two, is greater equal than the weight of the third edge.

   (hint: You can use the same kind of reduction (tweaked) we used in class to prove that the general TSP is NP-Complete.)

   (b.) Given a weighted complete graph $G = (V, E, w)$ with positive weights. Prove that the weight of an MST of G is lower equal than the optimal value of a TSP in G.

   (c.) Consider a pre-order (e.g. a DFS that allocate the next number upon the discovery of a new node) enumeration of the nodes of an MST of Euclidean G starting at an arbitrary node. We could use that sequence of nodes to traverse our graph G, i.e we traverse in the order $1 \longrightarrow 2 ... \longrightarrow n \longrightarrow 1$. Show that the total value of this traversal is less then 2*opt(TSP) solution.

   (d.) Prove that the size of the set of odd degree nodes S in an MST in G is even.

   (e.) There exists a polynomial time algorithm for finding minimum weight perfect matching in a complete weighted graph $G' = (V', E'.w')$ where $|V|$ is even.

   Consider the minimum weight matching in the weighted graph G' where G' is the graph induced by the weighted graph G over the set of nodes S. Argue that the total value of this minimum weight matching is at most $\frac{1}{2}*$opt(TSP) of G.

   (f.) Consider the bag of edges that contains the edges of an MST of the euclidean G plus the edges of the minimum weight perfect matching of S nodes of this MST. Show that three exists a traversal of this set of edges where every edge in the bag is traversed only once (if the bag contains two copies of the "same" edge then this edge is traversed twice).

   Also. prove that the sum of the the weights of the edges in the bag is at most $1.5*$opt(TSP).

   (g.) Using pre-order as before find a TSP whose length is at most 1.5opt(TSP)

Solution 3

a. Let HS be the Hamiltonian cycle problem. We'll show that HS $\leq_P$ Euclidean TSP. Given a graph G=(V,E) on which we want to solve the HS problem.
   Let us define a new graph $G^{'} = (V^{'}, E^{'})$ where $V^{'} = V$ and $E^{'} = (u, v)$ for any $u, v \in V^{'}$. For edges in $G^{'}$ that are also in G we assign the weight 1, and for other edges we assign the weight 2. If $G^{'}$ has a Hamiltonian cycle, then there is a solution to the TSP problem with a final sum $\leq |V| + 1$. If there is a TSP of length $|V| + 1$, then every distance between points in the cycle must be 1.

   *Notice that 0s and 1s do not translate to the Euclidean TSP problem.
b. Consider the optimal solution to TSP in G. If we remove a single edge from the solution we will get a tree. the MST is the minimal possible tree representing the graph G therefore the weight of the MST in $G \leq$ then the optimal TSP.
c. Considering the order in which MST discovers nodes, one node is added into the MST set at each iteration. When a node is included, it is connected to the current discovered set by an edge, In the worst case for our traversal, each discovered node will be connected the same root node causing our traversal to essentially repeat each edge twice. $1 \longrightarrow 2 \longrightarrow 1 \longrightarrow 3 \longrightarrow 1 ... \longrightarrow n$. Overall this traversal will be exactly 2*opt(TSP) and any other MST case will be less than that.
d. Every edge in a graph is connected to 2 nodes. the degree of a node is the number of edges it is incident with. From this we can get that the sum of degree equals twice the number of edges. If the graph had an odd number of odd degree nodes the sum was odd as well.

e. Consider the tour opt(TSP) of the graph G. If we follow the path and remove the least weighted edge connected to each node we will end up with a matching of size at most $\frac{1}{2}$opt(TSP). Since $G^{'}$ contains less edges than G the matching will be strictly less than the matching we found for G.

f. We know that $|S|$ is even. Add those edges to complete Euclidean tour of G and the degree of each node in the bag remains even and therefore a complete tour is possible. Since we have shown that the sum of the minimum weight matching is at most $\frac{1}{2}opt(TSP)$ and a complete MST is at most $opt(TSP)$ their sum will be at most $\frac{3}{2}opt(TSP)$.

g. We can traverse the graph in the following way. We start traversing the graph based on the pre-order. Whenever we encounter a next node who we don't have an edge to from the current node, we use the edge available from the maximal matching set. We know this will result in a complete tour of the graph as been shown before and its weight will be at most $\frac{3}{2}opt(TSP)$

4. **(a.)** CS students participate in many recognized Student Groups e.g. ACM-W, UPE, etc.. A student may be a member of many student organizations. Once a year the Department holds an advocacy session in which a single member of a group is advocating for an increase in the budget of a group. But, to avoid seeing same face for too long, a member can only advocate for a single group among the groups the student belongs to.

   We want to find whether each group can choose a unique member to represent it.

   E.g. If we have two singelton groups containing the same student then obviously the problem has no solution.

   Given groups and their membership determine whether choosing unique member from each group is NP-Complete problem or solvable in polynomial time. Either prove the NP-Completeness, or give a Poly-time algorithm.

   (b.) Given an undirected connected graph $G = (V, E)$,

   - A cut is the set of edges connecting nodes in different part in a 2-partition of the set of nodes of the graph. A cut is minimal, if no subset of it is a cut. Argue that the two parts corresponding to a minimal cut, each is connected subgraph of G.
   - Give an algorithm to find a minimum-cut *min-cut*$_{a,b}$. A min-cut *min-cut*$_{a,b}$ is a min-cut of G under the constraint that two given nodes $a \in V, b \in V$ are in two different partitions induced by the min cut.
   - Give a polynomial time reduction (Cook reduction) from min-cut of G to *min-cut*$_{a,b}$ for various a and b. Analyze the number of times *min-cut*$_{a,b}$ was called.
   - Given a partition A, B of the nodes $v_0, v_1, ...v_{n-1}$ of G, prove that there exist a number $i \in \{0, 1. ... n-1\}$ such that $v_i \in A$ and $v_{(i+1) \bmod n} \in B$.
   - Describe an algorithm the finds min-cut of G that calls on *min-cut*$_{a,b}$ at most n times.

Solution 4

(a) This is almost identical to the matching problem we have seen in class. We can reduce it to a network flow problem.

For each member we define a node $a_i \in A$ and for each group a node $b_i \in B$. A member has an edge a_i, b_i if a_i is a member of b_i with weight 1. We connect a super source node to all members with all edges weight of 1 and a super sink connected to all nodes B with weight 1. We then run Ford-Fulkerson and if max flow is $|B|$ then a unique member can be chosen for each group. Total run time $O(FE)$

* notice that if the question asked if we can find at most k members representing all groups, then the problem was NP-Complete.

(b)
- By contradiction, if a part was not a connected component, there must have been a vertex in the non connected component connected by an edge belonging in the minimal cut. However, if we remove that edge from the cut, it remains a minimal cut. Contradiction.
- We can assign a as source and b as a sink, and run Ford-Fulkerson. By the min-cut theory we are guaranteed that a and b will be on opposite sides of the min-cut in the resulting graph.
- We can reduce from min-cut to min-cut$_{a,b}$ by running our running our previous algorithm for every pair a, b and choosing the minimal max flow graph. The process will run for every pair once (a, b and b, a) are the same) so $\binom{|V|}{2}$.
- The idea is to call each node v_i and its neighbor $v_{(i+1) \bmod n}$ and check their participation in a partition. By the definition of a partition of G there must be a pair of nodes that will be in two different partitions. This will take us $O(n)$ time.

- Based on the last algorithm, we know for sure that some node v_i and its neighbor $v_{(i+1) \bmod n}$ will be in two different partitions. So, we can just call our algorithm for min-cut$_{a,b}$ n times, for each node and its MOD neighbor. The min-cut will be the minimum of those n calls.