CSI 4105 Design and Analysis of Algorithms II — Winter 2012
Computer Science — University of Ottawa

Homework Assignment #1 (100 points, weight 6.67%)
Due: Friday Feb 10, at 11:30 p.m. (in lecture)

---

1. (20 points) Let LongestPath denote the decision problem that asks, given a graph $G = (V, E)$, two vertices $s, t \in V$ and an integer k, whether G has a simple path between s and t with at least k edges.

   Consider the optimization problem FindLongestPath that given a graph $G = (V, E)$ we need to find the longest simple path between s and t in G.

   Prove that if LongestPath $\in \mathcal{P}$, then the problem of listing the longest simple path between given vertices s and t, FindLongestPath, can be solved in polynomial time.

   Hint: Design an algorithm for FindLongestPath *that performs a polynomial number of steps plus makes a polynomial number of calls to the algorithm that solves* FindLongestPath. *Justify that your algorithm does the promised job (i.e. is correct and runs in polytime).*

   The algorithm detailed in Algorithm 1 finds vertices s and t and integer k such that a longest simple path, being of length k, lies in G between s and t. We show that this algorithm executes in polynomial time by showing that it makes a polynomial number of calls to LongestPath. Indeed, it makes precisely $|V|^3$ such calls, and thus if LongestPath runs in polytime, then so does FindLongestPath1.

   We now give a second algorithm, in Algorithm 2, that, given a graph $G = (V, E)$, vertices $s, t \in V$, and an integer k such that there is a longest path of size k in G from s to t, finds the path and returns it. The first for loop executes in polynomial time: it iterates $|V| - 2$ times, and for each iteration, makes one call to LongestPath, which is polytime, and possibly removes a vertex from the graph and performs a graph copy, which both can be done in polynomial time. The second for loop also executes in polynomial time: it iterates at most $|E|$ times, and for each iteration, makes one call to LongestPath, which is polytime, and possibly removes an edge from the graph and performs a graph copy, which both can be performed in polytime.

   Now it remains to show that the graph returned by Algorithm 2 is an s-t path of size k: the construction of the algorithm ensures that the final graph, $G' = (V', E')$, has the property that LongestPath(G', s, t, k) returns true, so G' necessarily contains an s-t path of size k. We now show that V' has exactly k vertices: assume it has more than k vertices. Then there is an s-t path in G' of length k not involving some vertex $v \in V'$. But then $G' \setminus \{v\}$ still contains an s-t path of length k, so v would have been removed from G' by the first for loop, and thus, by contradiction, V' has exactly k vertices. We now show that G' contains no cycles: if it contained some cycle, then some edge, say e, cannot be in the s-t simple path of length k since the path is simple. Thus, e would have been removed by the second for

Algorithm 1 FindLongestPath1($G = (V, E)$)

```
$s_b \leftarrow 0$
$t_b \leftarrow 0$
$k_b \leftarrow -1$
for $s \in V$ do
  for $t \in V$ do
    for $k \in \{0, \dots, |V|\}$ do
      if LongestPath($G$, $s$, $t$, $k$) and $k > k_b$ then
        $s_b \leftarrow s$
        $t_b \leftarrow t$
        $k_b \leftarrow k$
      end if
    end for
  end for
end for
return $(s_b, t_b, k_b)$
```

Algorithm 2 FindLongestPath2($G = (V, E)$, s, t, k)

```
$G' = (V', E') \leftarrow G$
for $v \in V \setminus \{s, t\}$ do
  $G'' = G' \setminus \{v\}$
  if LongestPath($G''$, $s$, $t$, $k$) then
    $G' \leftarrow G''$
  end if
end for
for $e \in E'$ do
  $G'' = G' \setminus \{e\}$
  if LongestPath($G''$, $s$, $t$, $k$) then
    $G' \leftarrow G''$
  end if
end for
return $G$
```

loop. Thus, since G' is an acyclic graph over k vertices with a length k path from s to t, it is precisely an s-t path of length k, as required.

2. (20 marks) For each of the following statements about problems and class of problems discussed in lecture, specify whether the statement is "true" or "false"; **briefly justify your answer**. You may use any known results.

   (a) For all problems $X \in \mathcal{NP}$, we have $X \leq_P$ SUBSETSUM.

   True. We know that SUBSETSUM is NP-complete, and thus, by definition, every other problem in $\mathcal{NP}$ must be polynomially time reducible to it.

   (b) Given the current knowledge on the theory of NP-completeness, it is still conceivable that HAMCYCLE $\in P$ and INDEPENDENTSET $\notin P$.

   False. Since HAMCYCLE is in NPC, we have that HAMCYCLE $\in \mathcal{P}$ if and only if $\mathcal{P} = \mathcal{NP}$. Since INDEPENDENTSET is in $\mathcal{NPC}$, and $\mathcal{NPC} \subseteq \mathcal{NP}$, if $\mathcal{P} = \mathcal{NP}$, then INDEPENDENTSET must also be in $\mathcal{P}$,

   (c) If $\mathcal{P} = \mathcal{NP}$ then every problem in $\mathcal{NP}$ is NP-complete.

   True. If $\mathcal{P} = \mathcal{NP}$, then any problem $C \in \mathcal{NP}$ is polynomially time solvable, so every problem in $C' \in \mathcal{NP}$ can be solved by a polynomial time reduction to C (we can simply solve the instance of C' in the reduction), and thus $C \in \mathcal{NPC}$.

   (d) If LONGESTPATH $\in \mathcal{P}$ then INDEPENDENTSET $\in \mathcal{P}$.

   True. Since LONGESTPATH is NP-complete, if it is in $\mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$, so then INDEPENDENTSET is also in $\mathcal{P}$.

   (e) (Requires extra research) Suppose that P $\neq$ NP. Then for any problem $X \in \mathcal{NP}$ then, either $X \in P$ or X is NP-complete.

   False: In 1975, Ladner showed that if $\mathcal{P} \neq \mathcal{NP}$, then there exists a non-empty class of problems which he called $\mathcal{NP}$-intermediate that is disjoint from both $\mathcal{P}$ and $\mathcal{NP}$-complete. Thus, equivalently, $\mathcal{P} = \mathcal{NP}$ if and only if $\mathcal{NP}$-intermediate is empty. The graph isomorphism problem is a problem that may lie in $\mathcal{NP}$-intermediate.

3. (30 points) Textbook chapter 8, exercise 4:
Suppose you are consulting for a group that manages a high-performance real-time system in which asynchronous processes make use of shared resources. Thus the system has a set of n *processes* and a set of m *resources*. At any given point in time, each process specifies a set of resources that it requests to use. Each resource might be requested by many processes at once; but it can only be used by a single process at a time. Your job is to allocate resources to processes that request them. If a process has allocated all the resources it requests, then it is *active*; otherwise it is *blocked*. You want to perform the allocation problem so that as many processes as possible are active. Thus we phrase the RESOURCE RESERVATION PROBLEM as follows: Given a set of processes and resources, the set of requested resources for each process, and a number k, is it possible to allocate resources to processes so that at least k processes will be active?

Consider the following list of problems, for each problem either give a polynomial time algorithm or prove that the problem is NP-complete:

(a) The general RESOURCE RESERVATION PROBLEM defined above.

This problem is NP-complete. We can restate the problem as follows: we have a set of m resources and n processes, each of which requires a subset of the resources to execute. Then the problem to be solved is to decide if there is a set of k processes such that their requested resources are disjoint.

We show that the problem is in $\mathcal{NP}$: if we are given a set of k processes, we can easily check in polynomial time that no resource requested by one of the processes is needed by any other of the processes in time at most $O(k^2m)$: if we have processes $\{n_1, \ldots, n_k\}$, loop over all k^2 pairs of resources. For each pair, say n_i and n_j, loop over all m resources to make sure that no resource is required by both n_i and n_j. If there is no such resource across all combinations of pairs of processes, then the solution is valid; if not, clearly the solution is invalid.

We now show that the problem is NP-Hard by detailing a polynomial time reduction from INDEPENDENTSET. Assume we have an INDEPENDENTSET instance over graph G and integer k, i.e we want to determine whether or not G has an independent set of vertices of size at least k. We take the processes to correspond to the nodes of G, and the resources to correspond to the edges of G. Then a process (node) requires a resource (edge) if and only if the node is adjacent to the edge in G. It is easy to see that this reduction can be performed in polynomial time simply by looping over the vertices and edges. It remains to show that the problem we created is indeed equivalent to the original INDEPENDENTSET problem: that is, we show that we have a yes instance to one problem if and only if it is a yes instance to the other. If there are k processes whose required resources are disjoint, then the k nodes corresponding to those processes have no edges in common, so they form an independent set. If there is an independent set of size k, then those k nodes have no edges in common: thus, the processes corresponding to the nodes have no resources in common, so the corresponding k processes have disjoint resources as required.

(b) The special case of the problem when $k = 2$.

When $k = 2$, we can solve the problem via a polytime brute force algorithm. For each of the n^2 pairs of processes, loop over the m resources to see if they have any resources in common. If they do not, then the answer is yes, i.e. there exists a set of $k = 2$ processes with disjoint resources. If no such pair exists, then the answer is no, i.e. there is no set of $k = 2$ processes with disjoint resources.

(c) The special case of the problem when each resource is request by at most 2 processes.

Note that the reduction we detailed in part (a) is a special case of this form, since edges there correspond to resources requested by exactly two processes. Thus, since any IndependentSet problem can be expressed by a problem of this form, even this limited special case is still NP-complete.

4. (30 points) Textbook chapter 8, exercise 18: You've been asked to help some organizational theorists analyze data on group decision-making. In particular, they've been looking at a dataset that consists of decisions made by a particular governmental policy committee, and they are trying to decide whether it's possible to identify a small set of influential members of the committee.

Here is how the committee works. It has a set $M = \{m_1, \ldots, m_n\}$ of n members, and over the past year it has voted on t different issues. On each issue, each member can vote either "yes", "no", or "abstain"; the overall effect is that the committee presents an affirmative decision on the issue if the number of "yes" votes is strictly greater that the number of "no" votes (the "abstain" votes don't count for either side), and it delivers a negative decision otherwise.

Now we have a big table consisting of the vote cast by each committee member on each issue, and we'd like to consider the following definition. We say that a subset of members $M' \subseteq M$ is *decisive* if, had we looked just at the votes cast by the members of M', the committee's decision on *every* issue would have been the same. (In other words, the overall outcome of the voting among the members of M' is the same on every issue as the outcome of the voting by the entire committee.) Such a subset can be viewed as a kind of "inner circle" that reflects the behaviour of the committee as a whole.

Here is the question: Given the votes cast by each member on each issue, and given a parameter k, we want to know whether there is a decisive subset consisting of at most k members. We'll call this an instance of the Decisive Subset Problem.

Example: Suppose we have four committee members and three issues. We're looking for a decisive set of size at most $k = 2$, and the voting went as follows:

Issue#	m_1	m_2	m_3	m_4
Issue 1	yes	yes	abstain	no
Issue 2	abstain	no	no	abstain
Issue 3	yes	abstain	yes	yes

Then the answer to this instance is "yes", since members m_1 and m_3 constitute a decisive subset.

Prove that Decisive Subset is NP-complete.

Hint: Remember to justify why the problem is in $\mathcal{NP}$, and then prove that an NP-complete problem X of your choice polynomial-time reduces to Decisive Subset *($X \leq_P$* Decisive Subset*).*

We first show that DecisiveSubset is in $\mathcal{NP}$, i.e. we can verify a certificate is a solution to DecisiveSubset in polynomial time. Given a set of committee members, we can easily check that the size is k, and by iterating over all n members and the k members in the subet for each issue, we can check whether or not the votes of the k members on the issue represent the same votes of all n members. If this is the case for all issues, then the certificate is a valid solution, and if not, then it is not a valid solution. Thus, this can be verified in time $O((n+k)m)$.

We now show that there is an NP-complete problem with a polynomial time reduction to DecisiveSubset. We choose VertexCover: let $G = (V, E)$ be a graph, and let k be a bound such that we want to know if G contains a vertex cover of size at most k. For each edge $e \in E$, create an issue I_e, and for each node $v \in V$, create a committee member m_v. If $e = (u, v)$, then we have that members m_u and m_v vote yes on issue I_e, and all other committee members abstain on issue I_e. Note then that the vote of the entire committee on issue I_e is thus yes. We then ask if there is a decisive set of size at most k on this collection of issues and voting members.

We want to show that G has a vertex cover of size at most k if and only if there exists a decisive subset of at most k members on the instance generated according to the reduction given above. Assume there is a vertex cover of size at most k. Then, for each edge $e = \{u, v\}$, at least one of u or v must be in the vertex cover. For the set of members corresponding to the nodes in the vertex cover, the voting outcome for each issue I_e will then be yes, so the corresponding set of members is indeed decisive as required. Now, assume there is indeed a decisive set of size at most k. Then, for each issue, we must have that the decisive set voted affirmatively on the issue. Thus, we must have at least one of the two members m_u or m_v voting yes on the issue I_e in our decisive set. This corresponds to having at least one of the two endpoints - either u or v - of the edge $e = \{u, v\}$ in the vertex cover, and since this is the case across all issues, or edges, the decisive set indeed represents a vertex cover.