Algorithms
Computer Science 140 & Mathematics 168
Spring 2012
Homework 11b
Due Tuesday, April 10

Note 1: On this assignment there are several places where you will need to prove that a problem is NP-complete. On the last page of this assignment is a list of problems that we have shown to be NP-complete. You can use any of these in your reductions.

Note 2: Takehome Exam 2 will be given out on Tuesday, April 17 and due back on Thursday, April 19. The exam will cover graph algorithms, NP-completeness, and approximation algorithms. Graph algorithms, of course, rely on data structures, so you should feel comfortable with the data structure ideas that we explored as well. The exam will again be a 3 hour and 30 minute exam (15 minutes to read over the exam, 3 hours to take it, and a 15 minute intermission), open notes, and closed "everything else."

1. **[30 Points] Network Reliability is NP-Complete!** Millisoft has decided to purchase the entire Internet and rent links to Internet Service Providers (ISPs). We represent the internet as an undirected graph. Assume that each edge in the graph has a positive integer *rental cost* associated with it. The ISPs are confronted with the challenging problem of spending the minimum amount of money on link rental so that they can provide adequately reliable service to their customers.

   Here is how we quantify reliability: We say that two paths in the network are *disjoint* if they have no vertices in common, except for the endpoints. For example, there can be two disjoint paths from vertex v_{42} to v_{100}, but v_{42} and v_{100} can be the only vertices that these paths have in common (since these vertices are the endpoints of the paths). In the interest of reliability, it is desirable to have multiple disjoint paths between pairs of nodes in the network. Some ISPs provide more reliability than others and they can charge their clients more accordingly.

   The Network Reliability Optimization Problem (NROP) is now defined as follows: Given is an undirected graph with n vertices $v_1, \ldots, v_n$, an $n \times n$ symmetric matrix R_{ij} of positive integers and a positive integer cost associated with each edge. The objective is to find a subset S of the edges such that the total cost of the edges in S is minimized *and* for every pair of vertices v_i and v_j there exist at least R_{ij} disjoint paths from v_i to v_j such that all paths use only edges in S.

   Your boss, Gill Bates, has asked you and your team members to develop an efficient algorithm for solving NROP optimally for the client ISPs. After working on an algorithm for awhile, your team conjectures that the problem is NP-complete. State the decision version of this problem, which we will call NRDP, and prove that NRDP is NP-complete using a reduction from a problem that we have already shown to be NP-complete.

If you are careful, the entire write-up of this problem can be done rigorously in a total of one page or less.

2. **[20 Points] Bin Packing Revisited!**

Recall that last week you showed that Bin Packing Problem is NP-complete. That's a bummer, because - as was noted, the problem comes up in a wide variety of practical applications from operating systems to container shipping. For example, an operating system typically stores data in chunks called "pages", each with some fixed size C. If the page size is 4096 bytes then whether you wish to store 1 byte or 4096 bytes, you will need to reserve an entire 4096 byte page for your data. If you have many pieces of data to save, the question becomes what is the least number of pages required to store that data. That's a bin packing problem where the pages are bins. Similarly, container ships require that all goods be packed in containers of some fixed size C and the shipping companies wish to determine the smallest number of containers (bins) required for a given set of items.

So, recall that the Bin Packing decision problem is formulated as follows: We are given a collection of objects $S = \{a_1, \ldots, a_n\}$, a function $s : S \to \mathbb{Z}_+$ indicating the positive integer "size" of each object, a bin capacity C such that C is greater than or equal to the size of the largest object, and a target t. The question is whether or not it is possible to pack all of the objects in S into t or fewer bins, each of capacity C.

The corresponding optimization problem gets rid of the target t and asks us to pack the objects into the least number of bins. In this problem, we will consider an approximation algorithm called "First Fit" which works as follows: Initially, allocate just a single bin and number it 1. Consider the objects one-by-one in no particular special order (just whatever order the items are given to us). Place the object in the lowest number bin in which it fits. If the object doesn't fit in any of the bins allocated so far, allocate a new bin and give it a counting number that is the next available number.

So, initially there is just one bin. We place items in that bin until we get to an item that doesn't fit in the bin's remaining capacity. We now allocate a new empty bin 2 and place our item there. Now, for each subsequent item, if it fits in bin 1 put it there. If not, try bin 2. If it doesn't fit there either, allocate a new bin numbered 3 and place the object there, etc.

(a) Give a small example that shows that the First Fit algorithm can use more than the optimal number of bins.

(b) Here's a little technical factoid that will be useful to you: For any positive number x, $\lceil 2x \rceil \leq 2\lceil x \rceil$. (Recall that the "ceiling" $\lceil y \rceil$ is the least integer greater than or equal to y. In other words, this is rounding up to the nearest integer.) Prove this little factoid.

(c) Now prove that First Fit uses at most twice the optimal number of bins. In other words, show that First Fit is an approximation algorithm with ratio at most 2.

Your proof should be precise and rigorous. It's likely that you'll need to appeal to the little factoid that you just proved. (Remember, the idea in these analyses is to find a "proxy" that can be used to lower-bound the optimal number of bins and then relate the number of bins that First Fit uses to this proxy as well. In this way, we'll be able to relate First Fit to the optimal solution via the proxy! That's the key to these "a-proxy-mation" analyses!)

(d) Explain briefly why First Fit runs in polynomial time. Thus, you can conclude that First Fit is a polynomial time 2-approximation algorithm!

3. **[20 Points] The Disk Storage Problem!** First the gratuitious story. You have been hired by greenhat to work on their popular Lunatix operating system. When Lunatix does a backup, it typically uses two large backup disks and a tape drive. Since disks have lower access time than tape, it is desirable to store as many files as possible on disk and the remainder will go on tape. Let $F = \{f_1, \ldots, f_n\}$ denote the n files to be backed up and let ℓ_i denote the length of file i. Without loss of generality, let $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n$.

   There are two identical disks, each with storage capacity of L. A file stored on disk cannot be broken up - it must be stored entirely on one of the two disks. The Disk Storage Optimization Problem is to store the maximum number of files from F onto the disks.

   Your boss has suggested that you implement a greedy algorithm for the Disk Storage Optimization Problem: Since the files in F appear in non-decreasing order of size, simply go through F from shortest file to longest file, putting as many of the files on the first disk as possible. When the first disk fills up, continue iterating through F putting as many of the remaining files as possible on the second disk.

   (a) Give an example that demonstrates that the greedy algorithm does not necessarily find an optimal solution. That is, give a set of specific file sizes, show the solution found by the greedy algorithm, and then the optimal solution. Show that the solution found by the greedy algorithm is not optimal.

   (b) State the decision problem corresponding to this optimization problem. Then prove that the decision problem is NP-complete. Use a reduction from an NP-complete problem that we have seen in class or on a previous homework assignment.

   (c) Prove that the greedy algorithm is an approximation algorithm which finds solutions that are at most 1 smaller than optimal. This is really neat, since all of the approximation algorithms we've seen so far were some *multiplicative* factor worse than optimal (typically 2 times optimal). Here, the algorithm finds solutions which are an *additive* amount worse than optimal! (Please use a "proxy" argument here.)

4. **[20 Point OPTIONAL Bonus Problem!] Multicommodity Network Flow!**

We know how to solve the network flow problem in polynomial time. A generalization of network flow is called *multicommodity network flow* and it goes like this: We are given a flow network except now there are multiple source/sink pairs in the network. That is, there are some number k of sources $s_1, \ldots, s_k$ and the same number of sinks $t_1, \ldots, t_k$. Each source s_i pushes a different kind of "fluid" or "commodity" to its corresponding sink t_i. However, the edges of the network are shared by these flows and each edge still has a single integer capacity. For example, an edge with capacity 5 might accommodate 2 units of flow from s_1 to t_1 and 3 units of flow from s_2 to t_2. Now we are interested in finding a set of flows, one for each source/destination pair. Of course, we are still subject to capacity constraints and conservation of flow (i.e. for any given commodity, the amount of flow for that commodity entering a given vertex is equal to the amount of flow for that commodity leaving that vertex).

It turns out that maximizing the total amount of flow in a multicommodity network can still be solved in polynomial time by using polynomial time linear programming algorithms! However, the optimal solution found by the linear program will, in general, not have integer flows.

If we now simply restrict the flows for each commodity to have integer values, the problem becomes NP-complete! Amazing, but true! In particular, the decision problem that we will consider here is this: Given is a flow network with k commodities and k source/sink pairs, (s_i, t_i) (one for each commodity), and a demand d_i for each sink t_i. The question is: "Does there exist a valid set of flows, one for each commodity, such that each sink receives d_i of commodity i?"

Prove that this problem is NP-complete. A reduction from 3SAT is particularly nice!

A List of Some Known NP-Complete Problems

1. 3SAT: Given an expression in conjunctive normal form with 3 literals per clause, is it satisfiable?
2. Vertex Cover: Given a graph G and number k, is there a vertex cover of size k or less?
3. Dominating Set: Given a graph G and a number k, is there a dominating set of size k or less.
4. Independent Set: Given a graph G and number k, is there an independent set of size k or more?
5. Clique: Given a graph G and a number k, is there a clique of size k or more?
6. Graph Coloring: Given a graph G and a number k, is it possible to color the vertices with k or fewer colors so that no two adjacent vertices have the same color?
7. Directed Hamiltonian Path From s to t: Given a directed graph G and vertices s and t, is there a directed Hamiltonian path from s to t?
8. Directed Hamiltonian Path: Given a directed graph G, is there a directed Hamiltonian path in the graph.
9. Directed Hamiltonian Cycle: Given a directed graph G, is there a directed Hamiltonian cycle in the graph?
10. Undirected Hamiltonian Path From s to t: Given an undirected graph G and vertices s and t, is there an undirected Hamiltonian path with endpoints s and t.
11. Undirected Hamiltonian Path: Given an undirected graph G, is there an undirected Hamiltonian path in the graph.
12. Undirected Hamiltonian Cycle: Given an undirected graph G, is there an undirected Hamiltonian cycle in the graph?
13. Traveling Salesman Problem: Given a completely connected weighted undirected graph and a number k, does there exist a traveling salesman tour of cost k or less?
14. Subset Sum: Given a set S of positive integers and a target integer k, does there exist a subset of S with sum equal to k?
15. Partition: Given a set S of positive integers, can the integers be partitioned into two sets S_1 and S_2 such that the sum of the elements in S_1 is equal to the sum of the elements in S_2?
16. Bin Packing: Given a multiset S of positive integers "objects", a bin capacity C such that C is greater than or equal to the size of the largest object, and a target t, is it possible to pack all of the objects in S into t or fewer bins, each of capacity C?