CS 170 Homework 10 (Optional)

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Dijsktra's Sort

Show how to use Dijsktra's algorithm to sort n real numbers (not necessarily non-negative or integral) in ascending order in $O(n \log n)$ time. You may use the intermediate outputs of Dijsktra's to do the sorting (as opposed to using it as a blackbox).

Argue that this means that improving upon the $O(m + n \log n)$ run-time of Dijsktra's algorithm (with Fibonacci heap) would lead to a faster sorting algorithm than merge and quick sort.

Hint: Given the numbers, construct a graph such that the order of vertices being extracted from the priority queue by Dijsktra's algorithm corresponds to the right sorting order.

3 Reduction Warm-ups

Let us define the following problems below:

> **Undirected Hamiltonian Path:** given an undirected graph G as input, determine if there exists a path in G that uses every vertex exactly once.
>
> **Longest Simple Path:** given a directed (not necessarily acyclic) graph G as input, determine if there exists a path in the graph that is of length k or more.
>
> **Longest Path in a DAG:** given a DAG and a variable k as input, determine if there exists a path in the DAG that is of length k or more.

Both the Undirected Hamiltonian Path problem and Longest Simple Path problem are known to be NP-complete. However, recall that the Longest Path in a DAG problem is in P.

Consider the following two incorrect proofs that the longest path in a DAG problem is NP-hard. For each proof, explain why the proof is incorrect.

(a) In order to show that Longest Path in a DAG is NP-hard, we will reduce Longest Simple Path to Longest Path in a DAG.

Consider a DAG G on which we wish to determine whether there exists a path of length at least k. We construct an instance of Longest Simple Path as follows: use the same graph G and ask whether there is a simple path of length at least k in G.

Since G is a DAG, any path in G is automatically simple (because cycles are not possible). Therefore, the answer to the Longest Simple Path instance on G is yes if and only if the answer to the Longest Path in DAG instance is yes. Therefore, Longest Path in a DAG is NP-hard.

(b) In order to show that the longest path in a DAG is NP-hard, we will reduce Undirected Hamiltonian Path to Longest Path in a DAG.

Consider an undirected graph G on which we wish to see if a Hamiltonian Path exists. We will convert it to an instance of Longest Path in a DAG as follows. Use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed, and back edges will be omitted, giving us a DAG.

This new directed graph will be our instance for Longest Path in a DAG – we need to check if its longest path has at least $|V| - 1$ edges. If so, there must be a Hamiltonian path in G, as any simple path with at least $|V| - 1$ edges must visit every vertex.

4 Upper Bounds on Algorithms for NP Problems

(a) Recall the 3-SAT problem: we have n variables x_i and m clauses, where each clause is the OR of at most three literals (a literal is a variable or its negation). Our goal is to find an assignment of variables that satisfies all the clauses, or report that none exists.

Give a $O(2^n m)$-time algorithm for 3-SAT. Just the algorithm description is needed.

(b) Using part (a) and the fact that 3-SAT is NP-hard, give an $O(2^{n^c})$-time algorithm for every problem in NP, where c is a constant (that can depend on the problem). Just the algorithm description and runtime analysis are needed.

Hint: Since 3-SAT is NP-hard, you can use it to solve any problem in NP!

Note: This result is known as $\mathsf{NP} \subseteq \mathsf{EXP}$.

(c) Recall the halting problem from CS70: Given a program (e.g., a `.py` file), determine if the program runs forever or eventually halts. Also, recall that there is no finite-time algorithm for the halting problem. Let us define the input size for the halting problem to be the number of characters used to write the program.

Let's now consider a "search" variant of the halting problem: given a program, we not only determine if the program runs forever, but if it halts, we also return its output.

Given an instance of 3-SAT with n variables and m clauses, we can write a size $O(n+m)$ program that outputs a valid assignment if the instance is satisfiable and runs forever otherwise. So there is a polynomial-time reduction from 3-SAT to the search variant of the halting problem.

Based on this reduction and part (b): Is the search variant of the halting problem NP-hard? Is it NP-complete? Justify your answer.

5 Some Sums

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $S \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in S} a_i = \sum_{j \in ([n] \setminus S)} a_j$. In other words, determine whether there is a way to partition A into two disjoint subsets such that the sum of the elements in each subset is equal.

2 **Subset Sum**: Given some integer x, determine whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = x$. In other words, determine whether there is a subset of A such that the sum of its elements is x.

3 **Knapsack**: Given some set of items each with weight w_i and value v_i, and fixed numbers W and V, determine whether there is some subset $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i \geq V$.

For each of the following, clearly describe your reduction and justify its correctness.

(a) Find a linear time reduction from Subset Sum to Partition.

(b) Find a linear time reduction from Subset Sum to Knapsack.

6 k-XOR

In the k-XOR problem, we are given n boolean variables $x_1, x_2, \ldots, x_n$, a threshold integer $t > 0$, and a list of m clauses each of which is the XOR of exactly k distinct literals (that is, the clause is true if and only if an odd number of the k literals in the clause are true). Some examples of k-XOR clauses for $k = 3$ are $(x_1 \oplus x_2 \oplus x_3)$ and $(\neg x_1 \oplus x_4 \oplus \neg x_5)$. Our goal is to decide if there is some assignment of variables that satisfies at least t clauses.

(a) In the Max-Cut problem, we are given an undirected unweighted graph $G = (V, E)$ and an integer α and want to find a cut $S \subseteq V$ such that at least α edges cross this cut (i.e., have exactly one endpoint in S). Give and argue the correctness of a reduction from Max-Cut to 2-XOR.

Hint: every clause in 2-XOR is equivalent to an edge in Max-Cut.

(b) Give and argue the correctness of a reduction from 3-XOR to 4-XOR.

