Final Exam

CIT 596 – Algorithms

1. True or False? For each answer provide a short justification.

   (a) If G is a weighted connected graph, and C is any cycle in G, then any minimum spanning tree of G includes all but one of the edges of C.

   Answer: False. Consider the complete graph on 4 vertices, 1,2,3,4 where edges from 1 to each other vertex have weight 1 and edges between any other 2 vertices have weight 2. Then no edge from the cycle (2,3,4) will be in the MST.

   (b) Let A and B be decision problems in NP, defined on the same set of inputs. Let C be another decision problem defined as follows: x is a YES-instance of C if and only if it is both a YES-instance of A and a YES-instance of B. Then C is also in NP.

   Answer: True. A certificate for a YES-instance x of C is a concatenation of the certificates showing that x is a YES-instance of A and a certificate showing that x is a YES-instance of B. The verification algorithm will just verify each of these certificates using the verification algorithms for A and B.

   (c) If we use a balanced search tree to maintain a dictionary S, we can find the minimum element in S in $O(1)$ time.

   Answer: False. In general you will need $\Omega(\log n)$ steps to find the minimum element.

   (d) In a directed graph if e is an edge on the shortest path from x to y, and it is also on the shortest path from u to v, then it is on the shortest path from x to v.

   Answer: False. Consider a graph where there is a direct edge from x to v. If you want to think about weighted graphs, make this the edge of lowest weight.

2. The 0-1 Integer Programming Problem (0-1-IP) is described below.

   Instance: A set of inequality and equality constraints on a set of variables $\{x_1, x_2, \ldots, x_n\}$: The instance consists of some number of such constraints. All the coefficients and right-hand sides in these constraints are integers (positive or negative).

   Question: Is there a way to assign each variable a value of 0 or 1 to satisfy all the given constraints?

   For example, one instance of the problem is

$$\begin{aligned} x_1 + x_2 + x_7 - x_3 &\geq 0 \\ 2x_3 - x_2 + x_5 &\geq 1 \\ x_2 - x_4 + x_6 - x_8 &= 0 \\ 3x_1 + x_5 - x_4 = 1 & \end{aligned}$$

   This particular instance is a YES-instance because the setting where $x_5 = 1$ and all the other variables are 0 satisfies all the constraints.

   Using the fact that 3-SAT is NP-complete, prove that 0-1-IP is NP-complete.

   Answer:

   In NP: Given a Yes-instance of 0-1-IP a certificate would be a setting of the variables that satisfies all the constraints. Given these values for the variables you can verify in polynomial time that it satisfies each of the constraints.

Reduction: We reduce from 3-SAT as suggested. Given a 3-SAT formula $\Phi(x_1, x_2, \dots, x_n)$ with m clauses, we will create a 0-1-IP instance with $2n$ variables, one corresponding to each literal in the 3-SAT instance. Specifically, we will let y_i be the variable in the 0-1-IP instance corresponding to x_i in the 3-SAT instance, and we will let z_i be the variable in the 0-1-IP instance corresponding to $\bar{x}_i$.

To ensure that exactly one of x_i and $\bar{x}_i$ gets set to 1, we add the constraint $y_i + z_i = 1$ for $i = 1, 2, \dots n$. To ensure that some literal in each clause is set to 1, we add the constraint stating that the sum of the variables corresponding to the 3 literals in a clause is greater than or equal to 1. We write m such constraints, one for each clause. (If a literal in a clause is an uncomplemented variable x_i, we use the variable y_i in the constraint in 0-1-IP, and if a literal is a complemented variable $\bar{x}_i$, we use the variable z_i in the constraint.)

Clearly this reduction can be accomplished in polynomial time. We are producing n equality constraints and m inequality constraints, which can each be written down in $O(1)$ time.

3. You are given a connected, weighted graph $G = (V, E)$ where all edge weights are non-negative. Your goal is to remove a subset S of minimum total weight so that the remaining graph $G' = (V, E - S)$ does not have any cycles. Design an efficient algorithm for doing this. Argue that your algorithm is correct and state its running time.

   Answer: Since the graph G' does not have any cycles, the edges in this graph must form a spanning tree or a subset of a spanning tree. Since every edge you remove adds to the weight of the set of edges being removed, we should stop removing edges once we get to a spanning tree. Of course, we want to leave a spanning tree with the greatest total weight of edges possible, since these are the edges we are not removing. Thus, we want to find a maximum spanning tree using a small variant of Kruskal's algorithm say, and remove all the edges not in this tree. The running time is $O(m \log n)$ from the analysis of Kruskal's algorithm we did.

4. You are given a sorted array A of n **distinct integers** and you want to determine if there is an index i such that $A[i] = i$. Design an $O(\log n)$ time algorithm for this problem and prove it correct.

   Answer: This algorithm follows the pattern of binary search in choosing which index of A to probe. It relies on the following fact. Suppose we probe index j. If $A[j] < j$, then for all indices k less than j, $A[k] < k$. This follows from the fact that A is sorted, it contains integers, and these integers are distinct. Thus, for example $A[j-1] < A[j] < j$, and these inequalities are strict, implying that $A[j] \le j - 1$ and $A[j-1] \le j - 2$. Extending this reasoning inductively we can prove that for all k as above, $A[k] < k$.

   By a symmetric argument, if $A[j] > j$, then for all indices $k > j$, we have that $A[k] > k$.

   Thus, the result of probing $A[j]$ eliminates one half of the array, either the indices less than j or the indices greater than j. Using this we can use binary search to find an index j (if it exists) such that $A[j] = j$.

   The running time is $O(\log n)$ from the standard analysis of binary search.

5. Let $G = (V_1 \cup V_2, E)$ be a bipartite graph with $|V_1| = |V_2| =$ n. Recall that Hall's Theorem shows that G has a perfect matching if and only if for every subset $S \subseteq V_1$, $|\Gamma(S)| \ge |S|$. Here $\Gamma(S)$ is the set of neighbors of S.

   Let G be a bipartite graph as above with $|V_1| = |V_2|$. Define the **Hall-deficit** $H(S)$ of a set $S \subseteq V_1$ to be 0 if $|\Gamma(S)| \ge |S|$ and $|S| - |\Gamma(S)|$ otherwise. Using max flows and min cuts, design an efficient algorithm to find a set $S \subseteq V_1$ of maximum Hall-deficit in G. Prove your algorithm correct.

Answer:

The algorithm is pretty straightforward. Set up the standard flow network as we do for bipartite matching. Then find the max flow and then the min-$s - t$cut (A, B) with $s \in A$ and $t \in B$. Claim: The set $S = A \cap V_1 isasetofmaximumHall - deficit.$

Proof: Let us take one finite-capacity cut where s is on one side and all the other vertices are on the other side as the 'reference' cut. This cut has capacity n. For any other cut, we will see how much we 'save' on capacity compared to this reference cut. Recall that in any finite-capacity cut (C, D) if $T = C \cap V_1$, then $\Gamma(T) \subseteq C$. So, by putting T in the source-side, we save $|T|$ in capacity, with respect to the reference cut. But we pay at least $|\Gamma(T)|$ more than the reference cut, because of edges from $\Gamma(T)$ to t. In fact, to minimize what we pay, we should put exactly $\Gamma(T)$ in C. The capacity of this cut would be $n - |T| + |\Gamma(T)|$, which is exactly n minus the Hall-deficit of T. Thus to get a minimum capacity cut, we should choose T to be a set of maximum Hall-deficit. in other words, the set $T \subseteq V_1$ on the source-side of a min-capacity cut is a set of maximum Hall-deficit.

The running time of the algorithm is just the usual running time to solve the flow problem. The max-flow possible in this network is n, and so the basic Ford-Fulkerson method will terminate in $O(n)$ augmentations, each of which takes $O(m)$ time, for an overall time of $O(mn)$.

6. Weighted directed acyclic graphs are important tools in project management. Suppose you are given such a DAG $G = (V, E)$ where the weight of an edge e is $w(e)$. There are also two special nodes — a start node s and a finish node t. The length $\ell(P)$ of a path P is defined in the usual way: $\ell(P) = \sum_{e \in P} w(e)$. A *critical path* in G is a path from s to t of greatest total length. (In project management, the weights on the edges may represent the time to complete the task on the edge, and critical paths give a lower bound on how long a project will take.)

   (a) Design an efficient algorithm to find the length of a critical path in G. Analyze the running time of your algorithm. You do not need to prove it correct.

   Answer:

   One solution is to do a topological sort of G. For each vertex v, let $\ell(v)$ denote the length of the longest path starting at v. We can initialize $\ell(t)$ to 0. Then in reverse topological-sort-order we consider each vertex v and update the longest path from v as $\max_{u:(v,u) \in E} (\ell(u) + w(v, u))$.

   Top-sort takes $O(m + n)$. Then in the computation of $\ell(v)$ for all vertices, we go through each edge once. $\ell(s)$ is the required answer. So this step is also $O(m + n), whichistheoveralltimebound.$

(b) For an edge e, define the *slack*, $s(e)$, on edge e to be the amount by which the weight of e can be increased without increasing the length of the critical path in G. Design an efficient algorithm that computes $s(e)$ for a given edge e. Argue that your algorithm is correct. For this part there is no need to analyze the running time.

Answer:

Let $e = (u, v)$. Then, first compute $\ell(v) asabove. Nextinthereversedgraph, computethelongestpathfromutosandcallthis \ell'($
Then the slack of $(u, v) = \ell(s) - \ell(v) - \ell'(u) - w(u, v)$, which can be computed in $O(m + n)$. To compute the slack of the edge, we are computing the length of the longest path from s to t that passes through that edge and then seeing how much less this is than the overall longest path.