CSE 202: Design and Analysis of Algorithms Winter 2013

Problem Set 1

Instructor: Kamalika Chaudhuri **Due on:** Tue. Jan 22, 2012

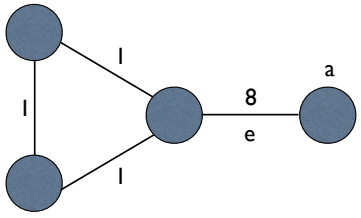
Problem 1

The following statements may or may not be correct. For each statement, if it is correct, provide a short proof of correctness. If it is incorrect, provide a counter-example to show that it is incorrect. In the following, assume that the graph $G = (V, E)$ is undirected and connected. Do not assume that all the edge weights are unique unless stated.

1. If G has more than $|V| - 1$ edges, and there is a unique heaviest edge e, then e cannot be part of any minimum spanning tree of G.
2. The worst case time for a find operation in the Union-Find data structure can be $\Theta(\log n)$ even with path compression.
3. A lightest edge in a cut is present in all minimum spanning trees of G.
4. If e is a heaviest edge in a cycle, then e cannot occur in any minimum spanning tree of G.
5. Prim's algorithm works correctly when some of the edge weights are negative.

Solution

1. False. In the following weighted graph, e is the unique heaviest edge. But e has to belong to any MST because it is the unique edge in the $(a, V - a)$ cut.



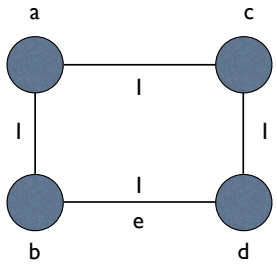
2. True. Let n be a power of 2 and let $x_1, \ldots, x_n$ be the variables. Consider the following sequence of unions. There are $\log_2 n$ levels of union operations. In the first level, we do the following $n/2$ unions: $(x_1, x_2), (x_3, x_4), \ldots, (x_{n-1}, x_n)$. Thus, after this level, there are $n/2$ disjoint sets.

   Let $T_1, \ldots, T_m$ be the disjoint sets at the beginning of level k. We then do the following unions: $(\mathbf{root}(T_1), \mathbf{root}(T_2)), (\mathbf{root}(T_3), \mathbf{root}(T_4)), \ldots)$. As the number of disjoint sets decreases by a factor of 2, and as there are $n/2$ sets after level 1, after level k, there are $n/2^k$ sets.

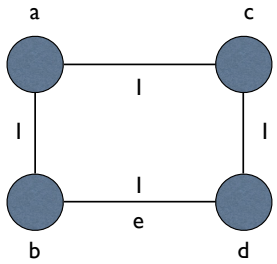
This indicates that after level $\log_2 n - 1$, there is one disjoint set T. Moreover, by the induction argument shown in the proof of Property 2 in class, the height of this set T is $\log_2 n - 1$. (As we union the roots of the trees, the find step inside the union does not need to do a path compression.)

Now, after this sequence of unions, if we execute a find on x, where x is a node with height $\log_2 n - 1$ in T, then this find operation will take $\Theta(\log n)$ time. Thus the worst case time for a find operation can be $\Theta(\log n)$.

3. False. In the following weighted graph, e is a lightest edge in the $(\{a, b\}, \{c, d\})$ cut. But $T = \{(a, b), (a, c), (c, d)\}$ is a MST which does not contain e.



4. False. In the following weighted graph, e is a heaviest edge in the cycle $a - b - d - c$, but it occurs in the MST $T = \{(a, b), (b, d), (c, d)\}$.



5. True. The heap works correctly with negative weights, and all the invariants in the analysis of Prim's algorithm hold when the edge weights are negative.

Problem 2

Given a list of n positive integers $d_1, d_2, \ldots, d_n$, we want to efficiently determine whether there exists an undirected graph $G = (V, E)$ whose nodes have degrees precisely $d_1, \ldots, d_n$. That is, if $V = \{v_1, \ldots, v_n\}$, then the degree of v_i should be exactly d_i. We call $(d_1, \ldots, d_n)$ the degree sequence of G. This graph G should not contain self-loops (edges with both endpoints equal to the same node) or multiple edges between the same pair of nodes.

1. Give an example of d_1, d_2, d_3, d_4 where all the $d_i \leq 3$ and $d_1 + d_2 + d_3 + d_4$ is even, but for which no graph with degree sequence (d_1, d_2, d_3, d_4) exists.

2. Suppose that $d_1 \geq d_2 \geq \ldots \geq d_n$ and that there exists a graph $G = (V, E)$ with degree sequence $(d_1, \ldots, d_n)$. We want to show that there must exist a graph that has this degree sequence and where in addition the neighbors of v_1 are $v_2, v_3, \ldots, v_{d_1+1}$. The idea is to gradually transform G into a graph with the desired additional property.

   (a) Suppose the neighbors of v_1 in G are not $v_2, v_3, \ldots, v_{d1+1}$. Show that there exists $i < j \leq n$ and $u \in V$ such that $(v_1, v_i), (u, v_j) \notin E$ and $(v_1, v_j), (u, v_i) \in E$.

   (b) Specify the changes you would make to G to obtain a new graph $G_0 = (V, E_0)$ same degree sequence as G and where $(v_1, v_i) \in E$.

   (c) Now show that there must be a graph with the given degree sequence but in which v_1 has neighbors $v_2, v_3, \ldots, v_{d_1+1}$.

3. Using the result from part (2), describe an algorithm that on input $d_1, \ldots, d_n$ (not necessarily sorted) decides whether there exists a graph with this degree sequence. Your algorithm should run in time polynomial in n and in $m = \sum_{i=1}^{n} d_i$.

Solution

1. Let $(d_1, d_2, d_3, d_4) = (3, 3, 1, 1)$. Their sum is 8, all $d_i \leq 3$, but no graph exists with this degree sequence.

2. Let $G = (V, E)$ with degree sequence $(d_1, \ldots, d_n)$ as given in the problem.

   (a) It is given that v_1 has d_1 neighbors. Because they are not $v_2, v_3, \ldots, v_{d_1+1}$, then $(v_1, v_j) \in E$ for some j with $d_1 + 1 < j \leq n$ and $(v_1, v_i) \notin E$ for some i with $1 < i \leq d_1 + 1$. (This implies that $i < j \leq n$.) By defintion, v_j has d_j neighbors, and v_i has d_i neighbors. Since we know v_j is a neighbor of v, $d_j > 0$. The ordering of the degrees implies $d_i \geq d_j$ since $i < j$. The neighbors of v_j include v_1 and the neighbors of v_i do not, so $d_i \geq d_j$ implies that there must be some vertex $u(\neq v_j)$ that is a neighbor of v_i and not v_j. In other words, $(u, v_i) \in E$ and $(u, v_j) \notin E$.

   (b) Find a v_i, v_j, and u as in the previous part. These vertices are guaranteed to exist as proven in the previous part. Let $E_0 = E \cup \{(v_1, v_i), (u, v_j)\} \setminus \{(v_1, v_j), (u, v_i)\}$. Note that for vertices $v_1, v_i, v_j, u \in V$, the degrees remain unchanged, so $G_0 = (V, E_0)$ has the same degree sequence and (v_1, v_i) is an edge.

   (c) We can simply iterate the above procedure until the neighbors of v_1 are $v_2, \ldots, v_{d_1+1}$. Note that this process will always terminate because d_1 is finite.

3. Let $d_1, \ldots, d_n$ be the desired degree sequence. To decide whether there exists a graph with this degree sequence, use the following recursive procedure given a sequence $d_1, \ldots, d_n$: If the sequence has length 1, output YES if $d_1 = 0$ and NO otherwise. If not, first sort the sequence into $d'_1 \geq \ldots \geq d'_n$. If $d'_1 < 0$, or if $d'_1 \geq n$, then return NO. Otherwise, remove d'_1 from the sequence, subtract 1 from $(d'_2, \ldots, d'_{d'_i+1})$, do appropriate operations as explained in implementation so that the new sequence is still sorted and continue.

   Implementation: Store the sequence $d_1 \geq d_2 .... \geq d_n$ as a sorted linked list D. Separately store two other sorted linked lists – $First$ and $Last$. $First(i)$ and $Last(i)$ represent the first and last occurence of degree i in the linked list D; these two lists are maintained in decreasing order of degree.

   In each recursive step, the head of the linked list D will be deleted and the first few elements in the list will be decremented by 1. There are two cases.

In the first case, this operation will not disrupt the sorted order of the linked list D. If the head had degree d_{max} before the operation, updating D as well as the $First$ and $Last$ lists will take $O(d_{max})$ time, and thus the total update operation will take $O(d_{max})$ time.

In the second case, the sorted order of the linked list D will be disrupted. However, we can take advantage of the fact that the first few elements of D are decremented by exactly 1. Suppose after the decrement operation, $d_j, \ldots, d_i$ becomes smaller than d_{i+1}. The observation is that they become smaller by exactly 1; so we can use the $Last$ array to determine the last node with degree d_{i+1} in D and insert $d_j, \ldots, d_i$ after this last node. This insertion will take constant time. We'll need to update $First$ and $Last$, which again takes $O(d_{max})$ time, where the head had degree d_{max} before the decrement.

Correctness: The proof of correctness is by induction on n. Suppose $n = 1$; there are no other nodes to connect to, so d_1 must be 0 to make a graph. Now, suppose that the algorithm is correct for $n = k$, and we will show that it is also correct for $n = k + 1$. Let $d_1, \ldots, d_{k+1}$ be the desired degree sequence and $d'_1 \geq \ldots \geq d'_{k+1}$ be the sequence sorted in nondecreasing order. The previous part showed that if a graph exists with this sequence, then there is a graph with this sequence that also has $\{v_2, \ldots, v_{d'_1+1}\}$ as the neighbors of v_1. Such a graph, if v_1 is removed, has k vertices and degree sequence given by $d'_2 - 1, \ldots, d_{d'_1+1} - 1, d_{d'_1}, \ldots, d'_n$. The induction hypothesis implies that the existence of such a graph is correctly determined by the algorithm. If such a graph exists, then the $k + 1$-vertex graph exists as well by attaching the new vertex v_1 to $v_2, \ldots, v_{d'_1+1}$.

Running time: Sorting the original sequence $d_1, d_2, \ldots, d_n$ can be done in $O(n \log n)$ time (using merge-sort, for example). There is at most one such recursive call corresponding to each d_i in the original sequence and or each recursive call, we perform $d'_1 \leq d_i$ decrements. Searching for $First(d'_i)$, $Last(d'_i)$, rearranging the linked list D'' and updating the list L can also be completed in $O(d'_1)$ number of operations. Thus, the total running time $\in O(n \log n + \sum_{i=1}^{i=n}(1 + d_i)) = O(n \log n + m)$ where $m = \sum_{i=1}^{i=n} d_i$.

Problem 3

Consider the traffic camera problem: the police department would like to set up traffic cameras to catch traffic offenders on every road. However, the department budget is being cut, and there is not enough money to place a camera at every intersection. Fortunately there is no need to place cameras at every intersection because a camera placed at an intersection can monitor all roads adjacent to this intersection. Given a network of roads, the traffic camera problem is to find the minimum number of traffic cameras that need to be placed at intersections (as well as the intersections where we need to place these cameras) such that all roads can be monitored. (Note that each road is adjacent to two intersections, one at each end.)

1. First, write down a graph-theoretic formulation of the problem. Do not forget to completely specify the inputs, any specific constraints, and the required outputs.

2. Below is a greedy algorithm for this problem. Does this algorithm work correctly? If so, write down a proof of correctness. If not, write down a counterexample.

   Initialize: Insersection set $S = \emptyset$. Road set R is the set of all roads.
   while R is non-empty **do**
      Pick an arbitrary road $r \in R$, and pick an intersection v adjacent to r. Add v to S.
      Delete from R all roads r' that are adjacent to v.
   end while
   Output S.

3. Suppose we modify the greedy algorithm to the algorithm below. Does this algorithm work correctly? If yes, write down a proof of correctness. If not, write down a counterexample.

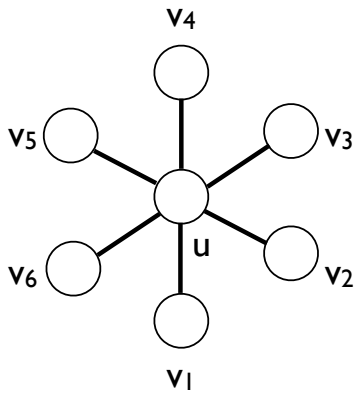
   Initialize: Insersection set $S = \emptyset$. Road set R is the set of all roads.

2: **while** R is non-empty **do**
3: Pick an intersection v from the set of intersections that is adjacent to *the maximum number of roads in* R. Add v to S.
4: Delete from R all roads r' that are adjacent to v.
5: **end while**
6: Output S.

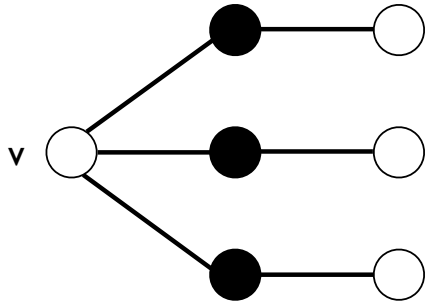
Solution

1. To formulate the problem graph-theoretically, we represent the network of roads by a graph $G = (V, E)$ where each vertex v represents an intersection, and each edge $e = (u, v)$ represents a road between intersections u and v. The input to the traffic camera problem is the graph $G = (V, E)$ representing the network of roads. The required output is a number m and a set S of m intersections such that (a) for every edge $e \in E$, there exists a vertex $v \in S$ which is adjacent to e and (b) there is no set $S' \subseteq V$ of size $< m$ such that property (a) holds. This problem is called the *minimum vertex cover problem* in the graph theory literature.

2. The algorithm does not work correctly. For example, in the following graph $G = (V, E)$, placing a camera at u will allow us to monitor all roads. However, as the algorithm selects roads and their adjacent intersections in arbitrary order, it may pick intersections $v_1, v_2, \ldots, v_6$ instead, which is highly suboptimal.



3. This greedy algorithm does not work correctly either. For example, in the following graph, the optimal solution is to place the cameras at the three vertices colored black, and has size 3. However, the greedy algorithm will always pick the degree the vertex v in the first round, and will thus always give a solution of size 4.


v