College of Computer & Information Science — Fall 2014
Northeastern University — Handout 9
CS7800: Advanced Algorithms — 30 September 2014

Sample Solution to Problem Set 2

Instructions:

- The assignment is due at the beginning of class on the due date specified. Late assignments will not be accepted.
- We encourage you to attempt and work out all of the problems on your own. You are permitted to study with friends and discuss the problems; however, *you must write up you own solutions, in your own words.*
- If you do collaborate with any of the other students on any problem, please list all your collaborators in your submission for each problem.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is prohibited.
- We require that all homework submissions be neat, organized, and *typeset.* You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions.

1. (10 points) Hopping from one minimum spanning tree to another

Let G be a weighted connected undirected graph.

(a) Prove that if T and T' are two minimum spanning trees of G, then there exists a sequence $\langle T_0, \ldots, T_k \rangle$, $k \geq 0$, such that: (i) T_i is a minimum spanning tree of G, $0 \leq i \leq k$, (ii) $T_0 = T$, (iii) $T_k = T'$, and (iv) $|T_{i+1} \setminus T_i| = 1$, $0 \leq i < k$ (i.e., T_i and T_{i+1} differ in exactly one edge).

Answer: The proof is by induction on $|T \setminus T'|$.

Induction Base: $|T \setminus T'| = 0$. The sequence is $\langle T \rangle$.

Induction Step: Let the statement be true for $|T \setminus T'| = k$, $k \geq 0$. For any MSTs T and T' such that $|T \setminus T'| = k$, there exists a sequence $\langle T_0, \ldots, T_k \rangle$, such that: (i) T_i is an MST, $0 \leq i \leq k$, (ii) $T_0 = T$, (iii) $T_k = T'$, and (iv) $|T_{i+1} \setminus T_i| = 1$, $0 \leq i <$.

Given a T' such that $|T \setminus T'| = k + 1$, we show the existence of an MST Y, such that $|T \setminus Y| = k$ and $|Y \setminus T'| = 1$. Together with the induction hypothesis, this gives us a sequence $\langle T_0, \ldots, T_k, T_{k+1}$, such that: (i) T_i is an MST, $0 \leq i < k$, (ii) $T_0 = T$, (iii) $T_{k+1} = T'$, (iv) $|T_{i+1} \setminus T_i| = 1$, $0 \leq i \leq k$.

Let x' be the minimum weight edge in $|T' \setminus T|$. (Note that since T and T' are MSTs , $|T' \setminus T| = |T \setminus T'| \geq 1$, and therefore x' exists.) Consider the tree $T' \setminus \{x'\}$. Note that removing x' from T' defines a cut and x' is a light edge of the cut. Clearly, there exists an edge $x \in T$ that crosses the cut; and at least one edge $x \neq x'$ of the cut has weight at most $w(x')$. Thus, $(T' \setminus \{x'\}) \cup \{x\}$ is a MST, and indeed $w(x') = w(x)$.

Let $Y = (T' \setminus \{x'\}) \cup \{x\}$. From the above arguments, Y is an MST. Also $|T \setminus Y| = |T \setminus T'| - 1 = k$, and $|Y \setminus T'| = 1$.

(b) Prove that if T and T' are two minumum spanning trees, then T and T' have the same "weight distribution" (i.e., for any weight w, both T and T' contain the same number of edges with weight w).

Answer: If T and T' are the same, there is nothing to prove. Else by (a), there exists a sequence $\langle T_0, \ldots, T_k \rangle$, $k \geq 0$ such that: (i) T_i is an MST, $0 \leq i \leq k$, (ii) $T_0 = T$, (iii) $T_k = T'$, and (iv) $|T_{i+1} \setminus T_i| = 1$, $0 \leq i < k$. If we show for $0 \leq i < k$, T_i and T_{i+1} have the same weight distribution we are done (by transitivity).

Let $T_i = (T_{i+1} \setminus \{x'\}) \cup \{x\}$. Then since T_i and T_{i+1} are MSTs, it follows that $w(x) = w(x')$. Since T_i and T_{i+1} differ in no element other than x and x', they have the same weight distribution.

2. (5 + 3 × 5 = 20 points) Broadcast via gossiping

The paradigm of gossiping is being considered as a robust mechanism for spreading information in a distributed network, or influence in a social network. Suppose we have an undirected connected network G with n nodes. A node, say r, has a piece of information M that it wants to broadcast to the entire network. Consider the following gossiping protocol.

In each step, each node that has a copy of M, sends a copy of M to a neighbor chosen uniformly at random. Assume that all the nodes are sychronized in their steps.

In this exercise, we want to place a bound on the *completion time* of the above protocol; that is the number of steps it takes before every node receives a copy of M.

(a) Write programs to simulate the above protocol on the following graphs: the star graph (one node having an edge to $n-1$ others), the complete graph (n nodes with all pairwise edges), the line, and the $\sqrt{n}\sqrt{n}$ two-dimensional grid. Plot the expected completion time, for each graph, as a function of n.

The rest of this exercise presents a sequence of steps leading to an analysis.

(b) Suppose a node u has a copy of M and degree d. What is the expected number of steps, in terms of d, before u sends a copy of M to a specific neighbor v?

Answer: The probability that u sends a copy of M to v in any given step is $1/d$. Thus, the expected number of steps it takes before u sends a copy of M to v equals:

$$\frac{1}{d} + 2 \cdot \frac{1}{d} \cdot \frac{d-1}{d} + 3 \cdot \frac{1}{d} \cdot \left(\frac{d-1}{d}\right)^2 + \ldots \sum_{i=1}^{\infty} i \cdot \frac{1}{d}\frac{d-1}{d}^{i-1}.$$

Using elementary algebra/calculus (see Equation A.8 on page 1148 of text), we simplify the above to obtain d as the answer.

(c) Let P be a shortest path from u to v. Prove that the sum of the degrees of all the nodes on P is at most $3n$.

Answer: We argue that a node x can be a neighbor of at most 3 nodes on a shortest path. Note that this is sufficient to establish the desired claim.

Suppose otherwise; let x be a neighbor of distinct nodes u_1, u_2, u_3, and u_4. Without loss of generality, assume that P first visits u_1, then u_2, then u_3, and then u_4. It follows that the subpath of P from u_1 to u_4 has at least three edges. However, replacing this subpath by the two-hop path $u_1 \to x \to u_4$ contradicts the fact that P is a shortest path from u to v.

(d) Using parts (b) and (c), derive an upper bound, in terms of n, on the expected number of steps it takes for an arbitrary node v to receive a copy of M.

Answer: By part (b) and linearity of expectation, the expected number of steps it takes for an arbitrary node v to receive a copy of M is at most the sum of the degrees of the nodes along the shortest path from r to v, which is at most $3n$ by part (c).

Unfortunately, part (d) does not give us a bound on the expected completion time, since it only bounds the time taken for an arbitrary node v – not *all nodes* – to receive M.

(e) Let us revisit part (b). Again, suppose a node u has a copy of M and degree d. Find an upper bound, in terms of d, on the number of steps it takes for a specific neighbor v of u to receive a copy of M from u with probability at least $1 - 1/n^3$.

Answer: Let t be the number of steps it takes for v to receive a copy of M from u with probability at least $1 - 1/n^3$. The probability that v has not received a copy of M from u in t steps is $(1 - 1/d)^t$. So t is the first step at which this probability is at most $1/n^3$; in other words

$$t \le \ln(1/n^3)/\ln(1 - 1/d) \le 3d \ln n,$$

where we use the inequality $(1 - 1/d)^d \le 1/e$ for $d \ge 1$.

(f) Using parts (c) and (e), derive an upper bound, in terms of n, on the number of steps it takes for an arbitrary node v to receive a copy of M with probability at least $1 - 1/n^2$. Argue that the same bound yields an upper bound on the number of steps it takes for *all nodes* to receive a copy of M with probability at least $1 - 1/n$.

Answer: Consider a shortest path from r to v. In at most $3d_r \ln n$ steps, where d_r is the degree of r, the message crosses the first hop (to, say node u) with probability at least $1 - 1/n^3$. Conditioned on the fact that M has reached u, in at most $3d_u \ln n$ additional steps, where d_u is the degree of u, the message crosses the second hop with probability at least $1 - 1/n^3$. Thus, in at most $3(d_r + d_u) \ln n$ steps, M has reached u with probability at least $1 - 2/n^3$ (using Boole's inequality – see C.2-2 on page 1195 of text). Continuing with this argument and invoking part (c), we obtain that M reaches an arbitrary node v in at most $3n \log n$ steps with probability at least $1 - n/n^3 = 1 - 1/n^2$.

The probability that M has failed to reach a *specific node* v in $3n \log n$ steps is at most $1/n^2$. Thus, the probability that there exists a node v that M has failed to reach in $3n \log n$ steps is at most $1/n$ (using Boole's inequality – see C.2-2 on page 1195 of text).

3. (10 points) Clustering to maximize separation

Many scientific applications, including medical imaging, classification of astronomical objects, and web document categorization, require partitioning a given set of objects into disjoint sets. Here is one of many variants of clustering problems. You are given a set S of n points $v_1, \ldots, v_n$, together with their pairwise distances. You may assume that the distances are symmetric and nonnegative, and that the distance between a point and itself is zero, while that between two distinct points is nonzero. Let $d(u, v)$ denote the distance between two points u and v. For any two nonempty sets

X and Y of points, define the *distance* $d(X,Y)$ *between* X *and* Y to be

$$\min_{u\in X, v\in Y} d(u,v).$$

Given a number $m \le n$, you are asked to partition S into m nonempty disjoint sets $S_1, \ldots, S_m$ so as to maximize the minimum distance between any pair of disjoint sets; that is, determine disjoint sets S_1 through S_m that maximizes $\min_{i\neq j} d(S_i, S_j)$ subject to the constraint that $\cup_i S_i = S$.

Give an algorithm to solve the above problem. Analyze the running time. Make your algorithm as efficient as you can, in terms of its worst case running time.

Answer: We first construct a complete graph K in which we have a vertex for each point, and an edge between every pair of vertices. The weight of each edge equals the distance between the two endpoints.

We need to maximize the minimum distance, subject to the condition that we partition the points into m nonempty disjoint sets. If m is 1, then there is nothing to do.

If m is 2, then we need to split the nodes into two nonempty groups so that the maximum distance between the two groups is minimized. Suppose the minimum value achieved by an optimal solution is M. Then, if we remove all edges in K of weight at least M, we will split the graph into at least two connected components. And, removing only edges of any higher weight will not yield us two connected components (since M is the optimal value).

More generally, we have the following. Suppose we remove all the edges in the graph of weight at least W. And suppose this is when the graph splits into at least k connected components. Then, every pair of vertices residing in two different groups is within distance at least W. And any different way of partitioning the nodes into k groups will include an edge from within one of the connected components, whose length is at most W. So any other partitioning will have an objective value of at most W.

Here is the algorithm:

1. Build the complete weighted graph K.
2. Sort the edges in nonincreasing order of their weight.
3. Keep removing edges, until the number of connected components is at least m.
4. The weight of the last edge removed is the objective value achieved, and the connected components the desired groups.

The unoptimized running time of the algorithm is $O(n^2 \log n + n^3)$ – for constructing the graph K and sorting the edges. The maintenance of the connected components can be done in $O(n)$ time per each of the $O(n^2)$ iterations. Using a more clever connected components maintenance, one can bring the running time down to $O(n^2 \log n)$.

4. (10 points) Communication on Planet Anti-Huffman

Alice and Bob find themselves in the strange planet of *Anti-Huffman*. Suspicious of their neighbors as usual, they decide to use an encoding scheme for communication. Their encoding is based on a set S of m *code words* that they both share.

Alice wants to send a data string D of length n to Bob. Alice would like to determine whether D can be encoded as a concatenation of a sequence of code words from S. Furthermore, if the encoding exists, she would like to determine an encoding that uses the minimum length sequence of code words.

For example, consider the binary alphabet and let $S = \{0, 10, 0101\}$, and $D = 0101010100$. The encoding that splits D as $0101; 0; 10; 10; 0$ has length five, while a minimum-length encoding that splits D as $0101; 0101; 0; 0$ has length four. On the other hand, there is no encoding for the string 111 using code words in S.

Alice would like to solve the encoding problem efficiently. Fortunately for her, planet Anti-Huffman only supports *prefix-full codes.* A set S of code words is prefix-full if it satisfies the following property: if $s \in S$, then every nonempty prefix of s is also in S. An example of such a set is $\{0, 1, 01, 011, 010, 0111, 01110, 01111\}$. (A string x is a *prefix* of string y if there exists another string z such that y is a concatenation of x and z; that is, $y = x; z$. Thus, for example, 011 is a prefix of 01110 since $01110 = 011; 10$.)

Give an efficient algorithm for Alice to determine a minimum-length encoding of a given string D using code words from a prefix-full code. If no such encoding exists, then the algorithm must indicate so. Justify the correctness of your algorithm. Analyze the running time of your algorithm. Make your algorithm as efficient as you can, in terms of its worst-case running time.

Answer: We present a greedy algorithm for the above problem. The algorithm proceeds as follows:

```
GREEDY($D$)
0. If $D$ is empty, then return empty string
1. Find the largest code word that is a suffix of $D$
2. If no such code word exists, then return "No match"
3. Let $D = D'; c$
4. Return GREEDY($D'$);$c$
```

Optimal substructure property. Let $\sigma^*(D)$ denote an optimal encoding for D. If $\sigma^*(D) = \sigma^*(D'); c$, then we claim that $\sigma^*(D')$ is an optimal encoding for D'. Otherwise, we can replace it by a smaller encoding and contradict the assumption that $\sigma^*(D)$ is an optimal encoding for D.

Greedy choice property. Let c be the largest code word that is a suffix of D. If possible, suppose an optimal encoding contains $c' \neq c$ as the last code word. Clearly $|c'| < |c|$. Let $D = D_1; c'$. Let c_1 denote the last code work in D_1. We consider two cases. If $c_1; c'$ is a prefix of c, then there exists a code word $c_1; c'$ in S, which implies the suboptimality of the alleged optimal solution. Otherwise, we can replace $c_1; c'$ by $c_2; c$, where c_2 is a prefix of c_1 to obtain a new optimal solution with the desired property.

The optimal substructure and greedy choice properties establish that algorithm GREEDY yields an optimal solution. We now analyze the running time of the algorithm.

Running time. To analyze the running time, we need to describe how to implement step 1 of the algorithm. Suppose, the input to the problem is simply an array S of code words and the word D.

One simple way to determine the longest code word that matches a suffix of D is the following: (i) Maintain a list L of code words that match the current suffix s of D and the current best match;

these are initialized to S and ε, respectively; (ii) Set i to 1; (iii) Compare the ith position of D from the end with the ith position of each of code words in D, from the end, and set L to the list of those code words where there is a match; (iv) If L is empty, then return s; otherwise, if there exists a code word of length i in L, then set s to that code word; (v) Increment i and go to step (iii).

If the code word returned in the above algorithm is empty, then there is no code word that matches a suffix, and hence the given word cannot be expressed as a sequence of code words.

The running time for step 1 of Greedy is $O(m\ell)$, where ℓ is the length of the longest code word that matches a suffix of D. The length of D' is $n - \ell$. Therefore, the overall running time is given by the recurrence:

$$T(|D|, m) = T(|D'|, m) + O(m(|D| - |D'|)),$$

where $T(n, m)$ is the time taken for finding a solution for string D and m code words. The above recurrence yields $T(n, m) = O(nm)$.

Improvement in running time. In certain scenarios, we can improve the running time by processing the set of codewords so as to store them in the form of a tree. In particular, one can preprocess the given set of code words and obtain a tree with labels on the edges and marking a subset of the nodes such that each path from any marked node to the root represents a code word. For the prefix-full case, every internal node would be marked.

Constructing the tree takes time $O(mks)$, where m is the number of codewords, k is the length of the largest codeword and s is the size of the alphabet. The term s appears in the running time since the edges of the tree are labeled by the alphabet letters, and it takes s time, in the worst-case, to determine which of the edges to traverse. You may ignore the s term if the alphabet is binary. One can improve the expected running time using hashing, but the worst-case time remains the same.

With this preprocessing, the running time of Greedy is $O(mks + sn)$. Thus, for a binary alphabet, the running time is $O(mk + n)$. If we wish to obtain a time bound in terms of n and m only, then we can bound k by n (since we can ignore code words that are longer than n) and obtain a running time of $O(mn)$.