College of Computer & Information Science
Northeastern University
Fall 2017

CS5800
Algorithms
23 October 2017

Sample Solutions to Midterm I

Problem 1. (2 + 2 + 1 + 2 = 7 points) Gale-Shapley algorithm

For the 3 men m_1, m_2, m_3 and the 3 women w_1, w_2, w_3 the following lists of preferences are given:

man	first	second	third
m_1	w_1	w_2	w_3
m_2	w_2	w_1	w_3
m_3	w_1	w_3	w_2

woman	first	second	third
w_1	m_2	m_1	m_3
w_2	m_3	m_2	m_1
w_3	m_1	m_2	m_3

(a) List the stable matching returned by the Gale-Shapley algorithm for the above lists of preferences assuming the **men** propose to the **women**.

Answer: $m_1 - w_1$, $m_2 - w_2$, $m_3 - w_3$

(b) List the stable matching returned by the Gale-Shapley algorithm for the above lists of preferences assuming the **women** propose to the **men**.

Answer: $w_1 - m_2$, $w_2 - m_3$, $w_3 - m_1$

(c) Does there exist a man that got a better result in part (a) than in part (b)? If so, indicate which man.

Answer: This is true for m_1.

(d) Prove that the following algorithm determines if there is a unique stable matching for a given lists of preferences.

1. Apply the Gale-Shapley algorithm with the men proposing to the women.
2. Apply the Gale-Shapley algorithm with the women proposing to the men.
3. If the result of the two Gale-Shapley algorithms are the same, then there is a unique stable matching. Otherwise there is more than one possible stable matching.

Answer: There are different ways to prove this. Here is one.

We know that when applying the Gale-Shapley algorithm on sets M and W such that the men propose to the women, the stable matching we get is:

$$S = \{(m, \text{best}(m)) \ : \ m \in M\}$$

Similarly, if the women propose to the men, we get:

$$S = \{(w, \text{best}(w)) \ : \ w \in W\}$$

Assume there is more then 1 possible stable matching. Let S_1 be a stable matching obtained by the G-S algorithm when the men propose to women, and let S_2 be a different stable matching. It follows that for some man m_0, there exists $w_1 \neq w_2$ such that $m_0 - w_1$ is a pair in S_1, and $m_0 - w_2$ is a pair in S_2. We know that m_0 prefers w_1 over w_2. In S_2 we have a pair $m_1 - w_1$ (w_1 must be paired with some man m_1). The fact that S_2 is stable means that w_1 prefers m_1 over m_0. (Otherwise, m_0 prefers w_1 over his pair and w_1 prefers m_0 over her pair. That would mean that S_2 is not stable.) It follows that if the women propose to the men, w_1 will end up with a men that she prefers over m_0. In particular w_1 will not end up with m_0.

We see that if there exist more than 2 different stable matchings, then the G-S algorithm returns different matchings depending on whether the men or women propose.

It follows that by applying the G-S algorithm twice (once with the men proposing to the women and once with the women proposing to the men), we can determine whether there exists a unique stable matching.

Problem 2. (3 + 3 = 6 points) Asymptotic notation and recurrences

(a) Let $f(n), g(n)$ and $h(n)$ be positive, monotonically non-decreasing functions. Prove or disprove the following statement.

If $f(n), g(n) = \Theta(h(n))$ then $f(n) \cdot g(n) = \Theta((h(n))^2)$.

Answer: True.

There exist constants c_1, c_2, n_0 such that for $n > n_0$ we have:

$$c_1 \cdot h(n) \leq f(n) \leq c_2 \cdot h(n)$$

Similarly, there exist constants c_3, c_4, n_1 such that for $n > n_1$ we have:

$$c_1 \cdot h(n) \leq g(n) \leq c_2 \cdot h(n)$$

It follows that for $n \geq \max\{n_0, n_1\}$ we have:

$$\underbrace{c_1 \cdot h(n) \cdot c_3 \cdot h(n)}_{c_1 c_2 \cdot (h(n))^2} \leq f(n) \cdot g(n) \leq \underbrace{c_2 \cdot h(n) \cdot c_4 \cdot h(n)}_{c_2 c_3 \cdot (h(n))^2}$$

(b) Let $T(n)$ be defined by:

$$T(n) = \begin{cases} 1, & n = 1; \\ 16T\left(\lfloor \frac{n}{4} \rfloor\right) + 2n\log n, & n > 1. \end{cases}$$

Solve the recurrence to find a function $h(n)$ such that $T(n) = \Theta(h(n))$. Show your work. You may ignore floors and ceilings in your calculation.

Answer: We apply the Master Theorem. We have $n^{\log_b a} = n^2$, and $f(n) = n \log n$. We see that $n \log n = O(n^{2-\epsilon})$ for any $eps < 1$. For instance, take $\varepsilon = 0.5$. Then, $(n \log n)/n^{1.5} = \log(n)/n^{0.5}$. We can take the limit as $n \to \infty$, and applying L'Hopital's rule once, we obtain that the limit of the ratio is 0. So $n \log n = O(n^{2-0.5})$. Applying the relevant case of Master Theorem, we obtain $T(n) = \Theta(n^2)$.

Problem 3. (5 points) A variant of Mergesort

Consider the following variant of Mergesort. If the array has more than two elements, it recursively sorts the first two-third and the last one-third. Then, it merges the first two-third with the last one-third. If the array has at most two elements, then it trivially sorts the array. For completeness, here is the pseudocode for sorting the array $A[\ell \ldots r]$.

$\textsc{Newsort}(A, \ell, r)$:
if $r - \ell + 1 = 1$:
  exit
if $r - \ell + 1 = 2$ **and** $A[\ell] > A[r]$:
  swap $A[\ell] \leftrightarrow A[r]$
  exit
if $r - \ell + 1 > 2$:
  $m \leftarrow \lfloor (r - \ell + 1)/3 \rfloor$
  $\textsc{Newsort}(A, \ell, \ell + 2m - 1)$ [Sort first two-third]
  $\textsc{Newsort}(A, \ell + 2m, r)$ [Sort last one-third]
  $\textsc{Merge}(A, \ell, \ell + 2m - 1, r)$ [Merge first two-third with last one-third]

Write a recurrence relation for the worst-case running time of $\textsc{Newsort}$. Solve the recurrence relation to obtain a Θ-bound on the running time of the algorithm, in terms of the length n of the array. You may use any of the methods to solve the recurrence.

Answer: We get the following recurrence.

$$T(n) = T(\lfloor 2n/3 \rfloor) + T(\lceil n/3 \rceil) + \Theta(n).$$

If we ignore floors and ceiling and use the recursion tree approach, each level contributes $\Theta(n)$ until level $\log_3 n$. There are at most $\log_{3/2} n$ levels. So we obtain $T(n) \leq n \log_{3/2} n$ and at least $n \log_3 n$. So $T(n) = O(n \log n)$.

We can also prove using substitution method and induction.

Problem 4. (6 points) Sorting a zig-zag array

We say that an array $A[0 \dots n-1]$ of distinct integers is *zig-zag*, if there exists an index i, $0 \leq i < n$ such that if we append the sub-array $A[0 \dots i-1]$ after sub-array $A[i \dots n-1]$, we obtain an array sorted in increasing order. Put it another way, A is zig-zag if the array $B[0 \dots n-1]$ given by $B[j] = A[(j+i) \bmod n]$ for $0 \leq j < n-1$ is a sorted array (in increasing order).

For example, the arrays $[3, 5, 9, -1, 0, 2]$, $[1, 2, 3, 4, 5, 6]$, and $[9, 1, 3, 5, 6, 7]$ are all zig-zag arrays while $[-1, 8, 5, 4, 9, 0]$ is not a zig-zag array.

Give an algorithm that takes as input a zig-zag array $A[0 \dots n-1]$ and determines the smallest element of A, while accessing at most $O(\log n)$ elements of A. For partial credit, you may give an algorithm that has a worse bound on the number of elements of A it accesses.

Answer: We will use binary search and find the index with the largest element and return the one following it (circularly, if needed).

1. $\ell = 0, r = n-1$.
2. while $r - \ell \geq 0$:
   (a) $m = \lfloor (\ell + r)/2 \rfloor$.
   (b) if $A[m] > A[\ell]$, then $\ell = m$, else $r = m - 1$.
3. return $A[(\ell + 1) \bmod n]$.

Worst-case running time is $O(\log n)$.

Problem 5. (6 points) Identifying useless edges

Let $G = (V, E)$ be an undirected unweighted graph and let s be a given vertex in G. Call an edge (u, v) of G *useless* if it does not appear on any shortest path from s to u *and* does not appear in any shortest path from s to v. Give an algorithm that takes as input a graph G with n vertices and m edges, and a vertex s, and returns the set of all useless edges of G. State the worst-case running time of your algorithm, in terms of n and m.

Your grade for this question will be determined on the basis of the correctness of your algorithm and its efficiency, given by its worst-case running time. Partial credit may be given for non-optimal algorithms provided they are correct and well explained.

Answer: Note that u and v are not given as input. Your algorithm needs to return all useless edges in G with respect to s.

An edge (u, v) is useless if and only if u and v are at the same level of the BFS tree.

1. Perform a BFS and calculate level for each vertex.
2. Set useless-set to empty.
3. For each edge (u, v), if level of u is the same as level of v, then add (u, v) to useless-set.
4. Return useless-set.