University of Washington
Department of Computer Science and Engineering
CSE 417, Winter 2020

October 27, 2023

Old midterm questions with solutions

Problem 1. Stable Marriage (10 points):

Show that the Gale-Shipley Stable Marriage algorithm can take $\Theta(n^2)$ steps with appropriate choice of preference lists. Give preference lists and an ordering of the proposals that require $\Theta(n^2)$ steps. Explain why your example achieves the bound.

Hint: This can be done with all of the M's having the same preference lists, and all of the W's having the same preference lists.

Solution :

Give the M's the preference list $\{w_1, w_2, \ldots, w_n\}$, and the W's the preference list $\{m_1, m_2, \ldots, m_n\}$. During the i-th round, each unmatched M proposes to the i-th in its list. Exactly one new M is matched each round, so there are $n - i + 1$ in round i. The total number of proposals is

$$\sum_{i=1}^{n}(n-i+1) = \sum_{j=1}^{n} j = \frac{n(n+1)}{2}.$$

The order of the M's and W's preference lists *don't* matter in this argument - it just depends on on the lists being all the same.

Problem 2. Big Oh (10 points):

Let q, r, and s be positive constants. Prove that $qn^2 + rn + s$ is $O(n^2)$ using the formal definition of $O(\cdot)$.

Big $O(\cdot)$ definition: $f(n)$ is $O(g(n))$ if there exists $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, $f(n) \leq cg(n)$.

Solution :

Let $c = q + r + s$ and $n_0 = 1$. For $n \geq 1$ we have:

$$qn^2 + rn + s \leq qn^2 + rn^2 + sn^2 = (q + r + s)n^2 = cn^2.$$

Problem 3. True or False (30 points):

Determine if the following statements are true or false. Provide a short justification for each answer.

a) *True or false:* If G is a directed graph on n vertices where every vertex has out degree at least two, then G has a cycle. Justify your answer.

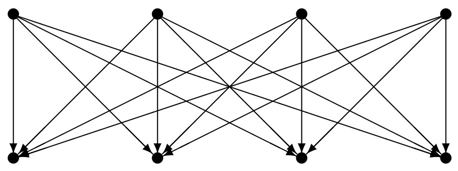
Solution :

True. Choose any vertex, and follow a path until there is a repeated vertex. Since the out degree of every vertex is at least one, there is always an edge to pick to lead to another vertex. A repeated vertex must always be found within n steps, hence, there is a cycle.

b) *True or false:* If G is a directed graph on n vertices with at least $2n$ edges, then G has a cycle. Justify your answer.

Solution :

False. The complete bipartite directed graph (or $K_{n,n}$) for $n \geq 4$. For $n = 4$, the graph has 8 vertices and 16 edges.



c) *True or false:* If G is a directed graph on n vertices, with distinct vertices r and s, where there is a path from r to every vertex in the graph, and there is a path from s to every vertex in the graph, then there is a cycle in the graph. Justify your answer.

Solution :

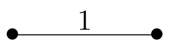
True.

The path from r to s followed by the path from s to r is a cycle.

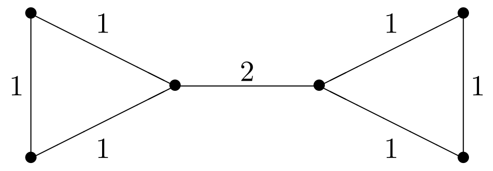
d) *True or false:* If G is an undirected graph with edge weights, and edge e has weight strictly greater than any other edge in the graph, then e cannot be in a minimum spanning tree for G. Justify your answer.

Solution :

False. A trivial counter example is a graph with a single edge:



A non-trivial example is where the most expensive edge is the only connection between two components:



e) *True or false:* If G is an undirected graph with edge weights, and edge e has weight strictly less than any other edge in the graph, then e must be in every minimum spanning tree for G. Justify your answer.

Solution :

True. This follows from the edge inclusion lemma. Suppose $e = (u, v)$ has cost less than any other edge. We use $\{u\}$ in the edge inclusion lemma as (u, v) is the minimum cost edge between $\{u\}$ and $V - \{u\}$.

f) *True or false:* If G is a undirected graph on n vertices with more than $n/2$ connected components, then at least one of the connected components is an isolated vertex. Justify your answer.

Solution :

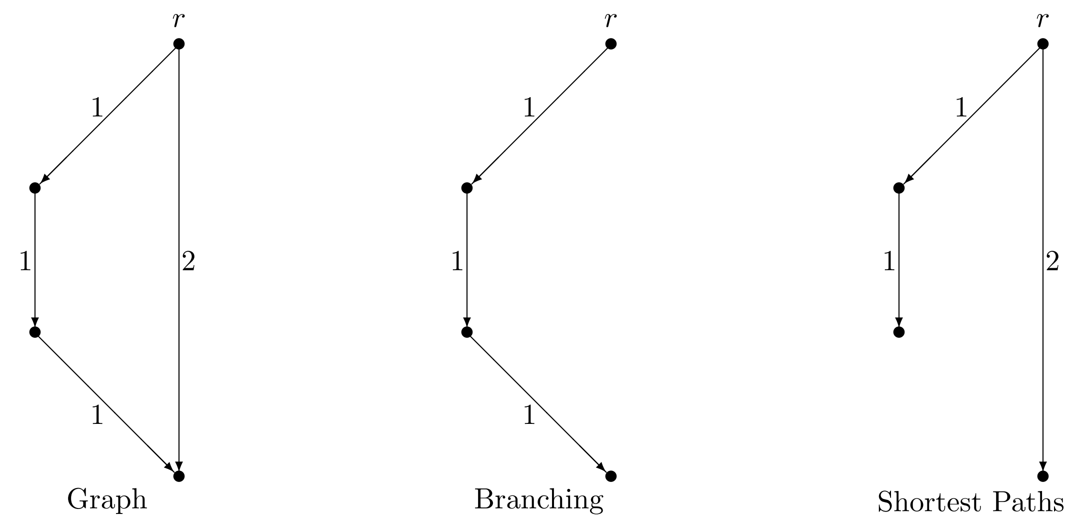
True. Suppose there are no isolated components, so each component has size at least 2. However, if we have more than $\frac{n}{2}$ components, we have more than $2 \cdot \frac{n}{2} = n$ vertices.

Problem 4. Minimum Weight Branching (10 points):

A branching is a rooted subtree in a directed graph where there is a path from the root r to every vertex in the graph. The *minimum branching problem* is: given a directed graph with weights on the edges and a specified vertex r, find a branching of minimum weight rooted at r.

Show that Dijkstra's shortest paths algorithm *does not* solve this problem. Specifically, give a graph where the shortest paths found by Dijkstra's algorithm do not form a minimum weight branching.

Solution :



(Note: This problem should have specified that the all edges have non-negative costs to avoid solutions with a malfunctioning Dijkstra's algorithm.)

Problem 5. One-Two Knapsack Problem (20 points):

The *Knapsack Problem* is: Given a collection of items $I = \{i_1, \ldots, i_n\}$ and an integer K where each item i_j has a weight w_j and a value v_j, find a subset of the items with weight at most K which maximizes the total value of the set. More formally, we want to find a subset $S \subseteq I$ such that $\sum_{i_k \in S} w_k \leq K$ and $\sum_{i_k \in S} v_k$ is as large as possible.

We define the *density* of d_j of item i_j to be $d_j = v_j/w_j$. A natural greedy algorithm for the knapsack problem is to consider the items in order of decreasing density, and place each item into the knapsack if there is still sufficient space for the item.

For this problem, we restrict the weights of the items to be either 1 or 2. For convenience, we assume the capacity K of the knapsack is an even number.

a) Given an example that shows that the greedy algorithm based on sorting items by density does not necessarily give an optimal solution, even if the weights are restricted to 1 and 2.

Solution L:

et $K = 2$, and we will have items $\{i_1, i_2, i_3\}$, where i_1 has weight 1 and value 2, i_2 has weight 1 and value 0, and i_3 has weight to and value 3.

The greedy algorithm finds the solution $\{i_1, i_2\}$ with value 2, while the optimal solution is $\{i_3\}$ with value 3.

b) Describe an efficient algorithm that finds an optimal solution to the knapsack problem when the weights are restricted to 1 and 2.

Solution :

Step 1. Sort the items of size 1 by value and combine adjacent items into new items of weight 2. (If there is a left over item, make it an item of weight 2.)

Step 2. Select the largest $\frac{K}{2}$ items of weight 2 (including both the original items of weight 2, and the combined items from Step 1.

c) Provide a justification that your algorithm is correct.

Solution :

For convenience, we assume the values are distinct. This assumption can be removed by sorting equal value items in a consistent manner, such as using the item number as a secondary key.

The solution S constructed by the algorithm has the following properties:

1. Every item of weight 1 in S has greater value than than every item of weight 1 not in S.
2. Every item of weight 2 in S has greater value than than every item of weight 2 not in S.
3. Every item of weight 2 in S has greater value than than every pair of items of weight 1 not in S.
4. Every pair of items of weight 1 in S has greater value than item of weight 2 not in S.

Let T be a set of items different from S. Suppose $x \in T$ and $x \notin S$.

Case 1: Weight of $x = 1$ and there is a $y \in S$, $y \notin T$ with weight 1. In this case, y can be added to T and x removed to show T is not optimal.

Case 2: Weight of $x = 1$ and there is no $y \in S$, $y \notin T$ with weight 1. In this case, there is another $z \in T$ of weight 1 with $z \notin S$. An item of weight 2 can be added to T and x and z removed to show T is not optimal.

Case 3: Weight of $x = 2$. In this case either an item of weight 2, or two items of weight 1 can be added to T and x removed to show T is not optimal.

This shows that T is not optimal.

Problem 6 (10 points):

Consider the stable matching problem.

a) Show that it is possible to have a *last-choice* match: There exists an instance of the problem with a stable matching M that has m matched with w, where w is m's last choice, and m is w's last choice.

Answer: An example of a problem instance is a 2×2 example with:

$$\begin{array}{ll} m_1 : w_1, w_2 & w_1 : m_1, m_2 \\ m_2 : w_1, w_2 & w_2 : m_1, m_2 \end{array}$$

Since m_1 and w_1 are each other's first choice, they are matched, leaving m_2 and w_2 to be matched.

Another example is the trivial example, with just m and w. In this case, m and w are matched, and are their last choices (as well as their first choices).

b) Is it possible for a stable matching to have two *last-choice* matches: could a stable matching M have m_1 matched with w_1 where m_1 is w_1's last choice and w_1 is m_1's last choice, and m_2 matched with w_2 where m_2 is w_2's last choice and w_2 is m_2's last choice? Justify your answer.

Answer: No. If there are two last choice matches (m_1, w_1) and (m_2, w_2), then (m_1, w_2) is an instability, since m_1 preferes w_2 to w_1 and w_2 prefers m_1 to m_2.

Problem 7 (10 points):
Show that
$$\sum_{k=0}^{\log n} 4^k$$
is $O(n^2)$.
Answer:
$$\sum_{k=0}^{j} x^k = \frac{x^{j+1}-1}{x-1},$$
so
$$\sum_{k=0}^{\log n} 4^k = \frac{4^{\log n+1}-1}{4-1} = \frac{4n^2-1}{3}$$
which is $O(n^2)$.

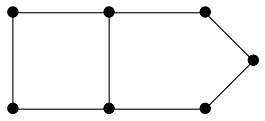
Problem 8 (10 points):
Let $G = (V, E)$ be an undirected graph.

a) True or false: If G is a tree, then G is bipartite. Justify your answer.

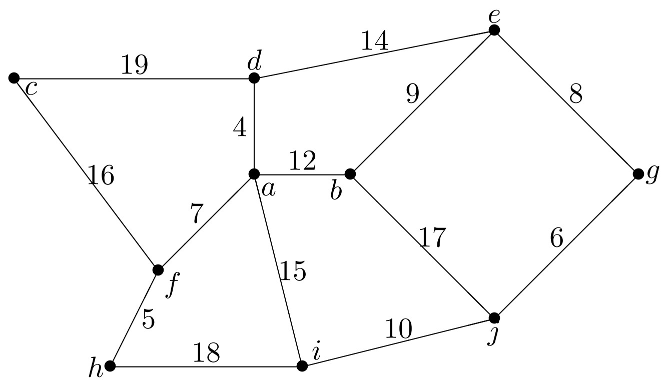
   True. If we label the vertices based upon their distance from the root, we observe that all edges go between even vertices and odd vertices.

b) True or false: If G is not bipartite, then the shortest cycle in G has odd length. Justify your answer.

   False. A counter example is a graph made up of a cycle of length 4 connected to a cycle of length 5.



Problem 9 (10 points):
Consider the following undirected graph G.



a) Use the Edge Inclusion Lemma to argue that the edge (a, b) is in every Minimum Spanning Tree of G.

Answer: (a, b) is the cheapest cost edge between $\{a, c, d, f, h\}$ and $\{b, e, i, j, g\}$.

b) Use the Edge Exclusion Lemma to argue that the edge (a, i) is never in a Minimum Spanning Tree of G.

Answer: (a, i) is the most expensive edge on the cycle $\{a, i, j, g, e, b\}$.

Problem 10 (10 points):

The knapsack problem is: Given a collection of items $I = \{i_1, \ldots, i_n\}$ and an integer K where each item i_j has a weight w_j and a value v_j find a subset of the items which has weight at most K and maximizes the total value in the set. More formally, we want to find a subset $S \subseteq I$ such that $\sum_{i_k \in S} w_k \leq K$ and $\sum_{i_k \in S} v_k$ is as large as possible.
Suppose that the items are sorted in decreasing order of value, so that $v_i \geq v_{i+1}$. A simple greedy algorithm for the problem is:

```
$CurrWeight := 0$;
$Sack := \emptyset$;
for $j := 1$ to $n$
  if $CurrWeight + w_j \leq K$ then
    $Sack := Sack \cup \{i_j\}$
    $CurrWeight := CurrWeight + w_j$
```

a) Show that the greedy algorithm does not necessarily find the maximum value collection of items that can be placed in the knapsack.

Answer: The following counter example shows that the greedy algorithm does not find the optimal soultion. Let $K = 2$ and suppose there are three jobs $\{i_1, i_2, i_3\}$ with $v_1 = 3$, $w_1 = 2$, $v_2 = 2$, $w_2 = 1$, and $v_3 = 2$, $w_3 = 1$. The greedy algorithm selects i_1, while the optimal solution is i_2 and i_3.

b) Prove that if all weights are the same, then the greedy algorithm finds the maximum value set. (For convenience, you may assume that each item has weight 1).

Proof: If there are fewer than K items, then the greedy algorithm selects all items, so assume there are at least K items. The greedy algorithm constructs the solution $\{i_1, \ldots, i_K\}$. Let $Opt = \{i_{j_1}, \ldots, i_{j_K}\}$ (where $j_r < j_{r+1}$). We must have $r \leq j_r$, so $v_r \leq v_{j_r}$ for all r, so the value of the set constructed by the greedy algorithm is no more the the optimal.

Problem 11 (10 points):

Give solutions to the following recurrences. Justify your answers.

a)

$$T(n) = \begin{cases} 2T(\frac{n}{3}) + n & \text{if } n > 1 \\ 1 & \text{if } n \leq 1 \end{cases}$$

Answer: Unrolling the recurrence, we get:

$$T(n) = 2T\left(\frac{n}{3}\right) + n = 4T\left(\frac{n}{9}\right) + \frac{2n}{3} + n = 8T\left(\frac{n}{27}\right) + \frac{4n}{9} + \frac{2n}{3} + n,$$

which gives us:

$$T(n) = \sum_{i=0}^{\log_3 n} \left(\frac{2}{3}\right)^i n \leq 3n,$$

so the solution is $O(n)$.

b)

$$T(n) = \begin{cases} 8T(\frac{n}{2}) + n^3 & \text{if } n > 1 \\ 0 & \text{if } n \leq 1 \end{cases}$$

Answer: Unrolling the recurrence, we get:

$$T(n) = 8T\left(\frac{n}{2}\right) + n^3 = 64T\left(\frac{n}{4}\right) + n^3 + n^3 = 512T\left(\frac{n}{8}\right) + n^3 + n^3 + n^3.$$

We observe that each level of the recurrence yields the n^3, and the depth of the recurrence is $\log 2n$, so the answer is $O(n^3 \log n)$.

Problem 12 (10 points):

A k-wise merge takes as input k sorted arrays, and constructs a single sorted array containing all of the elements of the input arrays.

a) Describe an efficient divide and conquer algorithm *MultiMerge*$(k, A_1, \ldots, A_k)$ which computes a k-wise merge of its input arrays.

Answer: We give a recursive algorithm, which makes use of a routine *Merge*(A_1, A_2) which merges a pair of sorted arrays, and returns the result. We assume that k is a power of two, and that $k \geq 2$.

```
MultiMerge($k, A_1, \ldots, A_k$)
  if $k = 2$
    return Merge($A_1, A_2$);
  else
    $B_1 :=$ MultiMerge($\frac{k}{2}, A_1, \ldots, A_{\frac{k}{2}}$);
    $B_1 :=$ MultiMerge($\frac{k}{2}, A_{\frac{k}{2}+1}, \ldots, A_k$);
    return Merge($B_1, B_2$);
```

b) What is the run time of your algorithm with input of k arrays of length n. Justify your answer.

The run time of the algorithm is $O(kn \log k)$. One way to see this is to write the run time as a recurrence. Let cn be a bound on the cost of merging two arrays of length n. The recurrence for the run time is $T(k) = 2T\left(\frac{k}{2}\right) + ckn$, so the solution is $ckn \log k$.