02/25/2022

- Q&A from Homework 4
- Review of
  - 2020 Midterm Exam
  - 2021 Midterm Exam

2020 Midterm Exam:

1. (10 pts.) We consider disjoint sets and wish to perform two operations on these sets.

   (1) **Union:** If S_i and S_j are two disjoint sets, then their union is $S_i \cup S_j = \{x \mid x \text{ is in } S_i \text{ or } S_j\}$.

   (2) **Find(i):** Given an element i, find the set containing i.

   We assume that each set is represented using a directed tree such that nodes are linked from children to parents. Three algorithms to perform the union operation $UNION(S_i, S_j)$ were discussed in class. Now, consider the following algorithm.

   Algorithm 4: Make the root of the tree with lower depth be a son of the root of the tree with higher depth. (If two trees have the same depth, choose arbitrarily.)

   Show that $UNION$ can be done $O(1)$ time and $FIND$ can be done in $O(\log n)$ time.

Sol. Let T be a tree with n nodes created by Algorithm 4. We will prove that $depth(T) \leq \lfloor \log n \rfloor + 1$ using induction on n. If $n = 1$, it is trivial, and assume that the claim is true for all trees with i nodes where $i \leq n - 1$. Now consider a tree T with n nodes where $UNION(S_k, S_j)$ is the last UNION operation to form T. Assume $m \leq n - m$ (i.e., $m \leq \frac{n}{2}$) where that the number of nodes in S_k and S_j, respectively, are $n - m$ and m. There are two cases considered.

Case 1. $depth(S_k) \geq depth(S_j)$: In this case, the root of S_j becomes a son of the root of S_k. Hence, $depth(T) = \max\{depth(S_k),\ depth(S_j) + 1\}$. Note that by induction hypothesis, $depth(S_k) \leq \lfloor \log(n-m) \rfloor + 1 \leq \lfloor \log n \rfloor + 1$. It is also note that $depth(S_j) \leq \lfloor \log m \rfloor + 1 \leq \lfloor \log \frac{n}{2} \rfloor + 1 \leq \lfloor \log n \rfloor$. Therefore, $depth(T) \leq \lfloor \log n \rfloor + 1$.

Case 2. $depth(S_k) < depth(S_j)$: In this case, the root of S_k becomes a son of the root of S_j. Note that $depth(T) = \max\{depth(S_j),\ depth(S_k) + 1\}$. Since $depth(S_k) + 1 \leq depth(S_j)$, $depth(T) = depth(S_j)$. By induction hypothesis, $depth(S_j) \leq \lfloor \log m \rfloor + 1 \leq \lfloor \log n \rfloor + 1$. Therefore, $depth(T) \leq \lfloor \log n \rfloor + 1$.

2. (10 pts.) The prim's algorithm works correctly when an arbitrary vertex is chosen as a start vertex. Assume that there is only one edge say e_1 whose weight is smaller than any other edges. Prove that e_1 is always included in a tree generated by the Prim's algorithm regardless which vertex is chosen as a start vertex.

Sol. Let T be a MST but does not include e_1. Consider $G' = T + \{e_1\}$. Note that G' now has a cycle C that includes e_1. We now define a new spanning tree $T' \subseteq G'$ such that $T' = G' - \{e'\}$ where e' is an arbitrary edge in C and $e' \neq e_1$. As $w(e') > w(e_1)$, $w(T') > w(T)$; hence, T' cannot be a MST. Therefore, e_1 must be included in any MST.

3. (10 pts.) We are given a list of n elements in non-decreasing order: $A : a_1 \leq a_2 \leq \cdots \leq a_n$, and we are also given a number p. Assume that the sum of the elements in A is at least p. We would like to compute a subset A' of A such that (i) the sum of the elements in A' is at least p and (ii) satisfying (i), the number of elements in A' is as small as possible. (For example, if $A = \{1, 2, 3, 5, 7, 8, 9\}$ and $p = 20$, then $A' = \{3, 8, 9\}$ is a solution.) Give an $O(\log n)$ time algorithm that solves this problem. **Note: You may do some preprocessing using $O(n)$ time.**

Sol. Note that this problem when the given list A is in an arbitrary order can be solved in $O(n)$ time as shown in #8 of Homework 1. A main idea in the proposed $O(n)$ time algorithm was the following:

- Suppose there exits a subset $T \subseteq A$ such that $|T| = k$ and $sum(T) \geq p$ where $sum(T)$ denotes the sum of the elements in a subset $T \subseteq A$. Then, a subset T containing the k largest elements of A must be such that $sum(T) \geq p$.

Before we describe algorithm, we do the following preprocessing, which takes $O(n)$ time.

Set $S(n+1) = 0$.
for $i = n$ to 1 do
    $S(i) = S(i+1) + a_n$
endfor

With $S(i)'s$ already computed, a solution $A' = \{a_j, a_{j_1}, \cdots, a_n\}$ can be found in $O(\log n)$ time using a binary search, where j is the largest index such that $S(j) \geq p$ but $S(j+1) < p$.

4. (10 pts.) Consider the following problem for n jobs, each one of which takes exactly one minute to complete. At any time $T = 1, 2, 3, \cdots$, we can execute exactly one job. Each job i earns a profit of p_i dollars if and only if it is executed no later than time d_i, where d_i is given as an input. Assume that d_i is an integer value. The problem is to schedule the jobs to maximize the profit.

   Consider the following greedy strategy: At each time T, pick the most-profitable one among jobs whose deadline is T or later. For example, consider $n = 4$, profits $P = (50, 10, 15, 30)$ and deadlines $D = (2, 1, 1, 2)$. This greedy strategy will yield the following solution: job 1 is first picked, and then job 4, resulting in the total profit \$80.

   Prove or disprove this greedy algorithm finds an optimal solution.

   Sol. See the following example to disprove: $P = (50, 10, 15, 30)$ and deadlines $D = (2, 2, 1, 1)$. The greedy strategy will pick job 1 first, and then pick job 2, resulting in total profit 60. However, an optimal solution is to first pick job 4 and then pick job 1, resulting in total profit 80.

   HW3, Q1

5. (5 pts each) Consider the following graph G. Show all your work for each of the following problems.

(a) Find a minimum spanning tree of G obtained using the Prim's algorithm with 3 as the start node.

(b) Find a minimum spanning tree of G obtained using the Kruskal's algorithm.

(c) Apply the Dikstra's shortest path algorithm and find a shortest path in G from vertex 5 to every other vertex. Note that each edge $e = (u, v)$ may be used in each direction in your path, i.e., you may assume two directed edges (u, v) and (v, u) exist in your graph with $w(u, v) = w(v, u) = w(e)$.

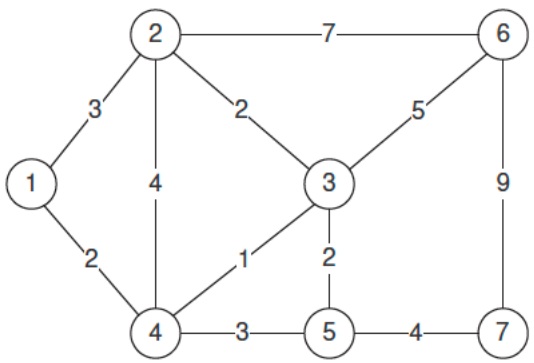


Figure 1: G

(a) Find a minimum spanning tree of G obtained using the Prim's algorithm with 3 as the start node.

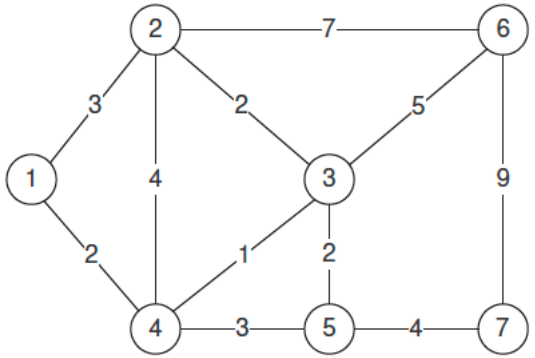


Figure 1: G

(b) Find a minimum spanning tree of G obtained using the Kruskal's algorithm.

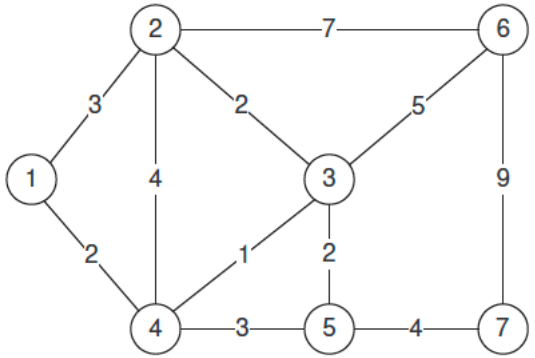


Figure 1: G

(c) Apply the Dikstra's shortest path algorithm and find a shortest path in G from vertex 5 to every other vertex. Note that each edge $e = (u, v)$ may be used in each direction in your path, i.e., you may assume two directed edges (u, v) and (v, u) exist in your graph with $w(u, v) = w(v, u) = w(e)$.

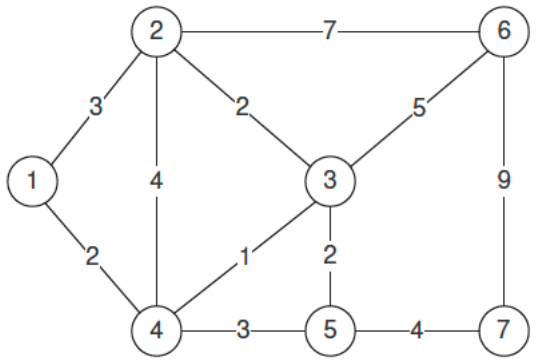


Figure 1: G

7. (5 pts.) Construct the Huffman codes for seven characters $a_1, \cdots, a_7$ with relative frequencies $(q_1, \cdots, q_7) = (1, 3, 3, 6, 7, 9, 9)$.

Sol. $q_1 = 0000$, $q_2 = 0001$, $q_3 = 001$, $q_4 = 100$, $q_5 = 101$, $q_6 = 01$, $q_7 = 11$.

8. (10 pts.) Consider a sequence of numbers $A = (\mathbf{5}, 5, 6, 5, 1, 9, 5, 4, 5, 8, 2, 5, 9, 4)$. Show how *Partition* algorithm works for A assuming that the first element **5** is the pivot. Upon completion of the algorithm, the numbers in A should be partitioned into two group A_1 and A_2 such that $A_1 = \{a \in A \mid a < 5\}$ and and $A_2 = \{a \in A \mid a \geq 5\}$. Show all your work.

9. (10 pts.) You are given an array A of n positive integers in an arbitrary order and an arbitrary integer k, where $1 \leq k \leq n$. Design an algorithm that outputs k smallest odd integers. If the number of odd integers in A is less than k, you should report all odd integers. For example, if $A = [2, 17, 3, 10, 28, 5, 9, 4, 12, 13, 7]$ and $k = 3$, your output should be $3, 5, 9$.

Note: Your algorithm should run in $O(n)$ time, i.e., $O(kn)$ is not acceptable.

Sol. Let $A' = \{a \in A \mid a \text{ is odd.}\}$ Find the kth smallest element, say x, of A' using the linear time Select algorithm. Then report $S = \{a \in A' \mid a \leq x\}$.

So for the given example $A = [2, 17, 3, 10, 28, 5, 9, 4, 12, 13, 7]$, $A' = \{17, 3, 5, 9, 13, 7\}$. The 3rd smallest element in A' is 9. We report $S = \{3, 5, 9\}$.

10. (10 pts.) A *forest* is defined to be a cycless graph, i.e., a graph that may have more than one component, but each component does not include a cycle (i.e., each component is a tree). Let $G(V, E)$ be a connected edge-weighted graph.

Given $V_0 = \{v_1, v_2, \cdots, v_k\} \subseteq V(G)$, design an algorithm to find a forest with exactly k components such that (i) each component contains exactly one of the vertices in V_0, (ii) each vertex of G is included in exactly one of k components, and (iii) satisfying (i) and (ii), the total edge weight over all k components is minimized.

Sol. (1) We construct an edge-weighted graph G' from G such that $V(G') = V(G) \cup \{v_0\}$ and $E(G') = E(G) \cup \{(v_0, v_i) \mid 1 \leq i \leq k\}$ where $w(v_0, v_i) = \epsilon$ for $1 \leq i \leq k$. (2) Then, find a MST T of G'. We then note that $\{(v_0, v_1), (v_0, v_2), \cdots, (v_0, v_k)\} \subseteq E(T)$. (3) Now, we remove k edges $(v_0, v_1), (v_0, v_2), \cdots, (v_0, v_k)$ from T, resulting in a forest of k components, a desired solution.

2021 Midterm Exam:

1. (5 pts. each)

(a) Using the definitions of O (big-Oh) and Ω (Omega) notations, prove that:

$$\text{If } g(n) = \Omega((f(n)) \text{ and } g(n) = O(h(n)), \text{ then } h(n) = \Omega(f(n)).$$

Sol. Since $g(n) = \Omega((f(n))$ and $g(n) = O(h(n))$, there must be constants c_1, c_2, n_1, and n_2 such that $g(n) \geq c_1 f(n)$ for any $n \geq n_1$ and $g(n) \leq c_2 h(n)$ for any $n \geq n_2$, which imply that $g(n) \geq c_1 f(n)$ for any $n \geq n_0$ and $h(n) \geq \frac{1}{c_2} g(n)$ for any $n \geq n_0$, where $n_0 = \max\{n_1, n_2\}$. Note that $h(n) \geq \frac{1}{c_2} g(n)$, when $g(n)$ is substituted by $g(n) \geq c_1 f(n)$, implies $h(n) \geq \frac{1}{c_2} c_1 f(n)$. Hence, we have $h(n) \geq cf(n)$, where $c = \frac{c_1}{c_2}$, for any $n \geq n_0$. Therefore, $h(n) = \Omega(f(n)$.

(b) Prove $\sqrt{n} \neq O(\log_2 n)$.

Sol. Suppose $\sqrt{n} = O(\log_2 n)$. We then have $\sqrt{n} \leq c \log_2 n$ for any $n \geq n_0$, where c and n_0 are some constants, which implies that $\frac{\sqrt{n}}{\log_2 n} \leq c$ for any $n \geq n_0$, i..e,

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n} \leq c$$

However, by the L'Hospital's rule, $\lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n} = \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{1}{n \log_e 2}}$.

Since $\frac{\frac{1}{2\sqrt{n}}}{\frac{1}{n \log_e 2}} = \frac{\sqrt{n} \log_e 2}{2}$ and $\lim_{n\to\infty} \frac{\sqrt{n} \log_e 2}{2} = \infty$,

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n} = \infty,$$

a contradiction. Therefore, $\sqrt{n} \neq O(\log_2 n)$.

2. (10 pts) You have n coins such that $n-2$ coins are of the same weight and two coins weigh more than these $n-2$ coins. Two heavier coins may be of the same weight, or their weights may be different. You have a balance scale: you can put any number of coins on each side of the scale at one time, and it will tell you if the two sides weigh the same, or which side is lighter if they do not weigh the same.
Give an algorithm for finding the two heavier coins using $O(\log n)$ weighings. Note that n is an arbitrary integer and your algorithm should consider all possible cases of n such as n is odd, even, or any other conditions.

Sol. We will first address the case when only one coin is heavier than the others in a certain group. (This is what we discussed in class.)

(A) Assume only one coin is heavier than the others. In this case, if n is even, we divided them into two groups with each $\frac{n}{2}$ coins where one group must weigh more than the other, and the group the heavier coin belongs to is identified. If n is odd, we set aside an arbitrary coin and divide $n-1$ coins into two groups with each $\frac{n-1}{2}$ coins. If two groups weigh same, the coin not in any of two groups is identified as the heavier one. Otherwise, the group where the heavier coin belong to is identified.

(B) Now, assume that two coins are heavier than the other $n-2$ coins where two heavier coins may have same or different weights. We consider two cases.

(B1) Suppose n is even. We then divide them into two groups, each with $\frac{n}{2}$ coins. If two groups weigh same, it must be that two heavier coins have the same weight and each belongs to each group. So we can identify the heavier one from each group using the process discussed in (A).

Now, assume that one group, say g_1, weighs more than the other group, say g_2. Then, there are two possible cases: (i) g_1 include both heavier coins, and (ii) two heavier coins have different weights such that the heaviest one belongs to g_1 and the second heaviest one belongs to g_2. For case (i), the problem can be recursively solved with a smaller size of input by following process in (B). For case (ii), the problem can be solved using the process discussed in (A). So we need to identify which case holds, and it can be done as follows.

Take g_2 and divide them into two groups of same size (set aside one coin if g_2 has an odd number of coins), and weigh them. If one group weighs more, then it is clear that g_2 also include a heavier coin, i.e., case (ii) holds. If two groups weigh same and the number of coins in g_2 is even, then case (i) holds. If two groups weigh same and the number of coins in g_2 is odd, we can also easily identify whether the coin set aside weighs more, which is case (ii) or same with other coins which is case (i).

(B2) Suppose n is odd. In this case, by setting aside one coin, we can divide $n-1$ coins into two groups of same size and identify two heavier coins by following the process discussed in (B1).

To analyze the time complexity, we note that the search space is reduced by a half after each iteration; hence, the whole process can be done in $O(\log n)$ time.

3. (5 pts. each) Let A be a list of m elements and B be a list of n elements. Given an integer k, $1 \leq k \leq \min(m, n)$, give an algorithm to find the kth smallest element in $A \cup B$.

(a) Each list in A and B is in an arbitrary order. Your algorithm must run in $O(\max(m, n))$ time.

Sol. First, we simply merge (or concatenate) A and B into a single file, which is in an arbitrary order, in $O(n+m)$ time and then apply the linear-time Select algorithm to find the kth smallest element in $O(m+n)$ time. Since $O(n+m) = O(\max(m, n))$, the whole process can be done in $O(\max(m, n))$ time.

(b) Each list in A and B is in a non-decreasing order. Your algorithm must run in $O(\log(\max(m, n)))$ time.

Sol. First note that we only need to keep only the first k elements of A and the first k elements of B, and discard the remaining elements. So we can assume that $|A| = |B| = k$, and the objective is to find the kth smallest element, i.e., the median of $A \cup B$. Note that this problem was discussed with solutions in #2(a) of *Binary Search Variations of Practice Problem Set.*

To explain the main idea of the algorithm, let's first assume k is even and let $A(\frac{k}{2}) = a$ and $B(\frac{k}{2}) = b$. If $a = b$, we are done.

• Suppose $a < b$. We can then remove $A(1), \cdots, A(\frac{k}{2})$ and $B(\frac{k}{2} + 1), \cdots, B(k)$ for the next iteration. Note that we have deleted $\frac{k}{2}$ elements from A whose values are all less than the kth smallest element we are search for; hence, we set $k' = k - \frac{k}{2}$ and search for the k'th smallest element in the next iteration.

• Now, assume $b < a$. We can then remove $A(\frac{k}{2} + 1), \cdots, A(k)$ and $B(1), \cdots, B(\frac{k}{2})$ for the next iteration. It is also noted that we have deleted $\frac{k}{2}$ elements from B whose values are all less than the kth smallest element we are search for; hence, we set $k' = k - \frac{k}{2}$ and search for the k'th smallest element in the next iteration.

When k is odd, it can be solved in a similar manner, and details can be found in the Practice Problem set.

To analyze the time complexity, we note that the search space is reduced by a half after each iteration; hence, the whole process can be done in $O(\log k) = O(\log(\max(m, n)))$ time.

4. (5 pts each)

(a) Apply the Partition algorithm to $A = (4, 4, 8, 1, 3, 8, 10, 3, 4, 3, 3, 1, 4, 2)$ using the first element of A as the pivot element. Upon completion, all elements equal to the pivot element should be placed in the left of the pivot element.
(Show all your work.)

(b) Apply the Select algorithm and find the kth smallest element of L where

$L = \{1, 3, 2, 25, 24, 7, 8, 9, 6, 15, 13, 15, 11, 22, 11, 5, 24, 22, 21, 23, 17, 29, 16, 22, 18, 26, 22, 13, 29\}$

and $k = 21$.

(Show all your work.)

5. (10 pts.) The Quicksort algorithm outlined below runs in $O(n^2)$ time in the worst case where the initial call is $Quicksort(1, n)$.

```
Quicksort(m, p)
    if m < p then
        {
        j ← p + 1
        Partition(m, j)  /* Upon completion of Partition, the pivot element is placed at index j. */
        Quicksort(m, j − 1)
        Quicksort(j + 1, p)
        }
endQuicksort
```

Discuss how the worst-case time complexity can be improved to $O(n \log n)$.

Sol. We can use the median as the pivot element for the Partition process to partition the list into two equal size sublists. Finding the median can bd done in $O(n)$ time using the linear-time Select algorithm, and the number of iterations of Quicksort algorithm is bounded by $O(\log n)$; hence, the Quicksort runs in $O(n \log n)$ time in the worst-case.

6 (10 pts.) Consider the problem of **Job sequencing with deadlines** discussed in class (also, presented in the textbook).

Assume that jobs are labeled with $1, \cdots n$ such that $p_1 \geq p_2 \geq \cdots \geq p_n$, i.e., jobs are given as a sorted list according to their profits. An optimal algorithm outlined below can be implemented using the Union-Find data structure to run in $O(n \log n)$ time in the worst case.

Procedure JS
    $J = \emptyset$
    for $i = 1$ **to** n **do**
        if all jobs in $J \cup \{i\}$ can be completed by their deadlines, then $J = J \cup \{i\}$
    endfor
    Return(J)
*end*JS

Find an optimal solution by applying the JS algorithm to the following example:

$n = 6$: profits $P = (10, 9, 8, 7, 4, 3)$ and deadlines $D = (5, 8, 2, 10, 1, 2)$.

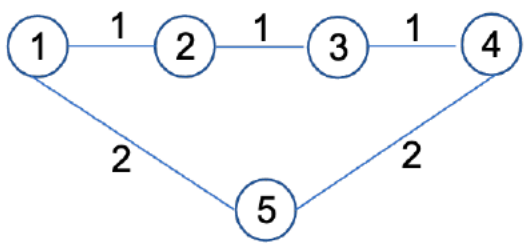
You should show details of all steps that include how the Union-Find data structure can be defined/applied to demonstrate $O(n \log n)$ running time in the worst case.

$n = 6$: profits $P = (10, 9, 8, 7, 4, 3)$ and deadlines $D = (5, 8, 2, 10, 1, 2)$.

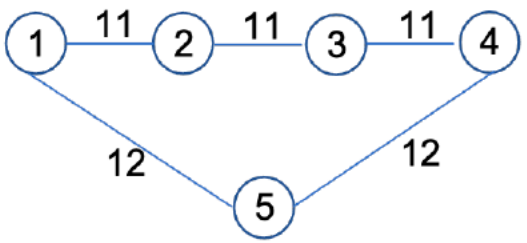
7. (10 pts.) Let G be an edge-weighted graph with $w(e)$ denoting the weight of edge $e \in E(G)$. Now, consider a new weight function $w'(e)$ such that $w'(e) = w(e) + c$ for each $e \in E(G)$, where c is a positive constant.

   Prove or disprove that if a path connecting two nodes, say s and t, is a shortest path using the old weight function w, it is also a shortest path connecting s and t when the new weight function w' is used.

Sol. See G in (a) where 1-2-3-4 is a shortest path from 1 to 4. By defining $w'(e) = w(e) + 10$ for each $e \in E(G)$ as shown in (b), a shortest path from 1 to 4 is 1-5-4, not 1-2-3-4.



(a) G with $w(e)$



(b) G with $w'(e) = w(e) + 9$

Figure 1:

8. (10 pts.) Consider the following variation of the fractional Knapsack problem:

We are given a set of n objects $S = \{1, \cdots, n\}$, a knapsack with capacity M, a weight function $W = (w_1, \cdots, w_n)$, a profit function $P = (p_1, \cdots, p_n)$, a subset $A \subseteq S$, and a subset $B \subseteq S$. The objective is to compute $X = (x_1, \cdots, x_n)$ such that

(i) $0 \leq x_i \leq 1/2$ for each $i \in A$,

(ii) $\frac{1}{2} \leq x_i \leq 1$ for each $i \in B$,

(iii) $0 \leq x_i \leq 1$ for each $i \in S - A \cup B$,

(iv) $\sum_{i=1}^{n} x_i w_i \leq M$, and

(v) $\sum_{i=1}^{n} x_i p_i$ is maximized satisfying (i)-(iv).

If no feasible solution exists, you should report so.

Give an algorithm that solves the problem, i.e., that finds an optimal solution. You do not need to prove the optimality of your algorithm.

(i) $0 \le x_i \le 1/2$ for each $i \in A$,

(ii) $\frac{1}{2} \le x_i \le 1$ for each $i \in B$,

(iii) $0 \le x_i \le 1$ for each $i \in S - A \cup B$,

(iv) $\sum_{i=1}^{n} x_i w_i \le M$, and

(v) $\sum_{i=1}^{n} x_i p_i$ is maximized satisfying (i)-(iv).

Sol. If $\sum_{i \in B} \frac{w_i}{2} > M$, then report no feasible solution exists. Otherwise, do the following.

(i) Let $M = M - \sum_{i \in B} \frac{w_i}{2}$. (We initially assign $x_i = \frac{1}{2}$ for each $i \in B$.)

(ii) We sort items such that $\frac{p_1}{w_1} \ge \cdots \ge \frac{p_n}{w_n}$ and consider items in this order.

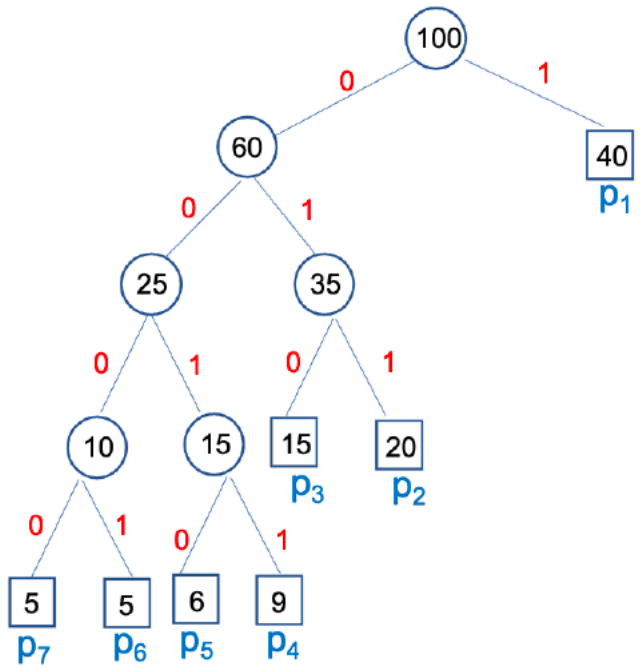
(iii) Add item i to the knapsack as much as we can such that

- $x_i \le \frac{1}{2}$ if item $i \in A \cup B$, and
- $x_i \le 1$ otherwise.

(For $i \in B$, we have already added $\frac{w_i}{2}$ in the knapsack; so we can add at most $\frac{w_i}{2}$ more in the knapsack.)

9. (5 pts.) Design a Huffman code for the following 7 letters $a_1, \cdots, a_7$ such that a_i appears $p_i\%$ of the times where $p_1 = 40$, $p_2 = 20$, $p_3 = 15$, $p_4 = 9$, $p_5 = 6$, $p_6 = 5$, and $p_7 = 5$.

Sol. $P_1 : 1$, $P_2 : 011$, $P_3 : 010$, $P_4 : 0011$, $P_5 : 0010$, $P_6 : 0001$, $P_7 : 0000$.

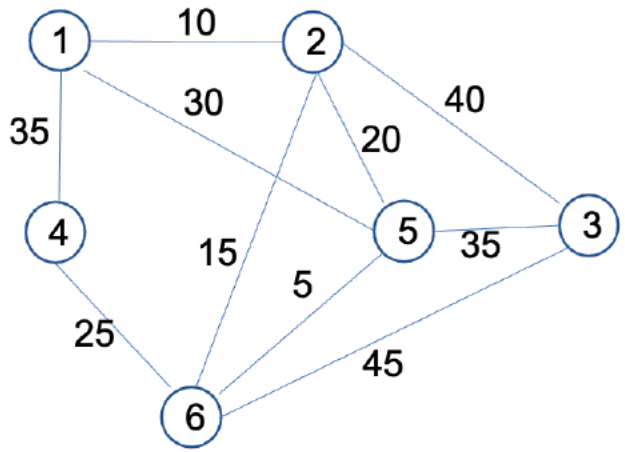


10. (5 pts. each) Consider graphs G_1 and G_2 shown in Figure 3.

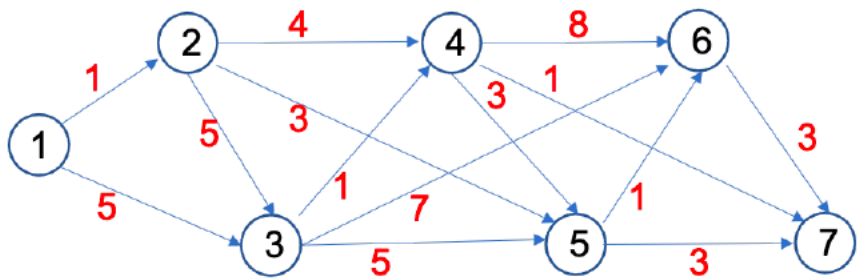
(a) Find a minimum spanning tree of G_1 using the Prim's algorithm. You should show details of all steps to demonstrate $O(n^2)$ running time, where n is the number of vertices.

(b) Find a minimum spanning tree of G_1 using the Kruskal's algorithm. You should show details of all steps to demonstrate $(|E| \log n)$ running time where $|E|$ denotes the number of edges and n denotes the number of vertices. You may use the Union-Find data structure to obtain the desired running time.

(c) Apply the Dijkstra's shortest path algorithm to G_2 and find shortest paths from node 1 to all other node. You should show details of all steps to demonstrate $O(n^2)$ running time where n is the number of vertices.



(a) G_1



(b) G_2