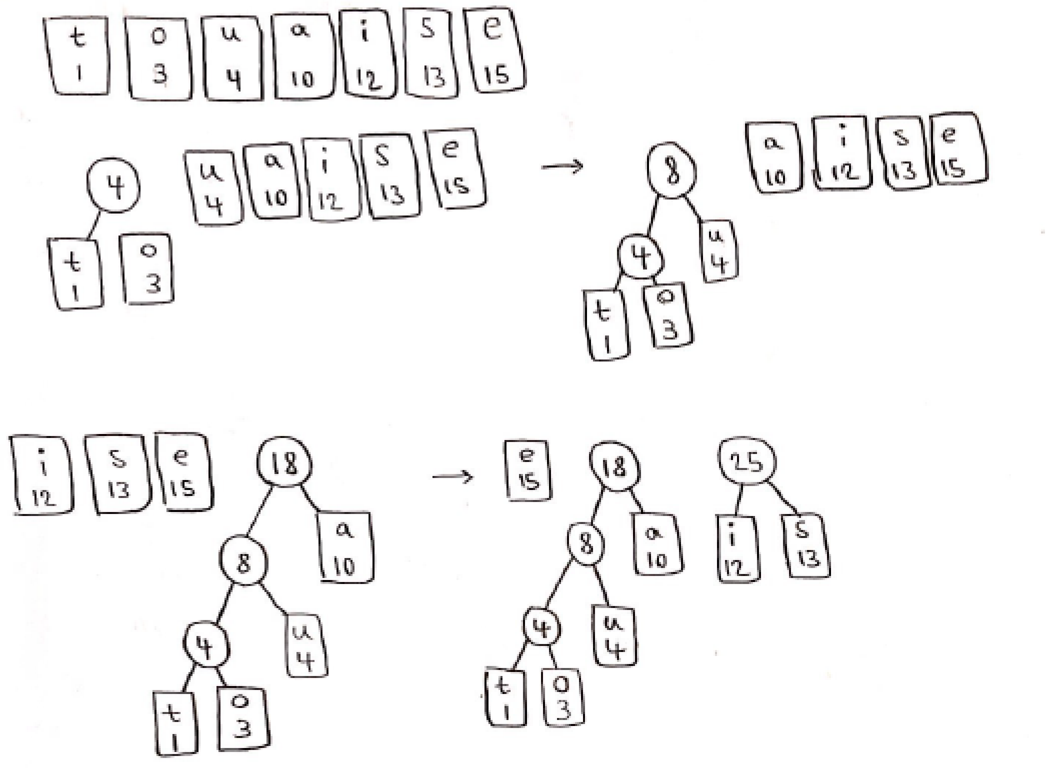
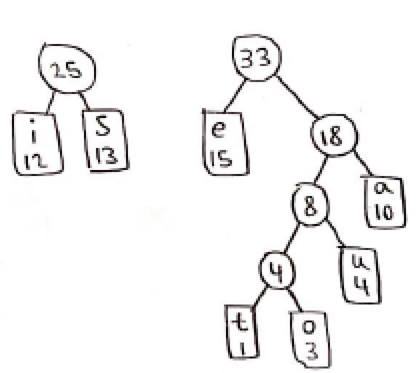
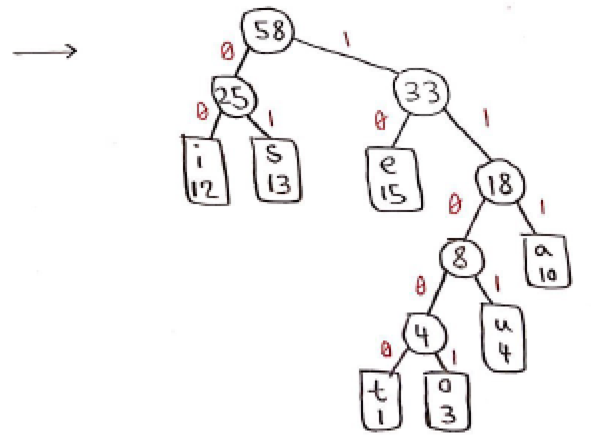
Algorithm Design 3992
June 2021
Instructor : Dr.Shafiei

1.





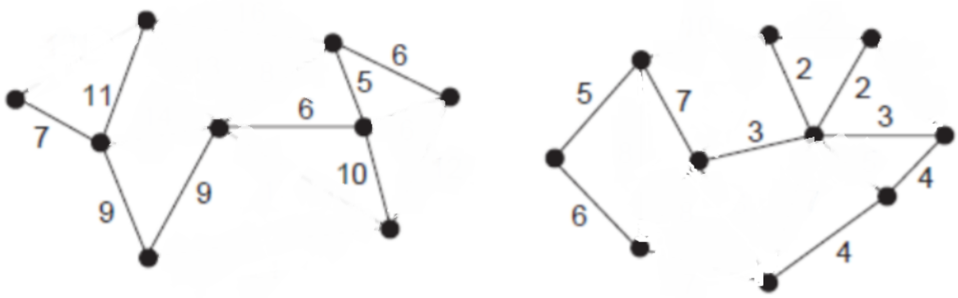


i = 00
s = 01
e = 10
a = 111
u = 1101

t = 11000
o = 11001

Algorithm Design 3992
June 2021
Instructor : Dr.Shafiei

2.



3.

```
int findMinPlatforms(Trains arrival, Trains departure)        // no-ref, no-
const
{
    // sort arrival time of trains
    sort(arrival.begin(), arrival.end());

    // sort departure time of trains
    sort(departure.begin(), departure.end());

    // maintains the count of trains
    int count = 0;

    // stores minimum platforms needed
    int platforms = 0;

    // take two indices for arrival and departure time
    int i = 0, j = 0;

    // run till all trains have arrived
    while (i < arrival.size())
    {
        // if a train is scheduled to arrive next
        if (arrival[i] < departure[j])
        {
            // increase the count of trains and update minimum
            // platforms if required
            platforms = max(platforms, ++count);

            // move the pointer to the next arrival
            i++;
        }

        // if the train is scheduled to depart next i.e.
        // `departure[j] < arrival[i]`, decrease trains' count
        // and move pointer `j` to the next departure.

        // If two trains are arriving and departing simultaneously, i.e.
        // `arrival[i] == departure[j]`, depart the train first
        else {
            count--, j++;
        }
    }

    return platforms;
}
```

4.

```
int result[V];

// Assign the first color to first vertex
result[0]  = 0;

// Initialize remaining V-1 vertices as unassigned
for (int u = 1; u < V; u++)
    result[u] = -1;  // no color is assigned to u

// A temporary array to store the available colors. True
// value of available[cr] would mean that the color cr is
// assigned to one of its adjacent vertices
bool available[V];
for (int cr = 0; cr < V; cr++)
    available[cr] = false;

// Assign colors to remaining V-1 vertices
for (int u = 1; u < V; u++)
{
    // Process all adjacent vertices and flag their colors
    // as unavailable
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
        if (result[*i] != -1)
            available[result[*i]] = true;

    // Find the first available color
    int cr;
    for (cr = 0; cr < V; cr++)
        if (available[cr] == false)
            break;

    result[u] = cr; // Assign the found color

    // Reset the values back to false for the next iteration
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
        if (result[*i] != -1)
            available[result[*i]] = false;
}
```

5. **The Floyd** algorithm runs in **O(v^3). Kruskal** time complexity worst case is **O(E log E)**,this is because we need to sort the edges. **Prim** time complexity worst case is **O(E log V)** with **priority queue** or even better, **O(E+V log V)** with **Fibonacci Heap**. **Dijkstra** calculates the **shortest path tree** , so the result is not necessarily a minimum spanning tree, the algorithms compute different things(but if we implement it wit**h Fibonacci heap** priority queue , it gives **O(|V|log|V| + |E|)**). In conclusion , we should use Kruskal when the graph is sparse, i.e.small number of edges,like E=O(V),when the edges are already sorted or if we can sort them in linear time.
6. First , we sort jobs by their profit .

work	deadline	profit
3	3	60
7	1	55
6	1	45
1	2	40
4	2	20
2	4	15
5	3	10

Now we solve the problem using the Greedy method . Maximum deadline is 4 , so we have 4 slots of time that we can do the jobs within it .

0-------1-------2-------3-------4

Job Considered	Slot assigned	Solution	Profit
j3	[2,3]	j3	60
j7	[2,3] [0,1]	j3,j7	115
j6 (X)	[2,3] [0,1]	j3,j7	115
j1	[2,3][0,1][1,2]	j3,j7,j1	155
j4(X)	[2,3][0,1][1,2]	j3,j7,j1	155
j2	[2,3][0,1][1,2][3,4]	j3,j7,j1,j2	170
j5(X)	[2,3][0,1][1,2][3,4]	j3,j7,j1,j2	170

So we can do the set of { j3,j7,j1,j2 } and the maximum profit is 170 !