Tutorial-7

Prashar Pande
D-28, 2016906

Q1 Greedy Algo Paradigm :- Greedy is an algorithmic paradigm that builds up a solution piece by piece always choosing the next most promising piece..

Greedy Algo are simply Intuitive algorithm used for optimization (either maximize or minimized) problems.

Q2 (i) Activity Selection :-
Time Complexity = $O(n \log n)$ (If input activities may not be stored).
$= O(n)$ (When input activities are stored)

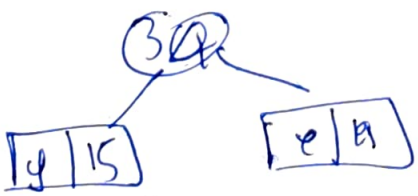
(ii) Job Sequencing => Time Complexity $= O(n \log(n))$.
Space Complexity $= O(n)$.

(iii) Fractional Knapsack => TC => $O(n \log n)$
SC => $O(1)$

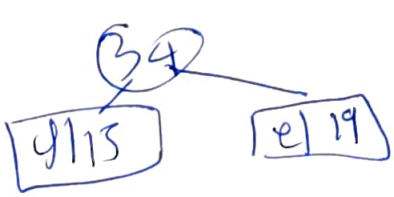
(iv) Huffmans Coding => TC => $O(n \log n)$
SC => $O(n)$.

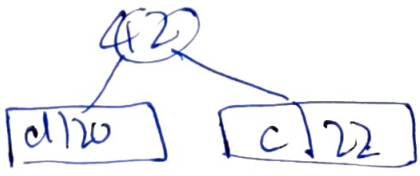
Q3 $a = 45$ $c = 22$ $e = 19$
$b = 23$ $d = 20$ $f = 15$

Step 1

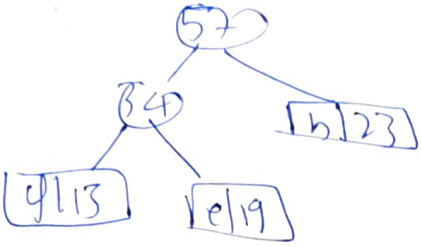


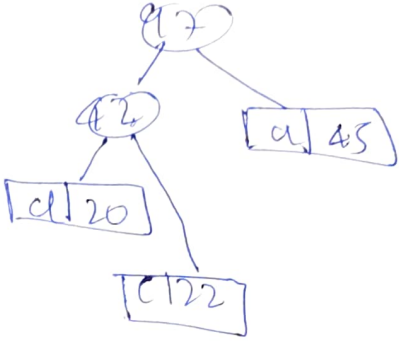
Step 2



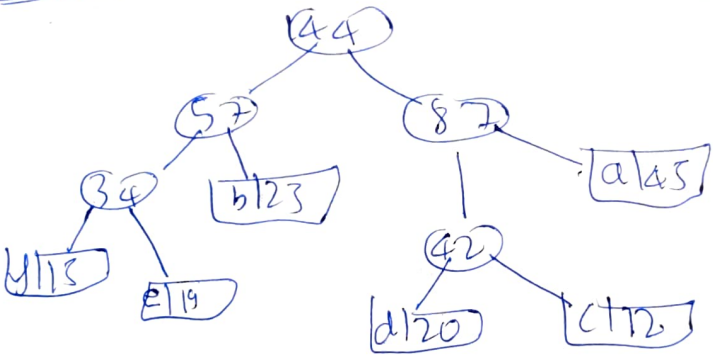


Step 3:-





Step-4:-



a = 11 c = 101 e = 001
b = 01 d = 100 y = 000

Q We can optimise the approach we use to solve the job sequencing problem by using priority queue (max heap)

Algo:

→ Sort the job based on deadlines.
→ Iterate from the end calculate the slot every 2 consecutive deadlines,
→ While the slot are available & there are jobs not left in the max heap.
→ Sort the result array based on their deadline.

Soln Disadvantages of Greedy Approch.

=> It is not suitable for problems where a solution is required for every subproblem. The greedy strategy can be wrong in most cases even when it is given optimal solution.

Ex: i) Dijkstra's Algorithm to find path with -ve edges.

ii) We can use greedy approch the problem by always going to the nearest possible city. We select any of the city as the 1st one.

iii) We can build a position of cities in a way that the greedy strategy, finds the worst possible solution.

Soln

Start Time	1	2	0	6	9	10
End Time	3	5	7	8	11	12

```
void Print maxActivities (Activity Arr[], int n)
{
    Sort (arr, arr+n, activity Compare).
    Cout<<" Following activities are selected";
    int i=0;
    Cout<<" ( "<<arr[i].start<<", "<<arr[i]
    for(int j=1, ; j<n; j++){
        if(arr[j].start >= arr[i].finish){
    Cout<<", "arr[j].start<<", "<<
                arr[j];
```

```
        finish << " , " ;
        i = j ;
    }
}

int main () {
    Activity arr[] = {{1,3}, {2,5}, {0,7}, {6,8}, {
                      {11,12}};
    int n = sizeof(arr)/sizeof(arr[0]);
    Print max Activity (arr, n);
      return 0;
}
```

Radha Parde
2016906