Unit : 4

Greedy Algorithm

Optimization Problem

→ Optimization problem is the problem of find the best solution from all feasible solution.

→ It involves in finding best solution among the set of possible choices, typically aming to maximize desired outcome.

→ An optimal solution can be defined as those choices in optimization problem that helps to achieve maximum desired outcome.

Greedy Strategy for Algorithm

→ Greedy Algorithms are type of heuristic approach for solving the optimization problem.

→ In Greedy algorithm,

we choose the locally optimal choices at each step, hoping that it eventually lead to globally optimal solution.

Some of the element of Greedy algorithm are

a) Greedy choice property

→ At each step, the algorithm choose the best choice available at that moment, based on current state of problem.

b) Optimal substructure property

→ The optimal solution for the entire problem is constructed from the optimal solution of sub-problem.

example of Greedy Algorithm are

- fractional knapsack
- Job Sequencing
- Prims Algorithm

Greedy Algorithm

Fractional KnapSack / Greedy KnapSack

Algorithm:-

```
fractionalKnapSack( W, n)
{
  for( i=1; i<=n; i++)
     x[i] = 0;
  tempW = W;

  for (i=1; i<=n; i++)
  {
     If ( W[i] > tempW)
       {
        break;
       }
     x[i] = 1.0;
     tempW = tempW - W[i] ;
  }

  If(i≤n)
  {
  x[i] = tempW/W[i];
  }

}
```

Time Complexity

Since, we can observe that there is no nested loop. Thus the algorithm runs for $O(n)$ time complexity.
However,
 We need to sort the items respetive to there weight to value ratio, which takes $O(n \log n)$ time complexity.

Thus, by including the sorting,
 The above algorithm complexity become.

$$T(n) = O(n \log n)$$

Theory about Fractional knapsack

A thief has a bag containing weight W. There are 'n' items with respective weight w_i and value v_i. Any amount of weight can be put in bag.

x_i fraction of item can be collected. $0 \le x_i \le 1$.
ie.
If $x_i = 1$, then we take that whole item into bag.

Here, the items are arranged in decreasing order by v_i/w_i ratio.

classmate PAGE

Job Sequencing with Deadlines

```
JobSequential (d[], j[], n)
{
    for(i=1 ; i<=n ; i++)
    {
        j[i]=0;
    }
    for(i=1 ; i<=n ; i++)
    {
        d = d[i];
        for( k=d ; k>=0 ; k--)
        {
            If ( j[k]==0)
            {
                j[k] = i;
                break ;
            }
        }
    }
}
```

Time Complexity :-

In this algorithm, we are using nested loops.
Hence,

$T(n) = O(n^2)$

Kruskal's Algorithm

Algorithm :-

KruskalMST (G)

{

$T = \{V\}$ // nodes

O(E log E) → E = set of edges sorted in non-decreasing order of weight

O(n) ← while (|T| < n-1 && E != ϕ)

{

select (u,v) from E in order

remove (u,v) from E

if (u,v) doesn't create cycle in T

then, $T = T \cup \{(u,v)\}$

}

}

Time Complexity :-

Creation of set of Edges takes O(E log E).
The while loop runs for O(n) times and the step inside while loop take almost linear time.

So, total time taken is given by

$T(n) = O(E \log E)$ or $O(E \log V)$.

Prims Algorithm

Algorithm:-

```
PrimMST(G)
  {
   T = ∅ ;
   S = {s}      // s is randomly chosen vertex

o(V) <-  while (S != V)
    {
o(E) <- e = (U, V) an edge of mimimum weight that is adjacent
           to vertices in T that doesn't form cycle
    T = T ∪ {(U, V)}
    S = S ∪ {V}
    }
  }
}
```

Time Complexity :-

while loop runs for $O(V)$.
The edge of minimum weight incident on a vertex can be found in $O(E)$.
So,
Total time is $O(EV)$

If, we use priority queue, then, running time of prim's algorithm is $O(E \log V)$.

Dijkstra's Algorithm

Algorithm :-

Dijkstra (G, w, s)
{
for each vertex $v \in V$
$d[v] = \infty$

$d[s] = 0$
$S = \emptyset$
$Q = V$
While ($Q \,!= \emptyset$)
{
u = Take minimum from Q and delete it
$S = S \cup \{u\}$
For each vertex v adjacent to u
If $d[v] > d[u] + w(u,v)$ then
$d[v] = d[u] + w(u,v)$

}

}

Time Complexity :-

1st loop takes $O(V)$ time.
Initiliz ation of priority queue take $O(V)$ time
Here,
While loop runs for $O(V)$,
where
each execution block inside while loop takes
$O(V)$ times,

Hence,
Total time complexity is $O(V^2)$

Huffman Coding:

It is a lossless data compression technique.

→ We assign variable length code to input character,
→ where length depends on the frequency of that input character.

Algorithm:-

① Create a leaf node for each unique character and build a min heap of all leaf nodes.

② Extract the two nodes with the minimum frequency from min heap.

③ Create new interior node with the frequency equal to sum of frequency of two nodes. Make 1st extracted node of left child and other extracted node as right child.

④ Repeate step ③ and ② until the heap contains only one node. The remaning node is the root node and tree is complete.

Time Complexity:-

We need to sort the input symbol in ascending order thus sorting takes $O(n \log n)$ time.

Thus, $T(n) = O(n \log n)$.

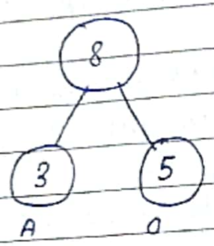
Example

Character	frequency
A	03
T	07
E	10
O	05

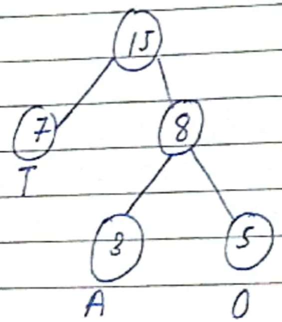
Soln

Now,

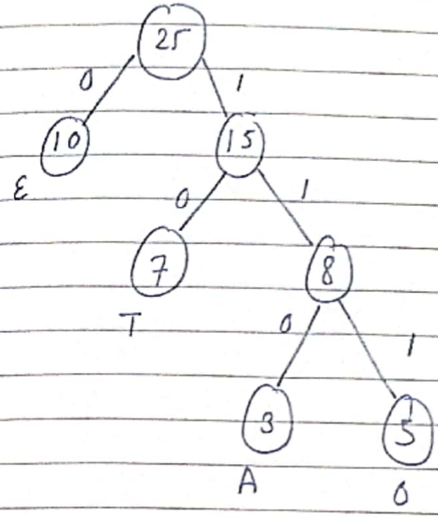
Taking the least minimum frequency and summing those frequency



Again,



again,



Now, assiging 0 to left side of node and 1 to right side of each node.

Thus,

character	frequency	code
A	03	110
O	05	111
T	07	01
E	10	0