Greedy Algorithms | Set 1 (Activity Selection Problem) - GeeksforGeeks

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems. An optimization problem can be solved using Greedy if the problem has the following property: *At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem*.
If a Greedy Algorithm can solve a problem, then it generally becomes the best method to solve that problem as the Greedy algorithms are in general more efficient than other techniques like Dynamic Programming. But Greedy algorithms cannot always be applied. For example, Fractional Knapsack problem (See this) can be solved using Greedy, but 0-1 Knapsack cannot be solved using Greedy.

Following are some standard algorithms that are Greedy algorithms.
1) Kruskal's Minimum Spanning Tree (MST): In Kruskal's algorithm, we create a MST by picking edges one by one. The Greedy Choice is to pick the smallest weight edge that doesn't cause a cycle in the MST constructed so far.
2) Prim's Minimum Spanning Tree: In Prim's algorithm also, we create a MST by picking edges one by one. We maintain two sets: set of the vertices already included in MST and the set of the vertices not yet included. The Greedy Choice is to pick the smallest weight edge that connects the two sets.
3) Dijkstra's Shortest Path: The Dijkstra's algorithm is very similar to Prim's algorithm. The shortest path tree is built up, edge by edge. We maintain two sets: set of the vertices already included in the tree and the set of the vertices not yet included. The Greedy Choice is to pick the edge that connects the two sets and is on the smallest weight path from source to the set that contains not yet included vertices.
4) Huffman Coding: Huffman Coding is a loss-less compression technique. It assigns variable length bit codes to different characters. The Greedy Choice is to assign least bit length code to the most frequent character.

The greedy algorithms are sometimes also used to get an approximation for Hard optimization problems. For example, Traveling Salesman Problem is a NP Hard problem. A Greedy choice for this problem is to pick the nearest unvisited city from the current city at every step. This solutions doesn't always produce the best optimal solution, but can be used to get an approximate optimal solution.

Let us consider the Activity Selection problem as our first example of Greedy algorithms. Following is the problem statement.
You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.
Example:

```
Consider the following 6 activities.
     start[]  =  {1, 3, 0, 5, 8, 5};
     finish[] =  {2, 4, 6, 7, 9, 9};
The maximum set of activities that can be executed
by a single person is {0, 1, 3, 4}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

1) Sort the activities according to their finishing time
2) Select the first activity from the sorted array and print it.
3) Do following for remaining activities in the sorted array.
…….a) If the start time of this activity is greater than the finish time of previously selected activity then select this activity and print it.

In the following C implementation, it is assumed that the activities are already sorted according to their finish time.

- C++
- Python

```
#include<stdio.h>// Prints a maximum set of activities that can be done by a single// person, one at a time.//  n   -->  Total number of activities// s[] -->  An array that contains start time of all activities//  f[] -->  An array that contains finish time of all activities
void printMaxActivities(int s[], int f[], int n)
{
    int i, j;
    printf("Following activities are selected \n");
    // The first activity always gets selected
    i = 0;
    printf("%d ", i);
    // Consider rest of the activities
    for (j = 1; j < n; j++)
    {
        // If this activity has start time greater than or
        // equal to the finish time of previously selected
        // activity, then select it
        if (s[j] >= f[i])
```

```
        {
            printf ("%d ", j);
            i = j;
        }
    }
}// driver program to test above function
int main()
{
    int s[] =  {1, 3, 0, 5, 8, 5};
    int f[] =  {2, 4, 6, 7, 9, 9};
    int n = sizeof(s)/sizeof(s[0]);
    printMaxActivities(s, f, n);
    getchar();
    return 0;
}
```