Greedy Algorithms | Set 1 (Activity Selection Problem)

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}
```

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.

```
#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 finis
      // 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;
}
```

Output:

```
Following activities are selected
0 1 3 4
```

References:

Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
Algorithms by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani
http://en.wikipedia.org/wiki/Greedy_algorithm

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Amazon Interview | Set 59 (Off-campus for SDE-1)
- Amazon Interview | Set 58 (On-campus for Software Development Engineer)
- Amazon Interview | Set 57 (Off-Campus for SDE-1)
- Amazon Interview | Set 56 (Off-Campus)
- Microsoft Interview | Set 25 (On-campus for Internship)
- Count all possible paths from top left to bottom right of a mXn matrix
- How to compare two arrays in Java?
- DELL Interview | Set 1 (On-Campus)

Writing code in comment? Please use **ideone.com** and share the link here.

- *new*

  What is we also need to give the sequence number of the activities we have selected.We preprocess to sort the arrays but the sequence number should be on the basis of the initial array

- *RAJAT*

  WHAT IF THE FINISHING TIME OF THE FIRST JOB WAS NOT 2 ,BUT SAY 8..THEN IN THIS CASE IT WOULD NOT HAVE GIVEN THE CORRECT ANS

  IT GIVES 0 4 BUT THE BEST CASE CAN BE THAT OF 1 3 4

  - *Sree Harsha Konduri*

    The finish list is to be sorted first. So when finish time of the first job is 8 it will be placed between activities 5-7 and 8-9.

- *arun*

Assuming that a person can work on only a single task at a time how can the following set be a solution
{0, 1, 3, 4}
The start time of 1 is before the end time of 0 which is 6.

- *Aadi*

  here 0 means the 0th job which is the first job above whose starting time is 1.

- *Dasharath*

  if the input is
  int s[] = {1, 3, 0, 5, 8, 7, 9};
  int f[] = {2, 4, 6, 7, 9, 9, 10};

  This algorithm gives (0,1,3,4) as output. The output should rather be (0,1,3,5,6)

  - *kartik*

    I think the algorithm will give {0, 1, 3, 4, 6} which is also a correct answer.

- *soha*

  How can greedy algorithms help in complex problems like shortest path problems?

- *Me*

  How can this algorithm be modified such that we need to choose maximum number of tasks?

  For eg.
  s[] = { 1, 3, 5, 7, 9}
  f[] = {10, 4, 6, 8, 10}

  As per this algorithm, only task 0 is printed in the output.
  But optimally, if we need to do the maximum number of tasks, output would be, 1, 2, 3, 4

  How can this algorithm be changed for this kind of modification?

  - *kartik*

    The implementation given in the post assumes that the input tasks are sorted according to their finish time.

    For unsorted inputs, you need to add a preprocessing step that first sorts the given tasks ccording to their finish time.

  - *agmafara*

    #include
    #include

```
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 = f[i]) {
printf("%d ", j);
i = j;
}
}
}

// SortArrays base on numbs Array param
// numbs[]=column or array to be used for sorting

void SortGreedyArrays(int numbs[], int numbs2[], int size) {
//bubble sorting
int a, b, t, t2;
for (a = 1; a = a; b--) {
if (numbs[b - 1] > numbs[b]) {
t = numbs[b - 1];
numbs[b - 1] = numbs[b];
numbs[b] = t;
t2 = numbs2[b - 1];
numbs2[b - 1] = numbs2[b];
numbs2[b] = t2;
}
}

}
}

// driver program to test above function

int main(int argc, char** argv) {

int s[] = {1, 3, 5, 7, 9};
int f[] = {10, 4, 6, 8, 10};

int n = sizeof(s) / sizeof(s[0]);

// sort base on f
SortGreedyArrays(f, s, n);
```

```
printMaxActivities(s, f, n);

getchar();
return 0;
}
```

0 1 2 4

- *agmafara*

Enhancement to last suggestion.
Your view?

```
#include
#include

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 = f[i]) {
printf("%d ", j);
i = j;
}
}
}

// SortArrays base on numbs Array param
// numbs[]=column or array to be used for sorting

void SortGreedyArrays(int numbs[], int numbs2[], int size) {
//bubble sorting
int a, b, t, t2;
for (a = 1; a = a; b–) {
if (numbs[b - 1] > numbs[b]) {
t = numbs[b - 1];
numbs[b - 1] = numbs[b];
numbs[b] = t;
t2 = numbs2[b - 1];
numbs2[b - 1] = numbs2[b];
numbs2[b] = t2;
}
```

```
}

}
}

void printSortedGreedyArray(int s[], int f[], int n) {
int a;
printf(" \n ");
for ( a = 0; a < n; a++) {

printf("Activity %d – %d,%d \n ", a , s[a] , f[a]);

}
printf(" \n ");

}

// driver program to test above function

int main(int argc, char** argv) {

int s[] = {1, 3, 5, 7, 9};
int f[] = {10, 4, 6, 8, 10};

int n = sizeof (s) / sizeof (s[0]);

// sort base on f
SortGreedyArrays(f, s, n);

printSortedGreedyArray(s,f,n);

printMaxActivities(s, f, n);

getchar();
return 0;
}
```

- *Sandeep*

  why only task 0 is printed. if you sort the finish times then your input becomes
  S[] = {3,5,7,1,9} ,
  f[] = {4,6,8,10,10} in which case your op will be 0,1,2,4

- *Sree Harsha Konduri*

  We need to preprocess the inputs before we can use the greedy algorithm of activity selection. The finish array needs to be sorted in increasing order to run this algorithm. Then it is intuitive to pick the activity with the smallest finish time which does not overlap with running activities.
  After preprocessing

s[] = { 3 5 7 1 9}
f[] = { 4 6 8 10 10}

- *Mad Coder*

  In your explanation of algorithm for activity selection problem,in 3rd point it is written that

  If the start time of this activity is less than the finish time of previously selected activity then select this activity and print it.

  It should rather be if start time of current activity is greater than the finish time of previous activity, then current activity should be selected otherwise both will overlap.

  - *GeeksforGeeks*

    @Mad Coder: Thanks for pointing this out. We have updated the post. Keep it up!

- *cracker*

  Another great article. Keep rocking GeeksforGeeks. Please post an article on Kruskal, how to implement it.

  - *PsychoCoder*

    @Cracker :

    http://geeksforgeeks.org/forum/topic/how-to-implement-kruskal-minimum-spanning-tree?replies=2#post-36795

    may be this can help you.

    - *http://www.facebook.com/profile.php?id=100003406021811 Rute*

      Any good search manektirg campaign is a multifaceted approach to reach on top of the rankings.a0Many SEO factors such as site structure, content relevance, and submission of press releases online will influence the position of your website / blog in search engine results.a0From 2011, the engagement is a key metric that can be measured by activity ona0social networksa0and how long visitors spend on your site / blog, here are some changes thata0Google Pandaa0brought.

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms

- Backtracking
- Pattern Searching
- Divide & Conquer
- Graph
- Mathematical Algorithms
- Recursion
- Java

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding "extern" keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

-
-  Subscribe

Recent Comments



-