## Greedy algorithm for activity scheduling (and applications)

**Contents**

1. **Problem statement**
2. **Input/output description**
3. **Ideas**
4. **Algorithm (Greedy algorithm)**
5. **Example**
6. **Proof of optimality**
7. **Implementation**
8. **Language details and differences**

**1. Problem statement:**

* You are given a list of programs to run on a single processor
* Each program has a start time and end time
* However the processor can only run one program at any given time
* There is no preemption - once a program is running, it must be completed
* Aim is to find the maximum subset of programs/tasks from the given list

Maximum subset – subset containing maximum number of elements

**2. Input/output description:**

* The program, in each implementation, takes its input from 2 matrices declared in the very beginning of the source code
* The matrices are startTime and endTime. Order of matrices must match.
* Ith element in each matrix signify the startTime and endTime of Ith task

**3. Ideas**

* Brute force: Examine every possible subset of tasks and find the largest subset of non-overlapping tasks
  + 2N  subsets for N element task list are to be analyzed for non-overlapping condition
  + The list thus obtained has to be further examined for maximum number of elements and this takes additional time
  + Optimal solution guaranteed but very high time order (O(>2N))
* One of the other alternatives is Greedy Algorithm
  + Does this give an optimal solution? – Yes
  + Proof can be found in the proof of optimality section

**4. Algorithm (Greedy approach)**

1. Sort the activities by their finish times
2. Add the first task to the final list of task that will be scheduled
3. Now in the remaining list, add a task to the final list if the startTime of a task is greater than the endTime of previous task

**5. Example**

* List of tasks given  
  A. ---  
  B. -----  
  C. ---   
  D. --  
  Note: Horizontal is time axis  
  The times of task are: (startTime, endTime)  
  A: (1, 4)   
  B: (3, 8)  
  C: (2, 5)  
  D: (5,7)
* Sorting according to endTime of tasks  
  A. ---  
  C. ---  
  D. --  
  B. -----  
  The times of sorted task are: (startTime, endTime)  
  A: (1, 4)   
  C: (2, 5)   
  D: (5,7)  
  B: (3, 8)
* Adding 1st task to final list and picking up non-overlapping elements from the rest of the list. ith in the final list if startTime of i > endTime of (i-1)  
    
  The new list of tasks thus obtained will be,   
  A. ---  
  D. --  
    
  The times of the tasks are: (startTime, endTime)  
  A: (1, 4)  
  D: (5,7)

**6. Proof of optimality**

**Method**

* Let A be set of activities selected by greedy algorithm
* Consider any non-overlapping set of activities B at random
* We will show that |A| >= |B| by showing that we can replace each activity in B with an activity in A
* This will show that A has at least as many activities as B. B is randomly chosen set of non-overlapping set so, this will be true for any such non-overlapping set and thus A will be optimal.

**Proof**

* Let A = a1, a2, a3,…, an, an+1,…  
  and B = a1, a2, a3,…, bn, bn+1,…
* That is an is the 1st activity in A that is different from B
* A is chosen using Greedy algorithm, which means that an has a finish time earlier than that of bn
  + Because in Greedy selection activities are arranged in increasing order of endTime
  + After an-1 Greedy algorithm gives an to final set means that an is the closest non-overlapping activity (in terms of endTime) to activity an-1
  + So, distance between any activity bn (non-overlapping with an-1) and an-1 has be greater than distance between an and an-1 in terms of endTime
  + This implies finish time of bn greater than finish time of an
* Consider B’ = B – {bn}U{an}  
  Thus B’ = a1, a2, a3,…, an, bn+1,… is also a valid set of scheduling, |B’| = |B|
* We now, continue this process on A, B’ and so on so forth
* As we can see in the previous process, each element in B can be replaced by an element in A
* Also, after replacing it is possible that A will be left out with few additional activities which implies |A| >= |B|

**7. Implementation**

* This algorithm is implemented in three languages – *Scilab, Python and C++*
* The details regarding the advantages of one implementation over other are provided below in the language differences section

**8. Language details and differences**

**Scilab:**

* As it was mandated, the initial implementation was done is Scilab. The sort algorithm used is a normal O(N2) sort
* As, this is not an OOP each activity is represented by the ith elements of two matrices
* Time order = O(N2)

**Python:**

* Each activity is represented as an ***object*** with startTime and endTime
* In-built sort of python is applied to the activity objects with respect to their endTime
* Time order = O(N\*logN), assuming in-built sort in python is O(N\*logN)

**Advantages**

* Greatly simplified code as objects are used
* Can make use of efficient in-built search algorithm

**C++:**

* Each activity represented as ith  element of two matrices – startTime and endTime
* Used Heap sort, O(N\*logN), for efficient sorting of activities
* Time order = O(N\*logN) fixed

**Advantages**

* Using heap sort => we can be more sure about the time order of our implementation
* C++ is much faster than the former two and with the implementation of most efficient sort algorithm this program runs significantly faster on huge data sets than the former two.