18CSC204J
DESIGN AND ANALYSIS OF ALGORITHM

TASK SCHEDULER

A PROJECT REPORT

Submitted by

Thrishalini Dwaraknath	RA2011030010141
Madavaram Reddy Vinathi	RA2011003010133
Devika Pullaikkodi Veetil	RA2011003010118

Under the guidance of

Kishore Anthuvan Sahayaraj K
(Assistant Professor, Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree of
BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

S.R.M. Nagar, Kattankulathur, Kancheepuram District

JULY 2022

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "TASK SCHEDULER Using Greedy Algorithm "is the Bonafide work of

Thrishalini Dwaraknath	RA2011030010229
Madavaram Reddy Vinathi	RA2011030010133
Devika Pullaikkodi Veetil	RA2011030010118

who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis ofwhich a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of the Internal Examiner

Signature of the External Examiner

TABLE OF CONTENTS

Contribution
Table

Thrishalini.D	RA2011003010141	Time Complexity Analysis
		Algorithm
		Conclusion
Madavaram Reddy Vinathi	RA2011030010133	Code testing and execution
		Abstract Information
		Final document
Devika Pullaikkodi Veetil	RA2011030010118	Research
		Definition and Objectives
		Analyzing Approach towards the problem

Introduction

A greedy algorithm is an approach for solving a problem by selecting the

best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result. The best thing about this algorithm is that it is easy to understand and implement. It works in a top-down approach. The algorithm never reverses the earlier decision even if the choice is wrong. This algorithm may not produce the best result for all the problems. It's because it always goes for the local best choice to produce the global best result.

Abstract

The Problem Statement

For an organizer, the main objective is to conduct as many events possible within the limited resources (say an auditorium) so as to make maximum profit out of it.

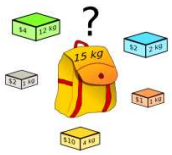
Why Greedy Approach?

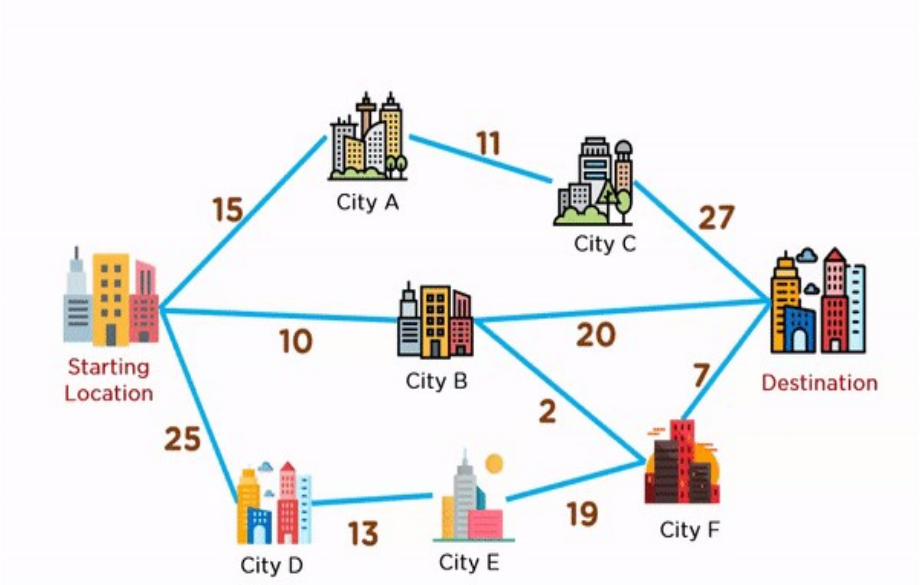
Since we want to maximize the count of activities that can be executed.This approach will greedily choose an activity with earliest finish time at every step, thus yielding an optimal solution.

Objectives

- It might not be possible to conduct all the events, since their timings can collapse.
- Two events, say **i** and **j**, are said to be non-conflicting or compatible if $startTime_i >= finishTime_j$ or $startTime_j >= finishTime_i$.
- The scheduler will return the maximum - event set from the given list.
- It is assumed that not more than one event is going to be conducted concurrently.

AN EXAMPLE
Here is another example. The objective is to fill the bag of some given volume with different materials such that the value of selected items is maximized.





Solving TASK SCHEDULER Problem

Greedy Algorithm:

An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seemsto provide an optimum solution is chosen.

Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions

Approach:

Working…

Following are the steps we will be following to proceed with task scheduler:
Step 1: Sort the given events in ascending order according to their finishing time.

Step 2: Select the first event from sorted array and add it to Solution array.

Step 3: Repeat steps 4 and 5 for the remaining events.

Step 4: If the start time of the currently selected event is greater than or equal to the finish time of previously selected event, then add it to the Solution array.

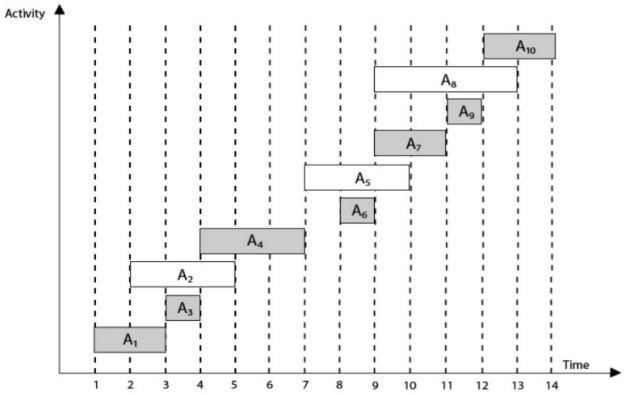
Step 5: Select the next event in sorted array.

Step 6: Print the Solution array.

- Now, schedule A1
- Next schedule A3 as A1.
- Next skip A2 as it is interfering.
- Next, schedule A4 as A1 A3 and A4 are non-interfering, then next, schedule A6.
- Next, schedule A7.

- Next, schedule A9
- Skip A8 as it is interfering.
- Next, schedule A10 as A1 A3 A4 A6 A7 A9 and A10 are non-interfering.

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Thus the final solution to the problem is:

Algorithm :

maxEvents (s, f,)
1. n ← length [s]
2. A ← {1}
3. j ← 1.
4. for i ← 2 to n
5. do if si ≥ fi
6. then A ← A ∪ {i}
7. j ← i
8. return A

Examp

le
Input:

1) Input : s[ ] = {1, 3, 0, 5, 8, 5};

f [ ] =
{2, 4,
6, 7, 9,
9};

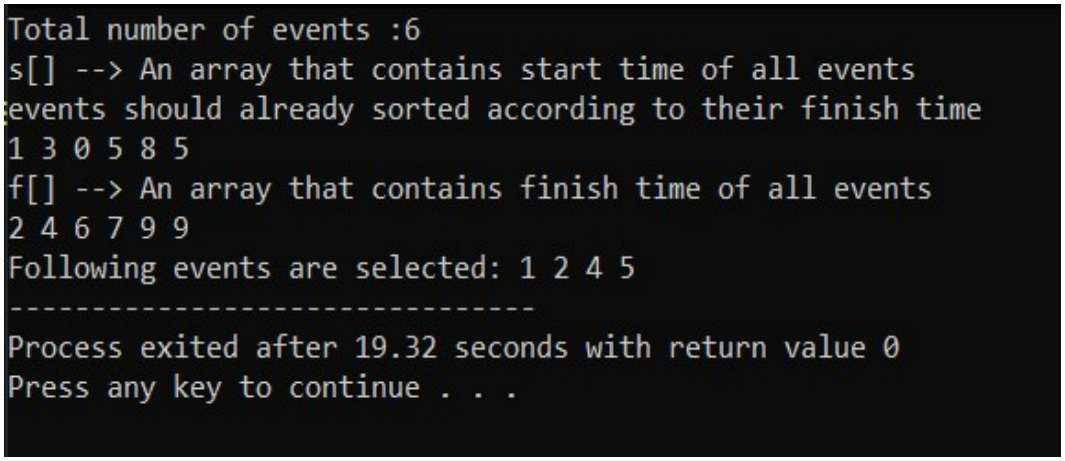
```c
#include<stdio.h>

void printMaxEvents(int s[], int f[], int n)
{
    int i, j;
    printf ("Following events are selected: ");
    i = 0;
    printf("%d ", i+1);
    for (j = 1; j < n; j++)
    {
    if (s[j] >= f[i])
    {
        printf ("%d ", j+1);
        i = j;
    }
    }
}

int main()
{
    int n,i;
    printf("Total number of events :");
    scanf("%d",&n);
    int s[n],f[n];
    printf("s[] --> An array that contains start time of all events\n");
    printf("events should already sorted according to their finish time\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &s[i]);
    }

    printf("f[] --> An array that contains finish time of all events\n");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &f[i]);
    }
    printMaxEvents(s, f, n);
    return 0;
}

//test case :
//Input = 1, 3, 0, 5, 8, 5  2, 4, 6, 7, 9, 9
//Output = 1,2,4,5
```

This is the result when we run the program.

Output : 1,2,4,5

2.

This is the result when we run the program.



CODE(IMPLEMENTATION IN C)

```
#include<stdio.h>
void printMaxEvents(int s[ ], int f[ ], int n)
{
        int i, j;
        printf ("Following events are selected n ");
        i = 0;
        printf("%d ", i);P
        for (j = 1; j < n; j++)
        {
        if (s[ j ] >= f[ i ])
        {
                printf ("%d ", j);
                i = j;
}
        }
}
```

```
int main()
{
  int n,i;
  printf("Total number of events :");
  scanf( "%d", &n);
  int s[ n ] ,f[ n ];
  printf("s[ ] --> An array that contains start time of all events\n");
  printf(" events should already sorted according to their finish time\n
  ");

for (i = 0; i < n; i++){
      scanf("%d", &f[ i ]);
   }
  printMaxEvents(s, f, n);
  return 0; }
```

OUTPUT:

```
Total number of events :6
s[] --> An array that contains start time of all events
events should already sorted according to their finish time
1 3 0 5 8 5
f[] --> An array that contains finish time of all events
2 4 6 7 9 9
Following events are selected: 1 2 4 5
--------------------------------
Process exited after 19.32 seconds with return value 0
Press any key to continue . . .
```

COMPLEXITY ANALYSIS:

- **Case 1:** When a given setof EVENTS are already SORTED according to their finishing time, then there is NO sorting mechanism involved, in such a case the complexity of the algorithm will be **O(n).**

- **Case 2:** When a given set of EVENTS is UNSORTED, then we will have to use the sort() method. The time complexity of this method will be **O(nlogn).**

s/e	frequency	total steps
1	1	1
1	1	1
0	-	-
1	1	1
1	1	1
1	1	1
1	1	1
1	n+1	n+1
0	-	-
1	n	n
0	-	-
1	n	n
1	n	n

s/e	frequency	Total steps
0	-	-
0	-	-
0	-	-
1	1	1
0	-	-
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
0	-	-
1	n+1	n+1
1	n	n
0	-	-
1	1	1
1	1	1

Conclusion:

Hence we have analysed the task scheduler problem using the greedy algorithm.

Using this greedy approach, we have arrived at decisions which are made from the given solution domain.

Also we have found out the time complexity for 2 cases mentioned above being O(n) and O(n logn) respectively.