SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University *u/s 3 of UGC Act, 1956)*

A Mini Project Report on

Activity Selection Problem

for the subject

Design and Analysis of Algorithms

By

Dikcha Singh
(RA2011027010096)
Roshana S V
(RA2011027010074)
Santhana Lakshmi
(RA2011027010129)

To subject in-charge

Mrs R. Radha

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

FACULTY OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR - 603203
Chengalpattu District

BONAFIDE CERTIFICATE

Register No RA2011027010074
RA2011027010096
RA2011027010129

Certified to be the bonafide record of work done by ROSHANA S V, DIKCHA SINGH, SANTHANA LAKSHMI K of DATA SCIENCE AND BUISINESS SYSTEM B.Tech Degree course in the Practical 18CSC204J - DAA in **SRM INSTITUTE OF SCIENCE & TECHNOLOGY**, Kattankulathur during the academic year 2022.

LAB INCHARGE

DATE: 20.06.22

HEAD OF THE DEPARTMENT

Submitted for University Examination held in _____,

in _____

SRM INSTITUTE OF SCIENCE & TECHNOLOGY, Kattankulathur.

DATE:

EXAMINER 1

EXAMINER 2

Name : ROSHANA S V
DIKCHA SINGH
SANTHANA LAKSHMI K

Class : W1

Reg. No. : RA2011027010074
RA2011027010096
RA2011027010129

Branch : DSBS (BDA)

INDEX

Expt. No.	Date of Performance	Title of Experiment	Page No.	Date of Submission	Marks	Signature
1.	18.06.22	Problem Definition	2	20.06.22		
2.	18.06.22	General Technique	2	20.06.22		
3.	18.06.22	Design Technique	3	20.06.22		
4.	18.06.22	Real life Applications	3	20.06.22		
5.	18.06.22	Explanation of algorithm	4	20.06.22		
6.	18.06.22	Steps of algorithm	5	20.06.22		
7.	18.06.22	Code	6	20.06.22		
8.	18.06.22	Output	8	20.06.22		
9.	18.06.22	Problem Explanation	9	20.06.22		
10.	18.06.22	Complexity Analysis	11	20.06.22		
11.	18.06.22	Conclusion	12	20.06.22		
12.	18.06.22	Reference	12	20.06.22		

CONTRIBUTION TABLE

STUDENT'S NAME	TOPICS COVERED
DIKCHA SINGH (RA2011027010096)	• Problem Definition • General Technique • Design Technique • Real-life applications
K. SANTHANA LAKSHMI (RA2011027010129)	• Explanation of algorithm • Steps of algorithm • Code • Output
ROSHANA S V (RA2011027010074)	• Problem explanation with diagram • Complexity Analysis • Conclusion • Reference

Activity Selection Problem

General Technique :-

The Activity Selection Problem is an optimization problem which deals with the selection of non-conflicting activities that needs to be executed by a single person or machine in a given time frame.

Each activity is marked by a start and finish time. Greedy technique is used for finding the solution since this is an optimization problem.

Problem Definition :-

Given are n activities with their start and finish times, the objective is to find solution set having maximum number of non conflicting activities that can be executed in a single time frame, assuming that only one person or machine is available for execution.

Some points to note here:

- It might not be possible to complete all the activities, since their timings can collapse.
- Two activities, say i and j, are said to be non-conflicting if $s_i >= f_j$ or $s_j >= f_i$ where s_i and s_j denote the starting time of activities i and j respectively, and f_i and f_j refer to the finishing time of the activities i and j respectively.

Design Technique :-

Our problem statement uses a greedy algorithm to get the best optimal solution. 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 algorithm never reserves the earlier decision even if the choice is wrong. It works in a top-down approach. Greedy approach can be used to find the solution 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.

Real-Life Applications :-

Following are some of the real-life applications of this problem :-

- Scheduling multiple competing events in a room, such that each event has its own start and end time.
- Scheduling manufacturing of multiple products on the same machine, such that each product has its own production timelines.
- Activity selection is one of the most well-known generic problems used in operations research for dealing with real-life buisness problems.
- In automated programs and the coordination of these programs is done with the activity scheduling problem with in turn helps in smooth completion of the automated task like in data analysis.

ALGORITHM:-

INPUT DATA

- act[] Array Containing all the activities.
- S[] array containing the starting time
- f[] array containing the finishing time.

OUTPUT DATA

- Sol[] array refering to the solution set containing the maximum number of non conflicting activities.

ALGORITHM STEPS:-

Step 1:- Sort the given activities in ascending order according to their finishing time.

Step 2:- Select the first activity from sorted array act[] and add it to Sol[] array.

Step 3:- Repeat steps 4 and 5 for the remaining activities in act[].

Step 4:- If the start time of the currently selected activity is greater than or equal to

finish time of Previously selected activity, then add it to the sol[] array.

Step 5 :- Select the next activity in act[] array.

Step 6 :- Print the sol[] array.

- Initially arr[]
- Sort the arr[] ascending order to finish time
- Print first activity and make i:=0
- Start [j] >= finish [i]
  Make i=j, j++
- Start [j] < finish [i], j++
- Start [j] >= finish [i]
  Make i=j, j++
- Make i=j, i+1, arr[]
- Start [j] < finish [i], arr[]

CODE :-

```
#include<bits/stdc++.h>
using namespace std;
#define Nb    // defines the number of activities
// structure represents an activity having start time
and finish time.
Struct Activity {
    int start, finish;
};
// This function is used for sorting activities according
to finish time
bool Sort_Activity ( Activity s1, Activity s2 ) {
  return (s1.finish < s2.finish);
}
// Print maximum number of activities done by single
person or single machine at a time.
void Print_MAX_Activities( Activity arr[], int n) {
 // Sort activities according to finish time
Sort (arr, arr+n, Sort_Activity);
Cout << " Following activities are selected \n";
```

```
// Select the first activity
int i = 0;
Cout << "( " << arr[i].start << ", " << arr[i].finish
<< " )\n";
// Consider the remaining activies from 1 to n-1
for (int j=1; j<n; j++) {
// Select activity if start time greater than or equal to finish time
of Previously selected activity
if (arr[j].start >= arr[i].finish ) {
Cout << "( " << arr[j].start << ", " << arr[j].finish << " )\n";
i = j;
}
}
}
// Driver Program
int main() {
Activity arr[N];
for (int i = 0; i <= N-1; i++) {
Cout << "Enter the start and end time of " << i+1 <<
" activity\n";
Cin >> arr[i].start >> arr[i].finish;
}
```

```
print_Max_Activities(arr, N);
return 0;
}
```

OUTPUT:-

```
Enter the start and end time of 1 activity
5 9
Enter the start and end time of 2 activity
1 2
Enter the start and end time of 3 activity
3 4
Enter the start and end time of 4 activity
0 6
Enter the start and end time of 5 activity
5 7
Enter the start and end time of 6 activity
8 9
Following activities are selected
(1, 2)
(3, 4)
(5, 7)
(8, 9)
```

EXPLANATION OF ALGORITHM WITH EXAMPLE:

Consider the table given. here 6 activities with corresponding start and end time, the objective is to compute an execution schedule having maximum number of non-conflicting activities.

Start Time (s)	5	1	3	0	5	8
Finish Time (f)	9	2	4	6	7	9
Activity Name	a1	a2	a3	a4	a5	a6

Steps:

1) Sort the given activities in ascending order according to their finishing time.

Start Time (s)	1	3	0	5	5	8
Finish Time (f)	2	4	6	7	9	9
Activity Name	a2	a3	a4	a5	a1	a6

2) Follow the steps 2, 3, 4 and 5 given in the algorithm. So, for the data given in the above table,

→ Select activity a3. Since start time of a3 is greater than a2's finish time.

ie, $s(a3) > f(a2)$

So we add a3 to the solution set,

sol = {a2, a3}

→ Select a4. s(a4) < f(a3), so it is not added to the solution set.

→ Select a5. Since here s(a5) > f(a3), it is added to the solution set,

sol = {a2, a3, a5}

→ Select a1, as s(a1) < f(a5), a1 is not added to the solution set.

→ Select a6. a6 is added to the solution set as s(6) > f(a5). Thus, the solution set, sol = {a2, a3, a5, a6}.

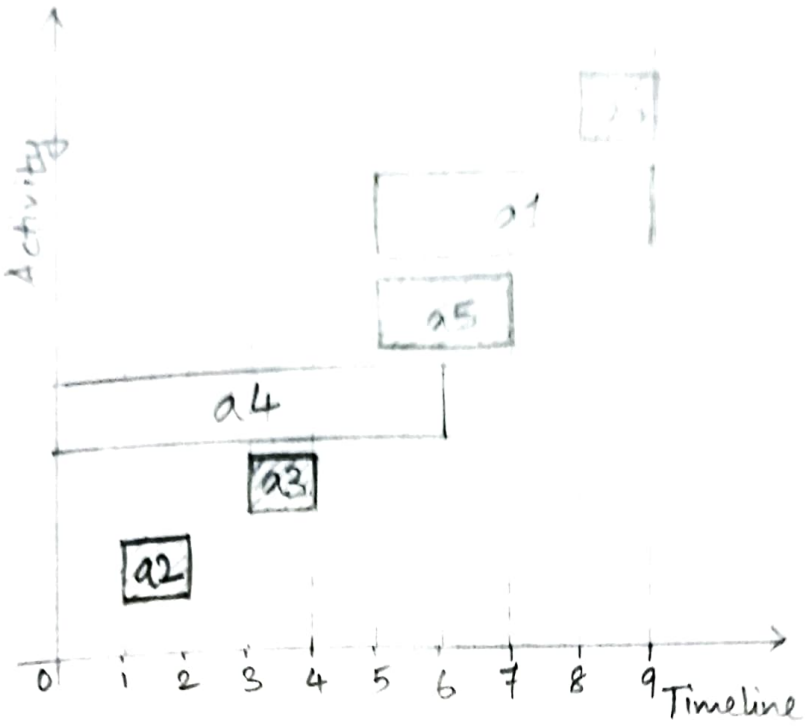
At last print the array sol[].

Hence the execution schedule of maximum number of non-conflicting activities will be,

OUTPUT:

```
(1, 2)    // a2
(3, 4)    // a3
(5, 7)    // a5
(8, 9)    // a6
```

the start and finish time of the activities in the solution set is printed.



In the above diagram, selected activity are shaded.

TIME COMPLEXITY ANALYSIS:

Case 1: When a given set of activities are sorted according to finishing time already, so in such a case no sorting mechanism involved, hence complexity of the algorithm will be $O(n)$.

Case 2: When a given set of activities is unsorted, in such a case we will have to include the sort() function defined for sorting activities list (bits/stdc++ header file). So, the time complexity of this method will be $O(n \log n)$, which also defines the complexity of the algorithm.

CONCLUSION:

Thus the activity selection problem was explained and executed successfully.

REFERENCE:

→https://www.studytonight.com/data-structres/activity-selection-problem.

→https://youtu.be/twipswztQr4

→ Activity Selection Problem (Using Greedy Method)

→ https://youtu.be/NE5erdkdrCk

→ Greedy Algorithm : Activity Selection Problem with example.

→ Book

Introduction to ALGORITHMS (Third Edition), Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.