Department of Computer Science and Engineering (Data Science)

Experiment 2

(Greedy Algorithm)

Aim: Implementation of activity selection problem using greedy approach.

Theory:

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.

Input Data for the Algorithm:

- act[] array containing all the activities.
- s[] array containing the starting time of all the activities.
- f[] array containing the finishing time of all the activities.

Ouput Data from the Algorithm:

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

Steps for Activity Selection Problem:

Following are the steps we will be following to solve the activity selection problem,

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 the 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.

Algorithm:

```
Activity-Selection(Activity, start, finish)
    Sort Activity by finish times stored in finish
    Selected = {Activity[1]}
    n = Activity.length
    j = 1
```

```
    for i = 2 to n:
        if start[i] ≥ finish[j]:
            Selected = Selected U {Activity[i]}
            j = i
 return Selected
```

Complexity:

Time complexity: O(n)

CODE:

```
#include <stdio.h>
#include <stdlib.h>
int* activity_selection(int a[], int s[], int f[], int n)
{
int* A = malloc(sizeof(int)*n);
A[0] = 0;
A[1] = a[1];
int k=1;
int i;
int t = 1;
for(i=2; i<=n; i++)
{
if(s[i] >= f[k])
{
t++;
A[t] = a[i];
k=i;
}}
A[0] = t;
return A;
}
int main() {
int a[] = {1,2,3,4,5,6};
int s[] = {5,1,3,0,5,8};
int f[] = {9,2,4,6,7,9};
printf("Jhanvi Parekh-60009210033\n");
```

int *p = activity_selection(a, s, f, 5);

int i;

for(i=1; i<=p[0]; i++)

{

printf("%d\n",p[i]);

}

return 0;

}

```c
#include <stdio.h>
#include <stdlib.h>
int* activity_selection(int a[], int s[], int f[], int n)
{
int* A = malloc(sizeof(int)*n);
A[0] = 0;
A[1] = a[1];
int k=1;
int i;
int t = 1;
for(i=2; i<=n; i++)
{
if(s[i] >= f[k])
{
t++;
A[t] = a[i];
k=i;
}}
A[0] = t;
return A;
}
int main() {
int a[] = {1,2,3,4,5,6};
int s[] = {5,1,3,0,5,8};
int f[] = {9,2,4,6,7,9};
printf("Jhanvi Parekh-60009210033\n");
int *p = activity_selection(a, s, f, 5);
int i;
for(i=1; i<=p[0]; i++)
{
printf("%d\n",p[i]);
}
return 0;
}
```

Department of Computer Science and Engineering (Data Science)

OUTPUT:

```
Output

/tmp/VOvoN1KBWw.o
Jhanvi Parekh-60009210033
2
3
5
6
```

Lab Assignment:

In the table below, we have 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)	Finish Time (f)	Activity Name
5	9	a1
1	2	a2
3	4	a3
0	6	a4
5	7	a5
8	9	a6