



## 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.

**Output 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



## Department of Computer Science and Engineering (Data Science)

```
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");
```



## Department of Computer Science and Engineering (Data Science)

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

```
int i;
```

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

```
{
```

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

```
}
```

```
return 0;
```

```
}
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int* activity_selection(int a[], int s[], int f[], int n)
4  {
5  int* A = malloc(sizeof(int)*n);
6  A[0] = 0;
7  A[1] = a[1];
8  int k=1;
9  int i;
10 int t = 1;
11 for(i=2; i<=n; i++)
12 {
13 if(s[i] >= f[k])
14 {
15 t++;
16 A[t] = a[i];
17 k=i;
18 }}
19 A[0] = t;
20 return A;
21 }
22 int main() {
23 int a[] = {1,2,3,4,5,6};
24 int s[] = {5,1,3,0,5,8};
25 int f[] = {9,2,4,6,7,9};
26 printf("Jhanvi Parekh-60009210033\n");
27 int *p = activity_selection(a, s, f, 5);
28 int i;
29 for(i=1; i<=p[0]; i++)
30 {
31 printf("%d\n",p[i]);
32 }
33 return 0;
34 }
```



**Department of Computer Science and Engineering (Data Science)**  
**OUTPUT:**

```
Output
/tmp/V0voN1KBww.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

Step 1: Sort the array (finish time)

start time (s)	finish time (f)	Activities (a)
1	2	A <sub>2</sub>
3	4	A <sub>3</sub>
0	6	A <sub>4</sub>
5	7	A <sub>5</sub>
5	9	A <sub>1</sub>
8	9	A <sub>6</sub>

→ Select A<sub>2</sub> as the first activity and add it to the solution array  
sol = [A<sub>2</sub>]

→ Select A<sub>3</sub> since the start time of A<sub>3</sub> is greater than the finishing time of A<sub>2</sub>  
start(A<sub>3</sub>) > finish(A<sub>2</sub>)  
then add A<sub>3</sub> to solution array.

sol = [A<sub>2</sub>, A<sub>3</sub>]

→ Select A<sub>1</sub>, since start(A<sub>1</sub>) ≤ finish(A<sub>3</sub>)  
the activity is rejected and is not added in the solution array

→ Select A<sub>5</sub>, since start(A<sub>5</sub>) ≥ finish(A<sub>3</sub>)  
only selected activity are compared  
sol = [A<sub>2</sub>, A<sub>3</sub>, A<sub>5</sub>]

→ Select A<sub>4</sub>, since start(A<sub>4</sub>) ≤ finish(A<sub>5</sub>)  
the activity is rejected and is not added in the solution array

→ select (A<sub>6</sub>), since start(A<sub>6</sub>) ≥ finish(A<sub>5</sub>)  
sol = [A<sub>2</sub>, A<sub>3</sub>, A<sub>5</sub>, A<sub>6</sub>]

The final selected activities that give the optimal solution for the activities given are as  
sol = [A<sub>2</sub>, A<sub>3</sub>, A<sub>5</sub>, A<sub>6</sub>]

\* Activity chart