# **Activity Selection Problem**

## **Problem:**

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Example 1 : Consider the following 3 activities sortd by by finish time.

start[] = {10, 12, 20};

finish[] = {20, 25, 30};

A person can perform at most two activities. The maximum set of activities that can be executed is {0, 2} [ These are indexes in start[] and finish[] ]

Example 2 : Consider the following 6 activities sorted by by finish time.

start[] = {1, 3, 0, 5, 8, 5};

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

A person can perform at most four activities. The maximum set of activities that can be executed is {0,1,3,4} [ These are indexes in start[] and finish[] ]

## **Solution:**

Greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

## **Algorithm:**

1. Sort the activities according to their finishing time
2. Select the first activity from the sorted array and print it.
3. Do following for remaining activities in the sorted array.
   1. If the start time of this activity is greater than or equal to the finish time of previously selected activity, then select this activity and print it.

#include <iostream>

#include <cstdlib>

#include <cstring>

#include <algorithm>

using namespace std;

#define SIZE 100000

struct Activitiy {

int start, finish;

};

bool compare\_fun(struct Activitiy s1, struct Activitiy s2);

class ActivitySelection {

private:

int no\_act;

int start[SIZE];

int finish[SIZE];

public:

ActivitySelection() {

no\_act = 0;

memset(start, 0, SIZE);

memset(finish, 0, SIZE);

}

~ActivitySelection() {

no\_act = 0;

memset(start, 0, SIZE);

memset(finish, 0, SIZE);

}

int set\_noOfActivity(void);

int set\_startTime(void);

int set\_finishTime(void);

int get\_maxNoOfActivity(void);

};

int ActivitySelection::set\_noOfActivity(void) {

cin >> no\_act;

return 0;

}

int ActivitySelection::set\_startTime(void) {

for(int i = 0; i < no\_act; i++) {

cin >> start[i];

}

return 0;

}

int ActivitySelection::set\_finishTime(void) {

for(int i = 0; i < no\_act; i++) {

cin >> finish[i];

}

return 0;

}

bool compare\_fun(struct Activitiy s1, struct Activitiy s2) {

return (s1.finish < s2.finish);

}

int ActivitySelection::get\_maxNoOfActivity(void) {

int max\_act = 0;

int next\_act\_index = 0;

int prev\_act\_index = 0;

int n = 0;

struct Activitiy arr[no\_act];

for(int i = 0; i < no\_act; i++) {

arr[i].start = start[i];

arr[i].finish = finish[i];

}

n = sizeof(arr)/sizeof(arr[0]);

sort(arr, arr + n, compare\_fun);

//sort(arr, arr + no\_act, );

/\*

for(int i = 0; i < no\_act; i++) {

cout << finish[i] << " ";

}

\*/

prev\_act\_index = 0;

max\_act = 1; // to consider first activity

for(next\_act\_index = 1; next\_act\_index < no\_act; next\_act\_index++) {

if(arr[next\_act\_index].start >= arr[prev\_act\_index].finish) {

max\_act = max\_act + 1;

prev\_act\_index = next\_act\_index;

}

} // end of for loop

return max\_act;

}

int main(void) {

int maxAct = 0;

ActivitySelection objact;

objact.set\_noOfActivity();

objact.set\_startTime();

objact.set\_finishTime();

maxAct = objact.get\_maxNoOfActivity();

cout << maxAct;

return 0;

}

Output:

Input:

3

10 12 20

20 25 30

Output:

2

# **Huffman Coding**

# **Job Sequencing Problem**

# **Efficient Huffman Coding for Sorted Input**

# **END**