The task-scheduling problem

September 10, 2018

Student: Toma Alexandru

Computers and Information Technology (Engligh Language)

Group 1.3 B

Introduction

The problem can be solved using 3 sorting methods:

1.Earliest Activity First: Repeatedly select the activity with the earliest start time, provided that it does not overlap any of the previously scheduled activities. Shortest Activity First: Repeatedly select the activity with the smallest duration (fisi), provided that it does not conflict with any previously scheduled activities.

2.Earliest Activity First: Repeatedly select the activity with the earliest start time, provided that it does not overlap any of the previously scheduled activities. Shortest Activity First: Repeatedly select the activity with the smallest duration (fisi), provided that it does not conflict with any previously scheduled activities.

3.Lowest Conflict Activity First: Repeatedly select the activity that conflicts with the smallest number of remaining activities, provided that it does not conflict with of the previously scheduled activities. (Note that once an activity is selected, all the conflicting activities can be effectively deleted, and this affects the conflict counts for the remaining activities.)

Problem statement

Schedule several competing activities that require exclusive use of a common resource, with a goal of selecting a maximum-size set of mutually compatible activities. Given that an activity has a start time and a finish time two activities are compatible if the intervals between theirs start and finish times don not overlap. Implement two different algorithms that resolve this problem.

An example of an input and two possible solutions is given in the representation below.

latex2.png

Pseudocode

In the first function we will sort our task from the lowest duration to the higest duration using the bubblesort algorithm.In the second function we will check if the end time of a task overlaps the start time of the next task.
We will use functions which will be called by the main program.Here are the functions usedby the program:

```
void bubbleSort( task arr[], int n)
1.
2.    for i <- 0 to n-1 do
3.        for j <- 0 to n-i-1 do
4.            if arr[j].duration > arr[j+1].duration
5.                taskAux.start <- arr[j].start
6.                taskAux.finish <- arr[j].finish
7.                taskAux.duration <- arr[j].duration
8.
9.                arr[j].start <- arr[j+1].start
10.               arr[j].finish <- arr[j+1].finish
11.               arr[j].duration <- arr[j+1].duration
12.
13.               arr[j+1].start= taskAux.start
14.               arr[j+1].finish= taskAux.finish
15.               arr[j+1].duration= taskAux.duration
16.
17.          else if arr[j].duration = arr[j+1].duration
18.
19.              if arr[j].start > arr[j+1].start
20.
21.                  taskAux.start <- arr[j].start
22.                  taskAux.finish <- arr[j].finish
23.                  taskAux.duration <- arr[j].duration
24.
25.                  arr[j].start <- arr[j+1].start;
26.                  arr[j].finish <- arr[j+1].finish
27.                  arr[j].duration <- arr[j+1].duration
28.
29.                  arr[j+1].start <- taskAux.start
30.                  arr[j+1].finish <- taskAux.finish
31.                  arr[j+1].duration <- taskAux.duration
32.
33.
34.
35.
```

```
int sortare( task taskuri[],int n)
1
2    aux <- taskuri[0].finish
3
4    for i <- 0 to n-2 do
5       for j <- i+1 to n-1 do
6
7             if aux <= taskuri[j].start
8
9                 k <- k+taskuri[j].duration;
10                aux <- taskuri[j].finish;
11
12                i++;
13
14
15    return k;
```

Application design

The library contains the header functions.h which has all the function prototypes to compute the required operations. These are all of them:
—void bubbleSort(task arr[], int n)
—int sortare(task taskuri[],int n)

The source file functions.c has all the function implementations. Function void bubbleSort(task arr[], int n) includes some steps like:
1) First we order all task according to their duration.
2) Then if two task have the same duration we order them by their start time.

Function int sortare(task taskuri[],int n) does the following sorting:
1)We first save the finish time of the first activity in aux(that was previously sorted with the bubblesort algorithm).
2)Then we check if the aux(the finish time of the first activity) is smaller the than the begining time of the next task(second).If it is, we store the duration of the task in a variable k, aux(the finish time of the first activity) is now equal to the finish time of the next task(second) and we move on to the next activity, and so on.

The final source file,main.c itself contains:
1)It verifies if the file "in.txt" contains any values
2)It reads n which represents the total number of tasks
3)It verifies if the first entry of the file is odd,if it is odd than that value represents the start time of a task, if it is even it represents the finish time of a task.
4)It calcultaes the duration of a task(finish time-start time)
5)It calls the bubblesort function which we expained it earlier and than we print all the tasks(now sorted).
6)It calls the funnction sortare which calculates the total duration of the tasks and than it prints it.

Source Code

```
//---------------------------functions.h---------------------------

#ifndef MAIN_H_INCLUDED
#define MAIN_H_INCLUDED

#include <stdio.h>
#include <stdlib.h>

typedef struct{
    int start;
    int finish;
    int duration;
}task;

void bubbleSort( task arr[], int n);
int sortare(task taskuri[],int n);

#endif


//---------------------------functions.c---------------------------

#include "function.h"
int aux=0;
task taskuri[20],taskAux;
void bubbleSort( task arr[], int n)
{

    int i, j;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j].duration > arr[j+1].duration)
            {
                taskAux.start = arr[j].start;
                taskAux.finish=arr[j].finish;
                taskAux.duration=arr[j].duration;

                arr[j].start = arr[j+1].start;
                arr[j].finish = arr[j+1].finish;
                arr[j].duration = arr[j+1].duration;

                arr[j+1].start= taskAux.start;
                arr[j+1].finish= taskAux.finish;
                arr[j+1].duration= taskAux.duration;
            }
            else if(arr[j].duration == arr[j+1].duration)
            {
```

```c
                if(arr[j].start > arr[j+1].start)
                {
                    taskAux.start = arr[j].start;
                    taskAux.finish=arr[j].finish;
                    taskAux.duration=arr[j].duration;

                    arr[j].start = arr[j+1].start;
                    arr[j].finish = arr[j+1].finish;
                    arr[j].duration = arr[j+1].duration;

                    arr[j+1].start= taskAux.start;
                    arr[j+1].finish= taskAux.finish;
                    arr[j+1].duration= taskAux.duration;

                }

            }


}

int sortare( task taskuri[],int n)

{
    int i;
    int j;
    int k;

    aux=taskuri[0].finish;
    for(i=0; i<=n-2; i++)
    {
        for(j=i+1; j<=n-1; j++)
        {
            if(aux<=taskuri[j].start)
            {
                k=k+taskuri[j].duration;

                aux=taskuri[j].finish;
                i++;
            }
        }

    }

    return k;


}
```

```c
//-------------------------------main.c-------------------------------

#include "function.h"

int total_duration=1;
int i;
int j;
int number_of_tasks;
int p;
task taskuri[20];

int main()
{

    FILE *myFile;
    myFile = fopen("in.txt", "r");
    int numberArray[1000];
    int i,j;
    int n;
    int t;
    printf("time=");
    scanf("%d",&t);

    if (myFile == NULL) // daca avem cv de citit
    {
        printf("Error Reading File\n");
        exit (0);
    }

    fscanf(myFile, "%d,", &numberArray[0]);
    n=numberArray[0];
    j=0;

    for (i = 1; i <= n*2; i++)                          // scriere in
         fisier
    {
        fscanf(myFile, "%d,", &numberArray[i] );
        if(i%2!=0)
        {
            taskuri[j].start=numberArray[i];
        }
        if(i%2==0)
        {
            taskuri[j].finish=numberArray[i];
            j++;
        }
    }
```

```c
    for (i = 0; i < n; i++)                    // cat dureaza un taskk
    {
        taskuri[i].duration=taskuri[i].finish-taskuri[i].start;
    }

    bubbleSort(taskuri,n);          // apelam functia bublesort ce
        ordoneaza in functie de durata si de start

    for (i = 0; i < n; i++)
    {
        printf("task_%d: start_%d, finish_%d , duration_%d\n\n",
            i,taskuri[i].start,taskuri[i].finish,taskuri[i].duration );
    }

    total_duration=sortare(taskuri,n);

    printf("The maximum duration is=%d",total_duration);
    fclose(myFile);

    return 0;
}
```

Experiments and results

```
Number of tasks is: 8
The first row represents the total number of tasks.
The first column represents the start time of a task.
The second column represents the end time of a task.
8
5 10
1 2
1 3
2 4
4 7
6 7
2 5
9 22
```

The duration of each task is calculated by substracting the end time of
    the task form the start time of the task
(the duration of each task is representated in the third column)
```
5 10 5
1 2  1
1 3  2
2 4  2
4 7  3
6 7  1
2 5  3
9 22 13
```

The algorithm will start ordering all the tasks by the duration of each
    task.

```
1 2  1
6 7  1
1 3  2
2 4  2
4 7  3    *
2 5  3    *
5 10 5
9 22 13
```

Now, the algorithm will start ordering each task keeping in mind the
    duration and the start time of a task

1 2 1
6 7 1
1 3 2
2 4 2
2 5 3   *
4 7 3   *
5 10 5
9 22 13

Finally the algorithm will print the maximum duration(k):
1 2 1
6 7 1
9 22 13

k=1+1+13=15

References

1)http://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect07-greedy-sched.pdf

2)https://www.sharelatex.com

3)https:https://stackoverflow.com/questions

4)https://www.geeksforgeeks.org