|  |
| --- |
| Name: NGUYỄN ANH TÀI -ID: 20520924  Name: LÝ KIỀU CHÍ - ID: 20521131  Class: IT007.M22.1 |

OPERATING SYSTEM  
LAB 04’S REPORT

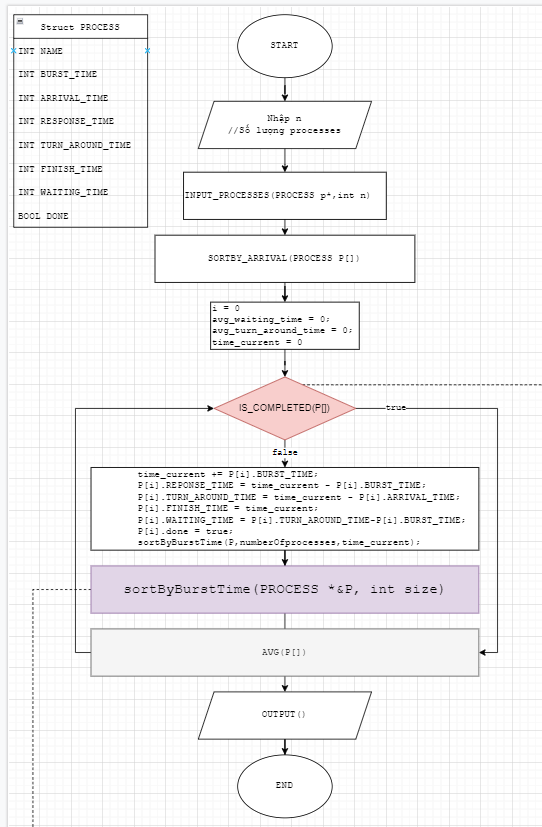
**SUMMARY**

|  |  |  |
| --- | --- | --- |
| **Task** | **Status** | **Page** |
| **1. Giải thuật SJF:** Vẽ lưu đồ giải thuật  -Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình  -Thực hiện code cho giải thuật  -Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình | DONE |  |
| **2. giải thuật SRTF:**  -Vẽ lưu đồ giải thuật  -Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình  -Thực hiện code cho giải thuật  -Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình | DONE |  |
| **3. (Bonus) Thực hiện các yêu cầu trên với giải thuật còn lại.** | UNDONE |  |

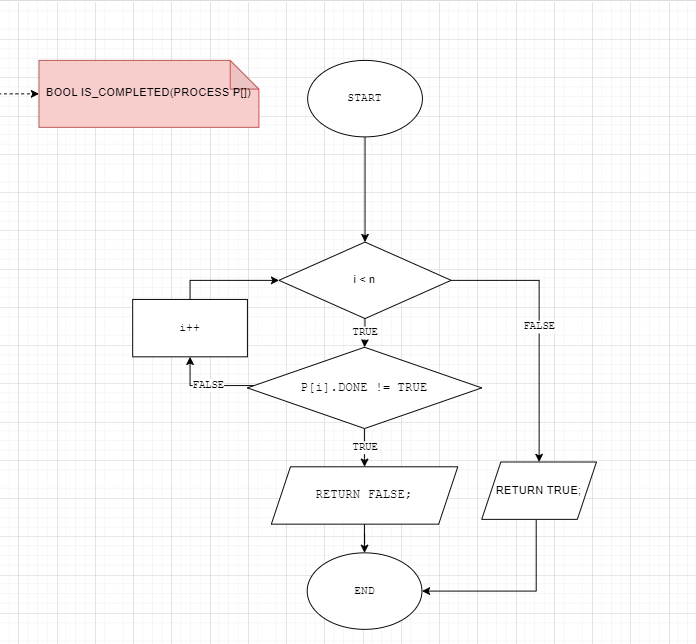
**Self-scrores: 8**

# TASK 01: GIẢI THUẬT SJF

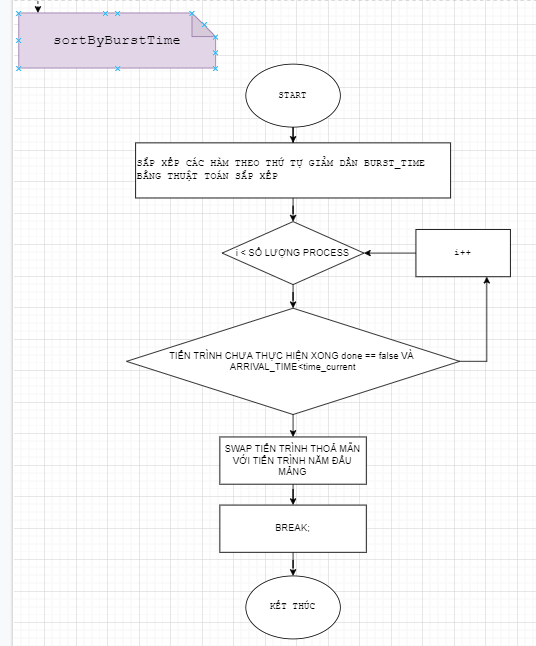
## 1. Lưu đồ giải thuật.

****

Hình Lưu đồ thuật toán SJF

****

Hình : Lưu đồ kiểm tra trạng thái hoàn thành của các tiến trình

****

Hình :Lưu đồ Hàm sắp xếp giảm mảng các tiến trình theo thứ tự giảm dần

## 2. Code giải thuật.

#include <iostream>

#include <string>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <vector>

using **namespace** std;

**struct** PROCESS

{

    string NAME;

**int** BURST\_TIME;

**int** ARRIVAL\_TIME;

**int** REPONSE\_TIME;

**int** TURN\_AROUND\_TIME;

**int** FINISH\_TIME;

**int** WAITING\_TIME;

**bool** done = false;

};

**double** avg\_waiting\_time = 0;

**double** avg\_turn\_around\_time = 0;

**void** Add\_Process(**int** **&**size, PROCESS **\*&**P, PROCESS process)

{

    PROCESS \*newArr = new PROCESS[size + 1];

    for (**int** index = 0; index < size; index++)

    {

        newArr[index] = P[index];

    }

    newArr[size] = process;

    size++;

    P = newArr;

}

**void** swap(PROCESS **&**p1, PROCESS **&**p2)

{

    PROCESS tmp;

    tmp = p1;

    p1 = p2;

    p2 = tmp;

}

**void** sortByArrivalTime(PROCESS **\*&**P, **int** size)

{

    for (**int** i = 0; i < size; i++)

    {

        for (**int** j = i + 1; j < size; j++)

        {

            if (P[i].ARRIVAL\_TIME > P[j].ARRIVAL\_TIME)

            {

                swap(P[i], P[j]);

            }

        }

    }

}

**void** sortByBurstTime(PROCESS **\*&**P, **int** n, **int** time\_current)

{

    for (**int** i = 0; i < n; i++)

    {

        for (**int** j = i + 1; j < n; j++)

        {

            if (P[i].BURST\_TIME > P[j].BURST\_TIME)

            {

                swap(P[i], P[j]);

            }

        }

    }

       for (**int** i = 0; i < n; i++)

        {

            if (P[i].done == false &&P[i].ARRIVAL\_TIME<time\_current)

            {

                swap(P[i], P[0]);

                break;

            }

        }

}

**void** Input\_Process(PROCESS **\*&**P, **int** **&**size)

{

    cout << "Input the number of Processes: ";

    cin >> size;

    PROCESS \*newArr = new PROCESS[size];

    for (**int** i = 0; i < size; i++)

    {

        cin.ignore(32767, '\n');

        cout << "----------------[" << i << "]-----------------\n";

        cout << "ARRIVAL\_TIME: ";

        cin >> newArr[i].ARRIVAL\_TIME;

        cout << "BUSRT\_TIME: ";

        cin >> newArr[i].BURST\_TIME;

        newArr[i].NAME = (**char**)('A' + i);

    }

    P = newArr;

}

string space(**int** n)

{

    string res;

    for (**int** i = 0; i < n; i++)

    {

        res += " ";

    }

    return res;

}

**void** Output\_Process(**int** numberOfprocesses, PROCESS **\***P)

{

    sortByArrivalTime(P, numberOfprocesses);

    string Attribute[6] = {"Job", "Arrival\_Time", "Burst\_Time", "Finish\_Time", "Turnaround\_Time", "Waiting\_Time"};

**int** num\_columns = sizeof(Attribute) / sizeof(Attribute[0]);

    vector<vector<string>> board;

    vector<string> row {"\tSHORTEST JOB FIRST\n" } ;

    board.push\_back(row);

    for (**int** i = 0; i < numberOfprocesses + 1; i++)

    {

        vector<string> row;

        if (i == 0)

        {

            for (**int** j = 0; j < num\_columns; j++)

            {

                if (j == 0)

                    row.push\_back("|" + Attribute[0] + "|");

                else

                    row.push\_back(Attribute[j] + "|");

            }

            board.push\_back(row);

        }

        else

        {

**int** ele = i - 1;

            string Value[6] = {P[ele].NAME, to\_string(P[ele].ARRIVAL\_TIME), to\_string(P[ele].BURST\_TIME), to\_string(P[ele].FINISH\_TIME), to\_string(P[ele].TURN\_AROUND\_TIME), to\_string(P[ele].WAITING\_TIME)};

            for (**int** j = 0; j < num\_columns; j++)

            {

                if (j == 0)

                    row.push\_back("|" + space(Attribute[0].length() - Value[0].length()) + Value[0] + "|");

                else

                    row.push\_back(space(Attribute[j].length() - Value[j].length()) + Value[j] + "|");

            }

            board.push\_back(row);

        }

    }

       row = {"Average Turn Around Time: " +to\_string(avg\_turn\_around\_time)+"\n",

                         "Average Turn Waiting Time: "  +to\_string(avg\_waiting\_time)+"\n"  } ;

        board.push\_back(row);

    for (**int** i = 0; i < numberOfprocesses + 3; i++)

    {

        for (**int** j = 0; j < num\_columns; j++)

        {

            if (i==numberOfprocesses+2)

            {

                cout << board[i][0].c\_str();

                cout << board[i][1].c\_str();

                break;

            }

            if (i==0)

            {

                cout << board[i][0].c\_str();

                break;

            }

            cout << board[i][j].c\_str();

        }

        cout << "\n";

    }

}

**bool** isCompleted(PROCESS **\***P, **int** Size)

{

    for (**int** i = 0; i < Size; i++)

    {

        if (!P[i].done)

        {

            return false;

        }

    }

    return true;

}

**void** AVG(PROCESS **\***P,**int** Size){

    for (**int** i = 0; i < Size; i++)

    {

        avg\_turn\_around\_time += (**double**)P[i].TURN\_AROUND\_TIME/Size;

        avg\_waiting\_time += (**double**)P[i].WAITING\_TIME/Size;

    }

}

**int** main()

{

**int** numberOfprocesses = 1;

    PROCESS \*P = new PROCESS[numberOfprocesses];

    Input\_Process(P, numberOfprocesses);

    sortByArrivalTime(P, numberOfprocesses);

**int** i = 0;

**int** time\_current = 0;

    while (!isCompleted(P, numberOfprocesses))

    {

        time\_current += P[i].BURST\_TIME;

        P[i].REPONSE\_TIME = time\_current - P[i].BURST\_TIME;

        P[i].TURN\_AROUND\_TIME = time\_current - P[i].ARRIVAL\_TIME;

        P[i].FINISH\_TIME = time\_current;

        P[i].WAITING\_TIME = P[i].TURN\_AROUND\_TIME-P[i].BURST\_TIME;

        P[i].done = true;

        sortByBurstTime(P,numberOfprocesses,time\_current);

    }

    AVG(P,numberOfprocesses);

    Output\_Process(numberOfprocesses, P);

    return 0;

}

## 3. DIỄN GIẢI.

- Đầu tiên tạo kiểu dữ liệu tiến trình với các thuộc tính theo Lưu đồ trên.  
Ảnh có chứa văn bản, biên lai

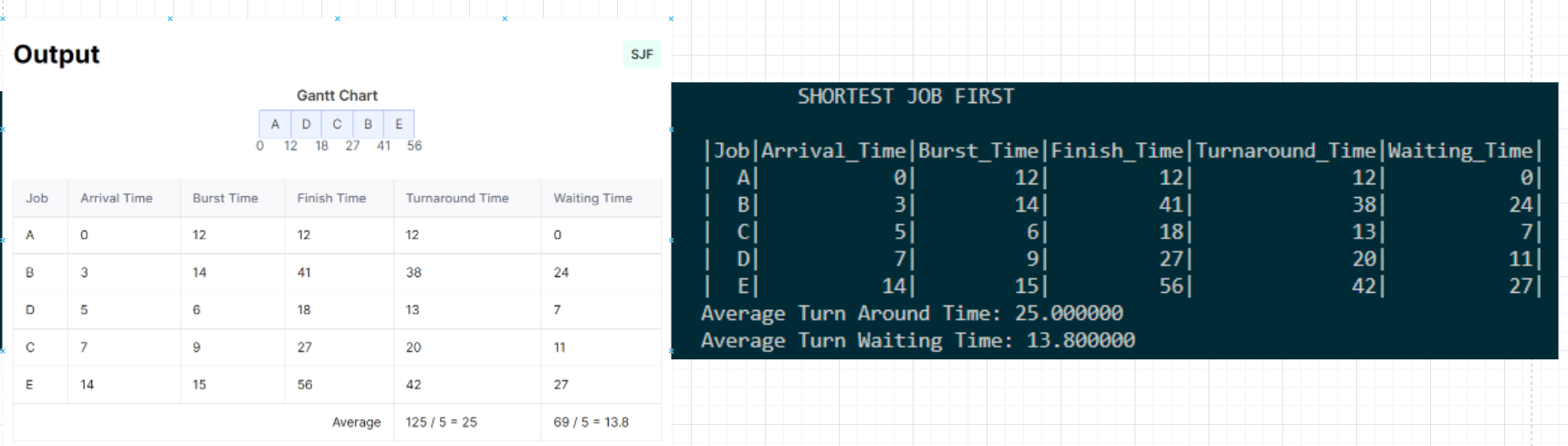
Mô tả được tạo tự động  
-Sau khi nhập dữ liệu cho mảng các tiến trình thì ta sắp tiến trình theo thứ tự giảm dần theo ARRIVAL\_TIME đưa tiến trình tới sớm nhất lên đầu mảng.  
Ảnh có chứa văn bản, thiết bị, ảnh chụp màn hình

Mô tả được tạo tự động  
-Khi mà tất cả các tiến trình vẫn chưa được hoàn thành thì thay đổi các thuộc tính của tiến trình nằm đầu mảng  
Ảnh có chứa văn bản

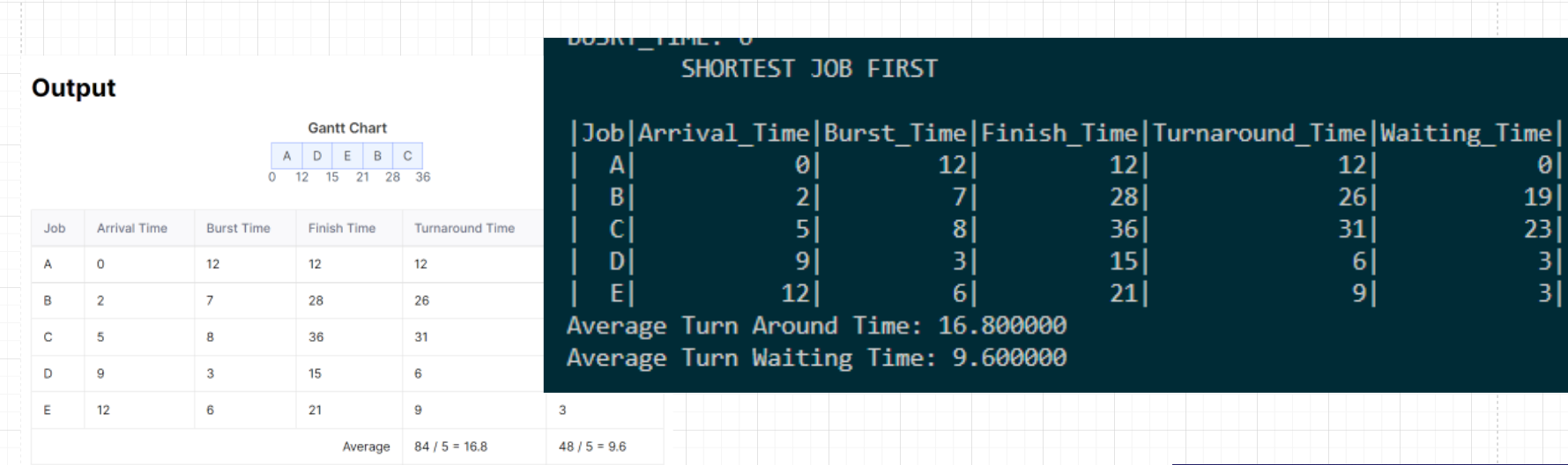
Mô tả được tạo tự động  
-Sau mỗi lần thay đổi các thuộc tính thì ta sắp xếp tiến trình lại một lần nữa theo thứ tự giảm dần về BURST\_TIME theo một điều kiện đặc biệt nhằm Tìm ra tiến trình có BURST\_TIME ngắn nhất và đưa nó lên đầu mảng. Chi tiết của hàm sortByBurstTime xem tại *Hình 3*

-Cứ như thế sau mỗi lần lặp thì tiến trình có BurstTime Nhỏ nhất sẽ được đưa lên đầu mảng và thực thi cho đến khi tất cả các tiến trình vào trạng thái hoàn thành.

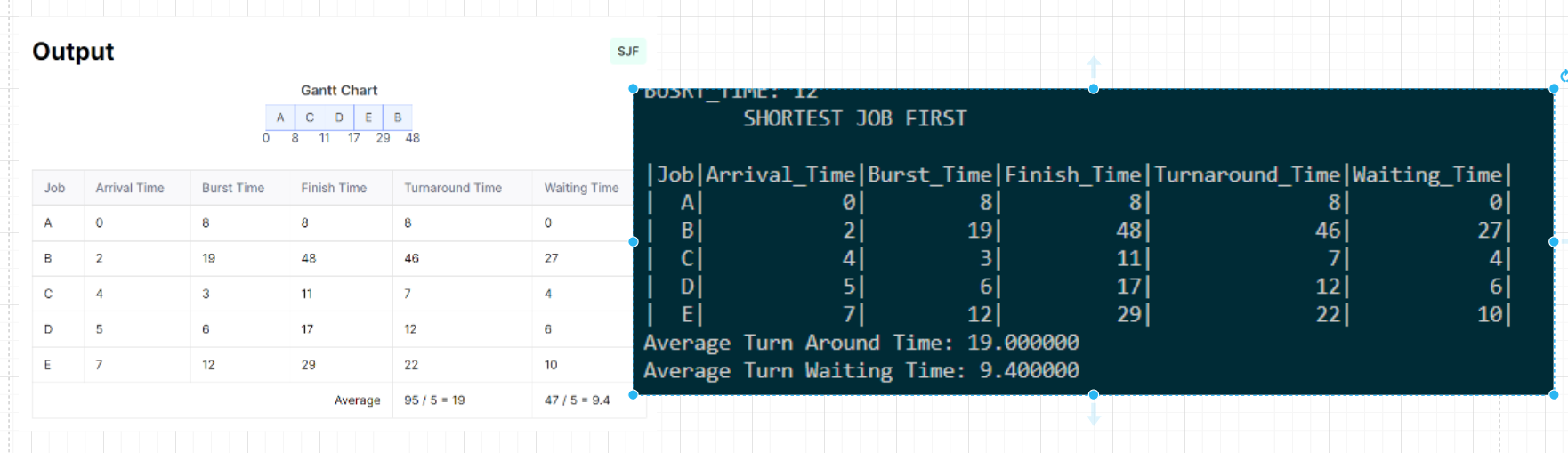
## 4. Tính đúng đắn của giải thuật và code.

****

Hình 4: Test Case 01 Chạy Bằng Tay Và Kết quả code

****

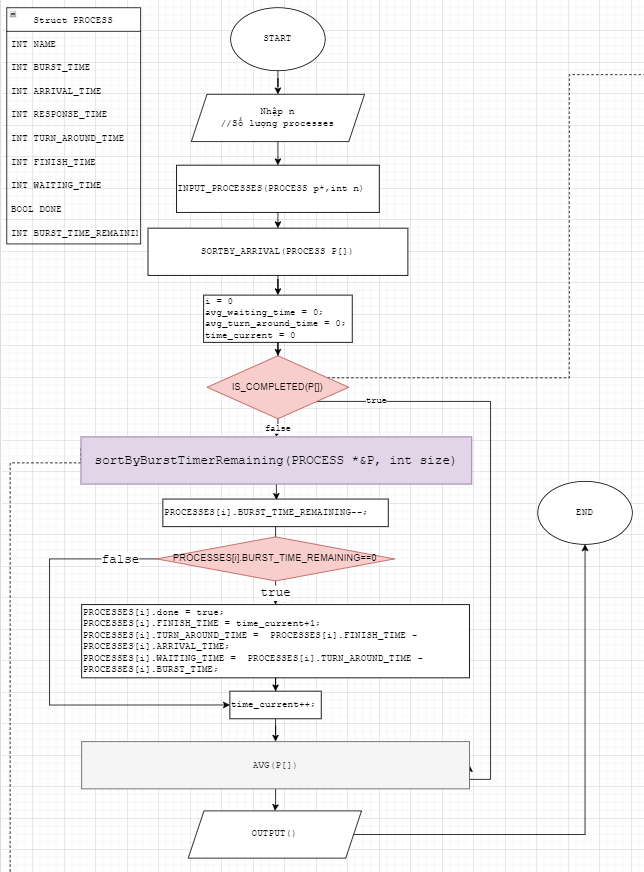
Hình 5: Test Case 02 Chạy Bằng Tay Và Kết quả code

****

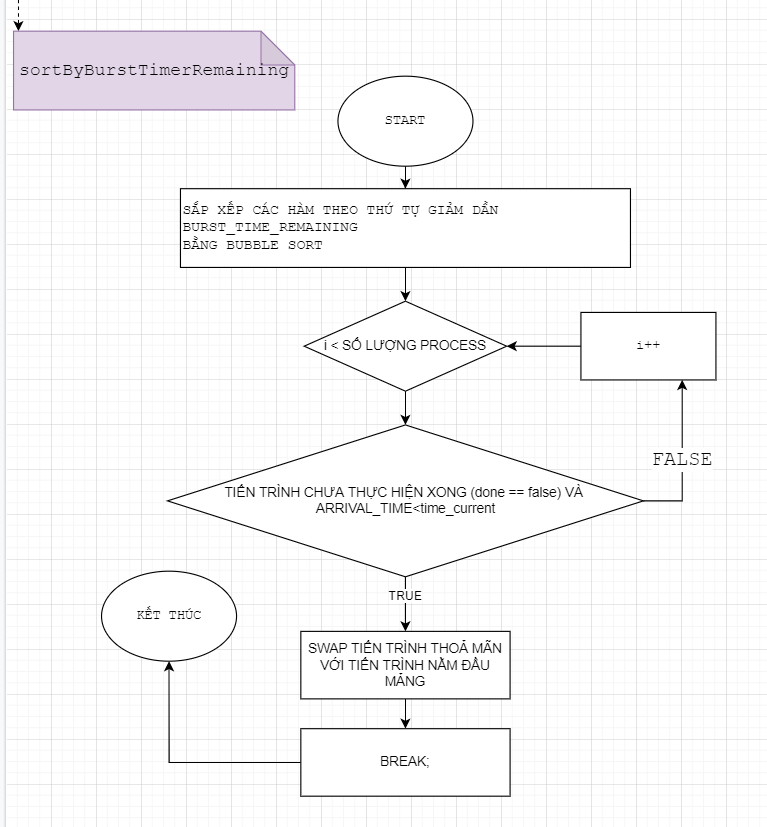
Hình 6: Test Case 03 Chạy Bằng Tay Và Kết quả code

# TASK 02: GIẢI THUẬT SRTF

1. **Lưu đồ giải thuật SRTF.**

****

Hình : Lưu đồ giải thuật SRTF



Hình : Lưu đồ thuật toán sắp xếp giảm dần theo REMAINING\_BURST\_TIME

1. **Code giải thuật SRTF.**

#include <iostream>

#include <string>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <vector>

using **namespace** std;

**struct** PROCESS

{

    string NAME;

**int** BURST\_TIME;

**int** ARRIVAL\_TIME;

**int** REPONSE\_TIME;

**int** TURN\_AROUND\_TIME;

**int** FINISH\_TIME;

**int** WAITING\_TIME;

**bool** done = false;

**int** BURST\_TIME\_REMAINING;

};

**double** avg\_waiting\_time = 0;

**double** avg\_turn\_around\_time = 0;

**void** Add\_Process(**int** **&**size, PROCESS **\*&**P, PROCESS process)

{

    PROCESS \*newArr = new PROCESS[size + 1];

    for (**int** index = 0; index < size; index++)

    {

        newArr[index] = P[index];

    }

    newArr[size] = process;

    size++;

    P = newArr;

}

**void** swap(PROCESS **&**p1, PROCESS **&**p2)

{

    PROCESS tmp;

    tmp = p1;

    p1 = p2;

    p2 = tmp;

}

**void** sortByArrivalTime(PROCESS **\*&**P, **int** size)

{

    for (**int** i = 0; i < size; i++)

    {

        for (**int** j = i + 1; j < size; j++)

        {

            if (P[i].ARRIVAL\_TIME > P[j].ARRIVAL\_TIME)

            {

                swap(P[i], P[j]);

            }

        }

    }

}

**void** sortByBurstTimeRemaining(PROCESS **\*&**P, **int** n, **int** time\_current)

{

     for (**int** i = 0; i < n; i++)

    {

        for (**int** j = i + 1; j < n; j++)

        {

            if (P[i].BURST\_TIME\_REMAINING > P[j].BURST\_TIME\_REMAINING)

            {

                swap(P[i], P[j]);

            }

        }

    }

       for (**int** i = 0; i < n; i++)

        {

            if (P[i].done == false &&P[i].ARRIVAL\_TIME<=time\_current)

            {

                swap(P[i], P[0]);

                break;

            }

        }

}

**void** Input\_Process(PROCESS **\*&**P, **int** **&**size)

{

    cout << "Input the number of Processes: ";

    cin >> size;

    PROCESS \*newArr = new PROCESS[size];

    for (**int** i = 0; i < size; i++)

    {

        cin.ignore(32767, '\n');

        cout << "----------------[" << i << "]-----------------\n";

        cout << "ARRIVAL\_TIME: ";

        cin >> newArr[i].ARRIVAL\_TIME;

        cout << "BUSRT\_TIME: ";

        cin >> newArr[i].BURST\_TIME;

        newArr[i].NAME = (**char**)('A' + i);

         newArr[i].BURST\_TIME\_REMAINING = newArr[i].BURST\_TIME;

    }

    P = newArr;

}

string space(**int** n)

{

    string res;

    for (**int** i = 0; i < n; i++)

    {

        res += " ";

    }

    return res;

}

**void** Output\_Process(**int** numberOfprocesses, PROCESS **\***P)

{

    sortByArrivalTime(P, numberOfprocesses);

    string Attribute[6] = {"Job", "Arrival\_Time", "Burst\_Time", "Finish\_Time", "Turnaround\_Time", "Waiting\_Time"};

**int** num\_columns = sizeof(Attribute) / sizeof(Attribute[0]);

    vector<vector<string>> board;

    vector<string> row {"\tShortest Remaining Time First, SRTF\n" } ;

    board.push\_back(row);

    for (**int** i = 0; i < numberOfprocesses + 1; i++)

    {

        vector<string> row;

        if (i == 0)

        {

            for (**int** j = 0; j < num\_columns; j++)

            {

                if (j == 0)

                    row.push\_back("|" + Attribute[0] + "|");

                else

                    row.push\_back(Attribute[j] + "|");

            }

            board.push\_back(row);

        }

        else

        {

**int** ele = i - 1;

            string Value[6] = {P[ele].NAME, to\_string(P[ele].ARRIVAL\_TIME), to\_string(P[ele].BURST\_TIME), to\_string(P[ele].FINISH\_TIME), to\_string(P[ele].TURN\_AROUND\_TIME), to\_string(P[ele].WAITING\_TIME)};

            for (**int** j = 0; j < num\_columns; j++)

            {

                if (j == 0)

                    row.push\_back("|" + space(Attribute[0].length() - Value[0].length()) + Value[0] + "|");

                else

                    row.push\_back(space(Attribute[j].length() - Value[j].length()) + Value[j] + "|");

            }

            board.push\_back(row);

        }

    }

         row = {"Average Turn Around Time: " +to\_string(avg\_turn\_around\_time)+"\n",

                         "Average Turn Waiting Time: "  +to\_string(avg\_waiting\_time)+"\n"  } ;

        board.push\_back(row);

    for (**int** i = 0; i < numberOfprocesses + 2+1; i++)

    {

        for (**int** j = 0; j < num\_columns; j++)

        {

            if (i==numberOfprocesses+1+1)

            {

                cout << board[i][0].c\_str();

                cout << board[i][1].c\_str();

                break;

            }

             if (i==0)

            {

                cout << board[i][0].c\_str();

                break;

            }

            cout << board[i][j].c\_str();

        }

        cout << "\n";

    }

}

**bool** isCompleted(PROCESS **\***P, **int** Size)

{

    for (**int** i = 0; i < Size; i++)

    {

        if (!P[i].done)

        {

            return false;

        }

    }

    return true;

}

**void** AVG(PROCESS **\***P,**int** Size){

    for (**int** i = 0; i < Size; i++)

    {

        avg\_turn\_around\_time += (**double**)P[i].TURN\_AROUND\_TIME/Size;

        avg\_waiting\_time += (**double**)P[i].WAITING\_TIME/Size;

    }

}

**int** main()

{

**int** numberOfprocesses = 1;

    PROCESS \*PROCESSES = new PROCESS[numberOfprocesses];

    Input\_Process(PROCESSES, numberOfprocesses);

**int** i = 0;

     for(**int** time\_current=0;!isCompleted(PROCESSES,numberOfprocesses); time\_current++)

    {

        sortByBurstTimeRemaining(PROCESSES,numberOfprocesses,time\_current);

        PROCESSES[i].BURST\_TIME\_REMAINING--;

        if(PROCESSES[i].BURST\_TIME\_REMAINING==0)

        {

            PROCESSES[i].done = true;

            PROCESSES[i].FINISH\_TIME = time\_current+1;

            PROCESSES[i].TURN\_AROUND\_TIME =  PROCESSES[i].FINISH\_TIME -  PROCESSES[i].ARRIVAL\_TIME;

            PROCESSES[i].WAITING\_TIME =  PROCESSES[i].TURN\_AROUND\_TIME - PROCESSES[i].BURST\_TIME;

        }

    }

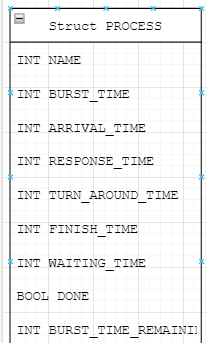
    AVG(PROCESSES,numberOfprocesses);

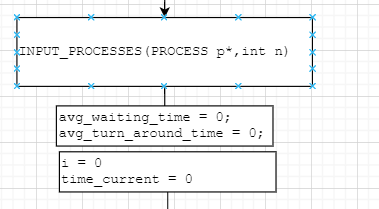
    Output\_Process(numberOfprocesses, PROCESSES);

    return 0;

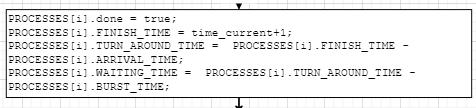
}

1. **DIỄN GIẢI SRTF**

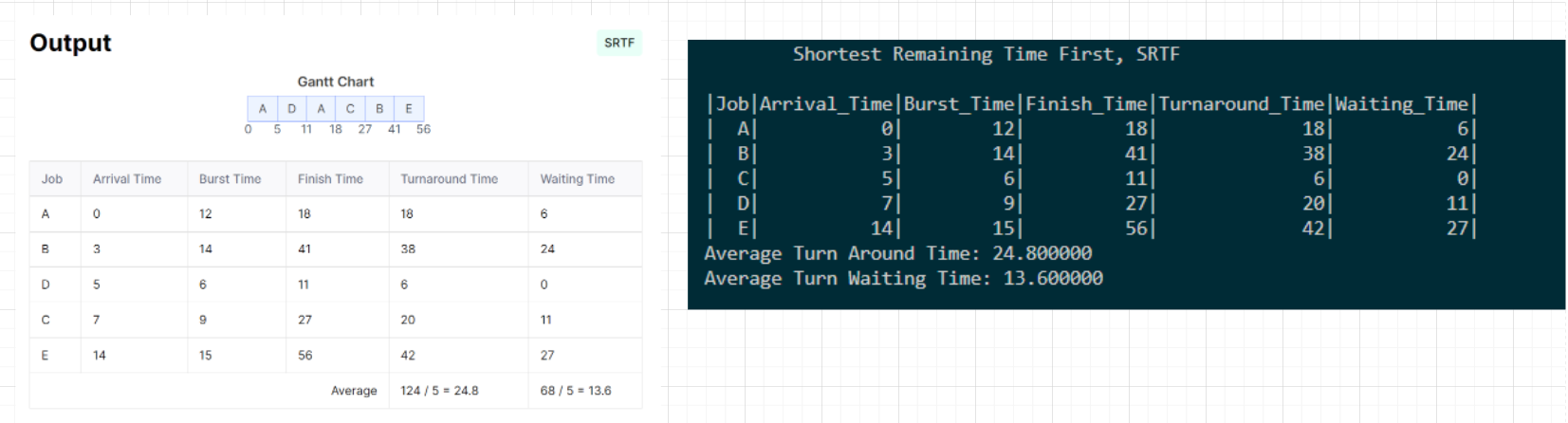
- Đầu tiên tạo kiểu dữ liệu tiến trình với các thuộc tính theo Lưu đồ trên.  


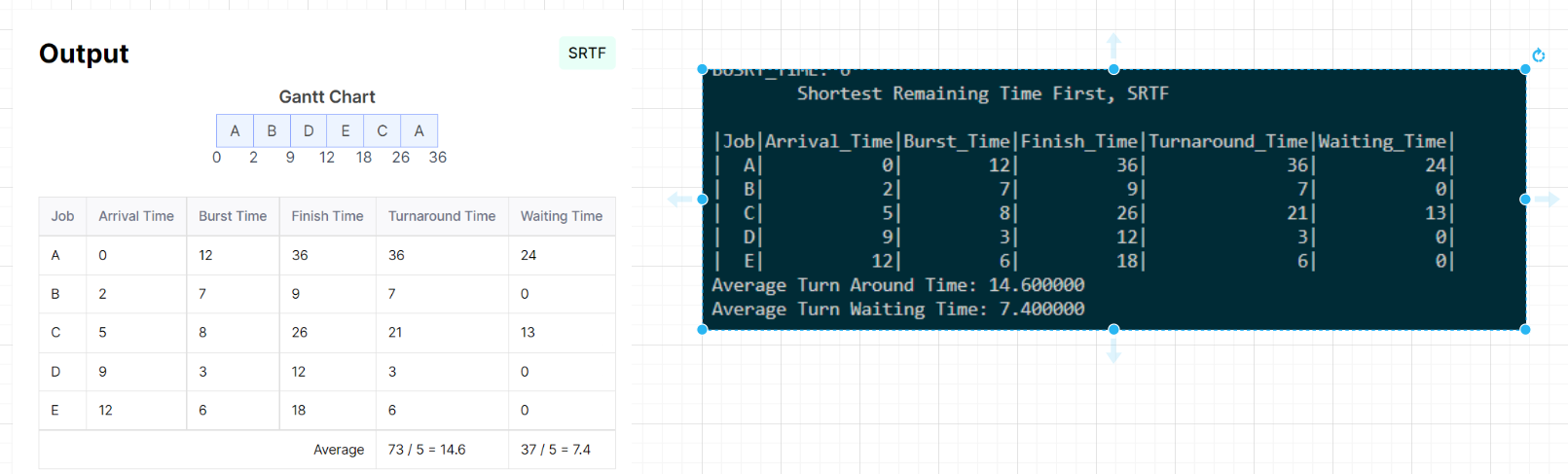
-Sau khi nhập dữ liệu cho mảng các tiến trình, ta khởi tạo các biến cần thiết cho vòng lặp  
  
- Khi mà các tiến trình chưa hoàn thành (isCompleted!=true):

+Ta sẽ sắp xếp lại các mảng theo thứ tự giảm dần theo RemainingBurstTime cứ sau mỗi vòng lặp, hàm sắp xếp này đồng thời cũng đẩy tiến trình có RemainingBurstTime ngắn nhất lên đầu mảng.  
 +Cứ sau mỗi vòng lặp ta giảm dần RemainingBurstTime và tằng dần current\_time   
 +Sẽ có một lệnh if nhằm chọn ra tiến trình đã thực thi xong (BURST\_TIME\_REMAINING == 0)

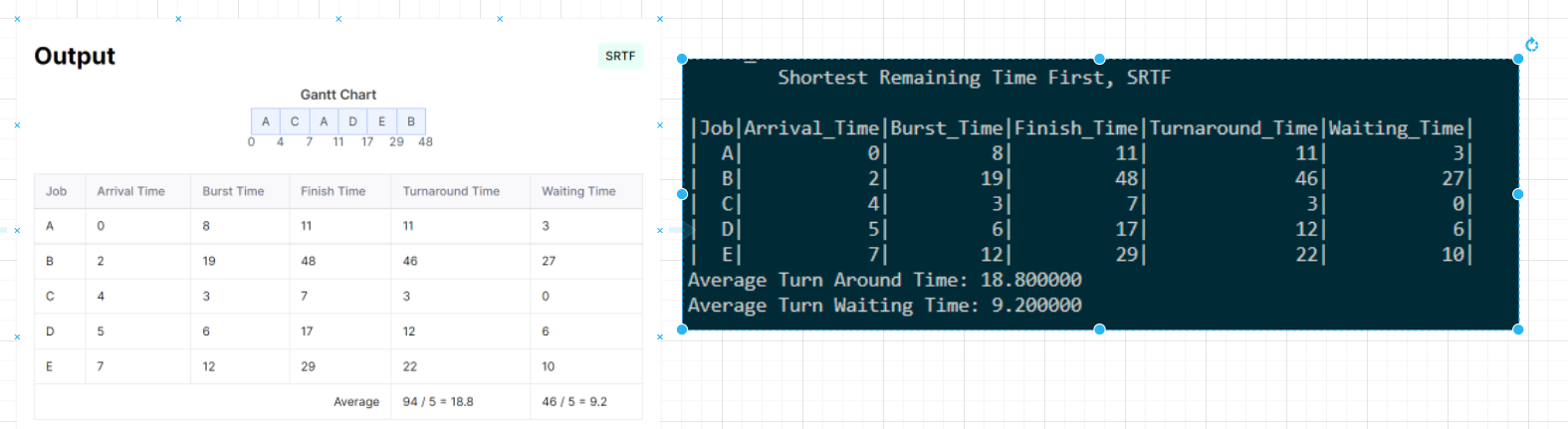
Và thực thi:   
  
-Sau khi tất cả các tiến trình hoàn thành, ta sẽ tính toán các chỉ số liên quan và in ra màn hình.

1. **Tính đúng đắn của giải thuật SRTF và code.**

****

Hình 4: Test Case 01 Chạy Bằng Tay Và Kết quả code****

Hình 6 Test Case 02 Chạy Bằng Tay Và Kết quả code

****

Hình 7:Test Case 03 Chạy Bằng Tay Và Kết quả code

# TASK 03: GIẢI THUẬT RR

**1.Lưu đồ giải thuật.**

**2.Code giải thuật.**

**3.Tính đúng đắn của giải thuật.**

**4. Tính đúng đắn của code.**