

**MỤC LỤC**

[**I. Đề bài**](#_heading=h.csmcu4nhhz4o)**………………………………………………………………………… 4**

[1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết…](#_heading=h.iqt3e0678syl)………………………………………………………………. 4

[2. Ứng dụng cây tìm kiếm nhị phân xây dựng lớp quản lý các đối tượng sinh viên., mỗi sinh viên gồm các thông tin](#_heading=h.ovyhfqkuir34)……………………………………… 4

[**II. Phân tích bài toán**](#_heading=h.46a41sx4d853)**………………………………………………………….. 5**

[1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết](#_heading=h.fds7j0trb1w2)…………………………………………………………………. 5

[2. Ứng dụng cây tìm kiếm nhị phân xây dựng lớp quản lý các đối tượng sinh viên., mỗi sinh viên gồm các thông tin](#_heading=h.dsdjhfecbj0i)……………………………………… 5

[**III. Cài đặt các lớp và hàm main bằng C++**](#_heading=h.wydjyiiudtvu)**..................................................... 6**

[1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết](#_heading=h.taahbqhyygfy)…………………………………………………………………. 6

[2. Tạo file bài 40.cpp](#_heading=h.gmico76o0w4w)………………………………………………………... 9

[3. Tạo file input.txt](#_heading=h.1yt5fvrq4jel)………………………………………………………… 17

[4. Chạy chương trình](#_heading=h.dmna4eb39nt6)………………………………………………………. 18

[**VI. Phân tích thời gian chạy từng phương thức có trong lớp**](#_heading=h.tbjat8cv2bza)**……………... 20**

[**V. Danh sách tài liệu tham khảo**](#_heading=h.rrz4qt9tdxjw)**…………………………………………….. 20**

# I. Đề bài

## 1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết.

## 2. Ứng dụng cây tìm kiếm nhị phân xây dựng lớp quản lý các đối tượng sinh viên., mỗi sinh viên gồm các thông tin:

+ MSV (MSV là duy nhất)

+ Họ tên

+ Tuổi

Lớp có các phương thức:

+ Đọc danh sách sinh viên từ tệp vào cây (*cấu trúc file lưu trữ danh sách sinh viên do người lập trình tự định nghĩa*)

+ Thêm 1 sinh viên vào cây, nếu sinh viên đã tồn tại thì thông báo lỗi.

+ Xóa 1 sinh viên trong lớp theo MSV. Nếu mã không có thì thông báo lỗi.

+ Cập nhật thông tin 1 sinh viên khi biết MSV và các thông tin cần cập nhật.

+ Tìm kiếm sinh viên theo MSV.

+ Hiển thị danh sách sinh viên lên màn hình.

+ Ghi toàn bộ sinh viên của cây vào file.

# II. Phân tích bài toán

## 1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết

- Cây tìm kiếm nhị phân được xây dựng dựa trên bài cài đặt cây tìm kiếm nhị phân mẫu đã được hướng dẫn trên lớp.

- Cài đặt danh sách liên kết kép gồm các hàm và thành phần sau:

* void update(node<T> \*&H, T x): Thêm phần tử vào cây
* void inorder(node<T> \*H, string p="\n"): In cây theo thứ tự
* T Max(node<T> \*H): Trả về giá trị lớn nhất trong cây
* void remove(node<T> \*&H, int x): Xóa phần tử trong cây
* node<T> \*search(node<T> \*H, int x): Tìm kiếm phần tử trong cây
* void print(node<T> \*H, int x): In phần tử trong cây
* void changeE(node<T> \*&H, int x, string a, string b, int c): Thay đổi giá trị phần tử trong cây

## 2. Ứng dụng cây tìm kiếm nhị phân xây dựng lớp quản lý các đối tượng sinh viên., mỗi sinh viên gồm các thông tin:

+ MSV (MSV là duy nhất)

+ Họ tên

+ Tuổi

Lớp có các phương thức tương ứng với các hàm:

+ Đọc danh sách sinh viên từ tệp vào cây (*cấu trúc file lưu trữ danh sách sinh viên do người lập trình tự định nghĩa*)

* + void INPUT(int &n, int &k, SVs \*&ds)

+ Thêm 1 sinh viên vào cây, nếu sinh viên đã tồn tại thì thông báo lỗi.

* + void ADD(int &n, int &k, SVs \*&ds)

+ Xóa 1 sinh viên trong lớp theo MSV. Nếu mã không có thì thông báo lỗi.

* + void DELETE(node<T> \*root)

+ Cập nhật thông tin 1 sinh viên khi biết MSV và các thông tin cần cập nhật.

* + void UPDATE(node<T> \*root)

+ Tìm kiếm sinh viên theo MSV.

* + void SEARCH(node<T> \*root)

+ Hiển thị danh sách sinh viên lên màn hình.

* + void inorder(root)

+ Ghi toàn bộ sinh viên của cây vào file.

* + void OUTPUT(int n, SVs \*ds)

# III. Cài đặt các lớp và hàm main bằng C++

Sau đây là toàn bộ bài làm được code trên Dev-C++

## 1. Cài đặt cấu trúc dữ liệu trừu tượng cây tìm kiếm nhị phân sử dụng danh sách liên kết

#include<bits/stdc++.h>

using namespace std;

template <class T>

struct node{

T elem;

node \*left, \*right;

node<T> (T e){

elem = e;

left = right = 0;

}

};

template <class T>

void update(node<T> \*&H, T x){

bool check = 1;

if(!H) H = new node<T> (x);

else if(x!=H->elem) update(x.getMSV()<H->elem.getMSV()?H->left:H->right, x);

else check = 0;

if(!check){

cout<<"Da ton tai MSV "<<x.getMSV()<<endl;

}

return;

}

template <class T>

void inorder(node<T> \*H, string p="\n"){

if(H){

inorder(H->left, p+"\t");

cout<<p;

cout<<H->elem;

inorder(H->right, p+"\t");

}

}

template <class T>

T Max(node<T> \*H){

if(!H) cout<<"\nKhong co du lieu";

return H->right?Max(H->right):H->elem;

}

template <class T>

void remove(node<T> \*&H, int x){

if(!H){

cout<<"Khong tim thay SV\n";

return;

}

if(H->elem.getMSV()!=x) return remove(x<H->elem.getMSV()?H->left:H->right,x);

if(!H->left) H=H->right;

else if(!H->right) H=H->left;

else{

H->elem=Max(H->left);

remove(H->left,H->elem.getMSV());

}

}

template <class T>

node<T> \*search(node<T> \*H, int x){

if(!H||H->elem.getMSV()==x) return H;

return H->elem.getMSV()>x?search(H->left, x):search(H->right, x);

}

template <class T>

void print(node<T> \*H, int x){

node<T> \*temp = search(H, x);

if(temp==NULL) cout<<"Khong tim thay SV\n";

else{

H = temp;

cout<<H->elem;

}

}

template <class T>

void changeE(node<T> \*&H, int x, string a, string b, int c){

node<T> \*temp = search(H, x);

if(temp == NULL) cout<<"Khong tim thay SV\n";

else{

H = temp;

H->elem.setName(a);

H->elem.setLop(b);

H->elem.setTuoi(c);

}

}

## 2. Tạo file bài 40.cpp

#include<bits/stdc++.h>

#include"tree.cpp"

using namespace std;

//Tao class SV

class SVs{

int MSV, tuoi;

string name, lop;

public:

friend istream& operator >> (istream&, SVs&);

friend ostream& operator << (ostream&, const SVs&);

friend bool operator != (SVs a, SVs b){

if(a.getMSV()==b.getMSV()) return 0;

return 1;

}

int getMSV(){

return this->MSV;

}

string getName(){

return this->name;

}

int getTuoi(){

return this->tuoi;

}

string getLop(){

return this->lop;

}

void setName(string name){

this->name = name;

}

void setMSV(int MSV){

this->MSV = MSV;

}

void setLop(string lop){

this->lop = lop;

}

void setTuoi(int tuoi){

this->tuoi = tuoi;

}

};

istream& operator >> (istream& is, SVs &a){

cout<<"\nTen SV: ";

cin.ignore();

getline(cin, a.name);

cout<<"MSV: ";

cin>>a.MSV;

cout<<"Lop: ";

cin.ignore();

getline(cin, a.lop);

cout<<"Tuoi: ";

cin>>a.tuoi;

return is;

}

ostream& operator << (ostream& os, const SVs& a){

cout<<"Ten SV: "<<a.name<<"; "<<"MSV: "<<a.MSV<<"; "<<"Lop: "<<a.lop<<"; "<<"Tuoi: "<<a.tuoi<<endl;

}

//Nhap

void INPUT(int &n, int &k, SVs \*&ds){

ifstream input("D:\\BTLfinal\\input.txt");

string x;

int a;

input>>k;

for(int i=n; i<n+k; i++){

input.ignore();

getline(input, x);

ds[i].setName(x);

input>>a;

ds[i].setMSV(a);

input.ignore();

getline(input, x);

ds[i].setLop(x);

input>>a;

ds[i].setTuoi(a);

}

}

//Xuat

void OUTPUT(int n, SVs \*ds){

ofstream output("D:\\BTLfinal\\output.txt");

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

output<<"SV"<<i+1<<endl;

output<<"Ten SV: "<<ds[i].getName()<<endl;

output<<"MSV: "<<ds[i].getMSV()<<endl;

output<<"Lop: "<<ds[i].getLop()<<endl;

output<<"Tuoi: "<<ds[i].getTuoi()<<endl;

}

output.close();

}

//Them SV

void ADD(int &n, int &k, SVs \*&ds){

cin>>k;

for(int i=n; i<n+k; i++){

cout<<"Nhap TT SV"<<i+1;

cin>>ds[i];

}

}

//Cap nhat thong tin SV

template <class T>

void UPDATE(node<T> \*root){

string a, b;

int c, x;

cin>>x;

cout<<"Ten SV: ";

cin.ignore();

getline(cin, a);

cout<<"Lop: ";

getline(cin, b);

cout<<"Tuoi: ";

cin>>c;

changeE(root, x, a, b, c);

}

//Xoa SV

template <class T>

void DELETE(node<T> \*root){

int x;

cin>>x;

remove(root, x);

}

//Tim kiem SV

template <class T>

void SEARCH(node<T> \*root){

int x;

cin>>x;

print(root, x);

}

//Menu

int Menu(){

cout<<"=================MENU================";

cout<<"\nNhap lua chon cua ban";

cout<<"\n1.Doc danh sach SV tu tep";

cout<<"\n2.Them SV";

cout<<"\n3.Xoa SV";

cout<<"\n4.Cap nhat thong tin SV";

cout<<"\n5.Tim kiem SV";

cout<<"\n6.Hien thi danh sach SV";

cout<<"\n7.Xuat danh sach SV";

cout<<"\n8.Thoat"<<endl;

cout<<"=====================================\n";

}

int ChonMenu(){

int n;

cout<<"Moi chon menu: ";

cin>>n;

if (n>0&&n<9)

return n;

else{

cout<<"Ban da chon sai hay chon lai\n";

return ChonMenu();

}

}

template <class T>

int Process(int n, int k, SVs \*ds, node<T> \*root){

while(true){

int z = ChonMenu();

switch(z){

case 1:{

cout<<"Doc danh sach SV...";

INPUT(n, k, ds);

for(int i=n; i<n+k; i++){

update(root, ds[i]);

}

n+=k;

cout<<"Successful!\n";

break;

}

case 2:{

cout<<"Them SV";

cout<<"\nNhap so luong SV: ";

ADD(n, k, ds);

for(int i=n; i<n+k; i++){

update(root, ds[i]);

}

n+=k;

break;

}

case 3:{

cout<<"Xoa SV";

cout<<"\nNhap MSV: ";

DELETE(root);

break;

}

case 4:{

cout<<"Cap nhat TT SV";

cout<<"\nNhap MSV: ";

UPDATE(root);

break;

}

case 5:{

cout<<"Tim kiem SV";

cout<<"\nNhap MSV: ";

SEARCH(root);

break;

}

case 6:{

cout<<"Danh sach SV\n";

inorder(root);

cout<<endl;

break;

}

case 7:{

cout<<"Xuat danh sach SV...";

OUTPUT(n, ds);

cout<<"Successful!\n";

break;

}

case 8:{

cout<<"Da ket thuc chuong trinh!";

exit(0);

break;

}

}

}

}

int main(){

int k, n = 0;

SVs \*ds;

ds = new SVs[1000000];

node<SVs> \*root = 0;

Menu();

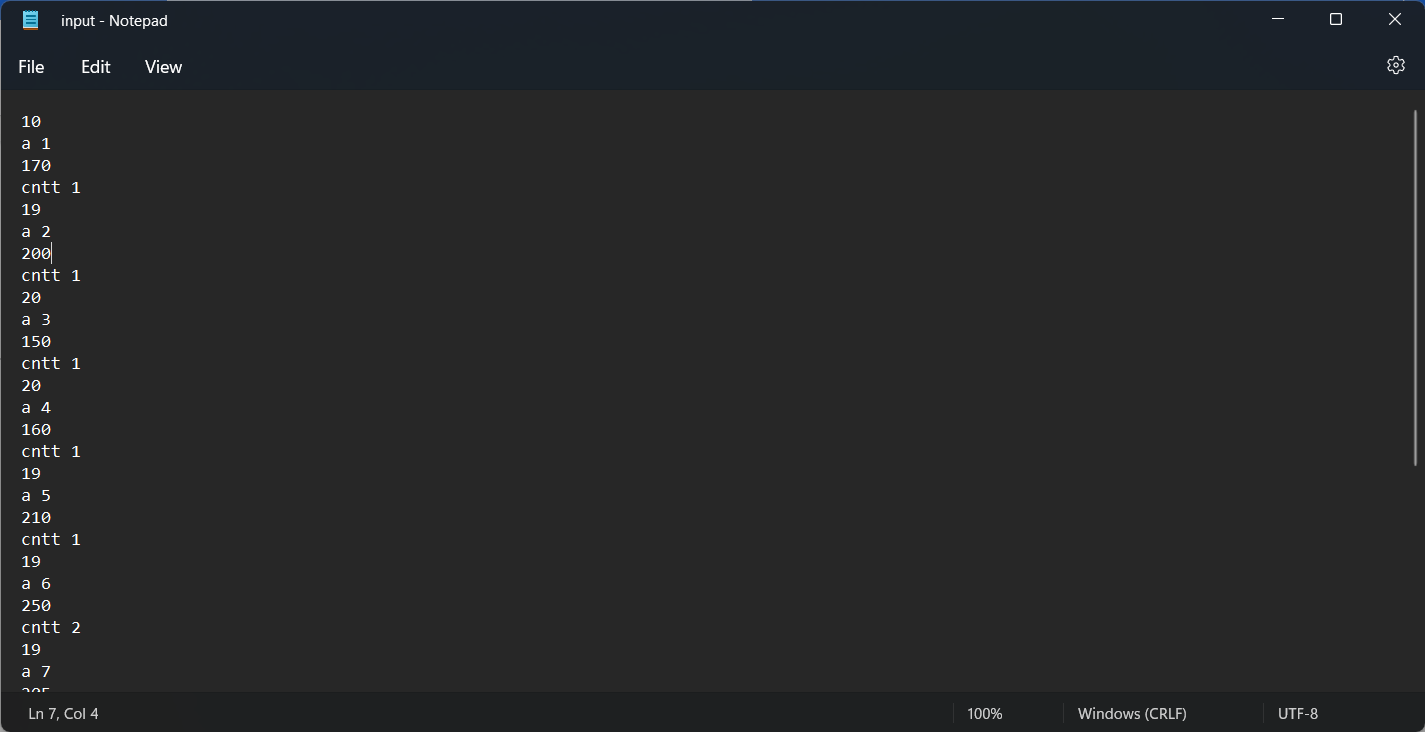
Process(n, k, ds, root);

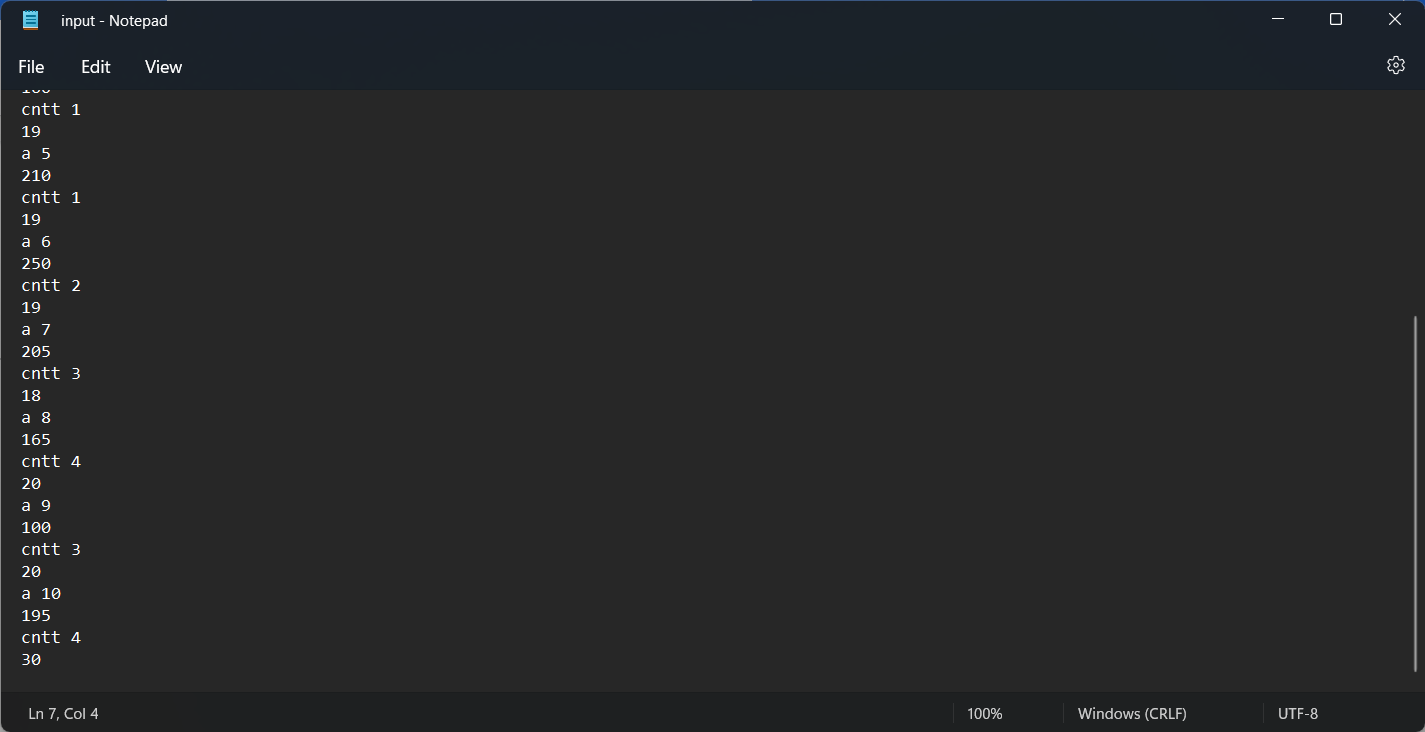
delete [] ds;

return 0;

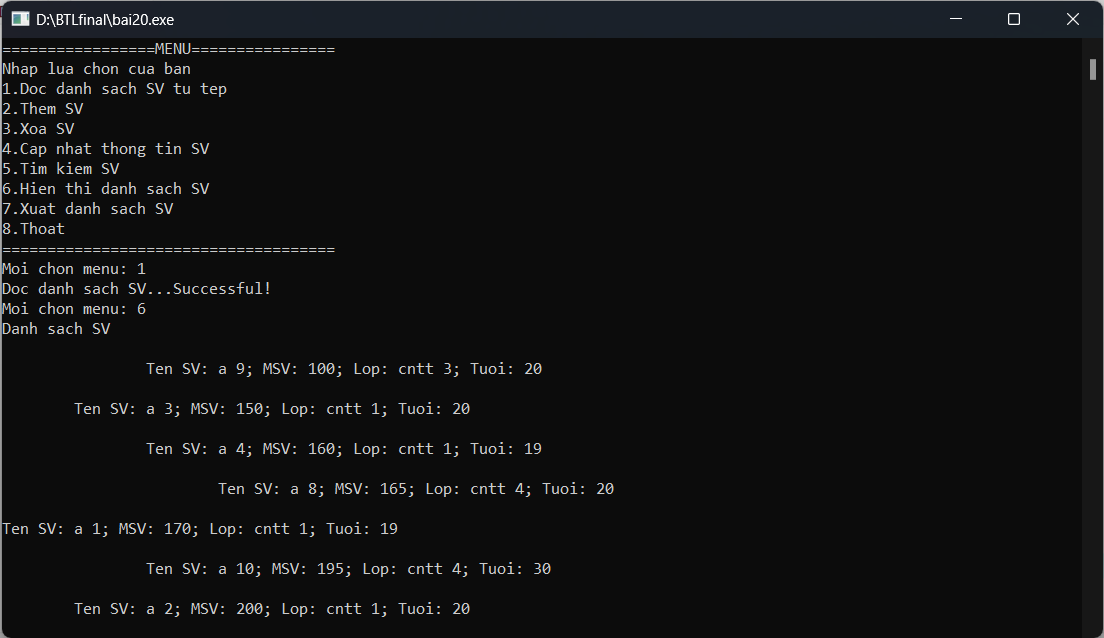
}

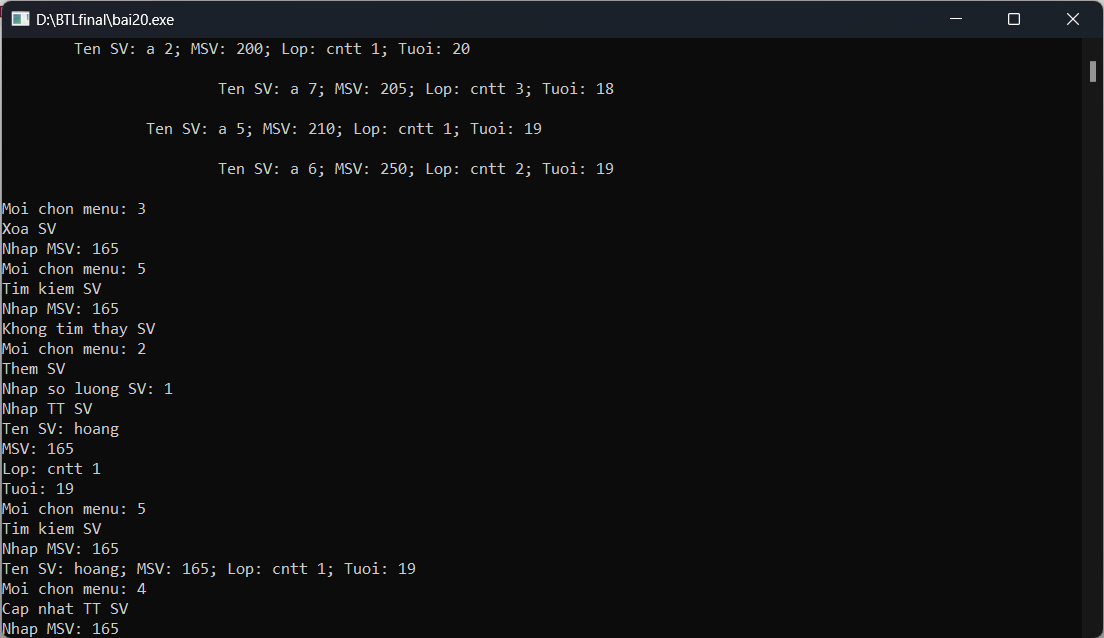
## 3. Tạo file input.txt

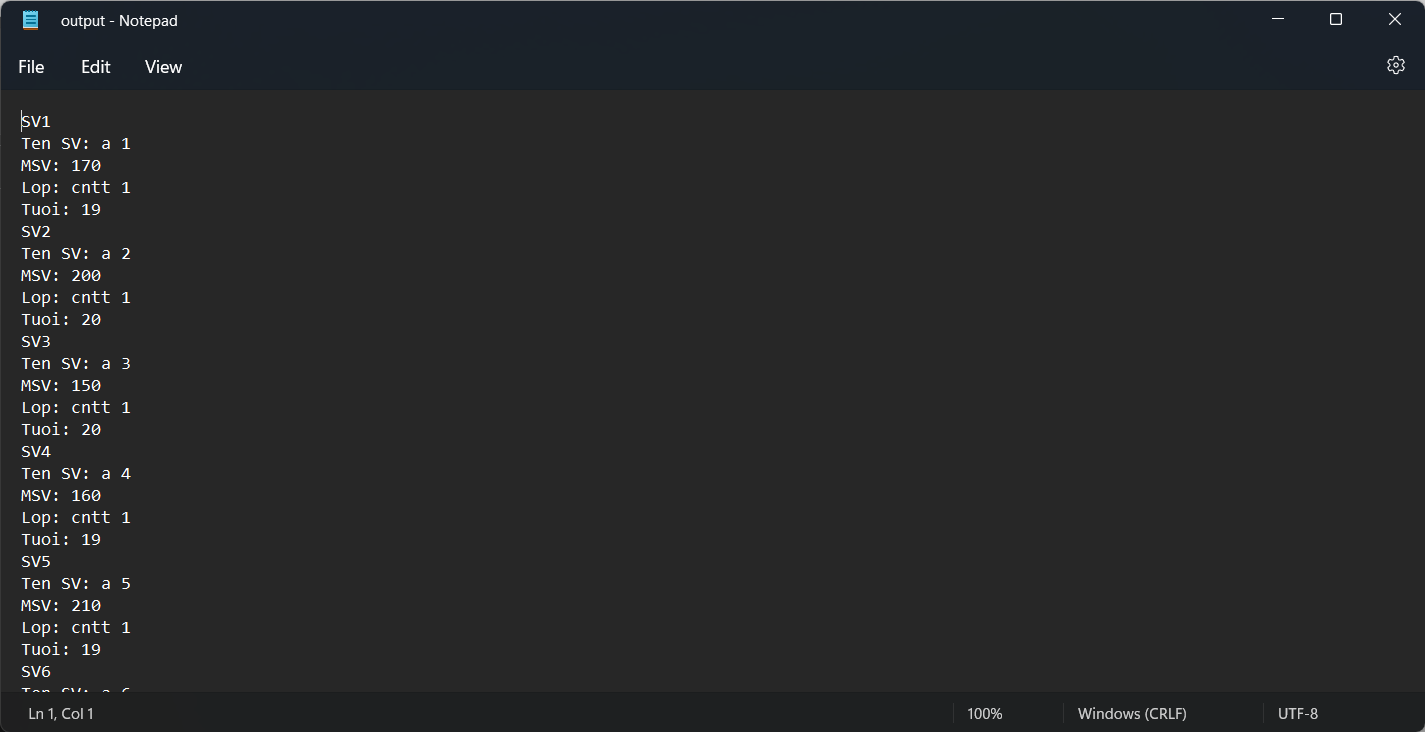
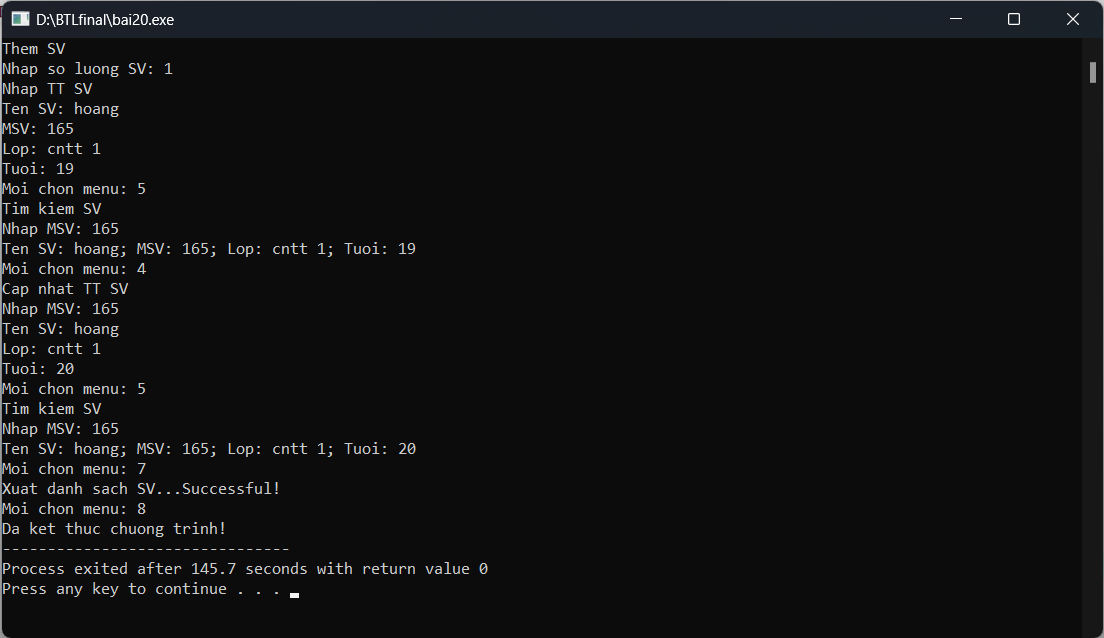


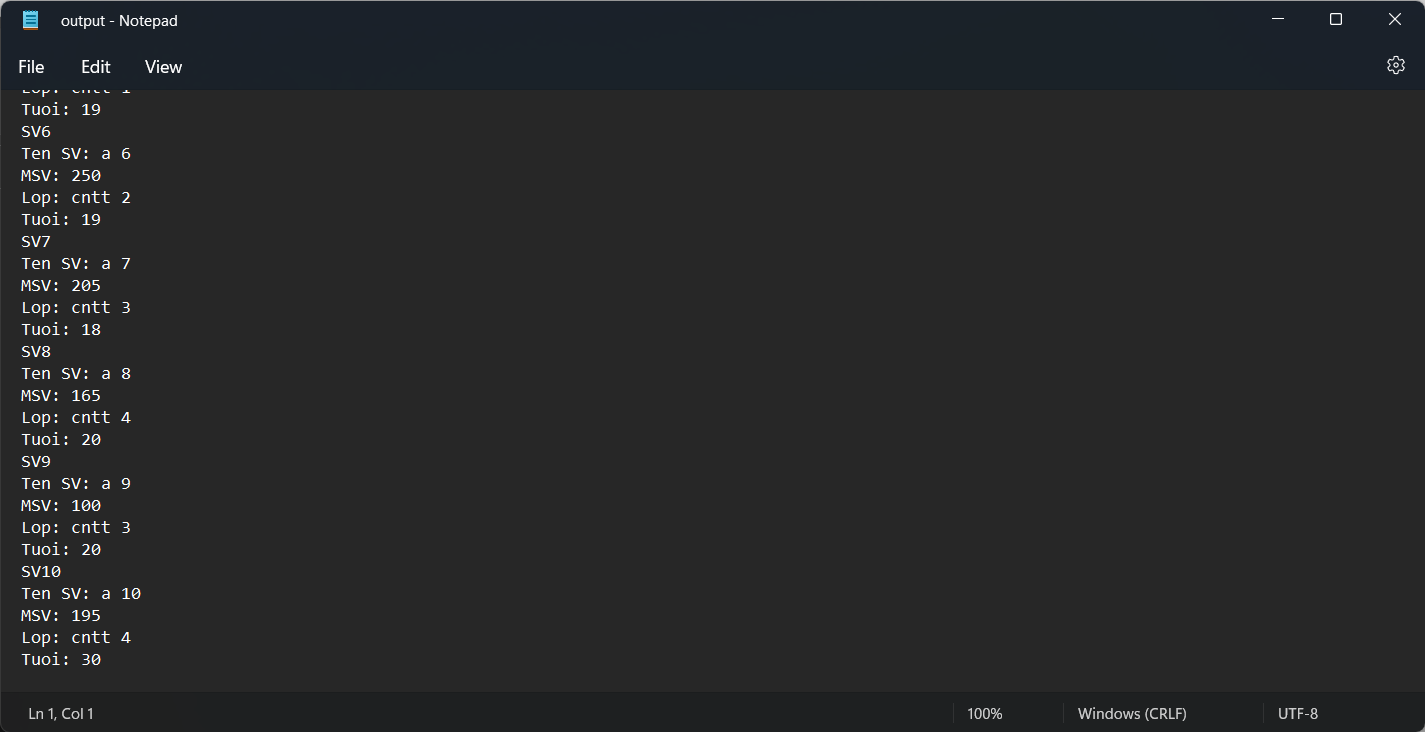


## 4. Chạy chương trình









# VI. Phân tích thời gian chạy từng phương thức có trong lớp

1. Cây nhị phân

- **struct** node: Thời gian chạy: T(n) = O(1)

- **void** update(node<T> \*&H, T x): Thời gian chạy: T(n) = O(logn)

- **void** inorder(node<T> \*H, string p="\n"): Thời gian chạy: T(n) = O(logn)

- T Max(node<T> \*H): Thời gian chạy: T(n) = O(logn)

- **void** remove(node<T> \*&H, int x): Thời gian chạy: T(n) = O(logn)

- node<T> \*search(node<T> \*H, int x): Thời gian chạy: T(n) = O(logn)

- **void** print(node<T> \*H, int x): Thời gian chạy: T(n) = O(logn)

- **void** changeE(node<T> \*&H, int x, string a, string b, int c): Thời gian chạy: T(n) = O(logn)

2. Bài 40

- **void** INPUT(int &n, int &k, SVs \*&ds): Thời gian chạy: T(n) = O(1)

- **void** OUTPUT(int n, SVs \*ds): Thời gian chạy: T(n) = O(1)

- **void** ADD(int &n, int &k, SVs \*&ds): Thời gian chạy: T(n) = O(1)

- **void** UPDATE(node<T> \*root): Thời gian chạy: T(n) = O(logn) vì sử dụng hàm **void** changeE(node<T> \*&H, int x, string a, string b, int c)

- **void** DELETE(node<T> \*root): Thời gian chạy: T(n) = O(logn) vì sử dụng hàm **void** remove(node<T> \*&H, int x)

- **void** SEARCH(node<T> \*root): Thời gian chạy: T(n) = O(logn) vì sử dụng hàm node<T> \*search(node<T> \*H, int x)

# V. Danh sách tài liệu tham khảo

- Một số bài được tham khảo từ bài giảng của giảng viên ThS. Phạm Xuân Tích

- Hàm search trong thiết kế cây tìm kiếm nhị phân - [Search in a Binary Search Tree - LeetCode](https://leetcode.com/problems/search-in-a-binary-search-tree/)

- Độ phức tạp của thuật toán với cây tìm kiếm nhị phân - [Complexity of different operations in Binary tree, Binary Search Tree and AVL tree - GeeksforGeeks](https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/)