CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

A. Các thuật toán sắp xếp:

Cho mảng một chiều a[5] các số tự nhiên. Các mô tả dưới đây đều được thực hiện trên mảng một chiều này.

i	0	1	2	3	4
a[i]	23	12	7	28	-5

1. Thuật toán sắp xếp nổi bọt (Bubble Sort):

- ***Ý tưởng:*** Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn về đầu. Sau đó ở bước tiếp theo không xét phần tử đó nữa. Do vậy lần xử lí thứ i sẽ có i sẽ có vị trí đầu dãy là i. Lặp lại xử lí trên cho đến khi không còn cặp phần tử nào được xét.
- Cài đặt đối với (số nguyên, số thực, kí tự):

```
void BubbleSort(int a[],int n)
{
	for (int i=0;i<n-1;i++)
		for (int j=n-1;j>i;j--)
			if (a[j-1]>a[j]) Swap(int,a[j],a[j-1]);
}
```

- Cài đặt đối với cấu trúc:

```
void BubbleSort(SV a[],int n)
{
	for (int i=0;i<n-1;i++)
		for (int j=n-1;j>i;j--)
			if (strcmp(a[j-1].ten,a[j].ten)<0) Swap(SV,a[j],a[j-1]);
}
```

- ***Ghi chú:*** Các cú pháp (**in đậm, gạch chân nét liền**) được hiểu là chỗ cần thay đổi khi đổi sang các dạng bài khác. Mặc định trong tài liệu này các thuật toán sẽ được sắp xếp tăng dần. Trong cấu trúc sẽ sử dụng cấu trúc Sinh viên (SV) và sắp xếp theo trường "tên", hàm strcmp(char *x, char *y) được định nghĩa trong thư viện **'stdlib.h'**. Hàm tráo đổi được định nghĩa macro ở đầu chương trình (sau khai báo thư viện) như sau:

```
#define Swap(type,a,b) {type tmp=a; a=b; b=tmp;}
```

- Minh hoạ thuật toán:

Lần duyệt i, j	**Dữ liệu**				
Ban đầu	23	12	7	28	-5
0, 4	23	12	7	-5	28
0, 3	23	12	-5	7	28
0, 2	23	-5	12	7	28
0, 1	-5	23	12	7	28
1, 3	-5	23	7	12	28
1, 2	-5	7	23	12	28
2, 4	-5	7	12	13	28

3, 4	-5	7	12	13	28
Kết quả	**-5**	**7**	**12**	**13**	**28**

2. Thuật toán sắp xếp đổi chỗ trực tiếp (Interchange Sort):

- ***Ý tưởng:*** Xuất phát từ đầu dãy, lần lượt tìm các phần tử còn lại không thoả thứ tự với phần tử đang xét. Với mỗi phần tử tìm được mà không thoả thứ tự, thực hiện hoán vị để thoả thứ tự. Lặp lại tương tự với các phần tử tiếp theo.
- Cài đặt đối với (số nguyên, số thực, kí tự):

```
void InterchangeSort(int a[],int n)
{
        for (int i=0;i<n-1;i++)
                for (int j=i+1;j<n;j++)
                        if (a[j]<a[i]) Swap(int,a[i],a[j]);
}
```

- Cài đặt đối với cấu trúc:

```
void InterchangeSort(SV a[],int n)
{
        for (int i=0;i<n-1;i++)
                for (int j=i+1;j<n;j++)
                        if (strcmp(a[j].ten,a[i].ten)>0) Swap(SV,a[i],a[j]);
}
```

- Minh hoạ thuật toán:

Lần duyệt i, j	Dữ liệu				
Ban đầu	23	12	7	28	-5
0, 1	12	23	7	28	-5
0, 2	7	23	12	28	-5
0, 4	-5	23	12	28	7
1, 2	-5	12	23	28	7
1, 4	-5	7	23	28	12
2, 4	-5	7	12	28	23
3, 4	-5	7	12	23	28
Kết quả	**-5**	**7**	**12**	**23**	**28**

3. Thuật toán sắp xếp chọn (Selection Sort):

- ***Ý tưởng:*** Chọn phần tử nhỏ nhất trong n phần tử trong dãy hiện hành ban đầu. Đưa phần tử này về đầu dãy hiện hành. Xem dãy hiện hành chỉ còn n-1 phần tử của dãy hiện hành ban đầu: bắt đầu từ vị trí 2, lặp lại quá trình trên cho đến khi dãy hiện hành chỉ còn 1 phần tử.
- Cài đặt cho (số nguyên, số thực, kí tự):

```
void SelectionSort(int a[],int n)
{
        for (int i=0;i<n-1;i++)
        {
                int minpos=i;
                for (int j=i+1;j<n;j++)
```

```
                if (a[j]<a[minpos]) minpos=j;
            Swap(int,a[minpos],a[i]);
        }
}
```

- Cài đặt cho cấu trúc:

```
void SelectionSort(SV a[],int n)
{
        for (int i=0;i<n-1;i++)
        {
            int minpos=i;
            for (int j=i+1;j<n;j++)
                if (strcmp(a[j].ten,a[minpos].ten)>0) minpos=j;
            Swap(SV,a[minpos],a[i]);
        }
}
```

- Minh hoạ thuật toán:

Lần duyệt i, minpos	Dữ liệu				
Ban đầu	23	12	7	28	-5
0, 4	-5	12	7	28	23
1, 2	-5	7	12	28	23
2, 2	-5	7	12	28	23
3, 4	-5	7	12	23	28
Kết quả	**-5**	**7**	**12**	**23**	**28**

4. Thuật toán sắp xếp kiểu chèn (Insertion Sort):

- ***Ý tưởng:*** Lần lượt chèn các nút vào danh sách đã có thứ tự.
- Cài đặt cho (số nguyên, số thực, kí tự):

```
void InsertionSort(int a[],int n)
{
        int pos; int x;
        for (int i=1;i<n;i++)
        {
            x=a[i]; pos=i-1;
            while (pos>=0 && a[pos]>x)
            {
                a[pos+1]=a[pos];
                pos--;
            }
            a[pos+1]=x;
        }
}
```

- Cài đặt cho cấu trúc:

```
void InsertionSort(SV a[],int n)
{
        int pos; char x[30];
```

```
        for (int i=1;i<n;i++)
        {
                strcpy(x,a[i].ten); pos=i-1;
                while (pos>=0 && strcmp(a[pos].ten,x)<0)
                {
                        a[pos+1]=a[pos];
                        pos--;
                }
                strcpy(a[pos+1].ten,x);
        }
}
```

- Minh hoạ thuật toán:

Lần duyệt chèn	Nút chèn	Dữ liệu				
Ban đầu		23	12	7	28	-5
0	23	23				
1	12	12	23			
2	7	7	12	23		
3	28	7	12	23	28	
4	-5	-5	7	12	23	28
Kết quả		**-5**	**7**	**12**	**23**	**28**

B. Danh sách liên kết đơn:

1. Danh sách liên kết đơn 1 trỏ quản lí số nguyên:

- Bước 1: Khai báo thư viện:

```
#include <stdio.h>
#include <conio.h>
```

- Bước 2: Khai báo cấu trúc của 1 node:

```
struct node
{
    int info;
    struct node *pnext;
}; // nhớ có dấu ; nhé ^^
```

```
typedef struct node NODE;
```

- Bước 3: Khởi tạo danh sách:

```
void Init(NODE *&phead)
{
    phead=NULL;
}
```

- Bước 4: Hàm tạo node:

```
NODE *GetNode(int x)
{
        NODE *p=new NODE;
        if (p==NULL) return NULL;
        p->info=x;
```

```
    p->pnext=NULL;
    return p;
}
```

- Bước 5: Hàm thêm 1 node vào đầu DSLK:

```
void AddHead(NODE *&phead, NODE *p)
{
        if (phead==NULL) phead=p;
        else
        {
            p->pnext=phead;
            phead=p;
        }
}
```

- Bước 6: Hàm Input. Lưu ý kĩ thuật "nhập đến khi nhập số 0 thì dừng".

```
void Input(NODE *&phead)
{
        Init(phead); int x;
        do
        {
                printf("Nhap gia tri: ");
                scanf("%d",&x);
                if (x!=0)
                {
                        NODE *p=GetNode(x);
                        if (p!=NULL) AddHead(phead,p);
                }
        }
        while (x!=0);
}
```

- Bước 7: Hàm Output xuất DSLK:

```
void Output(NODE *phead)
{
    for (NODE *p=phead;p!=NULL;p=p->pnext)
            printf("%d\t",p->info);
}
```

- Bước 8: Nếu yêu cầu 1 trong các hàm sau thì ghi ra:

Hàm tính tổng:

```
int Sum(NODE *phead)
{
    int s=0; NODE *p=phead;
    while (p!=NULL)
        {
            s+=p->info;
            p=p->pnext;
    }
```

```
    return s;
}
```

- Hàm tính tổng lẻ (Tổng chẵn tương tự):

```
int OddSum(NODE *phead)
{
        int s=0; NODE *p=phead;
        while (p!=NULL)
        {
            if (p->info%2!=0) s+=p->info;
            p=p->pnext;
        }
    return s;
}
```

- Hàm tìm giá trị lớn nhất:

```
int FindMax(NODE *phead)
{
        int m=phead->info; NODE *p=phead;
        while (p!=NULL)
        {
            if (p->info > m) m=p->info;
            p=p->pnext;
        }
        return m;
}
```

- Thêm nút p vào sau nút q:

```
void AddAfter(NODE *&phead, NODE *p, NODE *q)
{
        if (q!=NULL)
        {
                    p->pnext=q->pnext;
                    q->pnext=p;
        }
        else AddHead(phead,p);
}
```

- Tìm một phần tử trong DSLK:

```
NODE *Search(NODE *phead,int x)
{
    NODE *p=phead;
    while (p!=NULL && p->info!=x) p=p->pnext;
    return p;
}
```

- Sắp xếp DSLK tăng dần:

```
void InterchangeSort(NODE *phead)
{
    for (NODE *p=phead;p->pnext!=NULL;p=p->pnext)
```

```
        for (NODE *q=p->pnext;q!=NULL;q=q->pnext)
            if (q->info < p->info) Swap(int,q->info,p->info);
}
```

- Bước 9: Chương trình chính:

```
int main()
{
    clrscr(); //xoá màn hình
    NODE *phead; //khai báo biến phead để quản lí DSLK
    Input(phead);
    printf("DSLK vua nhap:\n");
    Output(phead);
    //thêm 1 phần tử có giá trị x vào sau phần tử có giá trị y
    int x,y;
    printf("Nhap gia tri can them: "); scanf("%d",&x);
    printf("Nhap vi tri can them sau: "); scanf("%d",&y);
    NODE *q=Search(phead,y);
    NODE *p=GetNode(x);
    AddAfter(phead,p,q);
    printf("DSLK sau khi chen:\n");
    Output(phead);
    //sắp xếp tăng dần
    InterchangeSort(phead);
    printf("DSLK sau khi sap xep:\n");
    Output(phead);
    getch(); //dừng chương trình
    return 0;
}
```

2. Danh sách liên kết đơn 1 trỏ quản lí số thực, kí tự: (thay những chỗ (**in đậm, gạch chân nét liền)** bằng kiểu dữ liệu phù hợp).

3. Danh sách liên kết đơn 1 trỏ quản lí cấu trúc Sinh viên (các cấu trúc khác tương tự):

- Bước 1: Khai báo các thư viện cần thiết và macro tráo đổi dữ liệu:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#define Swap(type,a,b) {type tmp=a; a=b; b=tmp;}
```

- Bước 2: Khai báo cấu trúc sinh viên:

```
struct student
{
    char id[11];
    char name[31];
    float mark;
}; //nhớ có dấu ; nhé ^^
```

```
typedef struct student SV;
```

- Bước 3: Khai báo cấu trúc Node:

```
struct node
{
    SV info;
    struct node *pnext;
}; //nhớ có dấu ; nhé ^^
typedef struct node NODE;
```

- Bước 4: Viết hàm khởi tạo DSLK:

```
void Init(NODE *&phead)
{
    phead=NULL;
}
```

- Bước 5: Viết hàm tạo Node:

```
NODE *GetNode(SV x)
{
    NODE *p=new NODE;
    if (p!=NULL)
    {
            p->info=x;
            p->pnext=NULL;
    }
    return p;
}
```

- Bước 6: Viết hàm thêm 1 node vào đầu DSLK:

```
void AddHead(NODE *&phead,NODE *p)
{
    if (phead==NULL) phead=p;
    else
    {
            p->pnext=phead;
            phead=p;
    }
}
```

- Bước 7: Viết hàm nhập 1 sinh viên:

```
void Enter1Stu(SV &x)
{
    printf("Nhap MSSV: ");
    fflush(stdin);
    gets(x.id);
    printf("Nhap ho ten: ");
    fflush(stdin);
    gets(x.name);
    float tmp;
```

```
    printf("Nhap diem trung binh: ");
    scanf("%f",&tmp);
    x.mark=tmp;
}
```

- Bước 8: Viết hàm Input. Lưu ý kĩ thuật "**hỏi xem người dùng có muốn nhập tiếp hay không**":

```
void Input(NODE *&phead)
{
    SV x; char ch;
    Init(phead);
    do
    {
            printf("\nNhap 1 sinh vien:\n");
            Enter1Stu(x);
            NODE *p=GetNode(x);
            if (p!=NULL) AddHead(phead,p);
            printf("\n----------\n");
            printf("Ban co muon tiep tuc? (Y/N): ");
            ch=getche();
    }
    while (ch!='n' && ch!='N');
}
```

- Bước 9: Viết hàm xuất 1 sinh viên:

```
void Display1Stu(SV x)
{
    printf("%s\t%s\t%.2f\n",x.id,x.name,x.mark);
}
```

- Bước 10: Viết hàm Output xuất DSLK:

```
void Output(NODE *phead)
{
    printf("MSSV\tTEN\tDIEM\n");
    for (NODE *p=phead;p!=NULL;p=p->pnext)
            Display1Stu(p->info);
}
```

- Bước 11: Các hàm liên quan, nếu đề yêu cầu hàm nào thì viết hàm đó vào:

Tìm điểm cao nhất của tất cả các sinh viên:

```
float FindMax(NODE *phead)
{
    float max=phead->info.mark;
    NODE *p=phead;
    while (p!=NULL)
    {
            if (p->info.mark > max) max=p->info.mark;
            p=p->pnext;
    }
```

```
    return max;
}
```

- Sắp xếp tăng dần theo tên:

```
void SelectionSort(NODE *phead)
{
    for (NODE *i=phead;i->pnext->pnext!=NULL;i=i->pnext)
    {
        NODE *minpos=i;
        for (NODE *j=i->pnext;j!=NULL;j=j->pnext)
            if (strcmp(j->info.name,minpos->info.name)<0) minpos=j;
                Swap(SV,minpos->info,i->info);
    }
}
```

- Sắp xếp tăng dần theo MSSV (dùng Bubble Sort nên lưu ý có hàm PreNode(NODE *p) trả về địa chỉ của NODE trước NODE p):

```
NODE *PreNode(NODE *phead,NODE *p)
{
    NODE *q;
    if (p==phead) return NULL;
    q=phead;
    while (q!=NULL && q->pnext!=p) q=q->pnext;
    return q;
}

void BubbleSort(NODE *phead)
{
    NODE *q;
    for (NODE *i=phead;i->pnext!=NULL;i=i->pnext)
        if (i->pnext==NULL) q=i;
    for (i=phead;i->pnext->pnext!=NULL;i=i->pnext)
        for (NODE *j=q;i->pnext!=j;j=PreNode(phead,j))
            {
                NODE *w=PreNode(phead,j);
                if (w->info.id>j->info.id) Swap(SV,j->info,w->info);
            }
}
```

- Bước 12: Chương trình chính:

```
int main()
{
    clrscr();
    NODE *phead;
    Input(phead);
    printf("\nDanh sach lien ket vua nhap la:\n");
    Output(phead);
    printf("\nDiem trung binh lon nhat: %.2f\n",FindMax(phead));
```

```
    SelectionSort(phead);
    printf("Danh sach sau khi sap xep theo ten:\n");
    Output(phead);
    BubbleSort(phead);
    printf("\nDanh sach sau khi sap xep theo mssv:\n");
    Output(phead);
    getch();
    return 0;
}
```

4. Danh sách liên kết đơn 2 trỏ quản lí số nguyên (số thực, kí tự làm tương tự):

- Bước 1: Khai báo các thư viện cần thiết, kiểu dữ liệu (int, float, char):

```
#include <stdio.h>
#include <conio.h>
typedef int KDL;
```

- Bước 2: Khai báo cấu trúc Node:

```
typedef struct node
{
    KDL info;
    struct node *pnext;
} NODE;
```

- Bước 3: Khai báo cấu trúc List (2 trỏ):

```
typedef struct list
{
    NODE *phead;
    NODE *ptail;
} LIST;
```

- Bước 4: Viết hàm khởi tạo DSLK:

```
void Init(LIST &l)
{
    l.phead=NULL;
    l.ptail=NULL;
}
```

- Bước 5: Viết hàm tạo Node:

```
NODE *GetNode(int x)
{
    NODE *p=new NODE;
    if (p==NULL) return NULL;
    p->info=x;
    p->pnext=NULL;
    return p;
}
```

- Bước 6: Viết hàm thêm 1 node vào đầu DSLK:

```
void AddHead(LIST &l,NODE *p)
```

```
{
    if (l.phead==NULL)
    {
            l.phead=p;
            l.ptail=p;
    }
    else
    {
            p->pnext=l.phead;
            l.phead=p;
    }
}
```

- Bước 7: Viết hàm thêm 1 node vào cuối DSLK:

```
void AddTail(LIST &l,NODE *p)
{
    if (l.phead==NULL)
    {
            l.phead=p;
            l.ptail=p;
    }
    else
    {
            l.ptail->pnext=p;
            l.ptail=p;
    }
}
```

- Bước 8 (nếu đề yêu cầu): Viết hàm thêm node p vào sau node q:

```
void AddAfter(LIST &l,NODE *p,NODE *q)
{
    if (q!=NULL)
    {
            p->pnext=q->pnext;
            q->pnext=p;
    }
    else AddHead(l,p);
}
```

- Bước 9: Viết hàm Input nhập DSLK (lưu ý kĩ thuật "**nhập đến khi nhập số 0 thì dừng**" đã trình bày ở phần trước):

```
void Input(LIST &l)
{
    int n;
    printf("Nhap n = ");
    scanf("%d",&n);
    Init(l);
    for (int i=1;i<=n;i++)
```

```
    {
        KDL x;
        printf("Nhap x = ");
        scanf("%d",&x);
        NODE *p=GetNode(x);
        if (p!=NULL) AddHead(l,p);
    }
}
```

- Bước 10: Viết hàm Output xuất DSLK:

```
void Output(LIST l)
{
    for (NODE *p=l.phead;p!=NULL;p=p->pnext)
        printf("%d\t",p->info);
}
```

- Bước 11: Viết các hàm mà đề yêu cầu (tìm max, tìm min, sắp xếp,… như các hàm đã trình bày ở phần trước).
- Bước 12: Chương trình chính:

```
int main()
{
    clrscr();
    LIST l;
    Input(l);
    printf("DSLK vua nhap:\n");
    Output(l);
    getch();
    return 0;
}
```

5. Danh sách liên kết đơn 2 trỏ quản lí Sinh viên (các cấu trúc khác làm tương tự):

- Bước 1: Khai báo các thư viện cần thiết.
- Bước 2: Khai báo các cấu trúc:

```
struct sinhvien
{
    char mssv[11];
    char ten[31];
    float diem;
};
typedef struct sinhvien SV;
struct node
{
    SV info;
    struct node *pnext;
};
typedef struct node NODE;
struct list
```

```
{
    NODE *phead;
    NODE *ptail;
};
typedef struct list LIST;
```

- Bước 3: Viết hàm Init, GetNode, AddHead, AddTail.
- Bước 4 (nếu đề yêu cầu): Viết hàm thêm node p vào sau node q.
- Bước 5: Viết hàm Nhap1SV, Xuat1SV.
- Bước 6: Viết hàm Input, Output.
- Bước 7: Viết các hàm mà đề bài yêu cầu.
- Bước 8: Chương trình chính:

```
int main()
{
    clrscr();
    LIST l;
    Input(l);
    printf("DSLK vua nhap:\n");
    Output(l);
    getch();
    return 0;
}
```

C. Cây nhị phân tìm kiếm (Binary Search Tree – BST):

1. Vẽ cây nhị phân:

Lần lượt thêm các NODE mà đề yêu cầu vào cây BST, **đảm bảo** các yêu cầu sau:

- Nội dung của tất cả các nút thuộc nhánh **cây con bên trái** đều **nhỏ hơn** nội dung của **nút gốc**.
- Nội dung của tất cả các nút thuộc nhánh **cây con bên phải** đều **lớn hơn** nội dung của **nút gốc**.
- Cây con bên trái và bên phải cũng tự thân hình thành hai cây nhị phân tìm kiếm.

2. Vẽ lại cây nhị phân khi xoá đi một NODE:

Nếu nút bị xoá là **nút lá**, tiến hành **xoá nút đó bình thường**. Ngược lại, nếu đó **không là nút lá**:

- Nếu nút cần xoá nằm bên **cây con bên trái** ➔ **Xoá nút**, **chọn nút phải nhất** (lớn nhất) để gắn lại vào cây.
- Nếu nút cần xoá nằm bên **cây con bên phải** ➔ **Xoá nút**, **chọn nút trái nhất** (nhỏ nhất) để gắn lại vào cây.

3. Các cách duyệt cây BST:

- Duyệt **NLR**: đầu tiên là **nút gốc**, sau đó duyệt **cây con bên trái**, rồi **cây con bên phải**.
- Duyệt **LNR**: duyệt **cây con bên trái**, **nút gốc**, rồi đến **cây con bên phải** (thứ tự tăng dần trong cây BST).

4. Các hàm với cây BST:

- Khai báo cây BST:

```
struct node
{
    KDL info;
    struct node *pleft;
    struct node *pright;
};
typedef struct node NODE;
typedef struct NODE *TREE;
```

- Khởi tạo cây BST:

```
void Init(TREE &t)
{
    t=NULL;
}
```

- Tạo nút cho cây:

```
NODE *GetNode(KDL x)
{
    NODE *p=new NODE;
    if (p==NULL) return NULL;
    else
    {
            p->info=x;
            p->pleft=NULL;
            p->pright=NULL;
    }
    return p;
}
```

- Thêm nút vào cây:

```
int InsertNode(TREE &t,int x)
{
    if (t!=NULL)
    {
            if (t->info < x) return InsertNode(t->pright,x);
            if (t->info > x) return InsertNode(t->pleft,x);
            return 0;
    }
    t=GetNode(x);
    if (t==NULL) return -1;
    return 1;
}
```

➢ *Các giá trị trả về của hàm:*

❖ **Giá trị 1**: Thêm thành công.
❖ **Giá trị 0**: Trùng với khoá 1 nút có sẵn trong cây.
❖ **Giá trị -1**: Không đủ bộ nhớ.

- Hàm nhập vào 1 cây:

```
void Input(TREE &t)
{
    int x;
    Init(t);
    printf("Nhan so 0 de dung!\n");
    do
    {
            printf("Nhap x = ");
            scanf("%d",&x);
            if (x!=0) InsertNode(t,x);
            else printf("Ket thuc nhap!\n");
    }
    while (x!=0);
}
```

- Hàm xuất cây:

```
void Output(TREE t) //LNR
{
    if (t==NULL) return;
    Output(t->pleft);
    printf("%d\t",t->info);
    Output(t->pright);
    return;
}
```

Cấu trúc chung các dạng hàm sau: Tính giá trị cần tìm ở **nhánh bên trái**, sau đó là **nhánh bên phải**, rồi **xét tại nút gốc**.

- Tính tổng các nút:

```
long Sum(TREE t)
{
    if (t==NULL) return 0;
    long a=Sum(t->pleft);
    long b=Sum(t->pright);
    return (a+b+t->info);
}
```

- Tính tổng các số chẵn (tổng số lẻ, tổng dương ,tổng âm làm tương tự, chỉ **thay đổi điều kiện tại nút gốc**):

```
long EvenSum(TREE t)
{
    if (t==NULL) return 0;
    long a=EvenSum(t->pleft);
    long b=EvenSum(t->pright);
    if (t->info%2==0) return (a+b+t->info);
    return (a+b);
}
```

- Đếm số nút:

```
int CountNode(TREE t)
{
    if (t==NULL) return 0;
    int a=CountNode(t->pleft);
    int b=CountNode(t->pright);
    return (a+b+1);
}
```

- Tính chiều cao của cây:

```
int Height(TREE t)
{
    if (t==NULL) return 0;
    int a=Height(t->pleft);
    int b=Height(t->pright);
    if (a>b) return (a+1);
    return (b+1);
}
```

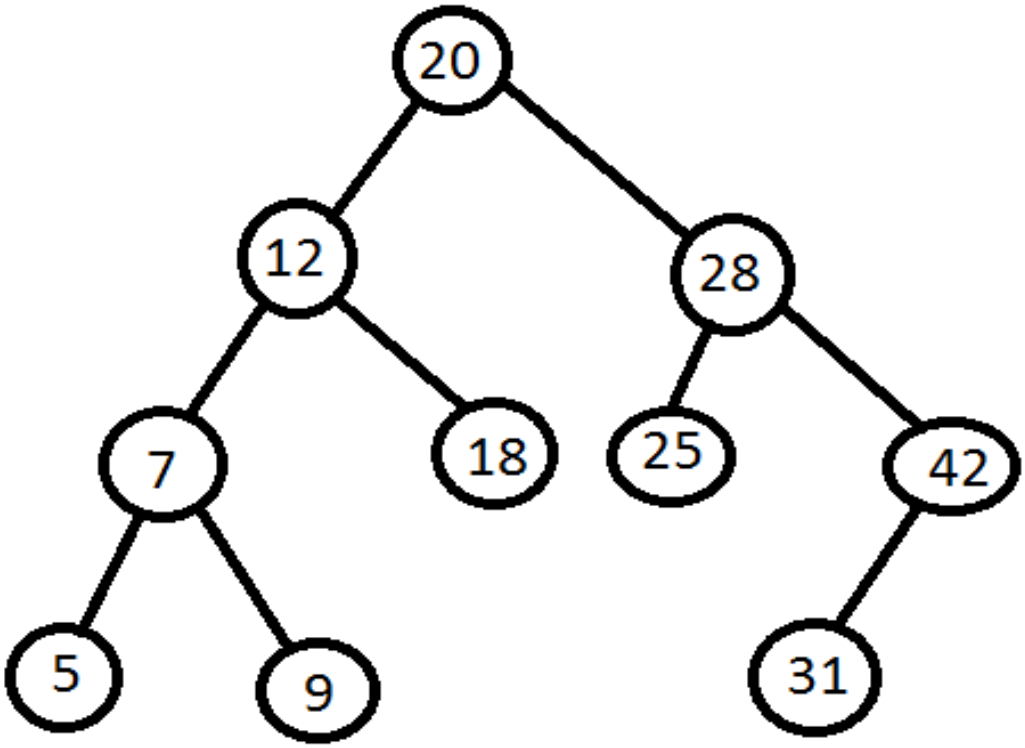
5. Ví dụ:

a) Vẽ cây nhị phân tìm kiếm nếu người ta lần lượt thêm vào các số dưới đây:

20 12 7 28 42 31 5 9 25 18

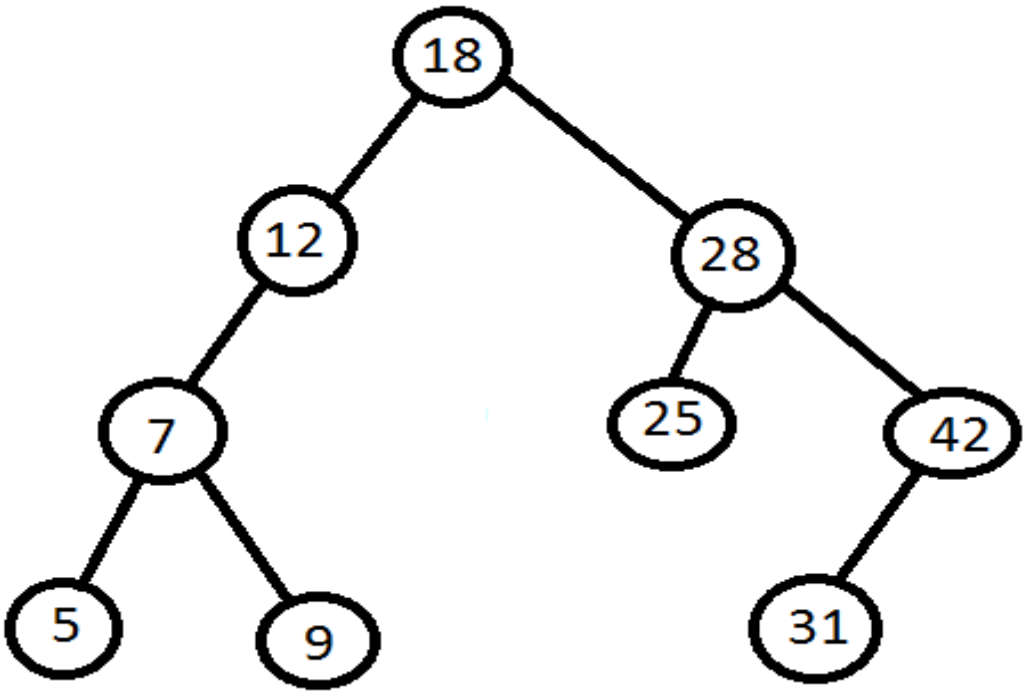
b) Vẽ lại cây nhị phân tìm kiếm nếu xoá lần lượt 20 và 28.

Câu a)

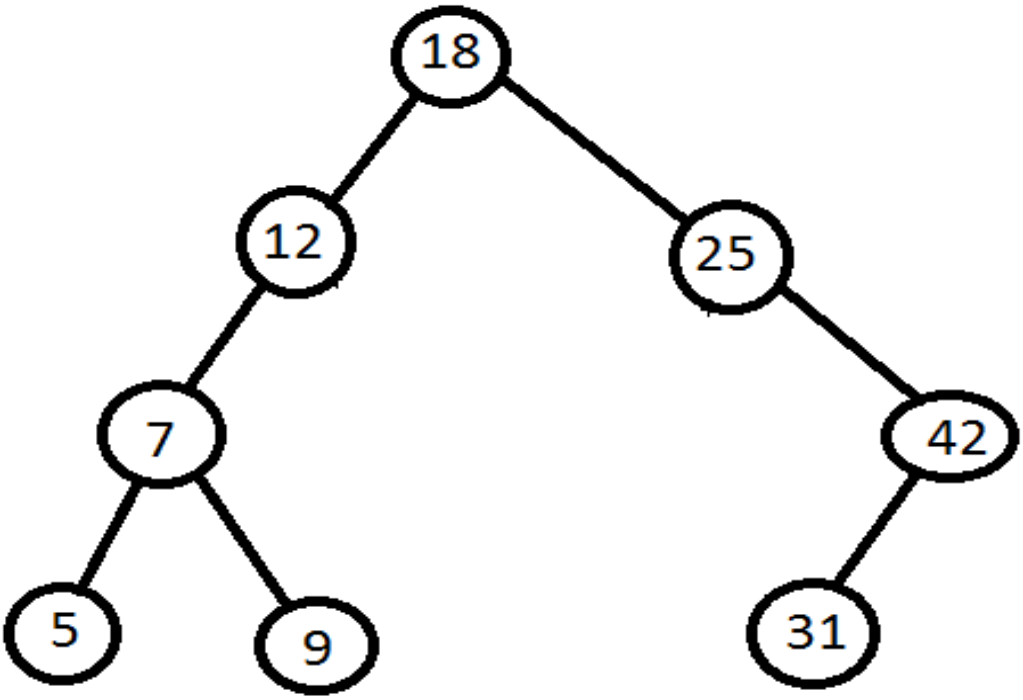


Câu b)

- Xoá 20: Chọn nút "thế mạng" là 18 (vị trí phải nhất hay lớn nhất bên nhánh trái):



- Xoá 28: Chọn nút "thế mạng" là 25 (vị trí trái nhất hay nhỏ nhất bên nhánh phải):



CHÚC CÁC BẠN THI TỐT ☺

NTMHP® 2016