SS1. Cho mảng số nguyên a[] có số phần tử n = 8 như sau, a[8] = {1,3,7,11,23,25,30,44}

a. Trình bày ý tưởng của giải thuật tìm kiếm nhị phân.

- Ý tưởng:

+ So sánh x với phần tử chính giữa mảng A.

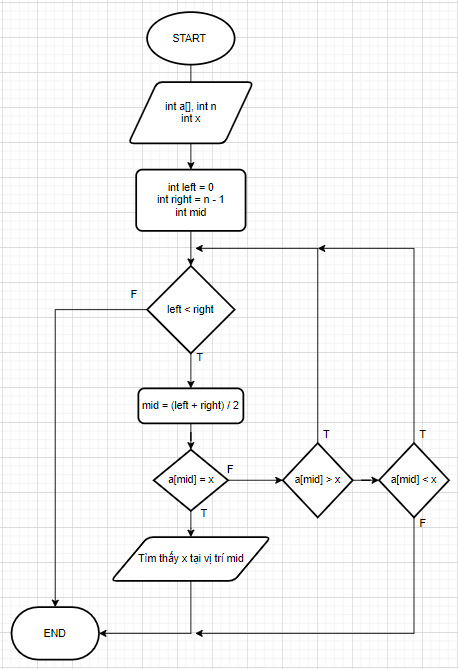
+ Nếu x là phần tử giữa thì dừng.

+ Nếu không: xác định xem ***x*** có thể thuộc nửa trái hay nửa phải của A.

+ Lặp lại 2 bước trên với nửa đã được xác định.

b. Vẽ lưu đồ giải thuật giải thuật tìm kiếm nhị phân.

- lưu đồ giải thuật giải thuật tìm kiếm nhị phân:



c. Viết hàm con tìm kiếm nhị phân cho mảng a[].

void TimKiemNhiPhan(int a[]) {

int x;

cout << "\nNhap phan tu can tim: ";

cin >> x;

int mid = 0, left = 0, right = 0;

left = 0;

right = 99;

bool check = false;

while(left <= right) {

mid = (left + right) / 2;

if(a[mid] == x) {

cout << "phan tu " << a[mid] << " nam o vi tri " << mid << "\n";

check = true;

break;

}

else if(a[mid] < x)

{

left = mid + 1;

}

else if(a[mid] > x)

{

right = mid - 1;

}

}

if(check == false)

{

cout << "Khong tim thay !";

}

}

2. Cho mảng số nguyên a[] có số phần tử n = 8 như sau, a[8] = {1,3,7,11,23,25,30,44}

a. Trình bày ý tưởng của giải thuật tìm kiếm tuần tự.

- Ý tưởng của giải thuật tìm kiếm tuần tự:

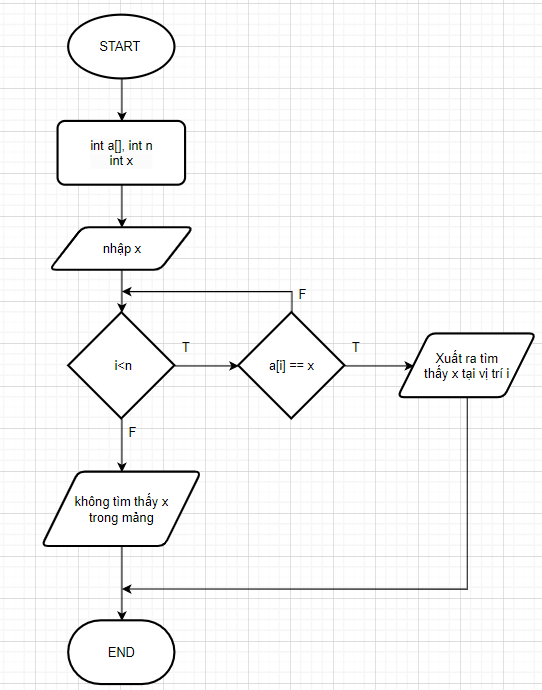
+ Dãy A, *n* phần tử

+ Giá trị *x* cần tìm

-> Duyệt từ đầu đến cuối mảng Nếu *x* xuất hiện trong A: trả về vị trí xuất hiện đầu tiên của *x*

b. Vẽ lưu đồ giải thuật giải thuật tìm kiếm tuần tự.

- lưu đồ giải thuật giải thuật tìm kiếm tuần tự:



c. Viết hàm con tìm kiếm nhị phân cho mảng a[].

void TimKiemNhiPhan(int a[]) {

int x;

cout << "\nNhap phan tu can tim: ";

cin >> x;

int mid = 0, left = 0, right = 0;

left = 0;

right = 99;

bool check = false;

while(left <= right) {

mid = (left + right) / 2;

if(a[mid] == x) {

cout << "phan tu " << a[mid] << " nam o vi tri " << mid << "\n";

check = true;

break;

}

else if(a[mid] < x)

{

left = mid + 1;

}

else if(a[mid] > x)

{

right = mid - 1;

}

}

if(check == false)

{

cout << "Khong tim thay !";

}

}

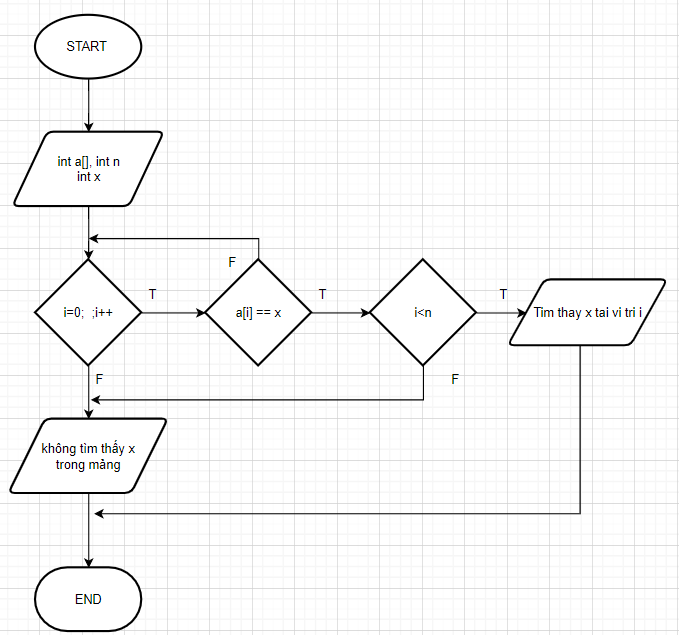
3. Cho mảng số nguyên a[] có số phần tử n = 8 như sau, a[8] = {1,3,7,11,23,25,30,44}

a. Trình bày ý tưởng của giải thuật tìm kiếm tuần tự có lính canh.

- Ý tưởng của thuật toán tìm kiếm tuần tự có lính canh: Trong thuật toán vét cạn, có 2 điều kiện được kiểm tra, có thể bỏ việc kiểm tra điều kiện cuối mảng bằng cách dùng “lính canh”.Lính canh là phần tử có giá trị bằng với phần tử cần tìm và đặt ở cuối mảng.

b. Vẽ lưu đồ giải thuật giải thuật tìm kiếm tuần tự có lính canh.

- Lưu đồ thuật toán của giải thuật tìm kiếm có lính canh:



c. Viết hàm con tìm kiếm nhị phân cho mảng a[].

void TimKiemNhiPhan(int a[]) {

int x;

cout << "\nNhap phan tu can tim: ";

cin >> x;

int mid = 0, left = 0, right = 0;

left = 0;

right = 99;

bool check = false;

while(left <= right) {

mid = (left + right) / 2;

if(a[mid] == x) {

cout << "phan tu " << a[mid] << " nam o vi tri " << mid << "\n";

check = true;

break;

}

else if(a[mid] < x)

{

left = mid + 1;

}

else if(a[mid] > x)

{

right = mid - 1;

}

}

if(check == false)

{

cout << "Khong tim thay !";

}

}

4. Cho mảng số nguyên a[] có số lượng phần tử n nhập từ bàn phím:

a. Viết hàm con nhập ngẫu nhiên mảng có n phần tử, gán giá trị ngẫu nhiên trong khoảng -99 đến 99.

void Random(int a[], int n) {

//Thuật toán xuất các giá trị random không trùng nhau

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

a[i] = rand() % 199 - 99;

for (short j = 0; j < i; j++) {

if (a[i] == a[j]) {

a[i] = rand() % 199 - 99;

j = -1;

}

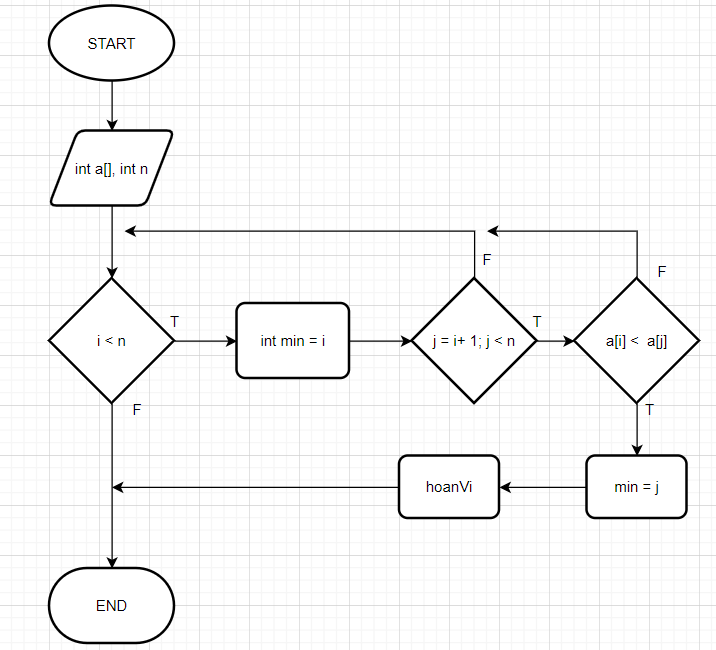
}

}

}

b. Trình bày ý tưởng, lưu đồ giải thuật và viết hàm sắp xếp mảng bằng giải thuật selection.

- Lưu đồ thuật toán của SelectionSort:



- Ý tưởng:

+ Chọn phần tử nhỏ nhất và đưa về vị trí đúng là đầu dãy hiện hành

+ Sau đó xem dãy hiện hành chỉ còn n – 1 phần tử

+ Lặp lại cho đến khi dãy hiện hành chỉ còn 1 phần tử  
- Hàm selection

void Selection\_soft(int a[], int n) {

for (short i = 0; i < n - 1; i++){

int min = i;

for (short j = i + 1; j < n ; j++){

if (a[min] > a[j])

{

min = j;

}

}

int tam = a[i];

a[i] = a[min];

a[min] = tam;

}

}

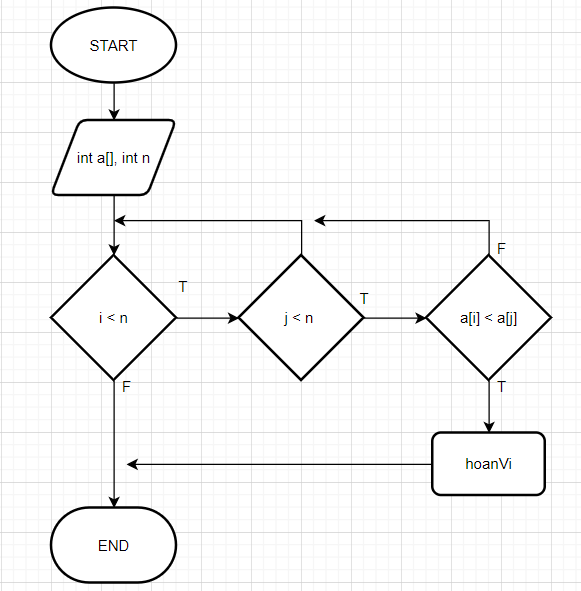
c. Trình bày ý tưởng, lưu đồ giải thuật và viết hàm sắp xếp mảng bằng giải thuật interchange.

- Ý tưởng:

+ Ý tưởng: bắt cặp tất cả các phần tử trong dãy cần sắp xếp và đổi chỗ hai phần tử trong cặp nếu chúng nghịch thế (không thỏa điều kiện thứ tự).

+ Xuất phát từ phần tử đầu tiên, tìm các cặp phần tử nghịch thế và đổi chỗ. Thực hiện cho đến phần tử cuối cùng, ta thu được một dãy đã sắp xếp.

- Lưu đồ giải thuật interchangeSort :



- Hàm interchange:

void InterChange\_sort(int a[], int n) {

for (short i = 0; i < n - 1; i++)

for (short j = i + 1; j < n; j++)

if (a[i] > a[j])

int tam = a[i];

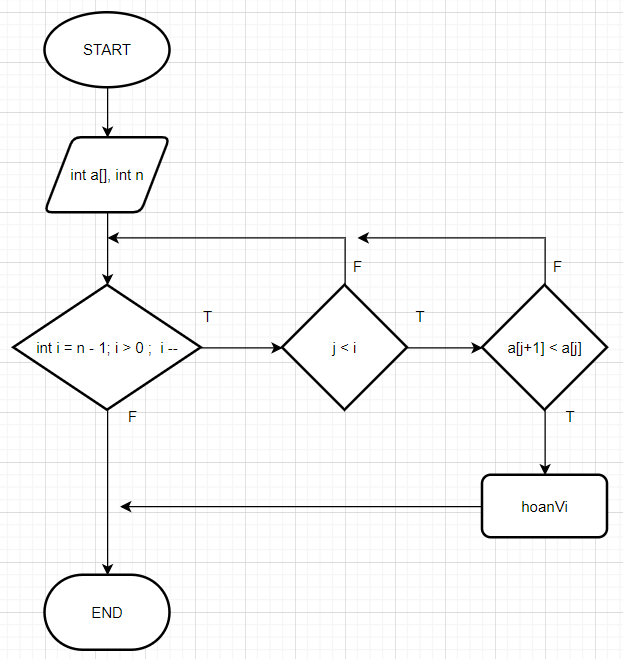
a[i] = a[j];

a[j] = tam;

}

d. Trình bày ý tưởng, lưu đồ giải thuật và viết hàm sắp xếp mảng bằng giải thuật bubble.

- Lưu đồ thuật toán của bubbleSort:



- Ý tưởng:

+ So sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (swap).

+ Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang).

+ Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp.

- Hàm bubbleSort:

void Bubble\_sort(int a[], int n) {

for (short i = n - 1; i > 0; i--)

for (short j = 0; j < n; j++)

if (a[j] > a[j + 1])

int tam = a[j];

a[j] = a[j + 1];

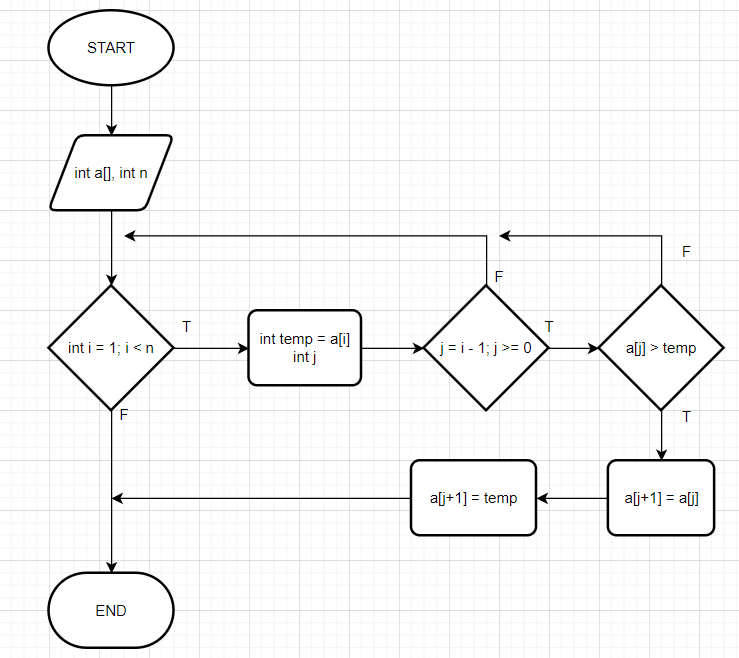
a[j + 1] = tam;

}

e. Trình bày ý tưởng, lưu đồ giải thuật và viết hàm sắp xếp mảng bằng giải thuật insertion.

- Ý tưởng: Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

- Lưu đồ thuật toán của insertsort:



- Hàm insertionSort:

void Insertion\_Sort(int a[], int n) {

for (short i = 1; i < n ; i++)

{

int tam = a[i];

short j;

for (j = i - 1; j >= 0; j--){

if (a[j] > tam)

a[j + 1] = a[j];

else

break;

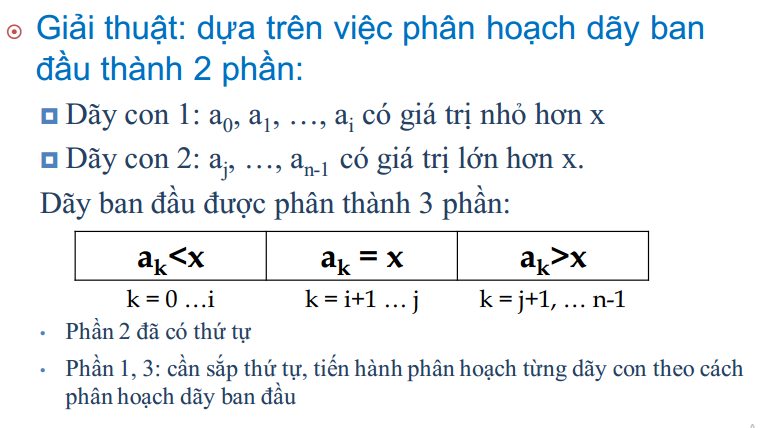
}

a[j + 1] = tam;

}

f. Trình bày ý tưởng và viết hàm sắp xếp mảng bằng giải thuật quick sort.

- ý tưởng:



- Hàm quick sort:

void QuickSort(int a[], int left, int right) {

int mid = (left + right) / 2;

int i = left;

int j = right;

while(i< j)

{

while (a[i] < a[mid])

i++;

while (a[j] > a[mid])

j--;

if (i<= j)

{

int tam = a[i];

a[i] = a[j];

a[j] = tam;

i++;

j--;

}

}

if (i < right)

{

QuickSort(a, i, right);

}

else if( j > left)

{

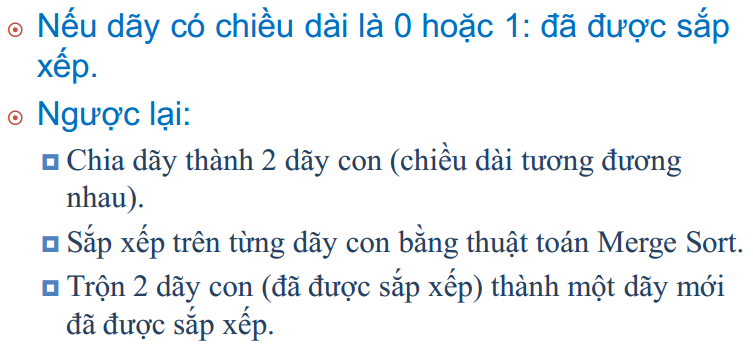
QuickSort(a, j, left);

}

}

g. Trình bày ý tưởng và viết hàm sắp xếp mảng bằng giải thuật merge sort.

- Ý tưởng:



-Hàm sắp xếp merge sort:

void MergeSort(int a[], int left, int right) {

if (left < right)

{

int mid = (left + right) / 2;

MergeSort(a, left, mid);

MergeSort(a, mid + 1, right);

Merge(a, left, mid, right);

}

}

#endif

void Merge(int a[], int left, int mid, int right) {

int n = right - left + 1;

int\* tam = new int[n];

int i = left;

int j = mid + 1;

int dem = 0;

while (dem <= n)

{

if (i > mid)

{

tam[dem] = a[j];

j++;

dem++;

}

else if (j > right)

{

tam[dem] = a[i];

i++;

dem++;

}

else

{

if (a[i] < a[j])

{

tam[dem] = a[i];

i++;

dem++;

}

if (a[j] < a[i])

{

tam[dem] = a[j];

j++;

dem++;

}

}

}//kết thúc trộn/chèn

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

a[left + i] = tam[i];

}

5. Cho mảng số nguyên a[] có số phần tử n = 6 như sau, a[6] = {3,1,5,2,9,7,}. Cho biết từng kết quả của mảng sau n lần chạy khi sắp xếp mảng bằng giải thuật selection.

Kết quả Lần chạy 1:

{1, 3, 5, 2, 9, 7}

Kết quả Lần chạy 2:

{1, 2, 5, 3, 9, 7}

Kết quả Lần chạy 3:

{1, 2, 3, 5, 9, 7}

Kết quả Lần chạy 4:

{1, 2, 3, 5, 9, 7}

Kết quả Lần chạy 5:

{1, 2, 3, 5, 7, 9}

* Sau khi sắp xếp, mảng trở thành {1, 2, 3, 5, 7, 9}

6. Cho mảng số nguyên a[] có số phần tử n = 6 như sau, a[6] = {3,1,5,2,9,7}. Cho biết từng kết quả của mảng sau n lần chạy khi sắp xếp mảng bằng giải thuật interchange.

Kết quả chạy lần 1:

{1, 3, 5, 2, 9, 7}

Kết quả chạy lần 2:

{1, 2, 3, 5, 9, 7}

Kết quả chạy lần 3:

{1, 2, 3, 5, 9, 7}

Kết quả chạy lần 4:

{1, 2, 3, 5, 9, 7}

Kết quả chạy lần 5:

{1, 2, 3, 5, 7, 9}

* Sau khi sắp xếp, mảng trở thành {1, 2, 3, 5, 7, 9}.

7. Cho mảng số nguyên a[] có số phần tử n = 6 như sau, a[6] = {3,1,5,2,9,7}. Cho biết từng kết quả của mảng sau n lần chạy khi sắp xếp mảng bằng giải thuật bubble.

Kết quả chạy lần 1:

{1,3, 2, 5, 7, 9}

Kết quả chạy lần 2:

{1,2, 3, 5, 7, 9}

Kết quả chạy lần 3:

{1,2, 3, 5, 7, 9}

Kết quả chạy lần 4:

{1,2, 3, 5, 7, 9}

Kết quả chạy lần 5:

{1,2, 3, 5, 7, 9}

8. Cho mảng số nguyên a[] có số phần tử n = 6 như sau, a[6] = {3,1,5,2,9,7}. Cho biết từng kết quả của mảng sau n lần chạy khi sắp xếp mảng bằng giải thuật insertion.(Mõi lần duyệt xem có thằng nào trước nó mà lớn hơn nó không)

Kết quả chạy lần 1: { **1, 3**, 5, 2, 9, 7}

Kết quả chạy lần 2: { **1, 3, 5**, 2, 9, 7}

Kết quả chạy lần 3: { **1, 2, 3, 5**, 9, 7}

Kết quả chạy lần 4: { **1, 2, 3, 5, 9**, 7}

Kết quả chạy lần 5: { **1, 2, 3, 5, 7, 9**}

* Sau khi sắp xếp, mảng trở thành {1, 2, 3, 5, 7, 9}.

9. Cho một Node trong cây nhị phân tìm kiếm có cấu trúc như sau:

struct Node{

int key;

Node\* left;

Node\* right;

};

a. Trình bày đặc điểm của cây nhị phân tìm kiếm

* Đặc điểm:
  + Có thứ tự
  + Không có phần tử trùng
  + Dễ dàng tạo dữ liệu sắp xếp, và tìm kiếm

b. Viết hàm con tạo một node mới của cây nhị phân tìm kiếm

Node \*newNode(int data)

{

Node \*temp = new Node;

temp->key = data;

temp->left = NULL;

temp->right = NULL;

return temp;

}

c. Viết hàm con thêm một node vào cây nhị phân tìm kiếm

void addNode(Node \*&root,int data)

{

if(root==NULL)

root = newNode(data);

else

{

if(root->key > data)

addNode(root->left,data);

if(root->key < data)

addNode(root->right,data);

}

}

d. Viết hàm duyệt cây bằng phương pháp duyệt trước

void preOrder(Node \*root)

{

if(root!=NULL)

{

cout<<" "<<root->key;

preOrder(root->left);

preOrder(root->right);

}

}

e. Viết hàm duyệt cây bằng phương pháp duyệt sau

void postOrder(Node \*root)

{

if(root!=NULL)

{

postOrder(root->left);

postOrder(root->right);

cout<<" "<<root->key;

}

}

f. Viết hàm duyệt cây bằng phương pháp duyệt giữa

void inOrder(Node \*root)

{

if(root!=NULL)

{

inOrder(root->left);

cout<<" "<<root->key;

inOrder(root->right);

}

}

g. Viết hàm xóa 1 node trong cây nhị phân tìm kiếm

Node\* findMin(Node\* root) {

while (root->left != nullptr) {

root = root->left;

}

return root;

}

Node\* deleteNode(Node\* root, int data) {

if (root == nullptr) return root;

if (data < root->key) {

root->left = deleteNode(root->left, data);

} else if (data > root->key) {

root->right = deleteNode(root->right, data);

} else {

if (root->left == nullptr) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == nullptr) {

Node\* temp = root->left;

delete root;

return temp;

} Node\* temp = findMin(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

h. Viết hàm tìm kiếm 1 node trong cây nhị phân tìm kiếm  
Node\* timKiem(Node\* t, int x) {

if (t == nullptr)

return nullptr;

else

{

if (x < t->key)

timKiem(t->pLeft, x);

else if (x > t->data)

timKiem(t->pRight, x);

else

{

return t;

}

}

}

k. Viết hàm đếm số node trong cây nhị phân tìm kiếm

int countNodes(Node\* root) {

if (root == nullptr)

return 0;

return 1 + countNodes(root->left) + countNodes(root->right);

}

m. Viết hàm quay trái/phải cây nhị phân tìm kiếm(Nhớ chọn 1 trong 2 để học th không cần học hết cả hai)

Node\* rotateRight(Node\* root) {

if (root == nullptr || root->left == nullptr)

return root;

Node\* newRoot = root->left;

root->left = newRoot->right;

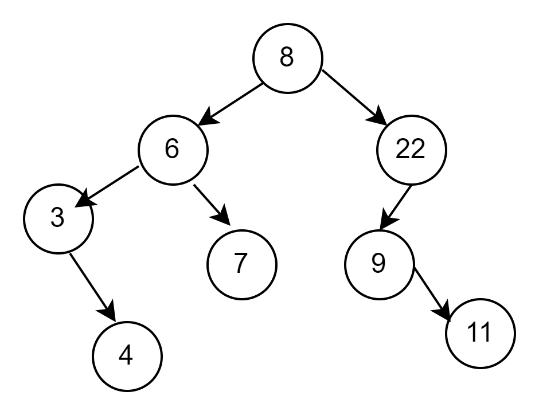
newRoot->right = root;

return newRoot;

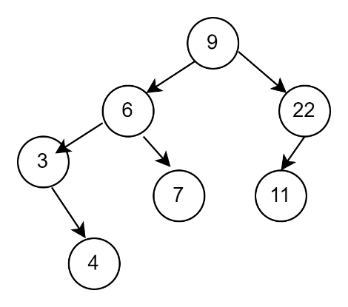
}

10. Tạo và lần lượt thêm vào cây nhị phân tìm kiếm các giá trị như sau: 8, 6, 3, 7, 4, 22, 9, 11

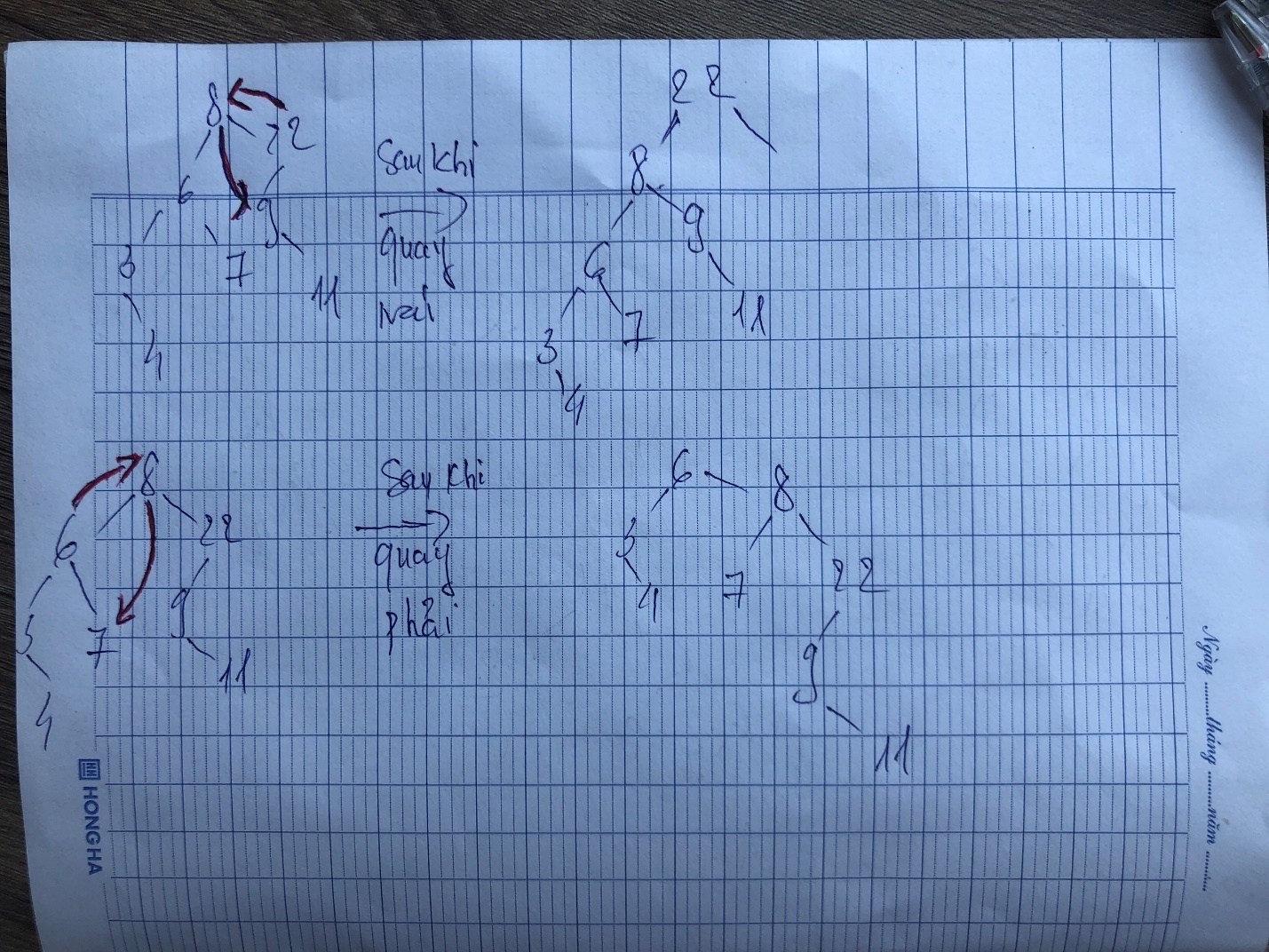
a. Vẽ cây nhị phân tìm kiếm(Quy tắc thêm vào cây là nhỏ qua trái lớn qua phải b nhớ nha tức có dãy như vầy 8, 6, 3, 7, 4, 22, 9, 11 thì phần tử đầu tiên đc thêm vào là phần tử gốc là số 8 sau đó thêm lần lượt b áp dụng quy tắc nhỏ hơn phần tử gốc thì b cho qua bên trái phần tử gốc còn lớn hơn phần tử gốc b cho qua phải phân tử gốc )



b. Vẽ cây nhị phân tìm kiếm sau khi xóa phần tử gốc(Xóa node gốc node thế node gốc vừa bị xóa là node trái nhất của cây con phải nhớ nha b ơiiii)



c. Vẽ cây nhị phân tìm kiếm sau khi thực hiện quay trái/phải cây



11. Cho cây nhị phân sau

A diagram of numbers and circles

Description automatically generated

a. Trình bày kết quả duyệt cây bằng phương pháp duyệt trước(NLR tức xuất node trước sau đó tới left cuối cùng là right nha b ơiiiii)

40, 5, 35, 15, 13, 16, 45, 56, 48, 47, 49

b. Trình bày kết quả duyệt cây bằng phương pháp duyệt giữa(LNR tức xuất left trước sau đó tới node cuối cùng là right nha b ơiiii)

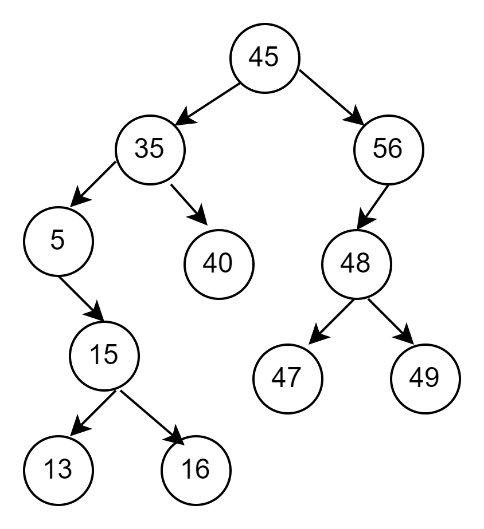
5, 13, 15, 16, 35, 40, 45, 47, 48, 49, 56

c. Trình bày kết quả duyệt cây bằng phương pháp duyệt sau(LRN tức xuất left trước sau đó tới right cuối cùng là node nha b ơiiiii)

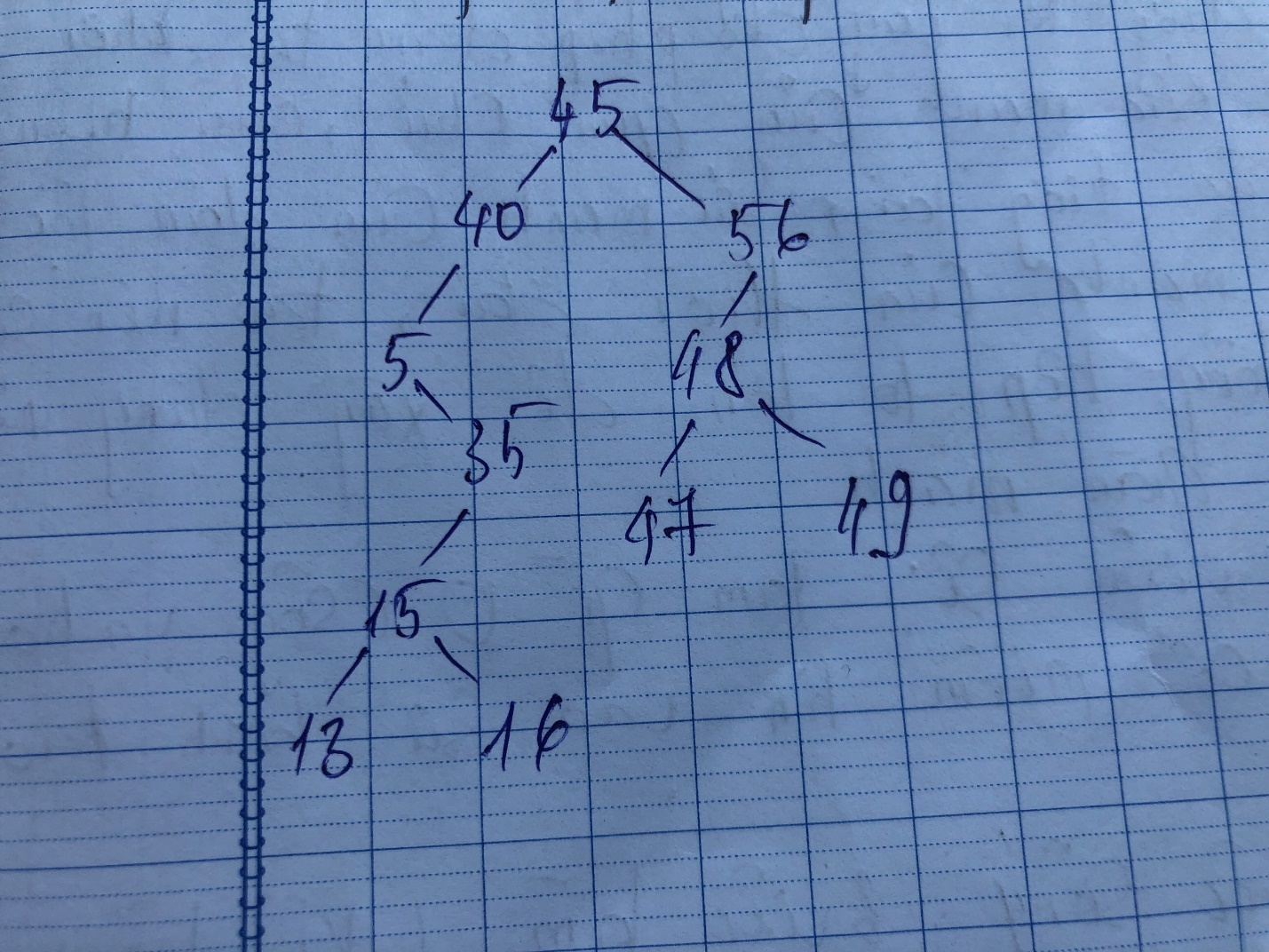
13, 16, 15, 35, 5, 47, 49, 48, 56, 45, 40

d. Trình bày kết quả cân bằng cây ( Cây cân bằng AVL là cây nhị phân tìm kiếm mà tại mỗi đỉnh của cây, độ cao của cây con trái và cây con phải không chênh lệch quá 1. Đây là kn cây cân bằng á b ơii)

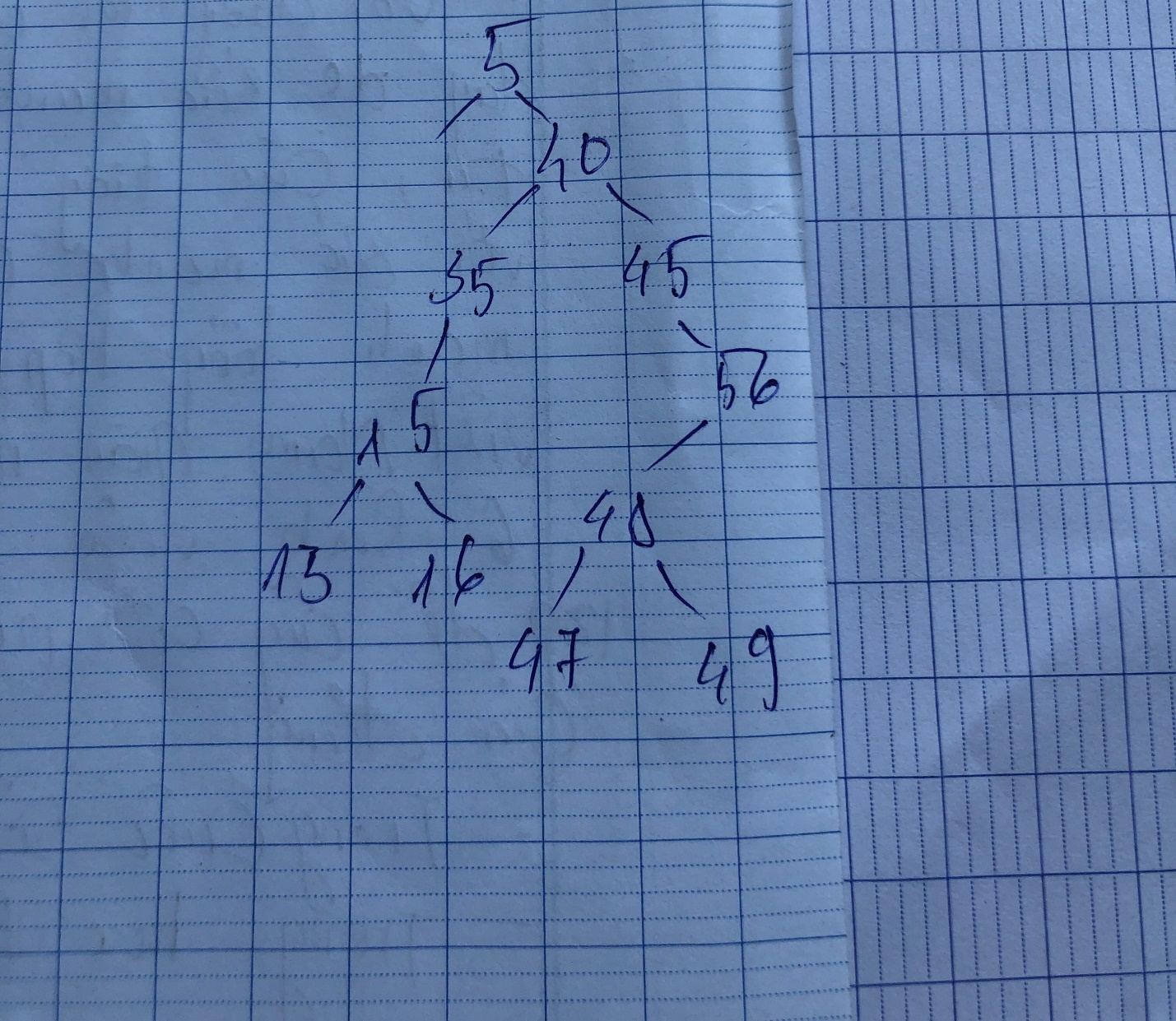
- Cân bằng cây bằng phương pháp quay trái cây:



e. Trình bày kết quả quay trái cây



f. Trình bày kết quả quay phải cây



12. Cho ngăn xếp được xây dựng bằng mảng một chiều, có n phần tử.

a. Trình bày nguyên lý hoạt động của ngăn xếp

Ngăn xếp hoạt động theo nguyên lí Last In Firt Out – vào sau ra trước tức là lấy thì lấy phần tử ở cuối danh sách và thêm thì cũng thêm vào cuối danh sách.

b. Viết hàm push thêm 1 phần tử vào ngăn xếp

void push(int stack[], int x) {

if (vitridinh > maxStack - 1)

cout << "\nStack day !";

else {

vitridinh++;

stack[vitridinh] = x;

}

}

c. Viết hàm pop lấy 1 phần tử khỏi ngăn xếp

void pop(int stack[]) {

if (vitridinh < 0)

cout << "\nStack rong !";

else

{

cout << char(stack[vitridinh]);

vitridinh--;

}

}

13. Cho hàng đợi được xây dựng bằng mảng một chiều, có n phần tử.

a. Trình bày nguyên lý hoạt động của hàng đợi

Hàng đợi hoạt động theo nguyên lí Firt In Firt Out – vào trước ra trước tức lấy là lấy phần tử ở đầu danh sách và thêm thì thêm phần tử vào cuối danh sách.

b. Viết hàm enqueue thêm 1 phần tử vào hang đợi

void enqueque(int queque[], int x)

{

if (sophantutrongQue >= MaxQueue)

cout << "\nhang doi da day";

else

{

if (vitricuoi == vitridau && vitricuoi == 0 && sophantutrongQue == 0) {

queque[vitridau] = x;

sophantutrongQue++;

}

else

{

vitricuoi++;

if (vitricuoi > MaxQueue – 1)

vitricuoi = 0;

queque[vitricuoi] = x;

sophantutrongQue++;

}

}

}

c. Viết hàm dequeue lấy 1 phần tử khỏi hang đợi

void dequeque(int queque[])

{

if (sophantutrongQue == 0)

cout << "\nHang doi da het !";

else

{

if (vitridau == vitricuoi && sophantutrongQue == 1){

cout << "\n" << (char)queque[vitridau];

vitridau = 0;

vitricuoi = 0;

sophantutrongQue = 0;

}

else

{

cout << "\n" << (char)queque[vitridau];

vitridau++;

if (vitridau > MaxQueue - 1)

vitridau = 0;

sophantutrongQue--;

}

}

}

14. Danh sách liên kết

Cho một Node trong danh sách liên kết đơn có cấu trúc như sau:

struct Node{

int key;

Node\* next;

};

a. Trình bày đặc điểm của cấu trúc danh sách liên kết, so sánh với cấu trúc mảng

- Đặc Điểm:

* Một dãy tuần tự các phần tử (node)
* Giữa hai phần tử có liên kết với nhau.
* Các phần tử không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
* Có thể mở rộng tuỳ ý (chỉ giới hạn bởi dung lượng bộ nhớ)
* Thao tác Chèn/Xóa không cần phải dịch chuyển phần tử
* Có thể truy xuất đến các phần tử khác thông qua các liên kết

- So sánh:



b. Viết hàm tạo 1 node mới trong danh sách liên kết đơn

Node \*newNode(int data){

Node \*p = new Node;

p->Key = data;

p->Next = NULL;

return p;

}

c. Viết hàm thêm 1 node mới vào (đầu/cuối) trong danh sách liên kết đơn

- Hàm thêm vào đầu danh sách liên kết đơn

void AddHead(Node\* &pHead, int data){

Node \*temp = newNode(data);

if(pHead == NULL)

pHead = temp;

else{

temp->Next = pHead;

pHead = temp;

}

}

- Hàm thêm vào cuối danh sách liên kết

void AddLast(Node \*pHead, int data)

{

if(pHead==NULL)

AddHead(pHead,data);

else

{

Node \*temp = newNode(data);

while(pHead->Next!=NULL)//di chuyển đến node cuối

pHead = pHead->Next;

pHead->Next = temp;

}

}

d. Viết hàm xóa 1 node (đầu/cuối) trong danh sách liên kết đơn

- Hàm xóa đầu trong danh sách liên kết đơn:

void RemoveHead(Node \*&pHead){

if(pHead==NULL)

cout<<"\ndanh sach rong";

else

{

Node \*temp = pHead;//lưu lại địa chỉ pHead ban đầu

pHead = pHead->Next;//pHead là node kế tiếp

delete(temp);//xóa vùng nhớ node pHead ban đầu

}

}

- Hàm xóa cuối trong danh sách liên kết đơn:

void RemoveLast(Node \*&pHead)

{

if(pHead==NULL)

cout<<"\ndanh sach rong";

else if(pHead->Next == NULL)//trường hợp danh sach chỉ còn 1 node

{

RemoveHead(pHead);

cout<<"\nda xoa het danh sach";

}

else

{

Node \*pHeadTemp = pHead;//vị trí pHead ban đầu

while(pHead->Next->Next != NULL)//xác định xem node kế tiếp là cuối

pHead = pHead->Next;//ko phải node cuối thì đi đến node kế

Node \*temp = pHead->Next;//lưu lại địa chỉ node cuối

pHead->Next = NULL;//xóa node cuối

delete(temp);//giải phóng vùng nhớ node cuối

pHead = pHeadTemp;//trả pHead về vị trí đầu

}

}

e. Viết hàm tìm kiếm 1 node trong danh sách liên kết đơn

- Hàm xóa 1 node trong danh sách liên kết đơn

bool FindX(Node \*pHead, int x)

{

while(pHead!=NULL)//khi chưa đến node cuối thì duyệt tiếp

if(pHead->Key==x)//nếu tìm thấy node x thì dừng lại

break;

else

pHead = pHead->Next;//ko tìm thấy thì đi đến node kế

if(pHead==NULL)

return false;//ko tìm thấy

else

return true;//có node x

}

f. Viết hàm xóa 1 node có giá trị = x trong danh sách liên kết đơn

- hàm xóa 1 node có giá trị = x trong danh sách liên kết đơn

bool FindX(Node \*pHead, int x)

{

while(pHead!=NULL)//khi chưa đến node cuối thì duyệt tiếp

if(pHead->Key==x)//nếu tìm thấy node x thì dừng lại

break;

else

pHead = pHead->Next;//ko tìm thấy thì đi đến node kế

if(pHead==NULL)

return false;//ko tìm thấy

else

return true;//có node x

}

void RemoveX(Node \*pHead, int x)

{

if(FindX(pHead,x)==true)

{

while(pHead->Next->Key!=x)//di chuyển đến node trước x

pHead = pHead->Next;

Node \*temp = pHead->Next;//lưu lại địa chỉ node x

pHead->Next = pHead->Next->Next;//trỏ node hiện tại đến node sau x

delete(temp);

}

else

cout<<"\nko co node x";

}

g. Viết hàm thêm 1 node sau node có giá trị = x trong danh sách liên kết đơn

- hàm thêm 1 node sau node có giá trị = x trong danh sách liên kết đơn

void AddAfterX(Node \*pHead, int x)

{

if(FindX(pHead,x)==true)//kiểm tra có x ko

{

while(pHead->Key != x)//di chuyển đến x

pHead = pHead->Next;//đi đến node kế tiếp

//sau khi đã đến node x

int data;

cout<<"\nnhap gia tri can them = ";

cin>>data;

Node \*temp = newNode(data);//tạo node tạm

temp->Next = pHead->Next;//tạm trỏ đến node sau x

pHead->Next = temp;//nối node x đến tạm mới tạo

}

}