BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI THÀNH PHỐ HỒ CHÍ MINH

----------o0o----------

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

(Lưu hành nội bộ)

Ths. Bùi Văn Thượng

Chương 1. TỔNG QUAN

1.1 Mô hình hóa bài toán thực tế

Mô hình hóa bài toán tức là nhận biết và giải quyết các bài toán thực tế theo hướng tin học hóa. Thông thường, khi khởi đầu, hầu hết các bài toán là không đơn giản, không rõ ràng. Để giảm bớt sự phức tạp của bài toán thực tế, ta phải hình thức hóa nó, nghĩa là phát biểu lại bài toán thực tế thành một bài toán hình thức (hay còn gọi là mô hình toán). Có thể có rất nhiều bài toán thực tế có cùng một mô hình toán.

Để giải một bài toán trong thực tế bằng máy tính ta phải bắt đầu từ việc xác định bài toán:

- *Phải làm gì ?*
- *Phải làm như thế nào?*

Ví dụ 1: Tô màu bản đồ thế giới.

Ta cần phải tô màu cho các nước trên bản đồ thế giới. Trong đó mỗi nước đều được tô một màu và hai nước láng giềng (cùng biên giới) thì phải được tô bằng hai màu khác nhau. Hãy tìm một phương án tô màu sao cho số màu sử dụng là ít nhất.

Ta có thể xem mỗi nước trên bản đồ thế giới là một đỉnh của đồ thị, hai nước láng giềng của nhau thì hai đỉnh ứng với nó được nối với nhau bằng một cạnh. Bài toán lúc này trở thành bài toán tô màu cho đồ thị như sau: Mỗi đỉnh đều phải được tô màu, hai đỉnh có cạnh nối thì phải tô bằng hai màu khác nhau và ta cần tìm một phương án tô màu sao cho số màu được sử dụng là ít nhất.

1.2 Cấu trúc dữ liệu

1.2.1 Tổng quan

Kiểu dữ liệu là một tập hợp các giá trị và một tập hợp các phép toán trên các giá trị đó.

Ví dụ 2:

- *Kiểu Boolean là một tập hợp có 2 giá trị TRUE, FALSE và các phép toán trên nó như OR, AND, NOT ….*
- *Kiểu Integer là tập hợp các số nguyên có giá trị từ -32768 đến 32767 cùng các phép toán cộng, trừ, nhân, chia, Div, Mod…*

Kiểu dữ liệu có hai loại là kiểu dữ liệu sơ cấp và **kiểu dữ liệu có cấu trúc** còn gọi là **cấu trúc dữ liệu**.

- *Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị dữ liệu của nó là đơn nhất. Ví dụ: kiểu Boolean, Integer, character, float….*
- *Kiểu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu là kiểu dữ liệu mà giá trị dữ liệu của nó là sự kết hợp của các giá trị khác. Ví dụ: Kiểu mảng, kiểu cấu trúc, danh sách liên kết….*

1.2.2 Tiêu chuẩn lựa chọn cấu trúc dữ liệu

Cấu trúc dữ liệu đóng vai trò quan trọng trong việc kết hợp và đưa ra cách giải quyết cho bài toán. Nó hỗ trợ cho các thuật toán thao tác trên đối tượng được hiệu quả hơn.

Việc lựa chọn cấu trúc dữ liệu cho bài toán có thể dựa vào các tiêu chuẩn sau:

- *Phải biểu diễn được đầy đủ thông tin nhập và xuất của bài toán.*
- *Phải phù hợp với các thao tác của thuật toán mà ta lựa chọn.*
- *Phù hợp với điều kiện cho phép của ngôn ngữ lập trình đang sử dụng.*
- *Tiết kiệm tài nguyên hệ thống.*

1.2.3 Các kiểu cấu trúc dữ liệu

Cấu trúc dữ liệu có thể chia thành các dạng phổ biến sau:

- ***Cấu trúc dữ liệu tuyến tính**: Là cấu trúc dữ liệu trong đó các phần tử được liên kết tuần tự, nối tiếp với nhau (ví dụ: Mảng, danh sách liên kết,…).*
- ***Cấu trúc dữ liệu dạng cây**: Mỗi phần tử có thể liên kết với nhiều phần tử khác theo từng mức.*
- ***Cấu trúc dữ liệu bảng băm**: Là một cấu trúc dữ liệu tương tự như mảng nhưng kèm theo một hàm băm để ánh xạ nhiều giá trị vào cùng một phần tử trong mảng.*
- ***Cấu trúc dữ liệu dạng đồ thị**: Là dạng cấu trúc dữ liệu bao gồm một tập các đối tượng được gọi là các đỉnh (hoặc nút) nối với nhau bởi các cạnh (hoặc cung).*

1.3 Giải thuật

1.3.1 Khái niệm

Giải thuật hay thuật toán là một chuỗi hữu hạn các thao tác để giải một bài toán nào đó.

Các tính chất quan trọng của giải thuật là:

- ***Hữu hạn**: Giải thuật phải luôn luôn kết thúc sau một số hữu hạn bước.*
- ***Xác định**: Mỗi bước của giải thuật phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.*
- ***Đúng**: Giải thuật phải đảm bảo tính đúng và chính xác;*
- ***Hiệu quả** : Các thao tác trong giải thuật phải được thực hiện trong một lượng thời gian hữu hạn.*
- *….*

Ngoài ra một giải thuật còn phải có đầu vào (input) và đầu ra (output).

1.3.2 Biểu diễn giải thuật

Có nhiều cách để biểu diễn thuật toán. Dưới đây là một số cách thường được sử dụng:

- ***Sử dụng ngôn ngữ tự nhiên**: Liệt kê tuần tự các bước để giải quyết bài toán. Cách này đơn giản và không cần kiến thức về biểu diễn thuật toán, tuy nhiên nó thường dài dòng và đôi khi khó hiểu.*
- ***Sử dụng lưu đồ** (sơ đồ khối): Sử dụng các hình khối khác nhau để biểu diễn thuật toán. Cách này trực quan và dễ hiểu , tuy nhiên hơi cồng kềnh.*
- ***Sử dụng mã giả**: Sử dụng ngôn ngữ tựa ngôn ngữ lập trình để biểu diễn thuật toán. Cách làm này đỡ cồng kềnh hơn sơ đồ khối tuy nhiên không trực quan.*
- ***Sử dụng ngôn ngữ lập trình**: Sử dụng các ngôn ngữ máy tính để biểu diễn (Pascal, C,…). Cách này đòi hỏi phải có kiến thức và kỹ năng về ngôn ngữ lập trình được sử dụng.*

1.3.3 Đánh giá độ phức tạp của thuật toán

Thời gian chạy một chương trình phụ thuộc vào các yếu tố sau:

- *Khối lượng của dữ liệu đầu vào.*
- *Chất lượng của mã máy được tạo ra bởi trình dịch.*
- *Tốc độ thực thi lệnh của máy.*
- *Độ phức tạp về thời gian của thuật toán*

Một thuật toán được gọi là hiệu quả nếu chi phí cần sử dụng tài nguyên của máy là thấp .

Để mô tả việc đánh giá độ phức tạp của thuật toán người ta sử dụng một hàm f(N), trong đó N là khối lượng dữ liệu cần được xử lý.

Có hai phương pháp để đánh giá độ phức tạp của thuật toán gồm:

- **Phương pháp thực nghiệm:**
  - *Cách làm: Cài thuật toán rồi chọn các bộ dữ liệu thử nghiệm, sau đó thống kê các thông số nhận được khi chạy các bộ dữ liệu đó để đánh giá.*
  - *Ưu điểm: Dễ thực hiện.*
  - *Nhược điểm:*
    - *Chịu sự hạn chế của ngôn ngữ lập trình.*
    - *Ảnh hưởng bởi trình độ của người lập trình.*
    - *Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: Khó khăn và tốn nhiều chi phí.*
    - *Phụ thuộc vào phần cứng.*
- **Phương pháp xấp xỉ toán học:**
  - *Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().*
  - *Ưu điểm: Ít phụ thuộc môi trường cũng như phần cứng hơn.*
  - *Nhược điểm: Phức tạp.*
  - *Các trường hợp độ phức tạp quan tâm:*
    - *Trường hợp tốt nhất (phân tích chính xác)*
    - *Trường hợp xấu nhất (phân tích chính xác)*
    - *Trường hợp trung bình (mang tích dự đoán)*
  - *Phân lớp độ phức tạp của giải thuật: Trong bảng dưới đây, độ phức tạp của thuật toán là tăng dần:*

Hằng số	*O(c)*
logN	*O(logN)*
N	*O(N)*
NlogN	*O(NlogN)*
N2	*O(N2)*
N3	*O(N3)*
2N	*O(2N)*
N!	*O(N!)*

1.4 Chương trình

Theo **Niklaus Wirth**:

Cấu trúc dữ liệu + Thuật toán = Chương trình

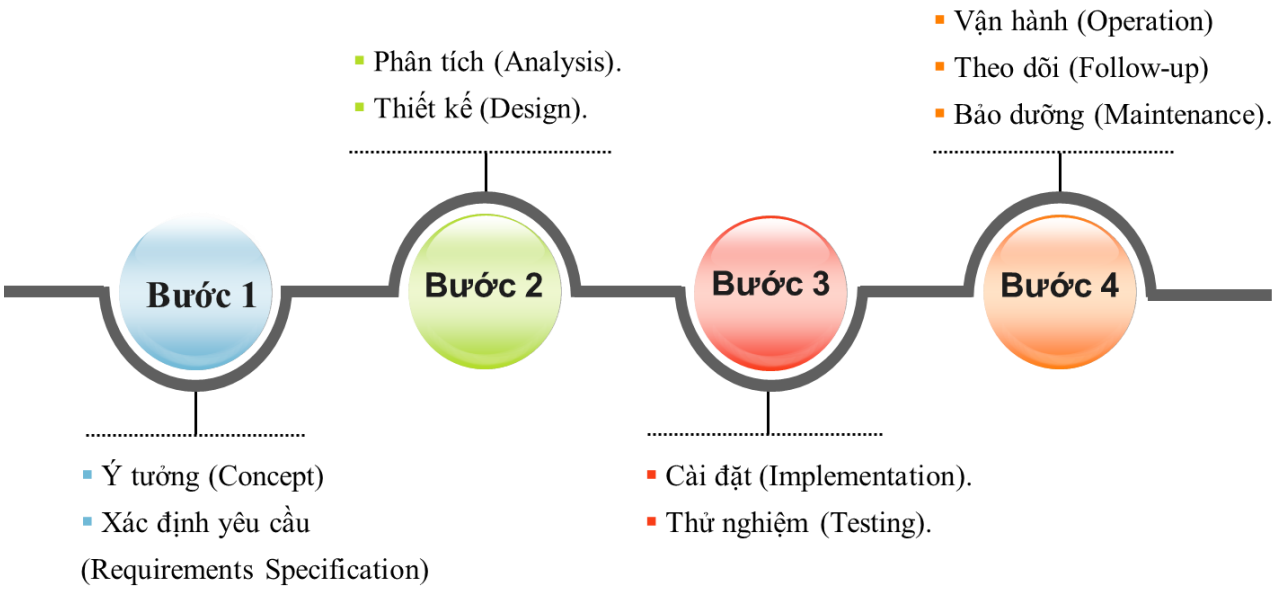
1.4.1 Tiểu chuẩn của một chương trình

Một chương trình máy tính cần đảm bảo các tiêu chuẩn sau:

- ***Tính tin cậy**: Chương trình phải chạy đúng như dự định, mô tả chính xác một giải thuật đúng.*
- ***Tính uyển chuyển**: Chương trình dễ sửa đổi, giảm bớt công sức của lập trình viên khi phát triển chương trình, đáp ứng các quy trình làm phần mềm.*
- ***Tính trong sáng**: Chương trình viết ra phải dễ đọc, dễ hiểu. Tính trong sáng phụ thuộc rất nhiều vào công cụ lập trình và phong cách lập trình.*
- ***Tính hữu hiệu**: Chương trình phải đáp ứng được yêu cầu là chạy nhanh và ít tốn tài nguyên bộ nhớ, điều này phụ thuộc vào chất lượng của giải thuật cũng như tiểu xảo của lập trình viên.*

Từ những phân tích trên ta thấy rằng việc tạo ra một chương trình đòi hỏi rất nhiều công đoạn và tiêu tốn rất nhiều công sức. Vì vậy đừng bao giờ viết chương trình mà chưa suy xét kỹ về giải thuật và những dữ liệu cần thao tác.

1.4.2 Quy trình làm phần mềm



Hình 1. Quy trình làm phần mềm.

Xác định yêu cầu bài toán:

Dữ liệu vào -> Xử lý -> Dữ liệu ra

Việc xác định bài toán tức là ta phải xác định xem cần giải quyết những vấn đề gì? Với giả thiết nào đã cho và lời giải cần phải đạt được những yêu cầu gì.

Phân tích, thiết kế: Lựa chọn cấu trúc dữ liệu và thiết kế giải thuật phù hợp cho bài toán.

Cài đặt: Sau khi đã có thuật toán ta cần lập trình để thể hiện thuật toán đó. Muốn lập trình đạt kết quả cao, cần phải có kỹ thuật lập trình tốt. Kỹ thuật lập trình tốt thể hiện ở kỹ năng viết chương trình, khả năng gỡ rối và thao tác nhanh.

Thử nghiệm: Chạy thử, tìm và sửa lỗi, xây dựng các bộ test, …

BÀI TẬP CHƯƠNG 1

Trình bày thuật toán để giải quyết các bài toán sau (*dùng ngôn ngữ tự nhiên và sơ đồ khối*):

1. Tìm bội chung nhỏ nhất của 2 số nguyên dương
2. Giải phương trình bậc hai: $ax^2+bx+c=0$
3. Tìm phần tử thứ n của dãy Fibonacci

Chương 2. CÁC GIẢI THUẬT TÌM KIẾM

2.1 Bài toán tìm kiếm

Tìm kiếm là một đòi hỏi rất thường xuyên trong các ứng dụng tin học. Tìm kiếm có thể định nghĩa là việc thu thập một số thông tin nào đó từ một khối thông tin lớn đã được lưu trữ trước đó. Bài toán tìm kiếm có thể được phát biểu như sau:

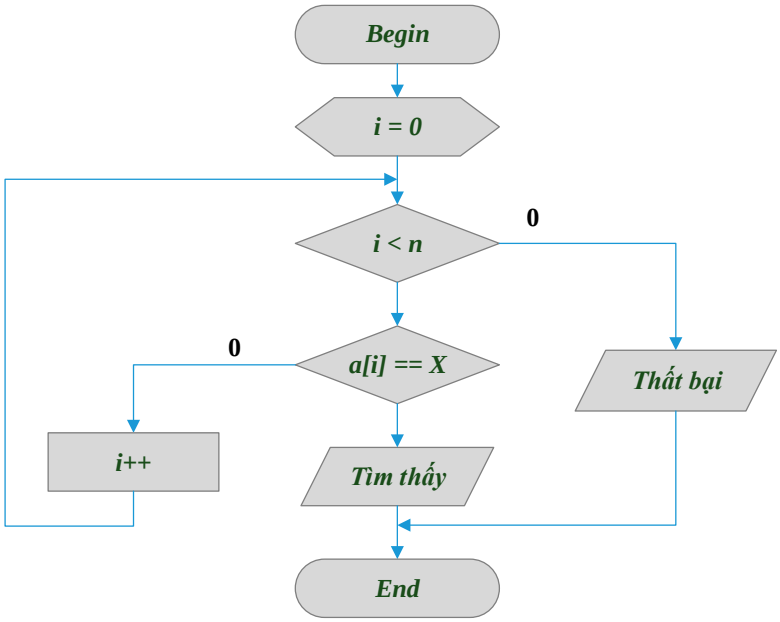
Cho một dãy gồm n bản ghi r[1…n]. Mỗi bản ghi r[i] tương ứng với một khóa k[i]. Hãy tìm bản ghi có khóa X cho trước. X được gọi là khóa tìm kiếm hay đối trị tìm kiếm (argument). Công việc tìm kiếm sẽ hoàn thành nếu như có một trong hai tình huống sau sảy ra:

- *Tìm được bản ghi có khóa tương ứng bằng X, lúc đó phép tìm kiếm thành công;*
- *Không tìm được bản ghi nào có khóa bằng X cả, phép tìm kiếm thất bại.*

Để đơn giản cho việc trình bày giải thuật thì bài toán có thể phát biểu như sau: Cho mảng A[n], hãy tìm trong A phần tử có khóa bằng X.

2.2 Giải thuật tìm kiếm tuyến tính

Ý tưởng của giải thuật tìm kiếm tuyến tính hay còn gọi là tìm kiếm tuần tự (Sequential search) như sau: So sánh X lần lượt với phần tử thứ 1, thứ 2,…của mảng A cho đến khi gặp được khóa cần tìm (tìm thấy), hoặc tìm hết mảng mà không thấy (không tìm thấy).



Hình 2. Giải thuật tìm kiếm tuyến tính

Hàm minh họa giải thuật tìm kiếm tuyến tính: Hàm trả về 1 nếu tìm thấy, ngược lại trả về 0:

```
int SequentialSearch(int A[], int n, int x)
{
        int i=0;
        while((i<n)&&(A[i]!=x))
                i++;
        if(i==n) return 0; //Tìm không thấy x
        else return 1; //Tìm thấy
}
```

Nhận xét: Số phép so sánh của thuật toán trong trường hợp xấu nhất là 2*n. Để giảm thiểu số phép so sánh trong vòng lặp cho thuật toán, ta thêm phần tử "lính canh" vào cuối dãy.

```
int SequentialSearch(int A[], int n, int x)
{
        int i=0, A[n]!=x ; //A[n] là phần tử lính canh
        while(A[i]!=x)
                i++;
        if(i==n)
                return 0; //Tìm không thấy x
        else
                return 1; //Tìm thấy
}
```

Thuật toán tìm kiếm tuyến tính cũng có thể cài đặt bằng for như sau:

```
int SequentialSearch(int A[], int n, int x)
{
        for(int i=0; i<n; i++)
        if(A[i]==x) return 1;
        return 0;
}
```

Tùy vào yêu cầu của bài toán mà ta có thể áp dụng giải thuật tìm kiếm tuyến tính một cách linh hoạt và phù hợp hơn. Chẳng hạn như các ví dụ sau:

Ví dụ 3: Tìm trong mảng A[n] các phần tử là số dương và chia hết cho 3.

```
#include<iostream>
using namespace std;
int main()
{
        int A[100], i, n, x, dem=0;
        cout<< "Nhap so phan tu cua mang: "; cin>>n;
        cout<< "Nhap du lieu cho mang:\n";
        for(i=0; i<n; i++)
        {
                cout<< "A["<<i<< "] = ";
                cin>>A[i];
        }
        cout<< "Cac phan tu duong ma chia het cho 3 la:\n";
        for(i=0;i<n;i++)
                if(A[i]>0&&A[i]%3==0)
                {
                        cout<<A[i]<< "\t";
                        dem++;
                }
        if(dem==0)
                cout<< "Khong co phan tu nao duong va chia het cho 3!";
}
```

Kết quả chạy thử chương trình trên:

```
Nhap so phan tu cua mang: 8
Nhap du lieu cho mang:
A[0] = 0
A[1] = 9
A[2] = 12
A[3] = -4
A[4] = 5
A[5] = -1
A[6] = 22
A[7] = 4
Cac phan tu duong ma chia het cho 3 la:
9       12
```

Ví dụ 4: Xóa khỏi mảng A[n] các phần tử dương có 2 chữ số.

```
#include<iostream>
using namespace std;
int main()
{
        int A[100], i, j, n, x;
        cout<< "Nhap so phan tu cua mang: "; cin>>n;
        cout<< "Nhap du lieu cho mang:\n";
        for(i=0; i<n; i++)
        {
                cout<< "A["<<i<< "] = ";
                cin>>A[i];
        }
        cout<< "Mang vua nhap la:\n";
        for(i=0; i<n; i++)
                cout<<A[i]<< "\t";
        for(i=0;i<n;i++)
                if(A[i]>9&&A[i]<100)
                {
                        for(j=i; j<n-1; j++)
                        A[j]=A[j+1]; //Phan tu dung truoc gan bang phan tu dung sau
                        n--; //Giam kich thuoc cua mang di 1
                        i--; //Giam I de kiem tra lai phan tu moi thay the
                }
        cout<< " \nMang sau khi xoa cac phan tu duong co 2 chu so la:\n";
        for(i=0; i<n; i++)
        cout<<A[i]<< "\t";
}
```

Kết quả chạy thử chương trình:

```
Nhap so phan tu cua mang: 7
Nhap du lieu cho mang:
A[0] = 3
A[1] = 11
A[2] = 44
A[3] = 55
A[4] = 6
A[5] = 9
A[6] = 122
Mang vua nhap la:
3       11      44      55      6       9       122
Mang sau khi xoa cac phan tu duong co 2 chu so la:
3       6       9       122
```

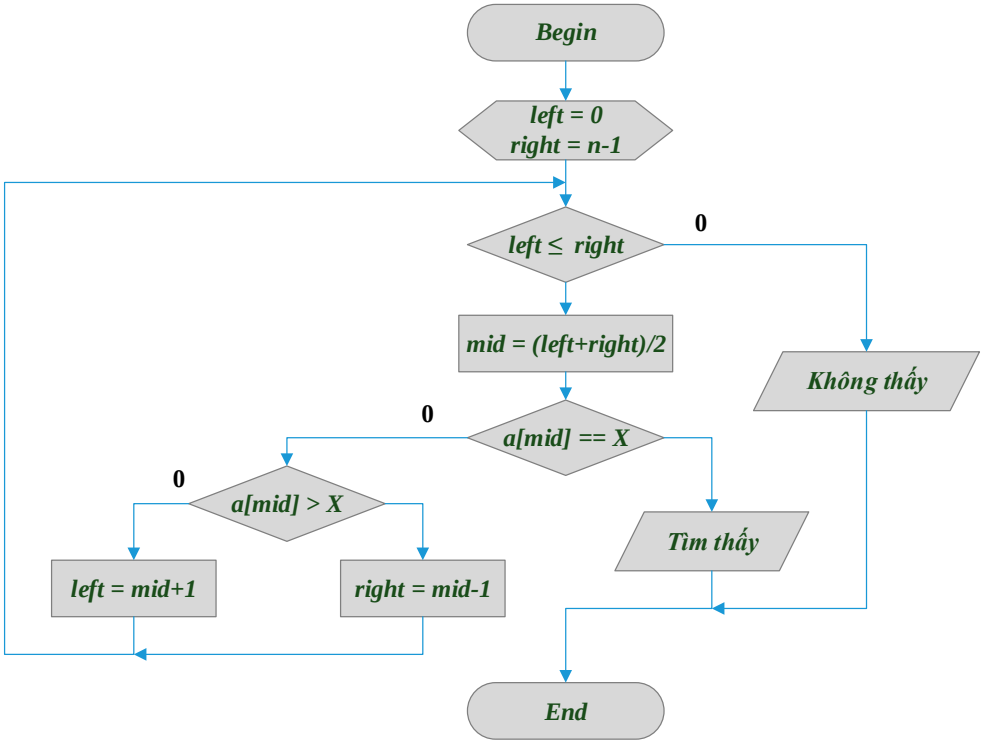
2.3 Giải thuật tìm kiếm nhị phân

Giải thuật tìm kiếm nhị phân được áp dụng trên mảng đã sắp xếp thứ tự. Giả sử mảng A[n] được sắp xếp tăng dần, khi đó ta có:

$$A[i-1] < A[i] < A[i+1]$$

Như vậy nếu X>A[i] thì X chỉ có thể nằm trong đoạn [A[i+1], A[n-1]], còn nếu X<A[i] thì X chỉ có thể nằm trong đoạn [A[0], A[i-1]].

Ý tưởng của giải thuật là tại mỗi bước ta so sánh X với phần tử đứng giữa trong dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này mà ta quyết định giới hạn dãy tìm kiếm ở nửa dưới hay nửa trên của dãy tìm kiếm hiện hành.



Hình 3. Sơ đồ khối của giải thuật tìm kiếm nhị phân

Hàm minh họa giải thuật tìm kiếm nhị phân: Hàm trả về giá trị 1 nếu tìm thấy, ngược lại hàm trả về giá trị 0:

```
int BinarySearch(int a[], int n, int x)
{
        int left, right, mid;
        left=0; right=n-1;
        do
        {
                mid=(left+right)/2;
                if(a[mid]==x) return 1;
                else
                        if(a[mid]<x)  left = mid+1;
                        else  right = mid-1;
        }
        while(left<=right);
        return 0;
}
```

Ví dụ 5: Minh họa sử dụng giải thuật tìm kiếm nhị phân.

```
#include<iostream>
using namespace std;
int BinarySearch(int a[], int n, int x)
{
	int left, right, mid;
	left=0; right=n-1;
	do
	{
		mid=(left+right)/2;
		if(a[mid]==x)
			return 1;
		else
			if(a[mid]<x)
				left = mid+1;
			else
				right = mid-1;
	}
	while(left<=right);
	return 0;
}
int main()
{
	int A[]={1, 2, 3, 4, 5, 6, 7, 8, 9}, i, x;
	cout<< "Nhap vao gia tri ma ban muon tim kiem: ";
	cin>>x;
	i=BinarySearch(A, 9, x);
	if(i==1)
		cout<<x<< " co trong mang A";
	else
		cout<<x<< " khong co trong mang A";
}
```

Kết quả chạy thử chương trình:

```
Nhap vao gia tri ma ban muon tim kiem: 5
5 co trong mang A
```

2.4 Đánh giá độ phức tạp của các giải thuật tìm kiếm

Giải thuật tìm kiếm	Trường hợp tốt nhất	Trường hợp trung bình	Trường hợp xấu nhất
Tìm kiếm tuyến tính	$O(1)$	$O((N+1)/2)$	$O(N)$
Tìm kiếm nhị phân	$O(1)$	$O(log_2N/2)$	$O(log_2N)$

BÀI TẬP CHƯƠNG 2

Bài 1. Viết chương trình nhập dữ liệu cho mảng số nguyên A[n], với 0<n<100. Hãy tìm trong A các phần tử là số lẻ và lưu vào mảng B.

Bài 2. Nhập dữ liệu và sắp xếp mảng A[n] tăng dần. Sử dụng giải thuật tìm kiếm nhị phân để tìm phần tử có giá trị bằng X ở trong mảng A sau đó xóa nó khỏi A nếu tìm thấy.

Bài 3. Nhập vào một chuỗi S bất kì. Đếm xem trong chuỗi S có bao nhiêu kí tự khoảng trống, bao nhiêu kí tự số, bao nhiêu kí tự là chữ cái in hoa ?

Chương 3. CÁC GIẢI THUẬT SẮP XẾP

3.1 Bài toàn sắp xếp

Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng nào đó theo một thứ tự nhất định. Chẳng hạn như thứ tự tăng dần hay giảm dần đối với một dãy số, thứ tự từ điển đối với các từ ….Yêu cầu sắp xếp thường xuyên xuất hiện trong các ứng dụng tin học với các mục đích khác nhau: sắp xếp dữ liệu trong máy tính để tìm kiếm cho thuận lời, sắp xếp các kết quả xử lý để in ra trên bảng biểu…

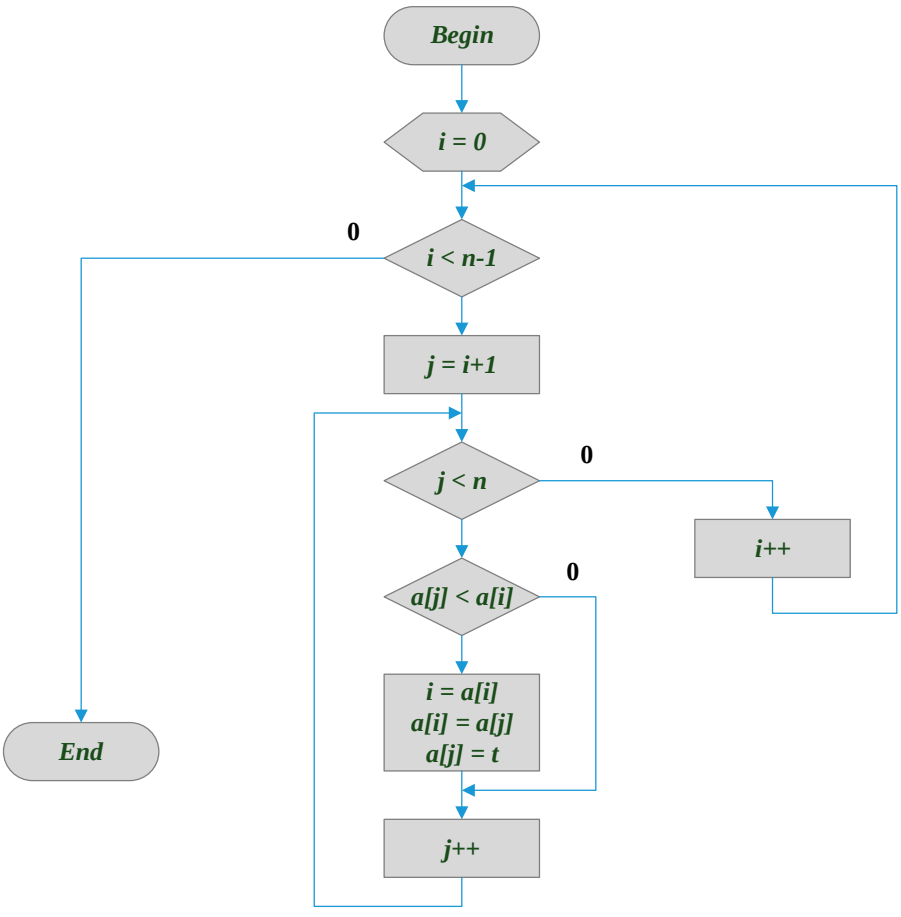
Dưới đây là một số giải thuật sắp xếp thông dụng, trong đó ta sẽ minh họa bằng việc sắp xếp mảng **a[n]** theo thứ tự tăng dần.

3.2 Đổi chỗ trực tiếp

Ý tưởng: Xuất phát từ đầu dãy, tìm tất các các nghịch thế chứa phần tử này (hai phần tử gọi là **ngịch thế** nếu khi sắp xếp tăng dần mà phần tử trước có giá trị lớn hơn phần tử đứng sau), triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy.

Các bước thực hiện:

- *Bước 1: i = 0; // bắt đầu từ đầu dãy*
- *Bước 2: j = i+1; //tìm các nghịch thế với a[i]*
- *Bước 3: Trong khi j < N thực hiện:*
  Nếu a[j]<a[i] thì đổi chỗ a[i], a[j]
  j = j+1;
- *Bước 4: i = i+1;*
  Nếu i < N-1: Lặp lại Bước 2. Ngược lại: Dừng.

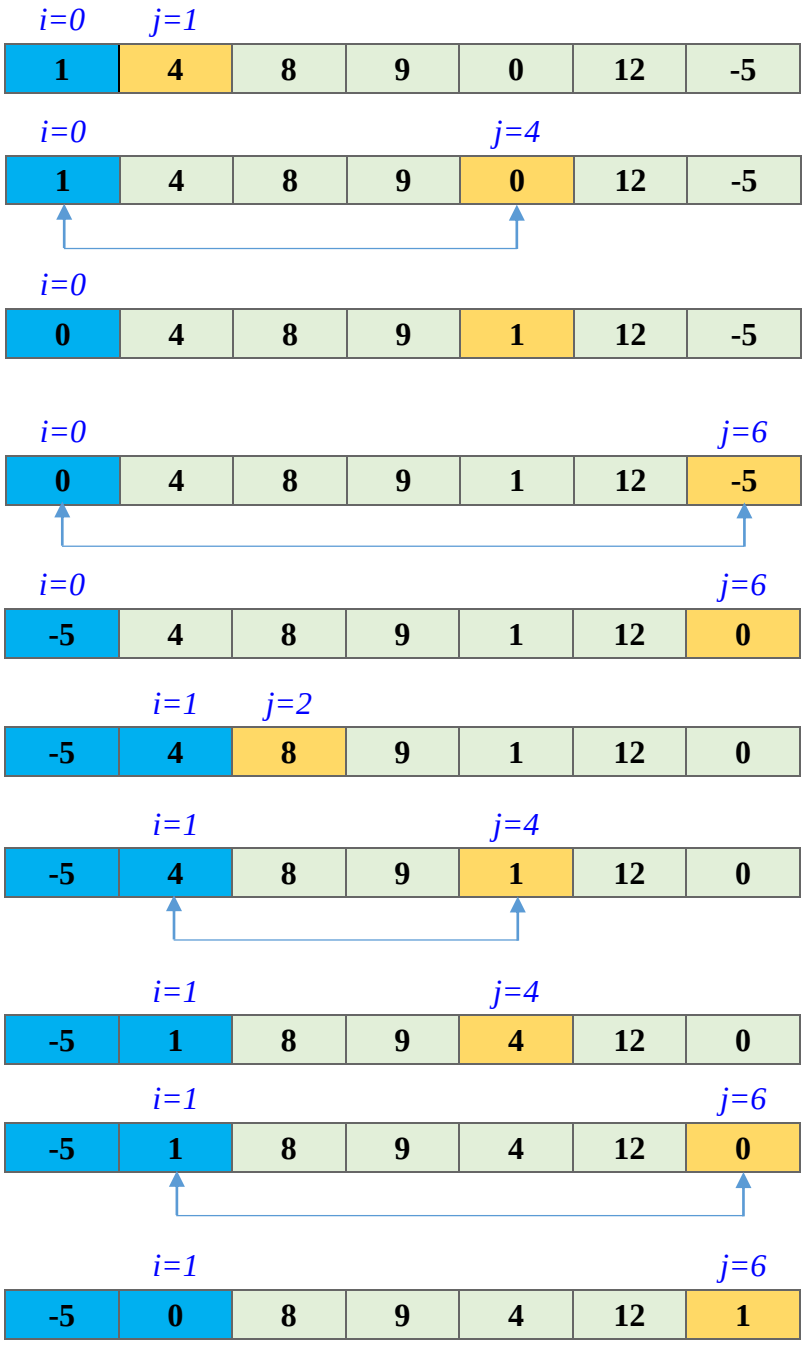


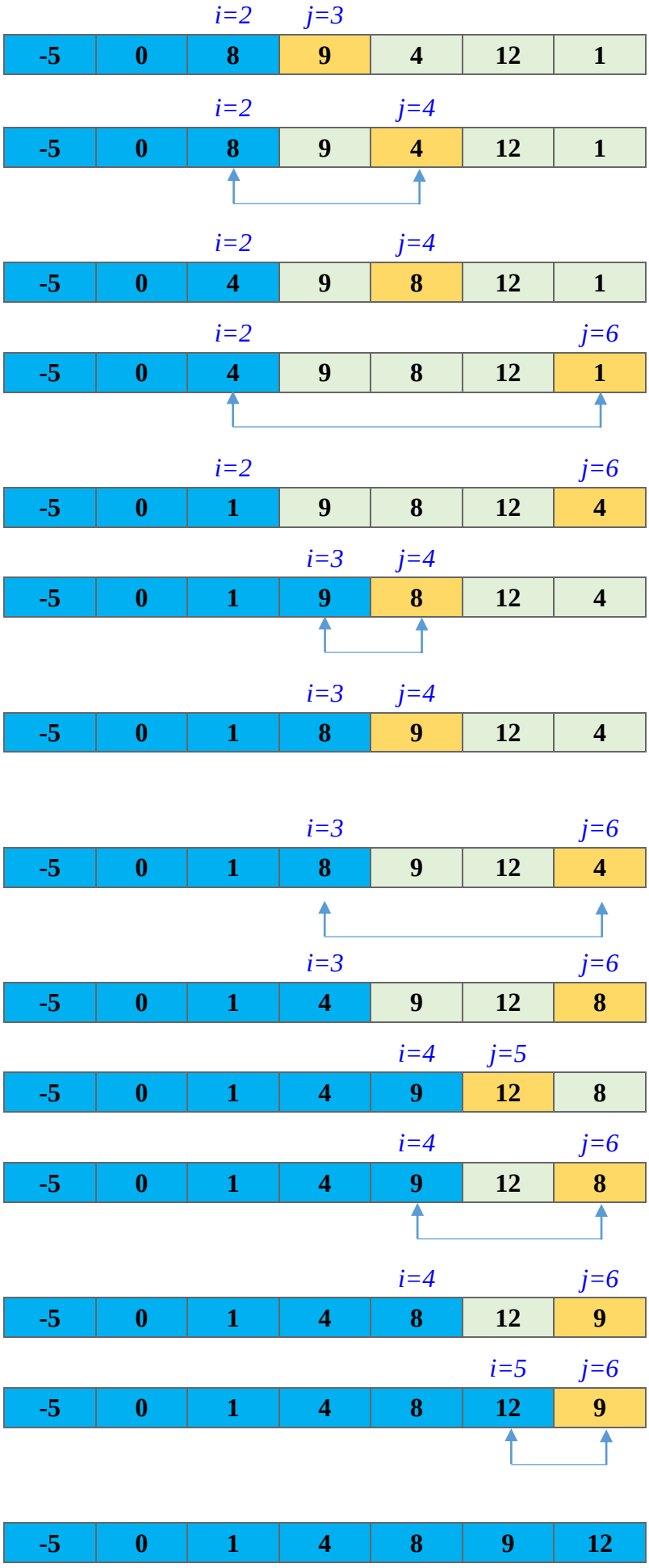
Hình 4. Giải thuật sắp xếp đổi chỗ trực tiếp

Hàm minh họa thuật toán đổi chỗ trực tiếp:

```
void InterchangeSort(int a[], int N)
{
        int i, j;
        for(i=0; i<N-1; i++)
                for(j=i+1; j<N; j++)
                if(a[j]<a[i]) // Thỏa mãn là cặp nghịch thế
                { // Đổi chỗ a[i] và a[min] cho nhau
                        int t=a[i]; a[i]=a[j]; a[j]=t;
                }
}
```

Ví dụ 6: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật đổi chỗ trực tiếp.




i=2 j=3
-5 0 8 9 4 12 1
i=2 j=4
-5 0 8 9 4 12 1
i=2 j=4
-5 0 4 9 8 12 1
i=2 j=6
-5 0 4 9 8 12 1
i=2 j=6
-5 0 1 9 8 12 4
i=3 j=4
-5 0 1 9 8 12 4
i=3 j=4
-5 0 1 8 9 12 4
i=3 j=6
-5 0 1 8 9 12 4
i=3 j=6
-5 0 1 4 9 12 8
i=4 j=5
-5 0 1 4 9 12 8
i=4 j=6
-5 0 1 4 9 12 8
i=4 j=6
-5 0 1 4 8 12 9
i=5 j=6
-5 0 1 4 8 12 9
-5 0 1 4 8 9 12

3.3 Chọn trực tiếp

Ý tưởng của giải thuật:

- *Tại vị trí đầu tiên của dãy, chọn phần tử nhỏ nhất trong N phần tử trong dãy hiện hành ban đầu và đưa phần tử này về vị trí 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 thứ 2, chọn phần tử nhỏ nhất trong số N-1 phần tử kể trên và đưa nó về vị trí đầu của dãy gồm N-1 phần tử đang xét.*
- *Lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử.*

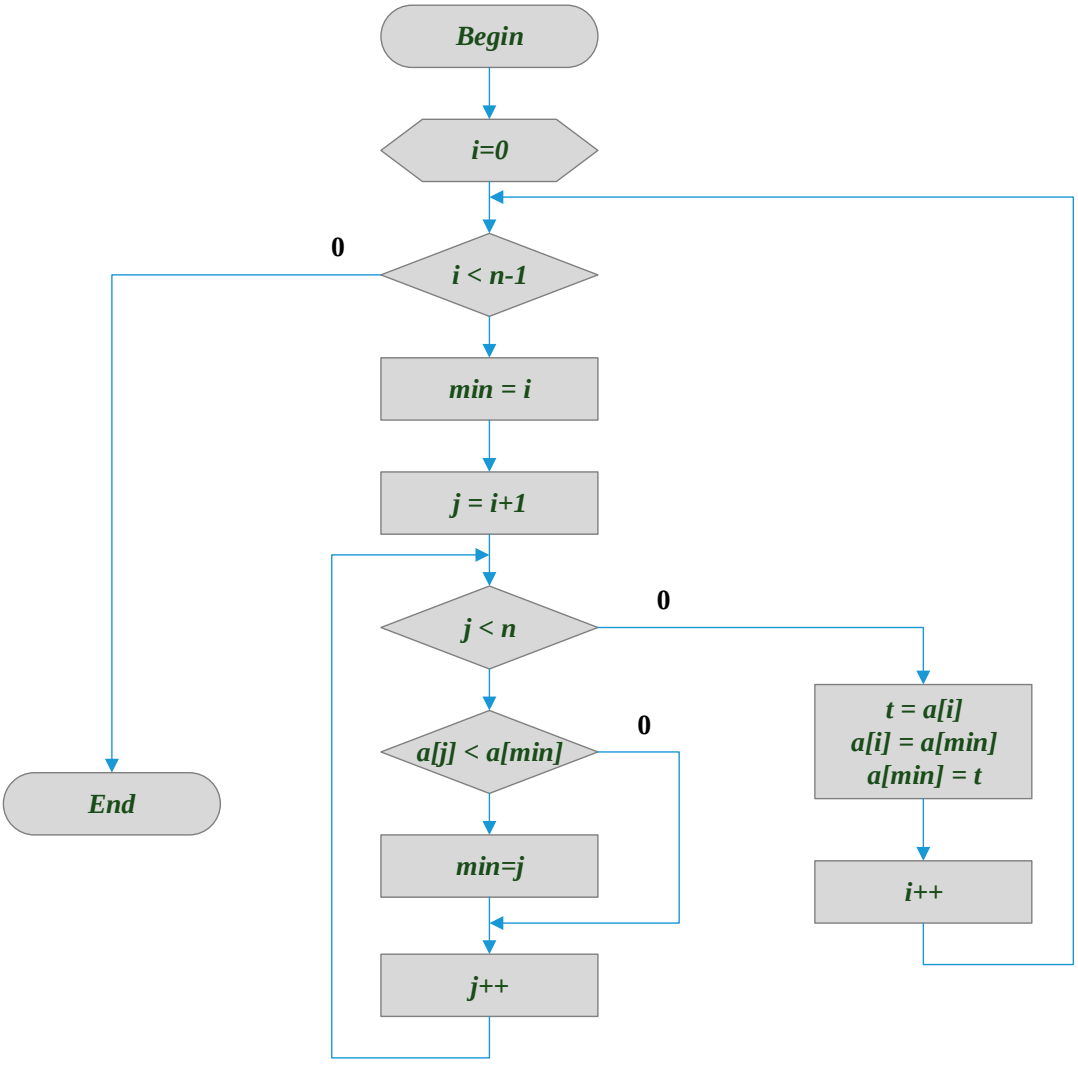
Các bước thực hiện:

- *Bước 1: i = 0;*
- *Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]*
- *Bước 3 : Đổi chỗ a[min] và a[i]*
- *Bước 4 : Nếu i < N-1 thì:*

  i = i+1;

  Lặp lại Bước 2;

  Ngược lại: Dừng.

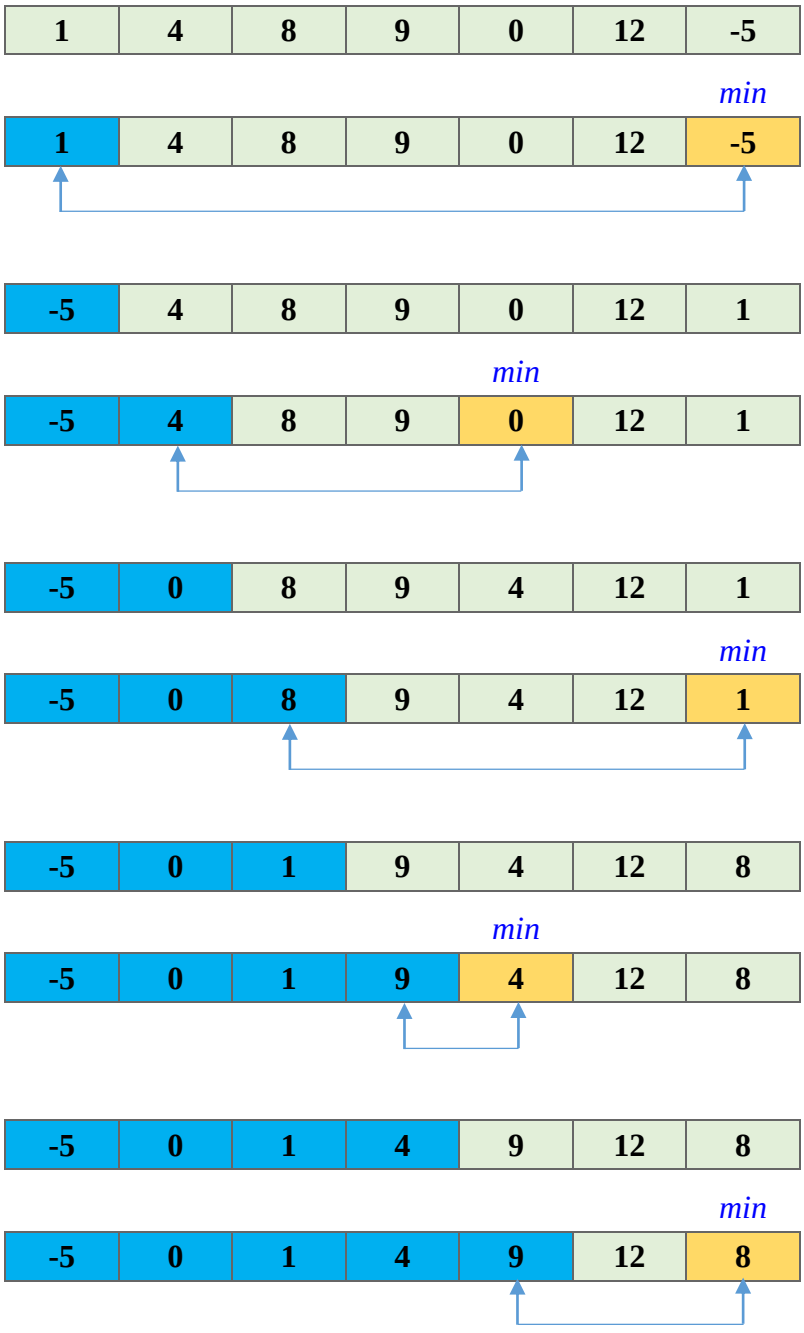


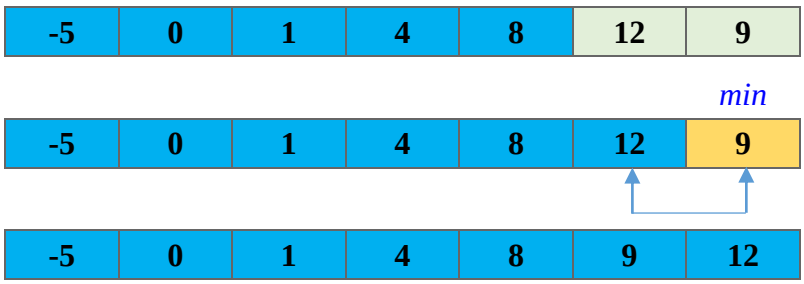
Hình 5. Giải thuật sắp xếp chọn trực tiếp

Hàm minh họa thuật toán chọn trực tiếp:

```
void SelectionSort(int a[], int n )
{
        int min, i, j; // min dùng để lưu vị trí của phần tử nhỏ nhất
        for (i=0; i<n -1 ; i++)
        {
                min = i;
                for(j = i+1; j <n ; j++)
                        if (a[j] < a[min]) min = j;
                {       int t=a[i]; a[i]=a[min]; a[min]=t;    }
        }
  }
```

Ví dụ 7: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật chọn trực tiếp.





3.4 Chèn trực tiếp

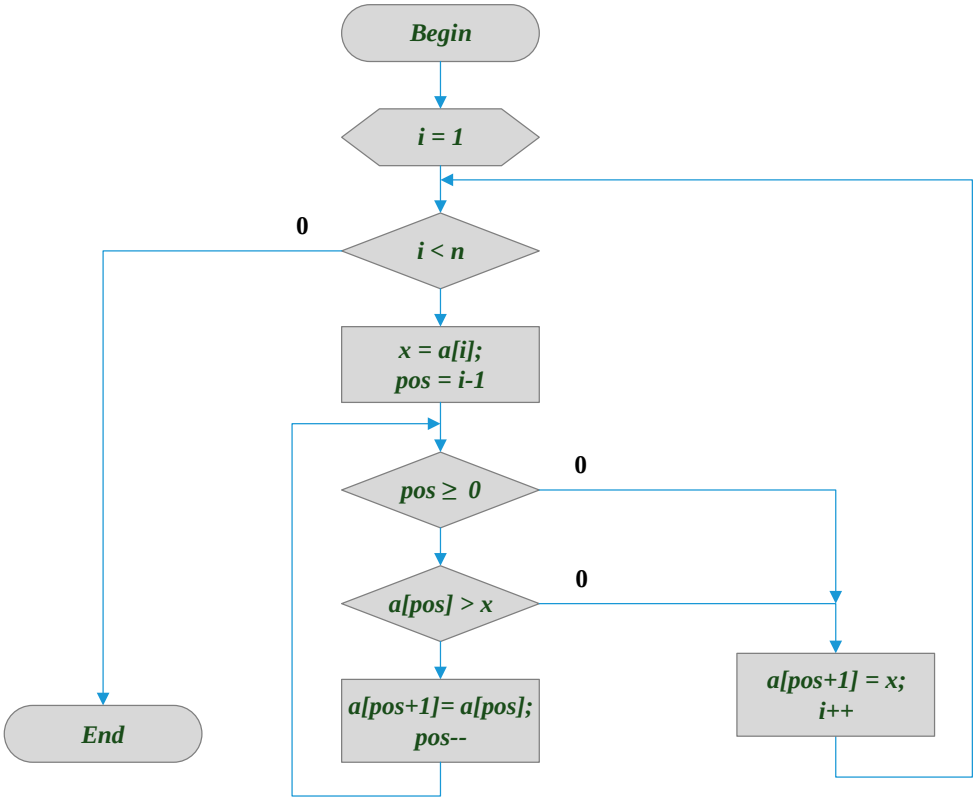
Ý tưởng của giải thuật:

- *Ban đầu coi như phần tử đầu tiên đã sắp xếp.*
- *So sánh phần tử thứ 2 với phần tử đầu tiên để đổi chỗ nếu cần để sao cho 2 phần tử này được sắp xếp theo thứ tự chỉ định.*
- *Tìm vị trí để chèn phần tử thứ 3 của dãy hiện hành vào dãy 2 phần tử đầu đã được sắp xếp ở trên sao cho dãy vẫn được sắp xếp đúng thứ tự.*
- *Lặp lại liên tục các quá trình trên cho tới khi hết dãy.*

Các bước thực hiện:

- *Bước 1: i = 1; //Giả sử có đoạn a[0] đã được sắp*
- *Bước 2: x = a[i]; //Tìm vị trí position thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào*
- *Bước 3: Dời chỗ các phần tử từ a[position] đến a[i-1]sang phải 1 vị trí để dành chổ cho a[i]*
- *Bước 4: a[position] = x; //Chèn a[i] vào vị trí tìm được => có đoạn a[1]..a[i] đã được sắp*
- *Bước 5: i = i+1;*

Nếu i < n => Lặp lại Bước 2 . Ngược lại => Dừng.

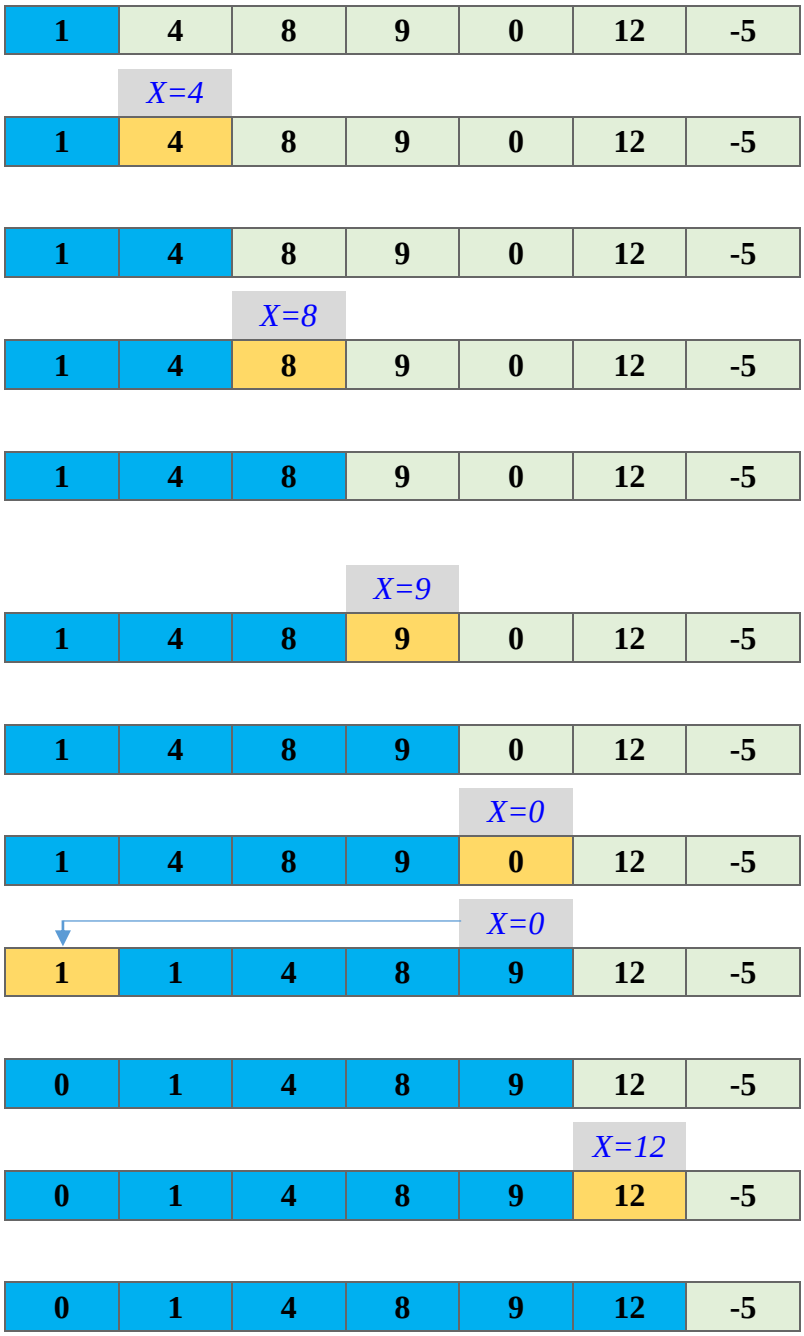


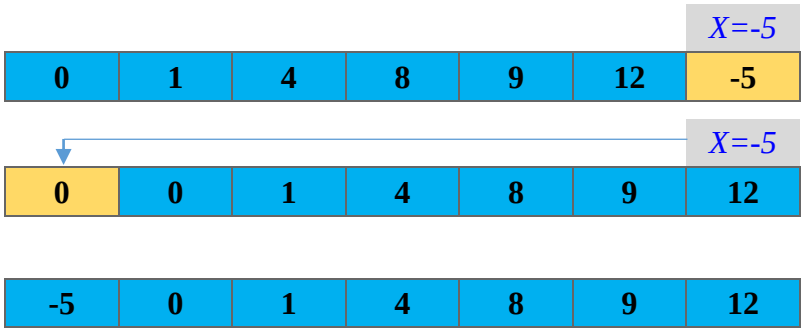
Hình 6. Giải thuật sắp xếp chèn trực tiếp

Hàm minh họa giải thuật sắp xếp chèn trực tiếp:

```
void InsertionSort(int a[], int n )
{       int pos, i, x;
        for(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;
        }
}
```

Ví dụ 8: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật chèn trực tiếp.





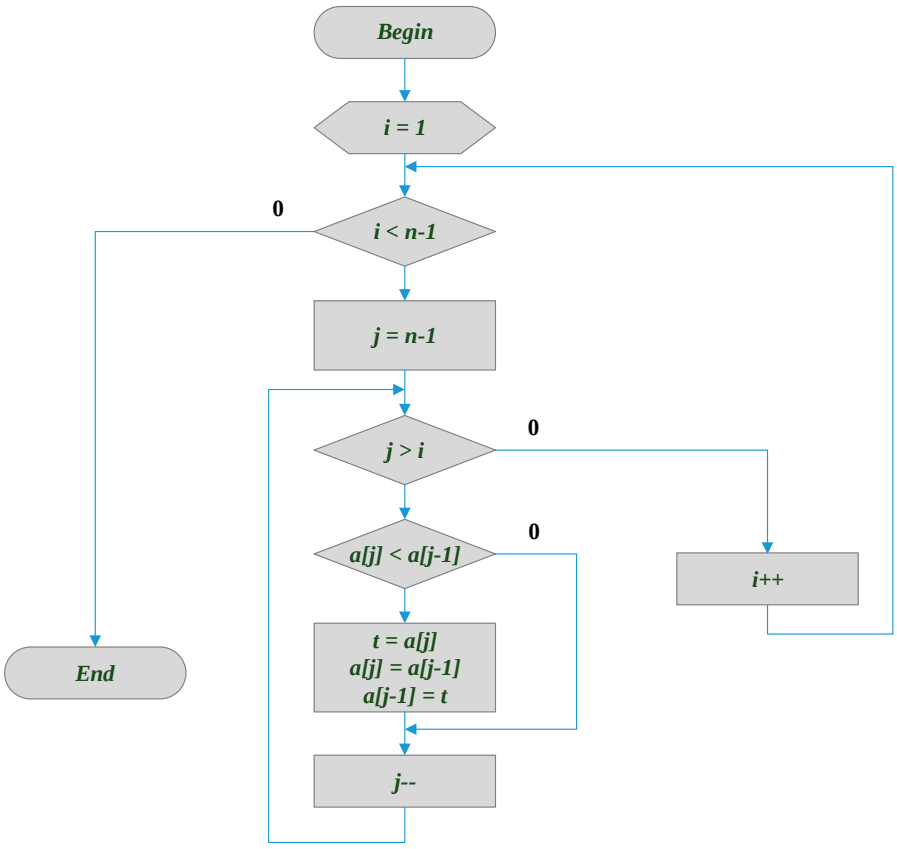
3.5 Bubble Sort

Ý tưởng của giải thuật sắp xếp Bubble Sort (sắp xếp nổi bọt):

- *Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i.*
- *Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.*

Các bước thực hiện giải thuật:

- *Bước 1 : i = 0; // Lần xử lý đầu tiên*
- *Bước 2 : j = N-1; //Duyệt từ cuối dãy ngược về vị trí i*
  Trong khi (j > i) thực hiện:
  Nếu a[j]<a[j-1] => Doicho(a[j],a[j-1]);
  j = j-1;
- *Bước 3 : i = i+1; // Lần xử lý kế tiếp*
  Nếu i =N-1 => Dừng
  Ngược lại: Lặp lại Bước 2.



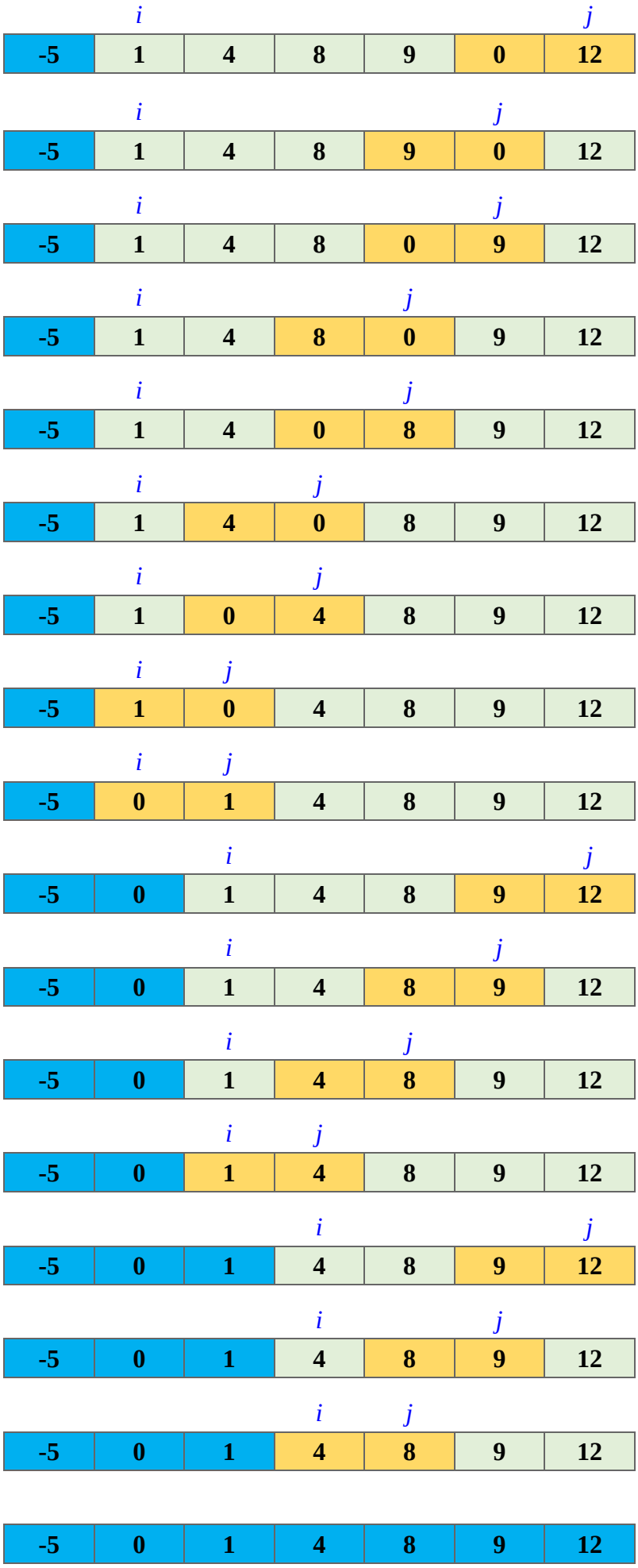
Hình 7. Giải thuật sắp xếp nổi bọt (bubble sort)

Hàm mình họa giải thuật sắp xếp nổi bọt:

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

Ví dụ 9: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật sắp xếp nổi bọt.

						j
1	4	8	9	0	12	-5
						j
1	4	8	9	0	-5	12
				j		
1	4	8	9	0	-5	12
				j		
1	4	8	9	-5	0	12
			j			
1	4	8	9	-5	0	12
			j			
1	4	8	-5	9	0	12
		j				
1	4	8	-5	9	0	12
		j				
1	4	-5	8	9	0	12
	j					
1	4	-5	8	9	0	12
	j					
1	-5	4	8	9	0	12
j						
1	-5	4	8	9	0	12
-5	1	4	8	9	0	12


i j
-5 1 4 8 9 0 12
i j
-5 1 4 8 9 0 12
i j
-5 1 4 8 0 9 12
i j
-5 1 4 8 0 9 12
i j
-5 1 4 0 8 9 12
i j
-5 1 4 0 8 9 12
i j
-5 1 0 4 8 9 12
i j
-5 1 0 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
i j
-5 0 1 4 8 9 12
.....
-5 0 1 4 8 9 12

3.6 Quick Sort

Ý tưởng của giải thuật sắp xếp nhanh (Quick Sort):

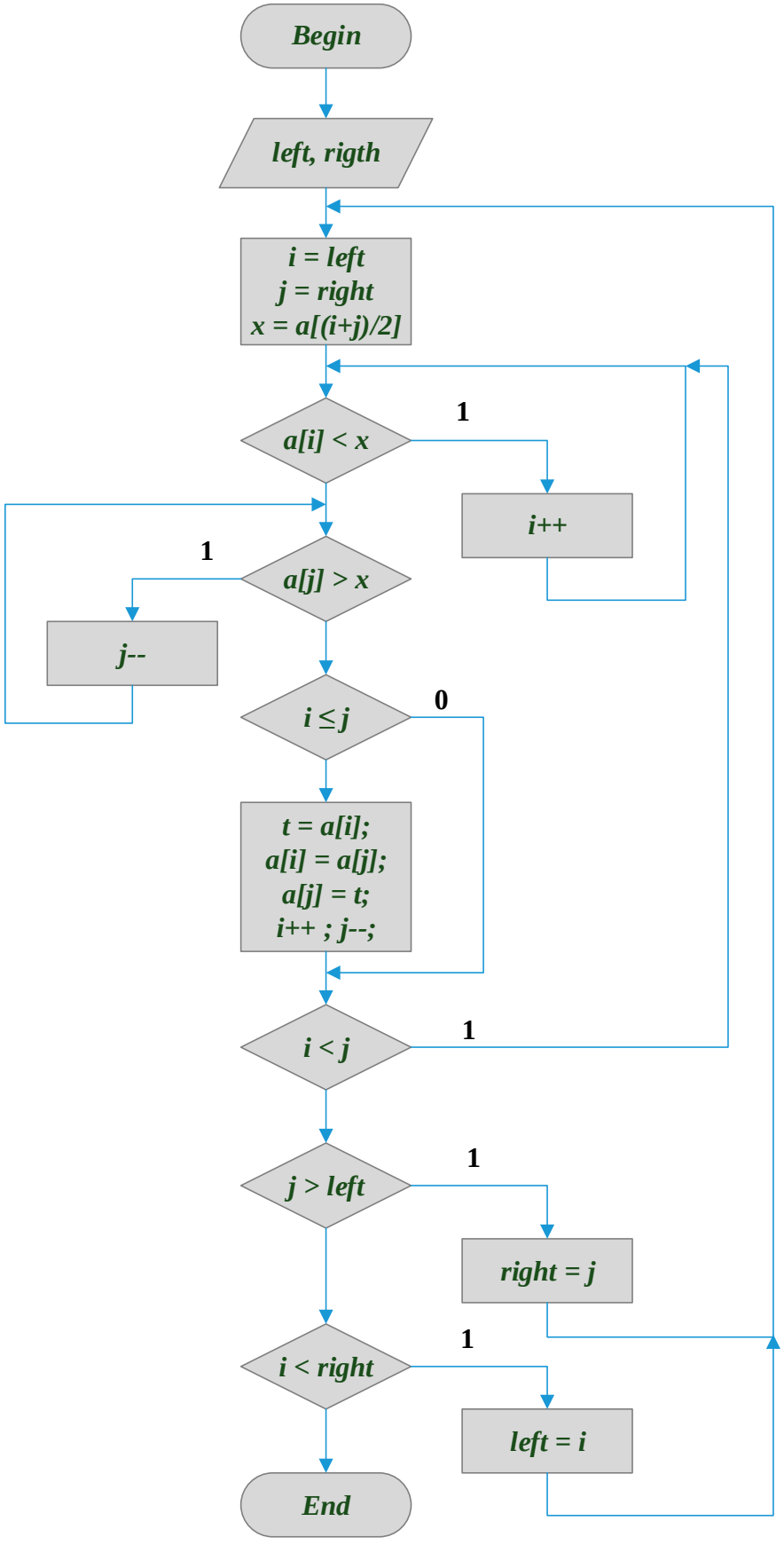
- *QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt.*
- *Những phần tử nhỏ hơn phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn hoặc bằng phần tử chốt được đưa về phía sau và thuộc danh sách con thứ hai.*
- *Cứ tiếp tục chia các danh sách con như vậy tới khi các danh sách con đều có độ dài bằng 1.*

Các bước thực hiện:

- *Bước 1: Lấy 1 phần tử bất kì của dãy làm phần tử chốt X (giả sử lấy phần tử đầu tiên hoặc phần tử giữa của dãy,...).*
- *Bước 2: Tiến hành duyệt từ bên trái dãy cho đến khi gặp phần tử lớn hơn hoặc bằng X thì dừng. Đồng thời duyệt từ bên phải dãy cho đến khi gặp phần tử nhỏ hơn hoặc bằng X thì dừng.*
- *Bước 3: Đổi chỗ 2 phần tử tìm được ở bước 2.*
- *Bước 4: Tiếp tục duyệt cho đến khi 2 biến duyệt từ 2 chiều gặp nhau. Khi đó dãy ban đầu đã phân thành 2 phần: phần trái gồm những phần tử nhỏ hơn X, phần phải gồm những phần tử lớn hơn hoặc bằng X.*
- *Bước 5: Lặp lại quá trình phân hoạch đối với phần trái và phần phải mới được tạo ở trên cho đến khi nào dãy được sắp xếp hoàn toàn.*

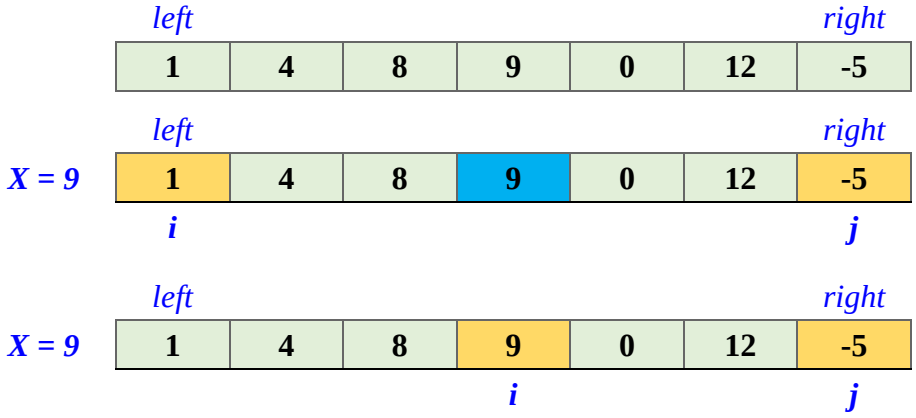
Hàm minh họa giải thuật Quick Sort:

```
void QuickSort(int a[], int left, int right)
{
        int i, j, x, t;
        x = a[(left+right)/2]; // x có thể chọn phần tử khác bất kì trong khoảng left - right
        i = left;
        j = right;
        do
        {
                while(a[i] < x)
                        i++;
                while(a[j] > x)
                        j--;
                if(i <= j)
                {
                        t = a[i];
                        a[i] = a[j];
                        a[j] = t;
                        i++ ; j--;
                }
        }
        while(i < j);
        if(left<j)
                QuickSort(a, left, j);
        if(i<right)
                QuickSort(a, i, right);
}
```

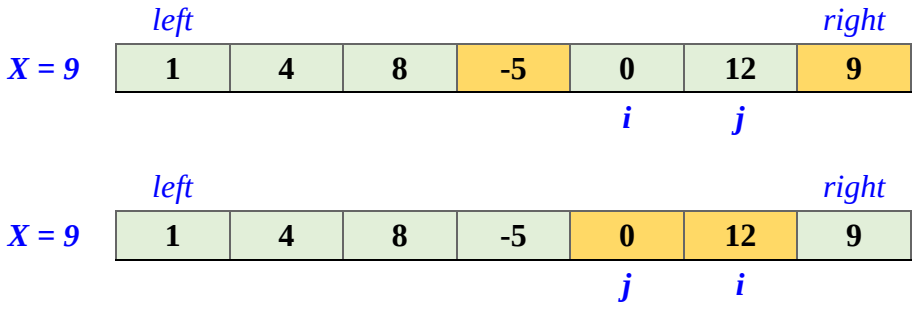


Hình 8. Giải thuật sắp xếp Quick Sort

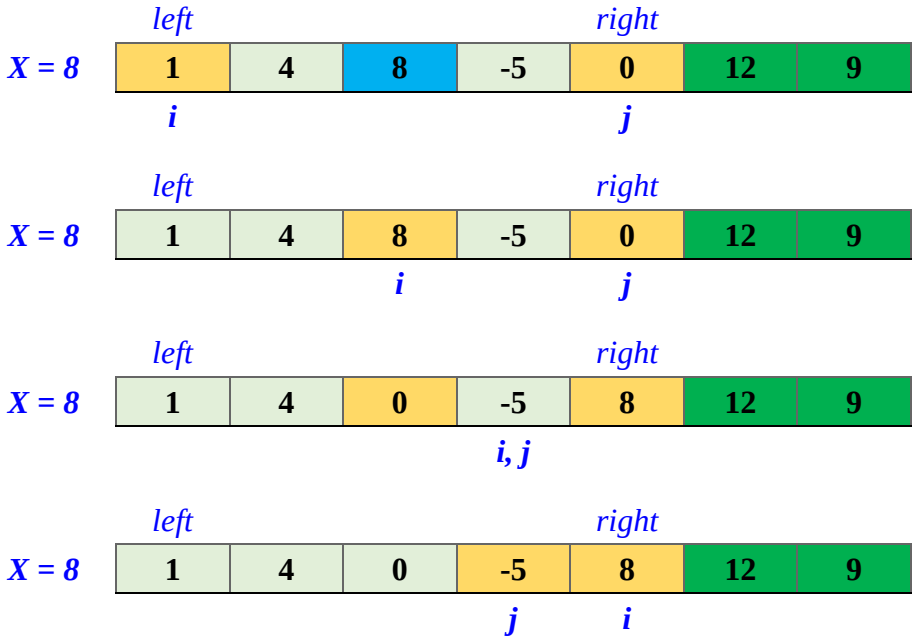
Ví dụ 10: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật sắp xếp Quick Sort.



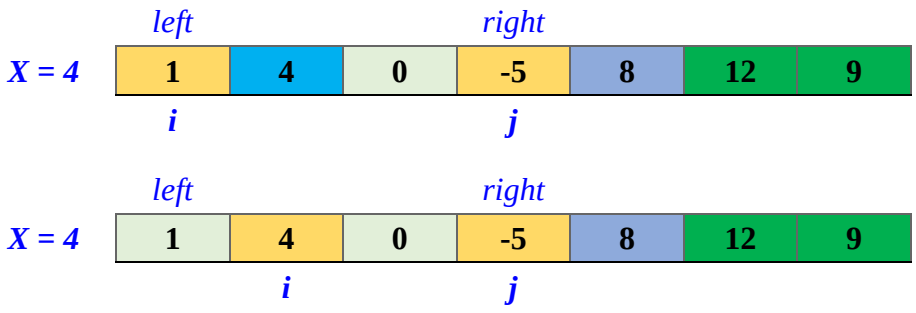
Đổi chỗ a[i], a[j] sau đó tăng i và giảm j đi 1 đơn vị.

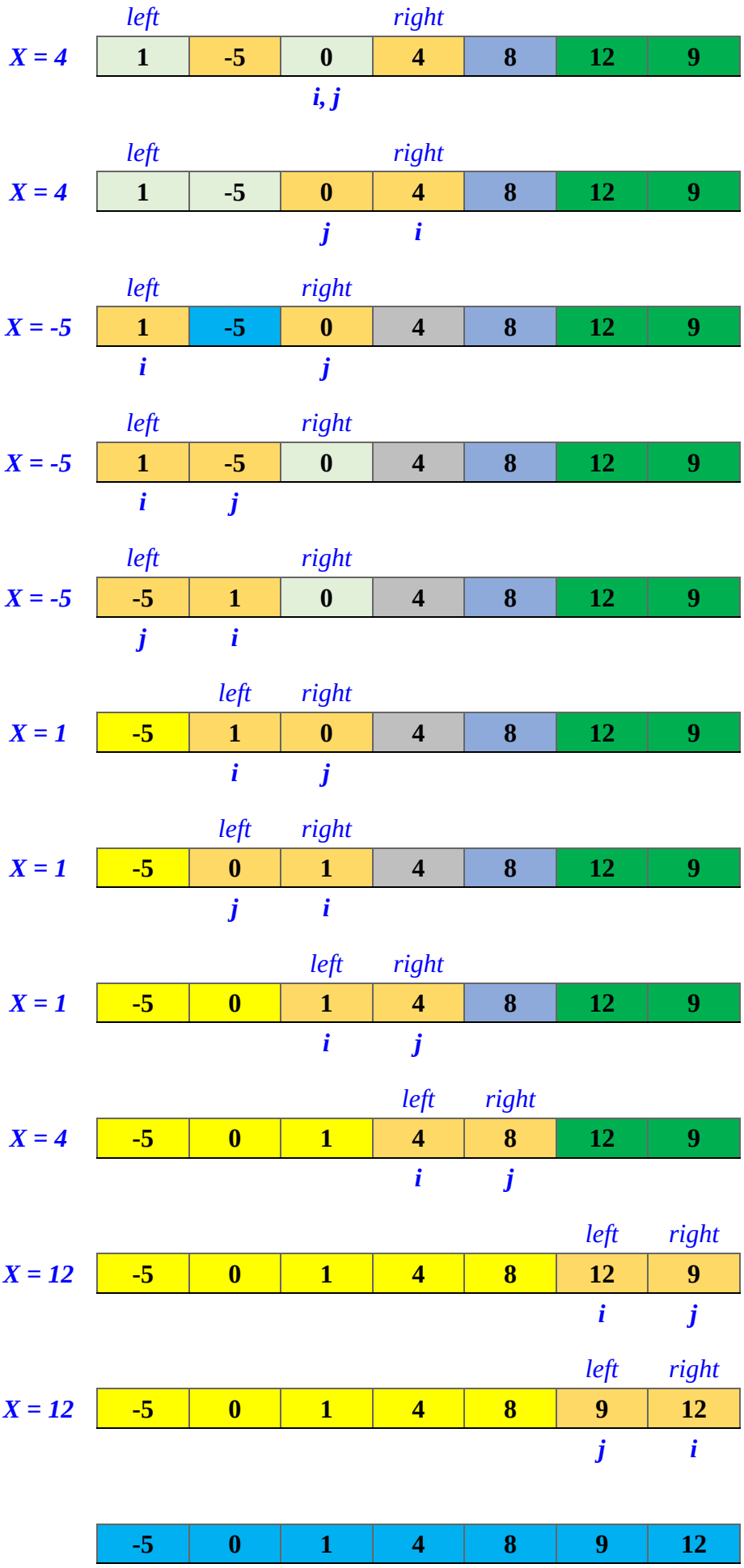


Phân hoạch đoạn left đến j.



Phân hoạch đoạn từ left đến j.




left
right
X = 4
1 -5 0 4 8 12 9
i, j
left
right
X = 4
1 -5 0 4 8 12 9
j i
left
right
X = -5
1 -5 0 4 8 12 9
i j
left
right
X = -5
1 -5 0 4 8 12 9
i j
left
right
X = -5
-5 1 0 4 8 12 9
j i
left
right
X = 1
-5 1 0 4 8 12 9
i j
left
right
X = 1
-5 0 1 4 8 12 9
j i
left
right
X = 1
-5 0 1 4 8 12 9
i j
left
right
X = 4
-5 0 1 4 8 12 9
i j
left
right
X = 12
-5 0 1 4 8 12 9
i j
left
right
X = 12
-5 0 1 4 8 9 12
j i
-5 0 1 4 8 9 12

3.7 Heap Sort

Ý tưởng của giải thuật Heap Sort: Sắp xếp thông qua việc tạo các heap (đống) từ dãy ban đầu, trong đó heap là 1 cây nhị phân hoàn chỉnh có tính chất là khóa ở nút cha bao giờ cũng lớn hơn khóa ở các nút con.

Các bước thực hiện: Việc thực hiện giải thuật được chia thành 2 giai đoạn:

- *Giai đoạn 1: Tạo heap từ dãy ban đầu, khi đó nút gốc của heap là phần tử lớn nhất.*
- *Giai đoạn 2: Sắp xếp dãy dựa trên heap được tạo.*
  - *Nút gốc là nút có giá trị lớn nhất, ta chuyển về vị trí cuối cùng của heap.*
  - *Loại phần tử lớn nhất khỏi heap.*
  - *Hiệu chỉnh phần còn lại thành heap.*
  - *Lặp lại quá trình cho tới khi heap chỉ còn 1 nút.*

Hàm hiệu chỉnh heap:

```
void Shift(int a[], int l, int r)
{
	int x, i, j;
	i=l;
	j=2*i+1;
	x=a[i];
	while(j<=r)
	{
		if(j<r)
			if(a[j]<a[j+1]) //Tim phan tu lon nhat giua a[j] va a[j+1]
				j++; //Luu chi so cua phan tu lon nhat trong hai phan tu
		if(a[j]<=x) return;
		else
		{
			a[i]=a[j];
			a[j]=x;
			i=j;
			j=2*i+1;
			x=a[i];
		}
	}
}
```

Hàm tạo heap từ dãy ban đầu:

```
void CreateHeap(int a[], int n)
{
	int l=n/2-1;
	while(l>=0)
	{
		Shift(a, l, n-1);
		l=l-1;
	}
}
```

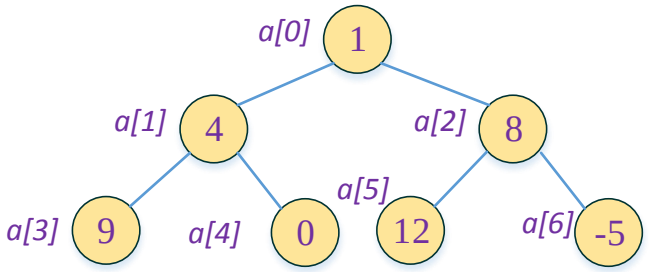
Hàm sắp xếp Heap Sort:

```
void HeapSort(int a[], int n)
{
	int r;
	CreateHeap(a, n);
	r=n-1;
	while(r>0)
	{
		int t=a[0];
		a[0]=a[r];
		a[r]=t;
		r--;
		if(r>0)
			Shift(a, 0, r);
	}
}
```

Ví dụ 11: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật sắp xếp Heap Sort.

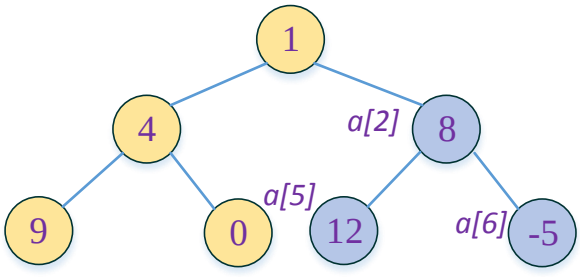
1	4	8	9	0	12	-5

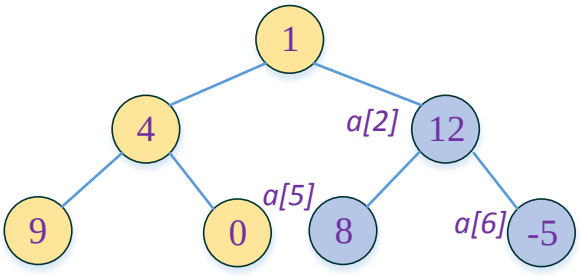
Tạo heap từ dãy ban đầu, để đơn giản ta biểu diễn lại dãy dưới dạng cây nhị phân như sau:



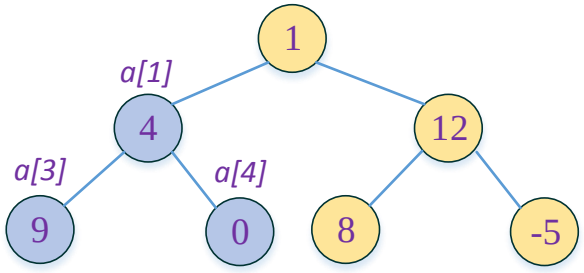
Các bước hiệu chỉnh dãy trên thành heap như sau:

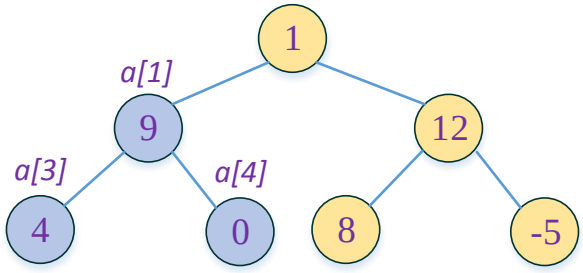
l=2, i=2, j=5, x=8:



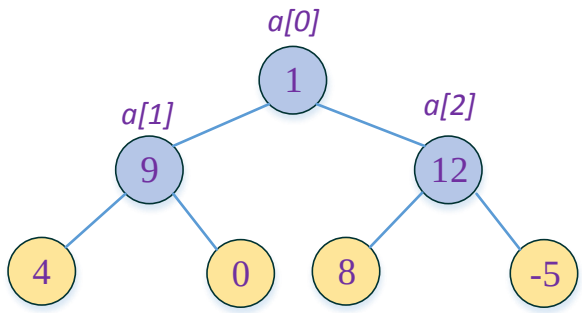


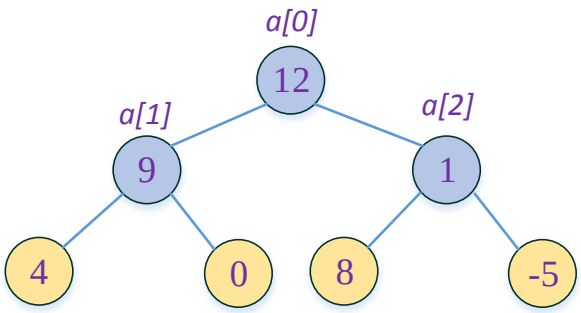
l=1, i=1, j=3, x=4:



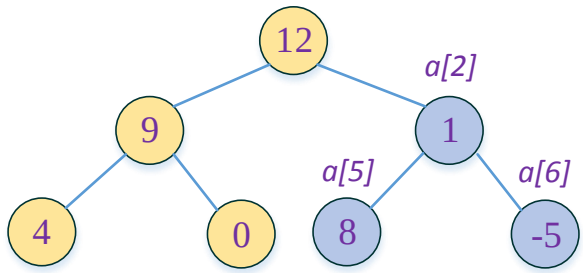


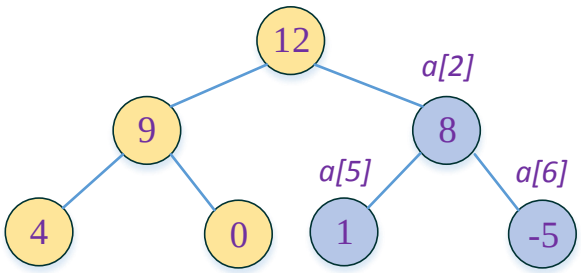
l=0, i=0, j=1, x=1. *Vì 9<12 nên j=j+1=2*



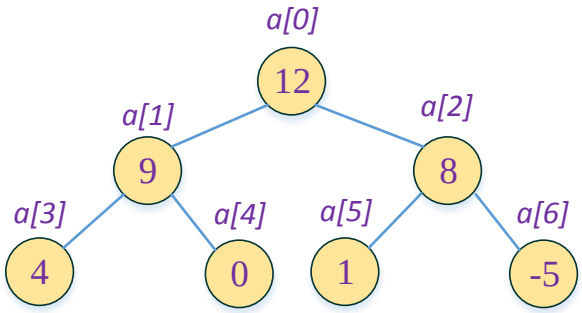


i=j=2, j=5, x=1:

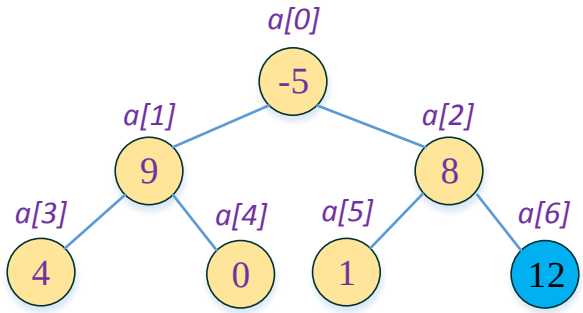


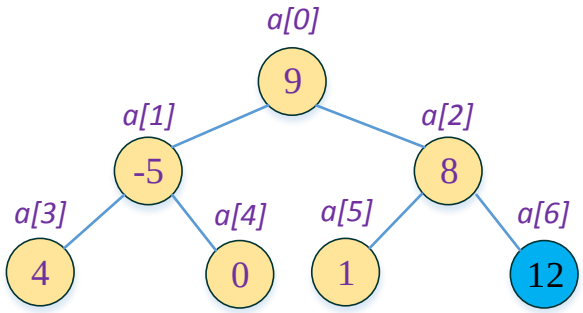


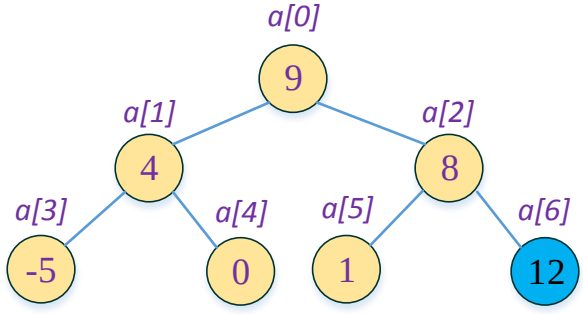
Kết quả ta có dãy ban đầu đã được tạo thành heap:

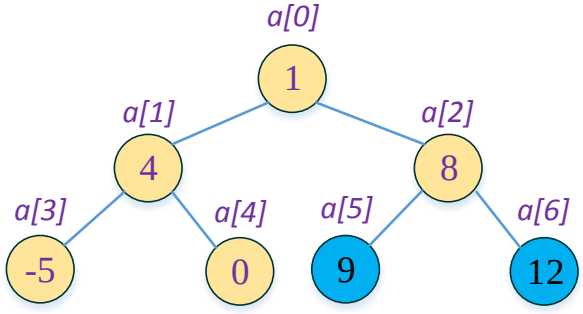


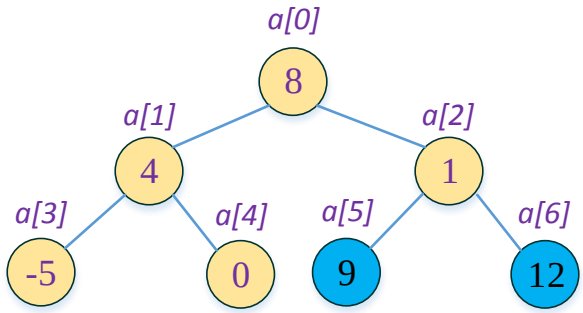
Lấy phần tử lớn nhất ra khỏi heap đưa về cuối dãy và hiệu chỉnh phần còn lại thành heap:

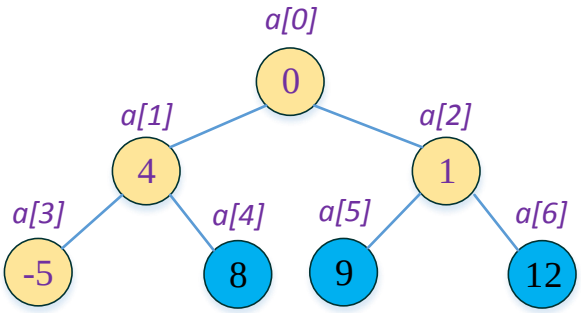


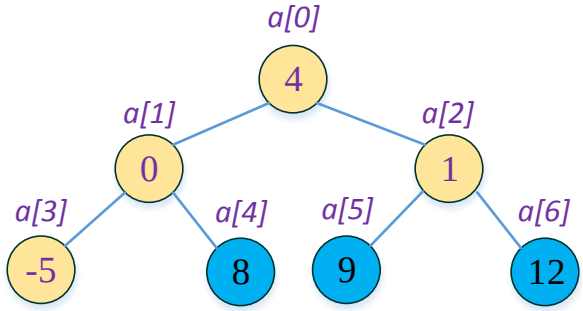


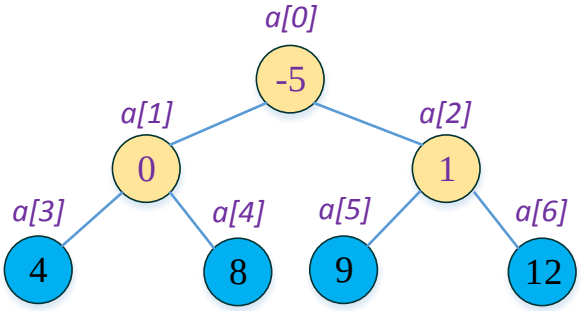


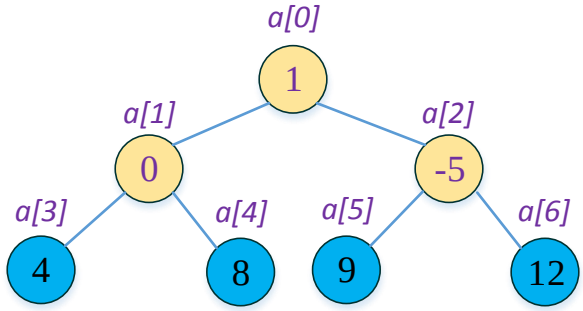


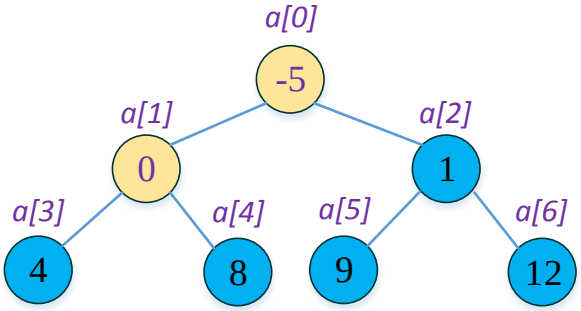


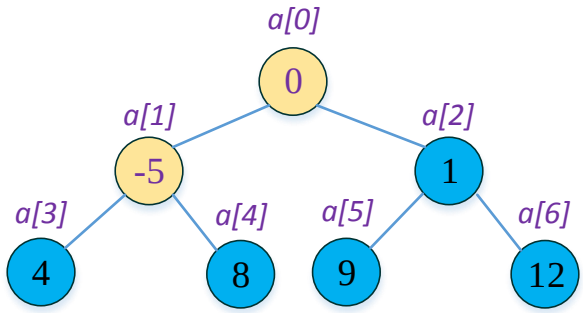


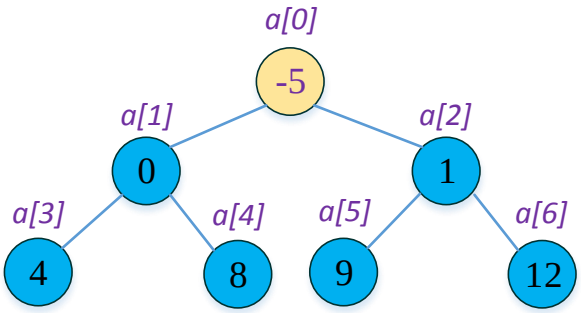


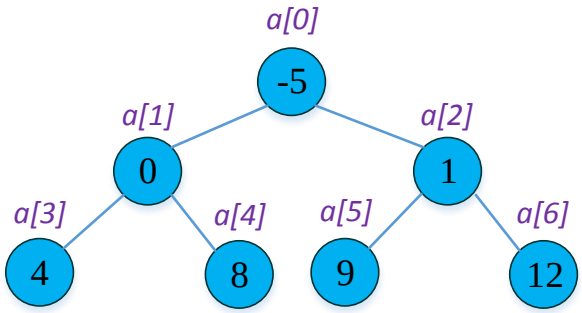












Dãy kết quả sau khi sắp xếp:

-5	0	1	4	8	9	12

3.8 Shell Sort

Ý tưởng của giải thuật sắp xếp Shell Sort:

- *Phân hoạch dãy thành các dãy con.*
- *Sắp xếp các dãy con theo phương pháp chèn trực tiếp.*
- *Dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy.*

Cách thực hiện giải thuật:

- *Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau h vị trí;*
- *Dãy ban đầu :* $a_1, a_2, ..., a_n$ *được xem như sự xen kẽ của các dãy con sau :*
  - *Dãy con thứ nhất:* $a_1, a_{h+1}, a_{2h+1}, ...$
  - *Dãy con thứ hai:* $a_2, a_{h+2}, a_{2h+2} ...$
  - *....*
  - *Dãy con thứ h:* $a_h, a_{2h}, a_{3h}, ...$
- *Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối ;*
- *Giảm khoảng cách h để tạo thành các dãy con mới ;*
- *Dừng khi h=1*

Các bước thực hiện giải thuật:

- ***Bước 1****: Chọn k khoảng cách h[1], h[2], ..., h[k];*

  i = 1;

- ***Bước 2****: Phân chia dãy ban đầu thành các dãy con cách nhau h[i] khoảng cách. Sắp xếp từng dãy con bằng phương pháp chèn trực tiếp;*
- ***Bước 3****: i = i+1;*

  Nếu i > k : Dừng

  Ngược lại : Lặp lại Bước 2.

Hàm minh họa giải thuật Shell Sort:

```
void ShellSort(int a[], int n, int h[], int k)
{
        int step, i, j, x, len;
        for (step = 0 ; step <k; step++)
        {
                len = h[step];
                for (i = len; i<n; i++)
                {
                        x = a[i];
                        j = i-len;
                        while ((x<a[j])&&(j>=0))
                        {
                                a[j+len]  = a[j];
                                j = j - len;
                        }
                        a[j+len] = x;
                }
        }
}
```

Ví dụ 12: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật sắp xếp Shell Sort.

1	4	8	9	0	12	-5

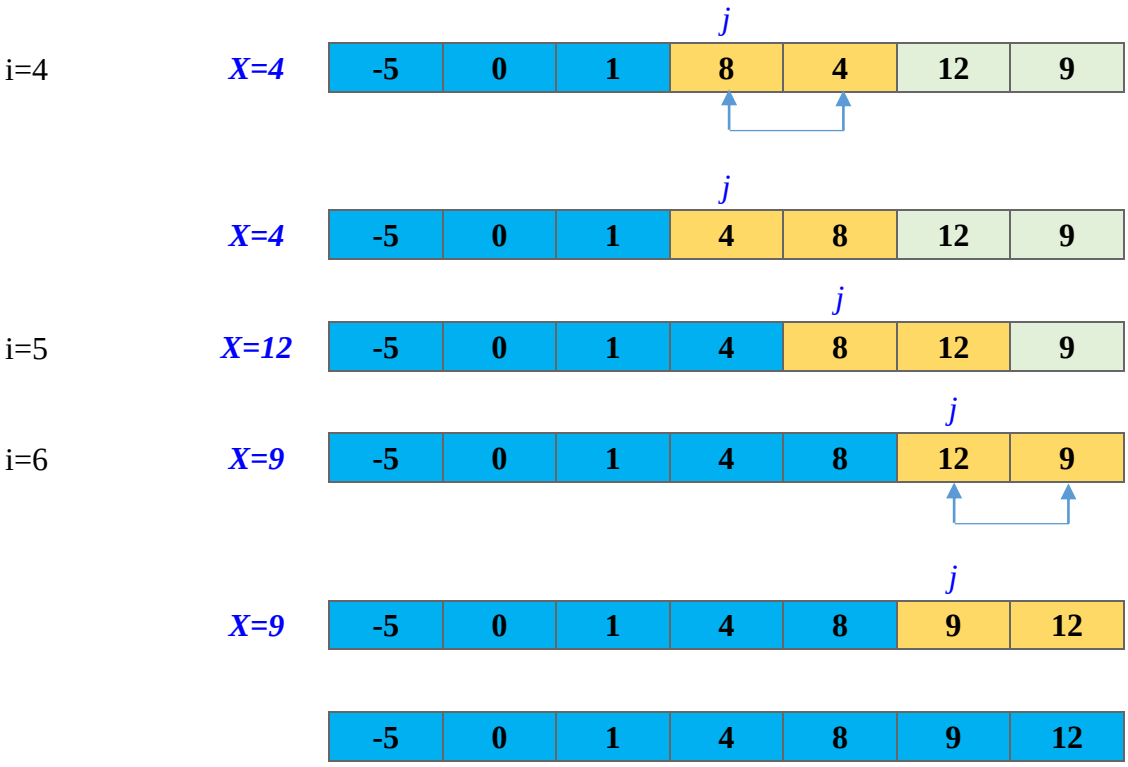
Ta chọn *h[] = {3, 1}*, *k=2;*

***step = 0**, **len = h[0] = 3**:*

		j						
i=3	***X=9***	1	4	8	9	0	12	-5
			j					
i=4	***X=0***	1	4	8	9	0	12	-5
			j					
	X=0	1	0	8	9	4	12	-5
				j				
i=5	***X=12***	1	0	8	9	4	12	-5
					j			
i=6	***X=-5***	1	0	8	9	4	12	-5
		j						
	X=-5	1	0	8	9	4	12	9
		j						
	X=-5	1	0	8	1	4	12	9
		j						
	X=-5	-5	0	8	1	4	12	9

***step = 1**, **len = h[1] =1**:*

		j						
i=1	***X=0***	-5	0	8	1	4	12	9
			j					
i=2	***X=8***	-5	0	8	1	4	12	9
				j				
i=3	***X=1***	-5	0	8	1	4	12	9
				j				
	X=1	-5	0	1	8	4	12	9



3.9 Merge Sort

Ý tưởng của giải thuật sắp xếp trộn (Merge Sort):

Ý tưởng của giải thuật là trộn 2 dãy đã được sắp xếp thành 1 dãy mới cũng được sắp xếp.

Đầu tiên ta coi mỗi phần tử của dãy là 1 danh sách con gồm 1 phần tử đã được sắp. Tiếp theo tiến hành trộn từng cặp 2 dãy con 1 phần tử kề nhau để tạo thành các dãy con 2 phần tử được sắp. Các dãy con 2 phần tử được sắp này lại được trộn với nhau tạo thành dãy con 4 phần tử được sắp. Quá trình tiếp tục đến khi chỉ còn 1 dãy con duy nhất được sắp, đó chính là dãy ban đầu.

Các bước thực hiện:

- ***Bước 1***: *Chuẩn bị*
  k = 1; // k là chiều dài của dãy con trong bước hiện hành
- ***Bước 2***:
  Tách dãy $a_0, a_1, ..., a_{n-1}$ thành 2 dãy b, c theo nguyên tắc luân phiên từng nhóm k phần tử:
  b = $a_0, a_k, a_{2k}, a_{3k}, ...$
  c = $a_{k+1}, a_{2k+1}, a_{3k+1}, ...$
- ***Bước 3***:
  Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.
- ***Bước 4***:
  *k = k*2;*
  Nếu k < n thì trở lại bước 2. Ngược lại: Dừng.

Ví dụ 13: Minh họa sắp xếp dãy số: 1, 4, 8, 9, 0, 12, -5 tăng dần bằng giải thuật sắp xếp Merge Sort.

Phân phối luôn phiên thành 2 mảng b,c 1 phần tử rồi trộn thành dãy 2 phần tử sắp xếp tăng dần:

1	4	8	9	0	12	-5

Phân phối luôn phiên thành 2 mảng b,c 2 phần tử rồi trộn thành dãy 4 phần tử sắp xếp tăng dần:

1	4	8	9	-5	0	12

Trộn thành dãy sắp xếp tăng dần:

-5	0	1	4	8	9	12

Hàm phân phối đều luân phiên các dãy con độ dài k từ mảng a vào hai mảng con b và c:

```
void Distribute(int a[], int N, int &nb, int &nc, int k)
{
	int  i, pa, pb, pc;
	pa = pb = pc = 0;
	while (pa < N)
	{
		for (i=0; pa<N && i<k; i++, pa++, pb++)
			b[pb] = a[pa];
		for (i=0; pa<N && i<k; i++, pa++, pc++)
			c[pc] = a[pa];
	}
	nb = pb;
	nc = pc;
}
```

Hàm trộn mảng b và c vào mảng a:

```
void Merge(int a[],int nb, int nc, int k)
{
	int p, pb, pc, ib, ic, kb, kc;
	p=pb=pc=ib=ic=0;
	while(nb>0&&nc>0)
	{
		kb = k<nb?k:nb; //min(k,nb)
		kc = k<nb?k:nc; //min(k,nc)
		if(b[pb+ib]<=c[pc+ic])
		{
			a[p++]=b[pb+ib];
			ib++;
			if(ib==kb)
			{
				for(;ic<kc;ic++)
					a[p++]=c[pc+ic];
				pb+=kb; pc+=kc;
				ib = ic=0;
				nb-=kb; nc-=kc;
			}
			else
			{
				a[p++]=c[pc+ic];
				ic++;
				if(ic==kc)
				{
					for(;ib<kb;ib++)
						a[p++]=b[pb+ib];
					pb+=kb; pc+=kc;
					ib = ic=0;
					nb-=kb; nc-=kc;
				}
			}
		}
	}
}
```

Hàm sắp xếp trộn Merge Sort:

```
void MergeSort(int a[], int N)
{
	int k;
	for (k = 1; k < N; k *= 2)
	{
		Distribute(a, N, nb, nc, k);
		Merge(a, nb, nc, k);
	}
}
```

3.10 Radix Sort

Ý tưởng của thuật toán sắp xếp theo cơ số Radix Sort: Radix Sort sắp xếp dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó Radix Sort còn có tên là Postman's Sort.

- *Trước tiên, ta có thể giả sử mỗi phần tử a_i trong dãy a_1, a_2, ..., a_n là một số nguyên có tối đa m chữ số.*
- *Ta phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, ... tương tự việc phân loại thư theo tỉnh thành, quận huyện, phường xã, ...*

Các bước tiến hành:

- ***Bước 1*** *:// k cho biết chữ số dùng để phân loại hiện hành*
  k = 0; // k = 0: hàng đơn vị; k = 1: hàng chục; ...
- ***Bước 2*** *: //Tạo các lô chứa các loại phần tử khác nhau*
  Khởi tạo 10 lô B_0, B_1, ..., B_9 rỗng;
- ***Bước 3*** *:*
  For i = 1 .. n do
  Đặt a_i vào lô B_t với t: chữ số thứ k của ai;
- ***Bước 4*** *:*
  Nối B_0, B_1, ..., B_9 lại (theo đúng trình tự) thành a.
- ***Bước 5*** *:*
  k = k+1;
  Nếu k < m thì trở lại bước 2.
  Ngược lại: Dừng

Ví dụ 14: Sắp xếp dãy sau tăng dần theo giải thuật sắp xếp cơ số Radix Sort:

7013, 8425, 1239, 428, 1424, 7009, 4518, 3252, 9170, 999, 1725, 701.

12	0701										
11	1725										
10	0999										
9	9170										
8	3252										
7	4518										
6	7009										
5	1424										
4	0428										
3	1239										
2	8425										
1	7013										
CS	A	0	1	2	3	4	5	6	7	8	9

K=0: Phân loại hàng đơn vị

12											
11											
10											
9											
8											
7											
6											
5											
4											
3											0999
2							1725			4518	7009
1		9170	0701	3252	7013	1424	8425			0428	1239
CS	A	0	1	2	3	4	5	6	7	8	9

12	0999										
11	7009										
10	1239										
9	4518										
8	0428										
7	1725										
6	8425										
5	1424										
4	7013										
3	3252										
2	0701										
1	9170										
CS	A	0	1	2	3	4	5	6	7	8	9

K=1: Phân loại hàng chục

12											
11											
10											
9											
8											
7											
6											
5											
4				0428							
3				1725							
2		7009	4518	8425							
1		0701	7013	1424	1239		3252		9170		0999
CS	A	0	1	2	3	4	5	6	7	8	9

12	0999										
11	9170										
10	3252										
9	1239										
8	0428										
7	1725										
6	8425										
5	1424										
4	4518										
3	7013										
2	7009										
1	0701										
CS	A	0	1	2	3	4	5	6	7	8	9

K=2: Phân loại hàng trăm

12											
11											
10											
9											
8											
7											
6											
5											
4											
3						0428					
2		7013		3252		8425			1725		
1		7009	9170	1239		1424	4518		0701		0999
CS	A	0	1	2	3	4	5	6	7	8	9

12	0999										
11	1725										
10	0701										
9	4518										
8	0428										
7	8425										
6	1424										
5	3252										
4	1239										
3	9170										
2	7013										
1	7009										
CS	A	0	1	2	3	4	5	6	7	8	9

K=3: Phân loại hàng nghìn

12											
11											
10											
9											
8											
7											
6											
5											
4											
3		0999	1725								
2		0701	1424						7013		
1		0428	1239		3252	4518			7009	8425	9170
CS	A	0	1	2	3	4	5	6	7	8	9

12	**9170**										
11	**8425**										
10	**7013**										
9	**7009**										
8	**4518**										
7	**3252**										
6	**1725**										
5	**1424**										
4	**1239**										
3	**0999**										
2	**0701**										
1	**0428**										
CS	A	0	1	2	3	4	5	6	7	8	9

Kết quả cuối cùng dãy đã được sắp xếp tăng dần:

428, 701, 999, 1239, 1424, 1725, 3252, 4518, 7009, 7013, 8425, 9170

3.11 Đánh giá độ phức tạp của các giải thuật sắp xếp

Giải thuật	Trường hợp tốt nhất	Trường hợp trung bình	Trường hợp xấu nhất
Đổi chỗ trực tiếp	$O(n^2)$	$O(n^2)$	$O(n^2)$
Chọn trực tiếp	$O(n^2)$	$O(n^2)$	$O(n^2)$
Chèn trực tiếp	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)
Radix Sort	O(nlogn)	O(nlogn)	O(nlogn)

BÀI TẬP CHƯƠNG 3

Bài 1. Cài đặt tất cả các thuật toán sắp xếp đã học để sắp xếp tăng dần 1 mảng a gồm n phần tử nhập từ bàn phím.

Bài 2. Sắp xếp mảng A gồm n phần tử nhập từ bàn phím như sau: Nửa đầu của A tăng dần, nửa sau giảm dần, phần tử ở giữa mảng nếu có sẽ giữ nguyên (trường hợp này xảy ra khi n là số lẻ). Yêu cầu là sử dụng 2 thuật toán sắp xếp khác nhau cho 2 nửa mảng A.

Bài 3. Nhập dữ liệu cho mảng cấu trúc gồm n sinh viên, mỗi sinh viên gồm các thông tin: Họ tên, mã số sinh viên, điểm tổng kết.

a) Sắp xếp mảng theo mã số sinh viên tăng dần.
b) Sắp xếp mảng theo điểm tổng kết giảm dần.
c) Sắp xếp mảng theo tên sinh viên (thứ tự A, B, C…).

Chương 4. DANH SÁCH LIÊN KẾT

4.1 Giới thiệu tổng quan

4.1.1 Biến tĩnh và biến động

Biến tĩnh:

- *Là biến được khai báo tường minh, có tên gọi, tồn tại trong phạm vi khai báo.*
- *Biến tĩnh có kích thước không đổi => không tận dụng hiệu quả bộ nhớ.*

Biến động:

- *Là biến không được khai báo tường minh, không có tên gọi.*
- *Biến động có thể xin khi cần, giải phóng khi sử dụng xong.*
- *Biến động linh động về kích thước => tận dụng hiệu quả bộ nhớ.*
- *Để thao tác với biến động, ta lưu địa chỉ của nó bằng biến con trỏ => truy xuất biến động thông qua biến con trỏ.*

4.1.2 Cấu trúc tự trỏ

Thông thường khi thiết kế chương trình chúng ta chưa biết được số lượng dữ liệu cần dùng là bao nhiêu để khai báo số biến cho phù hợp. Đặc biệt là biến dữ liệu kiểu mảng. Số lượng các thành phần của biến mảng cần phải khai báo trước và chương trình dịch sẽ bố trí vùng nhớ cố định cho các biến này. Do buộc phải khai báo trước số lượng thành phần nên kiểu mảng thường dẫn đến hoặc là lãng phí bộ nhớ (khi chương trình không dùng hết) hoặc là không đủ để chứa dữ liệu (khi chương trình cần chứa dữ liệu với số lượng thành phần lớn hơn).

Để khắc phục tình trạng này C++ cho phép cấp phát bộ nhớ động, nghĩa là số lượng các thành phần không cần phải khai báo trước. Bằng toán tử new chúng ta có thể xin cấp phát vùng nhớ theo nhu cầu mỗi khi chạy chương trình. Tuy nhiên, cách làm này dẫn đến việc dữ liệu của một danh sách sẽ không còn nằm liên tục trong bộ nhớ như đối với biến mảng. Mỗi lần xin cấp phát bởi toán tử new, chương trình sẽ tìm một vùng nhớ đang rỗi bất kỳ để cấp phát cho biến và như vậy dữ liệu sẽ nằm rải rác trong bộ nhớ. Từ đó, để dễ dàng quản lý trật tự của một danh sách các dữ liệu, mỗi thành phần của danh sách cần phải chứa địa chỉ của thành phần tiếp theo hoặc trước nó (điều này là không cần thiết đối với biến mảng vì các thành phần của chúng sắp xếp liên tục, kề nhau). Từ đó, mỗi thành phần của danh sách sẽ là một cấu trúc, ngoài các thành phần chứa thông tin của bản thân, chúng còn phải có thêm một hoặc nhiều con trỏ để trỏ đến các thành phần tiếp theo hay đứng trước. Các cấu trúc như vậy được gọi là cấu trúc tự trỏ vì các thành phần con trỏ trong cấu trúc này sẽ trỏ đến các vùng dữ liệu có kiểu chính là kiểu của chúng.

Ví dụ 15:

```
struct sinhvien{
        char Hoten[33];
        int MSSV;
        sinhvien *tiep;
}
```

4.1.3 Cấu trúc dữ liệu dạng danh sách

Danh sách là một kiểu dữ liệu dạng tuyến tính, nó bao gồm tập hợp các phần tử có cùng kiểu dữ liệu. Mỗi phần tử trong danh sách có nhiều nhất 1 phần tử đứng trước và nhiều nhất 1 phần tử đứng sau.

Hai hình thức tổ chức cơ bản của danh sách là mảng (liên kết ngầm) và danh sách liên kết (liên kết tường minh).

Mảng:

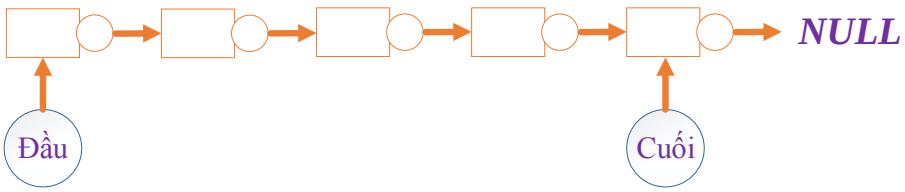
- *Mối liên hệ giữa các phần tử được thể hiện ngầm:*
  - *A[i] : phần tử thứ i+1 trong danh sách.*
  - *A[i] và A[i+1] là các phần tử kế cận trong danh sách .*
- *Phải lưu trữ liên tiếp các phần tử trong bộ nhớ*
- *Ưu điểm : Truy xuất trực tiếp, nhanh chóng*
- *Nhược điểm:*
  - *Sử dụng bộ nhớ kém hiệu quả*
  - *Kích thước cố định*
  - *Các thao tác thêm vào , loại bỏ không hiệu quả*

Danh sách liên kết:

- *Cấu trúc dữ liệu cho mỗi phần tử trong danh sách liên kết là 1 cấu trúc tự trỏ bao gồm:*
  - *Thông tin bản thân phần tử;*
  - *Địa chỉ của phần tử kế trong danh sách.*
- *Mỗi phần tử của danh sách liên kết là một biến động;*
- *Ưu điểm:*
  - *Sử dụng hiệu quả bộ nhớ*
  - *Linh động về số lượng phần tử*
- *Các dạng của danh sách liên kết bao gồm:*
  - *Danh sách liên kết đơn*
  - *Danh sách liên kết kép*
  - *Danh sách liên kết vòng*

4.2 Danh sách liên kết đơn

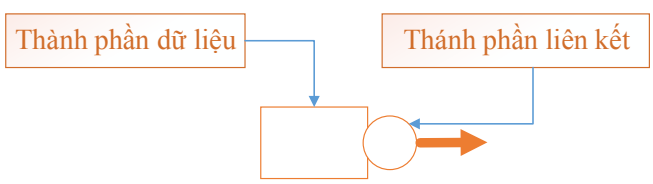
4.2.1 Cấu trúc dữ liệu của danh sách liên kết đơn



Hình 9. Minh họa danh sách liên kết đơn.

Danh sách liên kết đơn là DSLK mà mỗi phần tử trong danh sách sẽ liên kết với phần tử liền sau nó. Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần:

- ***Thành phần dữ liệu**: Lưu trữ thông tin về bản thân phần tử;*
- ***Thành phần liên kết**: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.*



Hình 10. Cấu trúc dữ liệu của mỗi phần tử (nút) của DSLK đơn

Cấu trúc dữ liệu của danh sách liên kết đơn bao gồm 2 con trỏ lần lượt lưu địa chỉ của phần tử (hoặc nút) đầu tiên và phần tử cuối cùng trong danh sách.

Cấu trúc dữ liệu của một nút trong dách sách:

```
struct SinhVien{
        char Hoten[33];
        int MSSV;
        SinhVien *tiep;
};
```

Cấu trúc dữ liệu của danh sách liên kết đơn cho danh sách sinh viên ở trên:

```
struct DanhSach{
        SinhVien *dau;
        SinhVien *cuoi;
};
```

4.2.2 Các giải thuật trên danh sách liên kết đơn

Khởi tạo danh sách liên kết đơn rỗng: Địa chỉ của nút đầu tiên và nút cuối cùng không có.

```
void  CreateList(DanhSach  &DSSV)
{
        DSSV.dau = DSSV.cuoi = NULL;
}
```

Tạo một phần tử mới: Để tạo phần tử mới ta thực hiện theo các bước sau đây:

- *Dùng toán tử **new** xin cấp phát một vùng nhớ đủ chứa một phần tử của danh sách.*
- *Nhập thông tin cần lưu trữ vào phần tử mới. Con trỏ tiếp được đặt bằng NULL.*
- *Gắn phần tử vừa tạo được vào danh sách.*

Hàm tạo 1 nút trong danh sách liên kết đơn.

```
SinhVien* CreateNode()
{
        SinhVien *x= new SinhVien[1];
        cout<< "Nhap vao ma so sinh vien: ";
        cin>>x->MSSV;
        cout<< "Nhap vao ho va ten sinh vien: ";
        cin.ignore(1);  cin.getline(x->Hoten, 33);
        x->tiep = NULL;
        return x;
}
```

Để gắn phần tử mới tạo vào danh sách liên kết ta có 3 cách thêm như sau:

- *Thêm phần tử vào đầu DSLK*
- *Thêm phần tử vào cuối DSLK*
- *Thêm phần tử vào sau một phần tử của DSLK*

Thêm phần tử vào đầu danh sách: Các bước tiến hành như sau:

- *Tạo và cấp phát bộ nhớ cho 1 nút mới.*
- *Nếu DSLK rỗng thì gán phần tử đầu và cuối của DSLK bằng chính nút mới tạo.*
- *Nếu DSLK khác rỗng thì cho con trỏ tiếp của nút mới trỏ đến phần tử đầu tiên của DSLK sau đó cho con trỏ đầu của DSLK trỏ vào nút mới tạo.*

```
void chendau(DanhSach &DSSV, SinhVien *x)
{
        if(DSSV.dau==NULL)
        {
                DSSV.dau=x;
                DSSV.cuoi=x;
        }
        else
        {
                x->tiep=DSSV.dau;
                DSSV.dau=x;
        }
}
```

Thêm phần tử vào cuối danh sách liên kết: Các bước tiến hành như sau:

- *Tạo và cấp phát bộ nhớ cho 1 nút mới.*
- *Nếu DSLK rỗng thì gán phần tử đầu và cuối của DSLK bằng nút mới tạo.*
- *Nếu DSLK khác rỗng:*
  - *Cho con trỏ tiếp của phần tử cuối của DSLK trỏ đến nút mới tạo.*
  - *Gán phần tử cuối của DSLK bằng nút mới tạo hoặc cho con trỏ tiếp của nút mới bằng NULL.*

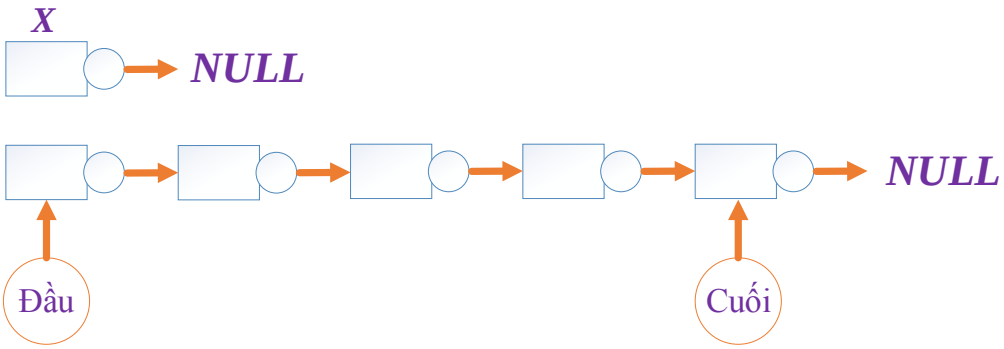
```
void chencuoi(DanhSach &DSSV, SinhVien *x)
{
        if(DSSV.dau==NULL)
        {
                DSSV.dau=x;
                DSSV.cuoi=x;
        }
        else
        {
                DSSV.cuoi->tiep=x;
                DSSV.cuoi=x;
        }
}
```

Thêm phần tử data vào sau nút q: Các bước thực hiện như sau:

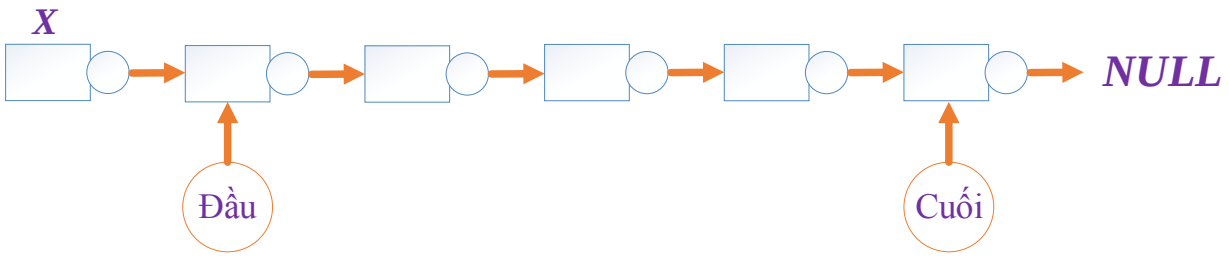
- *Tạo và cấp phát bộ nhớ cho 1 nút mới cần thêm.*
- *Nếu tìm thấy nút q:*
  - *Cho con trỏ tiếp của nút mới trỏ đến nút kế của q.*
  - *Cho con trỏ tiếp của q trỏ vào nút mới.*
  - *Trường hợp q là nút cuối thì gán phần tử cuối của DSLK bằng nút mới thêm*
- *Nếu không tìm thấy nút q.*

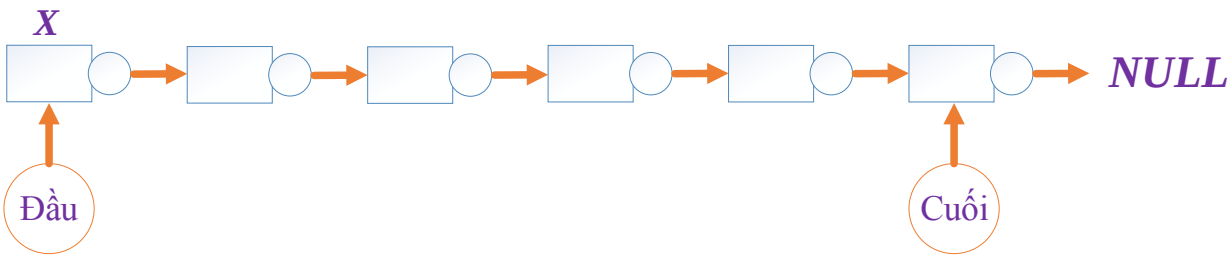
```
void InsertAfterQ(DanhSach &DSSV, SinhVien *Data, SinhVien *q)
{
        if(q!=NULL)
        {
                Data->tiep=q->tiep;
                q->tiep=Data;
                if(DSSV.cuoi==q)
                DSSV.cuoi=Data;
        }
        else
                chendau(DSSV, Data); //Them Data vao dau danh sach
}
```

Ví dụ 16: Minh họa thêm phần tử vào danh sách liên kết:

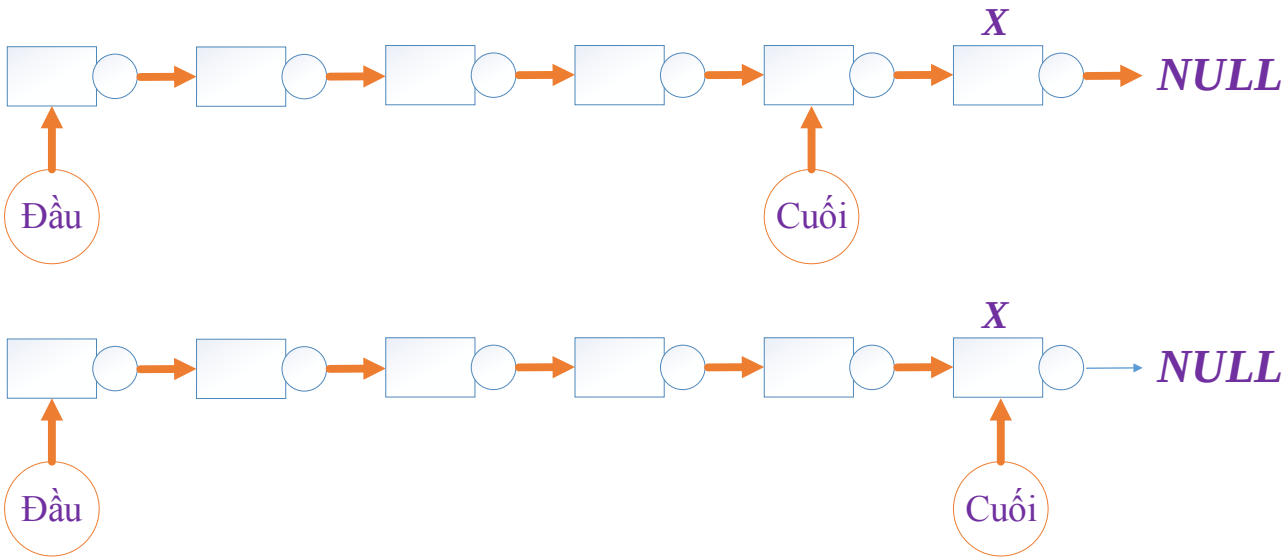


Thêm X vào đầu DSLK:

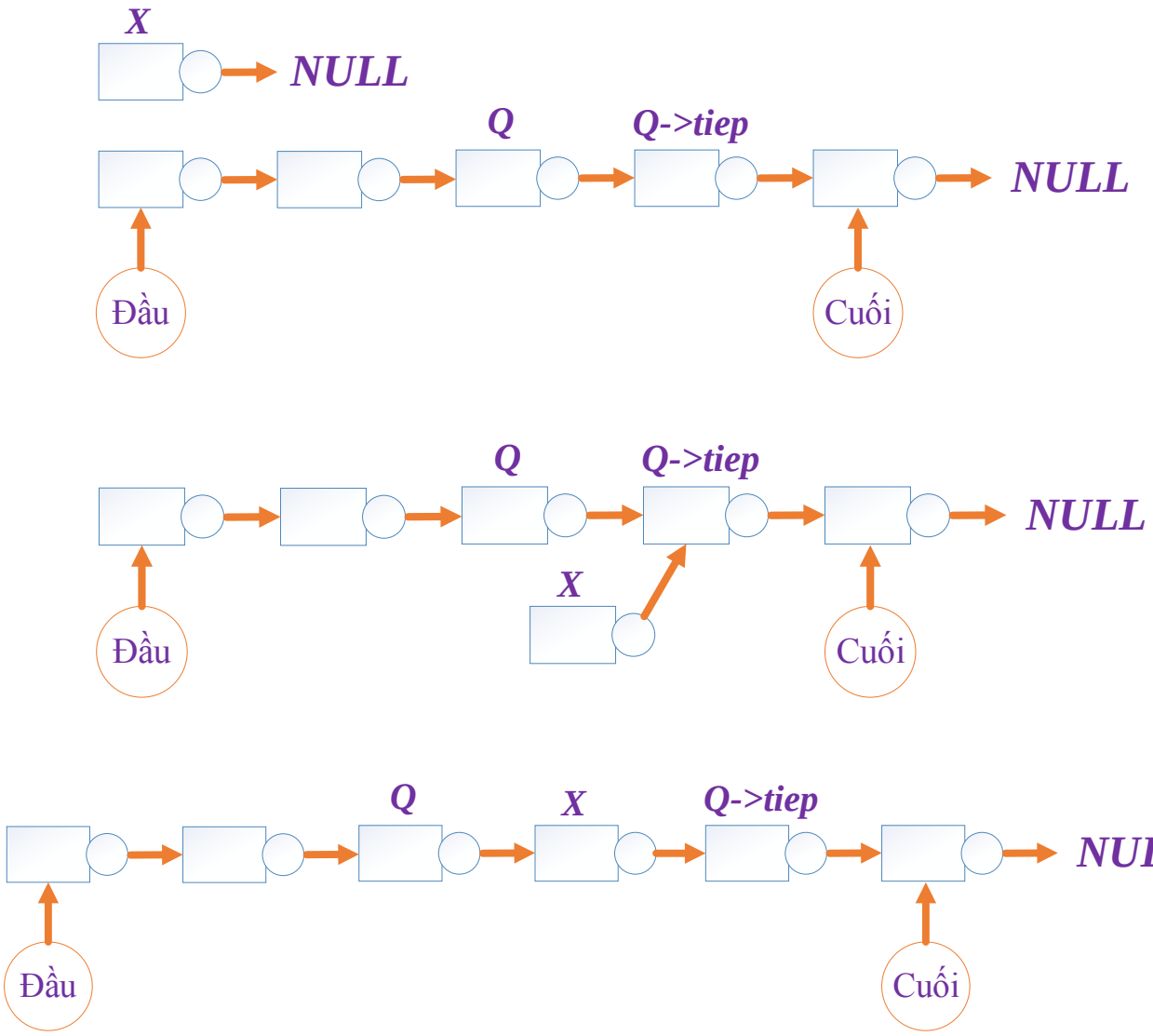




Thêm X vào cuối DSLK:

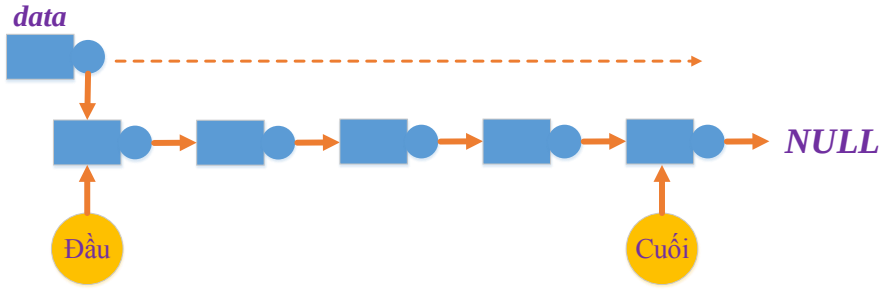


Thêm X vào sau phần tử q:



Duyệt danh sách liên kết đơn:

- *Ta sử dụng 1 biến con trỏ tạm dạng nút của DSLK rồi trỏ vào phần tử đầu của DSLK. Từ vị trí này, theo liên kết giữa các nút ta sẽ thực hiện việc duyệt qua từng phần tử trong DSLK.*
- *Trong quá trình duyệt, tại mỗi nút ta có thể thực hiện các thao tác như:*
  - *Lấy thông tin phần tử*
  - *Sửa thông tin phần tử*
  - *So sánh phần tử*
  - *Xóa phần tử…*



Hình 11. Duyệt danh sách liên kết đơn

Hàm minh họa duyệt qua DSLK:

```
void output(DanhSach &DSSV)//Hàm in ra DSLK
{
        SinhVien *data;
        data=DSSV.dau;
        cout<< "Danh Sach cac phan tu cua DSLK:\n";
        cout<< "Ma So\tHo Va Ten";
        while(data!=NULL)
        {
                cout<<endl<<data->MSSV<< "\t"<<data->Hoten;
                data=data->tiep;
        }
}
```

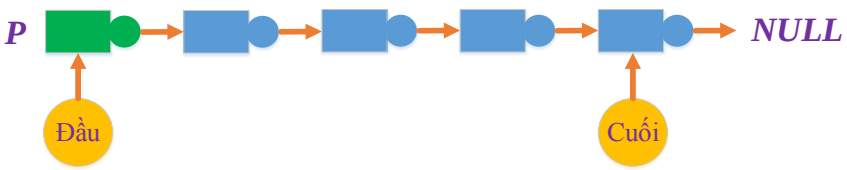
Xóa phần tử khỏi danh sách liên kết đơn:

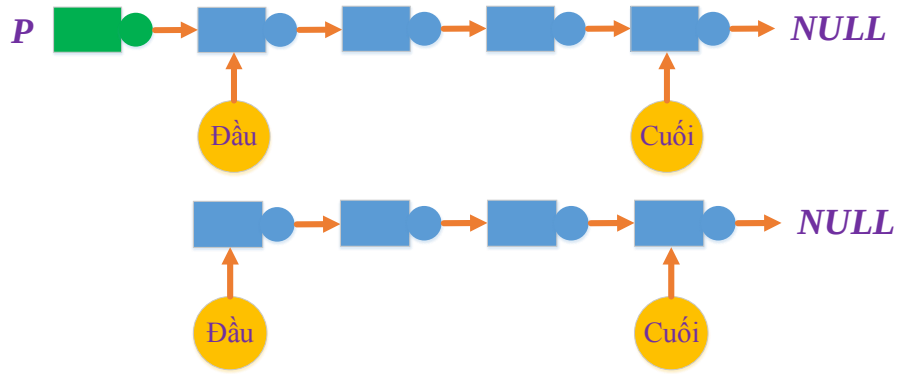
Để xóa một nút p ra khỏi DSLK đơn ta làm như sau:

- *Tìm nút q liền trước nút p.*
- *Nếu tìm thấy q, cho q liên kết với nút liền sau của p rồi giải phóng bộ nhớ đã cấp cho p.*
- *Nếu không tìm thấy q, tức p là nút đầu tiên, khi đó ta chỉ việc thay đổi nút đầu tiên của DSLK thành nút đứng liền sau p rồi thu hồi bộ nhớ của p.*

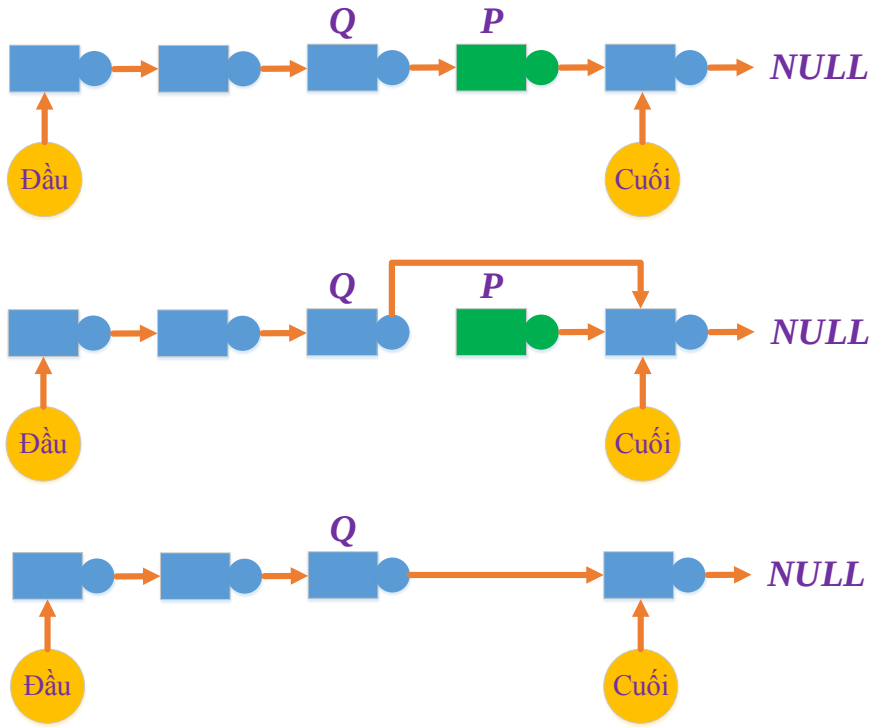
Ví dụ 17: Minh họa xóa nút P khỏi danh sách liên kết đơn:

- Trường hợp P ở đầu danh sách:





- Trường hợp P đứng sau 1 nút Q:



Hàm xóa 1 phần tử khỏi danh sách liên kết đơn:

```
int RemoveX(DanhSach &DSSV, SinhVien *x)
{       SinhVien *p; p=DSSV.dau;
        if(DSSV.dau==x){
                DSSV.dau = x->tiep;
                delete x; return 1; //Xoa thanh cong
        }
        while((p!=NULL)&&(p ->tiep!=x)) //Tim p lien truoc x
        p=p->tiep;
        if(p==NULL) return 0; //khong tim thay phan tu co khoa bang x
        else {
                p->tiep=x->tiep;
                delete x; return 1; //Xoa thanh cong
        }
}
```

Ví dụ 18: Chương trình minh họa về danh sách liên kết đơn sử dụng các hàm ở trên:

```
#include<iostream>
using namespace std;
//CTDL cua Node
struct SinhVien{
        char Hoten[33];
        int MSSV;
        SinhVien *tiep;
};
//CTDL cua danh sach lien ket
struct DanhSach{
        SinhVien *dau;
        SinhVien *cuoi;
};
//Ham tao danh sach lien ket don rong
void  CreateList(DanhSach  &DSSV)
{
         DSSV.dau = DSSV.cuoi = NULL;
}
//Ham tao 1 Node moi
SinhVien* CreateNode()
{
        SinhVien *x= new SinhVien[1];
        cout<< "Nhap vao ma so sinh vien: ";
        cin>>x->MSSV;
        cout<< "Nhap vao ho va ten sinh vien: ";
        cin.ignore(1);
        cin.getline(x->Hoten,33);
        x->tiep = NULL;
        return x;
}
//Ham them phan tu vao dau danh sach
void chendau(DanhSach &DSSV, SinhVien *x)
{
        if(DSSV.dau==NULL)
        {
                DSSV.dau=x;
                DSSV.cuoi=x;
        }
```

```
        else
        {
                x->tiep=DSSV.dau;
                DSSV.dau=x;
        }
}
//Ham them phan tu vao cuoi danh sach
void chencuoi(DanhSach &DSSV, SinhVien *x)
{
        if(DSSV.dau==NULL)
        {
                DSSV.dau=x;
                DSSV.cuoi=x;
        }
        else
        {
                DSSV.cuoi->tiep=x;
                DSSV.cuoi=x;
        }
}
//Them phan tu data vao sau nut q
void InsertAfterQ(DanhSach &DSSV, SinhVien *Data, SinhVien *q)
{
         if(q!=NULL)
         {
                Data->tiep=q->tiep;
                q->tiep=Data;
                if(DSSV.cuoi==q)
                DSSV.cuoi=Data;
         }
         else
                chendau(DSSV, Data);
}
//Ham duyet qua danh sach
void Output(DanhSach &DSSV)
{
        SinhVien *data;
        data=DSSV.dau;
        cout<< "Danh Sach cac phan tu cua DSLK:\n";
```

```
        cout<< "Ma So\tHo Va Ten";
        while(data!=NULL)
        {
            cout<<endl<<data->MSSV<< "\t"<<data->Hoten;
            data=data->tiep;
        }
}
//Ham xoa phan tu khoi DSLK
int RemoveX(DanhSach &DSSV, SinhVien *x)
{   SinhVien *p; p=DSSV.dau;
    if(DSSV.dau==x)
    {
        DSSV.dau = x->tiep;
        delete x; return 1; //Xoa thanh cong
    }
    while((p!=NULL)&&(p ->tiep!=x)) //Tim p lien truoc x
        p=p->tiep;
    if(p==NULL) return 0; //khong tim thay phan tu co khoa bang x
    else
    {
        p->tiep=x->tiep;
        delete x; return 1; //Xoa thanh cong
    }
}
int main()
{
    DanhSach CN14B;
    CreateList(CN14B);
    int i, n;
    cout<< "Nhap so luong sinh vien: ";
    cin>>n;
    SinhVien *t;
    for(i=0; i<n; i++)
    {
        cout<< "Nhap du lieu cho sinh vien thu "<<i+1<< ":\n";
        t=CreateNode();
        chendau(CN14B, t);
    }
```

```
        cout<< "\n==========================================\n";
        Output(CN14B);
        cout<<endl<< "Chen them 1 sinh vien vao cuoi danh sach:\n";
        t=CreateNode(); chencuoi(CN14B,t);
        cout<< "\n==========================================\n";
        Output(CN14B);
        cout<<endl<< "Chen them 1 sinh vien vao sau phan tu thu nhat:\n";
        t=CreateNode();
        InsertAfterQ(CN14B, t, CN14B.dau);
        cout<< "\n==========================================\n";
        Output(CN14B);
        cout<< "\n==========================================\n";
        cout<< "Xoa sinh vien dau tien khoi danh sach:\n";
        RemoveX(CN14B, CN14B.dau); Output(CN14B);
    }
```

Kết quả chạy chương trình:

```
Nhap so luong sinh vien: 3
Nhap du lieu cho sinh vien thu 1:
Nhap vao ma so sinh vien: 123
Nhap vao ho va ten sinh vien: Cao Tien Dung
Nhap du lieu cho sinh vien thu 2:
Nhap vao ma so sinh vien: 453
Nhap vao ho va ten sinh vien: Nhat Tam Tinh
Nhap du lieu cho sinh vien thu 3:
Nhap vao ma so sinh vien: 767
Nhap vao ho va ten sinh vien: Tran Can Nam

==========================================
Danh Sach cac phan tu cua DSLK:
Ma So   Ho Va Ten
767     Tran Can Nam
453     Nhat Tam Tinh
123     Cao Tien Dung
Chen them 1 sinh vien vao cuoi danh sach:
Nhap vao ma so sinh vien: 126
Nhap vao ho va ten sinh vien: Bo Kinh Van

==========================================
Danh Sach cac phan tu cua DSLK:
Ma So   Ho Va Ten
767     Tran Can Nam
453     Nhat Tam Tinh
123     Cao Tien Dung
126     Bo Kinh Van
Chen them 1 sinh vien vao sau phan tu thu nhat:
Nhap vao ma so sinh vien: 876
Nhap vao ho va ten sinh vien: Tran Duc Khai

==========================================
Danh Sach cac phan tu cua DSLK:
Ma So   Ho Va Ten
767     Tran Can Nam
876     Tran Duc Khai
453     Nhat Tam Tinh
123     Cao Tien Dung
126     Bo Kinh Van
==========================================
Xoa sinh vien dau tien khoi danh sach:
Danh Sach cac phan tu cua DSLK:
Ma So   Ho Va Ten
876     Tran Duc Khai
453     Nhat Tam Tinh
123     Cao Tien Dung
126     Bo Kinh Van
```

Sắp xếp danh sách liên kết đơn:

Có 2 cách tiếp cận:

- ***Cách 1**: Thay đổi thành phần dữ liệu của các nút.*
  - *Ưu điểm: Cài đặt đơn giản, tương tự như sắp xếp mảng.*
  - *Nhược điểm:*
    - *Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử -> chỉ phù hợp với những DSLK có kích thước dữ liệu nhỏ.*
    - *Khi kích thước dữ liệu lớn thì chi phí cho việc hoán vị thành phần dữ liệu cũng lớn.*
    - *Làm cho thao tác sắp xếp chậm.*
- ***Cách 2**: Thay đổi thành phần con trỏ tiếp theo trong nút (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn).*
  - *Ưu điểm:*
    - *Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.*
    - *Thao tác sắp xếp nhanh.*
  - *Nhược điểm: Cài đặt phức tạp.*

Các thuật toán sắp xếp DSLK bằng cách thay đổi thành phần liên kết có hiệu quả cao như: Quick Sort, Merge Sort, radix Sort,…

4.3 Ngăn xếp - Stack

4.3.1 Tổng quan

Stack (ngăn xếp): Là một danh sách mà ta giới hạn việc thêm vào hoặc loại bỏ một phần tử chỉ thực hiện tại một đầu của danh sách, đầu này gọi là đỉnh (TOP) của ngăn xếp. Chính vì nguyên tắc này mà ngăn xếp còn được gọi là kiểu dữ liệu có nguyên tắc LIFO (Last In First Out – Vào sau ra trước).

Ví dụ lưu trữ dữ liệu kiểu LIFO chẳng hạn như: Chồng sách trên mặt bàn, chồng đĩa trong hộp…

Các thao tác trên ngăn xếp:

- *Thêm đối tượng vào stack.*
- *Lấy đối tượng từ stack.*
- *Kiểm tra stack rỗng hay không?*
- *Lấy giá trị phần tử đỉnh của stack mà không hủy nó.*

4.3.2 Cài đặt ngăn xếp bằng mảng một chiều

- *Để cài đặt ngăn xếp bằng mảng, ta sử dụng mảng một chiều S để biểu biễn ngăn xếp.*
- *Thiết lập phần tử đầu tiên của mảng S[0] làm đáy ngăn xếp.Các phần tử tiếp theo được đưa vào ngăn xếp sẽ lần lượt được lưu tại các vị trí S[1], S[2],…*
- *Nếu hiện tại ngăn xếp có n phần tử thì S]n-1] sẽ là đỉnh của ngăn xếp.*
- *Để lưu trữ đỉnh hiên tại của ngăn xếp, ta sử dụng một biến top lưu chỉ số của đỉnh (ở đây top=n-1).*
- *Khi ngăn xếp chưa có phần tử nào ta quy ước top = -1.*

Chỉ số	***0***	***1***	***2***	***3***	***4***	***5***	**…**	***n-1***	***n***
Mảng	**7**	**9**	**6**	**4**	**3**	**12**	**…**	**20**	
								top	

Hình 12. Minh họa ngăn xếp bằng mảng

Cấu trúc của Stack:

```
const int max=1000;
typedef struct tagStack{
        int S[max];
        int top;
}Stack;
```

Khởi tạo Stack:

```
void CreateStack(Stack &st)
{
        st.top = -1;
}
```

Kiểm tra ngăn xếp rỗng:

```
int IsEmpty(Stack st)
{
        if(st.top==-1)
                return 1;
        else
                return 0;
}
```

Kiểm tra ngăn xếp đầy:

```
int IsFull(Stack st)
{
        if(st.top>=max-1)
                return 1;
        else
        return 0;
}
```

Thêm một phần tử x vào Stack:

```
int Push(Stack &st, int x)
{
        if(IsFull(st)==0) //Nếu Stack chưa Full
        {
                st.top++; //Tăng đỉnh lên 1
                st.S[st.top]=x;//Gán giá trị x cho đỉnh
                return 1; //Thêm thành công
        }
        else
                return 0; //Thêm thất bại
}
```

Lấy một phần tử ra khỏi Stack:

```
int Pop(Stack &st, int &x)
{
	if(IsEmpty(st)==0) //Nếu Stack khác rỗng
	{
		x=st.S[st.top]; //Lấy giá trị đỉnh cho vào x
		st.top--; // Đỉnh giảm đi 1
		return 1; //Lấy ra thành công
	}
	else
		return 0; //Lấy ra thất bại
}
```

Ví dụ 19: Ví dụ minh họa về cách sử dụng các hàm trên.

```
#include<stdio.h>
const int max=1000;
typedef struct tagStack
{
	int S[max], top;
}Stack;
void CreateStack(Stack &st)
{
	st.top=-1;
}
int IsEmpty(Stack st)
{
	if(st.top==-1) return 1;
	else return 0;
}
int IsFull(Stack st)
{
	if(st.top>=max) return 1;
	else return 0;
}
int Push(Stack &st, int x)
{
	if(IsFull(st)==0)
	{
		st.top++; st.S[st.top]=x; return 1;
	}
	else return 0;
}
int Pop(Stack &st, int &x)
{
	if(IsEmpty(st)==0)
	{
		x=st.S[st.top]; st.top--;
		return 1;
```

```
        }
        else return 0;
}
int main()
{
        Stack st;
        int x, result, i;
        CreateStack(st);
        for(i=2; i<=5; i++)
                Push(st, i);
        result = Pop(st, x);
        if(result==1)
        printf( "gia tri lay duoc tu Stack la: %d", x);
}
```

Ví dụ 20: Sử dụng stack để hiển thị một số nguyên dương ở dạng số hệ nhị phân.

```
#include<iostream>
using namespace std;
struct Stack{
        int a[100];
        int top;
};
int main()
{
        Stack NhiPhan;
        NhiPhan.top=-1;
        int i=0, x;
        Lap:
        cout<< "Nhap vao so nguyen duong: ";
        cin>>x;
        cout<< "So "<<x<< " o dang nhi phan la: ";
        if(x<=0)
                goto Lap;
        while(x>0)
        {
                NhiPhan.a[i]=x%2;
                x/=2;
                i++;
                NhiPhan.top++;
        }
        for(i=NhiPhan.top; i>=0;i--)
        cout<<NhiPhan.a[i];
}
```

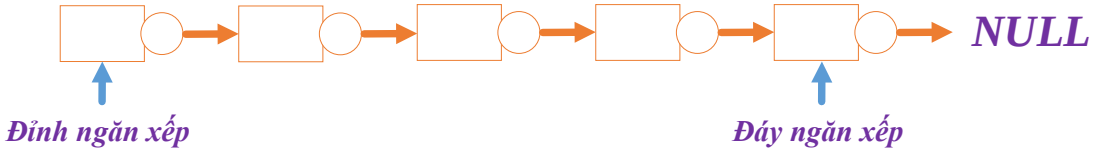
Kết quả chạy thử chương trình:

```
Nhap vao so nguyen duong: 21
So 21 o dang nhi phan la: 10101
```

4.3.3 Cài đặt ngăn xếp bằng danh sách liên kết đơn

Theo tính chất của DSLK đơn, việc bổ sung và loại bỏ 1 phần tử thực hiện đơn giản và nhanh nhất khi phần tử đó nằm đầu danh sách. Do vậy, ta sẽ chọn cách lưu trữ của ngăn xếp theo thứ tự:

- *Phần tử đầu danh sách là đỉnh của stack.*
- *Phần tử cuối danh sách là đáy của ngăn xếp.*
- *Để bổ sung 1 phần tử mới vào stack ta thêm nó vào đầu danh sách.*
- *Để lấy 1 phần tử ra khỏi ngăn xếp ta lấy giá trị nút đầu tiên và loại nó ra khỏi danh sách.*



Hình 13. Minh họa Stack bằng DSLK đơn.

Khai báo stack:

```
typedef struct node{
        int item;
        struct node *next;
} *stacknode;
 typedef struct Stack{
        stacknode top;
}stack;
```

Khởi tạo stack:

```
void CreateStack(stack &s)
{
        s.top=NULL;
}
```

Kiểm tra stack rỗng:

```
int CheckEmpty(stack &s)
{
        return s.top==NULL;
}
```

Thêm phần tử vào satck:

```
void Push(stack &s, int x)
{
        stacknode p;
        p = new node[1];
        p->item=x;
        p->next=s.top;
        s.top=p;
}
```

Lấy phần tử ra khỏi stack:

```
int Pop(stack &s)
{
        stacknode p; int x;
        if(CheckEmpty(s)) cout<<"\nDanh sach rong";
        else {
                p = s.top; x = p->item;
                s.top = s.top->next;
                delete p; return x;
        }
}
```

Ví dụ 21: Chương trình minh họa cho các hàm trên

```
#include<iostream>
using namespace std;
typedef struct node{
        int item; node *next;
}*stacknode;
typedef struct Stack{
        stacknode top;
}stack;
int CheckEmpty(stack &s){
        return s.top==NULL;
}
void Push(stack &s, int x){
        stacknode p = new node[1];
        p->item=x; p->next=s.top; s.top=p;
}
int Pop(stack &s)
{
        stacknode p;int x;
        if(CheckEmpty(s)) cout<< "\nDanh sach rong";
        else{
                p = s.top; x = p->item; s.top = s.top->next;
                delete p; return x;
        }
}
int main()
{
        stack s; s.top = NULL;
        CheckEmpty(s);
        for(int x=1; x<22; x+=2) Push(s, x);
        int t=Pop(s);
        cout<< "Phan tu lay ra duoc: "<< t;
}
```

Kết quả chạy chương trình:

```
Phan tu lay ra duoc: 21
```

4.3.4 Một số ứng dụng của ngăn xếp

Một số ứng dụng của ngăn xếp:

- *Đảo ngược xâu kí tự.*
- *Tính giá trị biểu thức dạng hậu tố (postfix).*
- *Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix).*
- *Khử đệ quy lui.*
- ….

Đảo ngược xâu kí tự:

Bài toán đảo ngược xâu kí tự yêu cầu hiển thị các kí tự của xâu theo chiều ngược lại (ví dụ: Xâu ngược của xâu "lap trinh" là "hnirt pal").

Để giải quyết bài toán ta chỉ việc duyệt từ đầu đến cuối xâu và đưa vào ngăn xếp. Sau đó ta chỉ việc lấy các kí tự ra khỏi ngăn xếp và hiển thị chúng lên màn hình.

Biểu thực dạng hậu tố (postfix):

Biểu thức toán học mà con người vẫn hay dùng gọi là biểu thức dạng trung tố (infix). Ví dụ biểu thức:

(	4	*	(	(	(	6	-	3	)	*	(	8	-	3	)	)	+	1	)	)

Tuy nhiên đối với biểu thức dạng này thì việc tính toán đối với máy tính rất khó khăn do vậy để dễ dàng hơn cho máy tính trong việc tính toán người ta đưa ra dạng trình bày biểu thức toán học khác gọi là biểu thức dạng hậu tố (postfix). Theo cách trình bày này toán tử không nằm giữa 2 toán hạng mà nằm ngay sau 2 toán hạng. Chẳng hạn biểu thức trên có thể viết dưới dạng hậu tố như sau:

4	6	3	-	8	3	-	*	1	+	*

Thuật toán tính giá trị của biểu thức dạng hậu tố như sau: Duyệt biểu thức từ trái qua phải:

- *Nếu gặp toán hạng, đưa vào ngăn xếp.*
- *Nếu gặp toán tử, lấy ra 2 toán hạng từ ngăn xếp và sử dụng toán tử trên để tính, đưa kết quả vào ngăn xếp.*

Minh họa tính giá trị biểu thức hậu tố trên như sau:

Duyệt qua biểu thức, đưa 4, 6, 3 vào ngăn xếp:

3
6
4

Duyệt tiếp gặp toán tử dấu – lấy 2 toán hạng 3 và 6 ra khỏi ngăn xếp và thực hiện phép tính 6-3 và đưa kết quả 3 vào ngăn xếp:

3
4

Duyệt tiếp biểu thức, đưa 8 và 3 vào ngăn xếp:

3
8
3
4

Duyệt tiếp gặp toán tử dấu – lấy 2 toán hạng 3 và 8 ra khỏi ngăn xếp và thực hiện phép tính 8-3 rồi đưa kết quả 5 vào ngăn xếp:

5
3
4

Duyệt tiếp biểu thức gặp dấu * ta lấy ra khỏi stack 2 phần tử là 5 và 3 rồi thực hiện phép toán 3*5 rồi đưa kết quả 15 vào ngăn xếp:

15
4

Duyệt tiếp biểu thức, đưa 1 vào stack:

1
15
4

Duyệt tiếp biểu thức, gặp toán tử + ta lấy 2 phần tử 1 và 15 ra khỏi stack và thực hiện phép tính 15+1 và đưa kết quả 16 vào ngăn xếp:

16
4

Duyệt tiếp gặp toán tử dấu * ta lấy 2 toán tử 16 và 4 ra khỏi stack và thực hiện phép tính 4*16, kết quả thu được sẽ là 64. Do đã duyệt hết biểu thức nên kết quả cuối cùng của biểu thức là 64.

Chuyển một biểu thức dạng trung tố sang hậu tố (infix to postfix):

Thuật toán chuyển đổi 1 biểu thức từ dạng trung tố sang dạng hậu tố như sau: Duyệt biểu thức từ trái qua phải:

- *Nếu gặp dấu mở ngoặc "(" thì bỏ qua.*
- *Nếu gặp toán hạng, đưa vào biểu thức mới.*
- *Nếu gặp toán tử, đưa vào ngăn xếp.*
- *Nếu gặp dấu đóng ngoặc ")", lấy toán tử ra khỏi ngăn xếp và đưa vào biểu thức mới.*

Ta sẽ minh họa giải thuật bằng việc chuyển dổi biểu thức sau từ dạng trung tố sang dạng hậu tố *(cần lưu ý điền đầy đủ các dấu ngoặc):*

(	4	*	(	(	(	6	-	3	)	*	(	8	-	3	)	)	+	1	)	)

Duyệt từ trái qua phải, gặp dấu ngoặc (bỏ qua, số 4 ghi vào biểu thức mới và đưa dấu * vào stack:

*

Duyệt tiếp biểu thức, bỏ qua các dấu mở ngoặc, gặp toán hạng số 6 thêm vào biểu thức mới, gặp toán tử dấu – ta bỏ thêm vào stack.

4	6

-
*

Duyệt tiếp biểu thức, gặp toáng hạng 3 thêm vào biểu thức mới, gặp đóng ngoặc lấy toán tử dấu – ra khỏi stack và thêm vào biểu thức mới.

4	6	3	-

*

Duyệt tiếp biểu thức, gặp toán tử dấu * ta đưa vào stack, gặp dấu (bỏ qua, gặp toán hạng 8 đưa vào biểu thức mới, gặp toán tử dấu – đưa vào stack, gặp toán hạng 3 đưa vào biểu thức mới:

4	6	3	-	8	3

-
*
*

Duyệt tiếp biểu thức, gặp 2 dấu) ta lấy toán tử dấu – và dấu * ra khỏi stack và đưa vào biểu thức mới:

4	6	3	-	8	3	-	*

*

Duyệt tiếp biểu thức, gặp toán tử dấu + đưa vào stack, gặp toán hạng 1 thêm vào biểu thức mới:

4	6	3	-	8	3	-	*	1

+
*

Duyệt tiếp biểu thức gặp 3 dấu đóng ngoặc ta lấy 3 toán tử còn lại trong stack đưa vào biểu thức mới. Vì ta đã duyệt hết biểu thức nên biểu thức hậu tố cuối cùng thu được là:

4	6	3	-	8	3	-	*	1	+	*

Khử đệ quy:

Khử đệ quy ở đây là biến một thủ tục đệ quy thành một thủ tục chỉ chứa vòng lặp mà không ảnh hưởng gì đến các yếu tố khác, chứ không phải là thay đổi thuật toán.

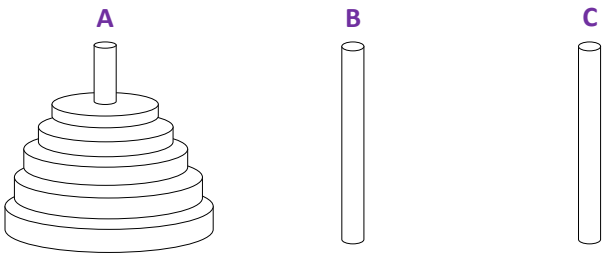
Khử đệ quy thực chất là chúng ta phải làm công việc của một trình biên dịch đối với một thủ tục, đó là: Đặt tất cả các giá trị của các biến cục bộ và địa chỉ của chỉ thị kế tiếp vào ngăn xếp (Stack), quy định các giá trị tham số cho thủ tục và chuyển tới vị trí bắt đầu thủ tục, thực hiện lần lượt từng câu lệnh. Sau khi thủ tục hoàn tất thì nó phải lấy ra khỏi ngăn xếp địa chỉ trả về và các giá trị của các biến cục bộ, khôi phục các biến và chuyển tới địa chỉ trả về. Các bước khử đệ quy dùng stack gồm:

- *Bước 1: Lưu các biến cục bộ và địa chỉ trở về.*
- *Bước 2: Nếu thoả điều kiện ngừng đệ qui thì chuyển sang bước 3. Nếu không thì tính toán từng phần và quay lại bước 1 (đệ qui tiếp).*
- *Bước 3: Khôi phục lại các biến cục bộ và địa chỉ trở về.*

Ví dụ sau đây minh hoạ việc dùng ngăn xếp để loại bỏ chương trình đệ qui của bài toán "tháp Hà Nội" (tower of Hanoi).

Ví dụ 22: Bài toán "tháp Hà Nội" được phát biểu như sau:

Có ba cọc A, B, C. Khởi đầu cọc A có một số đĩa xếp theo thứ tự nhỏ dần lên trên đỉnh. Bài toán đặt ra là phải chuyển toàn bộ chồng đĩa từ A sang B. Mỗi lần thực hiện chuyển một đĩa từ một cọc sang một cọc khác và không được đặt đĩa lớn nằm trên đĩa nhỏ.



Hình 14. Bài toán tháp Hà Nội

Giải bài toán tháp Hà Nội sử dụng giải thuật đệ quy như sau:

```
#include<iostream>
using namespace std;
void Move(int N, char A, char B, char C)
//N: so dia -  A,B,C coc nguon, dich, trung gian
{
	if (N==1)
		cout<<"Chuyen 1 dia tu cot "<<A<<" sang cot "<<B<<endl;
	else
	{
		Move(N-1, A, C, B);
		//chuyen n-1 dia tu coc nguon sang coc trung gian
		Move(1, A, B, C);
		//chuyen 1 dia tu coc nguon sang coc dich
		Move(N-1,C, B, A);
		//chuyen n-1 dia tu coc trung gian sang coc dich
	}
}
int main()
{
	int n;
	cout<<"Nhap so luong dia ban dau: ";
	cin>>n;
	cout<<"Cach chuyen nhu sau:\n";
	Move(n, 'A', 'B', 'C');
}
```

Kết quả chạy chương trình:

```
Nhap so luong dia ban dau: 4
Cach chuyen nhu sau:
Chuyen 1 dia tu cot A sang cot C
Chuyen 1 dia tu cot A sang cot B
Chuyen 1 dia tu cot C sang cot B
Chuyen 1 dia tu cot A sang cot C
Chuyen 1 dia tu cot B sang cot A
Chuyen 1 dia tu cot B sang cot C
Chuyen 1 dia tu cot A sang cot C
Chuyen 1 dia tu cot A sang cot B
Chuyen 1 dia tu cot C sang cot B
Chuyen 1 dia tu cot C sang cot A
Chuyen 1 dia tu cot B sang cot A
Chuyen 1 dia tu cot C sang cot B
Chuyen 1 dia tu cot A sang cot C
Chuyen 1 dia tu cot A sang cot B
Chuyen 1 dia tu cot C sang cot B
```

Giải bài toán tháp Hà Nội bằng khử đệ quy sử dụng stack:

```
#include<iostream>
using namespace std;
//Khai bao cau truc cho thap Ha Noi
struct HaNoi{
	int n;//So dia can di chuyen
	char Nguon, Dich, Tam;
};
//Khai bao cau truc cho node
typedef struct node{
	HaNoi info;
	node *Next;
}Node;
//Dinh nghia lai node dang con tro
typedef Node *NodePtr;
//Khoi tao Stack rong
void CreateStack(NodePtr &pTop)
{
	pTop=NULL ;
}
NodePtr CreateNode(HaNoi x)
{
	NodePtr p=new Node[1];
	p->info=x;	p->Next=NULL;
	return p;
}
//Kien tra stack rong?
int isEmpty(NodePtr pTop)
{
	 return pTop==NULL;
}
//Lay phan tu dau stack
HaNoi Pop(NodePtr &pTop)
{
	NodePtr p;   HaNoi value;
	p=pTop;//Luu phan tu dau stack vao p
	pTop=pTop->Next;//Cap nhat lai pTop
	value=p->info;
	delete p;   return value;
}
//Dua phan tu vao stack
void Push(NodePtr &pTop,HaNoi x)
{
	NodePtr data;
	data=new Node[1];
	data->info=x;
	data->Next=pTop;
	pTop=data;
}
```

```
//Ham giai bai toan thap Ha Noi
void ThapHaNoi(int &n)
{
	NodePtr pTop;//Stack chua thong tin di chuyen
	CreateStack(pTop);
	//Dua bai toan ban dau vao stack
	HaNoi hn;
	hn.n=n; hn.Nguon='A'; hn.Dich='B'; hn.Tam='C';
	Push(pTop, hn);
	while(!isEmpty(pTop))
	{
		hn=Pop(pTop);
		 //Truong hop chi co 1 dia,thi di chuyen dia nguon->dich
		if(hn.n==1)
		cout<<"\nDi chuyen dia tu cot "<<hn.Nguon<<" sang cot "<<hn.Dich;
		//Truong hop co nhieu hon 1 dia
		 else
		{
			HaNoi hn1;
			//Dua thong tin di chuyen n-1 dia tu tam(cot trung gian) -> dich
			hn1.n=hn.n-1;
			hn1.Nguon=hn.Tam;
			hn1.Dich=hn.Dich;
			hn1.Tam=hn.Nguon;
			Push(pTop, hn1);
			//Dua thong tin di chuyen 1 dia tu nguon -> dich
			hn1.n=1;
			hn1.Nguon=hn.Nguon;
			hn1.Dich=hn.Dich;
			hn1.Tam=hn.Tam;
			Push(pTop, hn1);
			//Dua thong tin di chuyen n-1 dia tu nguon -> tam(trung gian)
			hn1.n=hn.n-1;
			hn1.Nguon=hn.Nguon;
			hn1.Dich=hn.Tam;
			hn1.Tam=hn.Dich;
			Push(pTop, hn1);
		}
	}
}
int main()
{
	int n;
	cout<<"Nhap vao so dia ban dau: ";
	cin>>n;
	cout<<"Cach di chuyen nhu sau:";
	cout<<"\n==================================:";
	ThapHaNoi(n);
}
```

4.4 Hàng đợi – Queue

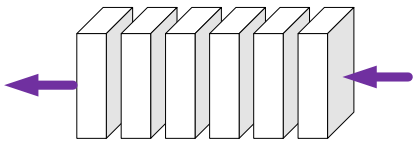
4.4.1 Tổng quan

Queue (hàng đợi): Là danh sách làm việc theo cơ chế FIFO (First In First Out), tức việc thêm 1 đối tượng vào hàng đợi hay lấy 1 đối tượng ra khỏi hàng đợi thực hiện theo cơ chế "vào trước ra trước".

Ví dụ tương tự hàng đợi là việc xếp hàng lên máy bay: Người nào xếp hàng trước thì lên máy bay trước tức là ra khỏi hảng đợi trước tiên.

Các thao tác trên hàng đợi bao gồm:

- *Thêm đối tượng vào cuối hàng đợi.*
- *Lấy đối tượng từ đầu hàng đợi.*
- *Kiểm tra hàm đợi có rỗng hay không?*
- *Lấy giá trị của phần tử đầu của hàng đợi mà không hủy nó.*



Hình 15. Minh họa Queue

4.4.2 Cài đặt hàng đợi bằng mảng

Ví dụ 23: Cài đặt hàng đợi bằng mảng.

```
#include<stdio.h>
const int max=100;
//Cau truc du lieu cua Queue
typedef struct tagQueue
{
	int a[max];
	int Front; //chi so cua phan tu dau trong Queue
	int Rear; //chi so cua phan tu cuoi trong Queue
}Queue;
//Khoi tao Queue rong
void CreateQueue(Queue &q)
{
	q.Front=-1;
	q.Rear=-1;
}
//Lay 1 phan tu tu Queue
int DeQueue(Queue &q, int &x)
{
	if(q.Front!=-1)		//Queue khong rong
	{
		x=q.a[q.Front]; //Lay ra phan tu dau tien cua mang
		q.Front++; //Thay doi phan tu dau tien cua Queue
		if(q.Front>q.Rear) //Xay ra khi Queue ban dau chi co 1 phan tu
		{
			q.Front=-1;
			q.Rear=-1;
		}
```

```
                return 1;
        }
        else //queue trong
        {
                printf( "Queue rong");
                return 0;
        }
}
//Them 1 phan tu vao Queue
int EnQueue(Queue &q, int x)
{
        int i, f, r;
        if(q.Rear - q.Front+1==max) //queue bi day khong the them vao duoc nua
                return 0;
        else
        {
                if(q.Front==-1) //Queue rong
                {
                        q.Front=0;
                        q.Rear=-1;
                }
                if(q.Rear==max-1) //Queue day
                {
                        f=q.Front;
                        r=q.Rear;
                        for(i=f; i<=r; i++)
                                q.a[i-f]=q.a[i]; //Reset lai cac chi so cho mang a
                        q.Front=0; //Thay doi lai chi so dau va cuoi cua Queue
                        q.Rear=r-f;
                }
                q.Rear++;
                q.a[q.Rear] = x; //Them phan tu moi vao cuoi
                return 1;
        }
}
int main()
{
        Queue q;
        int i, x, t;
        CreateQueue(q);
        for(i=6; i<=20; i+=1)
                EnQueue(q, i);
        t=DeQueue(q, x);
        if(t==1)
                printf( "Gia tri lay ra khoi hang doi la: %d", x);
}
```

Kết quả chạy chương trình:

```
Gia tri lay ra khoi hang doi la: 6
```

Minh họa hàng đợi ở ví dụ trên bằng mảng:

Chỉ số	0	1	2	…	14	…	max-2	max-1
Mảng a	6	7	8	…	20	…		
	Front				Rear			

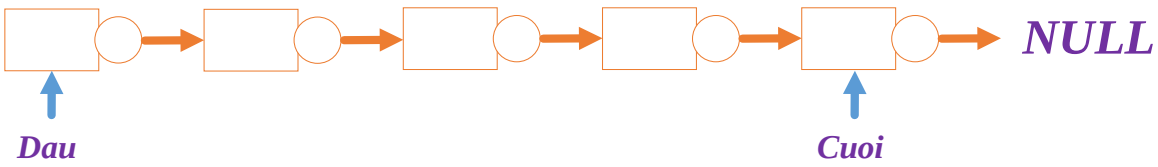
Sau khi lấy 1 phần tử ra khỏi hàng đợi:

Chỉ số	0	1	2	…	14	…	max-2	max-1
Mảng a	6	7	8	…	20	…		
		Front			Rear			

4.4.3 Cài đặt hàng đợi bằng danh sách liên kết đơn

Để cài đặt hàng đợi bằng DSLK, ta cũng sử dụng 1 DSLK đơn với 2 con trỏ ***dau** và ***cuoi** để lưu trữ nút đầu và cuối của danh sách.

Các thao tác thêm vào và lấy ra ta sẽ thực hiện theo thứ tự ở cuối (thêm vào) và đầu (lấy ra) của danh sách liên kết đơn.



Hình 16. Minh họa hàng đợi bằng DSLK đơn

Khai báo CTDL của hàng đợi bằng DSLK đơn:

```
//Khai báo cấu trúc của nút
struct Node{
        int item;
        Node *tiep;
};
//Khai báo hàng đợi
struct Queue{
        Node *dau;
        Node *cuoi;
};
```

Khởi tạo hàng đợi rỗng:

```
//Khởi tạo hàng đợi rỗng
void CreateQueue(Queue &q)
{
        q.dau = q.cuoi = NULL;
}
```

Hàm thêm phần tử vào hàng đợi:

```
void PutQueue(Queue &q, int x)
{
	Node *data = new Node[1];
	data->item = x; data->tiep = NULL; //Tạo nút mới
	if(q.dau==NULL) //Hàng đợi rỗng
		q.dau=q.cuoi=data;
	else  //Thêm nút vừa tạo vào cuối hàng đợi
	{
		q.cuoi->tiep = data;   q.cuoi =data;
	}
}
```

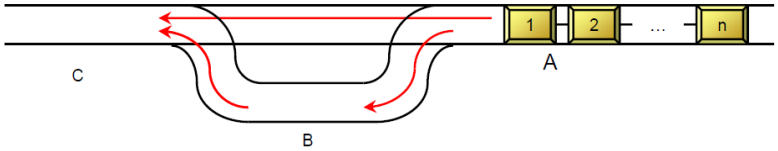
Lấy phần tử ra khỏi hàng đợi:

```
int GetQueue(Queue &q, int &Gia_Tri_Lay)
{
	Node *data=new Node[1];
	if(q.dau==NULL) return 0;//Lấy thất bại
	else
	{	data = q.dau;
		Gia_Tri_Lay = data->item;
		q.dau   = q.dau->tiep;
		delete data;//Thu hoi bo nho cua nut moi lay
		return 1;//Lấy thành công
	}
}
```

Cài đặt hàm main() cho chương trình:

```
int main()
{
	Queue q;
	int i, x;
	CreateQueue(q);
	for(i=1; i<=10; i++)
		PutQueue(q, i);
	if(GetQueue(q, x))
		cout<<"Gia tri lay ra khoi hang doi: "<< x;
}
```

Ví dụ 24: Bài toán di chuyển toa tàu (hình dưới): Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các vạch mũi tên màu đỏ. Ta cần di chuyển các toa từ A -> C sao cho tại C các toa tàu được sắp xếp các thứ tự mới nào đó. Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không ? Nếu có, hãy trình bày cách chuyển. Ta áp dụng hàng đợi tại B để giải quyết bài toán này như sau:



Hình 17. Bài toán di chuyển toa tàu.

```
#include<iostream>
using namespace std;
struct Node{
	int So_toa;
	Node *tiep;
};
struct QueueB{
	Node *dau;
	Node *cuoi;
};
//Ham them Node
void PutQueueB(QueueB &q, int x)
{
	Node *data = new Node[1];
	data->So_toa = x;
	data->tiep = NULL;
	if(q.dau==NULL)
		q.dau=q.cuoi=data;
	else
	{
		q.cuoi->tiep = data;
		q.cuoi =data;
	}
}
//Ham lay Node
int GetQueueB(QueueB &q, int &Gia_Tri_Lay)
{
	Node *data=new Node[1];
	if(q.dau==NULL)
		return 0;
	else
	{	data = q.dau;
		Gia_Tri_Lay = data->So_toa;
		q.dau   = q.dau->tiep;
		delete data;
		return 1;
	}
}
int main()
{
	int n, i, j, t;
	QueueB B;
	B.dau = B.cuoi = NULL;
	cout<< "Nhap so luong toa tau o vi tri A: ";
	cin>>n;
	int C[n];//Luu tru thu tu cac toa tau can sap xep o C
	cout<< "Nhap thu tu cac toa tau can xep vao vi tri C:\n";
	for(i=0;i<n;i++)
	cin>>C[i];
```

```
        //Kiểm tra điều kiện sắp xếp
        for(j=0, i=1; i<=n; i++)//i là số các toa ở A, j là chỉ số mảng C
        {
                if(C[j]==i)
                        j++;
                else
                        PutQueueB(B, i);
                while(B.dau!=NULL && B.dau->So_toa==C[j])
                {
                        GetQueueB(B, t);
                        j++;
                }
        }
        if(j<n-1)
                cout<< "Khong the sap xep!";
        else //In ra kết quả
        {
                cout<< "Cac buoc thuc hien di chuyen cac toa nhu sau:\n";
                for(j=0, i=1; i<=n; i++)
                {
                        if(C[j]==i)
                        {
                                cout<< "Chuyen toa so "<<i<< " tu A - > C\n";
                                j++;
                        }
                        else
                        {
                                PutQueueB(B, i);
                                cout<< "Chuyen toa so "<<i<< " tu A - > B\n";
                        }
                        while(B.dau!=NULL && B.dau->So_toa==C[j])
                        {
                                GetQueueB(B, t);
                                cout<< "Chuyen toa so "<<t<< " tu B - > C\n";
                                j++;
                        }
                }
        }
}
```

Chạy thử chương trình:

- Trường hợp không sắp xếp được:

```
Nhap so luong toa tau o vi tri A: 6
Nhap thu tu cac toa tau can xep vao vi tri C:
1       3       6       5       2       4
Khong the sap xep!
```

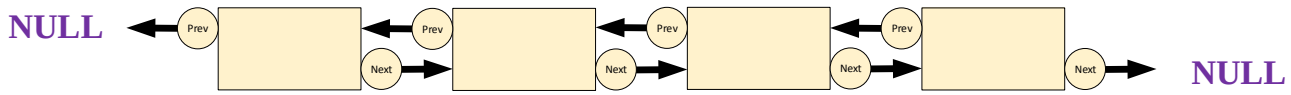
- Trường hợp sắp xếp được:

```
Nhap so luong toa tau o vi tri A: 6
Nhap thu tu cac toa tau can xep vao vi tri C:
1       2       5       6       3       4
Cac buoc thuc hien di chuyen cac toa nhu sau:
Chuyen toa so 1 tu A - > C
Chuyen toa so 2 tu A - > C
Chuyen toa so 3 tu A - > B
Chuyen toa so 4 tu A - > B
Chuyen toa so 5 tu A - > C
Chuyen toa so 6 tu A - > C
Chuyen toa so 3 tu B - > C
Chuyen toa so 4 tu B - > C
```

4.5 Một số dạng danh sách liên kết khác

4.5.1 Danh sách liên kết kép

Danh sách liên kết kép là danh sách liên kết mà mỗi nút trong đó sẽ liên kết với cả nút đằng sau (***next***) và nút đằng trước nó (***preview***).



Hình 18. Minh họa danh sách liên kết kép.

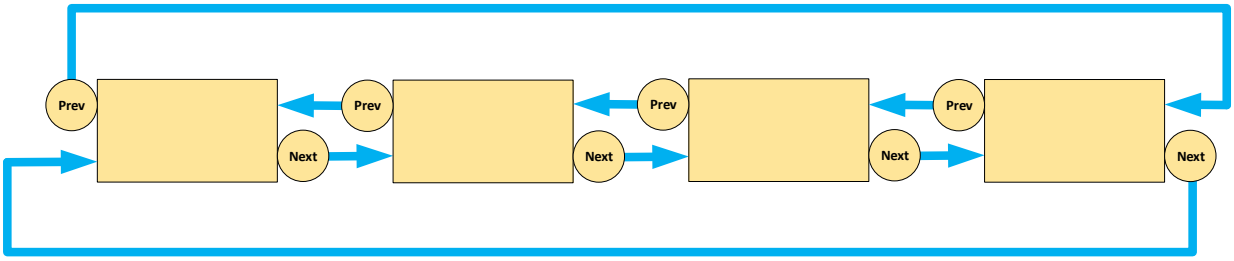
Cấu trúc dữ liệu của 1 nút trong DSLK kép:

```
struct Node{
        int item;
        Node *truoc;
        Node *sau;
};
```

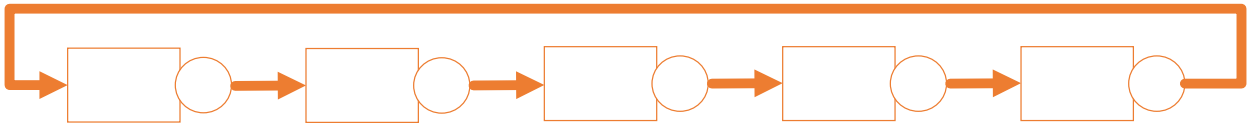
Dựa vào cấu trúc nút ở trên ta cũng có thể dễ dàng thực hiện các thao tác đối với danh sách liên kết kép tương tự như là với danh sách liên kết đơn.

4.5.2 Danh sách liên kết vòng

Danh sách liên kết vòng là danh sách liên kết trong đó nút cuối cùng sẽ liên kết với nút đầu tiên.



Hình 19. Danh sách liên kết vòng kép



Hình 20. Danh sách liên kết vòng đơn

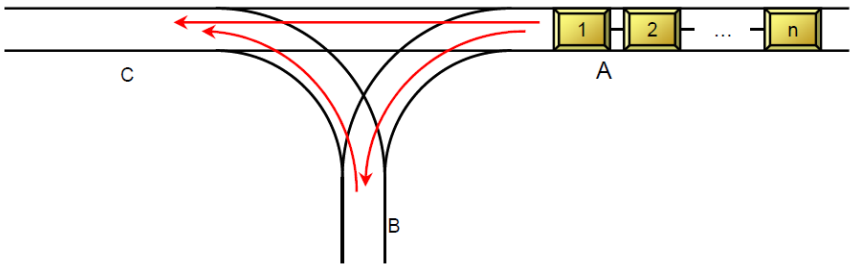
BÀI TẬP CHƯƠNG 4

Bài 1. Sử dụng kiểu dữ liệu danh sách liên kết đơn, viết chương trình quản lý danh sách sinh viên, thông tin mỗi sinh viên bao gồm: mã số, họ tên, năm sinh, điểm trung bình. Thực hiện các yêu cầu sau:

a) Khai báo, khởi tạo, và nhập danh sách.
b) Duyệt và in danh sách.
c) Tìm kiếm một sinh viên theo họ tên.
d) Chèn thêm một sinh viên mới vào sau một sinh viên có mã số chỉ định.
e) Xóa một sinh viên có mã sinh viên chỉ định.
f) Sắp xếp danh sách theo mã số sinh viên tăng dần.
g) Sắp xếp danh sách theo tên sinh viên theo thứ tự bảng chữ cái A, B, C,….
h) Hủy danh sách khi kết thúc chương trình.

Bài 2. Xây dựng cấu trúc dữ liệu stack cài đặt bằng danh sách liên kết để đổi số hệ 10 sang các hệ cơ số khác, hãy xây dựng chuỗi kết quả là số n ở hệ cơ số a.

Bài 3. Bài toán di chuyển toa tàu (hình dưới): Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các vạch mũi tên màu đỏ. Ta cần di chuyển các toa từ A -> C sao cho tại C các toa tàu được sắp xếp các thứ tự mới nào đó. Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không ? Nếu có, hãy trình bày cách chuyển.



Hình 21. Minh họa bài toán di chuyển toa tàu

Bài 4. Xây dựng CTDL queue cài đặt bằng DSLK đơn để mô phỏng qui trình cho thuê máy ở một phòng NET với các yêu cầu:

- *Danh sách các máy trống A – Dùng kiểu queue, mỗi nút chứa số máy;*
- *Danh sách khách đang chờ nhận máy Q – Dùng kiểu queue, mỗi nút chứa họ tên người thuê.*
- *Danh sách khách đang thuê máy H - Dùng kiểu DSLK đơn, mỗi nút có họ tên khách, giờ bắt đầu thuê máy, số máy.*
- *Các chức năng cần phải có:*
  - ***Đăng ký thuê máy**: Còn máy trong A thì thêm khách vào H và lấy máy vừa cho thuê ra khỏi queue A. Nếu A trống thì thêm khách vào queue Q.*
  - ***Trả máy**: Thêm máy trả vào queue A, bỏ người đó khỏi danh sách thuê H.*
  - ***Phục vụ**: Kiểm tra A và Q để phục vụ và cập nhật lại A, Q, H.*

Bài 5. Cài đặt 1 DSLK đơn biểu diễn thông tin cho n thành phố. Thông tin các thành phố gồm: Tên, diện tích, dân số.

a) Nhập/xuất dữ liệu cho DSLK.
b) Tìm thông tin thành phố có diện tích lớn nhất và thêm vào sau nó 1 thành phố.
c) Xóa khỏi DSLK thành phố "Ha Noi" nếu có.
d) Sắp xếp DSLK theo chiều tăng dần của dân số.

Bài 6. Sử dụng kiểu dữ liệu danh sách liên kết đôi vòng, viết chương trình quản lý các chuyến bay của một công ty hàng không, mỗi chuyến bay gồm: Mã chuyến, ngày, giờ khởi hành, điểm đến. Thực hiện các yêu cầu sau:

- *Khai báo và khởi tạo danh sách list.*
- *Nhập danh sách bằng cách thêm vào list ở vị trí phù hợp để danh sách có thứ tự tăng của mã chuyến.*
- *Hãy in tất cả các chuyến bay khởi hành trong ngày chỉ định.*
- *Hủy danh sách khi kết thúc chương trình.*

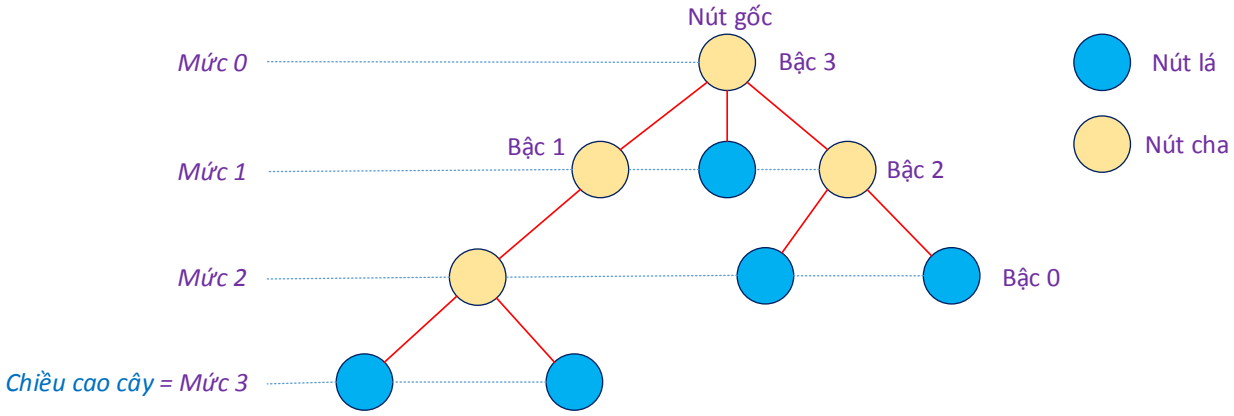
Chương 5. CÂY

5.1 Tổng quan về cấu trúc cây

Định nghĩa cây - Cây là một tập hợp T các phần tử (gọi là nút của cây), trong đó có một nút đặc biệt gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau $T_1, T_2, \ldots, T_n$ theo quan hệ phân cấp, trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp i+1. Quan hệ này người ta gọi là quan hệ cha – con.

Một số khái niệm cơ bản:

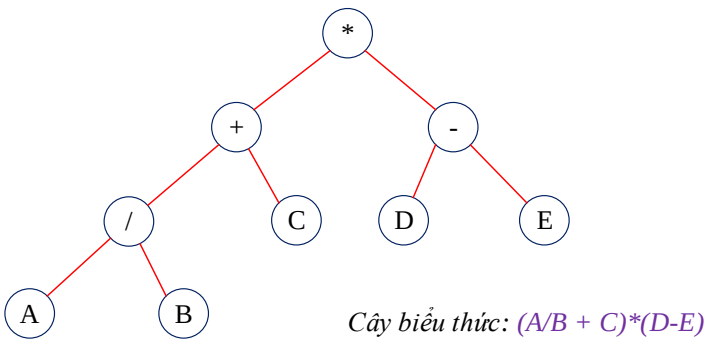
- *Bậc của một nút: Là số cây con của nút đó .*
- *Bậc của một cây: Là bậc lớn nhất của các nút trong cây*
- *Nút gốc: Là nút không có nút cha.*
- *Nút lá: Là nút có bậc bằng 0 .*
- *Mức của một nút:*
  - *Mức của nút gốc bằng 0.*
  - *Mức của các nút con bằng mức của nút cha cộng thêm 1.*
- *Độ dài đường đi từ gốc đến nút x: Là số nhánh cần đi qua kể từ gốc đến x.*
- *Chiều cao của cây: Là mức lớn nhất trong cây.*



Hình 22. Minh họa cây và một số khái niệm cơ bản.

Một số ví dụ về cây:

- *Sơ đồ tổ chức của 1 công ty*
- *Mục lục của một cuốn sách*
- *Gia phả của một họ tộc*
- *Cấu trúc của thư mục trên ổ đĩa*
- *Biểu thức toán học cũng có thể biểu diễn bằng cây*

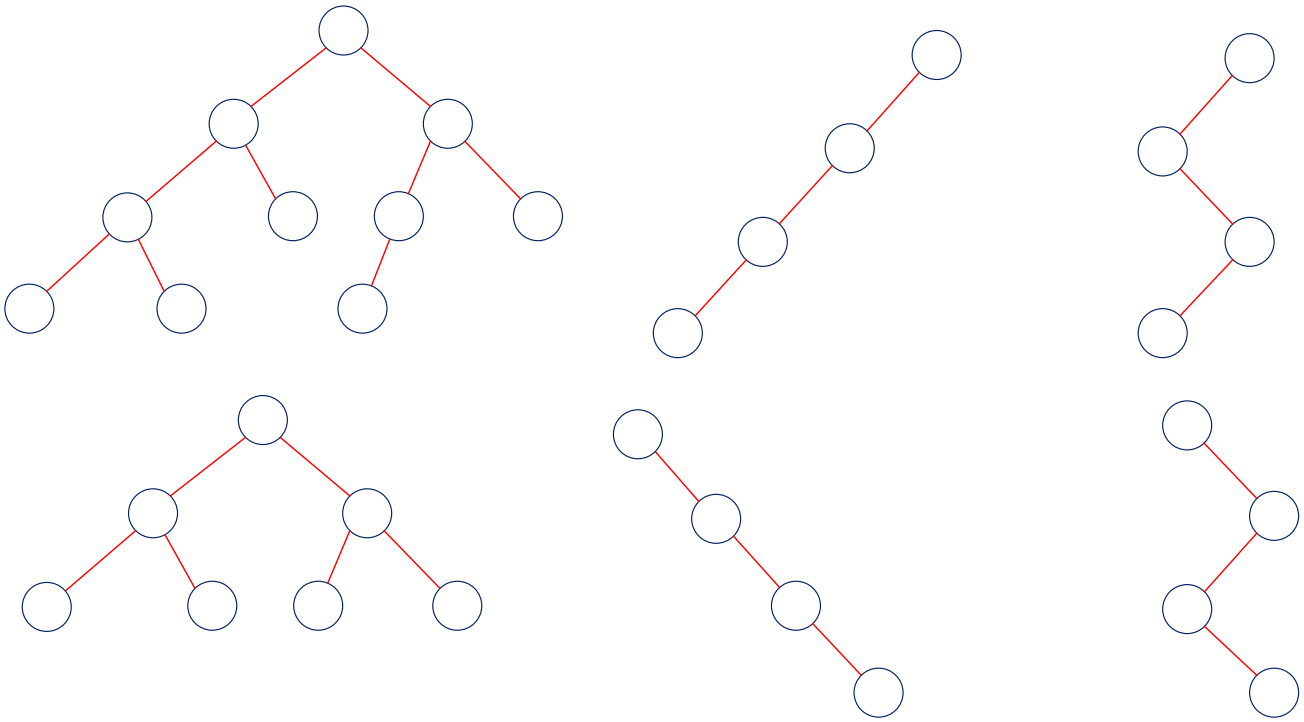


Hình 23. Cây biểu thức toán học

5.2 Cây nhị phân

5.2.1 Tổng quan về cây nhị phân

Khái niệm: Cây nhị phân là một dạng quan trọng của cấu trúc cây. Nó có đặc điểm là mọi nút trên cây chỉ có tối đa hai nhánh con.



Hình 24. Minh họa một số dạng cây nhị phân.

Một số tính chất của cây nhị phân:

- *Số lượng tối đa các nút ở mức i của cây nhị phân là $2*i$, tối thiểu là 1 ($i \geq 1$).*
- *Số lượng tối đa các nút trên một cây nhị phân có chiều cao h là: $2^{h+1}-1$ và số lượng nút tối thiểu là: $h+1$ ($h \geq 1$).*
- *Cây nhị phân hoàn chỉnh có n nút thì chiều cao của nó là: $h = log_2(n+1) -1$.*

5.2.2 Biểu diễn cây nhị phân bằng danh sách liên kết

- *Mỗi nút của cây nhị phân có tối đa 2 nút con, do vậy sử dụng DSLK để biểu diễn cây nhị phân sẽ hữu hiệu nhất. Mỗi nút của cây nhị phân khi đó sẽ có 3 thành phần:*
  - *Thành phần item chứa thông tin về nút.*
  - *Con trỏ left trỏ đến nút con bên trái.*
  - *Con trỏ right trỏ đến nút con bên phải.*
- *Nếu nút có ít hơn 2 nút con thì các thành phần left hoặc right sẽ trỏ tới NULL.*
- *Để tăng tính di chuyển trong cây ta có thể thêm con trỏ parent trỏ đến nút cha.*

Cấu trúc dữ liệu của nút và khai báo cây:

```
struct Node{
        int item;
        Node *left, *right;
};
typedef struct Node *TREE;
TREE test;
```

Duyệt cây nhị phân: Phép duyệt cây nhị phân được chia thành 3 loại:

- *Duyệt thứ tự trước – Duyệt nút trước tiên (**NLR** hoặc **NRL**)*
- *Duyệt thứ tự giữa – Duyệt nút ở giữa (**LNR** hoặc **RNL**)*
- *Duyệt thứ tự sau – Duyệt nút sau cùng (**LRN** hoặc **RLN**)*

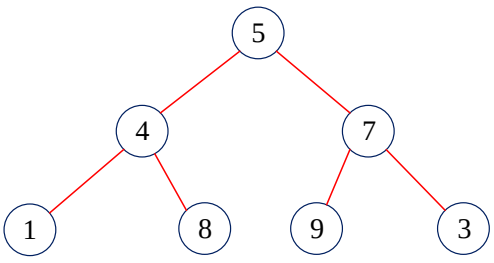
Độ phức tạp của giải thuật là $O(log_2 h)$. Trong đó h là chiều cao của cây.

```
// Hàm duyệt trước từ trái qua phải
void DuyetTruoc(TREE test)//NLR (Node – Left – Right)
{
	if(test != NULL)
	{
		cout<<test->key;
		DuyetTruoc(test->left);
		DuyetTruoc(test->right);
	}
}
// Hàm duyệt giữa từ trái qua phải
void DuyetGiua(TREE test)//LNR (Left – Node – Right)
{
	if(test != NULL)
	{
		DuyetGiua (test->left);
		cout<<test->key;
		DuyetGiua (test->right);
	}
}
// Hàm duyệt sau từ trái qua phải
void DuyetSau(TREE test)//LRN (Left – Right – Node)
{
	if(test != NULL)
	{
		DuyetSau (test->left);
		DuyetSau (test->right);
		cout<<test->key;
	}
}
```

5.2.3 Một số cách biểu diễn cây nhị phân khác

Biểu diễn cây nhị phân bằng mảng:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
5	**4**	**7**	**1**	**8**	**9**	**3**



Biểu diễn bằng mảng các nút cha:

5	**4**	**7**	**1**	**8**	**9**	**3**
0	**5**	**5**	**4**	**4**	**7**	**7**

Thêm con trỏ parent vào các nút:

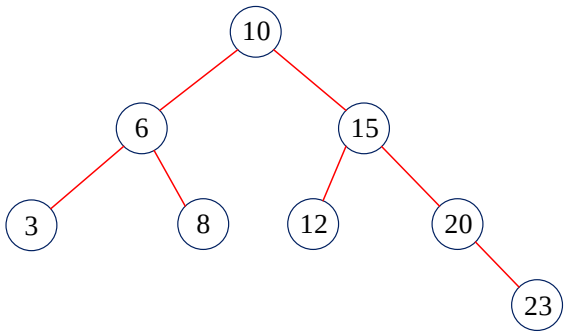
```
struct Node{
        int item;
        Node *left, *right, *parent ;
};
typedef struct Node *TREE;
TREE test;
```

5.3 Cây nhị phân tìm kiếm

5.3.1 Tổng quan về cây nhị phân tìm kiếm

Định nghĩa: Cây nhị phân tìm kiếm là cây nhị phân mà khoá tại mỗi nút cây lớn hơn khoá của tất cả các nút thuộc cây con bên trái và nhỏ hơn khoá của tất cả các nút thuộc cây con bên phải.

Lưu ý: Dữ liệu lưu trữ tại mỗi nút có thể rất phức tạp như là một record chẳng hạn, trong trường hợp này khoá của nút được tính dựa trên một trường nào đó, ta gọi là trường khoá. Trường khoá phải chứa các giá trị có thể so sánh được, tức là nó phải lấy giá trị từ một tập hợp có thứ tự.



Hình 25. Cây nhị phân tìm kiếm

Khai báo và khởi tạo cây nhị phân tìm kiếm:

```
//Cấu trúc của nút
struct Node{
        int key; //Trường dữ liệu ở ví dụ này là số nguyên
        Node *left;
        Node *right;
};
//Khai báo cấu trúc cây
typedef struct Node *TREE;
void Create(TREE &test) //Khởi tạo cây rỗng
{
        test=NULL;
}
```

5.3.2 Các giải thuật trên cây nhị phân tìm kiếm

Các thao tác chính trên cây nhị phân tìm kiếm:

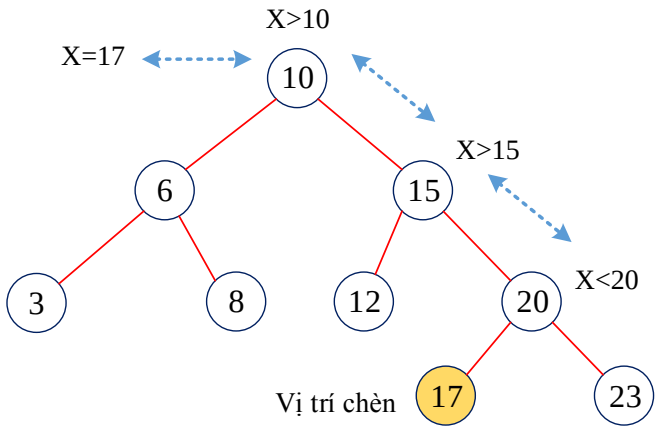
- *Tạo 1 nút có trường Key bằng x.*
- *Thêm 1 nút vào cây nhị phân tìm kiếm.*
- *Xoá 1 nút có Key bằng x trên cây.*
- *Tìm 1 nút có khoá bằng x trên cây.*

Tạo một nút có khóa bằng x:

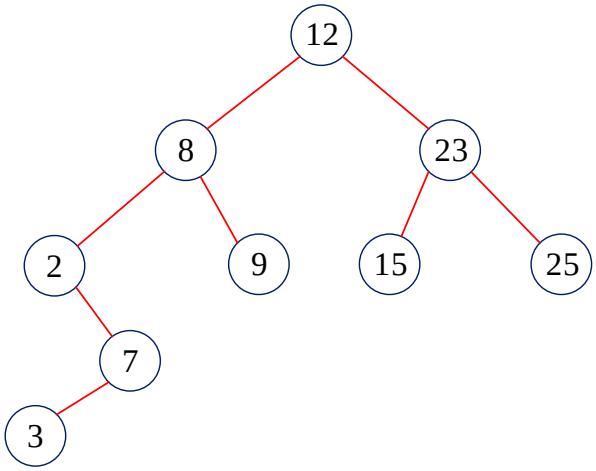
```
Node* CreateNode(int x)
{
	Node *data=new Node[1];
	if(data==NULL) return NULL; //Bộ nhớ full
	data->key = x;
	data->left = data->right =NULL;
	return data;
}
```

Thêm một nút có khóa x:

```
int Put(TREE &test, int x)
{
	if(test!=NULL)
	{
		if(test->key == x) return 0;
		if(test->key>x) return Put(test->left, x);
		else return Put(test->right, x);
	}
	test = CreateNode(x);
	return 1;
}
```



Hình 26. Minh họa thêm nút vào cây nhị phân tìm kiếm



Hình 27. Tạo cây nhị phân tìm kiếm từ dãy: 12, 8, 23, 15, 2, 7, 9, 3, 25.

Tìm nút có khóa x: Không dùng đệ quy.

```
Node* Search1(TREE t, int x)
{
	Node *p = t;
	while(p!=NULL) {
		if(x==p->key) return p; //Tim dc nut p co khoa x
		if(x<p->key) p=p->left; //Tim tiep o ben trai
		if(x>p->key) p=p->right; //Tim tiep o ben phai
	}
	return NULL; //Khong tim dc nut nao co khoa x
}
```

Tìm nút có khóa x: Dùng đệ quy.

```
Node* Search2(TREE t, int x)
{
	if(t!=NULL) {
		if(x==t->key) return t;
		if(x<t->key) return Search2(t->left, x);
		if(x>t->key) return Search2(t->right, x);
	}
	return NULL; //Khong tim dc nut nao co khoa x
}
```

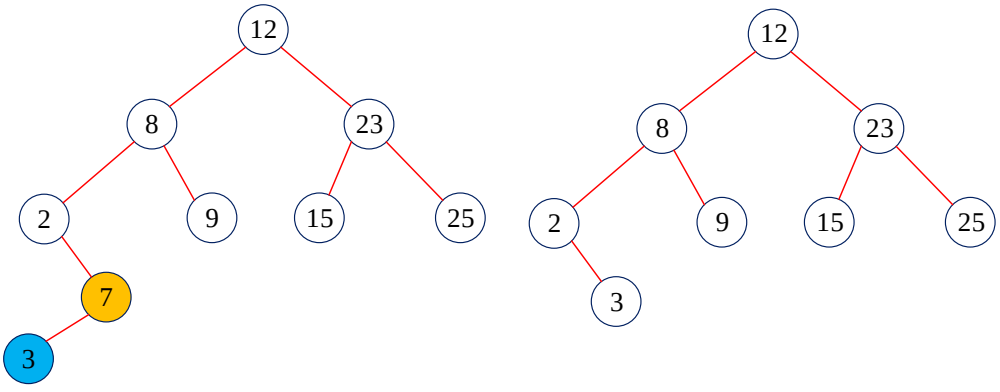
Hủy nút có khóa x trên cây:

Có 3 trường hợp khi hủy 1 nút X trên cây:

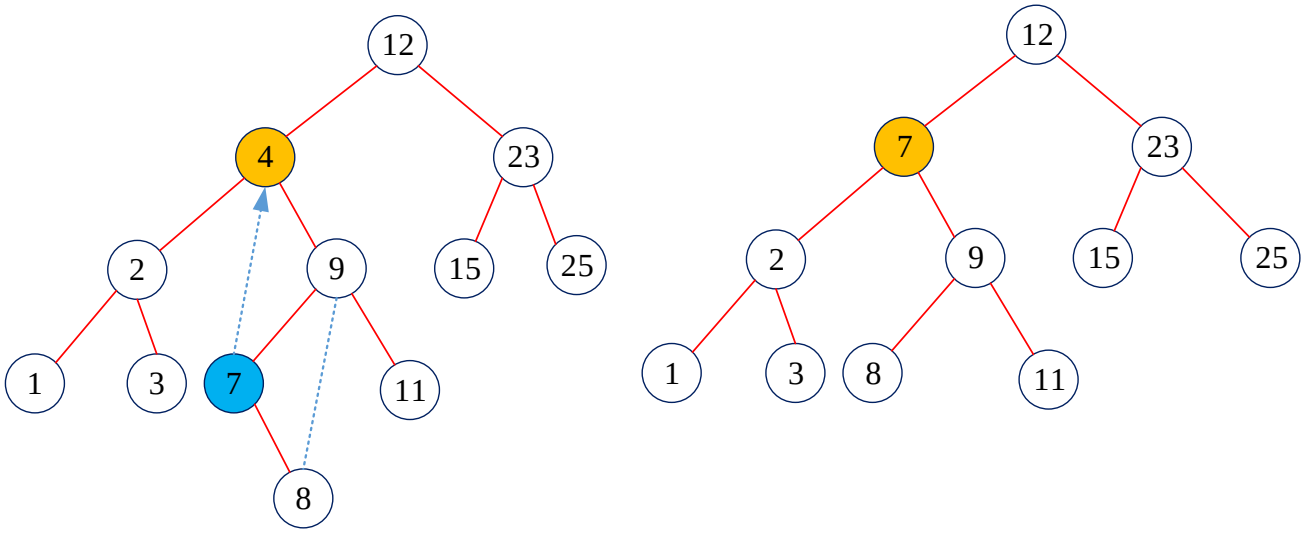
- *X là nút lá – Hủy bình thường mà không ảnh hưởng đến các nút khác trên cây.*
- *X chỉ có 1 cây con - Trước khi xoá x ta móc nối cha của X với con duy nhất cùa X.*
- *X có đầy đủ 2 cây con - Ta dùng cách xoá gián tiếp.*

Cách xóa gián tiếp như sau:

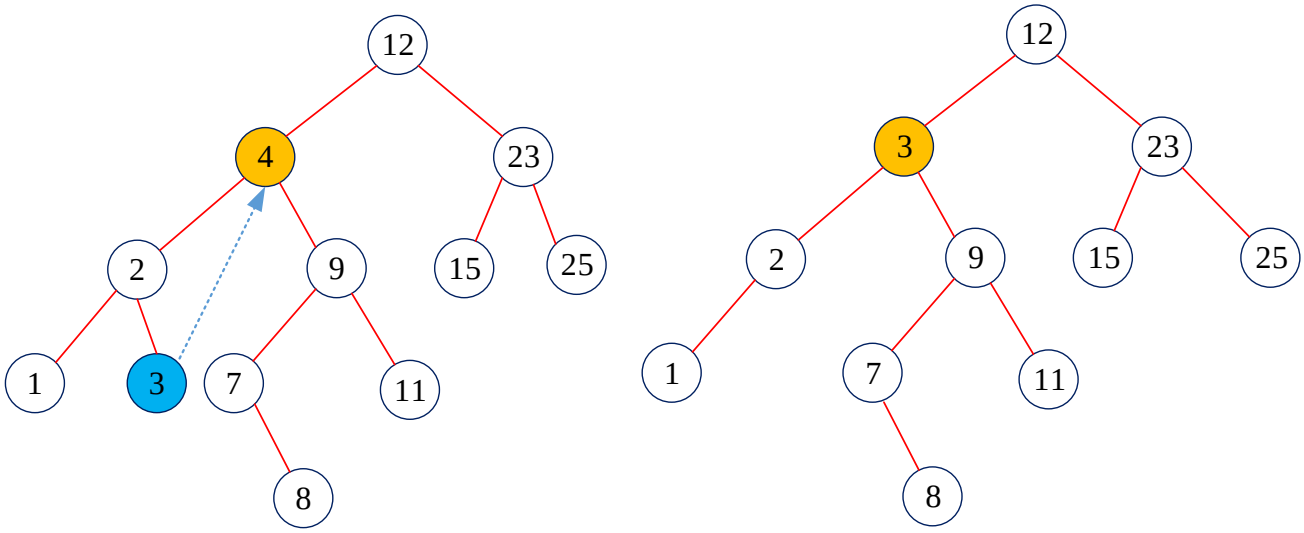
- *Thay vì hủy X ta tìm phần tử thế mạng Y. Nút Y có tối đa 1 cây con.*
- *Thông tin lưu tại nút Y sẽ được chuyển lên lưu tại X.*
- *Ta tiến hành xoá hủy nút Y (xoá Y giống 2 trường hợp đầu).*
- *Cách tìm nút thế mạng Y cho X: Có 2 cách*
  - *Cách 1: Nút Y là nút có khoá nhỏ nhất (trái nhất) bên cây con phải X.*
  - *Cách 2: Nút Y là nút có khoá lớn nhất (phải nhất) bên cây con trái của X.*



Hình 28. Hủy nút có 1 cây con (X=7, phần tử thế mạng Y=3).



Hình 29. Hủy nút có 2 cây con, phần tử thế mạng ở câu con bên phải (X=4, Y =7).



Hình 30. Hủy nút có 2 cây con, phần tử thế mạng ở câu con bên trái (X=4, Y =3).

Hàm tìm phần tử thay thế:

```
//Ham thay the trong truong hop phan tu the mang o cay con ben phai
void thaythe(TREE &nut, TREE &t)
//'t' la nut con ben phai cua 'nut', 'nut' la nut can thay the
{
	if(t->left!=NULL) //'t' van con co nut con ben trai
		thaythe(nut, t->left);
	else
	{
		//Thay key cho 'nut' bang key cua nut nho nhat cua cay con ben phai
		nut->key = t->key;
		nut=t;
		t=t->right;
	}
}
```

Cài đặt hàm xóa nút có key là x:

```
void DelNode(TREE &test, int x)
{
	if(test==NULL) cout<<"Khong tim thay nut can xoa\n";
	else
	{
		if(test->key<x) DelNode(test->right, x);
		else
		{
			if(test->key>x) DelNode(test->left, x);
			else
			{
				Node *p;
				p=test;
				if(test->left==NULL)test=test->right;
				else
				{
					if(test->right==NULL)test=test->left;
					else thaythe(p,test->right);
				}
				delete p;
			}
		}
	}
}
```

Ví dụ 25: Minh họa cây nhị phân tìm kiếm:

```
#include<iostream>
using namespace std;
//Cau truc cua nut
struct Node{
	int key;
	Node *left;
	Node *right;
};
//Khai bao cay
typedef struct Node *TREE;
//Khoi tao cay rong
void Create(TREE &test)
{
	test=NULL;
}
//Tao nut
Node* CreateNode(int x)
{
	Node *data=new Node[1];
	if(data==NULL)return NULL;//Bo nho full
	data->key = x;
```

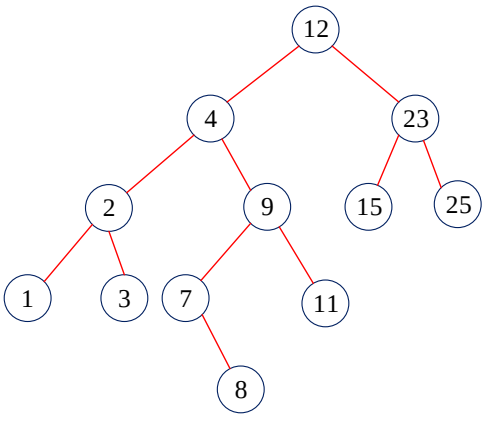
```
        data->left = data->right =NULL;
        return data;
}
//Them nut vao cay
int Put(TREE &test, int x)
{
        if(test!=NULL)//Chua co cho de them nut
        {
                if(test->key == x) return 0;//Cac nut khong dc co key giong nhau
                if(test->key>x) return Put(test->left, x);//Them x vao nut con ben trai
                else return Put(test->right, x);//Them x vao nut con ben phai
        }
        //da co cho de them nut
        test = CreateNode(x);
        return 1;
}
//Tim nut co khoa x khong dung de quy
Node* Search(TREE t, int x)
{
        Node *p = t;
        while(p!=NULL)
        {
                if(x==p->key)return p;//Tim dc nut p co khoa x
                if(x<p->key) p=p->left;//Tim tiep o nut ben trai
                if(x>p->key) p=p->right; //Tim tiep o nut ben phai
        }
        return NULL;//Khong tim dc nut nao co khoa x
}
//Tim phan tu thay the (dung cho truong hop xoa nut co day du 2 cay con)
void thaythe(TREE &nut, TREE &t) //'t' la nut con ben phai cua 'nut' can thay the
{
        if(t->left!=NULL) //'t' van con co nut con ben trai
                thaythe(nut,t->left);
        else //'t'khong con nut con nao ben trai
        {
                nut->key = t->key; //Thay key cho 'nut' bang key cua nut nho nhat ben phai
                nut=t;
                t=t->right;
        }
}
//Xoa nut co khoa x khoi cay
void DelNode(TREE &test, int x)
{
        if(test==NULL) cout<<"Khong tim thay nut can xoa\n";
        else
        {
                if(test->key<x) DelNode(test->right, x);
                else
                {
```

```
                        if(test->key>x) DelNode(test->left, x);
                        else
                        {
                                Node *p;
                                p=test;
                                if(test->left==NULL) test=test->right;
                                else
                                {
                                        if(test->right==NULL) test=test->left;
                                        else thaythe(p, test->right);
                                }
                                delete p;
                        }
                }
        }
}
//Duyet cay NLR
void DuyetTruoc(TREE test)
{
        if(test != NULL)
        {
                cout<<test->key<<"  ";
                DuyetTruoc(test->left);
                DuyetTruoc(test->right);
        }
}
//Duyet cay LNR
void DuyetGiua(TREE test)
{
        if(test != NULL)
        {
                DuyetGiua(test->left);
                cout<<test->key<<"  ";
                DuyetGiua(test->right);
        }
}
//Duyet cay LRN
void DuyetSau(TREE test)
{
        if(test != NULL)
        {
                DuyetSau(test->left);
                DuyetSau(test->right);
                cout<<test->key<<"  ";
        }
}
int main()
{
        TREE test, t;
```

```
        int i, n, x;
        Create(test);
        cout<<"Nhap so luong nut trong cay: "; cin>>n;
        cout<<"Nhap vao cac nut cho cay:\n";
        for(i=1; i<=n; i++)
        {
                cin>>x;
                Put(test, x);
        }
        cout<<"\nKet qua duyet truoc tu trai qua phai:\n";
        DuyetTruoc(test);
        cout<<"\nKet qua duyet giua tu trai qua phai:\n";
        DuyetGiua(test);
        cout<<"\nKet qua duyet sau tu trai qua phai:\n";
        DuyetSau(test);
        cout<<"\nNhap mot gia tri ma ban muon tim kiem: "; cin>>x;
        t=Search(test, x);
        if(t!=NULL) cout<<"Tim thay "<<x<<" co trong cay";
        else cout<<"khong tim thay "<<x<<" trong cay";
        cout<<"\nNhap mot gia tri ma ban muon xoa: "; cin>>x;
        DelNode(test, x);
        cout<<"Ket qua duyet truoc tu trai qua phai sau khi xoa nut "<<x<<endl;
        DuyetTruoc(test);
}
```

Kết quả chạy thử chương trình khi nhập thử dãy số: *12, 4, 23, 2, 9, 15, 25, 1, 3, 7, 11, 8.*

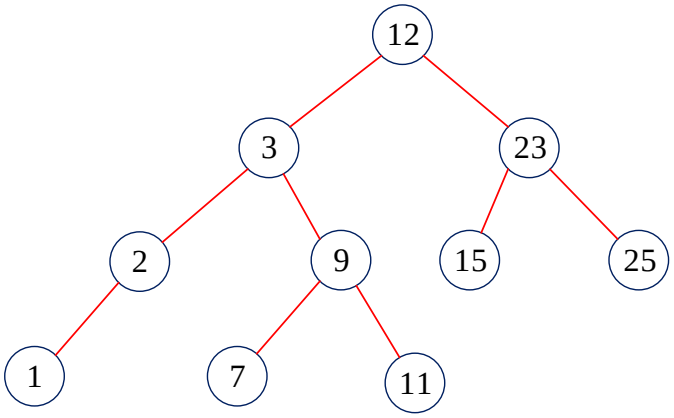
```
Nhap so luong nut trong cay: 12
Nhap vao cac nut cho cay:
12      4       23      2       9       15
25      1       3       7       11      8

Ket qua duyet truoc tu trai qua phai:
12  4  2  1  3  9  7  8  11  23  15  25
Ket qua duyet giua tu trai qua phai:
1  2  3  4  7  8  9  11  12  15  23  25
Ket qua duyet sau tu trai qua phai:
1  3  2  8  7  11  9  4  15  25  23  12
Nhap mot gia tri ma ban muon tim kiem: 9
Tim thay 9 co trong cay
Nhap mot gia tri ma ban muon xoa: 4
Ket qua duyet truoc tu trai qua phai sau khi xoa nut 4
12  7  2  1  3  9  8  11  23  15  25
```



Hình 31. Cây minh họa cho ví dụ 25.

5.4 Cây nhị phân tìm kiếm cân bằng

5.4.1 Tổng quan về cây nhị phân tìm kiếm cân bằng



Hình 32. Cây nhị phân tìm kiếm cân bằng.

Định nghĩa:

- *Cây nhị phân tìm kiếm cân bằng là cây mà tại mỗi nút của nó, độ cao của cây con trái và của cây con phải chênh lệch nhau không quá một.*
- *Độ lệch giữa cây trái và cây phải của một nút được gọi là chỉ số cân bằng.*
- *Các trường hợp hợp lệ của chỉ số cân bằng (CSCB):*
  - *CSCB = 1: Độ cao cây trái nhỏ hơn độ cao cây phải.*
  - *CSCB = 0: Độ cao cây trái bằng độ cao cây phải.*
  - *CSCB = -1: Độ cao cây trái lớn hơn độ cao cây phải.*

Cấu trúc dữ liệu của cây nhị phân tìm kiếm cân bằng:

```
#define LH -1 //Cây con trái cao hơn
#define EH 0 //Cây con trái bằng cây con phải
#define RH 1 //Cây con phải cao hơn
struct Node
{
        int     BalanceFactor; //chỉ số cân bằng
        int     key;
        Node *Left;
        Node *Right;
};
typedef Node *Tree;
```

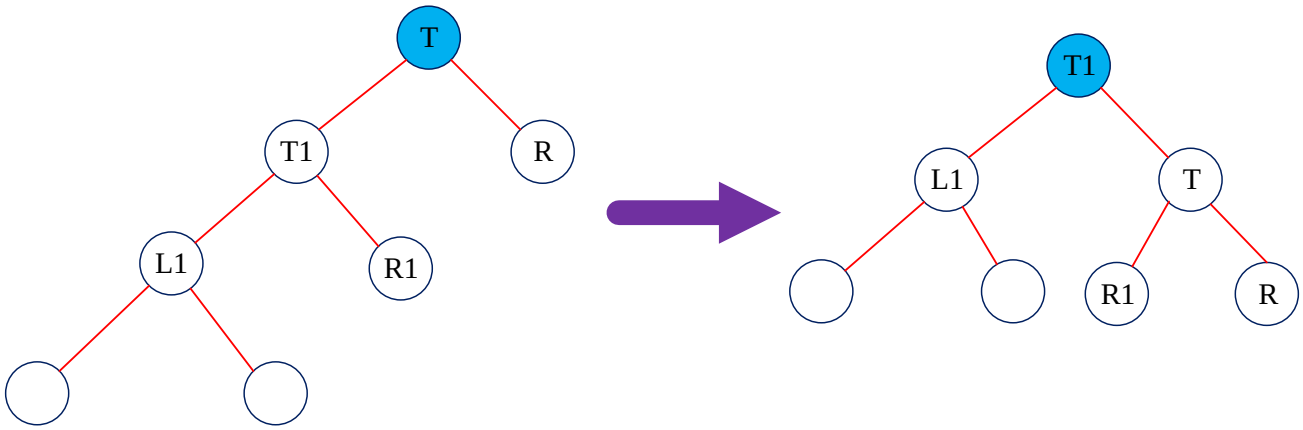
5.4.2 Các thao tác trên cây cân bằng

Khi thêm hay xoá 1 nút trên cây, có thể làm cho cây mất tính cân bằng, khi ấy ta phải tiến hành cân bằng lại. Cây có khả năng mất cân bằng khi thay đổi chiều cao:

- *Thêm bên trái -> lệch nhánh trái*
- *Thêm bên phải -> lệch nhánh phải*
- *Hủy bên phải -> lệch nhánh trái*
- *Hủy bên trái -> lệch nhánh phải*

Để cân bằng lại cây ta tìm cách bố trí lại cây sao cho chiều cao 2 cây con cân đối bằng cách kéo nhánh cao bù cho nhánh thấp sao cho bảo đảm cây sau đó vẫn là cây nhị phân tìm kiếm.

Cân bằng cho cây trong trường hợp 1 (Left – Left: Cây mất cân bằng trái – trái tại nút T):

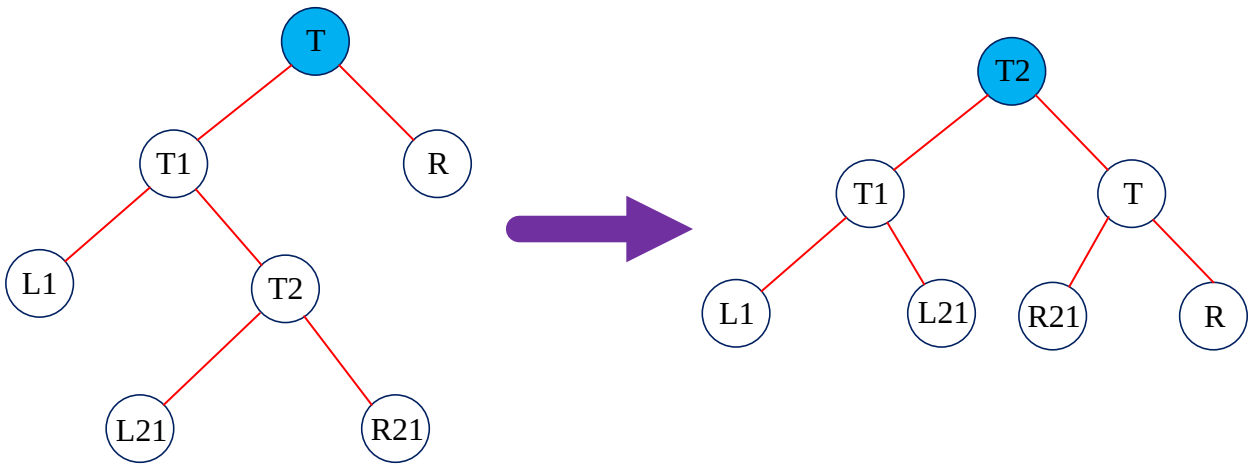


Hình 33. Cân bằng cây trường hợp Left – Left.

Hàm cân bằng cho trường hợp Left - Left:

```
void Lech_trai_trai(Tree &T)
{
        Node *T1 = T->Left;
        T->Left = T1->Right;
        T1->Right=T;
        switch(T1->BalanceFactor)
        {
                case LH:
                        T->BalanceFactor =EH;
                        T1->BalanceFactor=EH;
                        break;
                case EH:
                        T->BalanceFactor =EH;
                        T1->BalanceFactor=RH;
                        break;
        }
        T=T1;
}
```

Cân bằng cho cây trong trường hợp 2 (Left – Right: Cây mất cân bằng trái – phải tại nút T):

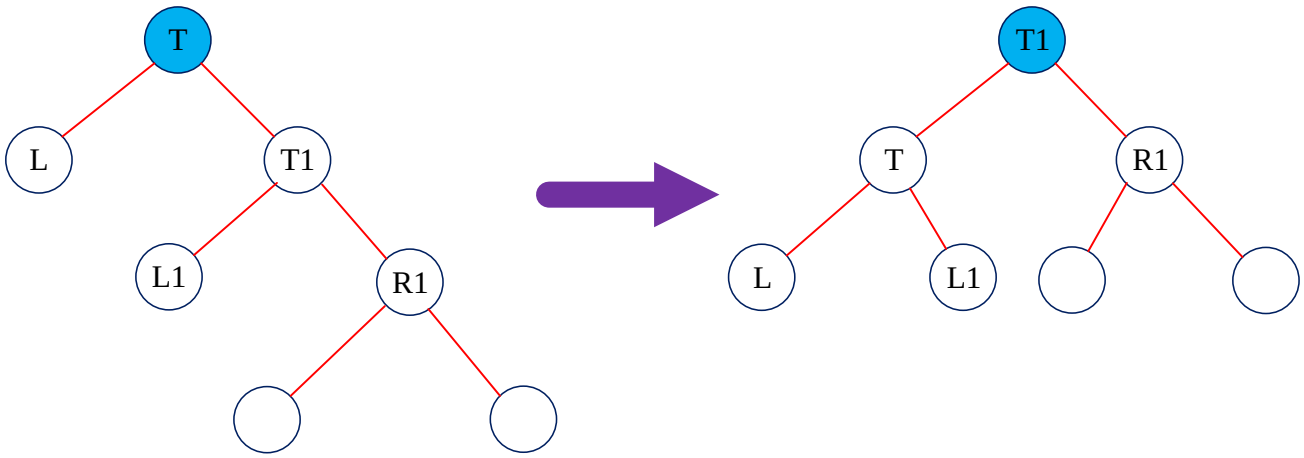


Hình 34. Cân bằng cây trường hợp Left – Right.

Hàm cân bằng cho trường hợp Left - Right:

```
void Lech_trai_phai(Tree &T)
{
	Node *T1=T->Left, *T2=T1->Right;
	T->Left=T2->Right;
	T2->Right=T;
	T1->Right= T2->Left;
	T2->Left = T1;
	switch(T2->BalanceFactor)
	{
		case LH:
			T->BalanceFactor = EH;
			T1->BalanceFactor= RH;
			break;
		case EH:
			T->BalanceFactor = EH;
			T1->BalanceFactor= EH;
			break;
		case RH:
			T->BalanceFactor = LH;
			T1->BalanceFactor= EH;
			break;
	}
	T2->BalanceFactor = EH;
	T=T2;
}
```

Cân bằng cho cây trong trường hợp 3 (Right – Right: Cây mất cân bằng phải – phải tại nút T):



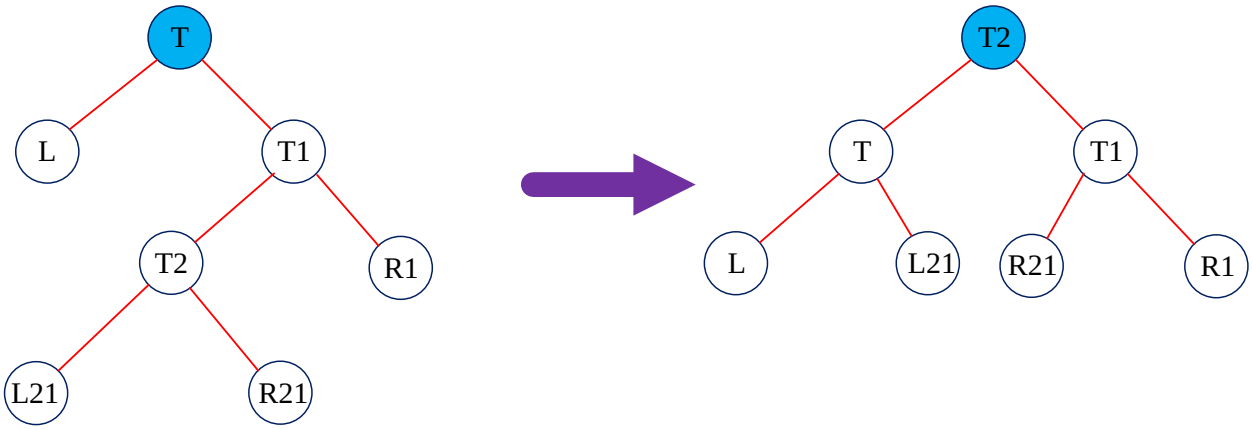
Hình 35. Cân bằng cây trường hợp Right – Right.

Hàm cân bằng cho trường hợp Right - Right:

```
void Lech_phai_phai(Tree &T)
{
	Node  *T1= T->Right;
	T->Right=T1->Left;
	T1->Left=T;
```

```
        switch(T1-> BalanceFactor)
        {
                case RH:
                        T-> BalanceFactor = EH;
                        T-> BalanceFactor = EH;
                        break;
                case EH:
                        T-> BalanceFactor = EH;
                        T1->BalanceFactor= LH;
                        break;
        }
        T=T1;
}
```

Cân bằng cho cây trong trường hợp 4 (Right – Left: Cây mất cân bằng phải – trái tại nút T):



Hình 36. Cân bằng cây trường hợp Right – Left.

Hàm cân bằng cho trường hợp Right - Left:

```
void Lech_phai_trai(Tree &T)
{       Node *T1= T->Right, *T2=T1->Left;
        T->Right = T2->Left;
        T2->Left = T;
        T1->Left = T2->Right;
        T2->Right = T1;
        switch(T2-> BalanceFactor)
        {
                case RH:
                        T-> BalanceFactor = EH;
                        T1-> BalanceFactor = LH; break;
                case EH:
                        T-> BalanceFactor = EH;
                        T1-> BalanceFactor = EH; break;
                case LH:
                        T-> BalanceFactor = RH;
                        T1-> BalanceFactor = EH; break;
        }
        T2-> BalanceFactor =EH; T=T2;
}
```

Thêm nút vào cây nhị phân tìm kiếm cân bằng:

- *Thêm bình thường như trường hợp cây NPTK.*
- *Nếu cây tăng trưởng chiều cao:*
  - *Lần ngược về gốc để phát hiện nút bị mất cân bằng.*
  - *Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp.*
- *Việc cân bằng lại chỉ cần thực hiện 1 lần nơi mất cân bằng.*

Hủy nút trên cây nhị phân tìm kiếm cân bằng:

- *Hủy bình thường như trường hợp cây NPTK.*
- *Nếu cây giảm chiều cao:*
  - *Lần ngược về gốc để phát hiện nút bị mất cân bằng.*
  - *Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp.*
  - *Tiếp tục lần ngược lên nút cha….*
- *Việc cân bằng lại có thể lan truyền lên tận gốc.*

BÀI TẬP CHƯƠNG 5

Bài 1. Vẽ hình cây nhị phân tìm kiếm tạo ra từ cây rỗng bằng cách lần lượt thêm vào các khoá là các số nguyên: 54, 31, 43, 29, 65, 10, 20, 36, 78, 59.

a) Cho biết dãy kết quả khi duyệt qua cây trên theo các cách LNR, RNL, LRN, RLN, NLR, NRL.
b) Vẽ lại cây trên khi xóa nút có khóa bằng 31.
c) Cân bằng lại cây trên để được cây NPTK cân bằng.
d) Viết chương trình minh họa cho bài tập.

Chương 6. BẢNG BĂM

6.1 Tổng quan về bảng băm

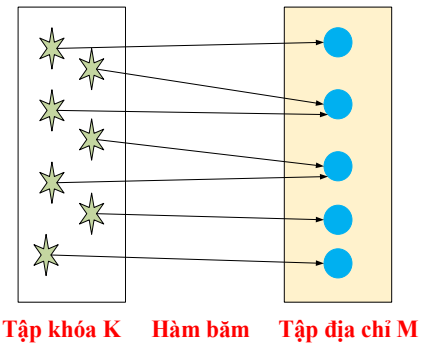
6.1.1 Một số khái niệm

Bảng băm là 1 CTDL tương tự như mảng nhưng kèm theo một hàm băm để ánh xạ nhiều giá trị vào cùng một phần tử trong mảng.

Hàm băm hay phép băm (hash function) là hàm được dùng để ánh xạ (hoặc biến đổi) giá trị của khóa vào một dãy các địa chỉ của bảng băm.

- *Tư tưởng của phép băm là dựa vào giá trị các khóa k[1..n], chia các khóa đó thành các nhóm.*
- *Những khóa cùng một nhóm sẽ có cùng một đặc điểm chung và đặc điểm này không có trong nhóm khác.*
- *Khi có một khóa tìm kiếm X, ta xác định xem nếu X có trong dãy khóa thì nó thuộc nhóm nào và ta tiến hành tìm trên nhóm đó.*

Khóa có thể là dạng số hay số dạng chuỗi. Giả sử có 2 khóa phân biệt ki và kj nếu h(ki) = h(kj) thì ta nói hàm băm bị đụng độ (với h(x) là hàm băm).



Hình 37. Minh họa bảng băm

6.1.2 Ưu điểm của bảng băm

- *Bảng băm là một cấu trúc dung hòa giữa thời gian truy xuất và dung lượng bộ nhớ:*
  - *Nếu không có sự giới hạn về bộ nhớ thì chúng ta có thể xây dựng bảng băm với mỗi khóa ứng với một địa chỉ với mong muốn thời gian truy xuất tức thời.*
  - *Nếu dung lượng bộ nhớ có giới hạn thì tổ chức một số khóa có cùng địa chỉ, khi đó tốc độ truy xuất sẽ giảm.*
- *Bảng băm được ứng dụng nhiều trong thực tế, rất thích hợp khi tổ chức dữ liệu có kích thước lớn và được lưu trữ ở bộ nhớ ngoài.*

6.1.3 Các phép toán trên bảng băm

- *Khởi tạo bảng băm.*
- *Kiểm tra bảng băm rỗng.*
- *Lấy kích thước (số phần tử) của bảng băm.*
- *Tìm kiếm một phần tử trong bảng băm theo khoá k chỉ định trước.*
- *Thêm một phần tử vào bảng băm.*
- *Loại bỏ một phần tử ra khỏi bảng băm.*
- *Tạo một bảng băm mới từ một bảng băm cũ đã có.*
- *Xử lý các khóa trong bảng băm: xử lý toàn bộ khóa trong bảng băm theo thứ tự địa chỉ từ nhỏ đến lớn.*

6.2 Phương pháp xây dựng hàm băm

Một hàm băm tốt phải thỏa mãn các điều kiện sau:

- *Tính toán nhanh.*
- *Các khoá được phân bố đều trong bảng.*
- *Ít xảy ra đụng độ.*
- *Xử lý được các loại khóa có kiểu dữ liệu khác nhau*

Một số phương pháp xây hựng hàm băm:

- *Hàm băm dạng bảng tra*
- *Phương pháp chia.*
- *Phương pháp nhân.*
- *Phép băm phổ quát (universal hashing).*

6.2.1 Hàm băm dạng bảng tra

Hàm băm có thể tổ chức ở dạng bảng tra (còn gọi là bảng truy xuất) hoặc thông dụng nhất là ở dạng công thức.

Ví dụ: Bảng tra với khóa là bộ chữ cái, bảng băm có 26 địa chỉ từ 0 đến 25. Khóa a ứng với địa chỉ 0, khoá b ứng với địa chỉ 1,… , z ứng với địa chỉ 25.

6.2.2 Hàm băm theo phương pháp chia

Hàm băm được xác định như sau:

$$h(k) = k \bmod m$$

Trong đó:

- *m là kích thước của bảng – thông thường ta chọn m là số nguyên tố.*
- *k là khóa*
- *Giá trị của h(k) sẽ là: 0, 1, 2, 3,…, m-1.*

6.2.3 Hàm băm theo phương pháp nhân

Hàm băm được xác định như sau:

$$h(k) = \lfloor m*(k*A \bmod 1) \rfloor$$

Trong đó:

- *m là kích thước của bảng – thông thường ta chọn m = 2^n*
- *k là khóa*
- *A là hằng số và 0<A<1.*
- *Theo Knuth thì chọn A bằng giá trị sau:*

$$A = (\sqrt{5} - 1)/2 = 0.6180339887\ldots$$

6.2.4 Phép băm phổ quát

Cho H là một tập hợp hữu hạn các hàm băm: ánh xạ các khóa k từ tập khóa U vào miền giá trị {0, 1, 2, …, m-1}.

Tập H là phổ quát (universal) nếu với mọi f thuộc H và 2 khoá phân biệt k1, k2 ta có xác suất:

$$Pr\{f(k1) = f(k2)\} <= 1/m.$$

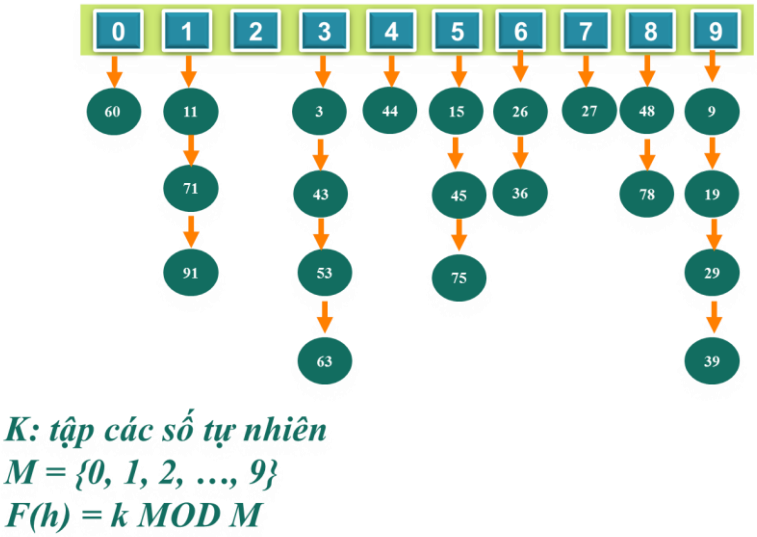
Trong đó: m là kích thước bảng.

6.3 Các phương pháp giải quyết đụng độ

6.3.1 Phương pháp kết nối trực tiếp

Bảng băm được cài đặt bằng các danh sách liên kết, các phần tử trên bảng băm được "băm" thành M danh sách liên kết (từ danh sách 0 đến danh sách M–1).

Các phần tử bị xung đột tại địa chỉ i được kết nối trực tiếp với nhau qua danh sách liên kết i.



Hình 38. Bảng băm theo phương pháp kết nối trực tiếp

6.3.2 Phương pháp kết nối hợp nhất

Bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có M phần tử. Các phần tử bị xung đột tại một địa chỉ được kết nối với nhau qua một danh sách liên kết.

6.3.3 Phương pháp dò tuần tự

Bảng băm trong trường hợp này được cài đặt bằng danh sách kề có M phần tử, mỗi phần tử của bảng băm là một mẫu tin có một trường key để chứa khoá của phần tử (ban đầu khởi tạo bằng -1):

- Khi thêm phần tử có khoá key vào bảng băm, hàm băm h(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1:
  - *Nếu chưa bị xung đột thì thêm phần tử mới vào địa chỉ này.*
  - *Nếu bị xung đột thì hàm băm lại lần 1, hàm h1 sẽ xét địa chỉ kế tiếp, nếu lại bị xung đột thì hàm băm lại lần 2, hàm h2 sẽ xét địa chỉ kế tiếp nữa, …, và quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm phần tử mới vào địa chỉ này.*
- Khi tìm một phần tử có khoá key trong bảng băm, hàm băm h(key) sẽ xác định địa chỉ i trong khoảng từ 0 đến M-1, tìm phần tử khoá key trong bảng băm xuất phát từ địa chỉ i.
- Hàm băm lại lần i được biểu diễn bằng công thức sau:

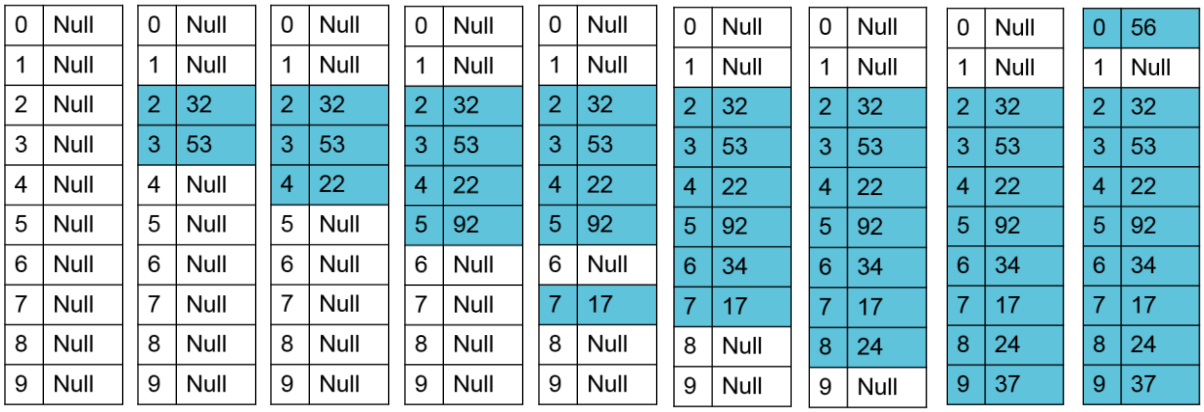
$$f(key) = (f(key)+i) \%M$$

  - Trong đó f(key) là hàm băm chính của bảng băm.
- **Lưu ý**: địa chỉ dò tìm kế tiếp là địa chỉ 0 nếu đã dò đến cuối bảng.

Ví dụ 26: Giả sử, khảo sát bảng băm có cấu trúc như sau:

- *Tập khóa K: tập số tự nhiên*
- *Tập địa chỉ M: gồm 10 địa chỉ (M={0, 1, …, 9}*
- *Hàm băm h(key) = key % 10.*

Hình sau thể hiện các bước thêm các nút 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm.



Hình 39. Thêm nút vào bảng băm theo phương pháp dò tuần tự

6.3.4 Phương pháp dò bậc hai

Tương tự dò tuần tự nhưng hàm băm lại lần thứ i được biểu diễn bằng công thức sau:

$$fi(key) = (f(key)+i^2) \%M$$

Trong đó: f(key) là hàm băm chính của bảng băm.

Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.

6.3.5 Phương pháp băm kép

Phương pháp băm kép dùng hai hàm băm bất kì, ví dụ chọn hai hàm băm như sau:

$$h1(key)= key \%M.$$

$$h2(key) =(M-2)-key \%(M-2).$$

- Khi thêm phần tử có khoá key vào bảng băm, thì i=h1(key) và j=h2(key) sẽ xác định địa chỉ i và j trong khoảng từ 0 đến M-1:
  - *Nếu chưa bị xung đột thì thêm phần tử mới tại địa chỉ i này.*
  - *Nếu bị xung đột thì hàm băm lại lần 1 h1 sẽ xét địa chỉ mới i+j, nếu lại bị xung đột thì hàm băm lại lần 2 h2 sẽ xét địa chỉ i+2j, …, quá trình cứ thế cho đến khi nào tìm được địa chỉ trống và thêm phần tử vào địa chi này.*
- Khi tìm kiếm một phần tử có khoá key trong bảng băm, hàm băm i=h1(key) và j=h2(key) sẽ xác định địa chỉ i và j trong khoảng từ 0 đến M-1. Xét phần tử tại địa chỉ i, nếu chưa tìm thấy thì xét tiếp phần tử i+j, i+2j, …, quá trình cứ thế cho đến khi nào tìm được khoá (trường hợp tìm thấy) hoặc bị rơi vào địa chỉ trống (trường hợp không tìm thấy).
- Bảng băm dùng hai hàm băm khác nhau, hàm băm lại của phương pháp băm kép được tính theo hai giá trị: i (kết quả hàm băm thứ nhất) và j (kết qủa hàm băm thứ hai) theo một công thức bất kì. Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.

TÀI LIỆU THAM KHẢO

[1]. Nguyễn Văn Linh, Trần Cao Đệ, Trương Thị Thanh Tuyền, Lâm Hoài Bảo, Phan Huy Cường, Trần Ngân Bình - "**Cấu trúc dữ liệu**" – Đại Học Cần Thơ 2003.

[2]. Dương Trần Đức – "Cấu trúc dữ liệu và giải thuật" – Học viên công nghệ bưu chính viễn thông – Hà Nội 2007.

[3]. Lê Minh Hoàng – "Giải thuật và lập trình" – Đại học sư phạm Hà Nội 1999-2004.

MỤC LỤC