ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG CAO ĐẲNG LÝ TỰ TRỌNG THÀNH PHỐ HỒ CHÍ MINH

GIÁO TRÌNH

MÔN HỌC/MÔ ĐUN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

NGÀNH/NGHỀ: LẬP TRÌNH MÁY TÍNH, THIẾT KẾ TRANG WEB

TRÌNH ĐỘ: CAO ĐẲNG

Ban hành kèm theo Quyết định số: /QĐ-LTT-ĐT ngày tháng năm 2021
của Hiệu trưởng Trường Cao đẳng Lý Tự Trọng Thành phố Hồ Chí Minh

LƯU HÀNH NỘI BỘ

LỜI GIỚI THIỆU

Giáo trình môn Cấu trúc dữ liệu và giải thuật được biên soạn dựa theo chương trình chi tiết học phần Cấu trúc dữ liệu và giải thuật của hệ Cao đẳng chuyên ngành Lập trình máy tính và Thiết kế web thuộc khoa Công nghệ Thông tin.

Giáo trình cung cấp một số giải thuật sắp xếp, tìm kiếm trên danh sách và các thao tác trên cây nhị phân, cây nhị phân tìm kiếm. Qua đó giúp HS-SV biết cách chọn lựa cấu trúc dữ liệu phù hợp cũng như cách lựa chọn giải thuật và tối ưu giải thuật cho các bài toán thực tế.

Giáo trình là tài liệu giảng dạy và học tập chính thức cho HS-SV khoa Công nghệ Thông tin trường CĐ Lý Tự Trọng TP. HCM.

Thành phố Hồ Chí Minh, ngày 01 tháng 08 năm 2020

Giảng viên biên soạn: Nguyễn Vân Giang

MỤC LỤC

Chương 1: MỞ ĐẦU

Mỗi bài toán thực tế đều bao gồm nhiều đối tượng dữ liệu và các yêu cầu cần xử lý trên những đối tượng đó. Để dùng máy tính giải quyết các bài toán thực tế một cách hiệu quả cần chú ý đến hai vấn đề là cấu trúc dữ liệu và giải thuật.

I. CẤU TRÚC DỮ LIỆU (DATA STRUCTURE)

Cấu trúc dữ liệu của bài toán là cách thức tổ chức dữ liệu sao cho phản ánh đúng thực tế bài toán. Cấu trúc dữ liệu là tập hợp các biến và các kiểu dữ liệu được liên kết với nhau.

Ví dụ: Trong bài toán quản lý sinh viên thì cách khai báo các biến biểu diễn các dữ liệu như họ tên, ngày sinh, điểm,…là cấu trúc dữ liệu của bài toán.

Một cấu trúc dữ liệu tốt phải thỏa mãn các tiêu chuẩn sau:

- **Phản ánh đúng thực tế**

Đây là tiêu chuẩn trọng nhất, quyết định tính đúng đắn của toàn bộ bài toán

Ví dụ: Khi cần lưu trữ mã sinh viên mà chọn biến kiểu số nguyên thì sẽ gặp trục trặc khi sử dụng chương trình nếu trường hợp mã sinh viên có ký thự chữ. Ở đây ta dùng chuỗi ký tự thì hợp lý hơn

- **Phù hợp với các thao tác xử lý trên dữ liệu đó**

Ví dụ: Nếu các thao tác xóa phần tử trong danh sách hay chèn phần tử vào một danh sách được thực hiện thường xuyên thì dùng danh sách liên kết phù hợp hơn danh sách kề (mảng).

- **Tiết kiệm tài nguyên của hệ thống (thời gian,bộ nhớ)**

Tùy thuộc vào trường hợp cụ thể, ta quan tâm đến thời gian xử lý hơn hay việc tiết kiệm bộ nhớ hơn mà ta chọn cấu trúc dữ liệu phù hợp vì hai yếu tố này thường mâu thuẫn với nhau.

Ví dụ: Khi cần sắp xếp dữ liệu, nếu ta dùng các biến tạm thì sẽ tốn bộ nhớ hơn nhưng thời gian xử lý lại nhanh hơn.

II. GIẢI THUẬT (ALGORITHMS)

Giải thuật là trình tự các thao tác xử lý dữ liệu để giải quyết bài toán.

Cách biểu diễn giải thuật, cách tính độ phức tạp của giải thuật đ được trình bày trong giáo trình môn *Toán ứng dụng cho tin học.*

III. MỐI QUAN HỆ GIỮA CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Theo Niklaus Wirth , một nhà khoa học máy tính nổi tiếng và là tác giả của ngôn ngữ lập trình Pascal:

Data structures + Algorithms = Programs

Cấu trúc dữ liệu và giải thuật có mối quan hệ chặt chẽ với nhau. Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý của giải thuật lại chính là dữ liệu. Để xác định được giải thuật phù hợp cần phải biết nó tác động đến những loại dữ liệu nào và khi chọn lựa một cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ được tác động lên nó.

Ví dụ: Viết chương trình nhập vào một số tự nhiên N (tối đa 9 chữ số). Hãy xuất N theo chiều đảo ngược. (chẳng hạn nếu INPUT là 1443493 thì OUTPUT là 3943441).

Bài toán này chỉ yêu cầu, nhập N và xuất N theo chiều đảo ngược. Ta có thể giải bài toán này theo hai cách sau đây:

Giải thuật 1: Nếu dữ liệu của N được tổ chức theo biến kiểu unsigned long thì đoạn xử lý chính được viết như sau:

```
while (n!=0)
{
  cout <<n%10;
  n=n/10;
}
```

Giải thuật 2: Nếu dữ liệu của N được tổ chức theo biến kiểu chuỗi. Thì ta có thể sử dụng ngay hàm strrev (hàm đảo ngược chuỗi của C).

Rõ ràng giải thuật 2 thuận lợi hơn so với giải thuật 1. Tuy nhiên nếu yêu cầu của bài toán thay đổi - chẳng hạn cần tính tổng các chữ số của số N thì giải thuật 1 lại tỏ ra có ưu thế hơn so với giải thuật 2.

Chương 2: TÌM KIẾM VÀ SẮP XẾP

2.1. TÌM KIẾM (SEARCHING)

Tìm kiếm thông tin là một công việc rất thường gặp trong thực tế và cũng là trong những ứng dụng quan trọng trong máy tính. Chẳng hạn ta đưa vào tên của một người và yêu cầu máy tính đưa ra danh sách các số điện thoại tương ứng. Hoặc ta đưa vào họ tên hay mã số của một người trong cơ quan và yêu cầu máy tính trích ra các thông tin cá nhân liên quan đến người đó.

Trong bài này, chúng ta sẽ tìm hiểu một số giải thuật tìm kiếm dữ liệu cơ bản và thường được sử dụng. Các giải thuật tìm kiếm dữ liệu được chia thành hai nhóm:

- Tìm kiếm nội (Internal Searching): Khi số lượng các dữ liệu được tìm kiếm nhỏ và có thể chứa ở bộ nhớ trong của máy tính (RAM).
- Tìm kiếm ngoại (External Searching): Khi số lượng dữ liệu được tìm kiếm lớn, phải lưu ở bộ nhớ ngòai (disk/tape).

Tuy nhiên trong chương này, chúng ta chỉ bàn đến các giải thuật tìm kiếm nội.

2.1.1. Bài toán tìm kiếm

INPUT:

Dãy (a) gồm n phần tử $a_0,a_2,...,a_{n-1}$.

x là giá trị của phần tử cần tìm

OUTPUT:

k ($0 \le k \le n-1$) nếu $a_k = x$, k là vị trí xuất hiện của x trong dãy (a)

- 1 nếu trong dãy không có phần tử nào bằng x

Để đơn giản, trong chương này ta xét dãy (a) gồm các số nguyên, x cần tìm cũng là số nguyên và dùng mảng để biểu diễn dãy (a).

2.1.2. Tìm tuyến tính (Linear search)

2.1.2.1. Ý tưởng của giải thuật

Lần lượt so sánh x với các a[i] (i=0,1,2,…,n-1).

Nếu có a[i] = x thì dừng và trả về i

Nếu không có a[i] nào bằng x thì trả về -1.

Ví dụ 1 : Cho dãy số (a)

i	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7(n-1)*
a[i]	**12**	**2**	**8**	**5**	**1**	**6**	**8**	**15**

Nếu x=8 thì kết quả trả về là 2

Nếu x=7 thì kết quả trả về là -1

Lưu ý : Giải thuật tìm tuyến tính luôn trả về vị trí xuất hiện đầu tin của x trong dãy số.

2.1.2.2. Cài đặt

```
int linearSearch (int a[], int n, int x)
{
  int i = 0;
  while ((i<n) && (a[i]!=x))
    i++;
  if( i==n)return -1;
  return i; }
```

2.2.2.3. Thuật toán cải tiến

Nếu đặt thêm phần tử cầm canh (sentinel) ở cuối mảng (cho a[n]=x) thì trong mảng luơn có phần tử bằng x nên vòng lặp while luôn kết thúc mà không cần kiểm tra điều kiện i<n.

```
int LinearSearch (int a[], int n, int x)
```

```
{
  int i = 0;a[n]=x;
  while (a[i]!=x)
    i++;
  if( i==n)return -1;
  return i;
}
```

Đánh giá giải thuật:

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	n + 1	Không có x trong mảng
Trung bình	(n+1)/2	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

Độ phức tạp của giải thuật là :T(n)= O(n)

2.3. Tìm nhị phân (Binary search)

2.3.1. Ý tưởng của giải thuật

Trong trường hợp dãy (a) đ được sắp thứ tự tăng dần thì ta có thể tìm nhanh phần tử có khóa x bằng phương pháp tìm nhị phân như sau, giống như cách tra từ trong từ điển:

Giả sử dãy tìm kiếm hiện thời là : a[l], a[l+1],…,a[r] ($l \le r$)

Gọi m=(l+r)/2 và so sánh a[m] với x :

Nếu a[m]<x thì x chỉ có thể xuất hiện ở bên phải a[m], phạm vi tìm kiếm được thu hẹp từ a[m+1],…,a[r] . Thực hiện lại giải thuật với l=m+1

Nếu a[m]>x thì x chỉ có thể xuất hiện ở bên trái a[m], phạm vi tìm kiếm được thu hẹp từ a[l],…,a[m-1] . Thực hiện lại giải thuật với r=m-1

Giải thuật dừng khi l> r và trả về -1

Ví dụ 2: Cho dãy số (a)

i	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7(n-1)*
a[i]	**2**	**3**	**5**	**7**	**8**	**8**	**10**	**15**

Nếu cần tìm x=10 :

- l = 0, r = 7: m = 3, a[3] = 7 <x
- l = 4, r = 7: m = 5, a[5] = 8 < x
- l = 6, r = 7: m = 6, a[6] = 10 = x (tìm thấy). Giải thuật dừng và trả về 6.

Nếu cần tìm x=6 :

- l = 0, r = 7: m = 3, a[3] = 7 > x
- l = 0, r = 2: m = 1, a[1] = 3 < x .
- l = 2, r = 2: m = 2, a[2] = 5 < x .
- l = 3, r = 2: giải thuật dừng vì l > r và trả về -1

2.3.2. Cài đặt:

```
int binarySearch(int a[],int n,int x)
{
  int l=0,r=n-1,m;
  do
  {
    m=(l+r)/2;
    if (x==a[m])return m;
    else
      if (x<a[m]) r=m-1;
      else l=m+1;
```

```
    }
    while (l<=r);
    return -1;
}
```

Đánh giá giải thuật:

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 n/2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

Độ phức tạp của giải thuật là :$T(n)= O(\log_2 n)$

2.2. SẮP XẾP (SORTING)

Sắp xếp cũng là một công việc thường gặp, chẳng hạn như sắp xếp theo thứ tự alphabet cột họ tên trong danh sách lớp hoặc trong danh bạ điện thoại, ….Phần lớn, mục đích của bài toán sắp xếp thường là để trợ giúp phép tìm kiếm được nhanh chóng.

Trong bài này, chúng ta sẽ tìm hiểu một số giải thuật sắp xếp cơ bản và thường được sử dụng. Các giải thuật sắp xếp được chia thành hai nhóm:

- Sắp xếp nội (Internal Sorting): Khi số lượng các dữ liệu được sắp xếp nhỏ và có thể chứa ở bộ nhớ trong của máy tính (RAM).
- Sắp xếp ngoại (External Sorting): Khi số lượng dữ liệu được sắp xếp lớn, phải lưu ở bộ nhớ ngòai (disk/tape).

Tuy nhiên trong chương này, chúng ta chỉ bàn đến các giải thuật sắp xếp nội.

2.2.1. Bài toán sắp xếp

Các đối tượng cần được sắp xếp là các mẩu tin gồm một hoặc nhiều trường. Một trong các trường được gọi là khóa (key), kiểu của nó là một kiểu có quan hệ thứ tự (như các kiểu số nguyên, số thực, chuỗi ký tự...). Mục đích của việc sắp xếp là tổ chức lại các mẩu tin sao cho các khóa của chúng được sắp thứ tự tăng dần hay giảm dần.

Trong phần này chúng ta giả sử cần sắp xếp dãy (a) gồm n số nguyên theo thứ tự tăng dần. Để đơn giản, ta sẽ dùng mảng để biểu diễn dãy (a).

2.2.2. Phương pháp đổi chỗ trực tiếp (Interchange Sort)

2.2.2.1. Ví dụ minh họa

Ví dụ 1: Sắp xếp dãy số sau theo thứ tự tăng dần.

Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	12	2	8	5	1	6	4	10
0	2	12	8	5	1	6	4	10
1	1	12	8	5	2	6	4	10
1	1	8	12	5	2	6	4	10
1	1	5	12	8	2	6	4	10
2	1	2	12	8	5	6	4	10
2	1	2	8	12	5	6	4	10
2	1	2	5	12	8	6	4	10
3	1	2	4	12	8	6	5	10
3	1	2	4	8	12	6	5	10
3	1	2	4	6	12	8	5	10
4	1	2	4	5	12	8	6	10
4	1	2	4	5	8	12	6	10
5	1	2	4	5	6	12	8	10
6	1	2	4	5	6	8	12	10
KQ	1	2	4	5	6	8	10	12

Nếu bỏ qua các thao tác trung gian trong mỗi bước ta sẽ được kết quả ở từng bước như sau:

Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]

0	12	2	8	5	1	6	4	10
1	1	12	8	5	2	6	4	10
2		2	12	8	5	6	4	10
3			4	12	8	6	5	10
4				5	12	8	6	10
5					6	12	8	10
6						8	12	10
KQ	1	2	4	5	6	8	10	12

- Phương pháp này được gọi là phương pháp đổi chỗ trực tiếp, vì ở bước i ta đổi chỗ trực tiếp a[i] với lần lượt từng phần tử trong dãy hiện thời nếu phần tử này nhỏ hơn nó.

2.2.2.2. Ý tưởng của giải thuật

Giải thuật gồm n-1 bước (từ bước i= 0 đến bước i= n-2).

Ở bước 0, ta phải sắp xếp sao cho a[0] sẽ là phần tử nhỏ nhất dãy a[0],…,a[n-1].

Ở bước 1, ta phải sắp xếp sao cho a[1] sẽ là phần tử nhỏ nhất dãy a[1],…,a[n-1]

Tổng quát:

Ở bước i, ta phải sắp xếp sao cho a[i] sẽ là phần tử nhỏ nhất dãy a[i],…,a[n-1], bằng cách : so sánh a[i] với các a[j] (i<j≤n-1), nếu thấy a[i]>a[j] thì hóan vị a[i] với a[j].

2.2.2.3. Cài đặt

```
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[i]>a[j])
        swap(a[i],a[j]);
}
```

Trong đó hàm swap() là hàm hoán vị 2 số trong thư viện <iostream>

Đánh giá giải thuật:

Số lượng các phép so sánh không phụ thuộc vào tình trạng của dãy số ban đầu, ở bước i luôn phải so sánh (n – i -1) lần .

Số lần so sánh = $\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}$

Số lượng phép hoán vị lại thuộc vào kết quả so sánh:

- Trường hợp tốt nhất (dãy có thứ tự đúng) không cần hoán vị.

- Trường hợp xấu nhất (dãy có thứ tự ngược) mỗi lần so sánh là một lần hoán vị.

Trường hợp	Số lần so sánh	Số lần hoán vị
Tốt nhất	n(n-1)/2	0
Xấu nhất	n(n-1)/2	n(n-1)/2

Độ phức tạp của giải thuật là : $O(n^2)$

2.2.3. Phương pháp sắp xếp bằng cách phân hoạch (Quick Sort)

2.2.3.1. Ý tưởng của giải thuật

Giả sử ta cần sắp tăng dần dãy a[l],...,a[r]

Bước 1: Chọn x=a[(l+r)/2] làm mốc, i=l, j=r

Bước 2:

Trong khi a[i]<x thì tăng i lên 1 đơn vị

Trong khi a[j]>x thì giảm j xuống 1 đơn vị

Nếu i<=j thì hóan vị a[i], a[j] rồi tăng i, giảm j

Nếu i<=j thì quay lại bước 2, ngược lại qua bước 3

Bước 3:

Nếu l<j (dãy a[l],...,a[j] có từ 2 phần tử trở lên) thì thực hiện lại giải thuật với r=j

Nếu i<r (dãy a[i],...,a[r] có từ 2 phần tử trở lên) thì thực hiện lại giải thuật với l=i

2.2.3.2. Ví dụ minh họa

Ví dụ 5: Sắp xếp dãy số : 2, 9, 5, 4, 7, 6, 8 theo thứ tự tăng dần.

	0	*1*	*2*	*3*	*4*	*5*	*6*
l=0, r=6, x=4	i **2**	i **9**	 **5**	j **(4)**	j **7**	j **6**	j **8**
	 2	j **(4)**	i j **5**	 **9**	j **7**	j **6**	j **8**
l=0, r=1, x=2	i j **(2)**	i j **4**					
l=2, r=6, x=7			i **5**	i **9**	 **(7)**	j **6**	j **8**
			 5	j **6**	i j **(7)**	i **9**	 **8**
l=2, r=3, x=5		j	i j **(5)**	i j **6**			
l=5, r=6, x=9						i **(9)**	j **8**
						j **8**	i **(9)**
Kết quả	2	4	5	6	7	8	9

2.2.3.3. Cài đặt

```
void quickSort(int a[],int l,int r)
{
  int i,j,int x;
```

```
    x=a[(l+r)/2];
    i=l;j=r;
    do
    {
      while (a[i]<x) i++;
      while (a[j]>x) j--;
      if (i<=j)
      {
        swap(a[i],a[j])
        i++;
        j--;
      }
    }while (i<j);
    if (l<j) quickSort(a,l,j);
    if (i<r) quickSort(a,i,r);
  }
```

Đánh giá giải thuật:

Trong trường hợp trung bình độ phức tạp tương đương với O(nlogn)

BÀI TẬP

1. Cho dãy số : -8 -5 -3 -2 1 1 7 15

Hãy tính số lần so sánh để tìm ra số x=1 trong dãy số trên bằng các phương pháp tìm kiếm tuyến tính (có dùng phần tử cầm canh) & tìm kiếm nhị phân.

Cho biết kết quả sau khi tìm kiếm theo hai phương pháp trên.

2. Viết hàm tìm tất cả vị trí xuất hiện của số x trong một dãy số.

3. Viết chương trình cho phép thực hiện lần lượt các công việc sau:

- Nhập một dãy số

- Sắp xếp dãy số đã nhập theo thứ tự tăng dần

- Nhập một số x và kiểm tra x có trong dãy số hay không, nếu có thì xuất tất cả các số bằng x trong dãy số, ngược lại chèn x vào dãy số để được dãy có thứ tự tăng dần.

4. Minh họa từng bước cách sắp xếp dãy số : 5, 15, 12, 2, 10, 12, 7, 0, 7 bằng các phương pháp đã học.

5. Viết chương trình tạo một menu cho phép người dùng chọn lựa để thực hiện thao sắp xếp dãy số tăng dần hoặc giảm dần bằng một trong các phương phép đã học.

6. Mở rộng bài tập 4, cho phép hiển thị trên màn hình kết quả của dãy số sau từng bước sắp xếp .

7. Giả sử trong một công ty, hồ sơ mỗi nhân viên được lưu với các thông tin sau: mã số nhân viên, họ tên, lương, số điện thoại. Hãy viết chương trình hòan để:

a. Nhập danh sách thông tin của các nhân viên.

b. Xuất danh sách nhân viên và các thông tin ra màn hình theo dạng bảng.

c. Tìm một nhân viên có mã số cho trước.

d. Sắp xếp danh sách nhân viên tăng dần theo lương

e. Sắp xếp danh sách nhân viên giảm dần theo lương, nếu cùng lương thì tăng theo họ tên.

8. Viết chương trình thực hiện các công việc sau :

a. Nhập vào một ma trận cấp m x n gồm các số nguyên

b. Sắp xếp các phần tử trên mỗi dòng theo thứ tự tăng dần.

c. Sắp xếp các phần tử của ma trận theo tứ tự tăng dần từ trái qua phải và từ trên xuống dưới

9. Viết hàm trộn 2 mảng số nguyên có thứ tự tăng (gồm m và n phần tử) thành một mảng có thứ tự tăng

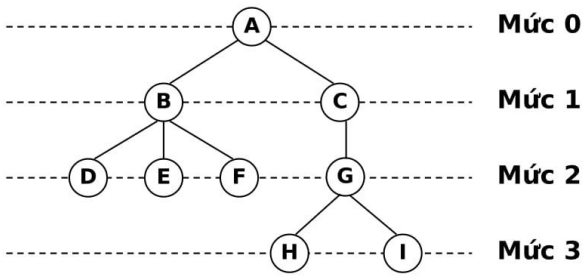
Chương 3: CÂY (TREE)

3.1. CẤU TRÚC CÂY

3.1.1. Định nghĩa

Cây là một tập hợp T (có thể bằng rỗng) các phần tử gọi là node, trong đó có một node đặc biệt được gọi là gốc, các node còn lại được chia thành những tập rời nhau T_1, T_2 , ..., T_n theo quan hệ phân cấp, trong đó T_i cũng là một cây.

Ví dụ sau đây hình ảnh của một cây gồm 9 node, mỗi node chứa một ký tự:



Node A gọi là node gốc của cây.

Các node B,C là node gốc của các cây con của A.

Node A là cha của các node B,C và B,C là các node con của A.

3.1.2. Một số khái niệm cơ bản

3.1.2.1. Bậc (degree) của một node

Là số cây con của node đó.

Ví dụ trong cây trên bậc của node A là 2, bậc của node B là 3, bậc của node C là 1

3.1.2.2.Bậc của một cây

Là bậc lớn nhất của các node trong cây. Cây có bậc n thì gọi l cây n-phân.

Ví dụ cây trên có bậc là 3, vì bậc của node B là 3 là bậc lớn nhất trong cây

3.1.2.3. Node lá (leaf)

Là node có bậc bằng 0.

Ví dụ D, E, F, H, I là các node lá của cây trên

3.1.2.1.. Node nhánh hay node trung gian (interior)

Là node có bậc khc 0 và khơng phải là node gốc .

Ví dụ B, C, G là các node nhnh của cây trên

3.1.2.5. Mức (level) của một node

Mức (level) của node gốc là 0

Mức của node khác gốc bằng mức của node gốc của cây con chứa nó cộng 1.

Ví dụ trong cây trên:

- Node A là gốc nên có mức là 0
- Các node B và C có mức là 1
- Các node D, E, F, G có mức là 2
- Các node H và I có mức là 3

3.1.2.6. Chiều cao của một cây (height)

Là số mức lớn nhất của node có trên cây đó cộng thêm 1

Ví dụ cây trên có chiều cao là 4.

3.1.2.7. Node trước (ancestor), node sau (descendant)

Node A gọi là node trước của node B nếu cây con có gốc là A chứa node B. Và lúc đó, node B được gọi là node sau của node A.

Ví dụ trong cây trên node C là node trước của các node H và I. H và I là các node sau của node C.

3.1.2.8. Node cha (parent), node con (child)

Nếu A là node trước của node B, và mức của node B bằng mức của node A cộng với 1, thì node A được gọi là node cha của node B, và node B được gọi là node con của node A.

Ví dụ trong cây trên B và C là các node con của node A. C là node cha của G, và G là node con của C.

3.1.2.9. Chiều dài đường đi từ gốc đến node x (path length)

Bằng số nhánh từ node gốc đến node x, nghĩa là bằng mức của node x.

Ví dụ trong cây trên đường đi từ A đến G là 2, đường đi từ A đến H l 3.

3.1.2.10. Chiều dài đường đi trong (internal path length) của cây

Chiều dài đường đi trong, ký hiệu P^I, là tổng chiều dài đường đi của tất cả các node trong cây. Gọi d_i là chiều dài đường đi của node thứ i, với cây có n node thì chiều dài đường đi trong của cây được tính theo công thức sau:

$$P^I = \sum_{i=1}^{n} d_i$$

Chiều dài đường đi trong trung bình của cây, gọi là P_A^I, là chiều dài đường đi trong của cây chia cho tổng số node trong cây, và được tính theo công thức:

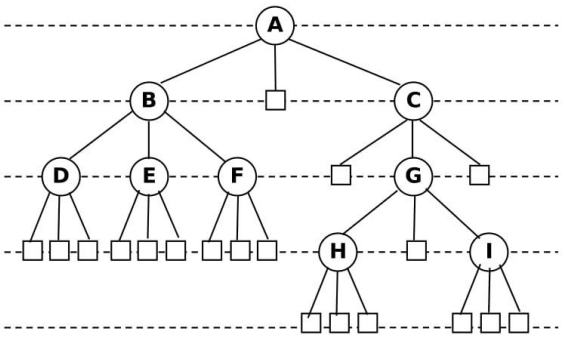
$$P_A^I = \frac{\sum_{i=1}^{n} d_i}{n}$$

Ví dụ cây trên có chiều dài đường đi trong là: 1x0 + (2x1) + (4x2) + (2x3) = 16

Chiều dài đường đi trong trung bình là: 16/9 ≈ 1.8

3.1.2.11. Chiều dài đường đi ngồi (external path length) của cây

Chiều dài đường đi ngoài của cây, ký hiệu P^E, là tổng chiều dài của tất cả các node đặc biệt trong cây. Node đặc biệt là những node giả được thêm vào cây sao cho bậc của tất cả các node bằng với bậc của cây. Hình bên minh họa việc mở rộng cây với các node đặc biệt hình vuông từ cây ban đầu. Cây sau khi mở rộng vẫn có bậc là 3, mỗi node gồm có 3 node con. Chiều dài đường đi ngoài được tính theo công thức dưới đây. Trong đó, d'_i là chiều dài đường đi của node đặc biệt thứ i:



$$P^E = \sum_{i=1}^{m} d'_i$$

Chiều dài đường đi ngoài trung bình của cây, gọi là P_A^E, là chiều dài đường đi ngoài của cây chia cho tổng số node đặc biệt m, và được tính theo công thức:

$$P_A^E = \frac{\sum_{i=1}^{m} d'_i}{m}$$

Ví dụ cây trên có:

- Chiều dài đường đi ngoài của cây:(1x1) + (2x2) + (10x3) + (6x4) = 59

- Chiều dài đường đi ngoài trung bình của cây: 59/19 ≈ 3.1

Lưu ý: Ta có thể tính nhanh số node đặc biệt của cây bằng công thức sau:

$$m = (k - 1) \times n + 1$$

Với n là số node của cây, m là số node đặc biệt, k là bậc của cây.

Chứng minh:

Cây mở rộng có :

n+m - 1 cạnh vì mỗi node có đúng một cạnh đi đến nó trừ node gốc

k x n cạnh vì có đúng k cạnh từ mỗi node đi ra, không có cạnh nào đi ra từ node đặc biệt

Vậy ta có: ***n+m - 1 = k x n*** $\Rightarrow m = (k-1) \times n + 1$

Ví dụ cây trên là cây có bậc ***k***=3, tổng số node của cây là ***n***=9, do đó số node đặc biệt ***m*** cần thêm vào để mở rộng cây sẽ là:

$$m = (3-1) \times 9 + 1 = 19$$

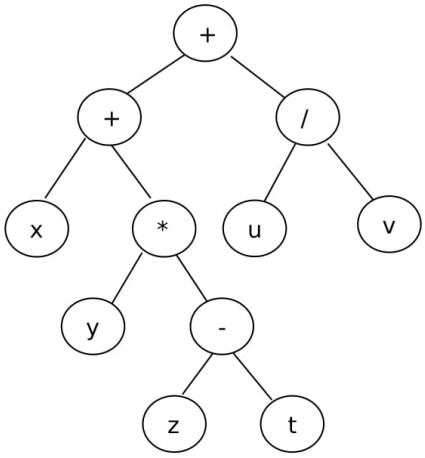
3.1.3. Cây nhị phân (Binary Tree)

3.1.3.1. Định nghĩa

Cây nhị phân là cây mà mỗi node có tối đa 2 cây con

Cây nhị phân có thể ứng dụng trong nhiều bài toán thông dụng..Ví dụ cây nhị phân dưới đây cho ta hình ảnh của một biểu thức tóan học:

.



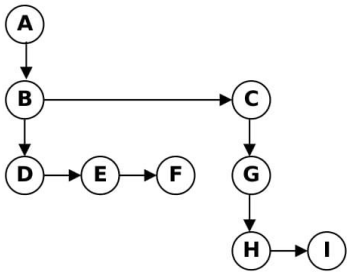
3.1.3.2. Cách chuyển từ cây bất kỳ về cây nhị phân

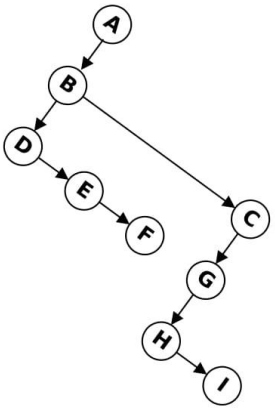
Trong thực tế ta thường gặp các cấu trúc có dạng cây nhị phân. Một cây tổng quát có thể biểu diễn thông qua cây nhị phân theo quy ước:

- Các node có cùng node cha, tức là node anh em, sẽ cùng nằm trên cùng một đường ngang.

- Các node con ở đường phía dưới

Ví dụ cây tam phân ở trên được mô tả lại theo dạng cây nhị phân ở hình dưới đây, mà trong đó mỗi node có tối đa 2 vùng liên kết đến các node con.



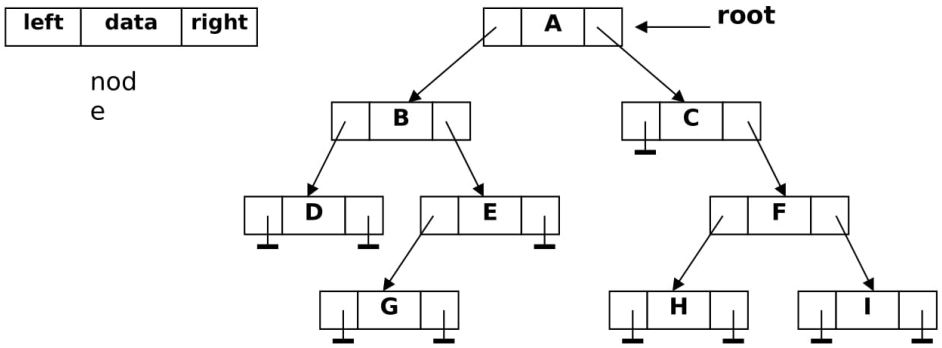


3.1.3.3. Biểu diễn cây nhị phân

Mỗi node trong cây nhị phân, ngoài vùng dữ liệu thực sự, còn có 2 vùng liên kết: một để liên kết đến node con (hay cây con) bên trái (left), và một để liên kết đến node con (hay cây con) bên phải (right).

Một cây nhị phân là tập các node với các liên kết trái và phải. Node lá là node có liên kết trái và liên kết phải bằng **NULL**.

Mỗi cây nhị phân có một node gốc gọi là **root**. Để trỏ đến vị trí node gốc này trong bộ nhớ, ta dùng một con trỏ có cùng kiểu với kiểu của node, và đặt tên là **root**.



Để đơn giản, ta khai báo cấu trúc dữ liệu như sau:

```
typedef struct node
{
   int data;        //Vùng chứa dữ liệu
   node* left;      //Trỏ sang node con bên trái
   node* right;     //Trỏ sang node con bên phải
};
typedef node* tree;
```

Trong chương trình hay một hàm, để khai báo một biến con trỏ có tên **root** để trỏ vào gốc của cây, ta viết khai báo như sau:

```
tree root;
```

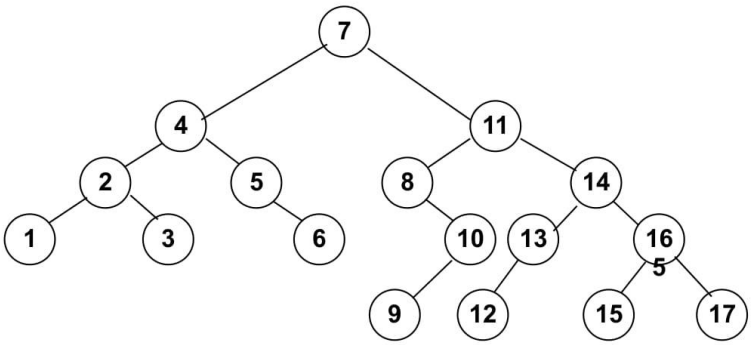
3.2. CÂY NHỊ PHÂN TÌM KIẾM (BINARY SEARCH TREE)

3.2.1. Định nghĩa

Cây nhị phân tìm kiếm, gọi tắt là cây **BST** là cây nhị phân mà trong đó tại mỗi node, giá trị của node lớn hơn giá trị của các node của cây con bên trái, nhỏ hơn giá trị của các node của cây con bên phải.

Khi duyệt cây BST theo phép duyệt **LNR**, ta sẽ được dãy các giá trị có thứ tự tăng dần.

Xét một ví dụ cây nhị phân tìm kiếm sau với giá trị các node là các số nguyên từ 1 đến 17:



Nếu dùng phép duyệt LNR, ta sẽ được dãy có thứ tự:

1, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **11**, **12**, **13**, **14**, **15**, **16**, **17**

3.2.2. Một số thao tác trên cây nhị phân tìm kiếm

3.2.2.1. Thêm một nút vào cây

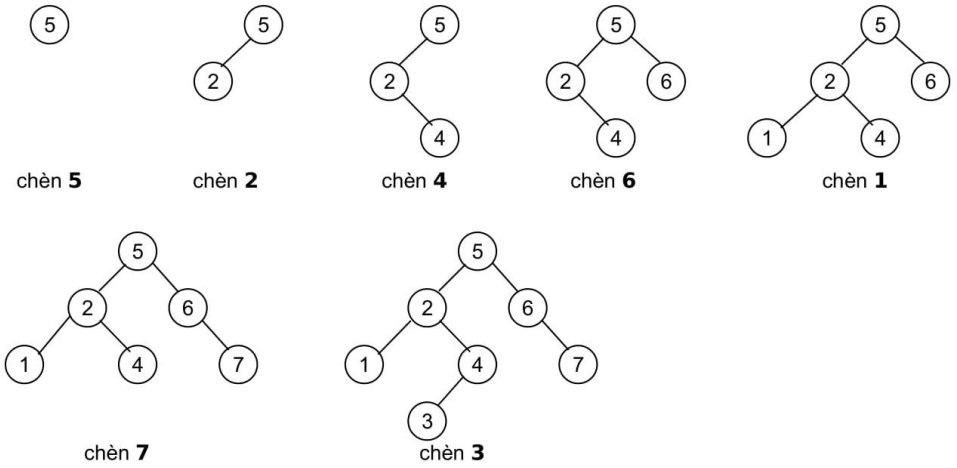
Hàm **InsertTree(root,x)** cho phép chn một node có khóa **x** vào cây có gốc **root** và trả về giá trị –1 khi không đủ bộ nhớ, giá trị 0 khi số x đã có trong cây, gi trị 1 khi thao tác chèn thành công.

```
int insertTree(tree &root,int x)
{
   if (root!=NULL)
   {
      if (root->data==x) return 0;
      if (root->data >x)   return insertTree(root->left,x);
      else    return insertTree(root->right,x);
   }
   else
   {
      root=new node;
      if (root==NULL) return -1;
```

```
        root->data=x;
        root->left=NULL;
        root->right=NULL;
        return 1;
    }
}
```

Ví dụ: Sau dây là một ví dụ tạo cây nhị phân tìm kiếm từ dãy số nguyên dùng giải thuật tìm kiếm và chèn thêm node vào cây. Dãy số nguyên cho trước gồm các giá trị:

5, 2, 4, 6, 1, 7, 3



3.2.2.2. Tạo cây

```
void createTree(tree &root)
{
   int x,n;
   cout<<"Nhap n = ";
   cin>>n;
   for (int i=1;i<=n;i++)
```

```
    {
       cin>>x;
       insertTree(root,x);
    }
}
```

3.2.2.3. Duyệt cây

Duyệt cây (traverse) là lần lượt đi qua tất cả các node trong cây, mỗi node chỉ được duyệt duy nhất một lần, tuy nhiên mỗi node có thể đi qua nhiều lần.

Thao tác duyệt cây trên cây nhị phân tìm kiếm nói riêng và cây nhị phân nói chung hòan toàn giống nhau, gồm 6 phép duyệt cây nhị phân theo thứ tự của node gốc như sau:

i. **PreOrder (node gốc trước):** Node–Left–Right (NLR) và Node–Right–Left (NRL).

ii. **InOrder (node gốc giữa):** Left–Node–Right (LNR) và Right–Node–Left (RNL).

iii. **PostOrder (node gốc sau):** Left–Right–Node (LRN) và Right–Left–Node (RLN).

Trong chương trình môn học này ta xét 3 kiểu duyệt chính có thể áp dụng trên cây nhị phân: duyệt theo thứ tự trước (NLR), thứ tự giữa (LNR) và thứ tự sau (LRN).

Ví dụ: Kết quả tương ứng với mỗi phép duyệt đã nêu trên cho cây ví dụ ở mục 3 như sau:

NLR: **A B D E G C F H I**

LNR: **D B G E A C H F I**

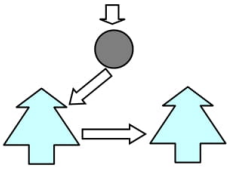
LRN: **D G E B H I F C A**

3.2.2.3.1. Duyệt theo thứ tự trước (Node-Left-Right)

Trong phép duyệt NLR, ta bắt đầu duyệt từ node gốc, sau đó duyệt cây con bên trái, rồi tới cây con bên phải. Đối với các cây con, ta cũng bắt đầu duyệt từ node gốc của cây con đó, sau đó là các cây con bên trái và bên phải. Tại mỗi node được duyệt, ta sẽ xuất giá trị vùng **data** của node đó.

Vì các thao tác duyệt hòan tòan giống nhau về mặt bản chất, nên ta sẽ cài đặt hàm duyệt **traverseNLR()** theo kỹ thuật đệ qui như sau:

```
void traverseNLR(tree root)
{
   if(root!=NULL)
   {
      cout<<root->data<<"\t";
      traverseNLR(root->left);
      traverseNLR(root->right);
   }
}
```

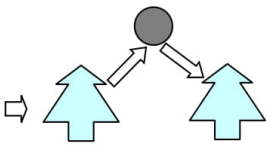


3.2.2.3.2. Duyệt theo thứ tự giữa (Left- Node-Right)

Trong phép duyệt LNR, ta bắt đầu duyệt từ cây con bên trái, sau đó tới node gốc, và cuối cùng là cây con bên phải. Ở mỗi cây con, ta cũng thực hiện các thao tác duyệt một cách đệ qui như vậy.

Hàm duyệt đệ qui **traverseLNR()** cho phép duyệt InOrder được cài đặt như sau:

```
 void traverseLNR(tree root)
{
   if(root!=NULL)
   {
      traverseLNR(root->left);
      cout<<root->data<<"\t";
```



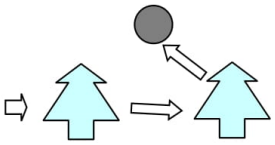
```
        traverseLNR(root->right);
    }
}
```

3.2.2.3.3. Duyệt theo thứ tự sau (Left -Right- Node)

Trong phép duyệt LRN, ta bắt đầu duyệt từ cây con bên trái, sau đó tới cây con bên phải, và cuối cùng duyệt node gốc của cây con đó. Ở mỗi cây con, ta cũng thực hiện các thao tác duyệt một cách đệ qui như vậy.

Tương tự như các phép duyệt ở trên, ta xây dựng hàm **traverseLRN()** theo kỹ thuật đệ qui như sau:

```
void traverseLRN(tree root)
{
    if(root!=NULL)
    {
        traverseLRN(root->left);
        traverseLRN(root->right);
        cout<<root->data<<"\t";
    }
}
```



3.2.2.4. Tìm phần tử có khóa x

Hàm trả về địa chỉ của phần tử có khóa (giá trị) bằng x:

```
node* searchTree(tree root,int x)
{
    while (root!=NULL)
    {
        if (root->data==x) return root;
        if (root->info>x) root=root->left;
```

```
        else root=root->right;
      }
      return NULL;
   }
```

Nhận xét: Số lần so sánh tối đa để tìm phần tử x là h, với h là chiều cao của cây. Như vậy thao tác tìm kiếm trên cây BST có n node tổng chi phí trung bình khoảng $O(\log_2 n)$.

3.2.2.5. Xóa phần tử có khóa x

3.2.2.5.1. Ý tưởng của giải thuật

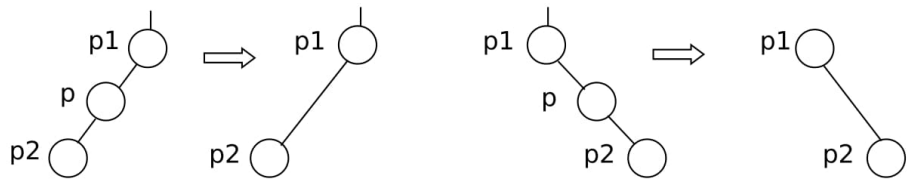
Gọi p là node có khóa x, ta có 3 trường hợp:

Trường hợp 1: p là node lá

Trước khi hủy p ta cho node cha của p thay vì trỏ vào p thì trỏ vào NULL.

Trường hợp 2: p có duy nhất một cây con

Trước khi hủy p ta cho node cha của p thay vì trỏ vào p thì trỏ vào node con của p.



Trường hợp 3: p có hai cây con

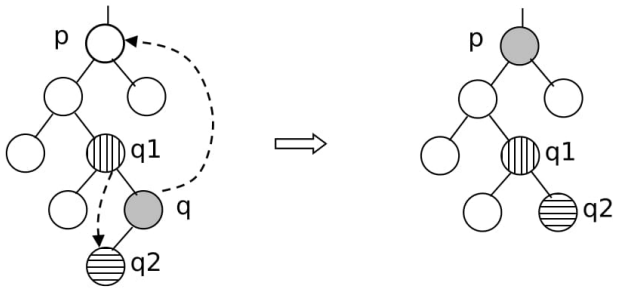
Ta sẽ hủy gián tiếp p bằng cách tìm phần tử thế mạng q (có tối đa một cây con). Dữ liệu lưu tại q sẽ được chuyển lên lưu tại p. Sau đó sẽ hủy q.

Vấn đề là phải chọn q sao cho sau khi hủy cây vẫn là cậy BST. Có 2 node thỏa mãn yêu cầu:

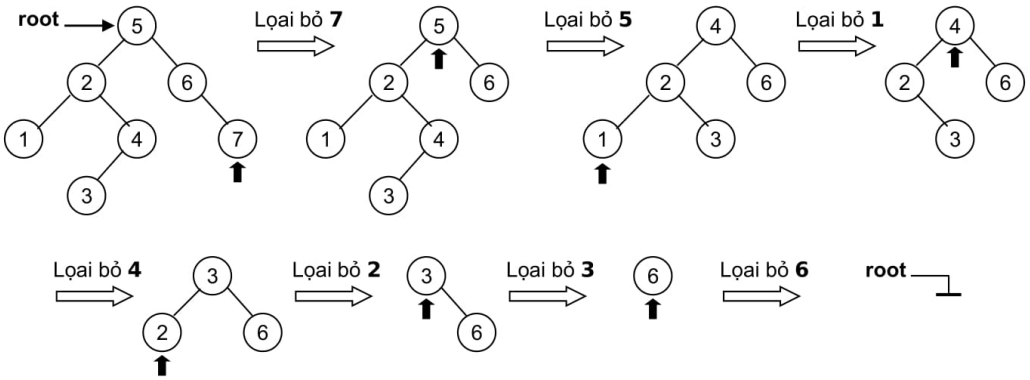
Node cực trái trên cây con phải của p.

Node cực phải trên cây con trái của p.

Việc chọn lựa phần tử nào là phần tử thế mạng hồn toàn phụ thuộc vào ý thích của người lập trình. Ở đây, chúng ta sẽ chọn node cực phải của cây tri làm phân tử thế mạng.



Ví dụ: Quá trình lọai bỏ các node theo thứ tự: **7**, **5**, **1**, **4**, **2**, **3**, **6**



3.2.2.5.2. Cài đặt

Hàm removeNode() trả về 1 nếu hủy thành công và trả về 0 nếu không tìm thấy node có khóa x.

```
int removeNode(tree &root,int x)
```

```
{
  if (root==NULL)   return 0;
  if (root->data< x) return RemoveNode(root->right,x);
  if (root->data> x)  return RemoveNode(root->left,x);
  node *p=root;
  if (root->right==NULL) oot=root->left;  //Trường hợp 1,2
  else
    if (root->left==NULL)  root=root->right;//Trường hợp 2
    else
      searchStandFor(p,root->left);          //Trường hợp 3
  delete p;
  return 1;
}
void searchStandFor(tree &p,tree &q)
{
  if (q->right!=NULL)
    SearchStandFor(p,q->right);     //Tìm node cực phải của
                                          cây con trái của p
  else                          //Tìm thấy  q là node cực phải
  {
    p->data=q->data;          //Ghi đè dữ liệu của q vào p
```

```
    p=q;              //Đặt lại p là node cực phải để xóa p
    q=q->left;           //Cho node cha của q trỏ vào node
                                                con trái của q
  }
}
```

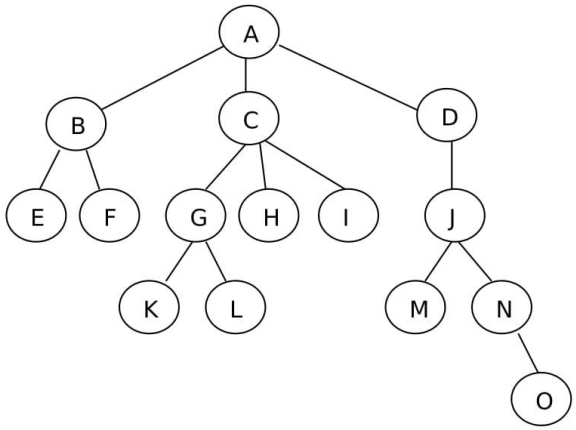
3.2.2.6. Xóa toàn bộ cây

Việc xóa toàn bộ cây có thể được thực hiện thông qua thao tác duyệt cây theo thứ tự sau. Nghĩa là ta sẽ hủy cây con trái, cây con phải rồi mới hủy node gốc.

```
void removeTree(tree &root)
{
  if (root!=NULL)
  {
      removeTree(root->left);
      removeTree(root->right);
      delete root;
  }
}
```

BÀI TẬP

1. Cho cây sau, hãy cho biết:



a. Các node lá ?

b. Các node nhánh ?

c. Node cha của node G ?

d. Các node con của node C ?

e. Các node anh em của node B ?

f. Mức của node D và node L ?

g. Bậc của node B và node D ?

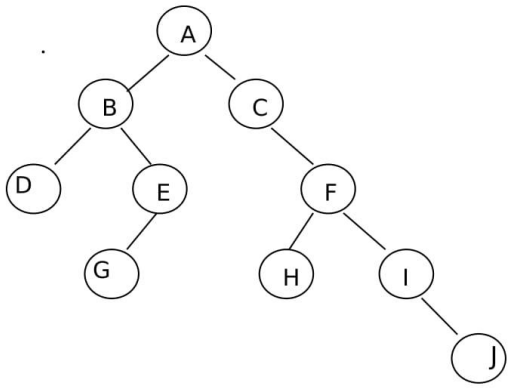
h. Bậc của cây ?

i. Chiều cao của cây này ?

j. Độ di đường đi từ A đến F và từ A dến G ?

k. Các đường đi từ gốc A có độ dài 3 trên cây này ?

2. Cho cây nhị phân sau. Hãy viết dãy các node được thăm khi duyệt cây này theo các thứ tự: LNR, NLR, LRN



3. Cho một cây nhị phân T. Hỏi ở mức thứ 100 cây T có tối đa bao nhiêu nút ?

4. Cho cây nhị phân tìm kiếm gồm 11 số nguyên, với thứ tự nhập các nút được cho như sau: 14, 11, 9, 18, 16, 20, 30, 10, 17, 1, 15

a. Hãy vẽ cây trên.

b. Hãy duyệt cây trên theo các thứ tự Node- Left-Right, Left- Node-Right, Left- Right-Node.

b. Hãy vẽ cây sau khi hủy node 14, 16, 11, 18

5. Cho cây nhị phân tìm kiếm gồm 11 số nguyên, với thứ tự nhập các nút được cho như sau: 8, 3, 5, 2, 20, 11, 30, 9,18, 4

a. Hãy vẽ cây trên.

b. Hãy duyệt cây trên theo các thứ tự Node- Left-Right, Left- Node-Right, Left- Right-Node.

b. Hãy vẽ cây sau khi hủy node 2, 5, 20, 8.

6. Cho cây nhị phân, hãy viết các hàm:

a. Đếm số nút lá

b. Đếm số nút có đúng 1 cây con

c. Đếm số nút có đúng 2 cây con

d. Đếm số nút có khoá nhỏ hơn x

e. Đếm số nút có khoá nhỏ hơn x và lớn hơn y

f. Tính chiều cao của cây

g. In ra tất cả các nút ở mức thứ k của cây

h. Tính chiều dài đường đi trong của cây

k. Tính chiều dài đường đi ngoài của cây

7. Viết các hàm :

a. Đọc các giá trị từ danh sách liên kết đơn và lưu vào một cây nhị phân tìm kiếm

b. Duyệt cây nhị phân tìm kiếm theo thứ tự NLR và lưu vào một danh sách liên kết đơn

8. Viết chương trình nhập và xuất một cây nhị phân có các node chứa ký tự.

9. Viết hàm kiểm tra xem hai cây nhị phân có giống nhau về hình dáng không.

10. Hãy viêt chương trình dung cấu trúc cây nhị phân cho phép:

a. Lưu gia phả của một dòng họ. Thông tin mỗi thành viên gồm họ tên, năm sinh, vợ/chồng, con ruột, con dâu/rể (nếu có).

b. Tìm một người theo họ tên xem có trong gia phả hay không. Nếu có thì in tất cả các thông tin về cha, me, anh, chị, em, vợ chồng của người này.