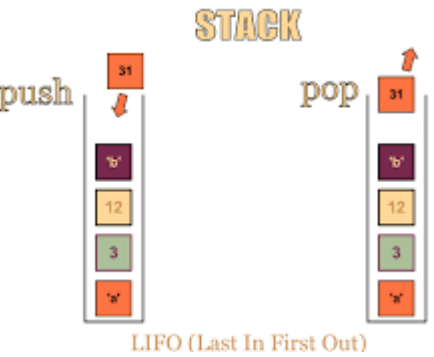
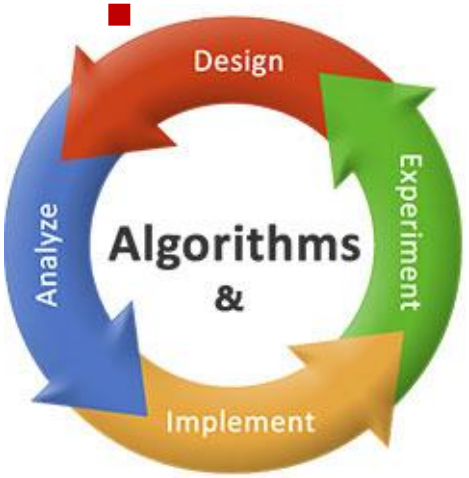
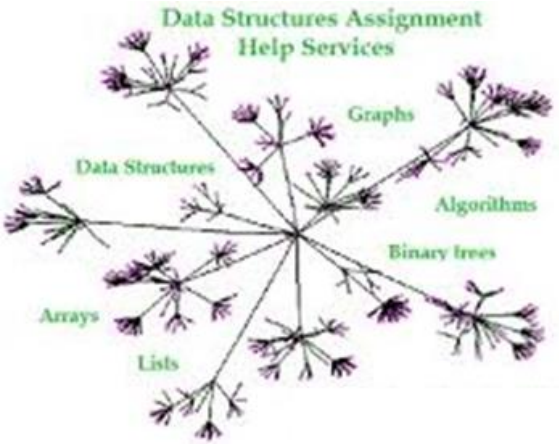


CẤU TRÚC DỮ LIỆU

&

GIẢI THUẬT





Lê Văn Hạnh

levanhanhvn@gmail.com

NỘI DUNG MÔN HỌC

- Chương 1: Ôn tập ngôn ngữ lập trình C
- Chương 2: Kiểu dữ liệu con trỏ
- Chương 3: Tổng quan về cấu trúc dữ liệu và giải thuật
- **Chương 4: Danh sách kề (Danh sách tuyến tính)**
- Chương 5: Các giải thuật tìm kiếm trên danh sách kề
- **Chương 6**: Các giải thuật sắp xếp trên danh sách kề
- **Chương 7**: Danh sách liên kết động (*Linked List*)
- **Chương 8**: Ngăn xếp *(Stack)*
- **Chương 9:** Hàng đợi *(Queue)*
- **Chương 10**: Cây nhị phân tìm kiếm (*Binary Search Tree*)
- **Chương 11**: Cây cân bằng (*Binary Search Tree*)
- **Chương 12**: Bảng băm (*Hash Table*)

Chương 4

DANH SÁCH KỀ

(Danh sách tuyến tính - Contiguous List)

NỘI DUNG

1. Giới thiệu danh sách
2. Danh sách kề
3. Các bước cài đặt chương trình có sử dụng danh sách kề
4. Các thao tác trên danh sách kề
5. Ưu khuyết điểm của danh sách kề
6. Thực hành

1. GIỚI THIỆU DANH SÁCH

- Danh sách là một tập hợp các phần tử có cùng kiểu dữ liệu (quy ước gọi mỗi phần tử là một node).
- Các node trong danh sách có mối liên hệ tương đối: nếu biết được vị trí node thứ i thì ta sẽ biết được vị trí node thứ i + 1 và i-1.

1. Giới thiệu danh sách

– Ví dụ: Bảng sau mô tả một bảng danh sách sinh viên, mỗi dòng của bảng là một node của danh sách mô tả thông tin về sinh viên bao gồm mã số sinh viên, họ tên sinh viên, năm sinh, địa chỉ

Mssv	Họ tên	Nsinh	Địa chỉ
0032	Lê Minh Triết	1976	Hoà Hưng
0056	Nguyễn Thuý Hà	1977	Nguyễn Huệ
1234	Huỳnh Văn Trọng	1976	Lữ Gia

0032	0056	1234
Lê Minh Triết	Nguyễn Thuý Hà	Huỳnh Văn Trọng
1976	1977	1976
Hoà Hưng	Nguyễn Huệ	Lữ Gia

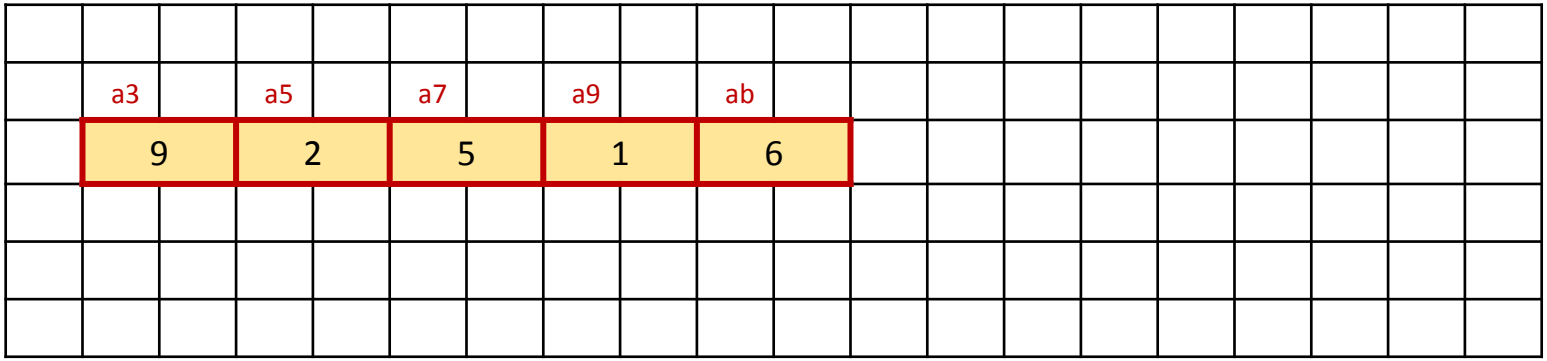
Đầu danh sách

Cuối danh sách

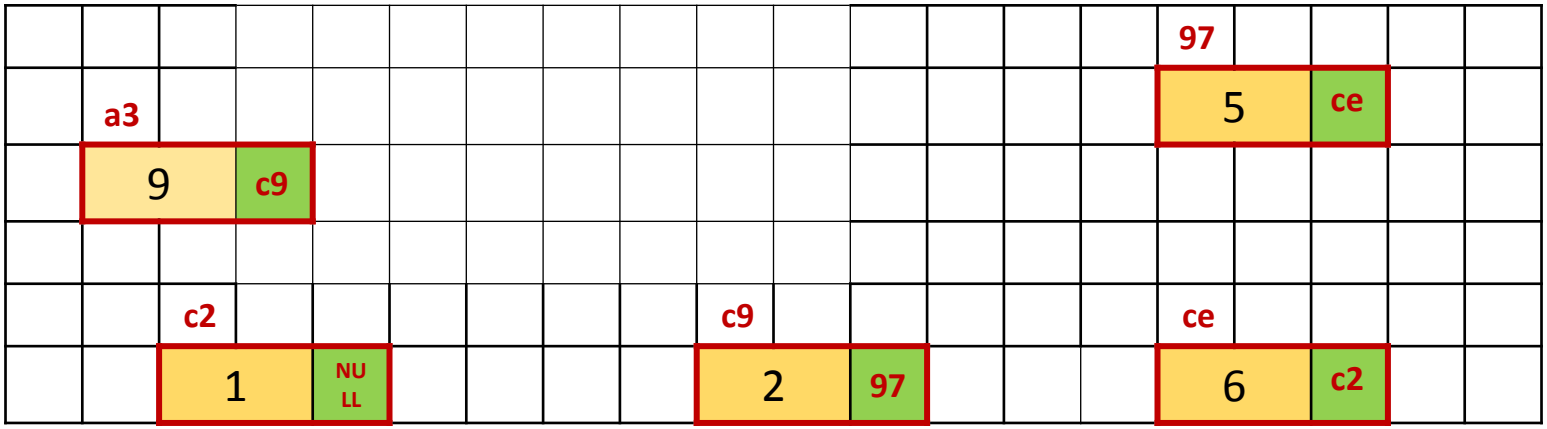
1. Giới thiệu danh sách

– Phân loại danh sách

- ***Danh sách kề*** (Danh sách tuyến tính - Contiguous List). Ví dụ: int A[5]={9,2,5,1,6}



- ***Danh sách liên kết*** (linked list)



2. Danh sách kề

Danh sách kề là một danh sách mà các node được sắp xếp kế tiếp nhau trong bộ nhớ, đứng ngay sau vị trí của node thứ i là vị trí của node thứ i +1.

0	*1*	*2*	*3*	*…*
0032	0056	1234	0027	…
Lê Minh Sửu	Lý Thị Mẹo	Phan Văn Dần	Trần Văn Tý	…
1996	1997	1996	1998	…
Hoà Hưng	Nguyễn Huệ	Lữ Gia	Lê Lợi	…

3. Các bước cài đặt chương trình có sử dụng danh sách kề

3.1. Các bước cần thực hiện

Khai báo cấu trúc danh sách kề (DSKe)

Khai báo biến kiểu DSKe sẽ dùng trong chương trình

Khởi tạo DSKe

Xây dựng các hàm giúp thao tác trên DSKe:

(1)Xác định số node trong DSKe

(2) Kiểm tra DSKe có rỗng hay không?

(3) Kiểm tra DSKe có đầy hay không?

(4) Thêm node vào DSKe

(5)Xóa node khỏi DSKe

Sử dụng DSKe trong chương trình

3.2. Khai báo cấu trúc danh sách kề

```
#define MAXLIST    50
struct LIST {
    int count; //Số node hiện có trong danh sách
    int nodes[MAXLIST]; /*Mỗi node của danh sách
                          là một phần tử trên mảng*/
};
```

4. Các thao tác trên danh sách kề

i. Tác vụ Init (khởi tạo)

- *Chức năng* : khởi tạo danh sách ban đầu rỗng.
- *Dữ liệu nhập*: danh sách được sử dụng làm tham biến.
- *Cài đặt* :

```
void Initialize(LIST &list)
{
      list.count = 0;
}
```

ii.Tác vụ isEmpty

- *Chức năng* : kiểm tra danh sách có rỗng hay không?
- *Dữ liệu nhập* : danh sách cần kiểm tra làm tham trị.
- *Dữ liệu xuất* : *TRUE|FALSE*
- *Cài đặt* :

```
bool isEmpty(LIST ds)
{
    return ds.count==0 ;
}
```

true

false

iii.Tác vụ isFull

- *Chức năng* : kiểm tra danh sách đã đầy hay chưa?
- *Dữ liệu nhập* : danh sách cần kiểm tra làm tham trị.
- *Dữ liệu xuất* : **TRUE | FALSE**
- *Cài đặt* :

```
bool isFull(LIST ds)
{
    return ds.count== MAXLIST ;
}
```

true

false

iv.Tác vụ Insert

- *Chức năng* : thêm node chứa info vào vị trí pos (vị trí thêm node mới) trên danh sách.
- *Dữ liệu nhập*: danh sách làm tham biến; node mới chứa info và vị trí pos làm tham trị.
- *Điều kiện:* `0 =< pos <= LIST.count`

  `isFull() = false//danh sách chưa đầy`
- *Dữ liệu xuất* : không.

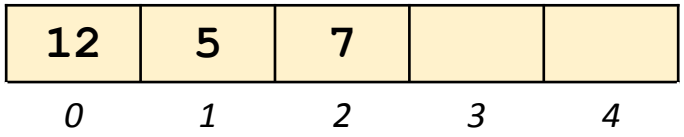
iv.Tác vụ AddTail

- *Cài đặt :*

```
bool AddTail (LIST &ds, int info)
{
    if ((isFull(ds)==true) || (pos<0) || (pos>ds.count))
        return false;
    else
    {
        ds.nodes[ds.count++] = info;
        return true;
    }
}
```

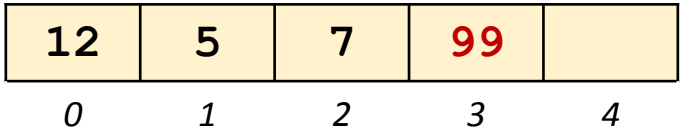
Before:

count=3



After add info=99

⇒ count=4



v. Tác vụ CreateList

- *Chức năng* : Cho người dùng thêm các node vào danh sách ban đầu (đang rỗng)
- *Dữ liệu nhập* : danh sách cần thêm làm tham biến.
- *Dữ liệu xuất* : không
- *Cài đặt* :

```
void CreateList(LIST &ds)
{   do
    {    printf("\nSo phan tu cua danh sach:");
         scanf("%d", &ds.count);
         if ((ds.count <= 0) || (ds.count >= MAXLIST))
              printf("So phan tu phai >0 va <%d",MAXLIST);
    } while ( (ds.count <= 0) || (ds.count >= MAXLIST) );
    for (int i = 0; i < ds.count; i++)
    {    int x;
         printf("ds[%d] =", i + 1);
         scanf("%d", &x);
         AddTail(ds,x,i);
    }
}
```

vi.Tác vụ Traverse

- *Chức năng*: duyệt và in giá trị của tất cả các *node* có trong danh sách.
- *Dữ liệu nhập*: không.
- *Dữ liệu xuất*: in giá trị của tất cả các *node* có trong danh sách.
- *Cài đặt*:

```
void Traverse(LIST list)
{    if(isEmpty(list))
          printf("Danh sach rong\n");
     else
     {    int i;
          for(i = 0; i < list.count; i++)
               printf("%5d", list.nodes[i]);
     }
}
```

vii.Tác vụ ListSize

- *Chức năng*: kiểm tra số node có trong danh sách.
- *Dữ liệu nhập*: danh sách cần biết số node là tham trị.
- *Dữ liệu trả về*: số node trong danh sách.
- *Cài đặt*:

```
int ListSize(LIST ds)
{
    return ds.count ;
}
```

viii.Tác vụ Retrieve

- *Chức năng*: truy xuất thông tin của node tại vị trí pos trong danh sách.
- *Dữ liệu nhập*: danh sách và pos là vị trí của node cần truy xuất, cả 2 tham số này đều là tham trị.
- *Điều kiện: 0 =< pos < LIST.count*
- *Dữ liệu trả về:* thông tin của node tại vị trí pos trong danh sách.
- *Cài đặt:*

```
int Retrieve(LIST ds, int pos)
{
    return ds.nodes[pos] ;
}
```

ix.Tác vụ Search

- *Chức năng* : tìm kiếm một node trong danh sách theo một khoá tìm kiếm.
- *Dữ liệu nhập*: danh sách sẽ được tìm kiếm trên đó, key (hay info) là khóa cần tìm. Cả 2 tham số này đều là tham trị.
- *Dữ liệu xuất* : vị trí (chỉ số) tìm thấy info, ngược lại, trả về -1 khi không tìm thấy.
- *Cài đặt:*

```
int Search(LIST ds, int info)
{
    for (int i = 0; i <ds.count; i++)
        if (ds.nodes[i] == info)
            return i;
    return -1;
}
```

x. Tác vụ Insert

- *Cài đặt :*

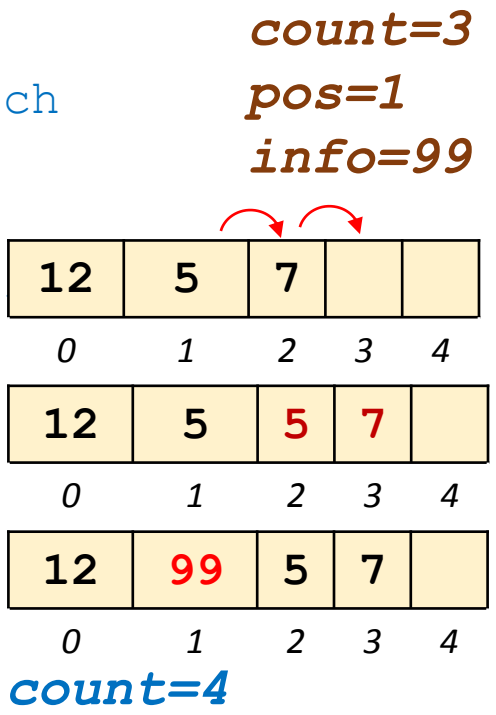
```
bool Insert (LIST &ds, int info, int pos)
{   if ((isFull(ds)==true) || (pos<0) || (pos>ds.count))
        return false;

    //B1: dời các phần tử từ pos đến ds.count-1 sang phải
    for (int i = ds.count; i>pos ; i--)
        ds.nodes[i] = ds.nodes[i-1] ;

   //B2: đưa info vào vị trí pos của danh sách
    ds.nodes[pos] = info;

   //B3: tăng số lượng node trong danh sách
    ds.count++;

    return true;
}
```



xi. Tác vụ Remove

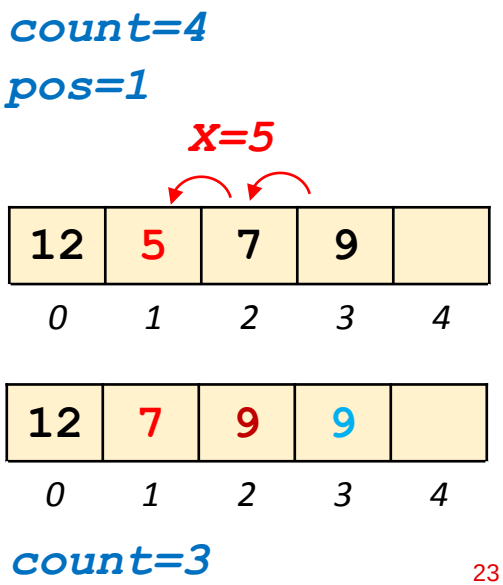
- *Chức năng*: Xóa node tại vị trí pos của danh sách.
- *Dữ liệu nhập*: danh sách có node cần xóa làm tham biến, biến kiểu số nguyên pos (vị trí của node xóa) làm tham trị.
- *Điều kiện*: cần thực hiện tác vụ Search trước khi xóa. Nếu sau khi tìm kiếm có kết quả là tìm thấy, chương trình sẽ gọi tác vụ này thực hiện. Ngược lại sẽ thông báo không tìm thấy.
- *Dữ liệu xuất*: không

xii. Tác vụ Remove

- *Cài đặt*

```
void Remove(LIST &ds, int pos)
{
    //B1: dời các phần từ vị trí pos đến n-1 sang trái
    for (int i = pos+1; i< ds.count; i++)
            ds.nodes[i - 1] = ds.nodes[i];

    //B2: giảm số lượng phần tử trong danh sach
    ds.count--;
}
```



xiii. Tác vụ Replace

- *Chức năng*: thay thế node tại vị trí pos của danh sách bằng node khác.
- *Dữ liệu nhập*: danh sách có node cần thay thế, node khác và vị trí thay thế pos. Cả 3 tham số đều là tham trị.
- *Điều kiện: 0 =< pos <= count-1*
- *Dữ liệu xuất*: không
- *Cài đặt:*

```
void Replace(LIST list, int info, int pos)
{
      if(isEmpty(list))
            printf("Danh sach rong\n");
      else if ((pos < 0) || (pos >= list.count))
                  printf("Vi tri %d không hop le\n", pos);
            else
                  list.nodes[pos] = info;
}
```

xiv. Tác vụ Sort

- *Chức năng* : sắp xếp lại danh sách tăng dần theo info.
- *Dữ liệu nhập* : danh sách cần sắp xếp làm tham trị.
- *Dữ liệu xuất* : không
- Cài đặt:

```
void Sort(LIST list)
{
     int i, j, tmp;
     for(i=0; i < list.count-1; i++)
          for( j=i+1 ; j<list.count; j++)
               if(list.nodes[i]>list.nodes[j])
                    Swap(list.nodes[i], list.nodes[j]);
}
void Swap(int &a, int &b)
{    int tmp = a;
     a = b;
     b = tmp;
}
```

xv. Tác vụ ClearList

- *Chức năng*: xoá info tất cả các node có trong danh sách.
- *Dữ liệu nhập*: danh sách cần xóa làm tham biến
- *Dữ liệu xuất*: không
- *Cài đặt:*

```
void ClearList(LIST &ds)
{
    ds.count = 0;
}
```

5. Ưu khuyết điểm của danh sách kề

5.1. Ưu

- Nếu gọi tỷ số giữa số bytes lưu trữ thông tin và tổng số byte của danh sách là mật độ sữ dụng thì mật độ sử dụng bộ nhớ của danh sách kề bằng 1, tức là tối ưu vì không lãng phí bộ nhớ.
- Do dùng phương pháp truy xuất trực tiếp nên thời gian truy xuất đến phần tử thứ i rất nhanh.
- Thao tác tìm kiếm một node trong trường hợp danh sách đã được sắp thứ tự bằng phương pháp tìm kiếm nhị phân sẽ rất nhanh.

5.2. Nhược

- Vì số node cấp phát cho danh sách kề là cố định khi chương trình đang chạy nên số node cần dùng lúc thừa lúc thiếu.
- Không phù hợp với phép thêm vào và loại bỏ vì mỗi lần thêm vào hoặc loại bỏ cần phải đổi chỗ nhiều lần. Đặc biệt trường hợp xấu nhất là khi thêm vào và loại bỏ ở đầu danh sách.

$\Rightarrow$ Không nên sử dụng cho các danh sách hay bị biến động. Khi cần sử dụng danh sách thường bị biến động, người ta chọn cấu trúc là danh sách liên kết.

6. THỰC HÀNH

1. Để quản lý 1 danh sách các phân số, cần thực hiện các yêu cầu sau:
   - Khai báo cấu trúc phân số
   - Xây dựng các hàm:
     a. Khởi tạo
     b. Kiểm tra DS có rỗng hay không?
     c. Kiểm tra DS đã đầy hay chưa?
     d. Tão 1 node (phân số). Khi tạo, cần thực hiện:
        - Rút gọn phân số
        - Nếu tử số và mẫu số đều là số âm, phải khử dấu âm.
        - Nếu tử số>0 và mẫu số<0 thì chuyển dấu âm cho tử số
     e. Thêm n phân số vào DS (sẽ gọi hàm của câu d)
     f. Xuất 1 node (phân số)
     g. Xuất toàn bộ DS (sẽ gọi hàm của câu f)
     h. Chèn 1 phân số vào vị trí pos cho trước.
     i. Tìm kiếm 1 phân số có trong DS hay không?
     j. Tính tổng các phân số có trong danh sách.
     k. Tìm phân số lớn nhất
     l. Sắp xếp danh sách

6. THỰC HÀNH

2. Để quản lý 1 danh sách các học viên, cần thực hiện các yêu cầu sau:

- Khai báo cấu trúc học viên gồm các thông tin: mã số, họ và tên, điểm lý thuyết, điểm thực hành, điểm trung bình với DTB=(DLT+DTH)/2. Như vậy, người dùng không cần nhập điểm trung bình, điểm này do chương trình tự tính.
- Xây dựng các hàm:
  a. Tạo 1 node (học viên).
  b. Thêm n học viên vào DS (sẽ gọi hàm của câu a)
  c. Xuất thông tin của 1 node (học viên) ra màn hình
  d. Xuất toàn bộ DS (sẽ gọi hàm của câu c)
  e. Chèn 1 học viên vào vị trí pos cho trước.
  f. Tìm kiếm 1 học viên có trong DS hay không?
  g. Tính điểm trung bình của tất cả các học viên có trong danh sách.
  h. Tìm tên của học viên có điểm trung bình lớn nhất.
  i. Sắp xếp danh sách tăng dần theo điểm trung bình

6. THỰC HÀNH

3. Để quản lý 1 danh sách các khách hàng, cần thực hiện các yêu cầu sau:

- Khai báo cấu trúc khách hàng gồm các thông tin: mã số, họ và tên, ngày sinh, địa chỉ với ngày sinh là 1 biến thuộc kiểu cấu trúc **NGAY**, trong cấu trúc này gồm 3 thành phần kiểu số nguyên là ngày, tháng, năm.
- Xây dựng các hàm:
  a. Tạo 1 node (khách hàng).
  b. Thêm n khách hàng vào DS (sẽ gọi hàm của câu a)
  c. Xuất thông tin của 1 node (khách hàng) ra màn hình
  d. Xuất toàn bộ DS (sẽ gọi hàm của câu c)
  e. Tìm kiếm 1 khách hàng có trong DS hay không?
  f. Tìm những khách hàng có tháng sinh là 8.
  g. Sắp xếp danh sách tăng dần theo mã số


Q&A
How
Why
When
Where
What
Who