ỦY BAN NHÂN DÂN TỈNH ĐỒNG THÁP
TRƯỜNG CAO ĐẲNG CỘNG ĐỒNG ĐỒNG THÁP

GIÁO TRÌNH
MÔ ĐUN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
NGÀNH, NGHỀ: CÔNG NGHỆ THÔNG TIN
TRÌNH ĐỘ: CAO ĐẲNG

(Ban hành kèm theo Quyết định số /QĐ-CĐCĐ ngày tháng năm 20…
của Hiệu trưởng trường Cao đẳng Cộng đồng Đồng Tháp)

Đồng Tháp, năm 2017

TUYÊN BỐ BẢN QUYỀN

CHƯƠNG I:

TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

I. Khái niệm giải thuật và đánh giá độ phức tập của giải thuật

1. Khái niệm

Khái niệm giải thuật hay thuật giải mà nhiều khi còn gọi là thuật toán dùng để chỉ phương pháp hay cách thức (method) để giải quyết vấn đề. Giải thuật có thể được minh họa bằng ngôn ngữ tự nhiên (natural), bằng sơ đồ (flow chart) hoặc bằng mã giả (pseudo code). Trong thực tế giải thuật thường được minh họa hay thể hiện bằng mã giả tựa trên một hay một số ngôn ngữ lập trình nào đó (thường là ngôn ngữ mà người lập trình chọn để cài đặt thuật toán), chẳng hạn như C, Pascal, …

Khi đã xác định được cấu trúc dữ liệu thích hợp, người lập trình sẽ bác đầu tiến hành xây dựng giải thuật tương ứng theo yêu cầu của bài toán đặt ra trên cơ sở của cấu trúc dữ liệu đã được chọn. Đề giải quyết một vấn đề có thể có nhiều phương pháp, do vậy sự lựa chọn phương pháp phù hợp là một việc mà người lập trình phải cân nhắc và tính toán. Sự lựa chọn này cũng có thể góp phần đáng kể trong việc giảm bớt công việc của người lập trình trong việc cài đặt thuật toán trên một ngôn ngữ cụ thể.

2. Đánh giá độ phức tạp của giải thuật

Các tiêu chuẩn đánh giá cấu trúc dữ liệu

Đánh giá một cấu trúc dữ liệu ta thường dựa vào một số tiêu chí sau:

- Cấu trúc dữ liệu phải tiết kiệm tài nguyên (bộ nhớ trong).
- Cấu trúc dữ liệu phải phản ánh đúng thực tế của bài toán.
- Cấu trúc dữ liệu phải dể dàng trong thao tác dữ liệu.

Đánh giá độ phức tạp của thuật toán

Việc đánh giá độ phức tạp của bài toán quả không dễ chút nào. Ở đây chúng ta chỉ mới ước lượng thời gian thực hiện bài toán T(n) để có sự so sánh tương đối giữa các thuật toán với nhau. Trong thực tế, thời gian thực hiện một thuật toán còn phụ thuộc rất nhiều vào các điều kiện khác như cấu tạo của máy tính, dữ liệu đưa vào, …, ở đây chúng ta chỉ xem xét trên mức độ của lượng dữ liệu đưa vào ban đầu cho thuật toán thực hiện.

Để ước lượng thời gian thực hiện thuật toán chúng ta có thể xem xét thời gian thực hiện thuật toán trong hai trường hợp:

- Trong trường hợp tốt nhất: T(min).
- Trong trường hợp xấu nhất: T(max).

Từ đó chúng ta có thể ước lượng thời gian thực hiện trung bình T(avg).

II. Các kiểu dữ liệu cơ bản

1. Khái niệm về kiểu dữ liệu

Kiểu dữ liệu T là sự hết hợp giữa 2 thành phần:

- Miền giá trị mà kiểu dữ liệu T có thể lưu trữ: V.
- Tập hợp các phép toán để thao tác dữ liệu: O. T = <V,O>

Mỗi kiểu dữ liệu thường được biểu diễn bằng một tên (biệt danh). Mỗi phần tử dữ liệu có kiểu T sẽ có giá trị trong miền V và có thể được thực hiện các phép toán thuộc tập hợp các phép toán trong O.

Để lưu trữ các phần tử dữ liệu này thường phải tốn một số byte(s) trong bộ nhớ, số byte(s) này gọi là kích thước của kiểu dữ liệu.

2. Các kiểu dữ liệu cơ sở

Hầu hết các ngôn ngữ lập trình đều có cung cấp các kiểu dữ liệu cơ sở. Tùy vào mỗi loại ngôn ngữ mà các kiểu dữ liệu cơ sở có các tên gọi khác nhau song chung quy lại có những loại kiểu dữ liệu cơ sở như sau:

TT	Kiểu dữ liệu (T)	Kích thước (V)	Các phép toán thực hiện (O)
1	Số nguyên	1 byte 2 bytes 4 bytes	{+, -, *, /, DIV, MOD, <, >, <=, >=, =, …}
2	Số thực	4 bytes 6 bytes 8 bytes 10 bytes	{+, -, *, /, <, >, <=, >=, =, …}
3	Ký tự	1 byte 2 bytes	{+, -, <, >, <=, >=, =, ORD, CHR, …}
4	Chuỗi ký tự	Tùy thuộc vào từng ngôn ngữ lập trình	{+, &, <, >, <=, >=, =, Length, Trunc, …}
5	Luận lý	1 byte	{NOT, AND, OR, XOR, <, >, <=, >=, =, …}

Một số kiểu dữ liệu cơ bản của ngôn ngữ lập trình C

TT	Kiểu dữ liệu (Type)	Kích thước (Length)	Miền giá trị (Range)

1	unsigned char	1 byte	0 đến 255
2	char	1 byte	– 128 đến 127
3	enum	2 bytes	– 32,768 đến 32,767
4	unsigned int	2 bytes	0 đến 65,535
5	short int	2 bytes	– 32,768 đến 32,767
6	int	2 bytes	– 32,768 đến 32,767
7	unsigned long	4 bytes	0 đến 4,294,967,295
8	long	4 bytes	– 2,147,483,648 đến 2,147,483,647
9	float	4 bytes	$3.4 * 10^{-38}$ đến $3.4 * 10^{38}$
10	double	8 bytes	$1.7 * 10^{-308}$ đến $1.7 * 10^{308}$
11	long double	10 bytes	$3.4 * 10^{-4932}$ đến $1.1 * 10^{4932}$

III. Các kiểu dữ liệu trừu tượng

1. Các kiểu dữ liệu có cấu trúc

Kiểu dữ liệu có cấu trúc là kiểu dữ liệu được xây dựng trên cơ sở các dữ liệu đã có (có thể là một kiểu dữ liệu có cấu trúc khác). Tùy vào từng ngôn ngữ lập trình, song thường có các loại sau:

- Kiểu mảng hay còn gọi là dãy: kích thước bằng tổng kích thước của các phần tử.
- Kiểu bảng ghi hay cấu trúc: kích thước bằng tổng kích thước thành phần (File).

2. Kiểu dữ liệu con trỏ

Các ngôn ngữ lập trình thường cung cấp cho chúng ta một kiểu dữ liệu đặt biệt để lưu trữ các địa chỉ của bộ nhớ, đó là con trỏ (Pointer).

3. Kiểu dữ liệu tập tin

Tập tin (File) có thể xem là một kiểu dữ liệu đặc biệt, kích thước tối đa của tập tin tùy thuộc vào không gian đĩa nơi lưu trữ tập tin.

IV. Các cấu trúc dữ liệu cơ bản

Có thể nói rằng không có chương trình máy tính nào mà không có dữ liệu để xử lý. Dữ liệu có thể à dữ liệu đưa vào (input data), dữ liệu trung gian hoạc dữ liệu đưa ra (output data). Do vậy việc tổ chức lưu trữ dữ liệu phục vụ cho chương trình có ý nghĩa rất quan trọng trong toàn bộ hệ thống chương trình. Việc xây dựng cấu trúc dữ liệu quyết định rất lớn đến chất lượng cũng như công sức của người lập trình trong việc thiết kế cài đặt chương trình.

V. Mối quan hệ của cấu trúc dữ liệu và giải thuật

Mối quan hệ giữa cấu trúc dữ liệu và giải thuật có thể minh họa bằng đẳng thức:

Cấu trúc dữ liệu + Giải thuật = Chương trình

Như vậy, khi đã có cấu trúc dữ liệu, nắm vững giải thuật thực hiện thì việc thể hiện chương trình bằng một ngôn ngữ cụ thể chỉ là vấn đề thời gian. Khi có cấu trúc dữ liệu mà

chưa tìm ra giải thuật thì không thể có chương trình và ngược lại không thể có thuật giải khi chưa có cấu trúc dữ liệu. Một chương trình máy tính chỉ có thể được hoàn thiện khi có đầy đủ Cấu trúc dữ liệu và Giải thuật xử lý dữ liệu bài toán theo yêu cầu đặt ra.

? Câu hỏi

1. Trình bày tầm quan trọng của Cấu trúc dữ liệu và Giải thuật đối với người lập trình?
2. Các tiêu chuẩn để đánh giá Cấu trúc dữ liệu và Giải thuật?
3. Khi xây dựng Giải thuật có cần thiết phải quan tân tới Cấu trúc dữ liệu hay không? Tại sao?
4. Liệt kê cá kiểu dữ liệu cơ sở, các kiểu dữ liệu có cấu trúc trong C, Pascal?

CHƯƠNG II:

ĐỆ QUI VÀ GIẢI THUẬT ĐỆ QUI

I. Khái niệm đệ qui

Bất cứ một hàm nào đó có thể triệu gọi hàm khác, nhưng ở đây một hàm nào đó có thể tự triệu gọi chính mình. Kiểu hàm như thế được gọi là hàm đệ qui. Vậy chương trình đệ qui là chương trình gọi đến chính nó.

Phương pháp đệ qui thường dùng phổ biến trong những ứng dụng mà cách giải quyết có thể được thể hiện bằng việc áp dụng liên tiếp cùng giải pháp cho những tập hợp con của bài toán.

Một chương trình đệ qui hoặc một định nghĩa đệ qui thì không thể gọi đến chính nó mãi mãi mà phải có một điểm dừng đến một trường hợp đặc biệt nào đó, mà ta gọi là trường hợp suy biến (degenerate case).

Ví dụ: Cho số tự nhiên n, ta định nghĩa n! như sau:

$$n! = \begin{cases} n*(n-1) \\ 0! = 1 \end{cases}$$

II. Giải thuật đệ qui và chương trình đệ qui

Phương pháp thiết kế một giải thuật đệ qui

a. Tham số hoá bài toán.
b. Phân tích trường hợp chung (đưa bài toán dưới dạng bài toán cùng loại nhưng có phạm vi giải quyết nhỏ hơn theo nghiã dần dần sẽ tiến đến trường hợp suy biến).
c. Tìm trường hợp suy biến.

Ví dụ 1: tính n!

n! = 1*2*3*…*(n-2)*(n-1)*n với n >= 1 và 0! = 1.

Viết hàm tính giai thừa không đệ qui

Chương trình	Kết qua in ra nàm hình

/* Ham tinh giai thua */
#include <stdio.h>
#include <conio.h>
void main(void)
{
int in;
long giaithua(int);
printf("Nhap vao so n: ");
scanf("%d", &in);
printf("%d! = %ld.\n", in, giaithua(in));
getch();
}
long giaithua(int in)
{
int i;
long ltich = 1;
if (in == 0)
return (1L);
else
{
for (i = 1; i <= in; i++)
ltich *= i;
return (ltich);
}
}
	Nhap vao so n: 5
	5! = 120.
	_

Viết hàm tính giai thừa theo đệ qui

Với n! = 1*2*3*…*(n-2)*(n-1)*n,

ta viết lại như sau: (1*2*3*…*(n-2)*(n-1))*n= n*(n-1)!…= n*(n-1)*(n-2)!…

```
/* Ham tinh giai thua */
long giaithua(int in)
{
int i;
if (in == 0)
return (1L);
else
return (in * giaithua(in – 1));
}
```

Giải thích hoạt động của hàm đệ quy giaithua

Ví dụ giá trị truyền vào hàm giaithua qua biến in = 5.

Thứ tự gọi thực hiện hàm giaithua

giaithua(in)	return(in * giaithua(in – 1))
5 4 3 2 1	5 * giaithua(4) = 5 * ? 4 * giaithua(3) = 4 * ? 3 * giaithua(2) = 3 * ? 2 * giaithua(1) = 2 * ? 1 * giaithua(0) = 1 * ?

Khi tham số in = 0 thì return về giá trị 1L (giá trị 1 kiểu long). Lúc này các giá trị? bắt đầu định trị theo thứ tự ngược lại.

Định trị theo thứ tự ngược lại

giaithua(in)	return(in * giaithua(in – 1))
1 2 3 4 5	1 * giaithua(0) = 1 * 1 = 1 2 * giaithua(1) = 2 * 1 = 2 3 * giaithua(2) = 3 * 2 = 6 4 * giaithua(3) = 4 * 6 = 24 5 * giaithua(4) = 5 * 24 = 120

III. Các bài toán đệ qui căn bản

1. **Đệ qui tuyến tính**: là một dạng đệ qui trực tiếp đơn giản nhất

Giải thuật P

Bắt đầu P

Nếu (thỏa điều kiện dừng) thì

Thực hiện lệnh S

Ngược lại

Bắt đầu

*Thực hiện các lệnh S**

Gọi lại P

Kết thúc

Kết thúc P

(S, S* là các thao tác, lệnh không đệ qui)

Ví dụ: Tính n!

```
int Giaithua(int n)
{
if (n==0)
return 1;
```

```
else
return n*Giaithua(n-1);
}
```

2. **Đệ qui nhị phân:** là một dạng đệ qui trực tiếp có dạng

Giải thuật P

Bắt đầu P

Nếu (thỏa điều kiện dừng) thì

Thực hiện lệnh S

Ngược lại

Bắt đầu

*Thực hiện các lệnh S**

Gọi lại P

Gọi lại P

Kết thúc

Kết thúc P

(S, S* là các thao tác, lệnh không đệ qui)

Ví dụ:

```
int Fib(int n) {
if (n==0) return 0; // mốc
else
if (n==1) return 1; // mốc
else
return Fib(n-1) + Fib(n-2);
}
```

3. **Đệ qui phi tuyến:** là một dạng đệ qui trực tiếp mà lời gọi đệ qui được thực hiện bên trong vòng lặp.

Giải thuật P

Bắt đầu P

For <giá trị đầu> to <giá trị cuối> do

Đầu lặp

Thực hiện lệnh S

Nếu (thỏa điều kiện dừng) thì

*Thực hiện các lệnh S**

Ngược lại

Gọi lại P

Cuối lặp

Kết thúc P

(S, S* là các thao tác, lệnh không đệ qui)

Ví dụ: Cho dãy $\{X_n\}$ xác định theo công thức truy hồi

```
int X_n(int n) {
int kq = 0;
if (n==0) return 1
else {
kq = 0;
for (int i= 0;i<n;i++) {
kq = kq + (n-i) * (n-i) * X(i);
}
return kq;
}
}
```

? Bài tập

1. Viết hàm đệ quy tính dãy Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, … Bắt đầu bằng 0 và 1, các số tiếp theo bằng tổng hai số đi trước.
2. Viết hàm đệ quy tính tổng n số nguyên dương đầu tiên: tong (n) = n + tong (n – 1).
3. Trình bày các dạng giải thuật đệ qui và cho ví dụ minh họa.
4. Cài đặt chương trình tìm USCLN theo thuật toán Euclide bằng pp đệ qui và không đệ qui.
5. Cài đặt chương trình minh họa thuật toán tìm kiếm nhị phân đệ qui và không đệ qui
6. Cho ma trận kích thước 3x3. Điền vào mỗi ô trên ma trận sao cho tổng trên mỗi dòng là số nguyên tố. Viết chương trình liệt kê tất cả các trường hợp có thể dùng đệ qui và không đệ qui.
7. Tìm hiểu về thuật toán quay lui và thuật toán nhánh cận
8. (*) Cài đặt chương trình minh họa bài toán Tháp Hà Nội bằng giải thuật đệ qui và không đệ qui.

9. (*) Cài đặt giải thuật tìm đường đi cho con kiến theo yêu cầu bài toán mô tả tại

CHƯƠNG III:

DANH SÁCH

I. Danh sách và các phép toán cơ bản trên danh sách

Có 2 loại danh sách là danh sách tuyến tính và danh sách liên kết.

1. Danh sách tuyến tính và các phép toán cơ bản

Khái niệm

Danh sách là một dãy hữu hạn các phần tửthuộc cùng một lớp các đối tượng nào đó.

Danh sách tuyến tính là các phần tử của nóđược sắp tuyến tính: nếu n>1 thì phần a_i ở trước phần tử a_i+1

Với a_i là phần tử ở vị trí thứ i của danh sách.

Gọi L là danh sách có n (n>=0) phần tử.

- L = (a_1,a_2, …,a_n)

Gọi n là độ dài của danh sách.

- Nếu n = 0:
  - L là danh sách rỗng
- Nếu n>=1
  - a_1 là phần tử đầu tiên của danh sách
  - a_n là phần tử cuối cùng của danh sách

Một đối tượng có thể xuất hiện nhiều lầntrong một danh sách.

VD:

– 1, 1, 2, 3, 5,8,13,21,34,55,89 là danh sách các số fibonacci

– (31,28,31,30,31,30,31,31,30,31,30,31) là danh sách số ngày của các tháng trong một năm

Các phép toán cơ bản

- Khởi tạo danh sách
- Xác định độ dài của danh sách
- Loại bỏ phần tử ở vị trí thứ p trong danh sách
- Xen phần tử x vào danh sách sau vị trí p
- Xen phần tử x vào danh sách trước vị trí p
- Tìm kiếm trong danh sách có chứa phần tử x hay không?
- Kiểm tra danh sách có đầy hay không?
- Kiểm tra danh sách có rỗng hay không?
- Duyệt danh sách

2. Danh sách liên kết và các phép toán cơ bản

Khái niệm

Các phần tử của danh sách gọi là node, nằm rải rác trong bộ nhớ. Mỗi node, ngoài vùng dữ liệu thông thường, còn có vùng liên kết chứa địa chỉ của node kế tiếp hay node trước đó.

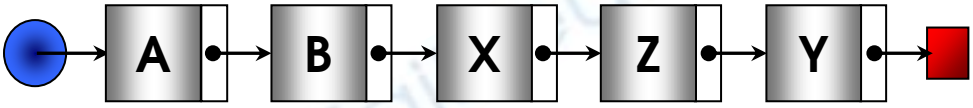
Danh sách liên kết là cấu trúc dữ liệu động, có thể thêm hay hủy node của danh sách trong khi chạy chương trình. Với cách cài đặt các thao tác thêm hay hủy node ta cần thay đổi lại vùng liên kết cho phù hợp.

Tuy nhiên, việc lưu trữ danh sách liên kết tốn bộ nhớ hơn danh sách kề vì mỗi node của danh sách phải chứa thêm vùng liên kết. Ngoài ra việc truy xuất node thứ i trong danh sách liên kết chậm hơn vì phải duyệt từ dầu danh sách.

Có nhiều kiểu tổ chức liên kết giữa các phần tử trong danh sách như :

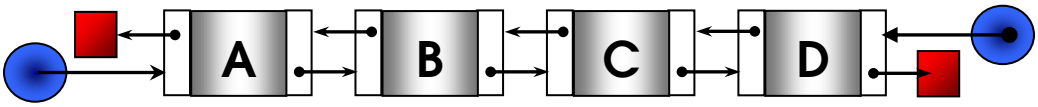
- Danh sách liên kết đơn
- Danh sách liên kết kép
- Danh sách liên kết vòng
- …

Danh sách liên kết đơn



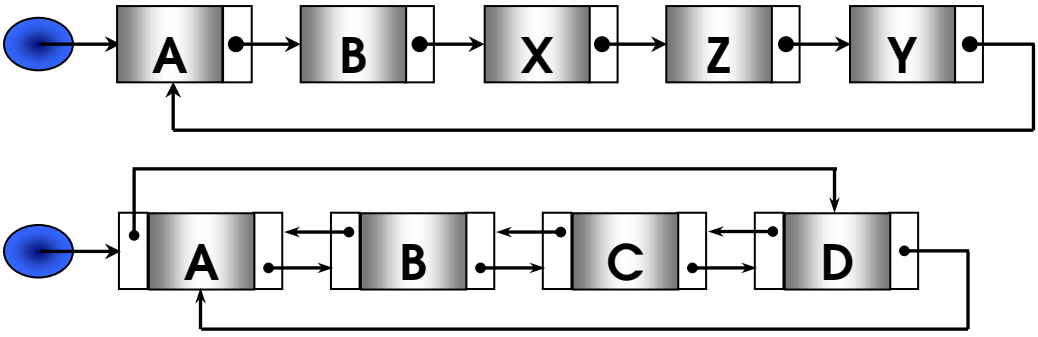
Danh sách liên kết kép

Mỗi phần tử liên kết với các phần tử đứng trước và sau nó trong danh sách



Danh sách liên kết vòng :

Phần tử cuối danh sách liên kết với phần tử đầu danh sách



Các phép toán cơ bản

- Khởi động vùng lưu trữ tự do
- Lấy một node tự do
- Xoá một node

- Tạo danh sách liên kết rỗng
- Kiểm tra danh sách có rỗng hay không?
- Kiểm tra danh sách có đầy không?
- Chèn, Xoá, tìm kiếm
- Duyệt danh sách.

II. Cài đặt danh sách theo cấu trúc mảng

Sử dụng mảng để lưu trữ các phần tử củadanh sách, trong đó mỗi thành phần củamảng sẽ lưu giữ một phần tử của danh sách.

Các phần tử liên tiếp nhau của danh sáchđược lưu trữ trong các thành phần liên tiếpnhau của mảng

```
Const max =N;
 struct List{
Item element[max];
longcount;
} L;
```

Max chiều dài tối đa của danh sách

Xây dựng cấu trúc List gồm 2 thành phần

– element là một mảng các phần tử thuộckiểu dữ liệu item

– count lưu chỉ số của thành phần mảng lưugiữ phần tử cuối cùng của danh sách

1. Khởi tạo danh sách rỗng

Khởi tạo danh sách rỗng, ta gán giá trị count = 0

```
void Init(List &L)
{
        L.count = 0;
}
```

2. Xác định số phần tử của danh sách

Xây dựng hàm Length:

– Input : Danh sách L

– Output: Số phần tử của danh sách

```
long Length(List L)
{
        return L.count;
}
```

3. Kiểm tra danh sách đầy

Xây dựng hàm isFull:

– Input : Danh sách L

– Output:

1: Danh sách đầy

0: Danh sách chưa đầy

```
int isFull(List L)
{
      if (L.count = = max)
            return 1;
```

```
        else return 0;
    }
```

4. Kiểm tra danh sách rỗng

Xây dựng hàm isEmpty:

– Input : Danh sách L

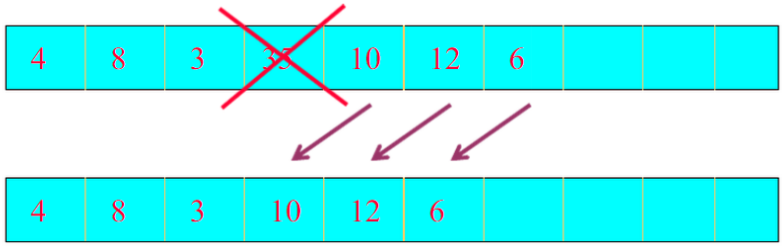
– Output:

1: Danh sách rỗng

0: Còn phần tử trong danh sách

```
int isEmpty(List L)
{
  if (L.count = = 0)
        return 1;
  else return 0;
}
```

5. Loại bỏ phần tử ở vị trí p trong danh sách



Thuật toán:

– Bước 1:Thực hiện kiểm tra nếu danh sáchkhông rỗng và p chỉ vào một phần tử trong danh sách thì thực hiện bước 2,ngược lại thì dừng.

– Bước 2: Ta tịnh tiến các phần tử ở các vịtrí p+1, p+2,…đến các vị trí p,p+1…(tịnh tiến lên một vị trí)

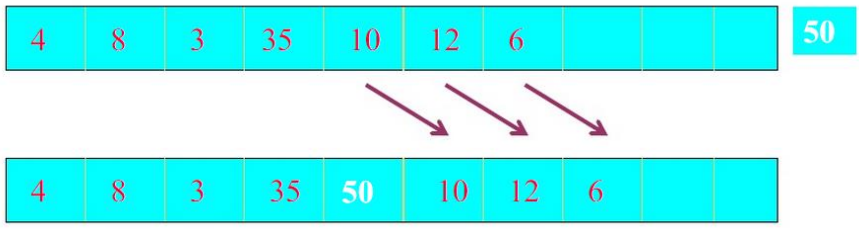
Xây dựng hàm Delete

- Input:
  - Danh sách L
  - Vị trị phần tử cần xoá p
  - Tham biến q cho biết việc xoá có thành công hay không?
- Output:
  - q =1 nếu việc xoá thành công
  - q = 0 nếu việc xoá không thành công

```
void Delete(List &L, long p, int &q)
    {
        int i;
        q = 0;
        if (p<=L.count)
            {
                i = p;
```

```
            while(i<L.count)
                {
                    L.element[i] = L.element[i+1];i++;
                }
            L.count =L.count -1;
            q = 1;
            }
        }
```

6. Chèn phần tử X vào danh sách sau vị trí p



Thuật toán:

– Bước 1:Thực hiện kiểm tra nếu danh sáchchưa đầy và p chỉ vào một phần tử trong danh sách thì thực hiện bước 2, ngược lạithì dừng.

– Bước 2: Ta tịnh tiến các phần tử ở các vị trí L.count, L.count -1,…đến các vị tríL.count +1,L.count…

– Bước 3: Chèn phần tử x vào vị trí p

Xây dựng hàm InsertAfter

– Input:

- Danh sách L
- Vị trị p
- Phần tử x thuộc kiểu Item cần chèn
- Tham biến q cho biết việc chèn có thànhcông hay không?

– Output:

- q =1 nếu việc chèn thành công
- q = 0 nếu việc chèn không thành công

```
void InsertAfter(List &L, long p,Item x, int &q)
    {
        int i;
        q = 0;
        if (!isFull(L) && (p <= L.count))
            {
                i = L.count+1;
                while(i >=p+1)
                    {
                    L.element[i] = L.element[i-1];
                    i--;
                    }
```

```
                    L.element[p] = x;
                    L.count = L.count +1;
                    q =1;
                }
}
```

7. Chèn phần tử X vào danh sách trước vị trí p

Thuật toán:

– Bước 1:Thực hiện kiểm tra nếu danh sáchchưa đầy và p chỉ vào một phần tử trongdanh sách thì thực hiện bước 2, ngược lạithì dừng.

– Bước 2: Ta tịnh tiến các phần tử ở các vị trí L.count, L.count -1,…đến các vị tríL.count +1,L.count…

– Bước 3: Chèn phần tử x vào vị trí p

Xây dựng hàm InsertBefore

– Input:

- Danh sách L
- Vị trị p
- Phần tử x thuộc kiểu Item cần chèn
- Tham biến q cho biết việc chèn có thành công hay không?

– Output:

- q =1 nếu việc chèn thành công
- q = 0 nếu việc chèn không thành công

```
void InsertBefore(List &L, long p,Item x, int &q)
  {
     int i;
     q = 0;
     if (!isFull(L) && (p <= L.count))
       {
            i = L.count+1;
            while(i >p)
                {
                    L.element[i] = L.element[i-1];
                    i--;
                }
            L.element[p] = x;
            L.count = L.count +1;
            q =1;
       }
  }
```

8. Tìm kiếm phần tử x trong danh sách

Ta sử dụng phương pháp tìm kiếm tuầntự xây dựng hàm Search

– Input:

- Danh sách L