KỸ THUẬT LẬP TRÌNH C

Chương 3:
Danh sách kề

1. Giới thiệu danh sách
2. Danh sách kề
3. Ưu khuyết điểm của danh sách kề

1. Giới thiệu danh sách

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

Ví dụ: Bảng sau mô tả một bảng danh sách sinh viên, mỗi dòng của bảng là một nút 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

2. Danh sách kề

a. Giới thiệu

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

b. Cài đặt danh sách kề

- Khai báo cấu trúc danh sách kề

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

- **Các thao tác trên danh sách kề**
  - Khởi động danh sách kề:

```
void Initialize(LIST *list) {
        list->num = 0;
}
```

  - Xác định số nút trong danh sách:

```
int ListSize(LIST *list) {
        return list->num;
}
```

- Kiểm tra danh sách có bị rỗng hay không

```
int Empty(LIST *list) {
        return (list->num == 0 ? 1 : 0);
}
```

- Kiểm tra danh sách có bị đầy hay không

```
int Full(LIST *list) {
        return (list->num == MAX ? 1 : 0);
}
```

- Thêm một nút vào danh sách

Giả sử chúng ta cần thêm một nút x vào danh sách tại vị trí pos

Điều kiện: 0 <= pos <= num

khi đó các nút từ vị trí num – 1 lên pos bị dời xuống một vị trí để chèn x vào vị trí pos.

```
void Insert(LIST *list, int pos, int x) {
    int i;
    if(pos < 0 || pos > list->num)
        printf("Vi tri %d không hop le\n", pos);
    else if(Full(list))
        printf("Danh sach bi day\n");
    else {
        //Dời các nút từ vt num-1 lên vt pos xuống một vt
        for(i = list->num – 1; i >= pos; i--)
            list->nodes[i + 1] = list->nodes[i];
        list->nodes[pos] = x;          //Thêm x vào vị trí pos
        list->num++;   //Tăng số nút lên 1
    }
}
```

- Xóa một nút khỏi danh sách

Thao tác này trả về nội dung nút bị xóa.

Giả sử chúng ta cần xóa nút tại vị trí pos
Điều kiện: 0 <= pos <= num-1

khi đó các nút từ vị trí pos + 1 xuống num – 1 dời lên một vị trí.

```
int Remove(LIST *list, int pos) {
        int i, x;
        if(pos < 0 || pos >= list->num)
            printf("Vi tri %d không hop le\n", pos);
        else if(Empty(list))
            printf("Danh sach bi rong\n");
        else {
            x = list->nodes[pos];
            //Dời các nút từ vt pos+1 xuống vt num–1 lên 1 vt
            for(i = pos; i < list->num – 1; i++)
                        list->nodes[i] = list->nodes[i + 1];
            list->num--;    //Giảm số nút xuống 1
            return x;
        }
}
```

- Thay nút trong danh sách bằng một nút khác

Giả sử chúng ta cần thay nút tại vị trí pos bằng một nút có nội dung là x

Điều kiện: 0 <= pos <= num-1.

```
void Replace(LIST *list, int pos, int x) {
        if(pos < 0 || pos >= list->num)
                printf("Vi tri %d không hop le\n", pos);
        else if(Empty(list))
                printf("Danh sach bi rong\n");
        else  list->nodes[pos] = x;
}
```

- Sắp xếp danh sách theo thứ tự tăng dần

```
void Sort(LIST *list) {
    int i, j;
    for(i = 0; i < list->num – 1; i++)
        for( j = i + 1; j < list->num; j++)
            if(list->nodes[i] > list->nodes[j]) {
                int tmp = list->nodes[i];
                list->nodes[i]=list->nodes[j];
                list->nodes[j] = tmp;
            }
}
```

- Tìm kiếm một nút

Giả sử chúng ta muốn tìm một nút có nội dung là x, nếu tìm thấy thao tác sẽ trả về vị trí tìm thấy, ngược lại trả về giá trị -1.

Ở đây chúng ta trình bày hai phương pháp tìm kiếm: *tìm kiếm tuyến tính* và *tìm kiếm nhị phân*.

```
//Tìm kiếm tuyến tính
void LinearSearch(LIST *list, int x) {
	int i;
	for(i = 0; i < list->num; i++) {
		if(list->nodes[i] == x)
			return i; //Tìm thấy tại vị trí i
	}
	return -1;  //Không tìm thấy
}
```

```
//Tìm kiếm nhị phân trên danh sách đã có thứ tự tăng
void BinarySearch(LIST *list, int x) {
	int dau = 0;
	int cuoi = list->num – 1 ;
	int giua ;
	while(dau <= cuoi) {
		giua = (dau + cuoi) / 2 ;
		if(x == list->node[giua])
			return giua; //Tìm thấy tại vị trí giữa
		if(x < list->nodes[giua])
			cuoi = giua – 1;
		else  dau = giua + 1;
	}
	return -1 ; //Không tìm thấy
}
```

- Duyệt tất cả các nút của danh sách

```
void Traverse(LIST *list) {
    if(Empty(list))
        printf("Danh sach rong\n");
    else {
        int i;
        for(i = 0; i < list->num – 1; i++)
            printf("%d\t", list->node[i]);
        printf("\n");
    }
}
```

- Xoá tất cả các nút trong danh sách

```
void ClearList(LIST *list) {
        list->num = 0;
}
```

3. Ưu khuyết điểm của danh sách kề

a. Ưu điểm

- Mật độ sử dụng bộ nhớ của danh sách kề bằng 1, nghĩa là không bị lãng phí bộ nhớ
- Thao tác truy xuất đến nút thứ i rất nhanh
- Thao tác tìm kiếm một nút trong trường hợp danh sách đã được sắp thứ tự bằng phương pháp tìm kiếm nhị phân sẽ rất nhanh.

b. Khuyết điểm

- Vì số nút tối đa cấp phát cho danh sách kề là cố định khi chương trình đang chạy nên số nút cần dùng lúc thừa lúc thiếu
- Danh sách kề không phù hợp với các thao tác thêm nút và xoá nút. Thời gian thực hiện các thao tác này phụ thuộc vào số nút của danh sách.

Hết