

HƯỚNG DẪN CODE DUYỆT THEO CHIỀU RỘNG (BFS)

Chú ý: Trong hướng dẫn này, chỗ nào có cụm từ “bạn viết code” hay đại loại thế. Thì bạn phải viết code chỗ đó hén.

Đề bài: Bạn có một đồ thị g (gồm đỉnh n và ma trận kề a), bạn muốn tìm đường đi từ một đỉnh S (Start) đến một đỉnh F (Finish) trong đồ thị đó. Bạn hãy viết chương trình tìm đường đi có thể theo thuật toán duyệt theo chiều rộng (BFS). Nếu tìm có đường đi từ đỉnh S (Start) đến đỉnh F (Finish) trong đồ thị g này thì bạn xuất ra đường đi, còn nếu không có đường đi thì bạn thông báo không có đường đi từ đỉnh S (Start) đến đỉnh F (Finish).

Hướng dẫn Code:

Khi bạn làm tới phần này, thì bạn **đã làm được việc đọc thông tin** của đồ thị từ một file nào đó vào chương trình của bạn rồi hén. Nếu bạn vẫn chưa làm được điều này thì đề nghị bạn mở lại file “**HƯỚNG DẪN CODE NHẬP XUẤT MA TRẬN KÊ TỪ FILE**” đọc và làm nhé. Còn nếu bạn đã làm được rồi thì chúng ta tiếp tục hén **J**

Nhắc lại: Thông tin đồ thị của bạn sẽ được lưu trữ trong chương trình thông qua một cấu trúc như sau đúng không?

```
#define MAX 10 // định nghĩa giá trị MAX
#define inputfile "C:/test.txt" // định nghĩa đường dẫn tuyệt đối đến file chứa thông tin của đồ thị
typedef struct GRAPH {
    int n; // số đỉnh của đồ thị
    int a[MAX][MAX]; // ma trận kề của đồ thị
}DOTHI;
```

Bước 1: Tạo 1 mảng 1 chiều **LuuVet** dùng để lưu vết đường đi từ S à F, và 1 mảng 1 chiều khác với tên là **ChuaXet** dùng để đánh dấu đỉnh nào trong đồ thị đã xét rồi, đỉnh nào chưa xét trong quá trình tìm đường đi từ S à F.

```
int LuuVet[MAX]; // LuuVet[i] = đỉnh liền trước i trên đường đi từ S à i
```

```
int ChuaXet[MAX]; // ChuaXet[i] = 0 là đỉnh i chưa được xét đến trong quá trình tìm đường đi, còn ChuaXet[i] = 1 là đỉnh i được xét đến rồi trong quá trình tìm đường đi.
```

Bước 2: Hướng dẫn code phần duyệt theo chiều rộng (BFS): cài đặt thuật toán không dùng đệ qui.

Đầu tiên tạo một cấu trúc hàng đợi như sau dùng để lưu thứ tự danh sách các đỉnh cần được duyệt.

```
struct QUEUE
```

```
{
```

```
    int size; // kích thước hiện tại hay số phần tử có trong hàng đợi
```

```
    int array[MAX]; //mảng lưu các giá trị trong hàng đợi
```

```
};
```

Để khởi tạo một hàng đợi thì dùng hàm **KhoiTaoQueue** như sau:

```
void KhoiTaoQueue(QUEUE &Q)
```

```
{
```

```
    Q.size = 0; // vì ban đầu hàng đợi rỗng nên size của nó là 0
```

```
}
```

Để đẩy một giá trị vào hàng đợi, ta dùng hàm **DayGiaTriVaoQueue** như sau:

```
int DayGiaTriVaoQueue(QUEUE &Q,int value)
```

```
{
```

```
    if(Q.size + 1 >= 100) // nếu hàng đợi đã đầy rồi thì không thể đẩy thêm giá trị vào hàng đợi được nữa
```

```
return 0; // trả về kết quả 0 báo cho người lập trình biết là không thể đẩy  
thêm giá trị vào hàng đợi được nữa.
```

```
Q.array[Q.size] = value; // đẩy giá trị vào hàng đợi.
```

```
Q.size++; // tăng số lượng phần tử trong hàng đợi
```

```
return 1; // trả về kết quả 1 báo cho người lập trình biết là đã đẩy thêm giá trị vào  
hàng đợi thành công.
```

```
}
```

Để lấy một giá trị ra từ hàm đợi ta dùng hàm **LayGiaTriRaKhoiQueue** như sau:

```
int LayGiaTriRaKhoiQueue(QUEUE &Q,int &value)
```

```
{
```

```
    if(Q.size <= 0) // nếu hàng đợi rỗng thì không thể lấy ra giá trị nào được
```

```
        return 0; // trả về kết quả 0 báo cho người lập trình biết là hàng đợi rỗng,  
không thể lấy giá trị nào
```

```
    value = Q.array[0]; // lấy giá trị đầu tiên trong hàng đợi.
```

```
    for(int i = 0; i < Q.size - 1 ; i++) // tiến hành loại bỏ giá trị đầu tiên ra khỏi hàng  
    đợi
```

```
        Q.array[i] = Q.array[i+1];
```

```
    Q.size--; // giảm kích thước hàng đợi vì đã lấy ra 1 giá trị hay phần tử
```

```
    return 1; // trả về kết quả 1 báo cho người lập trình biết là đã lấy giá trị ra khỏi  
hàng đợi thành công.
```

```
}
```

Để kiểm tra hàm đợi có rỗng hay không, ta dùng hàm **KiemTraQueueRong** như sau:

```
int KiemTraQueueRong(QUEUE Q)
```

```
{
```

```
    if(Q.size <= 0) // nếu hàm đợi rỗng thì trả về kết quả 1, ngược lại không rỗng là 0
```

```
        return 1;
```

```
    return 0;
}
```

Để tiến hành duyệt BFS tại một đỉnh v nào đó của đồ thị g , ta dùng hàm **BFS** sau:

```
void BFS(int v, GRAPH g)
{
    QUEUE Q;
    KhoiTaoQueue(Q); // khởi tạo hàng đợi vì ban đầu thuật toán hàng đợi ta chưa có
    gì
    DayGiaTriVaoQueue(Q,v); // đẩy đỉnh v vào hàng đợi theo như thuật toán đã trình
    bày
    while(!KiemTraQueueRong(Q)) // trong khi hàng đợi chưa rỗng thì tiến hành các
    bước sau
    {
        LayGiaTriRaKhoiQueue(Q,v); //lấy phần tử đầu tiên trong hàng đợi ra và
        gán giá trị đó vào v
        ChuaXet[v] = 1; //bắt đầu xét đỉnh v nên nhãn ChuaXet của đỉnh v được
        gán là 1, tức đang xét hoặc đã xét trong thuật toán BFS
        for(int u = 0; u < g.n ; u++)
        {
            if(g.a[v][u] != 0 && ChuaXet[u] == 0) // Xem xét có cạnh nào nối
            từ đỉnh v đến đỉnh u trong đồ thị g không (điều này tương ứng với g.a[v][u] != 0) và đỉnh
            u đã được xét hay duyệt đến hay chưa? Nếu có cạnh nối từ đỉnh v đến đỉnh u và đỉnh u
            chưa được xét hay duyệt đến thì tiến hành như sau
            {
                DayGiaTriVaoQueue(Q,u); // đẩy đỉnh u vào hàng đợi theo
                như thuật toán đã trình bày
            }
        }
    }
}
```

Duyệt theo BFS

```
        LuuVet[u] = v; // đánh dấu lại đỉnh u được đi đến từ đỉnh v
    }
    }
}
```

Trước khi tiến hành duyệt BFS để tìm đường đi S → F thì cần phải khởi tạo các giá trị thích hợp cho các mảng LuuVet và ChuaXet. Quá trình đó như sau

```
void duyettheoBFS (int S, int F, GRAPH g) // hàm này dùng để tìm đường đi từ S → F
trong đồ thị g theo thuật toán BFS
```

```
{
    // khởi tạo lại các giá trị thích hợp cho mảng LuuVet và ChuaXet
    /* các bạn tự viết code khởi tạo các giá trị cho mảng LuuVet và ChuaXet nhé. Gợi
    ý vì ban đầu chưa chạy thuật toán nên các đỉnh i đều chưa có vết đi đến đó nên đặt giá trị
    -1. Và ban đầu chưa chạy thuật toán nên đỉnh i trong đồ thị g đều chưa được xét nên giá
    trị là 0 */
    BFS(S,g); // tiến hành thuật toán BFS
    // xuất đường đi
    if (ChuaXet[F] == 1) // sau khi thuật toán duyệt xong mà đỉnh F được xét hay
    duyệt đến thì nhãn của nó = 1 điều này có nghĩa là có đường đi từ S → F. Tiến hành xuất
    đường đi.
    {
        printf("Duong di tu dinh %d den dinh %d la: \n\t",S,F);
        i = F;
        printf("%d ", F);
    }
}
```

/ các bạn tự viết code xuất ra đường đi từ F đến S hén. Gợi ý dựa vào mảng LuuVet để tiến hành truy vết ra đường đi từ S đến F*/*

```
    }  
    Else // sau khi thuật toán duyệt xong mà đỉnh F chưa được xét hay duyệt đến thì  
    nhãn của nó = 0 điều này có nghĩa là không có đường đi từ S đến F. Thông báo không có  
    đường đi từ S đến F  
    {  
        printf("Khong co duong di tu dinh %d den dinh %d \n",S,F);  
    }  
}
```

Bước 3: Code trong hàm main để gọi hàm các hàm tương ứng và chạy. Có thể làm như sau:

```
void main()  
{  
    DOTH1 g;  
    clrscr();  
    if (DocMaTranKe(inputfile, g) == 1)  
    {  
        printf("Da lay thong tin do thi tu file thanh cong.\n\n");  
        XuatMaTranKe(g);  
        printf("Bam 1 phim bat ki de bat dau duyet theo BFS ...\n\n");  
        getch();  
        duyetteoBFS(0,2,g);  
    }  
    getch();  
}
```

Chúc các bạn may mắn và học tốt môn này

GOOD LUCK TO U

-----HẾT-----