MỘT SỐ HƯỚNG DẪN VỀ ĐỒ THỊ

Đồ thị là kiểu dữ liệu có nhiều ứng dụng, chúng được sử dụng trong các bài toán cần thể hiện mối quan hệ giữa các thực thể chẳng hạn các mạng đường sắt, mạng xã hội.

I. BIỂU DIỄN ĐỒ THỊ

Khi số lượng đỉnh lớn, cách biểu diễn đồ thị phổ biến là danh sách kề. Ở đó mỗi nút có thể coi là một số nguyên, và người ta duy trì một mảng mà mỗi phần tử là 1 danh sách liên kết chỉ đến các nút kề của nút ở chỉ số mảng đó (xem lại bài giảng về biểu diễn đồ thị).

Khai báo có thể như sau (giả sử trọng số là một số nguyên):

```
struct Node{
     int vertex; // Đỉnh nối với đỉnh đó
     int weight; // Trong số cạnh
     struct Node* Next;
};
typedef struct Node* AdjList[MaxSize+1];
```

Hàm khởi tạo một danh sách kề được chỉ bởi con trỏ pL cho đồ thị có n nút như sau:

```
init(*pL, n){
     FOR i=1 TO n
          makenull(pL[i]);
}
```

Phép toán chèn cạnh (u, v) có trọng số w vào danh sách kề L:

```
insertNode(L,u,v,w){
     Chèn (v, w) vào đầu danh sách L[u] bằng cách:
     - Cấp phát một vùng nhớ mới chỉ bởi p
     - p->vertex =v
     - p->weight=w
     - p->Next=L[u]->Next
     - L[u]->Next=p
}
```

Liệt kê các đỉnh kề của đỉnh u của danh sách kề L:

```
p = L[u]
WHILE (p->Next != NULL)
     + Xử lý tại vị trí p
     + p=p->Next
```

Kiểm tra 2 đỉnh u và v của danh sách kề có kề nhau hay không?

```
p = L[u]
WHILE (p->Next != NULL)
     + IF (p->Next->vertex == v) return 1
     + p=p->Next
return 0
```

II. GIẢI THUẬT DIJSKTRA TÌM ĐƯỜNG ĐI NGẮN NHẤT

Giải thuật Dijsktra được dùng để tìm đường đi ngắn nhất từ một nút xuất phát đến các nút còn lại của đồ thị có trọng số dương.

Ý tưởng của giải thuật dựa trên kỹ thuật tham ăn. Giả sử chiều dài đường đi từ đỉnh bắt đầu tới đỉnh i là d[i]. Mục tiêu là tìm giá trị d[i] là nhỏ nhất.

Ban đầu đặt d[0]=0, còn các độ dài đến các nút còn lại là ∞.

Kế tiếp là các bước lặp, mỗi bước chọn đỉnh có độ dài đường đi có độ dài từ nút bắt đầu tới nút đó là nhỏ nhất, rồi tiến hành "RELAXATION", tức là cập nhật lại độ dài đường đi đến các đỉnh kề của nút đó.

Giả sử nút u có d[u] ngắn nhất được chọn, v là đỉnh kề của u, w[u,v] là trọng số của cạnh (u, v). RELAXATION tiến hành như sau:

IF (d[u] + w[u,v] < d[v])

d[v] = d[u] + w[u,v]

Lặp đi lặp lại tới khi không còn chọn được đỉnh nào để cập nhật nữa.

II.1 Hàng ưu tiên cho giải thuật Dijsktra

Do mỗi bước lặp, giải thuật chọn đỉnh có độ dài đường đi nhỏ nhất, nên hàng ưu tiên với cài đặt bằng min heap là giải pháp phù hợp cho giải thuật Dijsktra.

Mỗi phần tử của hàng ưu tiên là một cấu trúc gồm 2 trường: đỉnh v và độ dài đường đi tới đỉnh v – đây là khóa của hàng ưu tiên.

Có thể khai báo hàng ưu tiên trong trường hợp này như sau:

```
typedef struct{
     int Vertex;
     int Key;
} KeyType;
typedef struct{
     KeyType *Keys;
     int size;
}PriorityQueue;
```

Sau đó cần cài đặt các phép toán cơ bản trên hàng ưu tiên như: extractMin(), insert(), …

II.2 Giải thuật Dijsktra tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại

d[1]=0;

Chèn bộ (0, 1) vào hàng ưu tiên Q (nút 1 có độ dài đường đi là 0)

WHILE (Q!=ϕ){

Lấy 1 đỉnh u từ hàng ưu tiên extractMin(&Q);

IF (chưa thăm u){

- Đánh dấu thăm đỉnh u
- Duyệt qua các đỉnh kề v của u
  - IF (d[u]+w[u,v] < d[v])
    - d[v] = d[u]+w[u,v]
    - Chèn v với độ dài đường đi d[v] vào Q

➔ Mảng d chứa độ dài đường đi cần tìm

}

Độ phức tạp của giải thuật là ~ O(mlogn) với n là số đỉnh, m là số cạnh**.**

III. GIẢI THUẬT KRUSKAL TÌM CÂY KHUNG NHỎ NHẤT

Giải thuật Kruskal để tìm cây khung nhỏ nhất của đồ thị cũng là một giải thuật theo tiếp cận tham ăn. Ở đó các cạnh được sắp thứ tự tăng dần theo độ dài cạnh, quá trình cứ lặp đi lặp lại việc lấy 1 cạnh từ danh sách trên, kiểm tra xem khi thêm cạnh được lấy vào danh sách kết quả thì có tạo thành chu trình hay không? Nếu không thì đưa cạnh đó vào danh sách kết quả, còn có thì bỏ qua và không bao giờ xem lại cạnh đó nữa.

III.1 Disjoint Set

Để kiểm tra đồ thị có chu trình hay không, kiểu Disjoint Set được sử dụng (xem bài giảng Disjoint Set).

Ta khai báo 2 mảng parent[n+1] và rank[n+1] để cài đặt Disjoint Set với thao tác collapsing find và tính bậc của mỗi đỉnh.

Khởi tạo Disjoint Set

```
FOR i=0 TO n
+ parent[i]=i
+ rank[i]=0
```

find(x):

```
IF (parent[i] != i)
          parent[i] = find(parent[i])
return parent[i]
```

Union(x, y):

r1 = find(x)

r2 = find(y)

IF (rank[r1] < rank[r2])

parent[r1] = r2

ELSE IF (rank[r1] > rank[r2])

parent[r2] = r1

ELSE

- parent[r2] = r1;
- rank[r1]++;

III.2 Giải thuật Kruskal

```
Khởi tạo Disjoint Set
Sắp xếp các cạnh tăng dần theo độ dài cạnh
i=j=0
WHILE (j != n-2 AND i!=m-1){
     e =Cạnh thứ i;
     x = find(e.start);
     y = find(e.end);
```

```
      IF (x != y)  //Khong tao chu trinh
      -  w += e.weight
      -  Lưu e vào kết quả ở vị trí j
      -  Union(x, y)
      -  j++
      i++;
}
```

Độ phức tạp của giải thuật Kruskal là O(mlogm), với m là số cạnh của đồ thị.