Đồ thị có trọng số

Đồ thị có trọng số

- Chúng ta có thể thêm thuộc tính vào cạnh. Ta gọi thuộc tính là trọng số.
  - Ví dụ, nếu ta sử dụng đồ thị như một bản đồ trong đó các đỉnh là các thành phố và các cạnh là đường cao tốc nối các thành phố
  - Khi đó nếu ta muốn khoảng cách di chuyển ngắn nhất giữa các thành phố, một trọng số phù hợp là khoảng cách giữa các thành phố.
  - Nếu ta quan tâm đến chi phí của một chuyến đi và sẽ đi chuyến có chi phí thấp nhất thì trọng số phù hợp là chi phí đi lại giữa các thành phố.

Đường đi ngắn nhất

- Đồ thị $G = (V,E)$ với hàm trọng số $W: E \rightarrow R$ (gán giá trị thực cho các cạnh)
- Trọng số của đường đi $p = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ là

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

- Đường đi ngắn nhất là đường đi với trọng số nhỏ nhất
- Ứng dụng:
  - Định tuyến mạng tĩnh/động
  - Lập kế hoạch chuyển động của robot
  - Gợi ý đường đi/bản đồ giao thông

Bài toán đường đi ngắn nhất

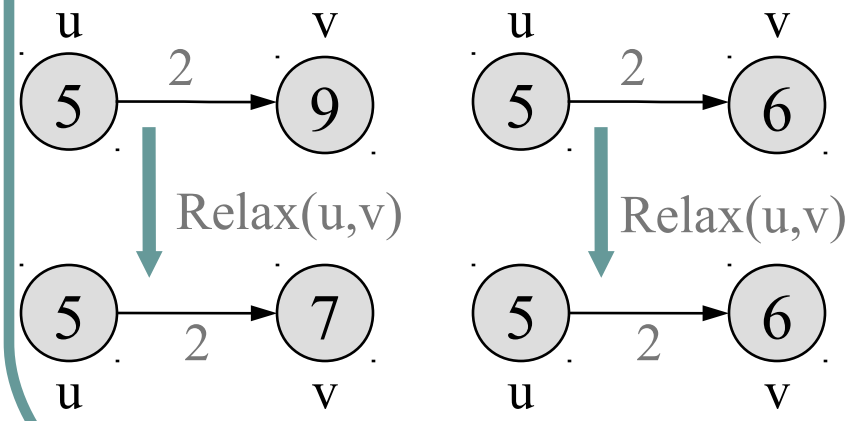
- Bài toán đường đi ngắn nhất
  - **Một nguồn (một đích).** Tìm đường đi ngắn nhất từ một nguồn (đỉnh *s*) tới mỗi đỉnh còn lại.
  - **Một cặp.** Cho một cặp đỉnh, tìm đường đi ngắn nhất giữa chúng. Lời giải của bài toán một nguồn có thể giải quyết bài toán này hiệu quả
  - **Tất cả các cặp.** Tìm đường đi ngắn nhất giữa mọi cặp đỉnh: Quy hoạch động

Trọng số âm và chu trình?

- Trọng số âm OK, miễn là không có *chu trình trọng số âm* (nếu không sẽ tồn tại đường đi với trọng số nhỏ tùy ý)
- Đường đi ngắn nhất không được chứa chu trình (nếu không ta có thể cải thiện bằng cách xóa chu trình)
  - Mỗi đường đi ngắn nhất trong đồ thị *G* không thể có nhiều hơn *n* – 1 cạnh, trong đó *n* là số đỉnh

Nới lỏng

- Cho mỗi đỉnh *v* trong đồ thị, ta duy trì một *v*.**d**(), ước lượng của đường đi ngắn nhất từ *s*, khởi tạo bằng ∞
- Nới lỏng một cạnh (*u*,*v*) nghĩa là kiểm tra xem ta có thể cải thiện đường đi ngắn nhất tới *v* bằng cách đi qua *u*




```
Relax (u,v,G)
if v.d() > u.d()+G.w(u,v)
   then
    v.setd(u.d()+G.w(u,v))
    v.setparent(u)
```

Giải thuật Dijkstra

- Trọng số cạnh không âm
- Tương tự BFS (nếu tất cả trọng số = 1, tương đương BFS)
- Sử dụng *Q*, một hàng đợi ưu tiên có khóa theo *v*.**d**() (BFS sử dụng hàng đợi FIFO, ở đây ta sử dụng hàng đợi ưu tiên, được sắp xếp lại mỗi khi có **d** giảm)
- Ý tưởng cơ bản:
  - duy trì một tập hợp *S* các đỉnh đã giải quyết
  - tại mỗi bước lựa chọn đỉnh *u* "gần nhất", thêm nó vào *S*, và nới lỏng tất cả các cạnh từ *u*

Demo

- demo-dijkstra.ppt

Dijkstra's Pseudo Code

- Input: Graph *G*, start vertex *s*

```
Dijkstra(G,s)
01 for each vertex u ∈ G.V()
02     u.setd(∞)
03     u.setparent(NIL)
04 s.setd(0)
05 // Set S is used to explain the algorithm
06 Q.init(G.V())  // Q is a priority queue ADT
07 while not Q.isEmpty()
08     u ← Q.extractMin()
09
10     for each v ∈ u.adjacent() do
11         Relax(u, v, G)
12         Q.modifyKey(v)
```

relaxing edges

Cài đặt

- Sửa đổi API đồ thị để hỗ trợ đồ thị có trọng số như sau

```
#define INFINITIVE_VALUE  10000000
typedef struct {
   JRB edges;
   JRB vertices;
} Graph;
void addEdge(Graph graph, int v1, int v2, double weight);
double getEdgeValue(Graph graph, int v1, int v2); // return
    INFINITIVE_VALUE if no edge between v1 and v2
int indegree(Graph graph, int v, int* output);
int outdegree(Graph graph, int v, int* output);
double shortestPath(Graph graph, int s, int t, int* path,
    int*length); // return the total weight of the path and the path is
    given via path and its length. Return INFINITIVE_VALUE if no
    path is found
```

Bài tập 1

- Cài đặt API đồ thị có trọng số. Kiểm tra API bằng ví dụ sau

```
Graph g = createGraph();
// add the vertices and the edges of the graph here
int s, t, length, path[1000];
double weight = shortestPath(g, s, t, path, &length);
if (weight == INFINITIVE_VALUE)
   printf("No path between %d and %d\n", s, t);
else {
   printf("Path between %d and %d:", s, t);
   for (i=0; i<length; i++) printf("%4d", path[i]);
   printf("Total weight: %f", weight);
}
```

Mã nguồn

- dijkstra.c

Bài tập 2

- Mô phỏng bản đồ xe bus Hà Nội.
- Đầu tiên, đầu tiên cần thu thập dữ liệu và bản đồ xe bus Hà Nội để xây dựng đồ thị trong đó
  - Mỗi đỉnh là một trạm xe bus ở Hà Nội
  - Các cạnh nối các trạm bằng cách tuyến xe bus
  - VD: Có 16 kết nối bởi xe bus số 1A: “Yên Phụ - Hàng Đậu - Hàng Cót - Hàng Gà - Hàng Điếu - Đường Thành - Phủ Doãn - Triệu Quốc Đạt - Hai Bà Trưng - Lê Duẩn - Khâm Thiên - Nguyễn Lương Bằng- Tây Sơn - Nguyễn Trãi - Trần Phú (Hà Đông) - Bến xe Hà Đông”
  - http://www.hanoibus.com.vn/InfobusVN/hanoibus/index.asp?pPage=lotrinh.htm

Bài tập 2 (tiếp)

- Mỗi cạnh trong đồ thị được đánh dấu với những tuyến bus kết nối hai trạm. VD: Cạnh “Yên Phụ - Trần Nhật Duật” có các tuyến 4A, 10A.
- Tổ chức và lưu trữ dữ liệu trong tệp để nạp vào chương trình khi thực thi
- Viết lại API đồ thị để lưu trữ đồ thị trong bộ nhớ
- Phát triển chức năng tìm “đường đi ngắn nhất” để đi từ một điểm đến điểm khác. VD: từ “Yên Phụ” tới “Ngô Quyền”.