Weighted graph

huydq@hust.edu.vn

Đồ thị có trọng số (Weighted Graph)

- Ta có thể thêm các thuộc tính vào các cạnh.
- Các thuộc tính này gọi là trọng số.
  - Ví dụ: Nếu ta đang dùng đồ thị như là 1 bản đồ với các đỉnh là các thành phố và các cạnh là đường cao tốc giữa các thành phố.
  - Trọng số của các cạnh là chi phí di chuyển 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 cạnh)
- Trọng số của đương 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 = đường đi với trọng số nhỏ nhất

Ứng dụng

- Định tuyến trên mạng
- Lập kế hoạch hoạt động của robot
- Tìm đường đi trên bản đồ, trong giao thông

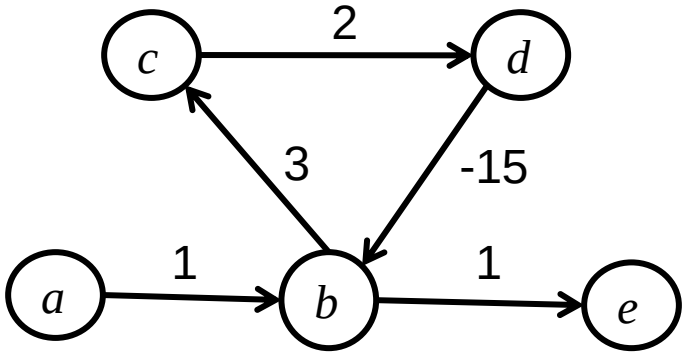
Bài toán tìm đường đi ngắn nhất

- Các bài toán tìm đường đi ngắn nhất
  - **1 nguồn (1 đích). Tìm đường đi ngắn nhất từ 1 nguồn cho trước đến đích**
  - **1 cặp. Cho 2 đỉnh, tìm đường đi ngắn nhất giữa chúng.**
  - **Tất cả các cặp. Tìm các đường đi ngắn nhất giữa các cặp đỉnh.**

Chu trình âm

- Đường đi ngắn nhất giữa hai đỉnh nào đó có thể không tồn tại.
  - Chẳng hạn, nếu không có đường đi từ *s* đến *t*, thì rõ ràng cũng không có đường đi ngắn nhất từ *s* đến *t*.
  - Ngoài ra, nếu đồ thị chứa cạnh có trọng số âm thì có thể xảy ra tình huống: độ dài đường đi giữa hai đỉnh nào đó có thể làm bé tuỳ ý.

Chu trình âm



- Xét đường đi từ a đến e:

  $a, k(b, c, d, b), e.$

  nghĩa là ta đi k lần vòng theo chu trình

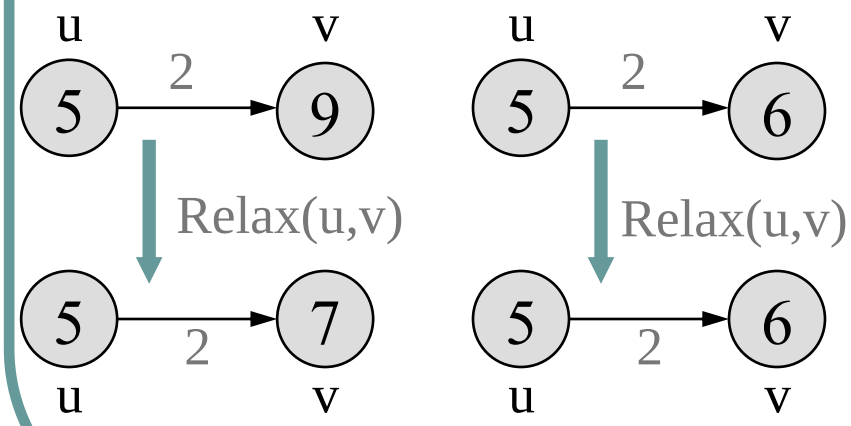
  $C: b, c, d, b$

  trước khi đến e. Độ dài của đường đi này là bằng:

  $c(a,b) + k\,\rho(C) + c(b,e) = 2 - 10k \longrightarrow -\infty$, khi $k \longrightarrow +\infty$.

Relaxation

- Với mỗi đỉnh v trên đồ thị, ta có 1 hàm giá/chi phí v.d() để quản lý chi phí đường đi ngắn nhất từ s, khởi tạo = ∞
- Relaxing cạnh(u,v) nghĩa là kiểm tra xem có thể cải thiện đường đi ngắn nhất từ s đến v bằng cách đi qua u hay không.




```
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)
```

Thuật toán Dijkstra

- Không có cạnh có trọng số âm
- Giống tìm kiếm theo chiều rộng (nếu tất cả các trọng số = 1, dùng BFS)
- Dùng hàng đợi ưu tiên (Priority Queue) PQ được quyết định bởi hàm v.d() (BFS sử dụng hàng đợi FIFO, ở đây ta dùng PQ, hàng đợi này được sắp xếp lại bất cứ khi nào 1 khoảng cách d nào đó giảm xuống)
- Ý tưởng cơ bản
  - Duy trì 1 tập *S* của các đỉnh đã thăm
  - Tại mỗi bước, chọn đỉnh gần nhất u, thêm vào S, giải phóng (relaxation) 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

- Biến đổi graph API cho phép đưa vào trọng số cạnh 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); // trả về
    INFINITIVE_VALUE nếu không có cạnh giữa v1 và 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 tổng giá trị đường đi, đường đi. Return
INFINITIVE_VALUE nếu không tìm thấy đường đi
```

Quiz

- Viết chương trình cài đặt đồ thị trọng số dùng API.
- Thử nghiệm với 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);
}
```

Mini Project II

- Mục đích của bài này là mô phỏng bản đồ xe buýt Hà Nội
- Trước tiên, bạn cần sưu tập dữ liệu về hệ thống các tuyến buýt Hà nội theo dạng đồ thị, trong đó:
- Mỗi đỉnh là 1 trạm xe buýt ứng với 1 điểm trong Hà nội
- Mỗi cạnh nối trạm xe buýt thông qua tuyến buýt.
- Ví dụ, có 16 trạm trên tuyến buýt 1A: “Yên Phụ - Hàng Đậu - Hàng Cót - Hàng Gà - Hàng Điếu - Đường Thành - Phủ Doãn- Triệu Quốc Đạt - Hai Bà Trưng - Lê Duẩn - KhâmThiên - Nguyễn Lương Bằng- Tây Sơn - Nguyễn Trãi - Trần Phú (Hà Đông) - Bến xe Hà Đông”
- Tham khảo tại: http://timbus.vn/fleets.aspx

Mini project II (cont.)

- Mỗi cạnh trên đồ thị được gắn với tuyến bus từ 1 điểm này đến điểm khác. Ví dụ, cạnh “Yên Phụ -Trần Nhật Duật” được gắn với 4A, 10A.

- Tổ chức và lưu trữ dữ liệu trong 1 file nạp vào chương trình khi chạy.
- Viết lại graph API để lưu bản đồ bus trong bộ nhớ
- Xây dựng hàm để tìm đường đi ngắn nhất giữa 2 điểm. Ví dụ từ “Yên Phụ” đến “Ngô Quyền”.