**BÀI TOÁN ĐƯỜNG ĐI NGẮN NHẤT**

**1. Giới thiệu**

**2. ĐỒ THỊ CÓ TRỌNG SỐ**

Đồ thị mà mỗi cạnh của nó được gán cho tương ứng với một số (nguyên hoặc thực) được gọi  
là đồ thị có trọng số. Số gán cho mỗi cạnh của đồ thị được gọi là trọng số của cạnh. Tương tự  
như đồ thị không trọng số, có nhiều cách biểu diễn đồ thị có trọng số trong máy tính. Đối với  
đơn đồ thị thì cách dễ dùng nhất là sử dụng ma trận trọng số:  
Giả sử đồ thị G = (V, E) có n đỉnh. Ta sẽ dựng ma trận vuông C kích thước n x n. Ở đây:  
 Nếu (u, v) ∈ E thì C[u, v] = trọng số của cạnh (u, v)  
 Nếu (u, v) ∉ E thì tuỳ theo trường hợp cụ thể, C[u, v] được gán một giá trị nào đó để có thể  
nhận biết được (u, v) không phải là cạnh (Chẳng hạn có thể gán bằng +∞, hay bằng 0, bằng  
-∞, v.v…)  
 Quy ước c[v, v] = 0 với mọi đỉnh v.  
Đường đi, chu trình trong đồ thị có trọng số cũng được định nghĩa giống như trong trường  
hợp không trọng số, chỉ có khác là độ dài đường đi không phải tính bằng số cạnh đi qua, mà  
được tính bằng tổng trọng số của các cạnh đi qua.

3. Bài toán đường đi ngắn nhất

3.1 Đặt vấn đề

3.2 Đường đi ngắn nhất không trọng số

3.3 Giải thuật Dijkstra, trường hợp trọng số trên các cung không âm - thuật toán dijkstra

3.4 Đồ thị chu trình âm

3.5 trường hợp đồ thị không có chu trình âm - thuật toán ford bellman

3.6 trường hợp đồ thị không có chu trình - sắp xếp tô pô

3.7 đường đi ngắn nhất giữa mọi cặp đỉnh - thuật toán floyd

4. Kết luận:

Bài toán đường đi dài nhất trên đồ thị trong một số trường hợp có thể giải quyết bằng cách đổi  
dấu trọng số tất cả các cung rồi tìm đường đi ngắn nhất, nhưng hãy cẩn thận, có thể xảy ra  
trường hợp có chu trình âm.  
Trong tất cả các cài đặt trên, vì sử dụng ma trận trọng số chứ không sử dụng danh sách cạnh  
hay danh sách kề có trọng số, nên ta đều đưa về đồ thị đầy đủ và đem trọng số +∞ gán cho  
những cạnh không có trong đồ thị ban đầu. Trên máy tính thì không có khái niệm trừu tượng  
+∞ nên ta sẽ phải chọn một số dương đủ lớn để thay. Như thế nào là đủ lớn? số đó phải đủ lớn  
hơn tất cả trọng số của các đường đi cơ bản để cho dù đường đi thật có tồi tệ đến đâu vẫn tốt  
hơn đường đi trực tiếp theo cạnh tưởng tượng ra đó.  
Xét về độ phức tạp tính toán, nếu cài đặt như trên, thuật toán Ford-Bellman có độ phức tạp là  
O(n3), thuật toán Dijkstra là O(n2), thuật toán tối ưu nhãn theo thứ tự tôpô là O(n2) còn thuật  
toán Floyd là O(n3). Tuy nhiên nếu sử dụng danh sách kề, thuật toán tối ưu nhãn theo thứ tự  
tôpô sẽ có độ phức tạp tính toán là O(m). Thuật toán Dijkstra kết hợp với cấu trúc dữ liệu  
Heap có độ phức tạp O(max(n, m).logn).  
Khác với một bài toán đại số hay hình học có nhiều cách giải thì chỉ cần nắm vững một cách  
cũng có thể coi là đạt yêu cầu, những thuật toán tìm đường đi ngắn nhất bộc lộ rất rõ ưu,  
nhược điểm trong từng trường hợp cụ thể (Ví dụ như số đỉnh của đồ thị quá lớn làm cho  
không thể biểu diễn bằng ma trận trọng số thì thuật toán Floyd sẽ gặp khó khăn, hay thuật  
toán Ford-Bellman làm việc khá chậm). Vì vậy yêu cầu trước tiên là phải hiểu bản chất và  
thành thạo trong việc cài đặt tất cả các thuật toán trên để có thể sử dụng chúng một cách uyển  
chuyển trong từng trường hợp cụ thể. Những bài tập sau đây cho ta thấy rõ điều đó.