Thuật toán Floyd-Warshall để duyệt đồ thị

algorithm 28 programming 60 graph 1

viethnguyen viết ngày 23/05/2015

Introduction

Khi nhắc đến các thuật toán duyệt đồ thị, có thể bạn đã biết (và đã từng implement) Depth-First Search, Breadth-First Search, hoặc Dijkstra. Xin nhắc lại về ý nghĩa của từng thuật toán, đứng ở khía cạnh bài toán tìm đường đi ngắn nhất. DFS dùng để giải các bài toán mà chúng ta muốn tìm được lời giải (không nhất thiết phải là quãng đường ngắn nhất), hoặc ta muốn thăm tất cả các đỉnh của đồ thị. BFS cũng để duyệt các đỉnh của đồ thị, nhưng có một tính chất quan trọng là: nếu tất cả các cạnh *không có trọng số*, lần đầu tiên một đỉnh được thăm, ta có ngay đường đi ngắn nhất đến đỉnh đó. Bây giờ đến thuật toán Disjkstra, đây là thuật toán nổi tiếng dùng để tìm đường đi ngắn nhất từ một đỉnh cho trước đến các đỉnh còn lại, trong một đồ thị có các cạnh *có trọng số không âm*. Như vậy, Dijkstra đã tiến hơn một bước so với BFS.

Đó là sơ qua về ba thuật toán mà có thể mọi người đều đã biết. Trong bài viết này, tôi xin giới thiệu một thuật toán ít biết đến hơn để duyệt đồ thị, đó là Floyd-Warshall.

Thuật toán Floyd-Warshall là gì?

Nếu như Dijkstra giải quyết bài toán tìm đường đi ngắn nhất từ *một đỉnh cho trước* đến mọi đỉnh khác trong đồ thị, thì Floyd-Warshall sẽ tìm đường đi ngắn nhất *giữa mọi đỉnh* sau một lần chạy thuật toán. Một tính chất nữa là Floyd-Warshall có thể chạy trên đồ thị có các cạnh có trọng số *có thể âm*, tức là không bị giới hạn như Dijkstra. Tuy nhiên, lưu ý là trong đồ thị không được có vòng (cycle) nào có tổng các cạnh là âm, nếu có vòng như vậy ta không thể tìm được đường đi ngắn nhất (mỗi lần đi qua vòng này độ dài quãng đường lại giảm, nên ta có thể đi vô hạn lần)

Thuật toán Floyd-Warshall so sánh tất cả các đường đi có thể giữa từng cặp đỉnh. Nó là một dạng của quy hoạch động (Dynamic Programming). Đặt hàm adj(i,j,k) là đường đi ngắn nhất từ i đến j, chỉ dùng các đỉnh trong tập {1,2,...,k}. Giả sử ta muốn tính adj{i,j,k+1}. Với mỗi cặp đỉnh i và j, đường đi ngắn nhất có thể là: (1) đường đi chỉ sử dụng các đỉnh trong tập {1,...k} hoặc (2) đường đi từ i đến k+1 rồi từ k+1 đến j, cũng chỉ sử dụng các đỉnh trong tập {1,...k}. Do vậy:

Trường hợp cơ bản: adj(i,j,0) = w(i,j)

Đệ quy: adj(i,j,k+1) = min{adj(i,j,k), adj(i,k+1, k) + adj(k+1, j, k)}

Đây là đoạn pseudocode của Floyd-Warshall (có một chút thay đổi, nhưng ý tưởng là như nhau)

```
            adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
```

Dễ thấy độ phức tạp thuật toán là O(n^3) với n là số đỉnh của đồ thị.

Thuật toán Floyd-Warshall dùng cho tìm tính chất kết nối

Tư tưởng của thuật toán Floyd-Warshall trong việc tìm đường đi ngắn nhất có thể áp dụng vào các bài toán dạng tìm tính chất kết nối giữa các đỉnh. Tôi xin lấy một ví dụ, đó là bài TopCoder SRM 184, Div 2, 1000-point problem

Đề bài như sau (xin chịu khó đọc hiểu đề bài)

> You are arranging a weird game for a team building exercise. In this game there are certain locations that people can stand at, and from each location there are paths that lead to other locations, but there are not necessarily paths that lead directly back. You have everything set up, but you need to know two important numbers. There might be some locations from which every other location can be reached. There might also be locations that can be reached from every other location. You need to know how many of each of these there are.
>
> Create a class TeamBuilder with a method specialLocations that takes a String[] paths that describes the way the locations have been connected, and returns a int[] with exactly two elements, the first one is the number of locations that can reach all other locations, and the second one is the number of locations that are reachable by all other locations. Each element of paths will be a String containing as many characters as there are elements in paths. The i-th element of paths (beginning with the 0-th element) will contain a '1' (all quotes are for clarity only) in position j if there is a path that leads directly from i to j, and a '0' if there is not a path that leads directly from i to j.
>
> **Examples**
>
> - {"010","000","110"}
> Returns: { 1, 1 }
> Locations 0 and 2 can both reach location 1, and location 2 can reach both of the other locations, so we return {1,1}.
>
> - {"0010","1000","1100","1000"}
> Returns: { 1, 3 }
> Only location 3 is able to reach all of the other locations, but it must take more than one path to reach locations 1 and 2. Locations 0, 1, and 2 are reachable by all other locations. The method returns {1,3}.
>
> - {"01000","00100","00010","00001","10000"}
> Returns: { 5, 5 }

Each location can reach one other, and the last one can reach the first, so all of them can reach all of the others.

- {"0110000","1000100","0000001","0010000","0110000","1000010","0001000"}
  Returns: { 1, 3 }

Solution
Về cơ bản, bài này cần tìm số lượng các đỉnh mà từ đó có thể đi đến tất cả các đỉnh khác, và số lượng các đỉnh mà các đỉnh khác đều có thể đi tới. Một ví dụ rất tốt để áp dụng thuật toán Floyd-Warshall tìm tính chất kết nối giữa 2 đỉnh bất kì.

Trong bài này, chúng ta chỉ cần phải kiểm tra xem có đường đi từ đỉnh i đến đỉnh j trong đồ thị hay không. Chúng ta sẽ áp dụng thuật toán Floyd-Warshall trên, nhưng có thay đổi một chút trong dòng xử lý bên trong vòng lặp. Về cơ bản, chúng ta vẫn sử dụng ý tưởng là update thông tin giữa 2 đỉnh i và j, mỗi khi ta có thêm thông tin giữa đỉnh i và đỉnh k, đỉnh k và đỉnh j, với k là một đỉnh khác i và j. Nhưng ta không cập nhật thông tin về * đường đi ngắn nhất* nữa, mà ta cập nhật thông tin về *có hay không đường đi từ i đến j*. Với mỗi cặp đỉnh i và j chưa có kết nối, ta sẽ kiểm tra xem nếu có đường đi từ i đến k và từ k đến j, thì ta cập nhật là có đường đi từ i đến j.

Sau đây là đoạn code C++ minh hoạ:

```
 vector<int> TeamBuilder::specialLocations(vector<string> paths){

   vector<vector<int> > adj;
                             ++){
                         [i];

            row.push_back(s[j] -'0');
        adj.push_back(row);
    }


                             == k || k==i)
                    continue;
                                    no-paths
                                     & adj[k][j] != 0){
                        adj[i][j] = 1;
                    }
                }
            }
        }
    }
                     locations that can reach all other locations

                             +){
            if(j==i) continue;

                        chAll = false;
                break;
            }
```

```
            firstNum ++;
        }
                        ocations that are reachable from other locations
                                        = true;
                        eachedFromAll = false;
                    break;
                }
                                omAll)
                secondNum ++;
        }
                    ck(secondNum);
        return res;
```

Tham khảo:

1. Floyd-Warshall Algorithm
2. TopCoder Graph Tutorial

Cơ bản về Arduino

arduino IoT embedded

Giới thiệu về Arduino Có thể bạn đã quen lập trình trên PC, với những ngôn ngữ như C, C++, C, Java, Python, Ruby... Nhưng bạn có biết là phần mềm...
viethnguyen viết hơn 2 năm trước
13 4

8 0

Haskell's laziness

Haskell functional programming

Trong bài viết này, tôi sẽ trình bày về một đặc tính của Haskell khá khác biệt so với các ngôn ngữ lập trình khác, đó là laziness (dịch tiếng việt ...
viethnguyen viết gần 3 năm trước
8 0

6 1

Thử parse logical expression với Parsec (Haskell)

Haskell Parsec

Đề bài Gần đây, thằng bạn tôi rảnh rỗi học cách parsing với Scala. Tôi nghe nói Scala có parser combinator khá là mạnh. Vừa hay, tôi cũng đang tìm...
viethnguyen viết hơn 2 năm trước
6 1