Contents

[- **DFS and experiments:** 1](#_Toc75124941)

[- **Dijkstra and experiments.** 3](#_Toc75124942)

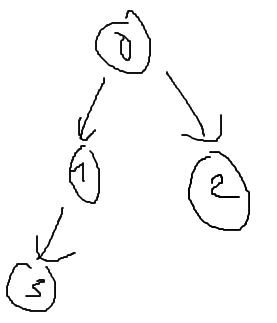
[- **Backtrack liệt kê hoán vị** 3](#_Toc75124943)

[Disjoin-set data structure. 4](#_Toc75124944)

* **DFS and experiments:**

Ở bậc đại học mình đã được đào tạo khá bài bản về các thuật toán, và DFS là một trong những thuật toán cơ bản về đồ thị mà mình được học. Tuy nhiên khi ấy số lượng bài tập về đồ thị mà mình làm không thực sự nhiều, chỉ là một vài bài cơ bản. Do muốn hiểu sâu hơn về đệ quy và quy hoạch động nên mình đã chọn DFS để làm một số thí nghiệm liên quan. Hi vọng sau lần luyện này sẽ làm được nhiều bài tập hơn.

DFS cơ bản và source code cơ bản. DFS là thuật toán tìm kiếm theo chiều sâu, ta đi từ một đỉnh và loang theo chiều sâu đến tất cả các đỉnh liên quan đến nó.



Ta sẽ đi lần lượt qua các đỉnh (0), (1), (3), (2)

DFS thường được code theo đệ quy, với bản chất là lưu các đỉnh sẽ được xử lý theo một stack. Ta xét mã cho đồ thị bên kia như sau : đầu tiên ta đi tới một đỉnh, sau đó ta đánh dấu nó lại, đi tiếp đến tất các các nút con của nó, và cứ thế tiếp tục

void dfs( int u ){

cout<<”push u :” << u << endl;

if (vis[u]) { return ;}

vis[u]= true;

for (auto v:g[u] ) { dfs(v); (1) cout<<"parent u: "<<u<< " child v: "<< v<<endl }

// vis[u] = false; (2)

cout<<”pop u :” << u << endl;

}

Từ rất lâu trước khi học đại học mình đã được nghe về đệ quy, nhưng chưa bao giờ có cảm giác thực sự hiểu về nó cả, mọi thứ khá mù mờ, lúc nào cũng là câu nói “Đệ quy là một hàm gọi lại chính nó và có thể biểu diễn dưới dạng công thức truy hồi trong toán học”. Lý thuyết thì là như vậy, thế nhưng mỗi lần thử phân tích một thuật đệ quy nào đó mình lại như gà mắc tóc, không hiểu vấn đề phát sinh từ đâu và tại sao lại như thế. Và thường mình sẽ học vẹt code. Điều này có thể có ích trong ngắn hạn, thế nhưng về lâu về dài nó biến mình thành một thằng thiểu năng thực sự, thấy cái gì mới cũng không làm được và không thể mở rộng để sáng tạo vấn đề mới của riêng mình được. Lần này để hiểu hơn về DFS, về đệ quy, chúng ta hãy làm một số thí nghiệm từ đoạn code dfs trên.

* Đầu tiên ta có kết quả thực hiện đoạn code dfs như sau:

|  |  |
| --- | --- |
| Những gì nhìn thấy trên màn hình  push u:0  u: 0 v: 1  push u:1  u: 1 v: 3  push u:3  pop u:3  parent u: 1 child v: 3  pop u:1  parent u: 0 child v: 1  u: 0 v: 2  push u:2  pop u:2  parent u: 0 child v: 2  pop u:0  đây là kết quả in ra màn hình của đoạn code trên, nhìn thực sự khó chịu, rút cục thì chuyện gì đã thực sự diễn ra ? | Thứ tự thực hiện trong máy tính   * push u:0 * u: 0 v: 1 * push u:1 * u: 1 v: 3 * push u:3 * pop u:3 * parent u: 1 child v: 3 * pop u:1 * parent u: 0 child v: 1 * u: 0 v: 2 * push u:2 * pop u:2 * parent u: 0 child v: 2 * pop u:0   trên thực tế đây là thứ tự được thực hiện, trông hơi giống python thế nhưng đây là những gì đã thực sự diễn ra. |

Các thuật toán đệ quy luôn khiến mình khó chịu nhất là lúc nó phối với một cái gì đấy lặp bên trong. DFS là một ví dụ cụ thể như vậy, mỗi đỉnh con được thăm theo một thứ tự đệ quy trong vòng lặp danh sách kề gắn với đỉnh đó. Để có thể hiểu được chi tiết hơn, ta sẽ xét 2 vị trí (1) và (2), xem những gì được thực hiện tại từng vị trí này.

1. Xuất hiện sau khi gọi đệ quy, (2) là điểm cuối của một hàm đệ quy. Như vậy (2) sẽ được thực hiện trước (1).
2. Được thực hiện lần đầu tiên tại trường hợp cơ bản của hàm đệ quy, và sau đó đẩy ngược lên (1). Vị trí (1) xuất hiện trước nhưng lại được thực hiện sau. Vị trí (1) là vị trí bắt đầu backtrack, nói cách khác là đi ngược lên.

* **Dijkstra and experiments.**

Bài toán: cho đồ thị gồm n đỉnh, m cạnh, để đi từ đỉnh đến cạnh cần chi phí x. Đường đi giữa các đỉnh không có chu trình, không có trọng số âm. Tìm đường đi ngắn nhất từ đỉnh nguồn đến tất cả các đỉnh khác.

Các bước của giải thuật:

Tạo một tập (set) giữ theo trình tự các đỉnh trong đường đi ngắn nhất. Các khoảng cách được tính toán và hoàn thiện. Tại thời điểm ban đầu. set rỗng.

Gán một giá trị khoảng cách cho tất cả các đỉnh từ đầu, vì cần tìm min -> inf

Thứ tự dựng set.

1. Pick một đỉnh u, mà nó không ở trong set và có khoảng cách ngắn nhất.
2. Thêm u vào set.
3. Cập nhật khoảng cách của tất cả các cạnh kề v

* **Backtrack liệt kê hoán vị**

Bài toán backtrack liệt kê hoán vị là một bài toán quen thuộc mình đã được làm quen từ khi vào đại học, để giải bài toán này thì cách tốt nhất và đơn giản nhất là viết đệ quy. Tuy nhiên việc không hiểu rõ đệ quy cộng với thói quen lười suy nghĩ và thử nghiệm luôn làm cho mình không thể làm được bài này. Lần này sau khi đã tìm tòi giải thuật đệ quy trong một thời gian và tự làm được một số bài về dfs thì mình sẽ quay lại bài toán này.

Ví dụ về hoán vị, ta có 3 phần tử: 1,2,3 các hoán vị từ tập đó như sau

1 2 3 | 1 3 2 | 213 | 231 | 312 | 321

Ta có 3 vị trí [ ] [ ] [ ] và ta muốn chọn lần lượt từng giá trị 1,2,3 cho 3 vị trí này.

Sau khi chọn xong giá trị cho phần tử n thì ta in.

Sau khi in xong, thì ta lui lại để chọn giá trị cho phần tử i

Có một điều rất khó hiểu của bài liệt kê hoán vị này là nó không hề có điều kiện trả về. không có điều kiện dừng cho trường hợp cơ bản. Trong khi cấu trúc cơ bản nhất của đệ quy là

recur(int n){

if ( n==1 ) \\ đây là trường hợp cơ bản, ta phải dừng chương trình và trả về

recur (n-1)

}

Tuy nhiên trong nhi

# **Disjoin-set data structure.**

Disjoint nghĩa là không giao nhau, set là tập hợp -> đây là cấu trúc dữ liệu về các tập con không giao nhau -> nếu giao nhau thì câu chuyện sẽ khác, rất khác.

Xét bài toán: cho tập x có n phần tử. Ta gom x1,x2,xk với k<n thành nhiều tập con khác nhau, tìm xem phần tử xi nằm trong tập con nào ?