I. Yêu cầu Project 1

1. Nhiệm vụ: Lập Trình thao tác liên quan cấu trúc dữ liệu đồ thị G(V,E) bằng danh sách liên kết (Adjacency List).

2. Chi tiết 2.1: InPut: Data: -roadNet-CA -roadNet-PA -roadNet-TX

Stanford Large Network Dataset Collection 2.2: Thuật toán

```
-Adjcency List
-Thuật toán tìm kiếm DFS
-Thuật toán tìm kiếm BFS
```

2.3: Output: Kết quả duyệt DFS từ 1 đỉnh Kết quả duyệt BFS từ 1 đỉnh Đếm số thành phần liên thông

II. Thuật Toán

3. Biểu diễn đồ thị Một đồ thị (đồ thị) là một dạng biểu diễn hình ảnh của một tập các đối tượng, trong đó các cặp đối tượng được kết nối bởi các link. Các đối tượng được nối liền nhau được biểu diễn bởi các điểm được gọi là các đỉnh (vertices), và các link mà kết nối các đỉnh với nhau được gọi là các cạnh (edges).

- Ta cần tìm hiểu một vài khái niệm sau:
- Đỉnh (Vertex): Mỗi nút của hình được biểu diễn như là một đỉnh.
- Cạnh (Edge): Cạnh biểu diễn một đường nối hai đỉnh.

Có nhiều cách biểu diễn đồ thị:

- Ma trận kề (Adjacency Matrix).
- Ma Trận trọng số
- Danh sách kề

Việc lựa chọn cách biểu diễn phụ thuộc vào từng bài toán cụ thể cần xét, từng thuật toán cụ thể cần cài đặt. Có 2 vấn đề chính cần quan tâm nhất là bộ nhớ cần đòi hỏi và thời gian cần thiết để trả lời truy vấn thường xuyên đối với đồ thị trong quá trình xử lý đồ thị.

Ta chọn biểu diễn đồ thị theo danh sách kề.

Code Build Graph

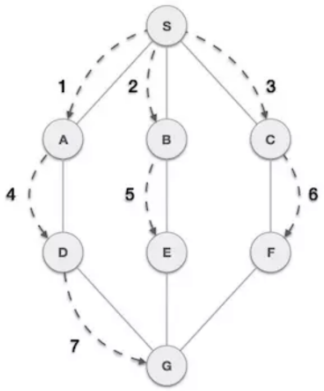
```

```

4. Duyệt đồ thị Phép duyệt đồ thị là một cách liệt kê tất cả các đỉnh của đồ thị này thành một danh sách tuyến tính. Hay nói cách khác, phép duyệt đồ thị cho ta một cách "đi qua" tất cả các đỉnh của đồ thị để truy nhập, thêm bớt thông tin ở các đỉnh của đồ thị đó.

Phép duyệt đồ thị không phụ thuộc vào hướng của các cạnh. Do vậy, với đồ thị có hướng thì ta vô hướng hóa trước khi duyệt.

- Ứng dụng: Xây dựng các thuật toán khảo sát các tính chất của đồ thị là thành phần cơ bản của nhiều thuật toán.
- Cần xây dựng thuật toán hiệu quả để thực hiện việc duyệt đồ thị. Ta xét hai thuật toán duyệt cơ bản:
    - Tìm kiếm theo chiều rộng (Breadth First Search – BFS)
    - Tìm kiếm theo chiều sâu (Depth First Search – DFS)
- Ý tưởng chung: Trong quá trình thực hiện thuật toán, mỗi đỉnh ở một trong ba trạng thái
    - Chưa thăm.
    - Đã thăm nhưng chưa duyệt xong (thể hiện bởi màu xám).
    - Đã duyệt xong (thể hiện bởi màu đen).
- Quá trình duyệt được bắt đầu từ một đỉnh v nào đó. Ta sẽ khảo sát các đỉnh đạt tới được từ v:
    - Thoạt đầu mỗi đỉnh đều có màu trắng (chưa thăm - not visited).
    - Đỉnh đã được thăm sẽ chuyển thành màu xám (trở thành đã thăm nhưng chưa duyệt xong).
    - Khi tất cả các đỉnh kề của một đỉnh v là đã được thăm, đỉnh v sẽ có màu đen (đã duyệt xong).

5. Tìm kiếm theo chiều rộng (BFS) 5.1: Tư Tưởng thuật toán. Giải thuật tìm kiếm theo chiều rộng duyệt qua một đồ thị theo chiều rộng và sử dụng hàng đợi (queue) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào.



5.2 Cài đặt hàng đợi. Hàng đợi (Queue): Hàng đợi (queue) là một cấu trúc dữ liệu hoạt động theo cơ chế FIFO (First In First Out), tạm dịch là “vào trước ra trước”. Có nghĩa là phần tử nào được thêm hàng đợi trước thì sẽ được lấy ra trước. Giả sử ta có một danh sách chứa những đỉnh đang "chờ" thăm. Tại mỗi bước, ta thăm một

đỉnh đầu danh sách và cho vào cuối danh sách những đỉnh chưa có trong danh sách mà kề với nó.

Giải thuật sử dụng hàng đợi:

- Quy tắc:
    - Duyệt tiếp tới đỉnh liền kề mà chưa được duyệt. Đánh dấu đỉnh mà đã được duyệt. Hiển thị đỉnh đó và đẩy vào trong một hàng đợi (queue).
    - Nếu không tìm thấy đỉnh liền kề, thì xóa đỉnh đầu tiên trong hàng đợi. Lặp lại Qui tắc 1 và 2 cho tới khi hàng đợi là trống.

5.3: Mã giả.

- Input: Đồ thị G = (V, E), vô hướng hoặc có hướng, và đỉnh xuất phát s
- Output:
    - Với mọi v $\in$ V
    - d[v]= khoảng cách từ s đến v
    - Xây dựng cây BFS gốc tại s chứa tất cả các đỉnh đạt đến được từ đỉnh s.
- Ta sẽ sử dụng màu để ghi nhận trạng thái của đỉnh trong quá trình duyệt
    - Trắng(White)- chưa thăm
    - Xám(Gray)- đã thăm nhưng chưa duyệt xong
    - Đen(Black)- đã duyệt xong

Tìm kiếm theo chiều rộng từ đỉnh s

BFS_Visit(s)

1. for u $\in $ V – {s}
2. do color[u] $\leftarrow $ white

3. d[u] $\leftarrow $ 0
4. π [u] $\leftarrow$ NULL
5. color[s] $\leftarrow$ gray
6. d[s] $\leftarrow$ 0
7. π[s] $\leftarrow$ NULL
8. Q $\leftarrow$ ϕ
9. enqueue(Q,s)
10. while Q $\ne $ ϕ
11. do u $\leftarrow$ dequeue(Q)
12. for v $\in $ Adj[u]
13. do if color[v] = white
14. then color[v] $\leftarrow$ gray
15. d[v] $\leftarrow$ d[u] + 1
16. $\pi $ [v] $\leftarrow$ u
17. enqueue(Q,v)
18. color[u] $\leftarrow$ black

5.5: Code.

```
De
 1:  void BFS(struct Graph* graph, int startVertex) {
 2:  struct queue* q = createQueue();
 3:
 4: graph->visited[startVertex] = 1;
 5: enqueue(q, startVertex);
 6:
 7: while (!isEmpty(q)) {
 8:  printQueue(q);
 9:  int currentVertex = dequeue(q);
 10:  printf("Visited %d\n", currentVertex);
 11:
 12:   struct node* temp = graph->adjLists[currentVertex];
 13:
 14:   while (temp) {
 15:     int adjVertex = temp->vertex;
 16:
 17:     if (graph->visited[adjVertex] == 0) {
 18:       graph->visited[adjVertex] = 1;
 19:       enqueue(q, adjVertex);
 20:     }
 21:     temp = temp->next;
 22:  }
 23: }
 24:}
```

5.6. Độ phức tạp của thuật toán BFS. • Thuật toán loại bỏ mỗi đỉnh khỏi hàng đợi đúng 1 lần, do đó thao tác DeQueue thực hiện đúng |V| lần. • Với mỗi đỉnh, thuật toán duyệt qua tất cả các đỉnh kề của nó và thời gian xử lý mỗi đỉnh kề như vậy là hằng số. Như vậy thời gian thực hiện câu lệnh if trong vòng lặp while là bằng hằng số nhân với số cạnh kề với đỉnh đang xét. • Do đó tổng thời gian thực hiện việc duyệt qua tất cả các

đỉnh là bằng một hằng số nhân với số cạnh |E|. Thời gian tổng cộng: O(|V|) + O(|E|) = O(|V|+|E|), hay O($|V|^2$).

6. Tìm kiếm theo chiều sâu DFS

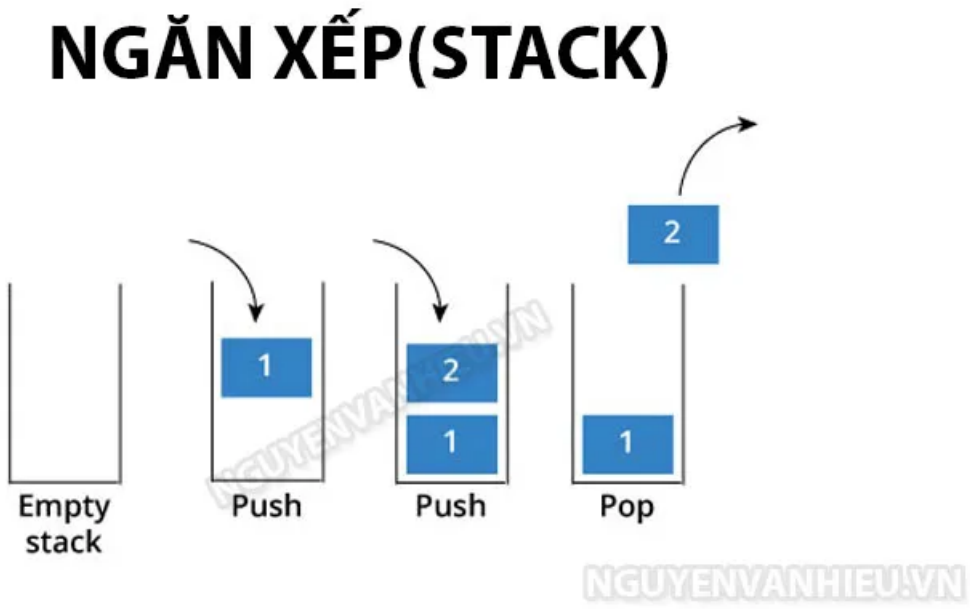
6.1: Tư tưởng thuật toán.

Giải thuật tìm kiếm theo chiều sâu còn được gọi là giải thuật tìm kiếm ưu tiên chiều sâu, là giải thuật duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị và sử dụng stack (ngăn xếp) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào.

Giải thuật tiếp tục cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước.

6.2: Cài đặt bằng ngăn xếp (Stack) Ngăn xếp (Stack): Một ngăn xếp là một cấu trúc dữ liệu dạng thùng chứa (container) của các phần tử (thường được gọi là các Node). Có hai thao tác cơ bản là push và pop.

- Push bổ sung một phần tử vào đỉnh (top) của ngăn xếp, nghĩa là nó sẽ được thêm vào sau các phần tử đã có sẵn trong ngăn xếp.
- Pop giải phóng và trả về phần tử đang đứng ở đỉnh của ngăn xếp. Phần tử sau khi được lấy sẽ bị xóa khỏi ngăn xếp.



Giải thuật sử dụng Stack

- Quy tắc Duyệt tiếp tới đỉnh liền kề mà chưa được duyệt. Đánh dấu đỉnh mà đã được duyệt. Hiển thị đỉnh đó và đẩy vào trong một ngăn xếp (stack). Nếu không tìm thấy đỉnh liền kề, thì lấy một đỉnh từ trong ngăn xếp (thao tác pop up). (Giải thuật sẽ lấy tất cả các đỉnh từ trong ngăn xếp mà không có các đỉnh liền kề nào). Lặp lại các qui tắc 1 và qui tắc 2 cho tới khi ngăn xếp là trống.

6.3: Mã giả.

- Input: G = (V, E) - đồ thị vô hướng hoặc có hướng.

- Output: Với mỗi v $\in$ V. d[v] = thời điểm bắt đầu thăm (v chuyển từ màu trắng sang xám) f [v] = thời điểm kết thúc thăm (v chuyển từ màu xám sang đen) π[v] : đỉnh từ đó ta đến thăm đỉnh v. Thuật toán tìm kiếm theo chiều sâu bắt đầu từ đỉnh u

Thuật toán tìm kiếm theo chiều sâu bắt đầu từ đỉnh u

> DFS-Visit(u) 1:color[u] $\leftarrow $ GRAY 2:time $\leftarrow $ time + 1 3:d[u] $\leftarrow $ time 4:for v $\in $ Adj[u] 5:do if color[v] = WHITE 6:then $\pi $[v] $\leftarrow $ u 7:DFS-Visit(v) 8:color[u] $\leftarrow $ BLACK 9:f[u] $\leftarrow $ time $\leftarrow $ time + 1

Thuật Toán tìm kiếm theo chiều sâu trên đồ thi G

DFS-Visit(u) 1: for u $\in$ V[G] 2: do color[u] $\leftarrow$ white 3: π[u] $\leftarrow$ NULL 4: time $\leftarrow$ 0 5: for u $\in$ V[G] 6: do if color[u]=white 7: then DFS-Visit(u)

6.4: Code.

```
DepthFirstSearch

1: void DFS(struct Graph* graph, int vertex) {
2:struct node* adjList = graph->adjLists[vertex];
3:struct node* temp = adjList;
4:
5:graph->visited[vertex] = 1;
6:printf("Visited %d \n", vertex);
7:
8:while (temp != NULL) {
9:  int connectedVertex = temp->vertex;
10:
11:  if (graph->visited[connectedVertex] == 0) {
12:    DFS(graph, connectedVertex);
13:  }
14:  temp = temp->next;
15: }
16:}
```

6.5: Độ phức tạp của thuật toán DFS.

- Thuật toán thăm mỗi đỉnh v$\in$V đúng một lần, việc thăm đỉnh đòi hỏi thời gian Θ(|V|)
- Với mỗi đỉnh v duyệt qua tất cả các đỉnh kề, với mỗi đỉnh kề thực hiện thao tác với thời gian hằng số. Do đó việc duyệt qua tất cả các đỉnh mất thời gian: Θ(|E|)
- Tổng cộng: Θ(|V|) + Θ(|E|) = Θ(|V|+|E|), hay Θ($|V|^2$)
- Như vậy, DFS có cùng độ phức tạp như BFS.

7. Số thành phần liên thông.

Bài toán liên thông. Cho đồ thị vô hướng G=(V,E). Hãy kiểm tra xem đồ thị G có phải liên thông hay không. Nếu G không là liên thông, cần đưa ra số lượng thành phần liên thông và danh sách các đỉnh của từng thành

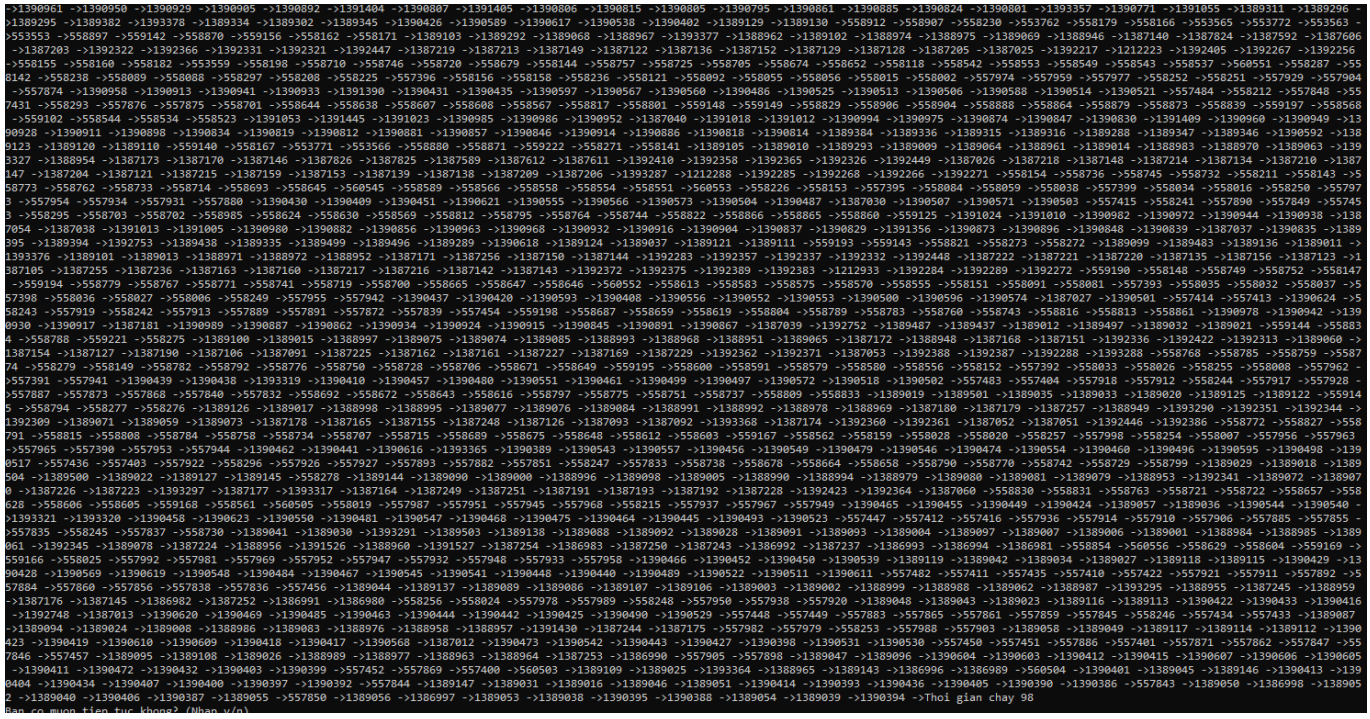
phần liên thông.

Để giải bài toán này, ta chỉ việc thực hiện DFS(G) (hoặc BFS(G)). Khi đó số lần gọi thực hiện BFS_Visit() (DFS_Visit()) sẽ chính là số lượng thành phần liên thông của đồ thị. Việc đưa ra danh sách các đỉnh của từng thành phần liên thông sẽ đòi hỏi phải đưa thêm vào biến ghi nhận xem mỗi đỉnh được thăm ở lần gọi nào trong BFS(G) (DFS(G)).

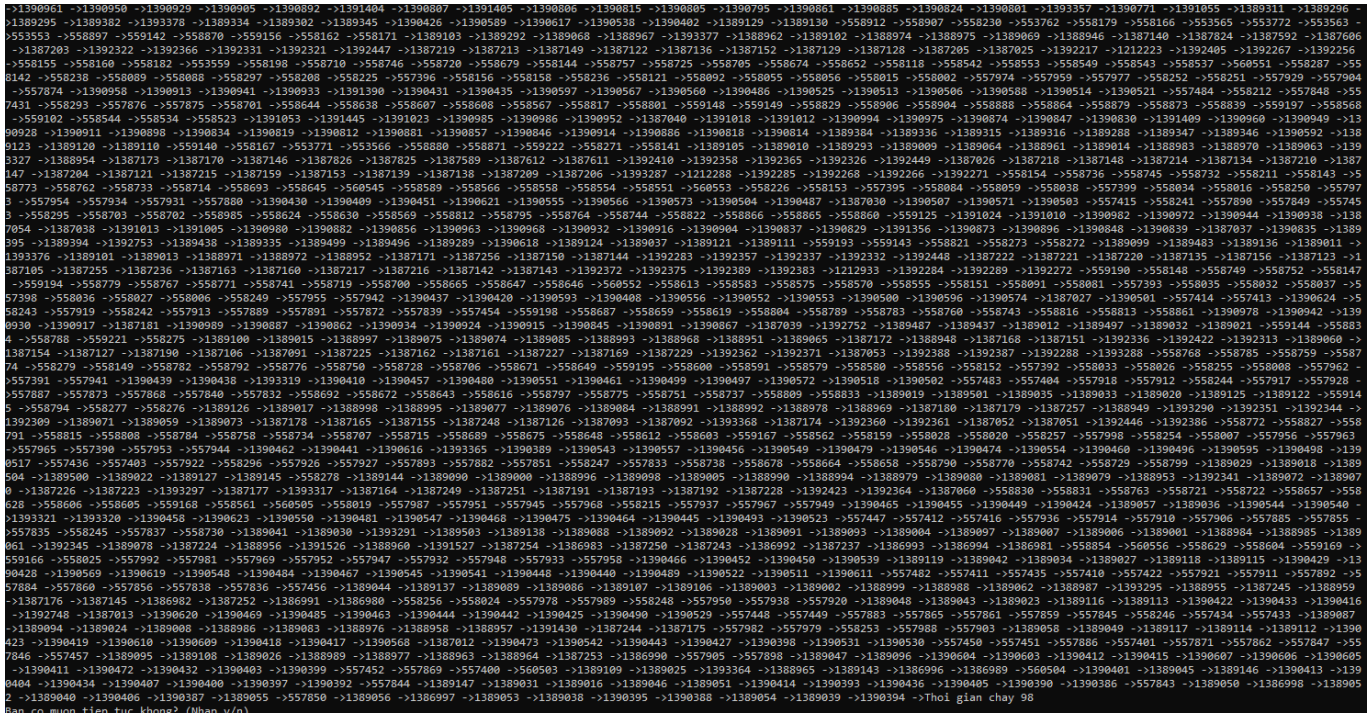
###III. Kết Quả

8.Test. DATA roadNet-TX.txt

Duyệt Thuật toán DFS Đỉnh bắt đầu 0



Duyệt Thuật toán BFS Đỉnh bắt đầu 0



Thời gian chạy 98s

Đếm số thành phần liên thông BFS !(BFS)[] DFS