lê yến

ĐH Khoa học Tự nhiên TP HCM

****

**Data Structures and Algorithms**

**THỰC HÀNH CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Đồ án cuối kì

Graph

Tài liệu này ghi nhận lại nội dung các thuật toán tìm kiếm trong đồ thị bao gồm:

1. DFS
2. BFS
3. UCS
4. Dijsktra

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

**Nội dung**

[I. Thông tin chung 1](#_Toc117510480)

[II. Nội dung 2](#_Toc117510481)

[III. Bảng phân công công việc 11](#_Toc117510482)

# Thông tin chung

Các thành viên tham dự:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **MSSV** | **Họ và tên** | **Email** |
| 1 | 21280115 | Trần Đức Trung | 21280115@student.hcmus.edu.vn |
| 2 |  |  |  |

Mục tiêu:

1. Hiểu được ý tưởng các thuật toán search và sort
2. Cài đặt thành công và vẽ trực quan so sánh các thuật toán

Thời gian bắt đầu: ………….3-10-2022 Thời gian kết thúc: ……………24-10-2022

# Nội dung

# 1. Giới thiệu thuật toán:

A. DFS

a. Ý tưởng chung:

- Tìm kiếm ưu tiên chiều sâu hay tìm kiếm theo chiều sâu ([tiếng Anh](https://vi.wikipedia.org/wiki/Ti%E1%BA%BFng_Anh): *Depth-first search* - DFS) là một [thuật toán](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n) duyệt hoặc tìm kiếm trên một [cây](https://vi.wikipedia.org/wiki/C%C3%A2y_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)) hoặc một [đồ thị](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B). Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

- Thông thường, DFS là một dạng tìm kiếm thông tin không đầy đủ mà quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật [quay lui](https://vi.wikipedia.org/wiki/Quay_lui_(khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh)) về đỉnh vừa mới tìm kiếm ở bước trước. Trong dạng không đệ quy, tất cả các đỉnh chờ được phát triển được bổ sung vào một [ngăn xếp](https://vi.wikipedia.org/wiki/Ng%C4%83n_x%E1%BA%BFp) [LIFO](https://vi.wikipedia.org/w/index.php?title=LIFO&action=edit&redlink=1)

- Ý tưởng DFS:

* + Chọn một đỉnh tùy ý của đồ thị làm gốc
  + Xây dựng đường đi từ đỉnh này bằng cách lần lượt ghép các cạnh sao cho mỗi cạch mới ghép sẽ nối đỉnh cuối cùng trên đường đi với một đỉnh còn chưa thuộc đường đi. Tiếp tục ghép thêm cạnh vào đường đi chừng nào không thể thêm được nữa
  + Nếu đường đi qua tất cả các đỉnh của đồ thị thì cây do đường này tạo nên là cây khung
  + Nêu chưa thì lùi lại đỉnh trước đỉnh cuối cùng của đường đi và xây dựng đường đi mới xuất phát từ đỉnh này đi qua các đỉnh còn chưa thuộc đường đi.

Nếu điều đó không thể làm được thì lùi thêm một đỉnh nữa trên đường đi và thử xây dựng đường đi mới. Tiếp tục quá trình như vậy cho đến khi tất cả các đỉnh của đồ thị được ghép vào cây. Cây T có được là cây khung của đồ thị.

b. Mã giả

*//Traverses a graph beginning at vertex v*

*//by using a depth-first search*

dfs (v: Vertex)

         s = a new empty stack

*// Push v onto the stack and mark it*

            s.push(v)

            Mark v as visited

*// Loop invariant: there is a path from vertex v at the*

*// bottom of the stack s to the vertex at the top of s*

            While(!s.isEmpty()){

                if(no unvisited vertices are adjacent to the vertex on the top of the stack)

                 s.pop() *//Backtrack*

                else{

                  Select an unvisited vertex u adjacent to the vertex on the top of the stack

                  s.push(u)

                  Mark u as visited

                }

        }

}

c. Đánh giá thuật toán:

- **Optimal**: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

- **Completeness**: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

- Time complexity: : O(V+E) khi đồ thị được biểu diễn ở dạng danh sách kề

O(V2) khi đồ thị được biểu diễn ở dạng ma trận kề

Với V là số đỉnh, E là số cạnh trong đồ thị

- Thuận lợi:

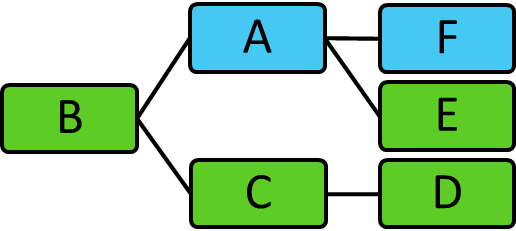
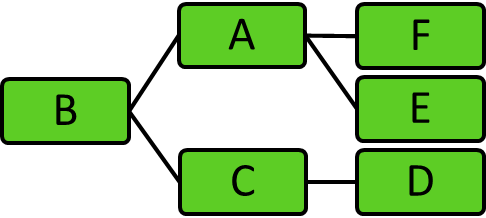
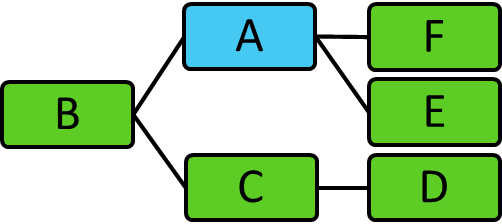
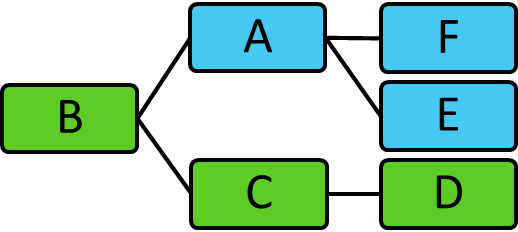
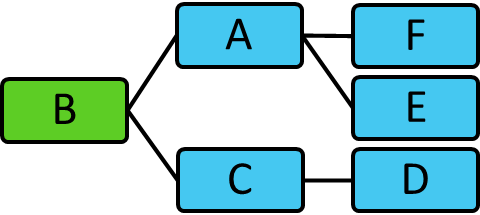
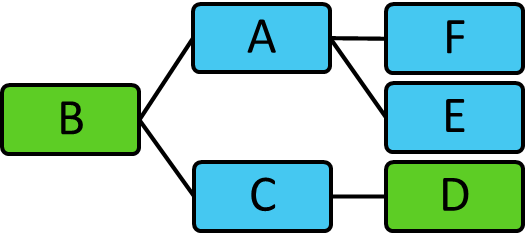
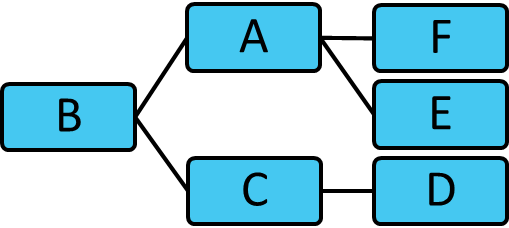
* DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
* It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

- Khó khăn:

* There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
* DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

d. Ví dụ:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Current | Opened set (Stack) | Closed set |
| B0 |  | A |  |
| B1 | A | [B,E,F] | A |
| B2 | F | [B,E] | F |
| B3 | E | [B,C] | E |
| B4 | C | [B,B,D] | C |
| B5 | D | [B,B] | D |
| B6 | B | [B] | B |
| B7 |  | [] |  |



B. BFS

a. Ý tưởng chung:

- Trong [lý thuyết đồ thị](https://vi.wikipedia.org/wiki/L%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B), tìm kiếm theo [chiều rộng](https://vi.wikipedia.org/wiki/Chi%E1%BB%81u_r%E1%BB%99ng) (BFS) là một [thuật toán tìm kiếm trong đồ thị](https://vi.wikipedia.org/w/index.php?title=Thu%E1%BA%ADt_to%C3%A1n_t%C3%ACm_ki%E1%BA%BFm_trong_%C4%91%E1%BB%93_th%E1%BB%8B&action=edit&redlink=1) trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo [chiều rộng](https://vi.wikipedia.org/wiki/Chi%E1%BB%81u_r%E1%BB%99ng) cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo [chiều rộng](https://vi.wikipedia.org/wiki/Chi%E1%BB%81u_r%E1%BB%99ng) luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại. Xem thêm thuật toán [tìm kiếm theo chiều sâu](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_s%C3%A2u), trong đó cũng sử dụng 2 thao tác trên nhưng có trình tự quan sát các đỉnh khác với thuật toán tìm kiếm theo [chiều rộng](https://vi.wikipedia.org/wiki/Chi%E1%BB%81u_r%E1%BB%99ng).

- Cấu trúc dữ liệu được sử dụng là hàng đợi (queue).

- Ý tưởng BFS:

* Thêm v1 như là gốc của cây rỗng
* Thêm vào các đỉnh kề v1 và các cạnh nối v1 với chúng. Những đỉnh này là đỉnh mức 1 trong cây
* Đối với mọi đỉnh v mức 1, thêm vào các cạnh kề với v vào cây sao cho không tạo nên chu trình. Ta thu được các đỉnh mức 2
* Tiếp tục quá trình này cho tới khi tất cả các đỉnh của đồ thị được ghép vào cây. Cây T có được là cây khung của đồ thị.

b. Mã giả

*// Traverses a graph beginning at vertex v by using a*

*// breadth-first search: Iterative version.*

bfs(v: Vertex)

q = a new empty queue

*// Add v to queue and mark it*

q.enqueue(v)

Mark v as visited

while (!q.isEmpty()){

q.dequeue(w)

*// Loop invariant: there is a path from vertex w to every vertex in the queue q*

for (each unvisited vertex u adjacent to w){

Mark u as visited

q.enqueue(u)

}

}

c. Đánh giá thuật toán:

- Tính tối ưu: Đối với đồ thị không trọng số, BFS sẽ tối ưu hơn so với DFS. BFS is optimal if the path cost is a non-decreasing function of d(depth). Normally, BFS is applied when all the actions have the same cost.

- Tính đầy đủ: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

- Độ phức tạp: O(V+E) khi đồ thị được biểu diễn ở dạng danh sách kề

O(V2) khi đồ thị được biểu diễn ở dạng ma trận kề

Với V là số đỉnh, E là số cạnh trong đồ thị

- Thuận lợi:

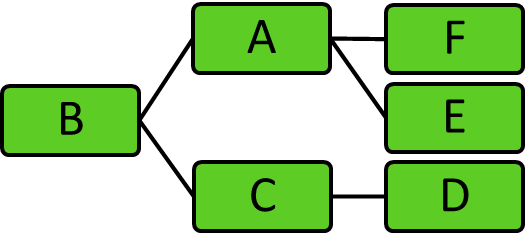
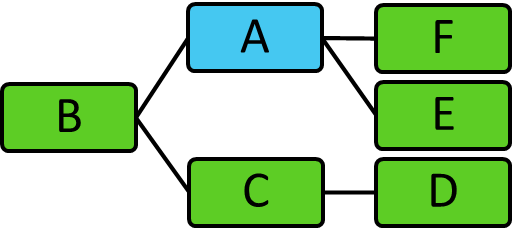
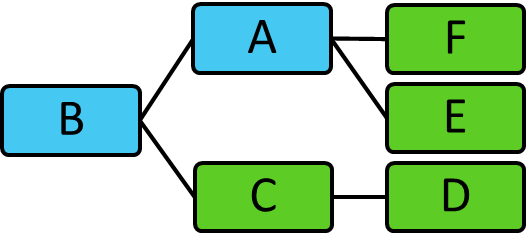
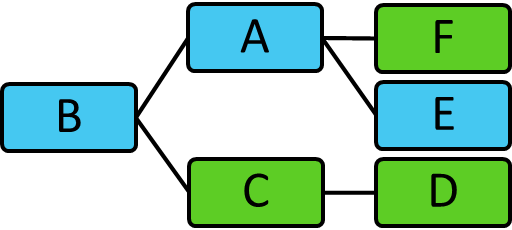
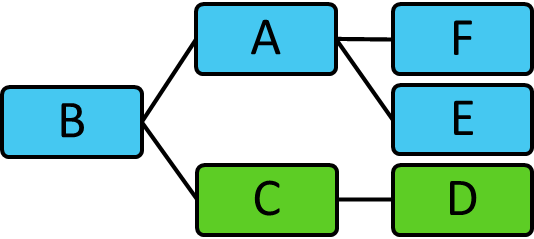
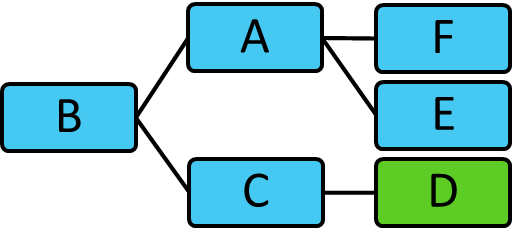
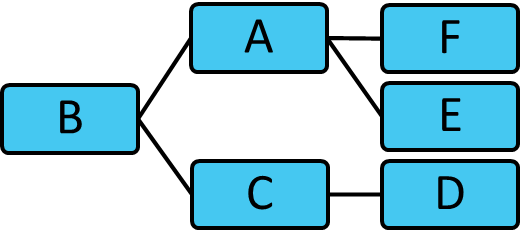
* BFS will provide a solution if any solution exists.
* If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

- Khó khăn:

* It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
* BFS needs lots of time if the solution is far away from the root node.

d. Ví dụ

|  |  |  |  |
| --- | --- | --- | --- |
|  | Current | Opened set | Closed set |
| B0 |  | [A] |  |
| B1 | A | [B,E,F] | A |
| B2 | B | [E,F,C] | B |
| B3 | E | [F,C] | E |
| B4 | F | [C] | F |
| B5 | C | [D] | C |
| B6 | D | [] | D |



2. So sánh giữa các thuật toán:

a. Giữa DFS và BFS

|  |  |  |
| --- | --- | --- |
| Tiêu chí | DFS | BFS |
| CTDL được sử dụng | Stack | Queue |
| Định nghĩa | Tiến hành duyệt các nodes xa  nhất có thể cho đến khi không còn node xung quanh chưa được duyệt | Duyệt toàn bộ nodes trên cùng một mức trước khi di chuyển tới mức tiếp theo |
| Kỹ thuật | Phải duyệt qua nhiều cạnh để  duyệt đến đỉnh cần tìm từ một  đỉnh nguồn | Có thể sử dụng để tìm kiếm  đường đi ngắn nhất trong đồ thị  không trọng số. Vì trong BFS,  có thể duyệt đến một đỉnh với  số cạnh nhỏ nhất từ một đỉnh |
| Thời gian | O(V+E) :Khi là một danh sách kề  O(V^2): Khi là ma trận kề  Với V cạnh và E đỉnh | O(V+E) :Khi là một danh sách kề  O(V^2): Khi là ma trận kề  Với V cạnh và E đỉnh |
| Bộ nhớ | DFS sử dụng ít bộ nhớ hơn | BFS sử dụng nhiều bộ nhớ hơn |
| Space complexity | DFS có ít hơn vì trong cùng  một thời gian DFS chỉ lưu trữ một đường đi đơn từ gốc đến  node lá | Nhiều hơn |
| Tốc độ | DFS nhanh hơn so với BFS | BFS chậm hơn so với DFS |

# Bảng phân công công việc

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Người phụ trách** | **Mô tả nội dung công việc** | **Bắt đầu** | **Kết thúc** | **Kết quả mong đợi** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |