Thuật toán Depth First Search

DATA STRUCTURE & ALGORITHM

STDIO Thuật toán Depth First Search (Tìm kiếm theo chiều sâu) cùng với Thuật toán Breadth First Search là một trong hai thuật toán tìm kiếm cơ bản trong bộ môn Trí tuệ nhân tạo, là tiền đề chuẩn bị để tìm hiểu các thuật toán phức tạp hơn.

Nội dung bài viết

Giới thiệu

Bài viết giới thiệu, nêu khái quát và trình bày về thuật toán Depth First Search (DFS – Tìm kiếm theo chiều sâu), cùng với thuật toán Breadth First Search là hai thuật toán cơ bản để chuẩn bị ra các thuật toán phức tạp hơn trong bộ môn Trí tuệ nhân tạo.

Tiền đề bài viết

Bài viết này nằm trong loạt bài tìm hiểu về thuật toán tìm kiếm của STDIO :: www.stdio.vn.

Đối tượng hướng đến

Những người muốn tìm hiểu về kiến thức cơ bản về trí tuệ nhân tạo nói chung, các thuật toán tìm kiếm nói riêng. STDIO

Thuật toán Depth First Search

Thuật toán Depth First Search (DFS – Tìm kiếm theo chiều sâu) là một dạng thuật toán duyệt hoặc tìm kiếm trên cây hoặc đồ thị. Trong lý thuyết khoa học máy tính, thuật toán DFS nằm trong chiến lược tìm kiếm mù (tìm kiếm không có định hướng, không chú ý đến thông tin, giá trị được duyệt) được ứng dụng để duyệt hoặc tìm kiếm trên đồ thị.

Ý tưởng thuật toán

Từ đỉnh (nút) gốc ban đầu, thuật toán duyệt đi xa nhất theo từng nhánh, khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo. Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.

Thuật giải

Trước khi bắt đầu, tôi sẽ nêu ra một số quy ước để dễ dàng trong trình bày:

- Open: là tập hợp các đỉnh chờ được xét ở bước tiếp theo theo ngăn xếp (ngăn xếp: dãy các phần tử mà khi thêm phần tử vào sẽ thêm vào đầu dãy, còn khi lấy phần tử ra sẽ lấy ở phần tử đứng đầu dãy).
- Close: là tập hợp các đỉnh đã xét, đã duyệt qua.
- s: là đỉnh xuất phát, đỉnh gốc ban đầu trong quá trình tìm kiếm.
- g: đỉnh đích cần tìm.
- p: đỉnh đang xét, đang duyệt.

Trình bày thuật giải:

- Bước 1: Tập Open chứa đỉnh gốc s chờ được xét.
- Bước 2: Kiểm tra tập Open có rỗng không.
  - Nếu tập Open không rỗng, lấy một đỉnh ra khỏi tập Open làm đỉnh đang xét p. Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm.
  - Nếu tập Open rỗng, tiến đến bước 4.
- Bước 3: Đưa đỉnh p vào tập Close, sau đó xác định các đỉnh kề với đỉnh p vừa xét. Nếu các đỉnh kề không thuộc tập Close, đưa chúng vào đầu tập Open. Quay lại bước 2.
- Bước 4: Kết luận không tìm ra đỉnh đích cần tìm.

Code

Tôi sẽ hiện thực thuật toán này theo ma trận kề bằng ngôn ngữ C++.

```
#include <stdio.h>
#include <stack>
using namespace std;

// dinh nghia lop do thi
class Graph
{
private:
    int n;
    int **edge;
public:
    Graph(int size = 2);
    ~Graph();
    bool isConnected(int, int);
    void addEdge(int x, int y);
    void depthFirstSearch(int, int);
};

Graph::Graph(int size)
{
    int i, j;
```

```
22.
23.     // xac dinh so dinh cua do thi
24.     if (size < 2)
25.         n = 2;
26.     else
27.         n = size;
28.
29.     // tao ra cac dinh trong do thi
30.     edge = new int*[n];
31.     for (i = 0; i < n; i++)
32.         edge[i] = new int[n];
33.
34.     // mac dinh giua cac dinh khong co ket noi voi nhau (= 0)
35.     for (i = 0; i < n; i++)
36.         for (j = 0; j < n; j++)
37.             edge[i][j] = 0;
38. }
39.
40. Graph::~Graph()
41. {
42.     for (int i = 0; i < n; ++i)
43.         delete[] edge[i];
44.     delete[] edge;
45. }
46.
47. // kiem tra giua hai dinh co ke nhau hay khong
48. bool Graph::isConnected(int x, int y)
49. {
50.     if (edge[x - 1][y - 1] == 1)
51.         return true;
52.
53.     return false;
54. }
55.
56. // thêm cạnh nối đỉnh x và đỉnh y
57. void Graph::addEdge(int x, int y)
58. {
59.     if (x < 1 || x > n || y < 1 || y > n)
60.         return;
61.
62.     edge[x - 1][y - 1] = edge[y - 1][x - 1] = 1;
63. }
64.
65. void Graph::depthFirstSearch(int s, int g)
66. {
67.     if (s > n || s < 0 || g > n || g < 0)
68.     {
69.         printf("Could not traverse this graph with your request\n");
70.         return;
71.     }
72.
73.     stack <int> open;
74.     bool *close = new bool[n];
75.     int i;
76.     int p;
77.
78.     // mac dinh cac dinh chua duoc duyet
79.     for (i = 0; i < n; i++)
80.         close[i] = false;
81.
82.     // dua dinh goc s vao stack open, chuan bi duyet
83.     open.push(s);
84.
85.     printf("With Depth first Search , we have vertex(s):\n");
86.
87.     while (!open.empty())
88.     {
89.         // lay mot dinh ra khoi open tro thanh dinh dang xet p
90.         do
91.         {
92.             if (open.empty())
93.                 return;
94.
```

```
            p = open.top();
            open.pop();
        } while (close[p - 1] == true);

        // in ra dinh dang xet
        printf("%d ", p);

        // p da duyet qua
        close[p - 1] = true;

        // ket thuc duyet khi tim ra ket qua can tim
        if (p == g)
            return;

        // tim dinh ke voi dinh dang xet, dinh nao chua duoc duyet dua vao open
        for (i = 1; i <= n; i++)
        {
            if (isConnected(p, i) && !close[i - 1])
            {
                open.push(i);
            }
        }
    }
    printf("\n");

    delete[] close;
}
```

Ví dụ

Tôi có một đồ thị như hình vẽ.

Tôi khởi tạo một đồ thị có 8 đỉnh và có cạnh nối các đỉnh kề như hình vẽ.

```
void main()
{
   // khoi tao do thi
    Graph g(8);

    // tao canh noi giua cac dinh ke
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(1, 5);
    g.addEdge(2, 6);
    g.addEdge(3, 4);
    g.addEdge(3, 8);
    g.addEdge(4, 8);
    g.addEdge(5, 8);
    g.addEdge(6, 7);
    g.addEdge(6, 8);
}
```

Tôi chọn đỉnh gốc để duyệt là đỉnh 1 và cần tìm đỉnh 4, tôi thực hiện duyệt ngay sau khi tạo cạnh nối giữa các đỉnh kề.

```
void main()
{
   // khoi tao do thi
    Graph g(8);

    // tao canh noi giua cac dinh ke
    g.addEdge(1, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(1
```

```
11.     g.addEdge(2, 6);
12.     g.addEdge(3, 4);
13.     g.addEdge(3, 8);
14.     g.addEdge(4, 8);
15.     g.addEdge(5, 8);
16.     g.addEdge(6, 7);
17.     g.addEdge(6, 8);
18.
19.     // duyet bang Depth First Search
20.     g.depthFirstSearch(1, 4);
21. }
```

Bảng bên dưới thể hiện quá trình duyệt từ đỉnh 1 đến đỉnh 4.

Đỉnh đang xét	Các đỉnh kề	Open	Close
		{1}	{}
1	{2; 3; 4; 5}	{5; 4; 3; 2}	{1}
5	{1; 8}	{8; 4; 3; 2}	{1; 5}
8	{3; 4; 5; 6}	{6; 4; 3; 4; 3; 2}	{1; 5; 8}
6	{2; 7; 8}	{7; 2; 4; 3; 4; 3; 2}	{1; 5; 8; 6}
7	{6}	{2; 4; 3; 4; 3; 2}	{1; 5; 8; 6; 7}
2	{1; 6}	{4; 3; 4; 3; 2}	{1; 5 ;8 ;6; 7; 2}
4	{1; 3; 8}	{3; 4; 3; 3; 2}	{1; 5; 8; 6; 7; 2; 4}

Nhận xét

Ưu điểm

- Xét duyệt tất cả các đỉnhđể trả về kết quả.
- Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả.

Khuyết điểm

- Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn.
- Mang tính chất mù quáng, duyệt tất cả đỉnh, không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết.

Tags depth first search thuật toán

Trần Minh Thắng

THẢO LUẬN

STDIO? Bugs

Liên Hệ

CÔNG TY TNHH STDIO - A18, TRUNG ĐÔNG PLAZA, 30, TRỊNH ĐÌNH THẢO, HÒA THẠNH, TÂN PHÚ, HỒ CHÍ MINH
028.36205514 - DEVELOPER@STDIO.VN
383/1 QUANG TRUNG, PHƯỜNG 10, QUẬN GÒ VẤP, THÀNH PHỐ HỒ CHÍ MINH
SỐ GIẤY PHÉP ĐKKD: 0311563559 DO SỞ KẾ HOẠCH VÀ ĐẦU TƯ TPHCM CẤP NGÀY 23/02/2012