

HỘI THẢO CÁC TRƯỜNG THPT CHUYÊN KHU VỰC
DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ – NĂM 2023

---o0o---

CHUYÊN ĐỀ

BFS VÀ CÁC ỨNG DỤNG

Tháng 7/2023

MỤC LỤC

Phần I. Phần mở đầu	3
Phần II. Nội dung	4
1. Lý thuyết	4
1.1 Thuật toán	4
1.2 Các đặc tính của thuật toán BFS	6
2. Ứng dụng của BFS – Bài tập vận dụng	8
2.1 Đếm số thành phần liên thông	8
2.2 Dầu Loang	10
2.3 Kiến sẵn mồi	14
2.4 Nước ngập	18
2.5 Đẩy hộp	21
3. Bài tập áp dụng	26
Phần III. Kết luận	27
Tài liệu tham khảo	28

PHẦN I: MỞ ĐẦU

Trong các kì thi học sinh giỏi môn tin học những năm gần đây ngày càng được nâng cao về chất lượng, nội dung thi có tính tổng hợp kiến thức rất cao, kết hợp nhiều kĩ thuật xử lý. Việc cung cấp kiến thức nền tảng và kĩ năng lập trình làm cơ sở cho học sinh là rất cần thiết. Các bài toán vận dụng lý thuyết đồ thị là một trong những nội dung quan trọng và có nhiều thuật toán liên quan. Trong phạm vi chuyên đề, tôi xin trình bày một phần nhỏ của nội dung này đó là **“BFS và các ứng dụng”**. Đây là phần kiến thức giúp học sinh bắt đầu tiếp cận các dạng bài toán trên đồ thị. Hệ thống bài tập được tìm hiểu, sưu tầm từ các tài liệu tin học và các trang web lập trình trực tuyến. Tôi hy vọng sẽ hỗ trợ một phần nhỏ cho quý thầy cô và học sinh trong công tác giảng dạy và học tập.

PHẦN II: NỘI DUNG

1. Lý thuyết

1.1 Thuật toán

Thuật toán **duyệt đồ thị ưu tiên chiều rộng** (*Breadth-first search - BFS*) là một trong những thuật toán tìm kiếm cơ bản và thiết yếu trên đồ thị. Mà trong đó, những đỉnh nào gần đỉnh xuất phát hơn sẽ được duyệt trước.

Ứng dụng của BFS có thể giúp ta giải quyết tốt một số bài toán trong thời gian và không gian tối thiểu. Đặc biệt là bài toán tìm kiếm đường đi ngắn nhất từ một đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số hoặc tất cả trọng số bằng nhau, thuật toán sẽ luôn trả ra đường đi ngắn nhất có thể. Ngoài ra, thuật toán này còn được dùng để tìm các thành phần liên thông của đồ thị, hoặc kiểm tra đồ thị hai phía, ...

Ý tưởng

Với đồ thị không trọng số và đỉnh nguồn s . Đồ thị này có thể là đồ thị có hướng hoặc vô hướng, điều đó không quan trọng đối với thuật toán.

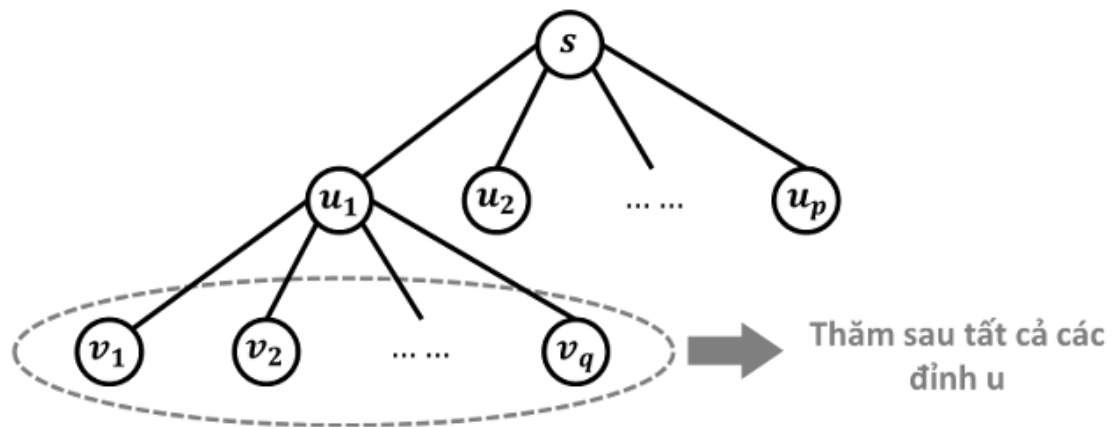
Có thể hiểu thuật toán như một ngọn lửa lan rộng trên đồ thị:

- Ở bước thứ 0, chỉ có đỉnh nguồn s đang cháy.
- Ở mỗi bước tiếp theo, ngọn lửa đang cháy ở mỗi đỉnh lại lan sang tất cả các đỉnh kề với nó.

Trong mỗi lần lặp của thuật toán, "vòng lửa" lại lan rộng ra theo chiều rộng. Những đỉnh nào gần s hơn sẽ bùng cháy trước.

Chính xác hơn, thuật toán có thể được mô tả như sau:

- Đầu tiên ta thăm đỉnh nguồn s .
- Việc thăm đỉnh s sẽ phát sinh thứ tự thăm các đỉnh (u_1, u_2, \dots, u_p) kề với s (những đỉnh gần s nhất). Tiếp theo, ta thăm đỉnh u_1 , khi thăm đỉnh u_1 sẽ lại phát sinh yêu cầu thăm những đỉnh (v_1, v_2, \dots, v_q) kề với u_1 . Nhưng rõ ràng những đỉnh v này "xa" s hơn những đỉnh u nên chúng chỉ được thăm khi tất cả những đỉnh u đều đã được thăm. Tức là thứ tự thăm các đỉnh sẽ là: $s, u_1, u_2, \dots, u_p, v_1, v_2, \dots, v_q, \dots$



Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách để chứa những đỉnh đang “chờ” thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách, loại nó ra khỏi danh sách và cho những đỉnh kề với nó chưa được thăm xếp hàng vào cuối danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng.

Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi (*queue*) để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng.

Bước 1: Khởi tạo

- Các đỉnh đều ở trạng thái chưa được đánh dấu. Ngoại trừ đỉnh nguồn s đã được đánh dấu.
- Một hàng đợi ban đầu chỉ chứa 1 phần tử là s .

Bước 2: Lặp lại các bước sau cho đến khi hàng đợi rỗng:

- Lấy đỉnh u ra khỏi hàng đợi.
- Xét tất cả những đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
 - Đánh dấu v đã thăm.
 - Lưu lại vết đường đi từ u đến v .
 - Đẩy v vào trong hàng đợi (đỉnh v sẽ chờ được duyệt tại những bước sau).

Bước 3: Truy vết tìm đường đi.

Cài đặt

```
int n; // Số lượng đỉnh của đồ thị
int d[maxN], par[maxN];
bool visit[maxN];
```

```

vector<int> g[maxN];
void bfs(int s) { // Với s là đỉnh xuất phát (đỉnh nguồn)
    fill_n(d, n + 1, 0);
    fill_n(par, n + 1, -1);
    fill_n(visit, n + 1, false);
    queue<int> q;  q.push(s);  visit[s] = true;
    while (!q.empty()) {
        int u = q.front();  q.pop();
        for (auto v : g[u]) {
            if (!visit[v]) {
                d[v] = d[u] + 1;  par[v] = u; visit[v] = true; q.push(v);
            }
        }
    }
}

```

Truy vết

- Cài đặt truy vết đường đi từ đỉnh nguồn s đến đỉnh u:

```

if (!visit[u]) cout << "No path!";
else {
    vector<int> path;
    for (int v = u; v != -1; v = par[v])
        path.push_back(v);
    reverse(path.begin(), path.end());

    cout << "Path: ";
    for (auto v : path) cout << v << ' ';
}

```

1.2 Các đặc tính của thuật toán BFS

Nếu sử dụng một ngăn xếp (*stack*) thay vì hàng đợi (*queue*) thì ta sẽ thu được thứ tự duyệt đỉnh của thuật toán tìm kiếm theo chiều sâu (*Depth First Search – DFS*). Đây chính là phương pháp khử đệ quy của DFS để cài đặt thuật toán trên các ngôn ngữ không cho phép đệ quy.

Định lý: Thuật toán BFS cho ta độ dài đường đi ngắn nhất từ đỉnh nguồn tới mọi đỉnh (với khoảng cách tới đỉnh u bằng $d[u]$). Trong thuật toán BFS, nếu đỉnh u gần đỉnh nguồn hơn đỉnh v, thì u sẽ được thăm trước.

- Chứng minh: Trong BFS, từ một đỉnh hiện tại, ta luôn đi thăm tất cả các đỉnh kề với nó trước, sau đó thăm tất cả các đỉnh cách nó một đỉnh, rồi các đỉnh cách nó hai đỉnh, v.v... Như vậy, nếu từ một đỉnh u khi ta chạy BFS, quãng đường đến đỉnh v luôn là quãng đường đi qua ít cạnh nhất.

2. Ứng dụng BFS – Bài tập vận dụng

2.1 Đếm số thành phần liên thông

Đề bài

Cho đơn đồ thị vô hướng gồm n đỉnh và m cạnh ($1 \leq n, m \leq 10^5$), các đỉnh được đánh số từ 1 tới n . Tìm số thành phần liên thông của đồ thị.

Dữ liệu: vào từ file connect.inp gồm

- Dòng đầu chứa 2 số n, m
- M dòng tiếp theo mỗi dòng chứa hai số u, v thể hiện cạnh nối giữa đỉnh u và v

Kết quả: Ghi ra file connect.out số lượng thành phần liên thông của đồ thị

Ví dụ:

Connect.inp	Connect.out
16 14 1 2 1 3 2 4 3 4 5 6 5 7 7 8 7 9 8 10 11 12 11 13 12 13 12 14 14 15	4

Ý tưởng

Một đồ thị có thể liên thông hoặc không liên thông. Nếu đồ thị liên thông thì số thành phần liên thông của nó là 1. Điều này tương đương với phép duyệt theo thủ tục BFS được gọi đến **đúng một lần**. Nếu đồ thị không liên thông (số thành phần liên thông lớn hơn 1) ta có thể tách chúng thành những **đồ thị con liên thông**. Điều này cũng có nghĩa là trong phép duyệt đồ thị, số thành phần liên thông của nó bằng số lần gọi tới thủ tục BFS.

Thuật toán

Thuật toán ứng dụng BFS để xác định thành phần liên thông:

- **Bước 0:** Khởi tạo số lượng thành phần liên thông bằng 0.

- **Bước 1:** Xuất phát từ một đỉnh chưa được đánh dấu của đồ thị. Ta đánh dấu đỉnh xuất phát, tăng số thành phần liên thông thêm 1.
- **Bước 2:** Từ một đỉnh i đã đánh dấu, ta đánh dấu tất cả các đỉnh j kề với i mà j chưa được đánh dấu.
- **Bước 3:** Thực hiện bước 2 cho đến khi không còn thực hiện được nữa.
- **Bước 4:** Nếu số số đỉnh đánh dấu bằng n (mọi đỉnh đều được đánh dấu) kết thúc thuật toán và trả về số thành phần liên thông, ngược lại quay về bước 1.

Cài đặt

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
const int maxN = 1e5 + 7;
```

```
int n, m, components = 0;
```

```
bool visit[maxN];
```

```
vector<int> g[maxN];
```

```
void bfs(int s) {
    ++components;
    queue<int> q;
    q.push(s);
    visit[s] = true;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (auto v : g[u]) {
            if (!visit[v]) {
                visit[v] = true;
                q.push(v);
            }
        }
    }
}
```

```
int main() {
    cin >> n >> m;
```

```

while (m--) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
}

fill_n(visit, n + 1, false);
for (int i = 1; i <= n; ++i)
    if (!visit[i]) bfs(i);
cout << components;
}

```

Đánh giá

Ta cũng có thể sử dụng thuật toán tìm kiếm theo chiều sâu (*Depth First Search – DFS*) để xác định thành phần liên thông.

Độ phức tạp

Độ phức tạp của thuật toán là $O(n+m)$.

2.2 Dầu loang

Đề bài

Một tai nạn hàng hải đã khiến dầu tràn ra biển. Để có được thông tin về mức độ nghiêm trọng của thảm họa này, người ta phải phân tích các hình ảnh chụp từ vệ tinh, từ đó tính toán chi phí khắc phục cho phù hợp. Đối với điều này, số lượng vết dầu loang trên biển và kích thước của mỗi vết loang phải được xác định. Vết loang là một mảng dầu nổi trên mặt nước.

Để tiện cho việc xử lý, hình ảnh được chuyển đổi thành một ma trận nhị phân kích thước $N \times M$ ($1 \leq N, M \leq 250$). Với 1 là ô bị nhiễm dầu, và 0 là ô không bị nhiễm dầu. Vết dầu loang là tập hợp của một số ô bị nhiễm dầu có chung cạnh.

Họ đã thuê bạn để giúp họ xử lý hình ảnh. Công việc của bạn là đếm số lượng vết loang trên biển và kích thước tương ứng của từng vết.

Dữ liệu: vào từ file `dauloang.inp` gồm

- Dòng đầu là hai số n, m
- N dòng tiếp theo, mỗi dòng chứa m số 0 hoặc 1 tương ứng với bằng 1 là ô (i,j) tương ứng có vết dầu loang

Kết quả: ghi ra file dauloang.out

- Dòng 1 ghi ra số lượng vết dầu loang
- Các dòng tiếp theo in ra nhiều dòng, mỗi dòng là độ lớn và số lượng vết dầu loang tương ứng với độ lớn đó theo thứ tự độ lớn vết dầu tăng dần

Ví dụ

Dauloang.inp	Dauloang.out
5 5	4
1 1 1 0 0	1 1
1 1 0 0 1	3 2
0 0 0 1 1	5 1
1 0 0 0 0	
0 1 1 1 0	

Ý tưởng

Ta xây dựng một **mô hình đồ thị** của bài toán như sau:

- Gọi mỗi đỉnh của đồ thị tương ứng với mỗi ô 1 (*ô bị nhiễm dầu*) của ma trận.
- Tồn tại một cạnh nối giữa cặp đỉnh (u,v) khi và chỉ khi ô tương ứng với đỉnh u kề cạnh với ô tương ứng với đỉnh v và cả hai ô đều là ô 1.

Khi đó, bài toán quy về thành bài toán **xác định thành phần liên thông của đồ thị**.

Trong đó, mỗi thành phần liên thông tương ứng với mỗi một vết dầu loang.

Nghĩa là, số lượng thành phần liên thông của đồ thị chính là số lượng vết dầu loang. Và số lượng đỉnh nằm trong cùng một thành phần liên thông là kích thước của vết loang tương ứng.

Thuật toán

Áp dụng **thuật toán loang trên ma trận** để xác định thành phần liên thông:

- Khởi tạo số lượng vết dầu bằng 0.
- Duyệt dần từng ô của ma trận, nếu ô đang xét là một ô bị nhiễm dầu (ô 1) và chưa được đánh dấu:
 - Đánh dấu lại ô đó.
 - Tăng số lượng vết dầu thêm 1.

- Thực hiện thủ tục BFS xuất phát từ ô đó để loang ra các ô xung quanh như sau:
 - Khởi tạo kích thước của vết dầu đang xét là 1.
 - Tiếp tục thực hiện công việc sau cho đến khi không còn thực hiện được nữa: Từ một ô đã đánh dấu, ta đánh dấu tất cả các ô bị nhiễm dầu kề cạnh với ô đó mà chưa được đánh dấu. Mỗi lần đánh dấu lại một ô thì ta tăng kích thước của vết dầu thêm 1.
 - Sử dụng 1 mảng để lưu lại kích thước của từng vết loang.
- Nếu tất cả các ô bị nhiễm dầu đều đã được đánh dấu, trả ra kết quả và kết thúc thuật toán.

Cài đặt

Đánh giá

Ta sử dụng 2 mảng `moveX[]` và `moveY[]` để có thể dễ dàng duyệt qua tất cả các ô kề cạnh với ô đang xét.

Độ phức tạp

Với mỗi bộ test:

- Vì mỗi ô của ma trận được duyệt đúng duy nhất 1 lần nên ta sẽ mất độ phức tạp $O(N \times M)$.
- Ta sẽ mất thêm $O(4 \times N \times M)$ vì ta phải duyệt qua 4 ô kề cạnh với mỗi ô của ma trận.

Nhìn chung, độ phức tạp của thuật toán là $O(t \times (N \times M + 4 \times N \times M))$. Với t là số lượng bộ test.

2.3 Kiến sẵn mồi

Một khu vườn được xem xét như một lưới ô vuông, có một tổ kiến ở ô có tọa độ (0,0) và có một số ô trên lưới có vật cản. Một chú kiến muốn đi tìm thức ăn, kiến sẽ đi theo quy tắc sau:

- Từ một ô kiến có thể đi được sang 4 ô chung cạnh
- Kiến không đi vào ô có vật cản
- Kiến không đi xa tổ quá S bước

Yêu cầu: cho biết tọa độ các ô có vật cản và số S , hỏi kiến có thể đến được tất cả bao nhiêu ô.

Dữ liệu: vào từ file ant.inp gồm

- Dòng đầu là 2 số C và S ($0 \leq C$ – là số ô có vật cản $\leq 10^4$; $1 \leq S \leq 10^7$)
- C dòng sau, mỗi dòng là 2 số nguyên x_i và y_i là tọa độ của các ô chứa vật cản ($|x_i|, |y_i| < 1001$)

Kết quả: ghi ra file ant.out một số duy nhất là số ô mà kiến có thể đến được.

Ví dụ:

Ant.inp	Ant.out	Hình mô tả ví dụ
4 5 -1 1 0 -1 0 1 1 0	26	

Thuật toán

- Sử dụng BFS để đếm số ô đến được từ tổ
- Nhận xét rằng nếu S lớn ta có thể bị quá thời gian cho phép, tuy nhiên tọa độ của các vật cản được giới hạn trong ô vuông có tọa độ 10^3 , khi đó hình vuông có tọa độ trái trên là $(-1001, 1001)$ và phải dưới $(1001, -1001)$ sẽ chứa tất cả vật cản, gọi số bước từ tổ đến các ô nằm trên cạnh của hình vuông này là $kc[i, j]$ khi đó số ô đến được từ ô (i, j) sẽ là:
 - + $S - kc[i, j]$ nếu (i, j) nằm trên cạnh của hình vuông trên
 - + $(S - kc[i, j] + 1) * (S - kc[i, j] + 2) / 2 - 1$ nếu (i, j) nằm ở góc của hình vuông
- Vì mảng không lưu được chỉ số âm nên ta có thể tịnh tiến tọa độ của tất cả các ô lên 1001. Trong quá trình BFS ta chỉ BFS đã đến đủ S bước hoặc BFS đến các ô nằm trong giới hạn 2002.

Cài đặt:**Đánh giá**

Độ phức tạp $O(\min(S, 2 * 10^6))$

2.4 Nước ngập

Đề bài

Trong khu rừng mê hoặc nước đang dâng lên bởi vì tên vua độc ác Cactus đã phá bỏ đập nước. Có cậu bé đang ở trong rừng và phải nhanh chóng về một địa điểm an toàn trước khi quá muộn.

Khu rừng mê hoặc có R hàng và C cột. Nếu là ô trống thì ký hiệu là '.', nếu là ô bị ngập nước ký hiệu là '*' còn nếu là ô có chứa đá ký hiệu là 'X'. Địa điểm an toàn ký hiệu là 'D' và vị trí xuất phát của cậu bé ký hiệu là 'S'.

Mỗi phút, cậu bé có thể có thể di chuyển đến một trong 4 ô chung cạnh (trên, dưới, trái, phải). Cũng mỗi phút, nước có thể mở rộng sang các ô trống chung cạnh với ô đang có nước. Chú ý rằng cậu bé không thể đi qua những ô có đá, cũng không thể đi qua những ô có nước. Và nước cũng không bao giờ ngập vị trí an toàn.

Viết chương trình tìm đường đi với thời gian ngắn nhất có thể để cậu bé có thể đến được vị trí an toàn.

Chú ý: Cậu bé không thể đến một ô nếu như ở cùng thời điểm đến, ô này bị ngập nước.

Dữ liệu: vào từ file drown gồm

- Dòng đầu tiên chứa hai số nguyên dương R và C không vượt quá 50
- R dòng sau, mỗi dòng chứa C ký tự ('.', '*', 'X', 'D', 'S'). Trên bản đồ có đúng một ô có chữ 'D' và đúng một ô có chữ 'S'.

Kết quả: ghi ra file drown.out

Ghi ra một số nguyên là thời gian ngắn nhất có thể hoặc ghi ra 'IMPOSSIBLE' nếu như điều này là không thể.

Ví dụ:

Drown.inp	Drown.out
3 3 D.*S.	3

Thuật toán

Trước tiên ta tính thời gian ngập nước $t[i,j]$ của tất cả các ô bằng cách BFS với các đỉnh xuất phát là các ô hiện đang có nước (với ô đích đến và các ô đá coi $t[i,j]=\infty$) Tiếp theo

BFS từ ô xuất phát của cậu bé với $kc[i,j]$ là khoảng cách nhanh nhất từ xuất phát đến (i,j) điều kiện chỉ đến được (i,j) khi $kc[i,j] < t[i,j]$.

Cài đặt

Đánh giá

Độ phức tạp $O(n*m)$