BFS 0-1

BFS 0-1 và các ứng dụng

Tác giả:

- Phạm Công Minh - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN

Reviewer:

- Nguyễn Minh Nhật - THPT chuyên Khoa học Tự nhiên, ĐHQGHN
- Lê Minh Hoàng - Đại học Khoa học Tự nhiên, ĐHQG-HCM

Giới thiệu

Lỡ một ngày bạn ngồi trong phòng thi quốc gia, bạn gặp một đề bài như sau:

> Cho một đồ thị n đỉnh m cạnh ($1 \leq n, m \leq 2 \times 10^6$), cạnh thứ i có trọng số là w_i ($0 \leq w_i \leq 1$). Tìm đường đi ngắn nhất từ s đến t ($1 \leq s, t \leq n$).

Bạn hì hục cài Dijkstra, tuy nhiên code lại chạy quá chậm để qua được giới hạn thời gian. Lúc ấy bạn mới nhớ lại blog về 0-1 BFS trên VNOI wiki, thứ mà bạn sắp đọc bây giờ.

0-1 BFS có thể thực hiện tìm kiếm đường đi ngắn nhất từ một nguồn (Single source shortest path) trong độ phức tạp tuyến tính, với điều kiện là cạnh có trọng số bằng 0 hoặc 1.

Thuật toán

0-1 BFS có rất nhiều điểm tương đồng với **BFS** và **Dijkstra**. Thay vì sử dụng hàng đợi (queue) để chứa các đỉnh sẽ được duyệt, ta sử dụng hàng đợi hai đầu (double-ended queue, thường gọi tắt là **deque**) để chứa các đỉnh sẽ được duyệt.

Bước 1: Khởi tạo

- Khởi tạo mảng D trong đó D_x là độ dài đường đi ngắn nhất từ s đến x, ban đầu $D_s = 0$, $D_v = \infty \, \forall \, v \neq s$.
- Ngoài ra ta cần một mảng khác để quản lý xem một đỉnh đã được xử lý hay chưa.
- deque ban đầu chỉ chứa s.

Bước 2: Thực hiện đến khi deque rỗng

- Lấy đỉnh u ra khỏi đầu hàng đợi.

- Nếu đỉnh u đã được xử lý, ta bỏ qua đỉnh này.
- Với mọi đỉnh v có cạnh nối từ u đến v, gọi w là trọng số của cạnh từ u đến v. Nếu $D_v > D_u + w$, ta cập nhật $D_v = D_u + w$ và thêm vào hàng đợi:
  - Nếu $w = 1$, ta đẩy v vào **cuối** hàng đợi.
  - Nếu $w = 0$, ta đẩy v vào **đầu** hàng đợi.

Chứng minh

Tính đúng đắn của 0-1 BFS có thể chứng minh như thuật toán Dijkstra. Tuy nhiên Dijkstra sử dụng cấu trúc dữ liệu hàng đợi ưu tiên để tìm đỉnh x có D_x nhỏ nhất trong các đỉnh với độ phức tạp $O(\log n)$ mỗi thao tác, còn 0-1 BFS tìm đỉnh này trong $O(1)$.

Trong hàng đợi hai đầu, gọi u là đỉnh nằm ở đầu hàng đợi, các đỉnh v trong hàng đợi có D_v tăng dần từ đầu đến cuối hàng đợi và D_v chỉ có thể bằng D_u hoặc $D_u + 1$:

$$deque = \underbrace{u, \ldots, v}_{D_v}, \underbrace{m, \ldots, n}_{D_v+1}$$

Ta sẽ chứng minh điều này bằng phương pháp quy nạp:

- Ban đầu hàng đợi chỉ có 1 đỉnh là s, nên chắc chắn hàng đợi thỏa mãn điều kiện trên.
- Giả sử hàng đợi đã thỏa mãn điều kiện trên. Khi ta lấy đỉnh u ra khỏi hàng đợi, sẽ có các đỉnh được thêm vào hàng đợi, cụ thể như sau:
  - Trong trường hợp đỉnh v được cập nhật lại $D_v = D_u + 0$, vì D_u là giá trị D nhỏ nhất trong hàng đợi nên khi đẩy vào **đầu** hàng đợi, nó vẫn sẽ thỏa mãn điều kiện sắp xếp tăng dần.
  - Trong trường hợp đỉnh v được cập nhật lại $D_v = D_u + 1$, vì $D_u + 1$ là giá trị D lớn nhất trong hàng đợi nên khi đẩy vào **cuối** hàng đợi, nó vẫn sẽ thỏa mãn điều kiện sắp xếp tăng dần.
  - Trong trường hợp D_v được cập nhật lại từ $D_u + 1$ thành D_u và đẩy lần thứ hai vào đầu hàng đợi, do ta bỏ qua lần duyệt thứ 2 của v nên chứng minh giống như trường hợp đầu tiên.

Từ đây ta cũng chứng minh được một đỉnh v chỉ được cập nhật D_v và đẩy vào hàng đợi tối đa 2 lần nên độ phức tạp sẽ tương đương thuật toán BFS là $O(|V| + |E|)$

Cài đặt

Cấu trúc dữ liệu:

- Biến `MAXN` : Độ lớn của mảng
- Mảng `D[]` : Tương tự trên
- Mảng `vis[]` : Mảng đánh dấu đỉnh đã được duyệt
- Vector `g[]` : Danh sách kề của các đỉnh
- Hàng đợi hai đầu `dq` : Chứa các đỉnh sẽ được duyệt

```
const int INF = 1e9;
const int MAXN = 2e6+5;

int n; // Số đỉnh của đồ thị
```

```cpp
int D[MAXN], vis[MAXN];
vector<pair<int, int>> g[MAXN];  // first là đầu mút của cạnh, second là trọng
deque<int> dq;

void BFS_01(int s){
    for(int i = 1; i <= n; i++){
        D[i] = INF;              // Khởi tạo
        vis[i] = 0;
    }
    D[s] = 0;
    dq.push_front(s);
    while(!dq.empty()){
        int u = dq.front();
        dq.pop_front();
        if(vis[u])continue;
        vis[u] = 1;              // Đánh dấu
        for(auto v: g[u]){
            if(D[v.first] > D[u] + v.second){              //
                D[v.first] = D[u] + v.second;             // Duyệt và sử lý c
                if(v.second == 1)dq.push_back(v.first);   //
                else dq.push_front(v.first);
            }
        }
    }
}
```

Nhận xét

Có rất ít bài toán yêu cầu bắt buộc phải sử dụng 0-1 BFS. Rất khó để ép thời gian sao cho Dijkstra chạy quá thời gian khi có rất nhiều đỉnh hoặc cạnh.

Thuật toán Dial (Bucket Dijkstra)

Ta có thể giải được bài toán tìm đường đi ngắn nhất từ một nguồn trong $O(|E| + |V| \times K)$ với K là trọng số cạnh lớn nhất trong đồ thị. Ta có thể lưu $K + 1$ hàng đợi, hàng đợi thứ i ban đầu lưu các đỉnh v có $D_v = i$. Khi đã lấy hết đỉnh trong hàng đợi i, ta có thể dùng lại hàng đợi đó để lưu các đỉnh v có $D_v = i + (K + 1)$, rồi $i + 2 \times (K + 1)$, ... Do các cạnh có trọng số không quá K, một đỉnh sẽ không bao giờ cập nhật vào một hàng đợi cách hàng đợi chứa đỉnh đó quá K, nên chỉ có tối đa $K + 1$ hàng đợi không rỗng tại mỗi thời điểm.

Ví dụ

Bài toán 1

Link nộp: Codeforces - Labyrinth

Đề bài

Bạn đang chơi một trò chơi điện tử. Một level nào đó đặt bạn vào một ô nào đó trong một mê cung n hàng m cột. Ta định nghĩa ô nằm trên hàng thứ a, cột thứ b là ô (a, b). Một ô có thể có chướng ngại vật hoặc không. Bạn chỉ được di chuyển trái, trên, phải, dưới, không được đi vào các ô có chướng ngại vật và không được ra ngoài mê cung.

Do bàn phím bạn sắp hỏng nên bạn chỉ được di chuyển sang trái không quá x lần và sang phải không quá y lần.

Biết rằng bạn đang bắt đầu ở ô (r, c), hãy đếm và in ra số ô trong mê cung đến được từ ô bắt đầu.

Giới hạn: $1 \leq n, m \leq 2000, 1 \leq x, y \leq 10^9, 1 \leq r \leq n, 1 \leq c \leq m$

Lời giải

Đây là một bài cơ bản để tập cài 0-1 BFS.

Giả sử ta muốn kiểm tra xem có đến được một ô (x, y) nào đó thì cần tối thiểu bao nhiêu lần đi trái/phải.

Gọi số lần đi sang trái là L, số lần đi sang phải là R. Dễ thấy $c - L + R = x \Rightarrow R - L = x - c = const$ Vì hiệu không đổi nên nếu ta tối thiểu hóa được L thì R cũng sẽ là tối thiểu.

Ta coi mê cung là một đồ thị, các ô vuông chung cạnh sẽ được nối với nhau trên đồ thị, cạnh nối có trọng số là 0 nếu lên/xuống/phải và là 1 nếu là trái. Sau khi chạy thuật toán, với mỗi ô ta sẽ biết được số bước sang trái tối thiểu, từ đó tính được số bước phải tối thiểu và kiểm tra với điều kiện đề bài.

Độ phức tạp thời gian: $O(n \times m)$

▶ *Code mẫu*

Bài toán 2

Link nộp: Codeforces - Chamber of Secrets

Đề bài

Cho căn phòng có thể chia thành bảng lưới gồm n hàng và m cột. Mỗi ô trong bảng lưới có thể chứa một chiếc gương ma thuật hoặc không chứa gì. Nếu kích hoạt gương ma thuật, ánh sáng chiếu vào gương sẽ bị phản chiếu ra 4 phía trái, trên, phải, xuống *(xem hình trong đề bài gốc để dễ hiểu hơn)*.

Có một tia sáng chiếu từ cạnh bên trái của ô $(1, 1)$ chiếu từ trái sang phải và một cánh cửa nằm trên cạnh bên phải của ô (n, m). Tìm số gương cần kích hoạt tối thiểu để tia sáng đến được cánh cửa.

Giới hạn: $2 \leq n, m \leq 1000$

Lời giải

Để có tia sáng tới được cánh cửa, tương đương với việc phải tồn tại một đường đi bắt đầu bằng một chiếc gương ở hàng 1, kết thúc bằng một chiếc gương ở hàng n, và hai gương liên tiếp cùng hàng hoặc cùng cột. Nói cách khác, cần tìm một dãy gương $\{(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)\}$ sao cho:

- $u_1 = 1$
- $u_k = n$

- u_i = u_{i+1} \style{font-family:Cambria Math}{\text{ hoặc }} v_i = v_{i+1} \forall \; 1 \leq i \leq k-1
- k là bé nhất

Ta có nhận xét: Nếu tồn tại i sao cho $u_{i-1} = u_i = u_{i+1}$ hoặc $v_{i-1} = v_i = v_{i+1}$ thì ta có thể bỏ chiếc gương thứ i này đi. Do đó, số gương tối ưu tương đương với số lần tia sáng phải đổi phương từ ngang thành dọc và ngược lại.

Đến đây, ta có lời giải như sau:

- Ta dựng hai đồ thị, mỗi đồ thị đều có $n \times m$ đỉnh, mỗi đỉnh tương ứng với một ô trong căn phòng.
- Ở đồ thị thứ nhất, tia sáng chỉ được đi ngang, tức là chỉ có cạnh hai chiều giữa ô (i, j) và $(i, j + 1)$ với $1 \leq j \leq m - 1$. Các cạnh này có trọng số là 0.
- Ở đồ thị thứ hai, tia sáng chỉ được đi dọc, tức là chỉ có cạnh hai chiều giữa ô (i, j) và $(i + 1, j)$ với $1 \leq i \leq n - 1$. Các cạnh này có trọng số là 0.
- Đối với các ô có gương, ở đây tia sáng được chuyển phương, nên nếu ô (i, j) có gương thì ô (i, j) ở đồ thị 1 và ô (i, j) ở đồ thị 2 được nối với nhau. Do số lần chuyển hướng tương đương đáp án, ta đặt trọng số cạnh mới này là 1.

Đáp án của bài toán là đường đi ngắn nhất từ ô $(1, 1)$ trong đồ thị 1 đến ô (n, m) trong đồ thị 1. Ta tìm đường đi ngắn nhất bằng 0-1 BFS.

Độ phức tạp thời gian: $O(n \times m)$

▶ *Code mẫu*

Bài tập tham khảo

- SPOJ - KATHTHI
- Codeforces - Three States
- Codeforces - Spiral Maximum
- Gym - Problem J: Jailbreak
- USACO Gold - Lasers and Mirrors