广度优先搜索

dbywsc

2025/7

目录

介绍

广度优先搜索（Breadth First Search）简称 **BFS** 。与 DFS 一样，原本是图论中的概念，但是也可以作为一种搜索算法。
广度优先搜索的原理是搜索到一个点时，优先搜索与它处在同一层的点，当前层的点搜索完后再搜索下一层。它的实现依靠的是**队列**。
创建一个存储点的队列。首先将起点入队并标记，之后当队列不空时，选取当前的**队首**出队，并依次将**直接与其相连**的点入队，以此往复，直到队列为空。

性质

下面是 BFS 算法的一些性质[1]：

1. 二段性：

可以发现，任何时候存储在队列中的所有的点的层数[2]要么等于当前的层数 k，要么是当前层数的下一层 $k+1$。

2. 最短路[3]：

由性质 1 可以发现，由于我们一定会优先的搜索离当前点最近的点，所以找到的到搜索到点的路径一定是最短的。

---

[1]关于本节提到的性质，我们不加以证明。有兴趣的同学可以参阅《算法导论》。

[2]由于 ppt 中插入图片不太方便，此处我们可以画一个图来表示。

[3]此处的最短路并不是图论中的“最短路算法”，但是许多最短路算法确实是借用了这个性质由 BFS 推广而来的。

模版

不同于 DFS ，由于 DFS 的本质是递归，作为搜索算法形式上灵活多变；但是 BFS 是有相应的模版[4]的。

```
void bfs(int s) {
    std::queue<points> q;
    q.push(s); dis[s] = 0; vis[s] = 1;
    while(q.size()) {
        auto u = q.front(); q.pop();
        for(auto v : G[u]) {
            if(!st[v]) {
                st[v] = 1;
                dis[v] = dis[u] + 1;
                q.push(v);
            }
        }
    }
}
```

[4]这里给出的实际上是图论中的 BFS 模版，但是在搜索问题中，BFS 的框架大致也和上面相同。

BFS 作为搜索算法，常用的场景有两个：
1. 由于 BFS 不依赖递归，没有段错误的风险，因此常被用来解决迷宫问题。
2. 鉴于 BFS 最短路的性质，有时可以用它解决一些别的问题[5]。

[5]事实上这已经属于图论的范畴了，我们后面会说到

BFS 解决迷宫问题 – I

B3625 迷宫寻路

事实上，用 BFS 解决迷宫问题是非常显然的。我们可以将起点入队，然后每次取出队首，依次枚举它的上下左右四个方向（有时是八个方向）是否可达，如果可达就标记并且继续入队。最后搜索结束后查看一下终点有没有被标记过就可以了。

BFS 解决迷宫问题 – II

```cpp
using PII = std::pair<int, int>;    //std::pair<type, type> 可以将两个变量凑成
                                    //一对，可以很方便的存储坐标

char G[N][N];   //存图
bool st[N][N];  //存有没有被走过
int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};
bool bfs(void) {
    std::queue<PII> q;
    q.push({1, 1}); st[1][1] = 1;
    while(q.size()) {
        auto u = q.front(); q.pop();
        for(int i = 0; i < 4; i++) {
            int a = u.first + dx[i], b = u.second + dy[i];
            if(a < 1 || a > n || b < 1 || b > m
                || G[a][b] == '#' || st[a][b]) continue;
            st[a][b] = 1;
            q.push({a, b});
        }
    }
    return st[n][m];
}
```

P1379 八数码难题

这样的题乍看似乎没有任何头绪，但是我们意识到，如果以初始状态为起点，初始状态经过一次变化就可以达到的状态作为直接连向起点的点，再将这些点通过一次变化就可以达到的点作为连向这些点的点…… 我们就建立起了一张图，那么根据最短路的性质，求初始状态变换到另一个状态的最短步数就是求两个对应的点之间的最短路（当然，从起点到终点和从终点到起点的距离理应是一样的），我们就可以用 BFS 解决了。

那么现在考虑的是如何存储这些状态，可以想到将 3×3 的举证展开，变成一个 1×9 的单行的字符串，那么原本的 x 行 y 列存储的数字应该在字符串的 $x \times 3 + y$ 个字符上。如果反过来，我们就可以将单行的字符串还原成 3×3 的矩阵。

那么如何记录呢？在 C++ 中，我们可以使用 *std :: unordered_map* 容器存储。

事实上，上面这种需要我们将不同的状态抽象成一张图，在图上求解的问题，我们称之为**图论建模**。

```cpp
std::unordered_map<std::string, int> dis;
std::string start;
std::string state = "123804765";
int bfs(void) {
    std::queue<std::string> q;
    q.push(start); dis[start] = 0;
    while(q.size()) {
        auto u = q.front(); q.pop();
        int d = dis[u];
        int k = u.find('0'); int x = k / 3, y = k % 3;
        for(int i = 0; i < 4; i++) {
            int a = x + dx[i], b = y + dy[i];
            if(a < 0 || a >= 3 || b < 0 || b >= 3) continue;
            std::swap(u[k], u[a * 3 + b]);
            if(!dis.count(u)) {
                dis[u] = d + 1;
                q.push(u);
            }
            std::swap(u[k], u[a * 3 + b]);
        }
    }
    return dis.count(state) ? dis[state] : -1;
}
```

习题

BFS 是图论算法的基石，尤其是在它的基础上衍生出了**最短路**算法，是大家日后一定要掌握的技能。在现阶段学好 BFS 能为大家打下良好的基础。

P1443 马的遍历
P1162 填涂颜色
P1332 血色先锋队
P1902 刺杀大使
P10578 [蓝桥杯 2024 国 A] 旋转九宫格