

# 双向搜索

本页面将简要介绍两种双向搜索算法：「双向同时搜索」和「Meet in the middle」。



## 双向同时搜索

### 定义

双向同时搜索的基本思路是从状态图上的起点和终点同时开始进行 [广搜](#) 或 [深搜](#)。

如果发现搜索的两端相遇了，那么可以认为是获得了可行解。

### 过程

双向广搜的步骤：

```
1  将开始结点和目标结点加入队列 q
2  标记开始结点为 1
3  标记目标结点为 2
4  while (队列 q 不为空)
5  {
6      从 q.front() 扩展出新的 s 个结点
7
8      如果 新扩展出的结点已经被其他数字标记过
9          那么 表示搜索的两端碰撞
10         那么 循环结束
11
12     如果 新的 s 个结点是从开始结点扩展来的
13         那么 将这个 s 个结点标记为 1 并且入队 q
14
15     如果 新的 s 个结点是从目标结点扩展来的
16         那么 将这个 s 个结点标记为 2 并且入队 q
17 }
```

### 例题



#### 例题 八数码难题



在  $3 \times 3$  的棋盘上，摆有八个棋子，每个棋子上标有 1 至 8 的某一数字。棋盘中留有一个空格，空格用 0 来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局（为了使题目简单，设目标状态为 123804765），找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

### 解题思路



很好想出暴力 bfs。本题使用暴力 bfs 也不会超时。但是这里把它作为双向同时搜索的例题。我们可以使用两个 bfs，一个从起点状态开始正着搜，一个从终点状态开始反着搜，交替使用两个 bfs，搜索树的大小会大大减小。当其中一个 bfs 搜出另一个 bfs 已经搜出的状态，即可得到答案。

## 参考代码

```
1  #include <iostream>
2  #include <map>
3  #include <queue>
4  #include <string>
5  using namespace std;
6
7  struct State {
8      int A[3][3];
9      State() = default;
10
11     State(string s) {
12         for (int i = 0; i < 3; i++) {
13             for (int j = 0; j < 3; j++) {
14                 A[i][j] = s[i * 3 + j] - '0';
15             }
16         }
17     }
18
19     friend bool operator<(const State &a, const State &b) {
20         for (int i = 0; i < 3; i++) {
21             for (int j = 0; j < 3; j++) {
22                 if (a.A[i][j] != b.A[i][j]) {
23                     return a.A[i][j] < b.A[i][j];
24                 }
25             }
26         }
27         return false;
28     }
29 };
30
31 int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
32
33 void bfs(queue<State> &q, map<State, int> &m1, map<State, int>
34 &m2) {
35     auto u = q.front();
36     q.pop();
37     int xx, yy;
38     for (int i = 0; i < 3; i++) {
39         for (int j = 0; j < 3; j++) {
40             if (u.A[i][j] == 0) {
41                 xx = i;
42                 yy = j;
43             }
44         }
45     }
46     for (int i = 0; i < 4; i++) {
47         int tx = dir[i][0] + xx, ty = dir[i][1] + yy;
48         if (tx >= 0 && tx < 3 && ty >= 0 && ty < 3) {
49             auto v = u;
```

```

50     swap(v.A[xx][yy], v.A[tx][ty]);
51     if (m2.count(v)) {
52         cout << m1[u] + m2[v] << endl;
53         exit(0);
54     }
55     if (!m1.count(v)) {
56         m1[v] = m1[u] + 1;
57         q.push(v);
58     }
59 }
60 }
61 }
62
63 int main() {
64     string I, O;
65     cin >> I;
66     O = "123804765";
67     State in = I, ou = O;
68     queue<State> q1, q2;
69     map<State, int> mp1, mp2;
70     q1.push(in);
71     mp1[in] = 0;
72     q2.push(ou);
73     mp2[ou] = 1;
74     if (I == O) {
75         cout << 0;
76         return 0;
77     }
78     while (1) {
79         bfs(q1, mp1, mp2);
80         bfs(q2, mp2, mp1);
81     }
82     return 0;
}

```

## Meet in the middle

### Warning

本节要介绍的不是 **二分搜索**（二分搜索的另外一个译名为「折半搜索」）。

## 引入

Meet in the middle 算法没有正式译名，常见的翻译为「折半搜索」、「双向搜索」或「中途相遇」。

它适用于输入数据较小，但还没小到能直接使用暴力搜索的情况。

## 过程

Meet in the middle 算法的主要思想是将整个搜索过程分成两半，分别搜索，最后将两半的结果合并。

## 性质

暴力搜索的复杂度往往是指数级的，而改用 meet in the middle 算法后复杂度的指数可以减半，即让复杂度从  $O(a^b)$  降到  $O(a^{b/2})$ 。

## 例题

### 例题 「USACO09NOV」 灯 Lights

有  $n$  盏灯，每盏灯与若干盏灯相连，每盏灯上都有一个开关，如果按下一盏灯上的开关，这盏灯以及与之相连的所有灯的开关状态都会改变。一开始所有灯都是关着的，你需要将所有灯打开，求最小的按开关次数。

$1 \leq n \leq 35$ 。

### 解题思路

如果这道题暴力 DFS 找开关灯的状态，时间复杂度就是  $O(2^n)$ ，显然超时。不过，如果我们用 meet in middle 的话，时间复杂度可以优化至  $O(n2^{n/2})$ 。meet in middle 就是让我们先找一半的状态，也就是找出只使用编号为 1 到 mid 的开关能够到达的状态，再找出只使用另一半开关能到达的状态。如果前半段和后半段开启的灯互补，将这两段合并起来就得到了一种将所有灯打开的方案。具体实现时，可以把前半段的状态以及达到每种状态的最少按开关次数存储在 map 里面，搜索后半段时，每搜出一种方案，就把它与互补的第一段方案合并来更新答案。

## 参考代码

```
1  #include <algorithm>
2  #include <iostream>
3  #include <map>
4  using namespace std;
5
6  int n, m, ans = 0x7fffffff;
7  map<long long, int> f;
8  long long a[40];
9
10 int main() {
11     cin >> n >> m;
12     a[0] = 1;
13     for (int i = 1; i < n; ++i) a[i] = a[i - 1] * 2; // 进行预处理
14
15     for (int i = 1; i <= m; ++i) { // 对输入的边的情况进行处理
16         int u, v;
17         cin >> u >> v;
18         --u;
19         --v;
20         a[u] |= ((long long)1 << v);
21         a[v] |= ((long long)1 << u);
22     }
23
24     for (int i = 0; i < (1 << (n / 2)); ++i) { // 对前一半进行搜索
25         long long t = 0;
26         int cnt = 0;
27         for (int j = 0; j < n / 2; ++j) {
28             if ((i >> j) & 1) {
29                 t ^= a[j];
30                 ++cnt;
31             }
32         }
33         if (!f.count(t))
34             f[t] = cnt;
35         else
36             f[t] = min(f[t], cnt);
37     }
38
39     for (int i = 0; i < (1 << (n - n / 2)); ++i) { // 对后一半进行
40 搜索
41         long long t = 0;
42         int cnt = 0;
43         for (int j = 0; j < (n - n / 2); ++j) {
44             if ((i >> j) & 1) {
45                 t ^= a[n / 2 + j];
46                 ++cnt;
47             }
48         }
49         if (f.count((((long long)1 << n) - 1) ^ t))
```

```
50     ans = min(ans, cnt + f[(((long long)1 << n) - 1) ^ t]);
51 }
52
53 cout << ans;
54
55 return 0;
}
```

## 外部链接

- [What is meet in the middle algorithm w.r.t. competitive programming? - Quora](#)
- [Meet in the Middle Algorithm - YouTube](#)

🔧 本页面最近更新：2025/9/7 21:50:39，[更新历史](#)

✎ 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

👤 本页面贡献者：Ir1d, NachtgeistW, Henry-ZHR, ksyx, Alisahhh, AndrewWayne, Chrogeek, ChungZH, Enter-tainer, FFjet, frank-xjh, hcx1204, hcx2012Git, hsfzLZH1, iamtwz, kenlig, leoleoasd, ouuan, StudyingFather, sundyloveme, Tiphereth-A, Xarfa, ZnPdCo

© 本页面的全部内容在 [CC BY-SA 4.0](#) 和 [SATA](#) 协议之条款下提供，附加条款亦可能应用