搜索—进阶搜索算法

前情提要～

1. 双向广搜、双向深搜
2. 堆优化的 Dijkstra
3. 一颗小小的 A-STAR
4. 不大聪明的 IDDFS（IDS）
5. 可爱的 IDA-STAR

广搜、深搜

这是进阶搜索算法，不说了直接上例题：

以“P1514 引水问题”为例：

▼ 点击查看代码

```cpp
const int N = 510;
const int dx[4] = {-1, 0, 0, 1};
const int dy[4] = {0, -1, 1, 0};
int n, m, a[N][N];
bool vis[N][N][N];
vector<pair<int, int>> nodes;
int main()
{
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            scanf("%d", a[i] + j);

    queue<pair<int, int>> q;
    for (int start = 1; start <= m; ++start)
    {
        q.push({1, start});
        vis[0][1][start] = true;
        vis[start][1][start] = true;
```

```
        while (q.size())
        {
            auto now = q.front();
            int x = now.first, y = now.second;
            q.pop();

            for (int i = 0; i < 4; ++i)
            {
                int tx = x + dx[i];
                int ty = y + dy[i];

                if (tx < 1 || tx > n || ty < 1 || ty > m || vis[start]
[tx][ty])
                    continue;
                if (a[tx][ty] < a[x][y])
                    q.push({tx, ty}), vis[0][tx][ty] = 1, vis[start][tx]
[ty] = 1;
            }
        }

        vector<int> g;
        for (int i = 1; i <= m; ++i)
            if (vis[start][n][i])
                g.push_back(i);

        if (g.size())
            nodes.push_back({g.front(), g.back()});
    }

    int connected = 0;
    for (int i = 1; i <= m; ++i)
        connected += vis[0][n][i];

    if (connected < m)
        printf("0\n%d\n", m - connected);
    else
    {
        sort(nodes.begin(), nodes.end());

        int now = 1, res = 0;
        for (size_t i = 0; i < nodes.size(); ++i, ++res)
        {
            while (nodes[i].first <= now)
```

```
                ++i;
            now = nodes[i - 1].second + 1;
        }

        printf("1\n%d\n", res);
    }

    return 0;
}
```

双向广搜、双向深搜

算法思想

- 在搜索的时候，搜索出来的树太大了
- 从起始状态和目标状态同时开始搜索一定层数
- 把搜索出来的所有状态扔到 hash 表里面，看看有没有重复的
- 能够提升一倍答案的效率，比如原来复杂度是 $O(2^n)$，现在可以变成 $O(2^{n/2})$，相当于复杂度开根号

代码

以“U319719 八城堡问题”为例：

▼ 点击查看代码

```
using namespace std;

const int N = 21;
const int M = (1 << 20) + 1;

int n, m;
int a[N];

int n2, ans;
int g[M];
int f[N][M];

void dfs1(int k, int now)
{
    if (g[now] > m)
        return;

```

```
    if (k > n2)
    {
        int r = now ^ ((1 << n) - 1);
        if (g[now] == m)
            ++f[m][0];
        for (int t = r; t; t = (t - 1) & r)
            ++f[g[now]][t];
        return;
    }

    dfs1(k + 1, now);
    for (int i = 1; i <= n; ++i)
    {
        int t = 1 << (n - i);
        if ((a[k] & t) && (now & t) == 0)
            dfs1(k + 1, now | t);
    }
}

void dfs2(int k, int now)
{
    if (g[now] > m)
        return;
    if (k > n)
    {
        ans += f[m - g[now]][now];
        return;
    }

    dfs2(k + 1, now);
    for (int i = 1; i <= n; ++i)
    {
        int t = 1 << (n - i);
        if ((a[k] & t) && (now & t) == 0)
            dfs2(k + 1, now | t);
    }
}

int main()
{
    for (int i = 1; i < M; ++i)
        g[i] = g[i & (i - 1)] + 1;

    scanf("%d%d", &n, &m);
```

```
    while (n && m)
    {
        memset(f, 0, sizeof f);
        ans = 0;

        char line[N];
        for (int i = 1; i <= n; ++i)
        {
            scanf("%s", line);
            for (int j = 1; j <= n; ++j)
                a[i] = a[i] << 1 | (line[j - 1] == 'H');
        }

        n2 = n >> 1;
        dfs1(1, 0);
        dfs2(n2 + 1, 0);

        printf("%d\n", ans);
        scanf("%d%d", &n, &m);
    }
    return 0;
}
```

堆优化的 Dijkstra

算法思想

考虑当前走过的距离，不考虑剩下的距离，当前走的少的优先考虑

优先队列里不允许出现相同的元素，但是同一个元素可以入队和出队多次，当第二及更多次入队是，必然是遇到了更加优化的路径（到起点的距离更近）

代码

以“P3371 P4779 单源最短路径”为例：

▼ 点击查看代码

```
typedef pair<int, int> PII;

const int INF = 2147483647;
const int N = 1e5 + 10;
const int M = 5e5 + 10;

```

```cpp

int n, m;

int h[N], e[M], w[M], ne[M], idx;

void add(int u, int v, int d)
{
    e[idx] = v;
    w[idx] = d;
    ne[idx] = h[u];
    h[u] = idx++;
}

int dis[N];
bool st[N];

void dijkstra(int s)
{
    memset(dis, 0x3f, sizeof dis);
    dis[s] = 0;

    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, s});

    while (heap.size())
    {
        int v = heap.top().second, d = heap.top().first;
        heap.pop();

        if (st[v])
            continue;
        st[v] = true;

        for (int i = h[v] ; i != -1 ; i = ne[i])
        {
            int j = e[i];
            if (dis[j] > d + w[i])
            {
                dis[j] = d + w[i];
                heap.push({dis[j], j});
            }
        }
    }
}
```

```cpp

unordered_map<int, int> dist[N];

int main()
{
    memset(h, -1, sizeof(h));

    int s;
    scanf("%d %d %d", &n, &m, &s);

    int u, v, w;
    for (int i = 1 ; i <= m ; ++i)
    {
        scanf("%d %d %d", &u, &v, &w);
        if (dist[u].count(v))
            dist[u][v] = min(dist[u][v], w);
        else
            dist[u][v] = w;
    }

    for (int i = 1 ; i <= n ; ++i)
        for (PII j : dist[i])
            add(i, j.first, j.second);

    dijkstra(s);

    for (int i = 1 ; i <= n ; ++i)
    {
        if (dis[i] == 0x3f3f3f3f)
            dis[i] = INF;
        printf("%d ", dis[i]);
    }
    return 0;
}
```

一颗小小的 A-STAR

基础知识

- 估价函数 $f(i) = g(i) + h(i)$，每次要选取 $f(i)$ 最小的更新
- $g(i)$：从起始状态到当前状态 i 的代价
- $h(i)$：从当前状态 i 到目标状态的**估计**代价

基础思想

好东西：https://zhuanlan.zhihu.com/p/54510444

重点在设计估价函数，估价函数 $h(i)$ 若选取不当，则可能找不到解，或找到的解也不是最优解。

- 定义 $h^*(i)$ 为从当前状态 n 到目标状态的实际代价
- 必须满足 $h(i) \leq h^*(i)$，否则嘿嘿嘿
- 估计总是过于乐观的

常见的估价函数

对于网格形式的图，有以下这些启发函数可以使用：

1. 如果图形中只允许朝上下左右四个方向移动，则可以使用曼哈顿距离

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```

2. 如果图形中允许朝八个方向移动，则可以使用对角距离

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
    // 这里的D2指的是两个斜着相邻节点之间的移动代价
```

3. 如果图形中允许朝任何方向移动，则可以使用欧几里得距离

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

4. 对于类似八数码问题的，可以使用各数码使用距离目标位置的曼哈顿距离之和

```
int f(string state)
{
    int res = 0;
```

```
    for (int i = 0; i < 9; ++i)
    {
        if (state[i] != '0')
        {
            int t = tar[state[i] - '1'];
            res += abs(i / 3 - t / 3) + abs(i % 3 - t % 3);
        }
    }
    return res;
}
```

5. 对于数列（二维也可以）问题的，可以使用有多少个元素位置不正确，或其前驱、后继不正确

```
int f(string state)
{
    int res = 0;
    for (int i = 0; i < 9; ++i)
        res += state[i] != gend[i];
    return res;
}
```

```
int f()
{
    int res = 0;
    for (int i = 0; i + 1 < n; ++i)
        if (q[i + 1] != q[i] + 1)
            ++res;
    return (res + 2) / 3;
}
```

设计要求

- 不要太过分就好，一般来说没问题的
- 脑洞要大

代码

以“P1379 八数码问题”为例：

▼ 点击查看代码

```cpp
typedef pair<int, string> PIS;

const int dx[4] = {-1, 0, 0, 1};
const int dy[4] = {0, -1, 1, 0};

const char op[4] = {'u', 'l', 'r', 'd'};

string gend = "123804765";
int tar[9] = {0, 1, 3, 5, 8, 7, 6, 3};

int f(string state)
{
    int res = 0;
    for (int i = 0; i < 9; ++i)
    {
        if (state[i] != '0')
        {
            int t = tar[state[i] - '1'];
            res += abs(i / 3 - t / 3) + abs(i % 3 - t % 3);
        }
    }
    return res;
}

string a_star(string start, string end = gend)
{
    unordered_map<string, int> dist;
    unordered_map<string, pair<char, string>> prev;
    priority_queue<PIS, vector<PIS>, greater<PIS>> heap;

    dist[start] = 0;
    heap.push({f(start), start});

    while (heap.size())
    {
        auto t = heap.top();
        heap.pop();

        string state = t.second;
        if (state == end)
            break;

        int x, y;

```

```
        for (int i = 0; i < 9; ++i)
        {
            if (state[i] == '0')
            {
                x = i / 3, y = i % 3;
                break;
            }
        }

        string source = state;
        for (int i = 0; i < 4; ++i)
        {
            int a = x + dx[i];
            int b = y + dy[i];
            if (a < 0 || a > 2 || b < 0 || b > 2)
                continue;
            state = source;
            swap(state[x * 3 + y], state[a * 3 + b]);
            if (!dist.count(state) || dist[state] > dist[source] + 1)
            {
                dist[state] = dist[source] + 1;
                prev[state] = {op[i], source};
                heap.push({dist[state] + f(state), state});
            }
        }
    }

    string res;
    while (end != start)
    {
        res = prev[end].first + res;
        end = prev[end].second;
    }
    return res;
}

int main()
{
    string start;
    cin >> start;
    // 保证可以达到目标
    // int cnt = 0;
    // for (int i = 0 ; i < 9 ; ++i)
    //  if (start[i] != '0')
```

```
    //      for (int j = i + 1 ; j < 9 ; ++j)
    //          if (start[j] != '0' && start[i] > start[j])
    //              ++cnt;
    // if (cnt & 1 == 0)
    //  cout << "-1" << endl;
    // else
    cout << a_star(start).size() << endl;
    return 0;
}
```

不大聪明的 IDDFS（IDS）

算法思想

迭代加深是一种**每次限制搜索深度的**深度优先搜索，目的是寻找最优解。

- 给出一个限制 `limit`，规定：当 **`搜索层数 > limit`** 时直接剪枝
- 在最外层 **`循环枚举 limit`**，如果无解就继续

特点

缺点：重复计算

与BFS的区别：BFS 的基础是队列，空间复杂度大，**当状态比较多或者单个状态比较大时**，迭代加深就类似于用 DFS 方式实现的 BFS，空间复杂度相对较小。

在大多数的题目中，广度优先搜索还是比较方便的，而且**容易判重**。当发现广度优先搜索在空间上不够优秀，而且要找最优解的问题时，就应该考虑迭代加深。

代码

以“LOJ10021 Addition Chains（加成序列问题）”为例：

▼ 点击查看代码

```
const int N = 110;

int read()
{
    int num = 0, flag = 1;
    char ch = getchar();
    for (; !isdigit(ch); ch = getchar())
        if (ch == '-')
```

```
            flag = -1;
    for (; isdigit(ch); ch = getchar())
        num = (num << 3) + (num << 1) + ch - '0';
    return num * flag;
}

int n;
int path[N];

bool iddfs(int k, int limit)
{
    if (k > limit)
        return false;
    if (path[k - 1] == n)
        return true;

    bool st[N] = {0};
    for (int i = k - 1; i >= 0; --i)
    {
        for (int j = i; j >= 0; --j)
        {
            int s = path[i] + path[j];
            if (s > n || s <= path[k - 1] || st[s])
                continue;
            st[s] = true;
            path[k] = s;
            if (iddfs(k + 1, limit))
                return true;
        }
    }
    return false;
}

int main()
{
    path[0] = 1;
    while (n = read())
    {
        int depth = 1;
        while (!iddfs(1, depth))
            ++depth;
        for (int i = 0; i < depth; ++i)
            printf("%d ", path[i]);
        putchar('\n');
```

```
    }
    return 0;
}
```

可爱的 IDA-STAR

算法思想

IDA-STAR 为采用了迭代加深（IDDFS）算法的 A-STAR 算法。

所以，在一般的问题中是这样使用 IDA-star 算法的：

- （类似 IDDFS）循环枚举 limit，当 $h(i)+g(i)>$ limit 时，就停止继续往下搜索
- 写成代码就是：`if depth + h() > limit then return;`
- 没了

估价函数

往上门口（见 A-STAR）

特点

- 省空间：各种游戏求最少步数，普通搜索会爆炸
- 省时间：不需要判重，不需要排序，利于深度剪枝
- 费时间（？）：每次深度变大都要再次从头搜索，有时可能**双向广搜**会更快

代码

以“P2324 骑士精神”为例：

▼ 点击查看代码

```
const int dx[8] = {-2, -2, -1, -1, 1, 1, 2, 2};
const int dy[8] = {-1, 1, -2, 2, -2, 2, -1, 1};
const int gcount[2][5] = {{0, 1, 2, 4, 5},
                          {5, 4, 2, 1, 0}};
string mp[5];

int f()
{
    int res = 0;
    for (int i = 0; i < 5; ++i)
    {
```

```
        for (int j = 0; j < gcount[0][i]; ++j)
            if (mp[i][j] != '0')
                ++res;
        for (int j = 0; j < gcount[1][i]; ++j)
            if (mp[i][5 - j - 1] != '1')
                ++res;
    }
    return res;
}

bool ida_star(int now, int limit)
{
    if (now + f() > limit)
        return false;
    if (!f())
        return true;

    int a, b;
    for (int i = 0; i < 5; ++i)
        for (int j = 0; j < 5; ++j)
            if (mp[i][j] == '*')
                a = i, b = j;

    for (int i = 0; i < 8; ++i)
    {
        int x = a + dx[i];
        int y = b + dy[i];
        if (x < 0 || x > 4 || y < 0 || y > 4)
            continue;
        swap(mp[a][b], mp[x][y]);
        if (ida_star(now + 1, limit))
            return true;
        swap(mp[a][b], mp[x][y]);
    }
    return false;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
```

```
57
58      while (T--)
59      {
60          for (int i = 0; i < 5; ++i)
61              cin >> mp[i];
62
63          int limit = 0;
64          while (limit <= 15 && !ida_star(0, limit))
65              ++limit;
66
67          if (limit > 15)
68              limit = -1;
69          printf("%d\n", limit);
70      }
71
72      return 0;
    }
```

练习题

见：https://www.luogu.com.cn/training/401467

Reference

[1] https://github.com/huzecong/oi-slides

[2] https://zhuanlan.zhihu.com/p/54510444

[3] https://oi-wiki.org/

[4] https://www.acwing.com/

合集：学习笔记

标签：算法 , 学习笔记