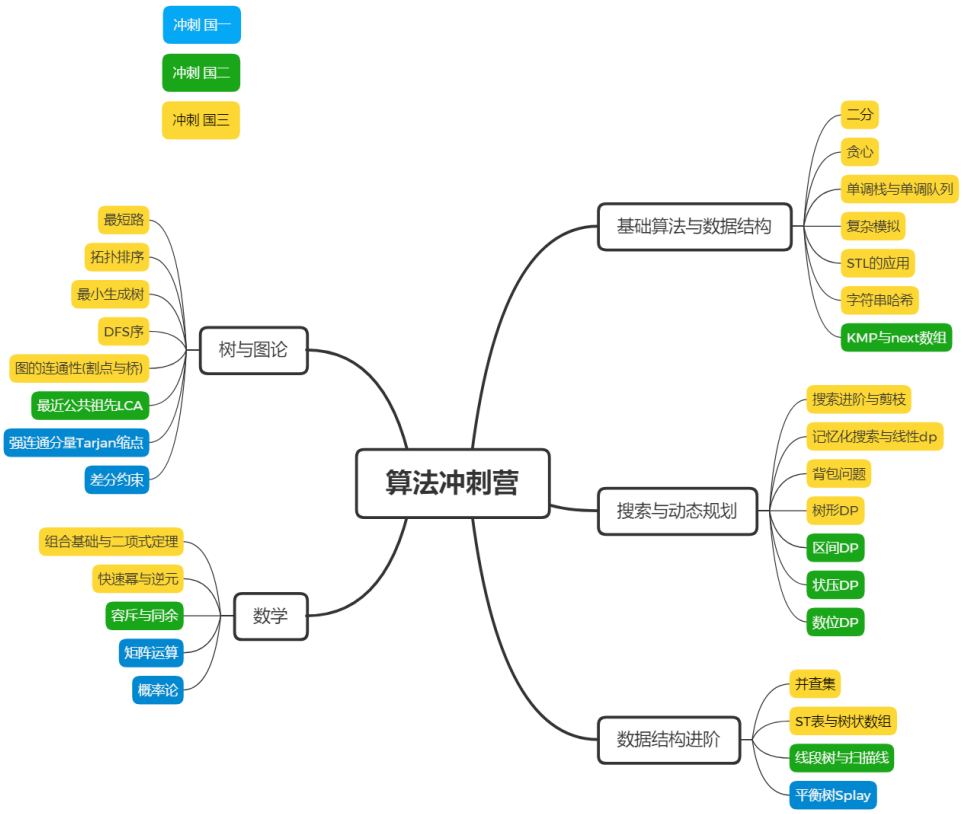
蓝桥杯30天算法冲刺集训



Day-6 搜索进阶与剪枝

DFS vs BFS

对比我们可以发现：广度优先搜索和深度优先搜索（尤其是非递归的栈实现）的基本过程很相似，都包含如下两个过程：

1. 判断边界：判断当前状态是否是目标状态，并进行相应处理
2. 扩展新状态：由当前状态（节点）出发，扩展出新的状态（节点）

它们的区别就是一个用队列实现和一个用栈实现，一个按层横向遍历，一个按列纵深遍历。数据结构的不同也导致了它们对待 **扩展出的新状态** 的处理策略不同。深度优先搜索会将所有的新状态压入栈中，采取 **先扩展出来的最后处理** 的策略。而广度优先搜索会将所有的新状态压入队列中，采取 **先扩展出来的先处理** 的策略。这也正是栈 **先入后出** 和队列 **先入先出** 的体现。

需要注意的是：DFS（深度优先搜索）是一种相对更常见的算法，大部分的题目都可以用 DFS 解决（而BFS广搜就不行了），但是大部分情况下，这都是骗分算法，很少会有爆搜为正解的题目。因为 DFS 的时间复杂度特别高。

但是，如果想掌握更高级的算法，DFS必须要先熟练掌握！

搜索的解题步骤

1. 读清题目，理清状态是什么，状态包含哪些要素，初始状态是什么，目标状态是什么；
2. 建立搜索树，可以想象一下，或者在草稿纸上画，以帮助自己理解整个搜索树的构成，以及搜索的过程；
3. 明确从每一个状态出发可以转移到哪些新的状态（通过题目中允许的操作进行转移），转移是如何影响状态的，哪些要素发生了变化？
4. 需要进行回溯吗？状态的保存需要用全局变量吗？适合用广搜还是深搜？
5. 可以进行哪些剪枝，以减少不必要的搜索？
6. 按照代码框架，实现代码

搜索的一些优化

剪枝

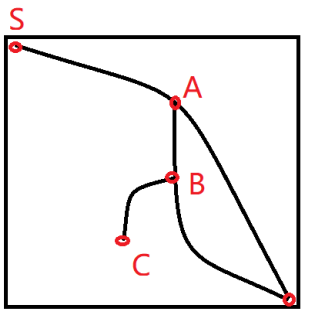
剪枝，顾名思义，就是通过一些判断，砍掉搜索树上不必要的子树。有时候，我们会发现某个结点对应的子树的状态都不是我们要的结果，那么我们其实没必要对这个分支进行搜索，砍掉这个子树，就是剪枝。

最常用的剪枝有三种，可行性剪枝、重复性剪枝、最优性剪枝。

可行性剪枝：在搜索过程中，一旦发现如果某些状态无论如何都不能找到最终的解，就可以将其“剪枝”了，比如越界操作、非法操作。一般通过条件判断来实现，如果新的状态节点是非法的，则不扩展该节点

重复性剪枝：对于某一些特定的搜索方式，一个方案可能会被搜索很多次，这样是没必要的。在实现上，一般通过一个记忆数组来记录搜索到目前为止哪些状态已经被搜过了，然后在搜索过程中，如果新的状态已经被搜过了，则不再扩展该状态节点。

最优性剪枝：对于求最优解的一类问题，通常可以用最优性剪枝，比如在求解迷宫最短路的时候，如果发现当前的步数已经超过了当前最优解，那从当前状态开始的搜索都是多余的，因为这样搜索下去永远都搜不到更优的解。通过这样的剪枝，可以省去大量冗余的计算，避免超时。在实现上，一般通过一个记忆数组来记录搜索到目前为止的最优解，然后在搜索过程中，如果新的状态已经不可能是最优解了，那再往下搜索肯定搜不到最优解，于是不再扩展该状态节点。



搜索从 S 到 E，假设先搜到了 S->A->E 这条路径，其解为 30。

然后在 A 点回溯并搜完了 A->B->E 这条路径，此时的 S->E 的解为 20，于是更新最优解为 20。

回溯到 B 点，然后搜到 C 点的时候，此时路径 S->A->B->C 的代价已经达到 20，那么再往下搜下去，结果肯定会大于 20，那肯定不会是最优解了，于是没必要再搜下去了，返回（于是剪枝了）

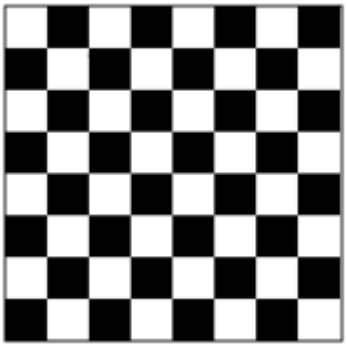
最优性剪枝的概念

其他的剪枝策略：

奇偶性剪枝

我们先来看一道题目：有一个 $n\times m$ 大小的迷宫。其中字符 S 表示起点，字符 D 表示出口，字符 X 表示墙壁，字符 . 表示平地。你需要从 S 出发走到 D，每次只能向上下左右相邻的位置移动，并且不能走出地图，也不能走进墙壁。每次移动消耗 1 时间，走过路都会塌陷，因此不能走回头路或者原地不动。现在已知出口的大门会在T时间打开，判断在 0 时间从起点出发能否逃离迷宫。数据范围$n,m\leq 10,T\leq 50$。

我们只需要用 DFS 来搜索每条路线，并且只需搜到 T 时间就可以了（这是一个可行性剪枝）。但是仅仅这样也无法通过本题，还需考虑更多的剪枝。



如上图所示，将 $n \times m$ 的网格染成黑白两色。我们记每个格子的行数和列数之和 x，如果 x 为偶数，那么格子就是白色，反之奇数时为黑色。容易发现相邻的两个格子的颜色肯定不一样，也就是说每走一步颜色都会不一样。更普遍的结论是：走奇数步会改变颜色，走偶数步颜色不变。

那么如果起点和终点的颜色一样，而 T 是奇数的话，就不可能逃离迷宫。同理，如果起点和终点的颜色不一样，而T是偶数的话，也不能逃离迷宫。遇到这两种情况时，就不用进行 DFS 了，直接输出 "NO"。

这样的剪枝就是奇偶性剪枝，本质上也属于可行性剪枝。

蓝桥杯真题

> *[!NOTE]*
>
> *前两题B站都有录播，也可以再参考B站的录播讲解。*

填字母游戏（蓝桥杯C/C++2017A组国赛）

这个题目需要注意的是怎么用一个DFS就能判断出来在给定的字符串上面操作，到底是返回 `0`， `-1` 还是 `1` 的结果呢?

首先这种博弈类型的问题我们都是进行输赢的状态的转移的，也就是说如果状态下我找到一个 `*` 填入 `L` 或者 `0` 之后下一步再走DFS，回溯回来知道是一定输的，那么我们就是能赢的了。如果是平的，那么我们也是能平的，否则就是输的。

题目要让我们找最好的结果，那么我们找到能赢就返回，找到能平，就记录一下可以平，最后如果找不到能赢，就看是不是可以平，否则的话就是要输的。

我们还需要对状态进行一下记录，殊途同归，不论我们前面是怎么选取的 L 或者 O ，只要到达这个状态，那么输赢就不会变换，所以需要记忆化搜索一下。

```
mk = {}
bk = ['L', 'O']

def dfs(s):
    if s in mk:
        return mk[s]
    if "*OL" in s or "L*L" in s or "LO*" in s:  # check win
        mk[s] = 1
        return 1
    if '*' not in s and "LOL" not in s:  # check draw
        mk[s] = 0
        return 0
    draw = 0
    for i, c in enumerate(s):
        if c == '*':
            for ch in bk:
                s = s[:i] + ch + s[i+1:]
                now = dfs(s)
                s = s[:i] + '*' + s[i+1:]
                if now == -1:
                    mk[s] = 1
                    return 1
                elif now == 0:
                    draw = 1
    if draw:
        mk[s] = 0
        return 0
    mk[s] = -1
    return -1

def solve():
    s = input().strip()
    print(dfs(s))

def main():
    T = int(input())
```

```
    for _ in range(T):
        solve()

if __name__ == "__main__":
    main()
```

大胖子走迷宫（蓝桥杯C/C++2019A组国赛）

这题主要不同于一般题目的地方在于，小明的身材还会变化，也就是说最开始的时候呢是5×5的身材，后面会慢慢的变成3×3以至于变成1×1的普通的身材。

我们可以用BFS进行遍历，每次都通过一个四元组遍历也就是（$x, y, step, size$）这个四元组，分别代表当前小明中心点的位置，小明走过的步数和小明当前的身材大小。

每次我们都有两种转移方式：

1. 小明以当前的身材向上下左右能走的地方去移动
2. 小明待在原地不动

然后我们需要注意到时间后，小明身材的变化，这里需要模拟一下。

最后只要能走到终点返回当前的$step$就可以了。

```
import os
import sys

# 请在此输入您的代码
from collections import deque

n,k = map(int,input().split())
s = [list(input()) for i in range(n)]
q = deque([(2,2,2,0)])
vir = [[0 for i in range(n)] for j in range(n)]
vir[2][2] = 1
dir = [[-1,0],[1,0],[0,-1],[0,1]]

def f(t):  # 检验时间返回状态
    if t < k:
        return 2
```

```
    if t <2*k:
        return 1
    else:
        return 0

def check(x,y,z): # 检查标记
    for i in range(x-z,x+z+1):
        for j in range(y-z,y+z+1):
            if s[i][j] == '*':
                return False
    return True

def bfs():
    while q:
        x,y,z,cnt = q.popleft()
        if x == n-3 and y == n-3:
            return cnt
        if z != 0:
            q.append((x,y,f(cnt + 1),cnt+1))
        for d in range(4):
            nx = x + dir[d][0]
            ny = y + dir[d][1]
            if 0 <= nx-z and nx+z < n and 0<= ny-z and ny+z < n
and vir[nx][ny] == 0:
                if check(nx,ny,z) == True:
                    q.append((nx,ny,f(cnt+1),cnt+1))
                    vir[nx][ny] = 1
print(bfs())
```

最优旅行（蓝桥杯C/C++2019A组国赛）

这是一道结果填空题。首先题意是需要从北京出发，每个城市浏览一次（北京除外，可以多次停靠），最终回到北京。很明显是一道dfs+剪枝的题目，需要注意的是以下几点：

- 需要把时间换算成分钟，因为最后的结果是按照分钟计算的，而且需要注意24小时
- 小明决定从一个城市去往另一个城市时，他只会选择有直接高铁连接的城市，不会在中途换乘转车。
- 可以用字符串到int的映射来简化题目。

- 在计算运行时间cost和最终dfs中消耗时间的时候，需要注意是隔天还是当天的车。

```
47373
```

```
import sys

class Node:
    def __init__(self, des, startt, endt, time):
        self.e = des
        self.starttime = startt
        self.endtime = endt
        self.cost = time

G = [[] for _ in range(25)]  # 存放所有班次信息
m = {
    "北京": 1, "上海": 2, "广州": 3, "长沙": 4, "西安": 5,
    "杭州": 6, "济南": 7, "成都": 8, "南京": 9, "昆明": 10,
    "郑州": 11, "天津": 12, "太原": 13, "武汉": 14, "重庆": 15,
    "南昌": 16, "长春": 17, "沈阳": 18, "贵阳": 19, "福州": 20
}
mintime = sys.maxsize  # 将最终的结果设为最大值
totaltime = 0
vis = [0] * 25  # 标志去过哪些城市了(每个城市只能去一次，除了北京)

def dfs(s, endt, len):
    global mintime, totaltime, vis
    if s == 1 and len > 0:
        flag = all(vis[i] == 1 for i in range(1, 21))
        vis[1] = 0
        if flag:
            mintime = min(mintime, totaltime)
        return
    for r in G[s]:
        if vis[r.e] == 0:
            vis[r.e] = 1
            temp = totaltime
            totaltime += r.cost
            if s != 1 and r.starttime > endt:
                totaltime += r.starttime - endt
```

```
            if s != 1 and r.starttime < endt:
                totaltime += r.starttime - endt + 1440
            if s == 1:
                if r.starttime > 720:
                    totaltime += r.starttime - 720
                else:
                    totaltime -= r.starttime - 720 + 1440
            if totaltime > mintime:
                totaltime = temp
                continue
            dfs(r.e, r.endtime, len + 1)
            vis[r.e] = 0
            totaltime = temp

def main():
    global mintime, totaltime, vis
    INIT()
    for _ in range(132):
        line = input().split()
        src = line[1]
        des = line[2]
        s = line[3]
        t = line[4]

        if src not in m or des not in m:
            print("Invalid city name:", src, "or", des)
            continue

        a = int(s[:2]) * 60 + int(s[3:])  # 换算出发时间为分钟
        b = int(t[:2]) * 60 + int(t[3:])  # 换算到达时间为分钟

        cost = b - a if a < b else b - a + 1440

        G[m[src]].append(Node(m[des], a, b, cost))

    dfs(1, 0, 0)
    mintime += 1440 * 19
    print(mintime)

def INIT():
```

```
    pass  # 可以在此处添加必要的初始化代码

if __name__ == "__main__":
    main()
```

习题

小蓝与迷宫

这个题目实际上还是贪心+搜索。

首先我们明白去拦截我们的人怎么样做才能最优。也就是怎么做才能在一定能拦截我们的情况下，拦截我们。显然我们可以走的路径不是唯一的，那该怎么办呢。实际上直接都在终点等候即可。也就是说不论我们自身如何去走，拦截我们的人只需要在走最短路先到终点等候即可。

那这样的话做两遍BFS直接能够解决。一遍BFS从终点出发，找到所有拦截的人到终点的最小步数。默认这个步数为一个极大值，如果步数最终还是一个极大值，那么说明他无法到达终点，否则就记录下这个最小步数。

然后再一个BFS从当前人物出发，走到最终的终点，记录下来这个最小步数。然后都可以在这个最小步数之前到达终点的拦截的人都可以要到小蓝的签名。

Python代码：

```
from collections import deque

class Node:
    def __init__(self, x, y, cur):
        self.x = x
        self.y = y
        self.cur = cur

def bfs1(x, y):
    global vis, ex, ey, ot
    vis[x][y] = 1
    q = deque()
```

```
    q.append(Node(x, y, 0))
    while q:
        p = q.popleft()
        if p.x == ex and p.y == ey:
            ot = p.cur
            break
        for i in range(4):
            xx = p.x + dx[i]
            yy = p.y + dy[i]
            if 1 <= xx <= n and 1 <= yy <= m and vis[xx][yy] ==
0 and a[xx][yy] != 'T':
                vis[xx][yy] = 1
                q.append(Node(xx, yy, p.cur + 1))

def bfs2(x, y):
    global vis
    vis[x][y] = 1
    q = deque()
    q.append(Node(x, y, 0))
    while q:
        p = q.popleft()
        d[p.x][p.y] = p.cur
        for i in range(4):
            xx = p.x + dx[i]
            yy = p.y + dy[i]
            if 1 <= xx <= n and 1 <= yy <= m and vis[xx][yy] ==
0 and a[xx][yy] != 'T':
                vis[xx][yy] = 1
                q.append(Node(xx, yy, p.cur + 1))

n, m = map(int, input().split())
N = 1010
a = [[''] * N for _ in range(N)]
vis = [[0] * N for _ in range(N)]
d = [[float('inf')] * N for _ in range(N)]
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]
sx, sy, ex, ey, ot, ans = 0, 0, 0, 0, 0, 0

for i in range(1, n + 1):
```

```
    row = input()
    for j in range(1, m + 1):
        a[i][j] = row[j - 1]
        if a[i][j] == 'S':
            sx, sy = i, j
        elif a[i][j] == 'E':
            ex, ey = i, j

bfs1(sx, sy)
vis = [[0] * N for _ in range(N)]  # Reset vis array
bfs2(ex, ey)

for i in range(1, n + 1):
    for j in range(1, m + 1):
        if '0' <= a[i][j] <= '9':
            if d[i][j] <= ot:
                ans += int(a[i][j])

print(ans)
```

小红与字符串矩阵

题解：这题的核心是搜索+剪枝

这题是一个迷宫寻路算法，只不过需要按照它给定的路径行走，所以立马想到使用dfs。

和朴素dfs不同的地方是，这题不需要check一下是否越界（java和python好像需要判断，不然越界会错误），只需要看一下下一步是否是我们要走的字符串中对应步数的字符。now==6说明找到了整个tencent字符串，直接返回。需要注意的是，如果你的now是从0开始的，我们需要匹配的字符串要从第二个字母e开始 代码如下：

Python代码

```
ten = "encent"
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

maxn = 1010
mp = [['' for _ in range(maxn)] for _ in range(maxn)]
ans = 0
```

```
def dfs(x, y, now):
    global ans
    if now == 6:
        ans += 1
        return

    for i in range(4):
        xx = x + dx[i]
        yy = y + dy[i]

        if 0 <= xx < n and 0 <= yy < m and ten[now] == mp[xx]
[yy]:
            dfs(xx, yy, now + 1)

if __name__ == "__main__":
    n, m = map(int, input().split())
    for i in range(n):
        mp[i] = list(input())

    for i in range(n):
        for j in range(m):
            if mp[i][j] == 't':
                dfs(i, j, 0)

    print(ans)
```