搜索 & NPC

IIIS, 许庭强

2024.8.10

搜索

- 搜索问题的解，其实就是在问题对应的状态空间中进行映射与遍历。使用递归，循环，数据结构等对状态空间的单调性，对称性，有解性进行排列归纳，加以搜索，以快速找到问题的答案。

深度优先搜索

- 深度优先搜索（dfs）属于图算法的一种，过程为对每一个可能的分支路径深入到不能再深入为止，且每个节点最多访问一次。
- 具体来说，从图的顶点 v 出发：
- 1. 访问顶点 v
- 2. 依次遍历 v 的所有未经过的临界点
- 3. 退回前一个点
- 把图中的每一个节点看作一种状态，则可以用来搜索一个问题的解。

迷宫问题

- 有一个 n*m 的网格图，有些格子是障碍。
- 你要从起点走到终点，每次可以走到周围四个格子之一，但不能经过任何障碍格子。找到一条能走到终点的路径。

迷宫问题

- 有一个 n*m 的网格图，有些格子是障碍。
- 你要从起点走到终点，每次可以走到周围四个格子之一，但不能经过任何障碍格子。找到一条能走到终点的路径。

- 可以用 dfs 解决。

广度优先搜索

- 与深度优先搜索不同，广度优先搜索能够找到从起点出发到所有点的最短路径长度。
- 如果把深度优先搜索看作栈，那么广度优先搜索可以被看作一个队列。具体来说，广度优先搜索维护一个队列，过程如下：
- 1. 将起点放入队列。
- 2. 不断重复：取出队列头，将所有该节点未被放入过队列的邻居放入队列中。
- 如果在迷宫问题中要找到最短路径，可以用广度优先搜索解决。

剪枝

- 剪枝指的是在深度优先搜索中去掉一些不符合题目要求的或是浪费时间而没有作用的答案，从而使得深度优先搜索能够更快得到正确答案。
- 因为在搜索树中去掉答案形似剪掉树的枝叶，所以这一方法被称为剪枝。

数独

- 给定一个数独的初始盘面，解出该数独的一个解。
- 如果直接枚举所有每个格子填什么，需要 9^81 的时间。
- 可以一行一行 dfs，依次枚举每个格子填什么，枚举时只用枚举与所有所在行、列、宫不同的数字。剪枝剪去了大部分无用的状态。

P1120 小木棍

- 乔治有一些同样长的小木棍，他把这些木棍随意砍成几段，直到每段的长都不超过 50。
- 现在，他想把小木棍拼接成原来的样子，但是却忘记了自己开始时有多少根木棍和它们的长度。
- 给出每段小木棍的长度，编程帮他找出原始木棍的最小可能长度。
- 1<=n<=65，1<=ai<=50

P1120 小木棍

- 枚举最终答案，用 dfs 尝试把所有木棍拼成该长度。
- 考虑一个木棍一个木棍拼，设 dfs(stick, len) 表示在拼第 stick 根木棍，以拼的长度为 len。

P1120 小木棍

- **可行性剪枝**
- 枚举木棍长度时，从数列最大值枚举木棍长度到数列的和的一半，如果超过长度直接输出数列
- 数组从大到小排序，优先选长的，以更快趋近边界（木棍顺序的等效性）
- 搜索时的木棍长度小于等于上一根木棍的长度（长度递减）（框架改为dfs(stick,len,last)）

P1120 小木棍

- **等效性剪枝**
- 长度相同的木棍不重复搜索（相同长度木棍的等效性）
- 如果搜索整根木棍时（len=0）失败，就不再搜索（每根要拼凑木棍长度相同，等效）
- **最优性剪枝**
- 当前长度与最短木棍的长度和都大于枚举的木棍长度，剪枝

P1120 小木棍

```
void dfs(int stick, int len, int last) {
  // 正在拼第 stick 根小木棍，已拼出的长度为 len，上一次选的木棍下标为 last
  if (stick * len == sum)
    printf("%d", sz), exit(0);
  if (len == sz) dfs(stick + 1, 0, 0); // 下一根
  if (sz - len < a[n]) return;
  for (int i = last + 1; i <= n; i++) {
    if (!vis[i] && len + a[i] <= sz) { // 可以使用
      vis[i] = 1;
      dfs(stick, len + a[i], i);
      vis[i] = 0;
      if (len + a[i] == sz || len == 0) return; // 不同木棍的等效性
      while (a[i + 1] == a[i]) i++; // 长度相同木棍的等效性
    }
  }
}
```

折半搜索

- 折半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搜索。
- 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。

灯

- 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<=n<=35

灯

- 显然每个开关只会被按 0 次或 1 次。如果暴力枚举每个开关是否按，则需要 2^n 的时间。
- 考虑分别枚举前一半/后一半的灯是否按得到的所有灯的状态。我们想要找到两个灯的状态互补的前一半/后一半方案。
- 具体实现时，可以把前一半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，搜索后一半时，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。
- 时间复杂度 n*2^(n/2)

为什么要搜索?

- 一个输入规模为 n 的问题，搜索要花费 2^n 的时间才能找到答案，但找到答案后检查是否正确却只需要 n^c 的时间。
- 是否一定需要这么长的时间?
- P：在多项式时间复杂度内能够解决的问题。
- NP：在多项式时间复杂度内能够判定一个解是否满足条件的问题。
- P vs NP

NP-Completeness

- 如果对于一个问题 A，任意 NP 里的问题都能归约到 A，那么称 A 为 NP-complete 的，简称 NPC。
- Karp Reduction: 将一个 B 问题在多项式时间内转化为一个 A 问题。
- Cook Reduction: 调用若干次 A 问题解决 B 问题。
- 如果我们能够证明一个 NPC 问题能在 O(poly(n)) 时间复杂度解决，我们就能够证明 P=NP。

3-SAT

- SAT 是 satisfiability 的缩写。SAT 问题就是问某一个布尔表达式是不是“可满足”的问题，即是否存在一组赋值使得布尔表达式为真。
- 3-SAT 即每一个布尔表达式由 3 个元素构成。
- 3-SAT 是 NP-complete 的。

最大独立集

- 给定一张无向图，问是否存在一个点集大小 >=k 且没有两个点集内的点相邻。
- 归约：对每个变量建立两个点，对每个 clause 建立三个点。

- 求最大独立集是 NP-Hard 的。
- NP-Hard：能从所有 NP 问题归约到的问题。

最大团

- 与最大独立集等价。

哈密顿回路

- 给定一张有向图，问是否存在一个回路经过所有点恰好一次。
- 归约：对每个变量建立一条双向链，两种走的方式表示 =0/1。对每个 clause 建立一个点，限制三条链中至少一条边的走向正确。

哈密顿链

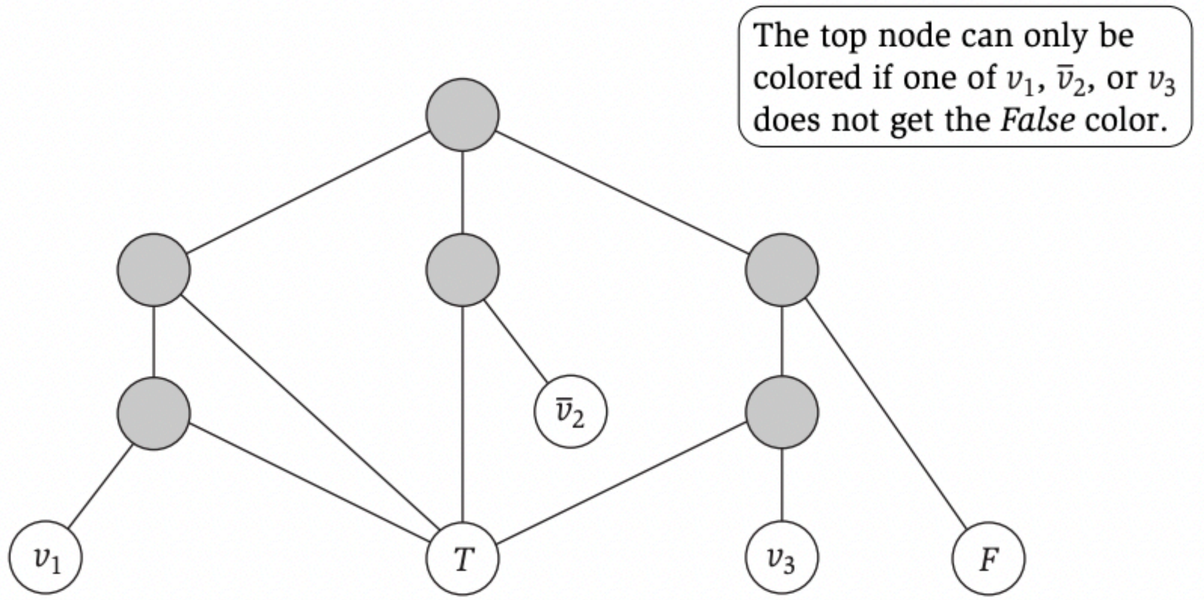
- 从哈密顿回路归约。

TSP

- 著名的 TSP (Travelling Salesman Problem) 问题是指一个邮递员要从自己的家出发，经过 n 座城市，最后回到家。给定两座城市之间的距离，找一条距离长度总和最小的路径。
- 归约：哈密顿回路。

三染色

- 给定一张无向图，将每个点染三种颜色之一，使得相邻的点颜色不同。
- 归约：



3-Dimensional Matching

- 给定三个不交的大小为 n 的集合 X,Y,Z，和一个”边”集 T 属于 X x Y x Z，求一个大小为 n 的边集使得边中的所有点互不相同。
- 归约：归约到 3-SAT。

子集和问题

- 给定自然数集合 w1,w2,…,wn 和目标 W，问是否存在一个子集使得和等于 W。
- 归约：到 3-Dimensional Matching Problem。

背包问题

- 有一些物品，每个物品有体积 v 和价值 w，给定体积上限 V，你要选择一个物品的集合使得体积和不超过 V，且价值和最大。
- 背包问题是一个 NP-Hard 问题。
- 归约：子集和问题。