实验舱
青少年编程
走近科学 走进名校

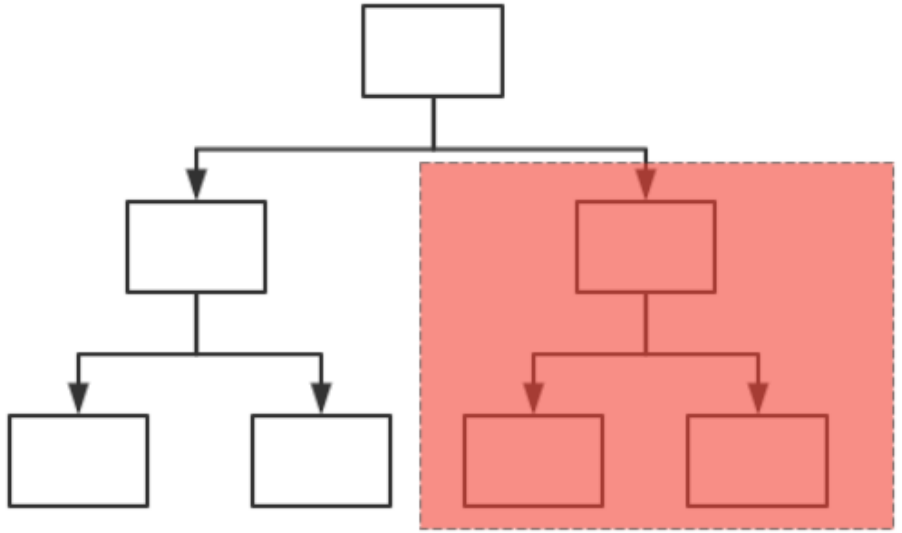
蛟龙四班

搜索的剪枝与优化

Mas

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。



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。

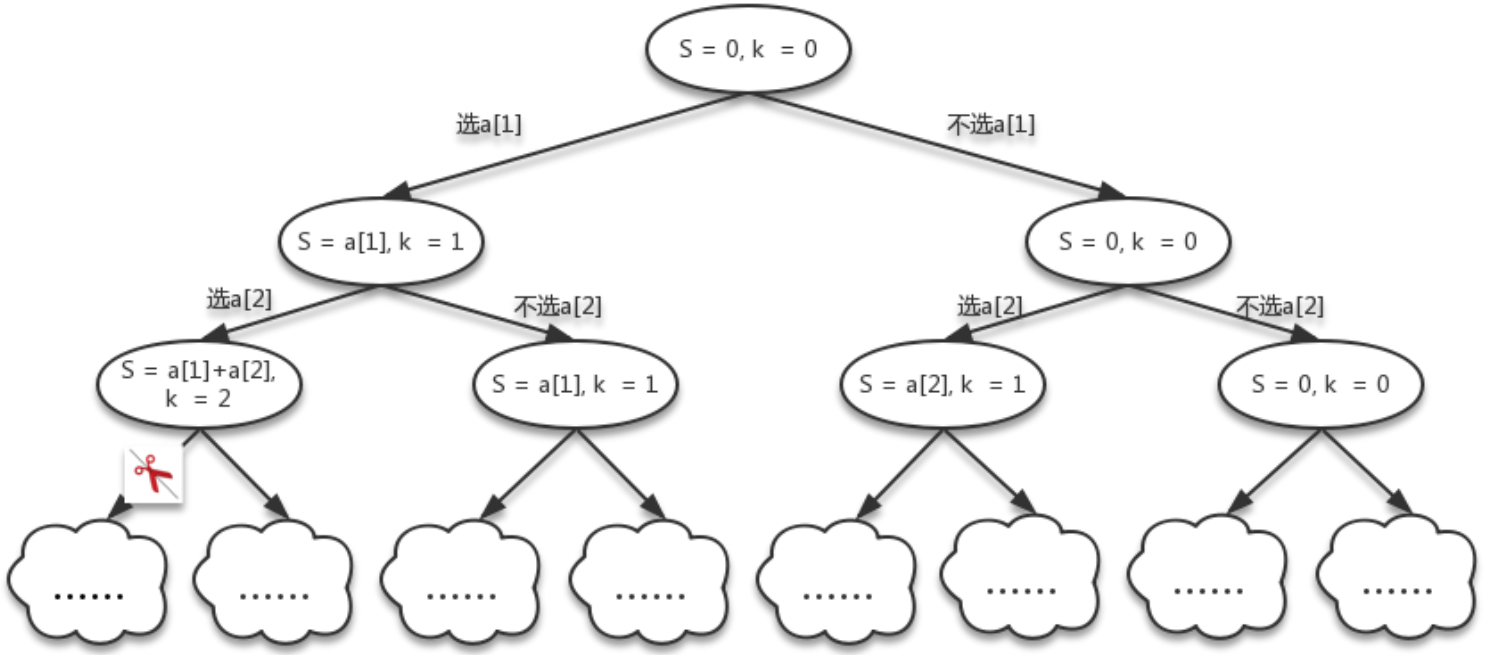
再来看这个问题：

给定 n 个整数，要求选出 K 个数，使得选出来的 K 个数的和为 sum。

如果搜索方法是每次从剩下的数里选一个数，一共搜到第 k 层，那么 1, 2, 3 这个选取方法能被搜索到 6 次，这是没必要的，因为我们只关注选出来的数的和，而根本不会关注选出来的数的顺序，所以这里可以用重复性剪枝。

我们规定选出来的数的位置是递增的，在搜索的时候，用一个参数来记录上一次选取的数的位置，那么此次选择我们从这个数之后开始选取，这样最后选出来的方案就不会重复了。

可行性剪枝



回顾一下之前讨论过的这个问题：

给定 n 个整数要求选出 K 个数，使得选出来的 K 个数的和为 sum。

如图，当 k=2 的时候，如果已经选了 2 个数，再往后选多的数是没有意义的。我们可以直接减去这个搜索分支。

又比如，如果所有的数都是正数，一旦发现当前的和值都已经大于 sum 了，之后怎么选和都不可能回到 sum 了，我们也可以终止这个分支的搜索。

我们在搜索过程中，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。

#1838 小猫爬山

题目描述

翰翰和达达饲养了 N 只小猫，这天，小猫们要去爬山。

经历了千辛万苦，小猫们终于爬上了山顶，但是疲倦的它们再也不想徒步走下山了（呜咕>_<）。

翰翰和达达只好花钱让它们坐索道下山。

索道上的缆车最大承重量为 W ，而 N 只小猫的重量分别是 C_1 、 C_2 $\ldots\ldots$ C_N 。

当然，每辆缆车上的小猫的重量之和不能超过 W 。

每租用一辆缆车，翰翰和达达就要付 1 美元，所以他们想知道，最少需要付多少美元才能把这 N 只小猫都运送下山？

输入格式

第 1 行：包含两个用空格隔开的整数， N 和 W 。
第 $2 \ldots N+1$ 行：每行一个整数，其中第 $i+1$ 行的整数表示第 i 只小猫的重量 C_i 。

输出格式

输出一个整数，表示最少需要多少美元，也就是最少需要多少辆缆车。

输入样例：

```
5 1996
1
2
1994
12
29
```

输出样例：

```
2
```

数据范围

$$1 \leq N \leq 18, 1 \leq Ci \leq W \leq 10^8$$

#1066 素数环

【问题描述】

给定一个 n，求 $1..n$ 组成的环，使得环上相邻的元素和为素数。

【输入格式】

输入一个正整数 $n(1 \leq n \leq 17)$

【输出格式】

把 1 放在第一位置，按照字典顺序不重复的输出所有解(顺时针，逆时针算不同的两种)，相邻两数之间严格用一个整数隔开，每一行的末尾不能有多余的空格。

如果没有答案请输出" $no\ answer$ "

【输入样例】

```
8
```

【输出样例】

```
1 2 3 8 5 6 7 4
1 2 5 8 3 4 7 6
1 4 7 6 5 8 3 2
1 6 7 4 3 8 5 2
```

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。

此外，在搜索是否有可行解的过程中，一旦找到了一组可行解，后面所有的搜索都不必再进行了，这算是最优性剪枝的一个特例。

最优性剪枝

【题目描述】

一个迷宫由R行C列格子组成，有的格子里有障碍物，不能走；有的格子是空地，可以走。

给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。

只能在水平方向或垂直方向走，不能斜着走。

【输入】

第一行是两个整数，R和C，代表迷宫的长和宽。（ $1 \leq R, C \leq 40$)

接下来是R行，每行C个字符，代表整个迷宫。

空地格子用.表示，有障碍物的格子用#表示。

迷宫左上角和右下角都是.。

【输出】

输出从左上角走到右下角至少要经过多少步（即至少要经过多少个空地格子）。计算步数要包括起点和终点。

【输入样例】

```
5 5
..###
#....
#.#.#
#.#.#
#.#..
```

【输出样例】

```
9
```

如果用DFS来解，第一个搜到的答案 ans不一定是正解，但是正解一定小于等于 ans。

如果当前步数大于等于 ans就直接剪枝，并且每找到一个可行的答案，都会更新 ans。

#1845 送礼物

题目描述

Mas 帮 $Moen$ 给女生送礼物，$Moen$ 一共准备了 n 个礼物，其中第 i 个礼物的重量是 g_i 。

Mas 的力气很大，他一次可以搬动重量之和不超过 w 的任意多个物品。

Mas 希望一次搬掉尽量重的一些物品，请你告诉达达在他的力气范围内一次性能搬动的最大重量是多少。

输入格式

第一行两个整数，分别代表 w 和 n 。

以后 n 行，每行一个正整数表示 g_i 。

输出格式

仅一个整数，表示 Mas 在他的力气范围内一次性能搬动的最大重量。

输入样例

```
20 5
7
5
4
18
1
```

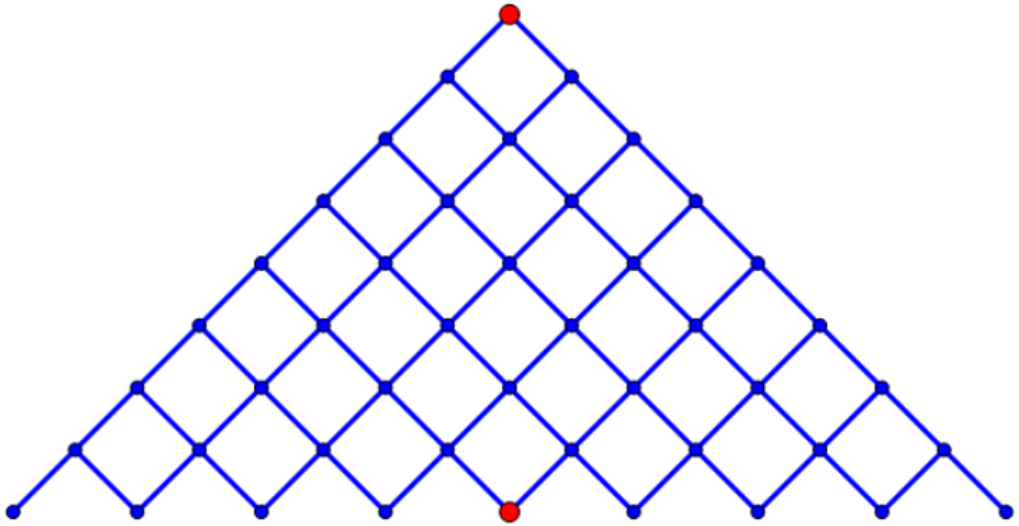
输出样例

```
19
```

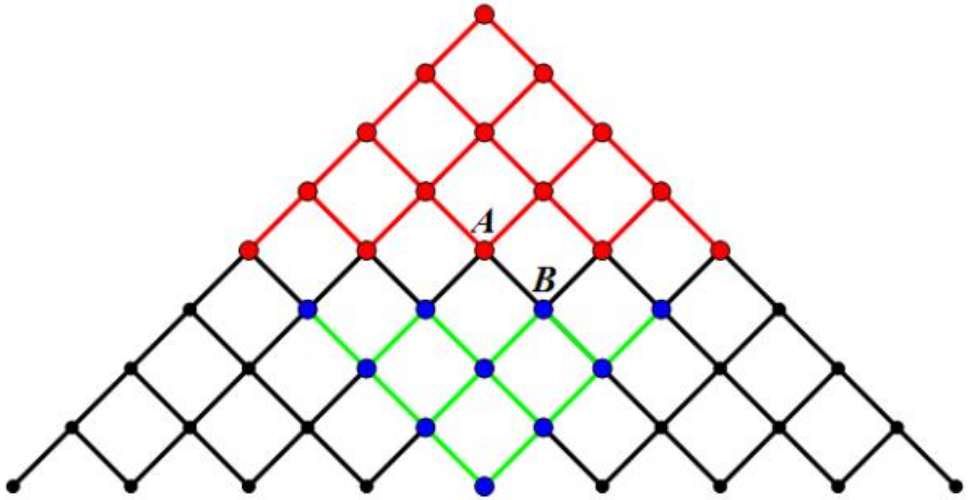
数据范围

对于 20% 的数据 $N \leq 20$
对于 40% 的数据 $1 \leq w, g_i \leq 2^{26}$
对于 100% 的数据 $1 \leq N \leq 45, 1 \leq w, g_i \leq 2^{31}-1$

DFS 与 meet-in-middle



深度优先搜索(depth-first-search)按照深度优先的方式进行搜索。
搜索是一种穷举的方式，把所有可行的方案都列出来，不断去尝试，直到找到问题的解。



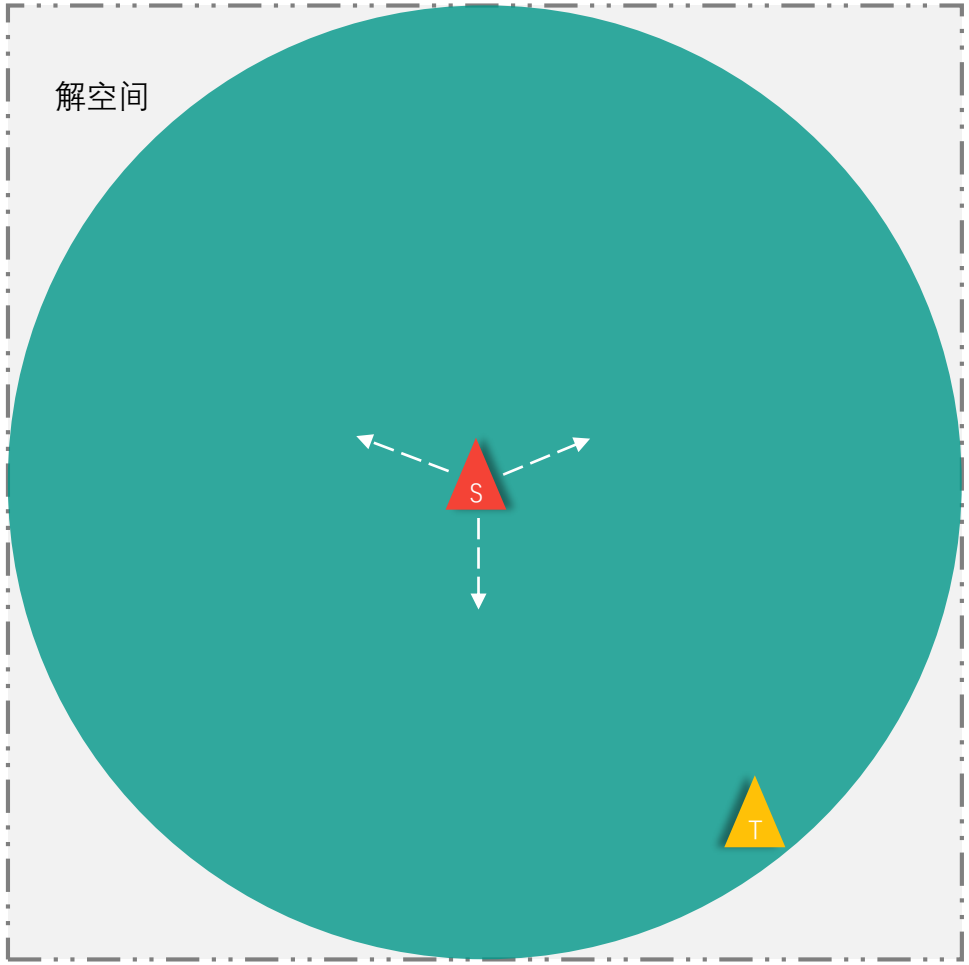
meet in the middle，主要思想是将整个搜索过程分成两半，分别搜索，最后将两半的结果合并。
由于搜索的复杂度往往是指数级的，而折半搜索可以使指数减半，也就能使复杂度开方。

单向BFS

传统的单向BFS是在有限的解空间里搜索。

从起点出发向外不断扩展结点(状态)，直到搜索到终点为止。

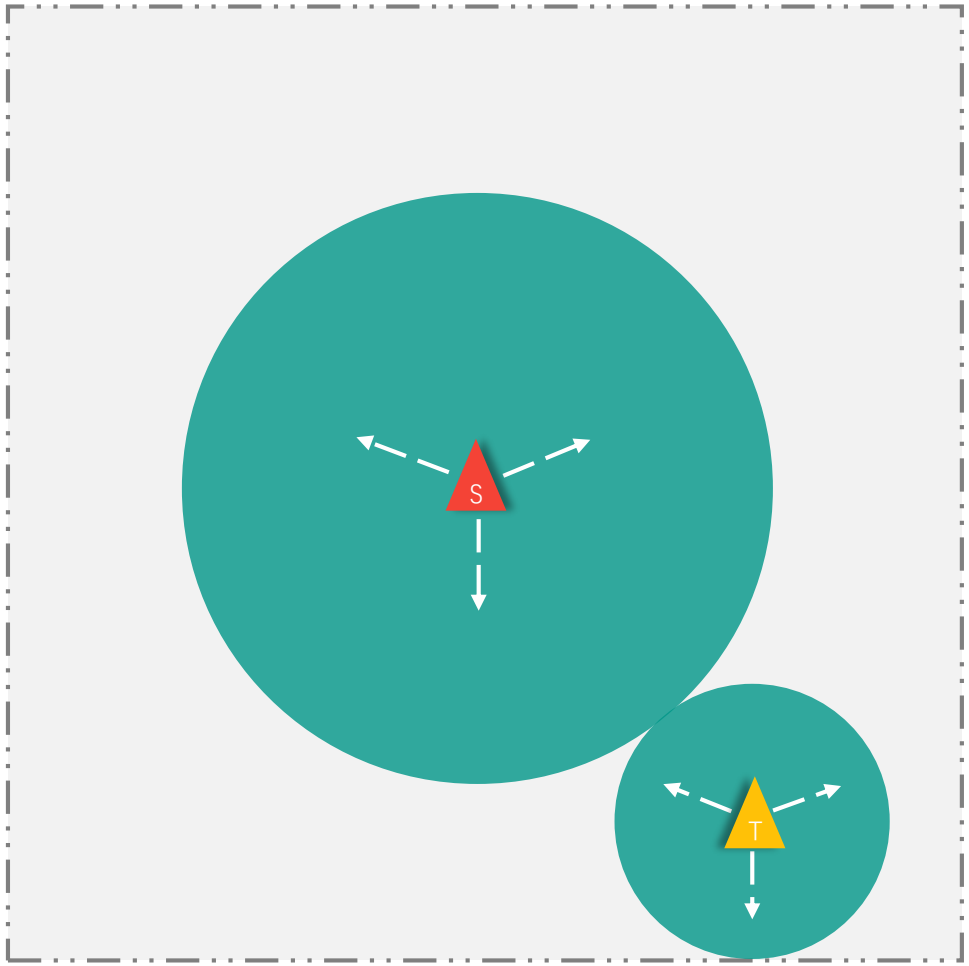
单向BFS的状态增长的指数级别增长的，如果起点和终点相距较远，状态的数量级会很大。



双向BFS

双BFS是同时从起点出发向外不断扩展结点(状态)，直到起点扩展的状态和终点扩展的状态相遇为止。

双向BFS使得，扩展的状态数少了很多，所以对于从起始状态和目标状态已知的情况，我们可以使用双向BFS来优化程序的时间/空间复杂度。

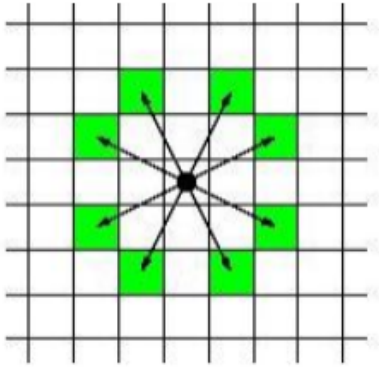


#634 Knight Moves

题目描述

原题来自：POJ 1915

编写一个程序，计算一个骑士从棋盘上的一个格子到另一个格子所需的最小步数。骑士一步可以移动到的位置由下图给出。



黑点表示骑士，绿色格子表示可跳到的位置

输入格式

第一行给出骑士的数量 n 。 在接下来的 $3n$ 行中，每 3 行描述了一个骑士。其中，

第一行一个整数 L 表示棋盘的大小，整个棋盘大小为 $L \times L$ ； 第二行和第三行分别包含一对整数 (x, y) ，表示骑士的起始点和终点。假设对于每一个骑士，起始点和终点均合理。

输出格式

对每一个骑士，输出一行一个整数表示需要移动的最小步数。如果起始点和终点相同，则输出 0 。

谢谢观看