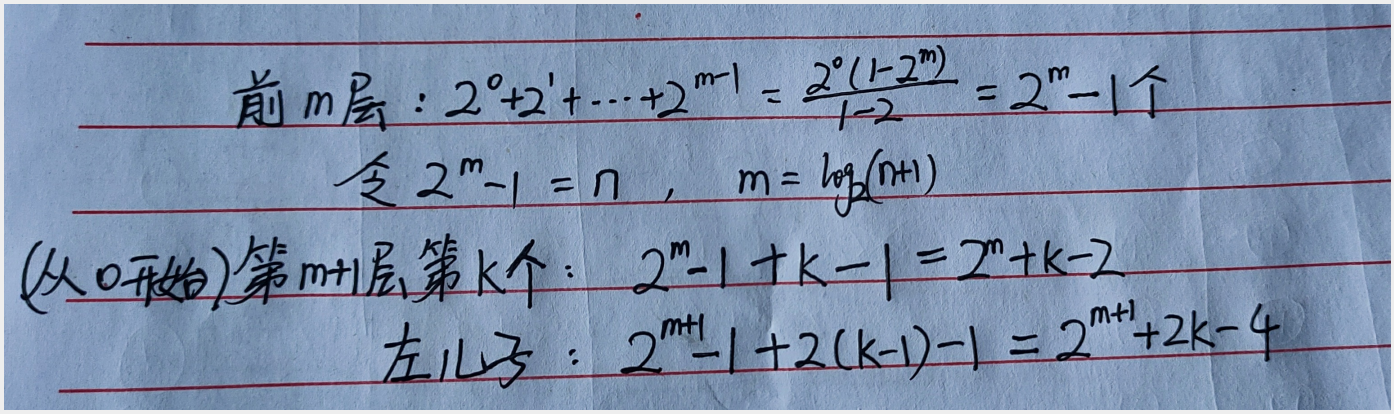
初级问题

堆、二叉搜索树、并查集

二叉堆（最小堆，优先队列）
二叉堆的特点：增删（可插入，只能删最小），儿子一定大于父亲，从上到下从左到右紧凑排列
堆的插入算法：在堆的末尾插入然后不断上升
堆删除最小值算法：“最后一个节点”移动并替换到根节点，优先选小儿子向下进行交换
堆的复杂度的证明
父亲与左儿子和右儿子的位置证明：



堆的插入的实现：sz的含义（插入节点的编号），父亲节点编号的计算方式（向下取整），向上交换时父亲下来儿子不用真上去（每次都让父亲和插入值比较，不合适就让父亲下降到上次改动的地方，找到合适的位置插值再进去）
堆的删除的实现：虽然向下交换之前是将最后一个结点移动到根节点，但可以让堆自行变换当找到这最后一个结点合适的位置后再插入，if(b<sz&&heap[b]<heap[a]) a=b;未越界情况下默认heap[a]代表小儿子，默认heap[i]代表父亲结点
priority_queue例子
/*POJ 2431
判断是否能到终点，和最少的加油次数
转换：在到达加油站i时，就获得了一次在之后的任何时候都可以加Bi单位汽油的权利
思路：燃料为0时再选择最大的加油量的进行加油
sp：遍历的是所有加油站而非距离按1自加
pos的取值范围
while循环中的操作：为完成当前的这段路程需要的尽可能加油
*/
PKU 3253

二叉搜索树--前中后遍历增大
增删查
查：根据和被查数的大小关系快速选择左右结点找到被查数，大右小左
增：与删类似，怎么插个18
删：提节点到需要删除的节点的三种情况，无左，左无右，左最大子孙

set容器和map容器
http://blog.sciencenet.cn/blog-3134052-1081994.html

并查集
查询是否ab同一组，合并ab所在组

查询：如果两个节点的根相同则在同一组
避免退化：高度小的成为高度大的子树，路径压缩（降低高度）

动态规划

动态规划

- 动态规划就是针对具有最优子序列的问题，先尝试进行遍历搜索然后增加记忆化搜索的技巧，然后提炼记忆化搜索的要点为递归关系式，最后将递推关系式转换为动态规划数组然后实现
- （异曲同工）具体来说，动态规划的一般流程就是三步：暴力的递归解法 -> 带备忘录的递归解法 -> 迭代的动态规划解法。就思考流程来说，就分为一下几步：找到状态和选择 -> 明确 dp 数组/函数的定义 -> 寻找状态之间的关系。
  -w.v; 2.3；1.2；3.4；2.2；

最大值在右上角=》最大值在右下角
从第几个物品开始=》从s数组的前几个元素中
包裹还剩多少称重=》t数组能遍历的部分不超过的元素
当前状态下的最大价值=》当前情况下最长公共子序列
//为什么LCS的长必须从左上方加1?
//书上代码实现时没考虑s[0]==j[0]

- dp数组常见为一个[m+1][n]的数组（），为了将初项初始化
- 先以从第i个物品开始挑选总重小于j的部分，建造一个最大值为右上角的数组训练
- 或者从前i个物品中挑选出总重量不超过j时总价值的最大值，建造一个最大值在右下角的数组
- 关于各种方向进行动态规划的一个选择
- 动态规划进阶
- 搜索的记忆化、利用递推关系的DP、从状态转移考虑的DP
- 问题1：用递推关系式递推出来的算法还是需要将所有的情况都遍历一遍吗（是否隐含了记忆化搜索，是否会自动剪枝跳过某些情况？）：就最长子序列这个问题的dp解法来看，没有剪枝，但是有记忆化搜索，核心就在dp数组他横纵坐标就是表示各种情况，并且每次都将当前情况下的最优解进行了一个保存，对于mn级数的情况，将得出最后结果的比较次数限制到了mn的多项式级数
- 最长公共子序列这个问题是只有s和t保存的值相等才往右下角+1
- 子序列解题模板
- 进一步探讨递归关系的例题仍可使用构造dp数组的办法
- 完全背包问题：从前i个物体中挑选总重量不超过j时总价值的最大值
- 58页代码要考虑的问题

1. 从前两种物体中挑选为什么默认选第二个进行考虑?
2. 思路是尽量挑选某种性价比更高的物体，对于最后剩下的空间也应当填满满足当前剩余空间要求的性价比更高的物体，这个事情在代码中有没有实现?

深搜与宽搜

- 递归函数

1. 优化：计算一次以后，用数列将结果存储起来，可以优化之后的计算

```
int memo[MAX_N + 1];

int fib(int n) {
  if (n <= 1) return n;
  if (memo[n] != 0) return memo[n];
  return memo[n] = fib(n - 1) + fib(n - 2);
}
```

- 栈

1. 举例

```
#include <stack>
#include <cstdio>

using namespace std;

int main() {
  stack<int> s;              // 声明存储int类型数据的栈
  s.push(1);                 // {} → {1}
  s.push(2);                 // {1} → {1,2}
  s.push(3);                 // {1,2} → {1,2,3}
  printf("%d\n", s.top()); // 3
  s.pop();                   // 从栈顶移除 {1,2,3}→{1,2}
  printf("%d\n", s.top()); // 2
  s.pop();                   // {1,2} → {1}
  printf("%d\n", s.top()); // 1
  s.pop();                   // {1} → {}
  return 0;
}
```

- 队列

1. 举例

```
#include <queue>
#include <cstdio>

using namespace std;

int main() {
  queue<int> que;               // 声明存储int类型数据的队列
  que.push(1);                  // {} → {1}
  que.push(2);                  // {1} → {1,2}
  que.push(3);                  // {1,2} → {1,2,3}
  printf("%d\n", que.front()); // 1
  que.pop();                    // 从队尾移除 {1,2,3}→{2,3}
  printf("%d\n", que.front()); // 2
  que.pop();                    // {2,3} → {3}
  printf("%d\n", que.front()); // 3
  que.pop();                    // {3} → {}
  return 0;
}
```

- 深度优先搜索

1. 设计：是否遍历了所有可能并对每种情况进行了判断，是否考虑了flase的情况，是否能进一步优化
2. 深搜的递归函数编写和状态管理都更简单，需要的内存空间与最大的递归深度成正比，与状态数相比深度并不会太大，相对来说更省内存?
3. 深搜的处理顺序：判断当前状态是否得到所求解，递归不加下一个状态的情况，递归加入下一个状态的情况，若没有得到所求解的情况返回false
4. 深搜适合处理：有没有正确解，正确解的个数

- ◦ 从一组数中能否选出特定的数使其运算结果为指定的数
- ◦ 八连通积水问题

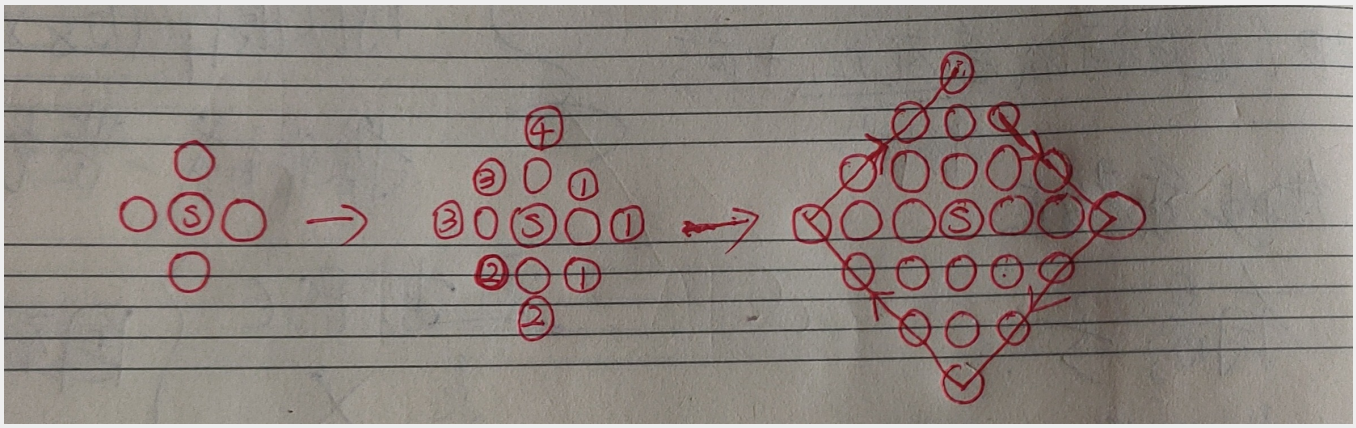
- 宽度优先搜索

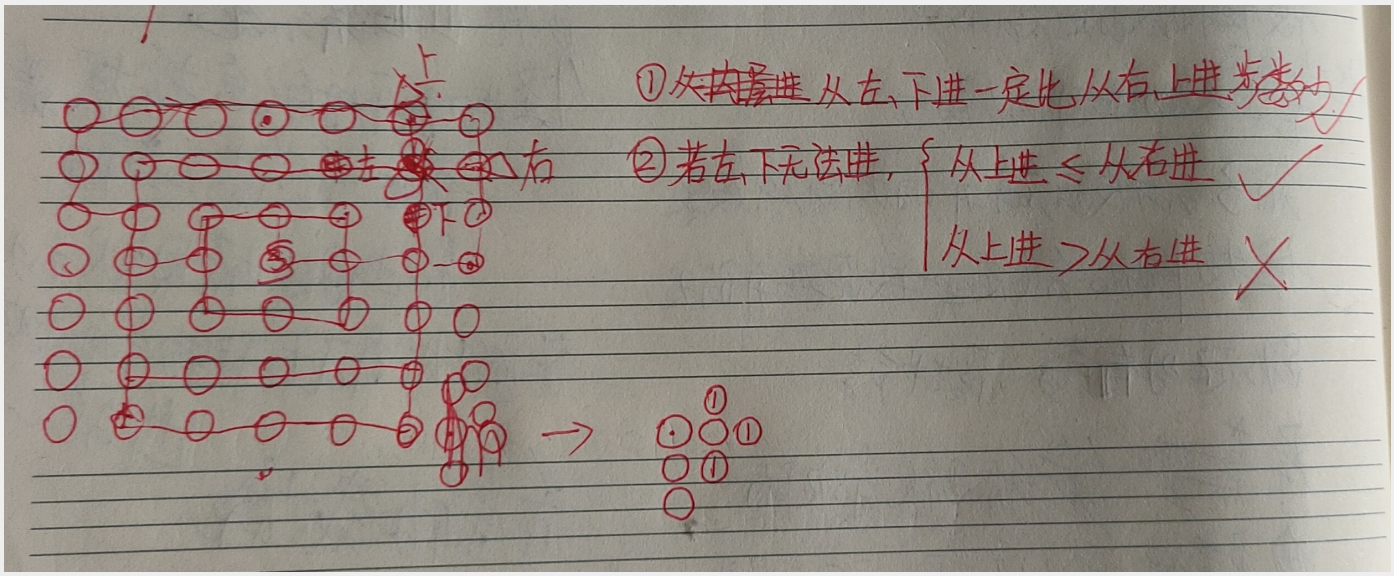
1. 深搜隐式利用了栈，宽搜利用了队列，会把状态逐个加入队列，通常需要与状态数成正比的内存空间，相对而言更耗内存
2. 宽搜按照距开始状态由近及远的顺序进行搜索，很容易用来求解最短路径、最少操作等问题
3. 当状态更加复杂时，可能需要封装成一个类来表示状态
4. 宽度优先搜索中，只要将已经访问过的状态用标记管理起来 ，就可以很好地做到由近及远的搜索。 这个问题中由于要求最短距离，不妨用d[N][M]数组把最短距离保存起来。初始时用充分大的常 数INF来初始化它，这样尚未到达的位置就是INF, 也就同时起到了标记的作用。虽然到达终点时就会停止搜索，可如果继续下去直到队列为空的话，就可以计算出到各个位置的 最短距离。此外，如果搜索到最后，d依然为INF的话，便可得知这个位置就是无法从起点到达的 位置。在今后的程序中，使用像INF这样充分大的常数的情况还很多。不把INF当作例外，而是直接参与普通运算的情况也很常见。这种情况下，如果INF过大就可能带来溢出的危险。

5. IDDFS迭代加深深度优先搜索与宽度优先搜索的状态转移顺序类似，但更加节约内存。最开始将深度优先搜索的递归次数限制在1次，在找到解之前不断增加递归深度?
6. 宽搜的处理顺序：从起点开始循环，每次都把取出队列头部的点进行检测，让后将其相邻的能走通的点依次放入队列的尾部，并将起点到该相邻点的距离+1

- ◦ 走出迷宫的最小步数

1. 这道题的前提是已经知道重点的坐标但不知道能否走通，如果不知道重点的坐标只需加入每一步都对被扫描的点进行检测看是不是终点
2. 思路：INF初始化，将起点加入队列，从起点开始循环进行宽搜，通过循环结束时终点到起点的距离值(到终点步数选最小的记录)来判断能否达到和能达到的最小步数
3. 宽搜的处理顺序：从起点开始循环，每次都把取出队列头部的点进行检测，让后将其相邻的能走通的点依次放入队列的尾部，并将起点到该相邻点的距离+1
4. INF的作用：充当了某个位置是否被检测过的标识符，充当了迷宫是否能走通的标识符（若走不通，最后返回终点到起点的距离值仍为INF）
5. typedef pair<int,int> P;? pair模板只能定义两个变量等价于只有两个变量的结构体，可以直接用pair.first和pair.second分别直接调用第一个和第二个变量，充当某一时刻被扫描的位置
6. char maze[MAX_N][MAX_M+1];?
7. d[MAX_N][MAX_M];//表示到各个位置的最短举例的数组
8. int dx[4]={1,0,-1,0},dy[4] = {0,1,0,-1};//二重数组配合for循环来表示二维向量，与八连通的处理办法不同
9. queue<. P > que;创建一个名字为que的队列，这个队列的每个元素都是P这种模板的结构体
10. 书中寻找最短步数的漏洞，终点步数赋值前进行比较





- 特殊状态的枚举
  不用深搜简短地实现寻找可行解：next_permutation函数或位运算?
- 剪枝
  当当前状态无论这样转移都不会存在解时不再搜索而是直接跳过

- 栈内存和堆内存
  栈内存区：主调函数拥有的局部变量等信息，程序启动后不能再扩大，因此函数递归深度有上限但仍可以在C和C++中进行上万次递归，java中可以指定栈的大小
  堆内存区：用new或malloc进行分配的内存区，也是全局变量保存的地方，当必须申请巨大的数组的时候，放在堆内存区可以减少栈溢出的危险，申请内存时一般申请稍微大一点