# 搜索入门

## **深度优先搜索DFS**

### **算法原理**

**深度优先搜索即Depth First Search，是图遍历算法的一种。**

**DFS的具体算法描述为选择一个起始点v作为当前结点，执行如下操作：**

**a. 访问 当前结点，并且标记该结点已被访问，然后跳转到b；**

**b. 如果存在一个和 当前结点 相邻并且尚未被访问的结点u，则将u设为 当前结点，继续执行a；**

**c. 如果不存在这样的u，则进行回溯，回溯的过程就是回退 当前结点；**

**上述所说的当前结点需要用一个栈来维护，每次访问到的结点入栈，回溯的时候出栈。**

### **算法实现**

**深搜最简单的实现就是递归，写成伪代码如下：**

**def DFS(v):**

**visited[v] = true**

**dosomething(v)**

**for u in adjcent\_list[v]:**

**if visited[u] is false:**

**DFS(u)**

**其中dosomething表示访问时具体要干的事情，根据情况而定，并且DFS是允许有返回值的。**

### **基础应用**

**a. 求N的阶乘；**

**b. 求斐波那契数列的第N项；**

**c. 求N个数的全排列；**

### **高级应用**

**a. 枚举：**

**数据范围较小的的排列、组合的穷举；**

**b. 容斥原理：**

**利用深搜计算一个公式，本质还是做枚举；**

**c. 基于状态压缩的动态规划：**

**一般解决棋盘摆放问题，k进制表示状态，然后利用深搜进行状态转移；**

**d.记忆化搜索：**

**某个状态已经被计算出来，就将它cache住，下次要用的时候不需要重新求，此所谓记忆化。下面会详细讲到记忆化搜索的应用范围；**

**e.有向图强连通分量：**

**经典的Tarjan算法；**

**求解2-sat问题的基础；**

**f. 无向图割边割点和双连通分量：**

**经典的Tarjan算法；**

**g. LCA：**

**最近公共祖先递归求解；**

**h.博弈：**

**利用深搜计算SG值；**

**i.二分图最大匹配：**

**经典的匈牙利算法；**

**最小顶点覆盖、最大独立集、最小值支配集 向二分图的转化；**

**j.欧拉回路：**

**经典的圈套圈算法；**

**k. K短路：**

**依赖数据，数据不卡的话可以采用2分答案 + 深搜；也可以用广搜 + A\***

**l. 线段树**

**二分经典思想，配合深搜枚举左右子树；**

**m. 最大团**

**极大完全子图的优化算法。**

**n. 最大流**

**EK算法求任意路径中有涉及。**

**o. 树形DP：**

**即树形动态规划，父结点的值由各个子结点计算得出。**

**好的剪枝可以大大提升程序的运行效率**

#### **剪枝原则**

**a. 正确性：剪掉的子树中如果存在可行解（或最优解），那么在其它的子树中很可能搜不到解导致搜索失败，所以剪枝的前提必须是要正确；**

**b. 准确性：剪枝要"准"。所谓"准"，就是要在保证在正确的前提下，尽可能多得剪枝。**

**c. 高效性：剪枝一般是通过一个函数来判断当前搜索空间是否是一个合法空间，在每个结点都会调用到这个函数，所以这个函数的效率很重要。**

#### **剪枝分类**

**可行性剪枝、最优性剪枝（上下界剪枝）。**

**1) 可行性剪枝**

**可行性剪枝一般是处理可行解的问题，如一个迷宫，问能否从起点到达目标点之类的。**

**2) 最优性剪枝（上下界剪枝）**

#### **即估价函数h = k + s，那么当前情况下存在最优解的必要条件是h < d，否则就可以剪枝了。最优性剪枝是不断优化解空间的过程。**

#### **基于DFS的A\*（迭代加深，IDA\*）**

## **二、广度优先搜索**

**1、BFS**

### **算法原理**

**广度优先搜索即Breadth First Search，也是图遍历算法的一种。**

**BFS的具体算法描述为选择一个起始点v放入一个先进先出的队列中，执行如下操作：**

**a. 如果队列不为空，弹出一个队列首元素，记为当前结点，执行b；否则算法结束；**

**b. 将与 当前结点 相邻并且尚未被访问的结点的信息进行更新，并且全部放入队列中，继续执行a；**

**维护广搜的数据结构是队列和HASH，队列就是官方所说的open-close表，HASH主要是用来标记状态的，比如某个状态并不是一个整数，可能是一个字符串，就需要用字符串映射到一个整数，可以自己写个散列HASH表，不建议用STL的map，效率奇低。**

**广搜最基础的应用是用来求图的最短路。**

### **算法实现**

**广搜一般用队列维护状态，写成伪代码如下：**

**def BFS(v):**

**resetArray(visited,false)**

**visited[v] = true**

**queue.push(v)**

**while not queue.empty():**

**v = queue.getfront\_and\_pop()**

**for u in adjcent\_list[v]:**

**if visited[u] is false:**

**dosomething(u)**

**queue.push(u)**

### **基础应用**

**a. 最短路：**

**bellman-ford最短路的优化算法SPFA，主体是利用BFS实现的。**

**b. 拓扑排序：**

**首先找入度为0的点入队，弹出元素执行“减度”操作，继续将减完度后入度为0的点入队，循环操作，直到队列为空，经典BFS操作；**

**c. FloodFill：**

**经典洪水灌溉算法；**

### **高级应用**

**a. 差分约束：数形结合的经典算法，利用SPFA来求解不等式组。**

**b. 稳定婚姻：二分图的稳定匹配问题，试问没有稳定的婚姻，如何有心思学习算法，所以一定要学好BFS啊；**

**c. AC自动机：字典树 + KMP + BFS，在设定失败指针的时候需要用到BFS。**

**d. 矩阵二分：矩阵乘法的状态转移图的构建可以采用BFS；**

**e. 基于k进制的状态压缩搜索：这里的k一般为2的幂，状态压缩就是将原本多维的状态压缩到一个k进制的整数中，便于存储在一个一维数组中，往往可以大大地节省空间，又由于k为2的幂，所以状态转移可以采用位运算进行加速，HDU1813和HDU3278以及HDU3900都是很好的例子；**

### **基于BFS的A\***

**在搜索的时候，结点信息要用堆（优先队列）维护大小，即能更快到达目标的结点优先弹出。**

**遇到搜索问题一般都是先分析状态，对于广搜来说这个状态量不算大，但是也不小，如果遇到无解的情况，就会把所有状态搜遍，所以这里必须先将无解的情况进行特判，采用的是曼哈顿距离和逆序数进行剪枝。**

**网上对A\*的描述写的都很复杂，首先还是从公式入手：**

**f(state) = g(state) + h(state)**

**g(state) 表示从初始状态 到 state 的实际行走步数，这个是通过BFS进行实时记录的，是一个已知量；**

**h(state) 表示从 state 到 目标状态 的期望步数，这个是一个估计值，不能准确得到，只能通过一些方法估计出一个值，并不准确；**

**f(state) 表示从 初始状态 到 目标状态 的期望步数，这个没什么好说的，就是前两个数相加得到，也肯定是个估计值；**

**对于广搜的状态，我们是用队列来维护的，所以state都是存在于队列中的，我们希望队列中状态的f(state)值是单调不降的（这样才能尽量早得搜到一个解），g(state)可以在状态扩展的时候由当前状态的父状态pstate的g(pstate)+1得到；那么问题就在于h(state)，用什么来作为state的期望步数，这个对于每个问题都是不一样的。**

#### **K短路问题**

**求初始结点到目标结点的第K短路，当K=1时，即最短路问题，K=2时，则为次短路问题，当K >= 3时需要A\*求解。**

**还是一个h(state)函数，这里可以采用state到目标结点的最短距离为期望距离；**

#### **双向广搜**

**1) 算法原理**

**初始状态 和 目标状态 都知道，求初始状态到目标状态的最短距离；**

**利用两个队列，初始化时初始状态在1号队列里，目标状态在2号队列里，并且记录这两个状态的层次都为0，然后分别执行如下操作：**

**a.若1号队列已空，则结束搜索，否则从1号队列逐个弹出层次为K(K >= 0)的状态；**

**i. 如果该状态在2号队列扩展状态时已经扩展到过，那么最短距离为两个队列扩展状态的层次加和，结束搜索；**

**ii. 否则和BFS一样扩展状态，放入1号队列，直到队列首元素的层次为K+1时执行b；**

**b.若2号队列已空，则结束搜索，否则从2号队列逐个弹出层次为K(K >= 0)的状态；**

**i. 如果该状态在1号队列扩展状态时已经扩展到过，那么最短距离为两个队列扩展状态的层次加和，结束搜索；**

**ii. 否则和BFS一样扩展状态，放入2号队列，直到队列首元素的层次为K+1时执行a；**

# **动态规划**

## **状态和状态转移**

**在介绍递推和记忆化搜索的时候，都会涉及到一个词---状态，它表示了解决某一问题的中间结果，这是一个比较抽象的概念，无论是递推还是记忆化搜索，首先要设计出合适的状态，然后通过状态的特征建立状态转移方程（f[i] = f[i-1] + f[i-2] 就是一个简单的状态转移方程）。**

## **最优化原理和最优子结构**

**如果问题的最优解包含的子问题的解也是最优的，就称该问题具有最有子结构，即满足最优化原理。**

## **决策和无后效性**

**一个状态演变到另一个状态，往往是通过“决策”来进行的。有了“决策”，就会有状态转移。而无后效性，就是一旦某个状态确定后，它之前的状态无法对它之后的状态产生“效应”（影响）。**

## **动态规划的经典模型**

### **1、线性模型**

**线性模型的是动态规划中最常用的模型，上文讲到的最长单调子序列就是经典的线性模型，这里的线性指的是状态的排布是呈线性的。**

### **2、区间模型**

**区间模型的状态表示一般为d[i][j]，表示区间[i, j]上的最优解，然后通过状态转移计算出[i+1, j]或者[i, j+1]上的最优解，逐步扩大区间的范围，最终求得[1, len]的最优解。**

### **3、背包模型**

**背包问题是动态规划中一个最典型的问题之一。**

**有N种物品（每种物品1件）和一个容量为V的背包。放入第 i 种物品耗费的空间是Ci，得到的价值是Wi。求解将哪些物品装入背包可使价值总和最大。f[i][v]表示前i种物品恰好放入一个容量为v的背包可以获得的最大价值。决策为第i个物品在前i-1个物品放置完毕后，是选择放还是不放，状态转移方程为： f[i][v] = max{ f[i-1][v], f[i-1][v - Ci] +Wi }时间复杂度O(VN)，空间复杂度O(VN) （空间复杂度可利用滚动数组进行优化达到O(V)，下文会介绍滚动数组优化）。**

### **4、状态压缩模型**

**状态压缩的动态规划，一般处理的是数据规模较小的问题，将状态压缩成k进制的整数，k取2时最为常见。**

### **5、树状模型**

**树形动态规划（树形DP），是指状态图是一棵树，状态转移也发生在树上，父结点的值通过所有子结点计算完毕后得出。给定一颗树，和树上每个结点的权值，求一颗非空子树，使得权和最大。**

**用d[1][i] 表示i这个结点选中的情况下，以i为根的子树的权和最大值;**

**用d[0][i]表示i这个结点不选中的情况下，以i为根的子树的权和最大值;**

**d[1][i] = v[i] + sum{ d[1][v] | v是i的直接子结点 && d[1][v] > 0**

**d[0][i] = max( 0, max{ max( d[0][v], d[1][v] ) | v是i的直接子结点 } )**

**这样，构造一个以1为根结点的树，然后就可以通过dfs求解了。**

**这题题目要求求出的树为非空树，所以当所有权值都为负数的情况下需要特殊处理，选择所有权值中最大的那个作为答案。**

## **常用状态转移方程**

### **动态规划算法三要素**

**（摘自黑书，总结的很好，很有概括性）：**

**①所有不同的子问题组成的表**

**②解决问题的依赖关系可以看成是一个图**

**③填充子问题的顺序（即对②的图进行拓扑排序，填充的过程称为状态转移）；**

**则如果子问题的数目为O(nt)，每个子问题需要用到O(ne)个子问题的结果，那么我们称它为tD/eD的问题，于是可以总结出四类常用的动态规划方程：下面会把opt作为取最优值的函数（一般取min或max）, w(j, i)为一个实函数，其它变量都可以在常数时间计算出来**

**1、1D/1D**

**d[i] = opt{ d[j] + w(j, i) | 0 <= i < j } (1 <= i <= n)**

**2、2D/0D**

**d[i][j] = opt{ d[i-1][j] + xi, d[i][j-1] + yj, d[i-1][j-1] + zij } (1<= i, j <= n)**

**3、2D/1D**

**d[i][j] = w(i, j) + opt{ d[i][k-1] + d[k][j] }, (1 <= i < j <= n)**

**区间模型常用方程。**

**另外一种常用的2D/1D的方程为：**

**d[i][j] = opt{ d[i-1][k] + w(i, j, k) | k < j } (1<= i <= n, 1 <= j <= m)**

**4、2D/2D**

**d[i][j] = opt{ d[i'][j'] + w(i', j', i, j) | 0 <= i' < i, 0 <= j' < j}**

**常见于二维的迷宫问题，由于复杂度比较大，所以一般配合数据结构优化，如线段树、树状数组等。**

**对于一个tD/eD 的动态规划问题，在不经过任何优化的情况下，可以粗略得到一个时间复杂度是O(nt+e)，空间复杂度是O(nt)的算法，大多数情况下空间复杂度是很容易优化的，难点在于时间复杂度，下一章我们将详细讲解各种情况下的动态规划优化算法。**

## **动态规划和数据结构结合的常用优化**

### **1、滚动数组**

**我们发现将d[i][j]理解成一个二维的矩阵，i表示行，j表示列，那么第i行的结果只取决于第i+1和第i行的情况，对于第i+2行它表示并不关心，那么我们只要用一个d[2][N]的数组就能保存状态了，其中d[0][N]为奇数行的状态值，d[1][N]为偶数行的状态值，当前需要计算的状态行数为奇数时，会利用到d[1][N]的部分状态，奇数行计算完毕，d[1][N]整行状态都没用了，可以用于下一行状态的保存，类似“传送带”的滚动来循环利用空间资源，美其名曰 - 滚动数组。**

**这是个2D/0D问题，理论的空间复杂度是O(n2)，利用滚动数组可以将空间降掉一维，变成O(n)。**

**背包问题的几个状态转移方程同样可以用滚动数组进行空间优化。**

### **2、最长单调子序列**

**d[i] = max{ d[j] | j < i && a[j] < a[i] } + 1;**

**那个问题的状态转移方程如下：**

**最长递增子序列的N变成100000，其余不变。**

**首先明确决策的概念，我们认为 j 和 k (j < i, k < i)都是在计算d[i]时的两个决策。那么假设他们满足a[j] < a[k]（它们的状态对应就是d[j] 和 d[k]），如果a[i] > a[k]，则必然有a[i] > a[j]，能够选k做决策的也必然能够选 j 做决策，那么如若此时d[j] >= d[k]，显然k不可能是最优决策（j的决策始终比它优，以j做决策，a[ j ]的值小但状态值却更大），所以d[k]是不需要保存的。基于以上理论，我们可以采用二分枚举，维护一个值 (这里的值指的是a[i]) 递增的决策序列，不断扩大决策序列，最后决策的数目就是最长递增子序列的长度。具体做法是：**

**枚举i，如果a[i]比决策序列中最大的元素的值还大，则将i插入到决策序列的尾部；否则二分枚举决策序列，找出其中值最小的一个决策k，并且满足a[k] > a[i]，然后用决策i替换决策k。**

**这是个1D/1D问题，理论的时间复杂度是O(n2)，利用单调性优化后可以将复杂度将至O(nlogn)。**

**给定n个元素(n <= 100000)的序列，将序列的所有数分成x堆，每堆都是单调不增的，求x的最小值。**

**结论：可以转化成求原序列的最长递增子序列。**

**证明：因为这x堆中每堆元素都是单调不增的，所以原序列的最长递增子序列的每个元素在分出来的每堆元素中一定只出现最多一个，那么最长递增子序列的长度L的最大值为x，所以x >= L。而我们要求的是x的最小值，于是这个最小值就是 L 了。**

### **3、矩阵优化**

**n的范围比较大，虽然是几个简单的加法方程，但是一眼看下去也不知道有什么规律可循。我们把状态转移用另外一种形式表现出来，存储图的连通信息的一种方法就是矩阵。令这个矩阵为A，Aij表示从i号岛到j号岛是否连通，连通标1，不连通标0，它还有另外一个含义，就是经过1天，从i岛到j岛的方案数，利用矩阵的传递性，A2的第i行的第j列则表示经过2天，从i岛到j岛的方案数，同样的，An 则表示了经过n天，从i岛到j岛的方案数，那么问题就转化成了求An MOD 100000007的值了。 An当n为偶数的时候等于(An/2)\*(An/2)；当n为奇数的时候，等于(An/2)\*(An/2)\*A，这样求解矩阵An就可以在O(logn)的时间内完成了，加法和乘法对MOD操作都是可交换的（即 “先加再模” 和 “先模再加再模” 等价），所以可以在矩阵乘法求解的时候，一旦超出模数，就执行取模操作。**

**最后求得的矩阵T = An MOD 100000007，那么T[1][1]就是我们要求的解了。**

### **4、斜率优化**

**那么可以用单调队列来维护一个决策队列的单调性，单调队列存的是决策序列。**

**一开始队列里只有一个决策，就是0这个点（虚拟出的初始决策），根据第一个结论，如果队列里面决策数目大于1，则判断slope( Q[front], Q[front+1] ) < 2\*s[i]是否成立，如果成立，Q[front]是个无用决策，front ++，如果不成立那么Q[front]必定是当前i的最优决策，通过状态转移方程计算f[i]的值，然后再根据第二个结论，判断slope(Q[rear-2], Q[rear-1]) > slope(Q[rear-1], i)是否成立，如果成立，那么Q[rear-1]必定是个无用决策，rear --，如果不成立，则将 i 作为当前决策 插入到队列尾， 即 Q[rear++] = i。**

**这题需要注意，斜率计算的时候，分母有可能为0的情况。**

### **5、树状数组优化**

**树状数组是一种数据结构，它支持两种操作：**

**1、对点更新，即在给你(x, v)的情况下，在下标x的位置累加一个和v（耗时 O(log(n)) ）。**

**函数表示 void add(x, v);**

**2、成端求和，给定一段区间[x, y]，求区间内数据的和（这些数据就是第一个操作累加的数据，耗时O(log(n))）。**

**函数表示 int sum(x, y);**

**用其它数据结构也是可以实现上述操作的，例如线段树（可以认为它是一种轻量级的线段树，但是线段树能解决的问题更加普遍，而树状数组只能处理求和问题），但是树状数组的实现方便、常数复杂度低，使得它在解决对点更新成端求和问题上成为首选。这里并不会讲它的具体实现，有兴趣请参见树状数组。**

### **6、线段树优化**

**线段树是一种完全二叉树，它支持区间求和、区间最值等一系列区间问题，这里为了将问题简化，直接给出求值函数而暂时不去讨论它的具体实现，有兴趣的可以自行寻找资料。线段树可以扩展到二维，二维线段树是一棵四叉树，一般用于解决平面统计问题，参见二维线段树。**

### **7、其他优化**

**a.散列HASH状态表示**

**b.结合数论优化**

**c.结合计算几何优化**

**d.四边形不等式优化**

**e.等等**

# 计算几何

Pick定理和叉积求多边形的面积。

Pick定理：一个计算点阵中顶点在格点上的多边形面积公式：S=a+b/2-1，其中a表示多边形内部的点数，b表示多边形边界上的点数，s表示多边形的面积。

多边形面积：

1) △ABC的面积为向量AB与向量AC的叉乘的一半。

2)对于一个多边形，选定一个顶点P1，与其他顶点连线，可将多边形分为若干个三角形。

3)多边形面积为 abs(Sum{CrossMul(A,B,P1)|A,B为相邻的两个顶点}) (先求和再取abs，否则对于凹多边形会出错）

求在边上的顶点数：

对于Pa(x1,y1),Pb(x2,y2)所连成的选段，经过的格点的个数为Gcd(abs(x1-x2),abs(y1-y2))+1.

# 数据结构汇总

ACAutoman AC自动机

Avltree 平衡树

Bellmanford 最短路

BM字符串模式匹配

BinarySearch 二叉搜索树

Dijkstra 最短路

FFT 傅里叶快速闭环

Floyed 最短路

HashTable 哈希

KMP 字符串模式匹配

Linklist 链表

LIS 最长非降子序列

MergeArray 归并

Mentecarlo 蒙特卡洛

Krusakal 最小生成树

Prim 最小生成树

NTT 快速数论变换

Radixsort 基数排序

RMQ 区间最值查询

RulerExtraction 尺取法

Segtree 线段树

SPFA 最短路

ThreeDevide 三分法

# 标准模板

#include<cstdio>

#include<queue>

#include<cmath>

#include<cstring>

#include<algorithm>

#include<iostream>

#define LL long long

#define clr(a, b) memset(a, b, sizeof(a)

#define height(p) ( (p==NULL) ? -1 : (((Node \*)(p))->height) )

#define max(a, b) ( (a) > (b) ? (a) : (b) )

const int MAXN = 100050;

const int HALF = 0x3f3f3f3f;

const int INF = (~(0x1<<31));

using namespace std;