USACO 课文简记

ADVENTop

简要记录了课文的学习体会等,可用于导读,仅供参考

ADVENTopLab
Hi.baidu.com/adventop

2010/2/9

USACO 课文学习简记

第一章:

1.1:TEXT Ad Hoc Problems

这是 USACO 的第一篇文章，《杂题》，所谓杂题，广义上讲:就是没有任何套路的题目,通常这种题目使用的是构造法，而模拟策略又居多，文章言简意赅的叙述了该类型的题目的出题形式，我个人的学习体会是，只要抓住这种题目的构造特点，**解题方法变幻无常**的性质。不难求解出本题的答案。

1.2 TEXT Complete Search

这是严格意义上的第一篇算法课文《完全搜索（枚举）》，枚举算法几乎是所有算法之母，因为后来的诸如高效的 DP,其实就是枚举的一种优化，毕竟你在求解问题时，特别是**离散**的问题模型，你不能猜测出问题的所有可能情况，那么你至少要遍历一遍（O(n)），而通常 O(1)的算法模型等等低于 O(n)的，必定是知道了全局的状况，而且极有可能的是有规律的（诸如：可以找到通项公式），或者是把时间几乎都用在了预处理上（等非算法核心的地方），所以完全搜索是算法的入门，在我看来它是**三大最基本算法**之一，另两者还有贪心，模拟策略。
枚举一定要遵循 **KISS** 原则，程序要尽量简洁，这种简洁不是说思考得简单，而是使用最巧妙的办法。这种算法一般应用于时间复杂度不高的问题模型，需要遍历所有的状态，因此这种算法要考虑到优化，优化一般是从状态数目，规模，算法实现本身来实现的。比如一个问题可以找到周期性将大大减少程序的运行时间，如果有适当的规律，问题的规模也就小了，所以即使是面对枚举题目，也要**动手模拟一下题目的状态，运行情况，以便于归纳到有效的优化方法。**
对于枚举不要一味的贪图时间效率而使程序变得错误百出，如果你已经尝试了所有的优化还是没有得到预期的结果那么枚举可能就不适合这道题了，这是需要注意的。

1.3:TEXT Greedy Algorithm

本篇课文介绍了最后一个最基本算法：《贪心》。至此 3 大最基本算法都有了介绍（基本算法不止这 3 个，这 3 个是最基本的），贪心法很重要，比如最短路算法，拓扑排序以及最小生成树的核心思想都是贪心。
贪心的中心就是**以局部最优表现全局最优，以全局最优说明局部最优**（对于贪心他们等价），和 DP 不同的是，DP 强调无后效性，然而贪心的每一步都影响最终的结果。贪心的难点在于证明，如何找到属于贪心的最优子结构？一般使用反证法证明，就是先假设不是最优的情况，如果推倒不出正确的结果那么贪心具有正确性，否则贪心是错误的。
面对状态的不确定性，贪心也往往得不到正确的结果。其实看出一道题目是贪心并不容易，我认为有以下几点：首先从状态上讲贪心应该比较单一或者简单，当状态繁纷复杂时，贪心本身不容易证明。其次是不要想当然，考虑是否具备最优

子结构。**认真分析**是所有算法的根基，面对贪心问题一般都会出现在组合算法中，单独出现的很少。

1.3:TEXT Winning Solutions

这是比较实用的文章，文章包括了优化手段，提高程序时效的方法，如何面对考试等等很多实际问题，比如一个程序的合理调试时间在20分钟以内，最后5分钟不要再写程序，尽量使用KISS原则，考虑所有可能的糟糕情况，并且试图避免。其中我的印象比较深刻的有：

1．KISS原则：简单就是智慧！

2．避免动态数据结构。

3．**DO THE MATH!**

其它的还有诸多的优化手段，比如对称性问题，分治。一个程序的优良，不只是算法本身，更有实现力度（健壮性），调试方便，简洁等多方面因素，想要养成一眼就能看出最优算法的本领，不能只依靠海量的书，阅读材料，更重要的是**上机动手**去做，只有经历过**千锤百炼**，方可练就：以无法为有法，以有限为无限的境界。

1.4:TEXT More Search Techniques

本章最具技术含量的课文《搜索技术》，少不了经典的 8 皇后问题与骑士覆盖。的确，搜索技术可以说基本上是能应付所有的常见问题，学好搜索，特别是回溯搜索技术，对以后更难的问题分析，起到至关重要的作用。3 大最基本算法是铸造金钥匙的前提，而**搜索技术**就是**迈入算法分析大门的金钥匙**，至此，有点算法分析的意思了。

课文中使用了经典的问题来描述算法，循序渐进的介绍了 DFS,BFS,ID 等多种常用搜索技术，搜索技术之所以通用是因为搜索技术**不局限于固定的套路和模式**，使用起来十分**灵活多变**。

DFS 通常是解决需要尽快找到可行解，或者所有的状态，它需要的是时间。

BFS 通常解决层次性强，而且是最优情况的，它对空间开销大。

DFS-ID 两者的综合，解决有充足的时间，但是空间不足的问题。

三者的时间复杂度都是指数级别的，BFS 可能略短，所以搜索方式的选择，对于搜索本身也是关键的一步。搜索的难点是状态的**全面性分析**以及**优化**，(相应的对于思维逻辑的要求较高，编程基本功要好)不能丢掉状态，更不能陷入死循环。优化一般是指剪枝，搜索方式等(比如 bit 运算优化)。找好了状态，配合一定的优化，加强练习，搜索不难掌握。

1.5:TEXT Introduction to Binary Numbers

第一个技术课文《二进制与位操作》，这篇课文比较基本，简单介绍了位的常用操作，以及数据类型的位与字节占用，其实位运算在优化搜索算法上有很大的优势，主要用位运算一般是用来**优化状态**，使用位表示状态，主要就是快，在 8 皇后问题中，派对灯(IOI98)中都能起到良好的优化效果，而且使程序长度缩短，不论从哪点都是好的选择，但是不是所有的搜索状态都可以用位运算优化，具体

的还需对具体问题的分析。

第二章:

2.1:TEXT Graph Theory

本章开始就有数据结构了,我们知道：**程序=数据结构+算法**。数据结构用来描述问题，使问题能够更高效、更方便的去解决。好的算法配合相应的数据结构，可使程序效率有本质的提升，比如：对于线性查找的优化，我们可以使用查找树，散列等方法。课文中主要讲的是《图论初步》，从图的基本概念，图的表示法以及图的基本算法几个方面来介绍图结构，课文涵盖的知识点丰富，且基础。对连通图的分析也浅显易懂，没有什么难以理解的部分，主要方法是如何建立图论模型，最主要的就是分析清楚题目的要点，看看“边”与“点”的关系，边点相应，就能初步建立图论模型。此外，课文最后还介绍了完全图、二分图和树及 DAG 的概念，为以后的学习打下坚固的基础。
我对数据结构的浅层次理解是：数据结构是对问题模型的一种**看法**，这种看法有时不止一种，看法不同，问题的解法也就不一，但是往往问题的解法比较集中，这样，一个好的“看法”就通常起到决定算法好坏的关键地步。所谓看法，其实是对问题模型的理解，你的理解深入，就能使用好的数据结构去描述题目。相应的算法也能抉择出更加高效的。所以，锻炼自己看法的好坏是十分重要的，不一定非要学习多么多的数据结构和多么深奥的算法（初期），而是扎扎实实的练就基础，以能做到深入剖析问题的能力。

2.1:TEXT Flood Fill Algorithms

这种算法是我在 USACO 才知道的《种子染色法》，其实就是用搜索打**标记**，不过不要小看了打标记，特别是对于搜索而言，标记能起到剪枝和使问题简化的地步。况且，这是实现 Floodfill 的基本方法。严谨的说应该叫做染色，通常有 DFS,BFS,广度优先扫描 3 种，其实广度优先扫描就是没有队列的 BFS,这样它的扩展会比较慢，但是省下了空间，以上的算法用于显式图可以，隐式图则需要考虑其他的实现方法，在这里，把 Floodfill 称作一种策略更准确，他本身不是明确的算法，包括实现手段也是借助搜索、扫描。Floodfill 一般用在求解连通性的问题，记录（染色）结点并进行扩展，其实连通性问题还可以用 Floyd 算法来实现，O(n^3)。但是对于染色,Floodfill 是个好方法。
只要做好标记，选择合适的实现手段，Floodfill 实现起来不难。

2.2:TEXT Data Structures

这是总体谈论《数据结构》的课文，纵观数据结构，就有可操作性，可以编程，按时解出，调试等多个方面需要考虑。单从编程角度讲，有状态的检查与调试时状态显示两方面，可以说剖析的具体透彻。当然最后还是谈论到了 **KISS 大法**与**速度**，也许对于 OI 来说：速度太重要了。同时课文还讲到了要尽量避免的两点：动态存储与“Cool”想法，主要是因为动态存储的处理麻烦容易出错，而“Cool”想法通常是临时想到的，所以它的可靠性难以保证。

接着课文就开始介绍基本数据结构了：
这些数据结构主要在检索方面发挥着重要的作用，比如最简单的二叉查找树，散列表，”Tries”,每个部分最后都又提到了这些数据结构的变种，需要注意的地方等。其中常用的二叉搜索树与 Hash 讲的比较详细，但是实现方法，就有普通的方法与链表两种。对于 Hash 函数的构造，令我印象深刻的是要用一个大的素数，在课文的最后与 Binary-Tree 的最后分别讲了：堆与平衡树，这是两个相对高等的数据结构。检索最值对于堆排序来说太适合了，特别是 Dijkstra 的优化，堆是不错的选择。而平衡二叉树，通过课文我了解到，对于检索可能会遇到最坏情况，或者使搜索变得麻烦，这时平衡树就起到作用了，也许因为它本身的性质使得最坏情况能得到有效的避免,能够永远保持平衡。
当然这些数据结构还有很多其它特性，课文所介绍的对于 OI 来说足够了。

2.2:TEXT Dynamic Programming

这篇文章介绍的比较简洁，《动态规划》，文章没有说 DP 的基本性质，而是开门见山从一道经典题目直接入手。从递归下降—>O(n^2)级别的 DP 方程：F[i]=max{F[j]}+1,—>使用滚动数组+二分查找进行优化，循序渐进。充分说明了这种程序设计技术的优势，接着又更进一步讲解了二维 DP，并一语道破 DP 的解题思想：寻找子问题。我想，**DP 问题其实是一种记忆化的程序设计思想**，它以空间换取时间，记录了所有状态以**尽量避免重复搜索**，不过这仅仅是 DP 之路的一扇门，若想了解更多，则应该加强训练，多研究其它经典 DP 模型，拓展知识面。

2.4:TEXT Shortest Paths

初等图论问题的首学算法《最短路》，课文从单源最短路问题与多源最短路问题两种来介绍，并使用例题加强说明。最短路问题的**核心思想就是贪心**，它满足贪心的最优子结构。体现了贪心扩展的思想，最短路问题并不难。对于单源的，贪心周围的结点并进行扩展。如果是多源的可以对每个结点进行单源查找，或者使用 Floyd，复杂度为 O(n^3)，实现起来比 Dijkstra 的多源要简单的多。Floyd 算法的本质是 DP，它不但可以查找最短路，还可以查找连通图，非常方便，主要使用中介结点更新。其实 Floyd 还可以应对负权图，Dijkstra 就不行了，另一种单源最短路算法 Bellman-Ford 则可以处理负权。而对于负权圈，Floyd 会陷入问题，应该用拓扑排序检查一遍。有了这些基础，最短路问题不难解决。

第三章:

3.1: TEXT Minimal Spanning Trees

依然是对于图论的介绍,不过这次介绍的是另一种基本的图论算法 MST,其实在遇到的这些**基本图论算法**的时候,主要的**考点依旧是建模**,而且无论是最短路还是 MST,应用于工程规划的居多,所以 MST 思想对于初步理解图论是重要的.
课文中主要介绍的是 Prim 算法,本算法相对容易,因为一个生成树的边是(n-1)条,所以先把初始节点加入(即最先扩展节点),然后加入所有与其相连的节点的权值,

这必定为(n-1)条,接着利用**贪心思想**,把当前树中最短的边刷新(加入新的节点),然后依次类推,直到扩展完毕.
方法简单易于实现是 Prim 的优点,但是他的某些局限性比如无法应用于有向图或者很难适应约束条件,使得他的应用受到限制.总之,我们还可以使用诸如 Kruskal 等算法.不过基本的思想仍然是贪心扩展.所以,对于最短路或者 MST 来说,贪心是非常重要的.

3.2: TEXT Knapsack Problems

背包问题是动态规划中的最常见问题之一,之所以这样,是因为它是**将大问题转化为小问题的典型代表**.同时也是常见的 NP 问题的一种.关于 NP 问题,有相当一部分用动态规划可以进行初步解决.可以说学习 DP 最常见的就是背包.因为他的延伸和实用价值都很高.特别是基本背包方程:
Dp[i][j]=MAX{Dp[i-1][j],Dp[i-1][j-w[i]]+v[i]}
就是 OI 界 DP 的代表.
背包拥有很多变种,包括小数背包,整数背包和基本的多重背包.而且背包的适用范围广到难以想象.通过背包可以映射其他的经典 DP 模型.而且他的方程表示浅显易懂.但是究竟为什么要使用背包?答案很简单:1.资源问题.2.最大价值问题.3.性价比问题.这些就是背包的初步动机.而且在日常生活中包括竞赛中,这些可以说是最常见的问题了.
而且背包的所有问题几乎都源自于他的基本方程的变种,深刻理解这个方程不但对于背包本身而且对于理解其他的经典 DP 问题都是必要的.特别是对于解决最值问题是至关重要的.

3.3: TEXT Eulerian Tour

这篇<欧拉路径问题>的出现也预示着图论的难度上升了一个高度,现在已经进阶到中等图论了,欧拉路径问题源于经典的哥尼斯堡七桥问题.也就是著名的一笔画问题.所谓欧拉路径就是在连通图中有一条路径:每个边都被访问且仅访问一次.若起点与终点不同,那么就是欧拉路径,若起点与终点相同,那么就是欧拉回路.判断是否存在欧拉路径只要判断基图的度即可.若每个点的度均为偶数那么存在欧拉回路,如果有且仅有两个点的度为奇数其他都为偶数那么存在欧拉路径.在强连通图中判定略有不同:如果所有点的出度等于入度那么有欧拉回路.如果有且仅有一个点的出度比入度大 1,并且有且仅有一个点的入度比出度大 1,那么存在欧拉路径.
欧拉路径的难点不在于寻找(当然有向图中的寻找要困难得多),而在于建立模型.其实**对于问题的建模程度源于对问题的理解程度**.理解的好自然建出的模型就适用于问题.欧拉路径解决的主要是寻找路径问题.这种路径是广义的.只要问题能得到相应的转化就能通过寻找欧拉路径的办法解决.

3.4: TEXT Computational Geometry

计算几何问题的应用十分广泛,可以说人工智能,机器人等等诸多领域都有其用武之地,USACO 的这篇课文极其概括的说明了计算几何问题的两个重要内容:叉积与

反三角函数.或许真的是这样,正可谓一招鲜吃遍天.叉积是计算几何的生命,有了向量计算几何就要比一般的解析方便多了,我们也可以进行更广阔的扩展.
比如根据向量的特性,可以轻松表述一条直线,结合叉积与点积,我们可以方便的计算面积与体积,计算点到直线的距离,判断两条线段是否相交,等类似的,还有判断点是否在形内,只要找到图形的重心,与那个点分别做出点到图形端点的向量计算叉积即可.还有比较实用的判断图形的凹凸性等.而反正切则大多用来计算角度,因为正切的好处是不用判断除数为 0 的情况,这样可以替代反正弦和反余弦.
文章还介绍了 3 个几何方法:蒙特卡罗法和分割法以及转化为图,事实上计算几何的难点是考虑问题的**全面性**.只要把问题考虑全面,细心分析,认真解答,计算几何并不难.

第四章:

4.1: TEXT Optimization Techniques

这一节的习题是 USACO 的搜索**剪枝思想**的一个最高峰，之所以说是思想而不是技术，是因思想比单纯的技术更体现他的“思”与“想”的成分。特别是好的思考方式可以给程序带来相当不错的时间效率，而且好的剪枝也使得程序更加有条理。
我个人认为一个完善的剪枝系统应该包含有如下几个方面：(针对 DFS)

1. 搜索的**对象**的选择：这是第一步，但是这也算是剪枝吗？其实对于一般的明显的问题，搜索对象都是显而易见的。所谓搜索对象的选择是针对不容易看出搜索要素的问题，这时选择好的对象，或者是从**侧面**巧妙地抉择对象，程序就会达到惊人的简洁和高效，这是第一步。
2. 搜索**顺序**的选择：同样是极其重要的部分。有时给凌乱的数据进行有序化，使得搜索能尽快的剪枝，意味着它是剪枝的一个铺垫。
3. **不重复**做过的，进行非重复状态决策。
4. 搜索的**可行性**：这是两大重要剪枝要素之一，对于不可行的解直接剪掉。
5. 搜索的**最优性**：另外一个重要剪枝要素，直接考查最优性，如果中途已经远离当前的“优秀” 答案了，那么何以毫不犹豫的剪掉。
6. 其他的辅助策略：像是 ID 可以用二分答案的手段,而 BFS 一般少有明显的剪枝，可以考虑 A*,或者结合两者的优点:IDA*.

其实搜索剪枝有时不那么明显，所以应该在上述铺垫的基础上进行一些其他的优化性**构造**。从而产生良好的效果。比如结果本身的正确性等等。

4.2: TEXT "Network Flow" Algorithms

这节介绍了一类相当重要的算法，可以说当学习搜索和 DP 初步的时候视为算法入门，这样来参加联赛等级的竞赛。而当学习了平衡树和网络时就是进阶到全国的要求了。这样说也许可以体现他的重要性。
何为网络流？网络流是一类特殊的线性规划问题，可以借助图论的手段方便的解决这类问题。先来复习一下网络流：首先给定一个有向带权图：G=(V,E,C),包含源点和汇点 S,T.我们要求求出一个从源点到汇点的流量，并且满足每条边上的流量都不超过容量 C’ 。

1. 定义：
   f*表示流量，对于一个节点到另一个节点的流量记为 f(i,j)，容量记为 c(i,j)。满足 0<f(i,j)<=c(i,j)的叫做**非零流弧**，满足 0<=f(i,j)<c(i,j)的叫做**非饱和弧**。满足 f(i,j)=0 的叫做**零流弧**，满足 f(i,j)=c(i,j)的叫做**饱和弧**。
   可行流：满足 0<=f(i,j)<=c(i,j)的一个从源到汇的流即为可行流。
   最大流：满足上述条件的一个流量最大的流。

2. 平衡条件：

$$\sum_{((vi,vj)\in E)} f(i,j) - \sum_{((vj,vi)\in E)} f(j,i) = 0(i,j \neq S,T)$$

$$\sum_{((vs,vj)\in E)} f(s,j) - \sum_{((vj,vs)\in E)} f(j,s) = F(s,t)$$

$$\sum_{((vt,vj)\in E)} f(t,j) - \sum_{((vj,vt)\in E)} f(j,t) = -F(s,t)$$

3. 增广路：
   是指存在一条从 S 到 T 的路径满足约束的同时，剩余流量**不为 0** 的路径。我们定义：凡是与增广路上的方向一致的弧记为**前向弧μ^+**，**反向弧**记为**μ^-**。

4. **割集与割量**：
   是指如果将网络中的节点划分为两个互不相交的集合，其中 S∈V,T∈ $\bar{V}$。那么存在一组边集使得从 S 到 T 必须经过此集合中的一条满足：vi∈V,vj∈ $\bar{V}$,如此则有**割量** $C(V,\bar{V})=\sum_{vi\in V,vj\in\bar{V},} c(i,j)$,所以显然有 $f(i,j)<= C(V,\bar{V})$.其中就包含了流量 $f(i,j)=c(i,j)(c(i,j)\in C(V,\bar{V}))$,这样的流量必然是最大流，同样这样的一组流量必然是最小割。即**最大流最小割定理**。

5. 网络流的算法：
   增广路法基本的主体思想是利用贪心的迭代,不断寻找增广路，一旦找不到增广路，则得到最大流。

   证明：设存在关于最大流的增广路μ，令$\theta = \min(\min(c(i,j) - f(i,j)) \in \mu^+, \min(f(i,j)) \in \mu^-)$.由定义可知：f'(i,j)=f(i,j)+θ (vi,vj∈ μ^+),如果 f'(i,j)大于当前流 ，则与最大流的假设矛盾。充分性得证。
   假设不存在关于最大流的增广路：
   令 S∈ V_1^*,若 Vi∈ V_1^*,且 f(i,j)<c(i,j)则 Vj∈ V_1^*，若 Vi∈ V_1^*,且 f(j,i)>0 则 Vj∈ V_1^*。
   因为不存在增广路，必然有 T 不属于V_1^*.令$\bar{V}$=V\V_1^*,

   则有$f(i,j) = \begin{cases} c(i,j)\ (vi,vj)\in(V_1^*,\bar{v}) \\ 0\ (vi,vj)\in(\bar{v},V_1^*) \end{cases}$ 所以流量 $F=c(V_1^*,\bar{v})$.

   最大流最小割定理与必要性得证。

- Ford-Fulkerson:
  基本标号法：
  1. 标号：分为标号和检查的点，标为(从哪里得到，当前流量)。
  2. 调整：反向追踪，更改流量。
- Dinic:
  一类基于分层图思想的多路增广的算法，每次要进行 BFS 进行层次标号，一旦可以抵达 T 则有增广路。使用 DFS 更新时，只更新相邻的层次，这样就会较 F-F 法大大提高效率，而实现时又不必记录当前的流量，只是将更新的正向边的容量改小，反向边改大，因此又称为**阻塞流**法，差值即为最终结果。
- 其他常用算法：
  1. 最短路径增值：
     1.E-K 法：直接使用改进过的 Dijkstra 进行增广，比 F-F 要高效。
     2.MPLA：在 E-K 的基础上用分层图优化，并 BFS 扩展，更高效。
     3.ISAP：在 MPLA 的基础上使用 GAP,反向弧等等再度优化，时效直逼 HLPP.
  2. 预流-推进：当今最快的成法(无其他优化的裸的模型).
  3. 单纯型法：使用线性规划解决。
- 算法总结：
  性价比最高的是 Dinic。Dinic 如果使用人工栈，则会达到惊人的效果。
  最具推广的的：ISAP,优化后性能极佳。
  最强大的：HLPP。

6. 扩展：
   节点限定流量：拆点法。
   二分图匹配以及多源多汇的网络流：增加超级源和汇。
   无向图流：每条边拆成两个反向的。

以上仅仅是最抽象的总结，而且不包含费用流和带权的二分图，网络模型就像搜索与 DP 一样，用途很广泛。重要的问题是如何建立适当的模型，从而达到游刃有余的处理效果。

4.3: TEXT Big Numbers

高精度运算是我小学时就会的一种处理技术，简而言之就是模拟人的运算方式去计算数字，并保存在数组中。对于多乘多的数字和除法要尤为注意，高精度运算的要点主要在实现。推荐写成运算符重载的 Class。

第五章:

5.1: TEXT 2D Convex Hull

二维凸包是一种很有用的，很具实际意义的问题模型。问题是：对于一个散列的点，如何找到面积最小，或者是周长最小的凸多边形将他围起来。这样我们一般使用 Graham 算法。就是首先对于所有顶点按照 y 为第一关键字，x 为第二关键字的顺序排序。然后先加入第一个顶点，并且依次按照顺序入栈，凡是遇到不满

足的（大于 180 度）就退栈，这样依次进行直到再次遇到顶点 1，就找到了凸包，时间复杂度 O(NlogN).
对于 Graham 算法来说，核心思想是在计算几何中常用的**扫描方法**。常用于坐标上的点，以某个轴为第一关键字，然后离散的扫描经过的所有点，并进行处理，不只是凸包，还包括其他的计算几何问题，可以使用这种方法简便的处理或整理后再用其他算法更方便的解决。

5.3: TEXT Heuristic Search

启发式搜索，这是 USACO 最后一篇课文。本文简要介绍了启发式搜索的一些应用，比如最佳优先搜索其实是贪心的 DFS，每次都先搜索当前最优的点，如果使用得当可以适当的提高程序速度。它配合启发式剪枝效果会更好，启发式剪枝是假如我们有个估价函数，而且现在已经有一个解价值是 C，而当前的价值是 A，利用估价函数计算出的价值是 B。如果 C > A + B 我们就不必要继续搜索可以进行剪枝了。
还有一种很常用的搜索 A*，这是一种带有估价函数的 BFS。一般在图中寻找一个最短路径使用这种算法速度表现会不错。关键就是如何**启发**，已及评定一个启发的好坏。我认为，设计估价函数如果表现的好说明能够抓住问题的本质，比如 A*的经典问题八数码问题，估价函数是所有数字与最终状态的曼哈顿距离。考虑到我们每一步都是移动数字，这样可定改变他当前距离最终态的曼哈顿距离，所以说这个估价函数很切合题意。但是随着问题越来越复杂使用诸如启发式剪枝时也许不能找到最优解，如果是 NP-C 问题也许只需要一个近似解，那么启发式近似搜索就很合适，一方面我们很难找到最优解或者很接近最优解就行了，另一方面启发式搜索速度要快些。
这样来看，启发式搜索的应用方面也就很明显了，而且难点是设计估价函数，如果能够理清问题本质，估价函数也是能够找到大致方向的。