**算法导论复习——常见算法思想**

**PPT2-1：分治算法**

1. **堆排序（选择类排序，不稳定）**

（1）初始化操作：将R[1..n]构造为初始堆；

（2）每一趟排序的基本操作：将当前无序区的堆顶记录R[1]和该区间的最后一个记录交换，然后将新的无序区调整为堆（亦称重建堆）。

时间复杂度是：O（nlgn）

1. **归并排序Merge sort**

归并[排序](http://baike.baidu.com/view/58783.htm)采用分治法思想的稳定的排序。算法思想是先使每个子序列有序，再使子序列段间有序。

第一步：申请空间，使其大小为两个已经[排序](http://baike.baidu.com/view/58783.htm)序列之和，该空间用来存放合并后的序列

第二步：设定两个[指针](http://baike.baidu.com/view/159417.htm)，最初位置分别为两个已经[排序](http://baike.baidu.com/view/58783.htm)序列的起始位置

第三步：比较两个[指针](http://baike.baidu.com/view/159417.htm)所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置

重复步骤3直到某一指针超出序列尾，将另一序列剩下的所有元素直接复制到合并序列尾。

时间复杂度是：O（nlgn）

1. **快速排序（交换类排序，不稳定）**

（1）通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据

都比另外一部分的所有数据都要小；

（2）然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以[递归](http://baike.baidu.com/view/96473.htm)进行，以此达到整个数据变成有序序列。

时间复杂度是：O（nlgn）。

1. **分治法实现大数相乘Large Integers**

将a，b写成前一半数字和后一半数字相加的形式，例如若a = 5423678，那么a1 = 542，a0 = 3678（注意若不是偶数截取较小一半）

这样a和b相乘就可以写为：a \* b = {  a1 \* 10^(n1/2)   +  a0  }   \*  {  b1 \* 10^(n2/2)  +  b0 }

展开后整理得：

 a  \*  b  =  a1\*b1 \* 10^[ (n1+n2)/2 ]   +  a1\*b0 \* 10^(n1/2)   +   a0\*b1 \* 10^(n2/2)  + a0\*b0  四项

这样就很容易递归的来求a \* b，如果你嫌分解后的数还太大，就可以继续分解。（你可以自己规定在何时结束递归）

时间复杂度是：O（nlgn）

参考资料见：

http://wenku.baidu.com/link?url=kkSSG6aAlxeI7qlflvPWBrWPbpAeNtoohJvwk9ARQ3TwTkIx3\_Fiwlt-5HLhrNUxg1C-i0Tus9oo\_cCLiINr2CHiN1mgsidH3gX5rsY6Nuu

1. **分治法求最近点对 closet-pair**

步骤一：找一条中垂线m(坐位S集合x坐标的中位数)把n个元素分成左右两部分元素，步骤二：递归查找两边点集的的最短距离d1,d2，然后取两者中的最小者记为d；

步骤三：在中线两边分别取d的距离，记录该距离范围内点的个数，中线左边有L个元素，右边有R个元素，分别将两边的点按y坐标升序排列，在左边集合中每一个点，找右边集合的点，找到与之距离小于d的点，更新最短距离，直到循环结束，即可求出最短距离。

时间复杂度是：O（nlgn）

1. **格雷厄姆Graham扫描法求凸包 convex hull**

基本思想：通过设置一个关于候选点的堆栈s来解决凸包问题。

操作：输入集合Q中的每一个点都被压入栈一次，非CH（Q）(表示Q的凸包)中的顶点的点最终将被弹出堆栈，当算法终止时，堆栈S中仅包含CH(Q)中的顶点，其顺序为个各顶点在边界上出现的逆时针方向排列的顺序。

具体步骤如下：

（1）设P0 是Q 中Y 坐标最小的点，如果有多个这样的点则取最左边的点作为P0；

（2）设<P1,P2,……,Pm>是Q 中剩余的点，对其按逆时针方向相对P0 的极角进行排序，如果有数个点有相同的极角，则去掉其余的点，只留下一个与P0 距离最远的那个点；

（3）PUSH(p0 , S)

（4）PUSH(p1 , S)

（6）PUSH(p3 , S)

（7）for i ← 3 to m

{

while 由点NEXT-TOP-TOP（S），TOP（S）和Pi 所形成的角形成一次非左转

do POP(S)

PUSH(pi , S)

}

（8）return S

时间复杂度是：O（nlgn）

注：Jarvis步进法更加容易理解，见算法导论中文版P609。

1. **D&C for 2D maxima Finding Problem**

（1）如果点集S只包含一个点，这直接输出该点。否则找一条中垂线m(作为8S集合x坐

标的中位数)把n个元素分成左右两部分个数相等的元素，分别为Sl和Sr。

（2）递归查找两边点集的的最大点；

（3）将获得的所有最大点按y值排序，并查找出Sl中y值小于Sr中最大点的y值的最大点，并将其排除；

（4）最终获得所有的点集S的所有maximal point。

时间复杂度是：O（nlgn）

Homework：Voronoi Diagram

**PPT2-2：**

1. **Depth First Search(DFS)**

深度优先搜索算法也叫回溯法，其基本策略是：尽可能深的深的搜索某一分支。从源节

点开始发现有一节点v，如果v还有未探测到的边，就沿此边继续探测下去，当节点v的所有边都被探测过，搜索过程将回溯到最初发现v点的源节点。这一过程一直进行到已发现从源节点可达的所有节点为止。这显然是一个递归过程。

**伪代码（使用栈实现），递归实现：**

long DFS(图g，结点v0)

{

//图深度优先遍历递归算法。从结点v0出发，深度优先遍历图g，返回访问到的结点总数

int nNodes; //寄存访问到的结点数目

访问v0;

为v0置已访问标志;

nNodes=1;

求出v0的第1个可达邻接点v;

while (v存在)

{

if (v未被访问过) nNodes=nNodes+DFS(g,v);

求出v0的下个可达邻接点v;

}

return nNodes;

}

**非递归实现（借用栈实现）：**

long DFS\_NR(图g，结点v0)

{ //图深度优先遍历非递归算法。从结点v0出发，深度优先遍历图g，返回访问到的结点总数

int nNodes; //寄存访问到的结点数目

访问v0;

为v0置已访问标志;

v0进栈S；

nNodes=1;

//求出v0的第1个可达邻接点v;

while (栈S不空)

{

v = 栈S顶部元素；

求v的下个未访问过的出点i；

访问i;

为i置已访问标志;

i进栈S；

nNodes++;

if (v已无未被访问过的出点) 出栈；

}

return nNodes;

}

1. **BFS（Breadth First Search）宽度优先搜索**

从初始结点开始，应用产生式生成第一层结点，检查目标结点是否在这些后继结点中，

若没有，再用产生式规则将所有第一层的结点逐一扩展，得到第二层结点，并逐一检查第二层结点中是否包含目标结点。若没有，再用算符逐一扩展第二层所有结点……，如此依次扩展，直到发现目标结点为止。

**伪代码（使用队列实现）：**

long BFS(图g, 结点v0）

{

//在图g中从v0出发按广度优先遍历方式遍历g，返回遍历到的结点数目

long nNodes=0;

初始化队Q;

v0入Q;

while (Q不空)

{

队Q头元素出队并送v;

访问v;

置v0为已访问标志;

nNodes++; //对已访问元素计数

求出v的第一个可达邻接点w;

while (w存在)

{

if (w尚未被访问过)

{

w入Q;

}

求v的下个可达邻接点w;

}

return nNodes;

}

}

两个算法的时间复杂度都是O（V+E）。

1. **拓扑排序（Topological Sorting）**

一个有向图能被拓扑排序的充要条件就是它是一个有向无环图(DAG：Directed Acyclic Graph)。

拓扑排序就是一个广度优先搜索。

拓扑排序的方法如下:

　　(1)从有向图中选择一个没有前驱(即入度为0)的顶点并且输出它.

　　(2)从网中删去该顶点,并且删去从该顶点发出的全部有向边.

　　(3)重复上述两步,直到剩余的网中不再存在没有前趋的顶点为止.

参考资料：http://blog.chinaunix.net/uid-26602509-id-3196077.html

1. **字典序法生成全排列（permutation）**

设P是1～n的一个全排列:p=p1p2......pn=p1p2......pj-1pjpj+1......pk-1pkpk+1......pn

1.从右向左找，找到第一个比下一个元素还小的地方，记下位置，标注为左元素。  
2.从右向左找，找到第一个比左元素大的元素，也是左元素右边比左元素大的数中最小的数，记下位置，标注为右元素。  
3.交换左元素和右元素。  
4.不管现在左元素位置上放的是谁，将左元素右边的序列逆序。  
5.这样就得到了一个新数了。  
6.可以继续重复1-5，来继续得到下一个排列。  
7.如果再也找不到一个比下一个元素还小的地方，那么意味着这个序列已经降序了，排列完成了，那就结束吧。

时间复杂度：O(n!)

参考资料：<http://blog.csdn.net/cpfeed/article/details/7376132>

1. **关节点（Articulation Point）**

见homework。

1. **约瑟夫问题（Josephus Problem）**

设有n个人围坐在一个圆桌周围，现从第s个人开始报数，数到第m的人出列，然后从出列的下一个人重新开始报数，数到第m的人又出列，…，如此反复直到所有的人全部出列为止。Josephus问题是：对于任意给定的n,s和m，求出按出列次序得到的n个人员的序列。

解决思想：循环链表表示

1 建立循环单链表

2 寻找第s个结点，输出并删除第m个节点。

1. **二查搜索树BST（Binary Search Tree）**

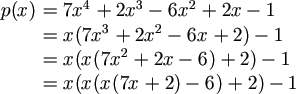
**PART2-3:（Transform &Conquer）**

1. **霍纳法则（Horner Rule）**

霍纳法则出现的原因：为多项式求值提供了一个高效的方法。用来简化朴素多项式的求值，在中国叫秦九韶算法。

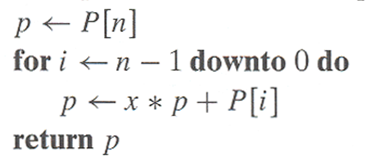
朴素多项式可以简化为：  
horners_method0

举个例子，多项式变换如下：



pseudo code:

其中P[n]是多项式的系数组成的数组，



1. **平衡二叉树AVL(Balanced Binary Tree)**
2. **红黑树R-B树**

参考链接：http://blog.csdn.net/eric491179912/article/details/6179908

1. **2-3树**

每个节点最多包含两个关键子，所有叶子节点在树的同一层，因此树总是高度平衡的。

1. **堆Heap**
2. **树堆Treap**

见homework，有可能考代码！

1. **优先队列Priority Queue**

重点:优先级队列，是要看优先级的，谁的优先级更高，谁就先得到权限。不分排队的顺序! 利用堆可以实现优先队列！

**21. 迷宫问题Maze Problem**

使用DFS或者BFS遍历搜索出口！

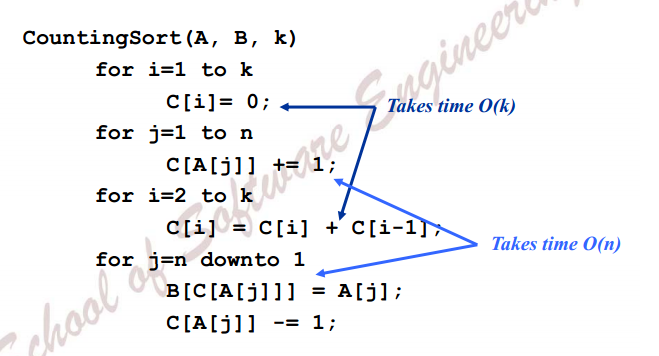
**PART2-4 时空权衡**

1. **计数排序Counting Sorting**

计数排序的基本思想是对于给定的输入序列中的每一个元素x，确定该序列中值小于x的元素的个数。一旦有了这个信息，就可以将x直接存放到最终的输出序列的正确位置上。

例如，如果输入序列中只有17个元素的值小于x的值，则x可以直接存放在输出序列的第18个位置上。当然，如果有多个元素具有相同的值时，我们不能将这些元素放在输出序列的同一个位置上，因此，上述方案还要作适当的修改。

**伪代码：A待排序数组，B排序后的数组，k待排序中最大的数**

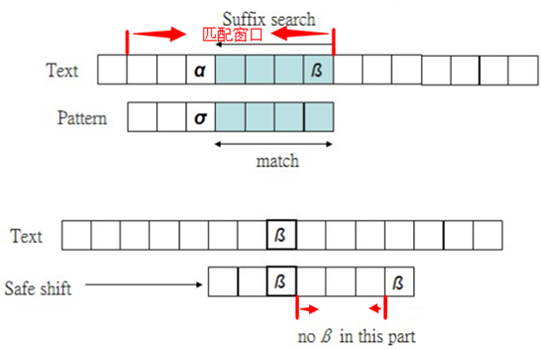


1. **字符串匹配算法Horspool Algorithms**

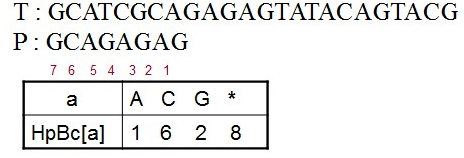
Horspool算法是对Boyer-Moore算法的简化。其基本思想（后缀匹配法）：

模式串是需要匹配的串，主串是在其中查找模式串的字符串，匹配窗口是模式串从左往右移动，在主串中需要比对的字符形成的窗口。模式串是从左往右移动，字符比对是从匹配窗口的右边往左。

horspool算法将主串中匹配窗口的最后一个字符跟模式串中的最后一个字符比较。如果相等，继续从后向前对主串和模式串进行比较，直到完全相等或者在某个字符处不匹配为止（如下图中的α与σ失配）。如果不匹配，则根据主串匹配窗口中的最后一个字符β在模式串中的从右往左下一个出现位置，将窗口向右移动与之对齐。



为了实现模式串的移动，必须先记录模式串中每一个字符（不包括最后一个字符）在模式中距离最右边的距离，如若出现相同字符的距离不同，取最短的距离：



**时间复杂度：**假设主串的长度为n,模式串的长度为m，那么Horspool算法最坏情况下的时间复杂度是O(mn)，但平均情况下它的时间复杂度是O(n)。

参考资料：<http://dsqiu.iteye.com/blog/1700312>

1. **字符串匹配算法BM：Boyer-Moore**

算法思想（后缀匹配法）：

Boyer-Moore算法是一种基于后缀匹配的模式串匹配算法，后缀匹配就是模式串从右到左开始比较，但模式串的移动还是从左到右的。字符串匹配的关键就是模式串的如何移动才是最高效的，Boyer-Moore为了做到这点定义了两个规则：坏字符规则和好后缀规则，具体的移动规则见参考资料链接。

Boyer-Moore算法步骤

1.对模式子串进行预处理

Boyer-Moore算法实现必须对模式串进行预处理，得到坏字符规则和好后缀规则移动的映射表，下面代码中MakeSkip是建立坏字符规则移动的映射表，MakeShift是建立好后缀规则的移动映射表。

MakeSkip是构造数组skip[]，skip[k]表示字符k距离模式串末尾的距离。

MakeShfit是构造数组shfit[]，shfit[k]表示模式串的以k为边界的后缀子串的最靠近的模式子串（或最前缀子串）到模式子串末尾的距离，例如：abcab，shfit[3]=3和shfit[2]=3（即都是第一个b到末尾的距离），k=2时，后缀子串为cab，这时只有最长前缀ab，shfit[2]=3。

2.从b\_idx开始查找，得到坏字符和好后缀，得到最大移动距离，移动b\_idx，直至b\_idx到达母串的末尾。

参考资料：<http://dsqiu.iteye.com/blog/1700312>

1. **Hash（去年未考）**
2. **字符串匹配算法KMP算法**

算法思想（前缀匹配法）：

KMP算法是一种高效的前缀匹配算法，在传统蛮力（BF）匹配算法的基础上改进的地方在于每次移动的距离不是1可以是更大，没有进行回溯，BF算法的时间复杂度是O(m\*n)，而KMP算法的时间复杂度是O(m+n)。

每次根据已经匹配的字符数目，通过前缀函数来求得移动的位数。Next[]表示前缀函数，其求法：

P 表示模式串，其中已经匹配的字符串为"ababa",那么这个已匹配的字符串中，满足既是自身真后缀(即不等于自身的后缀)，又是自身最长前缀的字符串为"aba"，我们设这个特殊字串的长度为L，显然，L = 3。q表示已经匹配的长度，故我们要求的移动位数s = q - L = 5 - 3 = 2 ，即next[q]=q-L=2；

若P模式字符串已经匹配的字符串为”abab”，则满足既是自身真后缀又是自身最长前缀的字符串为“ab”，故s=q-L=4-2=2，即next[q]=q-L=2；

若P模式字符串已经匹配的字符串为“aba”，则满足既是自身真后缀又是自身最长前缀的字符串为“a”，故s=q-L=3-1=2，即next[q]=q-L=2；

若P模式字符串已经匹配的字符串为“ab”，则满足既是自身真后缀又是自身最长前缀的字符串为“空”，故s=2-0=2，即next[q]=q-L=2；

以此类推，对模式串进行预处理，可能到前缀函数。

**参考资料：**http://billhoo.blog.51cto.com/2337751/411486/

1. **字符串匹配算法Sunday算法**

算法思想（前缀匹配法）：算法效率Sunday>BM(**Horspool**二者未知)>KMP>BF(Brute Force)

Sunday算法是Daniel M.Sunday于1990年提出的一种比BM算法搜索速度更快的算

法。其核心思想是：在匹配过程中，（模式串并不被要求一定要按从左向右进行比较还是从右向左进行比较，这句话应该是错的），它在发现不匹配时，算法能跳过尽可能多的[字符](http://baike.baidu.com/view/263416.htm)以进行下一步的匹配，从而提高了匹配效率。

sunday算法核心就是看主串string里面与pattern对比过程中第一个不相同的字符在pattern中的位置。

第一种情况如这个字符不在pattern里面，直接跳过，这时候就变成这样：

string： d g j h o l p k j h s i j w a v n m..........

pattern: j h h i

第二种情况：这个字符在pattern里面，我们就把pattern最右边的那个匹配的字符与其对齐：

string： d g j h o l p k j h s i j w a v n m..........

pattern: j h h i

就这样一直循环这个步骤，一直到找到或则找不到，返回结果。

参考资料：http://blog.csdn.net/cogbee/article/details/8443249

1. **动态Hash算法**

算法思想:

在实际的应用中，当hash表较小，元素个数不多时，采用以上方法完全可以应付。但是，一旦元素较多，或数据存在一定的偏斜性(数据集中分布在某个桶上) 时，以上方法不足以解决这一问题。我们引入一种称之为动态散列的方法：在hash表的元素增长的同时，动态的调整hash桶的数目。

动态hash不需要对hash表中所有元素进行再次插入操作(重组)，而是在原来基础上，进行动态的桶扩展。有多种方法可以实现：多hash表、可扩展的动态散列和线性散列。

参考资料：<http://blog.sina.com.cn/s/blog_5e4516af01019frj.html>

1. **B+树**

见homework。

**PART2-5 Greedy Technique**

1. **Prim算法for最小生成树MST（Minimum Spanning Tree）**

算法思想：

1).输入：一个加权连通图，其中顶点集合为V，边集合为E；

2).初始化：Vnew = {x}，其中x为集合V中的任一节点（起始点），Enew = {},为空；

3).重复下列操作，直到Vnew = V：

a.在集合E中选取权值最小的边<u, v>，其中u为集合Vnew中的元素，而v不在Vnew集合当中，并且v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b.将v加入集合Vnew中，将<u, v>边加入集合Enew中；

4).输出：使用集合Vnew和Enew来描述所得到的最小生成树。

伪代码如下：

1. 初始化两个辅助数组lowcost和adjvex；

2. 输出顶点v0，将顶点v0加入集合U中；

3. 重复执行下列操作n-1次 ，知道U=V

   3.1 在lowcost中选取最短边，取adjvex中对应的顶点序号k；

3.2 输出顶点k和对应的权值；

3.3 将顶点k加入集合U中；

     3.4 调整数组lowcost和adjvex

**时间复杂度：**O(ElgV )

**参考资料:** http://www.cnblogs.com/biyeymyhjob/archive/2012/07/30/2615542.html

代码见homework。

1. **Kruskal算法for MST**

**算法思想：**

克鲁斯卡尔Kruskal算法的时间复杂度由排序算法决定，若采用快排则时间复杂度为

O(N log N)。是最小生成树算法较为简单的算法。

求加权[连通图](http://baike.baidu.com/view/3148644.htm)的最小生成树的算法。kruskal[算法](http://baike.baidu.com/view/7420.htm)总共选择n- 1条边，所使用的贪婪准

则是：从剩下的边中选择一条不会产生[环路](http://baike.baidu.com/view/710129.htm)的具有最小耗费的边加入已选择的边的集合中。注意到所选取的边若产生环路则不可能形成一棵生成树。

算法开始需要对连通图中的边的权重进行排序，kruskal算法分e 步，其中e 是网络中边的数目。按耗费递增的顺序来考虑这e 条边，每次考虑一条边。当考虑某条边时，若将其加入到已选边的集合中会出现环路，则将其抛弃，否则，将它选入。

**时间复杂度：**Kruskal算法的时间复杂度由排序算法决定，若采用快排则时间复杂度为O(N log N)。

**参考资料：**

<http://baike.baidu.com/link?url=WlysQZRZhF4K1UVD1Gwans_6mxnJ_BaLY7J8HOC1jRUc_XxJB9DDwlGOBl2S10xtgI6i8GoLbMk94mMZ6-vv3a>

1. **Dijkstra算法for单源最短路径**

Dijkstra(迪杰斯特拉)解决的是带权重的有向图上单源最短路径的问题，即从一个点开始

到所有其他点的最短路。

算法步骤如下：

G={V,E}

单源最短路径——从一个点到其他各个顶点的最短路径。

分析：分成两个集合，S代表已经找到最短路径的顶点，T代表还没找到的。开始S中只有源点V0，然后不断从集合T中取到顶点V0的路径长度最短的顶点u加入S，每加入一个新顶点u，都要对u直接邻接的点更新（如果距离更小的话）。直到所有点加入集合S。

假设存在G=<V,E>，源顶点为V0，U={V0},dist[i]记录V0到i的最短距离，path[i]记录从V0到i路径上的i前面的一个顶点。

1.从V-U中选择使dist[i]值最小的顶点i，将i加入到U中；

2.更新与i直接相邻顶点的dist值。(dist[j]=min{dist[j],dist[i]+matrix[i][j]})

3.直到U=V，停止。

参考资料：

1.百度百科http://baike.baidu.com/view/1712262.htm?fromtitle=Dijkstra算法&fromid=215612&type=syn

2. <http://www.cnblogs.com/dolphin0520/archive/2011/08/26/2155202.html>

1. **Huffman算法for Coding Problem**

定义：它是由n个带权叶子结点构成的所有二叉树中带权路径长度最短的二叉树。为哈弗曼树，又叫最优二叉树。

1.初始化： 根据给定的n个权值{w1,w2,…wn}构成n棵二叉树的集合F={T1,T2,..,Tn}，其中每棵二叉树Ti中只有一个带权wi的根结点，左右子树均空。

2. 找最小树：在F中选择两棵根结点权值最小的树作为左右子树构造一棵新的二叉树，且至新的二叉树的根结点的权值为其左右子树上根结点的权值之和。

3. 删除与加入：在F中删除这两棵树，并将新的二叉树加入F中。

4. 判断：重复前两步（2和3），直到F中只含有一棵树为止。该树即为哈夫曼树

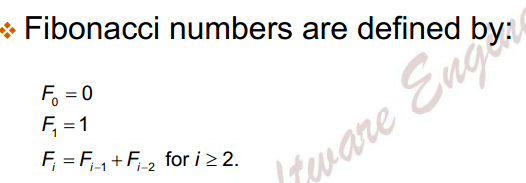
参考资料：百度百科：http://baike.baidu.com/view/1679065.htm

1. Offline Cache调度算法

见homework。

**PART2-6 Dynamic Programming**

1. **斐波那契数Fibonacci number**



即后一个数是前两个数之和。如：0、1、1、2、3、5、8、13、21….

1. **Cost-Based Optimization in DBMS**
2. **最优二叉搜索树:Optimal Binary Search Trees(OBST)**

问题描述：

n个键{a1,a2,a3......an},其相应的查找概率为{p1,p2,p3......pn}。构成最优BST,表示为T1n ，求这棵树的平均查找次数C[1, n]（耗费最低)。换言之，如何构造这棵最优BST，使得

C[1, n] 最小。

使用动态规划的思想生成最优二叉搜索树。比较难理解！

（看不懂）

1. **背包问题Knapsack Problem**

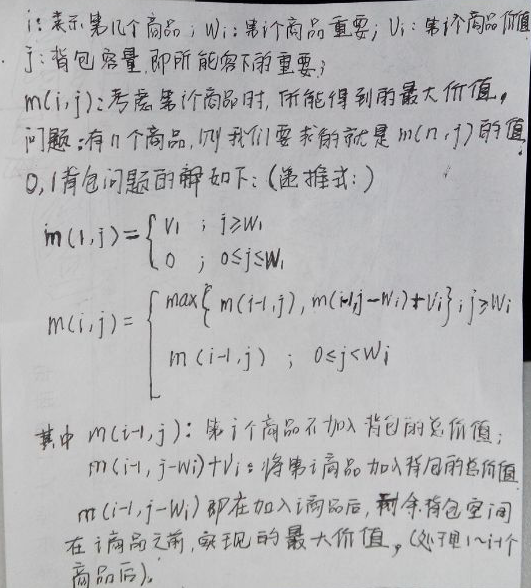
背包问题(Knapsack problem)是一种组合优化的[NP完全问题](http://baike.baidu.com/view/272414.htm)，是Non-deterministic Polynomial的问题，即[多项式](http://baike.baidu.com/view/613580.htm)复杂程度的非确定性问题。

有N件物品和一个容量为V的背包。第i件物品的重量是c[i]，价值是w[i]。求解将哪些物品装入背包可使这些物品的重量总和不超过背包容量，且价值总和最大。

今天主要说的是0、1背包问题，即每样东西最多只能取一次，解法是动态规划。

动态规划的方法：

用动态规划解问题首先要有效的找出子问题，可以通过这个子问题得推得原问题的解，通常子问题的实质是和原问题相同的，只是规模上的缩小，也就是说子问题和原问题可以有相同的表示形式，问题可通过不断的缩小规模（一般都会有一个界限）能找到子问题的解。



**参考资料：**<http://blog.csdn.net/dapengbusi/article/details/7463968>

1. **Warshall’s Algorithm**
2. **Floyd’s Algorithm: All pairs shortest paths**
3. **装配线调度算法Assembly Line Scheduling (ALS)**

见homework。

**PART2-7：Tree Searching Strategies**

1. **爬山算法Hill Climbing**

爬山算法是一种局部择优的方法，采用启发式方法，是对深度优先搜索的一种改进，它利用反馈信息帮助生成解的决策。 属于人工智能算法的一种。

算法思想：从当前的节点开始，和周围的邻居节点的值进行比较。 如果当前节点是最大的，那么返回当前节点，作为最大值(既山峰最高点)；反之就用最高的邻居节点来，替换当前节点，从而实现向山峰的高处攀爬的目的。如此循环直到达到最高点。

1. **The traveling salesperson optimization problem**

旅行商优化问题，It is NP-complete。见算法导论中文版第三版P644.

1. **A\*寻路算法**

**算法思想：**

1. **Akari Puzzle**

见homework。

**PART2-8（Iterative improvement，线性规划）**

1. **单纯形法simplex method**

是求解LP（Linear Problem）线性规划问题的通用方法。它的理论根据是：线性规划问题的可行域是 n维向量空间Rn中的多面凸集，其最优值如果存在必在该凸集的某顶点处达到。顶点所对应的可行解称为基本可行解。

单纯形法的基本思想是：先找出一个基本可行解，对它进行鉴别，看是否是最优解；若不是，则按照一定法则转换到另一改进的基本可行解，再鉴别；若仍不是，则再转换，按此重复进行。因基本可行解的个数有限，故经有限次转换必能得出问题的最优解。如果问题无最优解也可用此法判别。

1. **Ford-Fulkerson algorithm for maximum flow problem**

Ford-Fulkerson算法是一种迭代算法，首先对图中所有顶点对的流大小清零，此时的网络流大小也为0。在每次迭代中，通过寻找一条“增广路径”(augument path)来增加流的值。增广路径可以看作是源点s到汇点t的一条路径，并且沿着这条路径可以增加更多的流。迭代直至无法再找到增广路径位置，此时必然从源点到汇点的所有路径中都至少有一条边的满边（即边的流的大小等于边的容量大小）。

其伪代码如下：

Ford-Fulkerson 方法 (G,s,t)

1 将各边上流量 f 初始化为 0

2 while 存在一条增广路径 p

3 do 沿路径 p 增广流量 f

4 return f

参考资料：<http://www.cnblogs.com/luweiseu/archive/2012/07/14/2591573.html>,

<http://blog.csdn.net/fengchaokobe/article/details/7584781>

1. **maximum matching of graph vertices**
2. **求婚-拒绝算法:Gale-Shapley algorithm for the stable marriage problem**
3. **local search heuristics启发式**

**PART2-9**

**伪代码：**

1. 最小生成树：Prim
2. 深度优先
3. 广度优先

算法描述：A\*算法，字典排序，星期天算法

编程实现：机器人捡硬币