数算预习纲要

By wln, 2021.12

▶

数据结构

- 二元关系：设R是集合M上的一个二元关系：
  - 二元关系R的自反性：$\forall a \in M$, 有$(a,a) \in R$
  - 反自反性：$(a,a) \notin R$
  - 对称性：$(a,b) \in R \rightarrow (b,a) \in R$
  - 反对称性：$(a,b) \in R \rightarrow (b,a) \notin R$
  - 传递性：$(a,b) \in R, (b,c) \in R \rightarrow (a,c) \in R$
  - 等价关系：若非空集合上的二元关系R满足自反性、对称性和传递性。

    若R是M上的等价关系，且$a,b \in M$，若$(a,b) \in R$则称a等价于b（a~b）
  - 偏序关系：二元关系R满足自反性、反对称性和传递性
  - 拟序关系（严格偏序关系）：反自反性+反对称性+传递性
  - 全序关系：若二元关系R是M上一个偏序关系，如果集合M上任意两个元素a，b都可比，或者有aRb成立，或者有bRa成立，则称R是集合M上的全序关系（全序关系一定是偏序关系）

- 线性表中查找一个特定元素的复杂度：$O(L)$，其中L是表长
- 在顺序表中插入或删除一个元素，平均需要移动表中大约一半($\frac{1}{2}n$)的数据元素
- 顺序栈：进栈出栈、（基本数据类型下）创建销毁：$O(1)$
- 链式栈进栈、出栈：$O(1)$，创建和销毁链式栈：$O(n)$
- 递归消除：尾递归（递归语句只有一个且处于函数最后），单项递归（多个递归语句，但都处于最后，且互相参数之间无关）

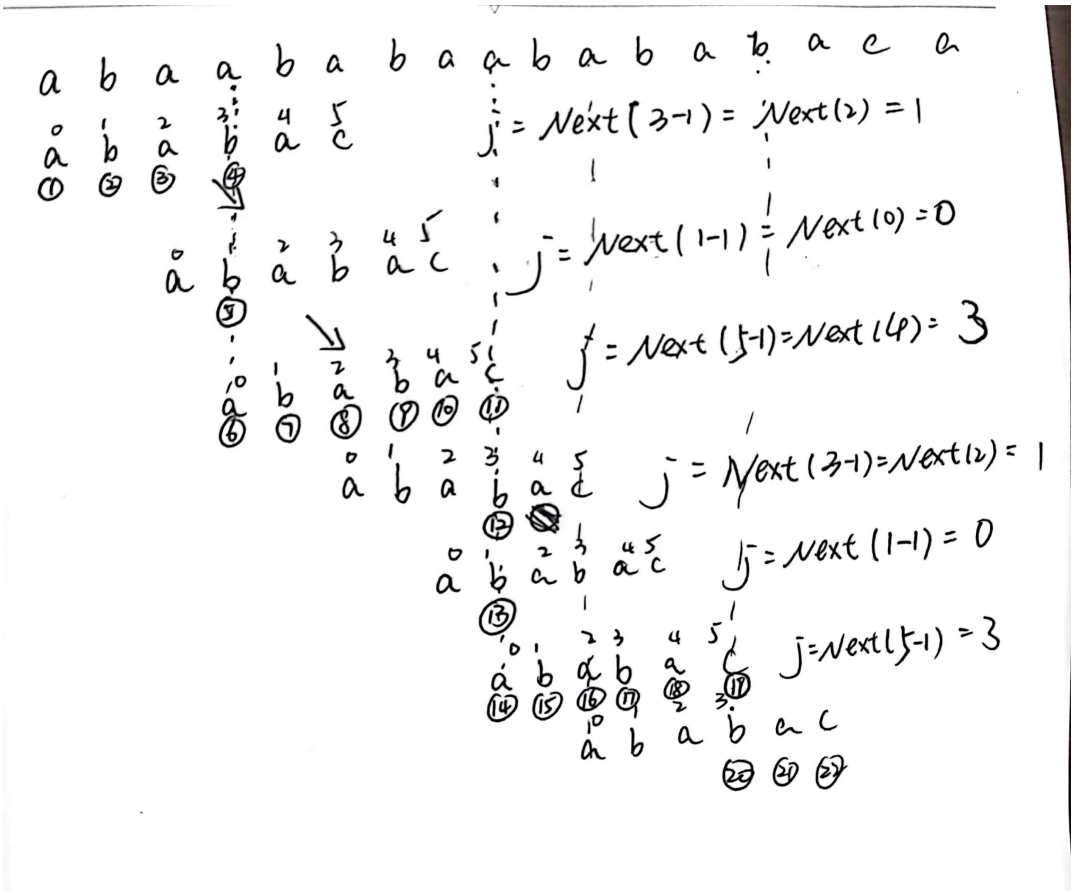
- 优先队列：
  - 基于无序表：入队/修改优先级$O(1)$，访问最高级元素/出队$O(N)$，N为队列长度
  - 基于有序表：入队/修改优先级$O(N)$，出队$O(N)$（保持线性表结构），访问最高级元素$O(1)$
  - 无序链表、有序链表：同上，唯一区别是有序链表出队$O(1)$

- 字符串匹配和KMP（目标串长度n，模式串长度m）：
  - 蛮力最坏：$O(n \times m)$

- KMP最坏：$O(n+m)$
- 部分匹配串：最大的一对相等的P[0,...,j]的前缀和P[1,...,j]后缀
- Next函数：Next[j]表示模式串P在j位置的部分匹配串长度
- 如果在模式串P的j位置出现失配，那么模式串中Next[j-1]位置就应该被移动到原来j位置
- eg:

  现有主串：a b a a b a b a a b **a b a b a c** a， 模式串：a b a b a c 。请写出模式串的 next 函数（画出表格即可），以及 KMP 方法进行匹配的过程，并统计出比较的次数。

j	0	1	2	3	4	5
P[j]	a	b	a	b	a	c
Next[j]	0	0	1	2	3	0



  共比较了22次

- 树
  - 结点的度：拥有子树个数。树的度：树中节点的最大度数，这树称为k叉树
  - 树中结点数目等于所有节点度数和+1
  - k叉树第i层最多有k^{i-1}个结点（i从1开始）
  - 真k叉树（结点度数仅有k或0）叶子节点数M，非叶子结点数n，则$M=1+n(k-1)$

    真二叉树叶子节点数为非叶子结点数+1
  - 满二叉树：深度为h且拥有2^h-1个结点。完全二叉树：第1~h-1层结点数都达到最大值，第h层从右到左连续缺若干结点
  - 二叉树遍历：

    （中序+前序或后续 可以还原树）

- 前序：根-左-右
- 中序：左-根-右
- 后序：左-右-根
- 层序：自上而下逐层遍历
- 若二叉树结点数为n，则遍历时间复杂度$O(n)$，空间复杂度最好$O(\log_2 n)$（递归工作栈最大深度与二叉树深度一致），最坏$O(n)$（单支树）
- 非递归遍历：p78

- 二叉搜索树
  - 理想下，查找一个特定元素的复杂度降低到$O(\log L)$，L为表长
  - 二叉搜索树为理想平衡树时，时间复杂度为$O(\log_2 n)$；若为单支树，其时间复杂度为$O(n)$；平均情况为$O(\log_2 n)$。类似地，空间复杂度平均为$O(\log_2 n)$，最坏为$O(n)$
  - 二叉搜索树（BST）或是一棵空树，或是具有以下性质的二叉树：
    - 每个节点有一个关键字
    - 任意节点关键字大于等于该结点左子树所有节点含有的关键字
    - 同时该结点的关键字小于等于右子树所有节点含有的关键字
  - 查找：若当前节点值为item，查找成功；

    若当前节点值大于item，进入左子树查找

    若当前节点值小于item，进入右子树查找

    ▶ eg

- Huffman树与编码
  - 树的路径长度：从树根到该结点的路径长度之和，记为L

    树的带权路径长度：树中所有以叶子结点为起点的带权路径长度之和称为树的带权路径长度，记为：$WPL=\sum_{i=1}^{n}W_iP_i$，其中，n为树中叶子结点的个数，W_i和P_i分别为叶子结点i的权值和根节点到其的路径长度

    ▶ eg
  - Huffman树：最优二叉树，WPL最小的树。直观上看，权值大的叶子离根节点近，权值小的叶子离根节点远。构造Huffman树的方法为Huffman算法

    Huffman树的节点的度只能是0或2

    深度为h的Huffman树，至少含有：$2(h-1)+1=2h-1$ 个节点（除了第一层有一个外，其他层都有且仅有两个节点）

    至多含有：2^h-1个节点（满二叉树）
  - Huffman算法执行过程：

    1. 根据给定的n个权值$\{w_1,w_2\cdots w_n\}$，构造n棵二叉树的集合$F=\{T_1,T_2\cdots T_n\}$，其中每棵二叉树中均只含一个带权值为w_i的根结点，其左、右子树为空树；
    2. 在F中选取其根结点的权值为最小的两棵二叉树，分别作为左、右子树构造一棵新的二叉树，并置这棵新的二叉树根结点的权值为其左、右子树根结点的权值之和；
    3. 从F中删去这两棵树，同时加入刚生成的新树；
    4. 重复(2)和(3)两步，直至F中只含一棵树为止。

- Huffman编码：即令结点每个左支为0，右支为1，每个字符（结点）出现的概率相当于结点权值，然后按照Huffman算法构造Huffman树即可

  ▶ eg

- 堆
  - 最大堆：任意节点都小于其父节点。最小堆：任意节点都大于其父节点
  - ▶ 堆化、堆的构造和删除

    堆化操作最坏的时间复杂度是$O(h)$，h是堆的高度

    自顶向下的堆构造的时间复杂度小于$O(n \log n)$，n是堆中结点数目

    自底向上构造堆的时间复杂度：$O(n)$

- 图
  - 邻接矩阵（适于稠密图）：行表示边的起点，列表示边的终点。若图中存在一条从顶点v到顶点w的边，则矩阵中处于v行、w列的元素为1，否则为0、

    ▶ eg

    基于邻接矩阵实现的插入边、删除边、判边存在等 时间复杂度：$O(1)$。访问图中某个结点的所有邻接顶点，时间复杂度是$O(n)$；依次访问所有顶点的邻接顶点的时间复杂度为$O(n^2)$
  - 邻接表（适于稀疏图）：用一个链表数组来表示图，每个单链表的头结点对应于图的一个节点，这个顶点的所有邻接顶点被链接在这个链表中。除了单链表的表头外，单链表的各个节点表示了图中的一条边

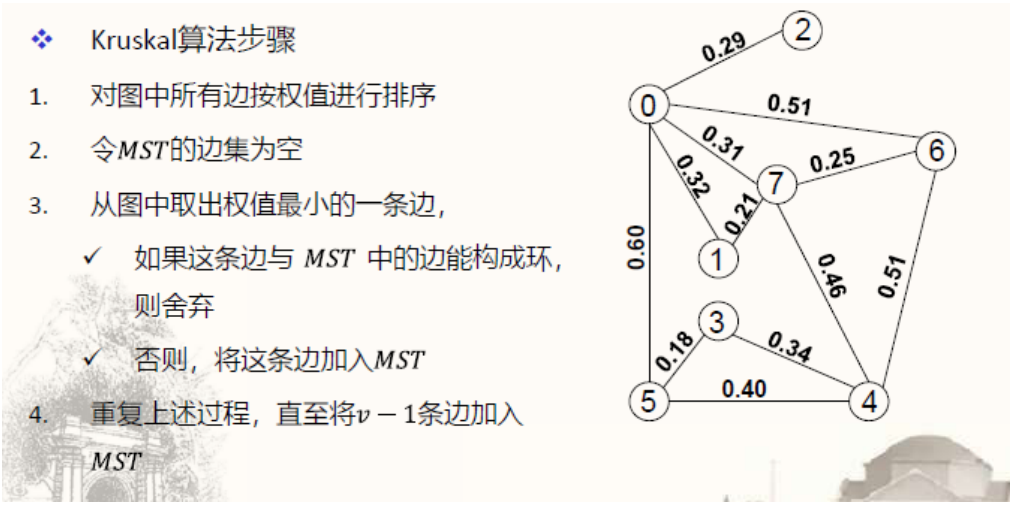
    ▶ eg

    基于邻接表实现的操作中，边插入时间复杂度为$O(1)$，边删除、判边存在等操作其时间复杂度为$O(n)$，要访问图中某个结点的所有邻接顶点其时间复杂度为$O(n)$，要依次访问所有顶点的邻接顶点，其时间复杂度为$O(n+e)$
  - 图的遍历：
    - 深度优先遍历（DFS）：邻接表表示下，遍历图的时间复杂度为$O(n+e)$。邻接矩阵表示下，查找每一个顶点的所有边时间复杂度为$O(n)$，遍历图中所有顶点的时间复杂度$O(n^2)$。邻接矩阵和邻接表下，空间复杂度都是$O(n)$
    - 广度优先遍历（BFS）：邻接表下，遍历的时间复杂度为$O(n+e)$，邻接矩阵下遍历的时间复杂度为$O(n^2)$。空间复杂度为$O(e)$（若遍历中对所有未访问的顶点都进行入队操作）或$O(n)$（若遍历中对所有入队顶点加入入队与否的标志）
  - 最小生成树（MST）
    - Prim算法（$O(n^2)$，适于稠密图）

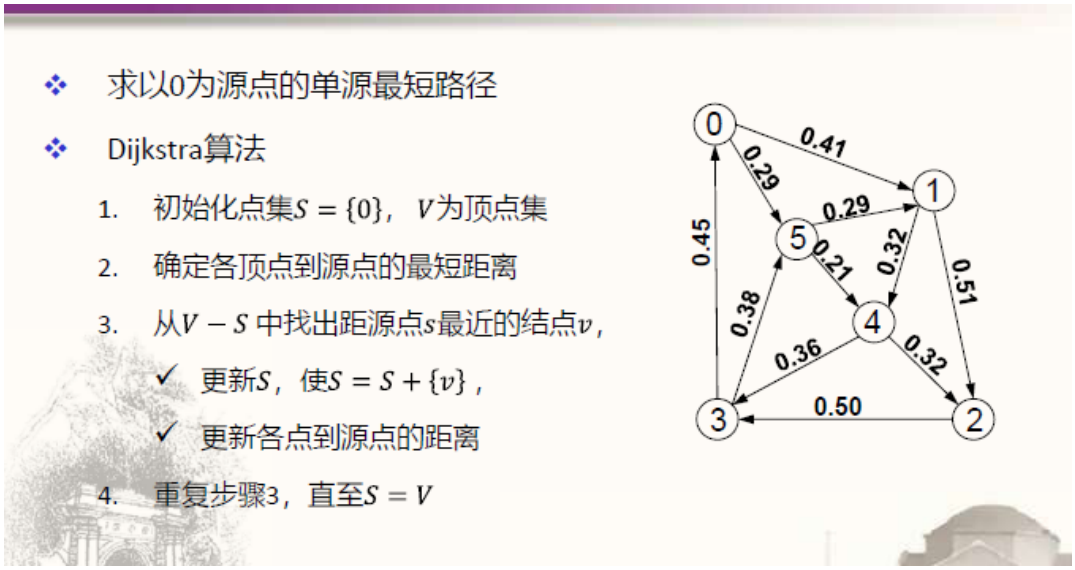
      对带权无向图G求最小生成树的过程：

- Kruskal算法（$O(e\log_2 e)$，取决于边数目，适于稀疏图）



  - ▶ eg(邻接表+prim+kruskal)
- 最短路径树（SPT）
- 单源最短路径（找出起始点s到各点的最短路径）（Dijkstra算法,$O(n^2)$）



  - ▶ eg
- 全源最短路径（求出图中任意两点之间的距离）（Floyd算法,$O(n^3)$）

  对有n个顶点的图，初始化$n \times n$的矩阵D_0为图的边值矩阵，按以下公式进行迭代：

$$D_k[i][j] = \min\{D_{k-1}[i][j],\ D_{k-1}[i][k] + D_{k-1}[k][j]\},\quad k = 1, 2, \cdots, n-1$$

$D_k[i][j]$是从顶点i到顶点j的路径长度，这条路径在所有不经过编号大于k的顶点的路径中是最短的。经过n次迭代，最后得到的就是全源最短路径矩阵

▶ eg

- 平均查找长度（ASL）：查找过程中，为确定目标数据的位置，需进行的关键字比较次数的期望值

  设P_i为第i 个元素被查找的概率；C_i为查找所需要的比较次数。则：

$$ASL = \sum_{i=0}^{n-1} P_i C_i$$

- 顺序查找：

$$ASL = \sum_{i=0}^{n-1} \frac{1}{n}(i+1) = \frac{n+1}{2}$$

因此对于长度为N的线性查找表，平均需要进行N/2次查找

顺序查找若满足表中元素不重复，等概率出现，则平均查找长度是表长的一半

- 折半查找：若线性查找表对关键字是有序的，则可折半查找。

  折半查找需要的比较次数不超过$\lfloor \log_2 n \rfloor + 1$，平均查找长度为$O(\log_2 n)$

  斐波那契查找的时间复杂度也是$O(\log_2 n)$。

❖ 不同查找表的效率比较

	平均情况		最坏情况	
	查找	插入	查找	插入
无序顺序表	$O(n)$	$O(1)$	$O(n)$	$O(1)$
无序线性链表	$O(n)$	$O(1)$	$O(n)$	$O(1)$
有序顺序表	$O(\lg n)$	$O(n)$	$O(\lg n)$	$O(n)$
有序线性链表	$O(n)$	$O(n)$	$O(n)$	$O(n)$
二叉搜索树	$O(\lg n)$	$O(\lg n)$	$O(n)$	$O(n)$

- 散列方法
  - 凡是基于关键字比较的方法，其时间复杂度范围为$O(\log_2 n) \sim O(n)$
  - 散列函数：实现从关键字到哈希表中存储位置的映射。

    散列函数的定义域必须包括需要存储的全部关键字，如果散列表允许有m个地址，其值域必须在0~m-1之间；

    散列函数应近似为随机的，对每一个输入，相应的输出在值域上是等概率的，这样有利于减少冲突
  - 冲突处理
    - 链地址法

      ▶ 解释

      定义装载因子：$\alpha = N/M$，其中N为关键字数目，M为地址链个数，$\alpha > 1$，表示了链表平均长度。每个地址链的长度最好为5~10
    - 开放定址法（线性探测）

      冲突发生时，顺序检查表中下一个位置，若不冲突则插入在这个（下一个）位置

      随着Hash表中元素数目增加，会出现元素聚集现象，称为聚类
    - 开放定址法（双重散列）

      不再是检查冲突点后面紧邻的表位置，而是采用第二个散列函数（再散列函数），得到一个固定增量序列来确定探测序列。

      在再散列函数的选择上，需要注意散列函数的取值必须与表长互素，否则某些序列将会非常短。另外，散列函数取值也不能是0，否则会使探测形成死循环。

- 对开放定址法同样定义装载因子$\alpha = N/M$，其中N为元素数目，M为表长。$\alpha < 1$，表示表中位置被占据的百分比。当α很小时，Hash表称为稀疏表。一般建议Hash表不要达到半满状态。
- 开放定址法相对于链地址法，具有更好的时间性能，但空间效率较低。
- 散列的删除：开放定址法中不能直接删除，一种解决办法是给被删关键字加上标志留在散列表中，另一种是重新散列
- 散列的性能：
  - 在Hash表足够稀疏时，线性探测性能最好，但散列接近于满时，性能下降
  - 双重散列使用内存最为有效，但时间复杂度高于线性探测散列。散列表趋近于满时，明显优于线性探测。
  - 链地址法容易实现，不会随着表中元素增加而出现性能迅速下降的情况。但其平均情况下的时间性能不如开放定址法。

- eg:

  给定 Hash 函数 H(k)=k mod M，表长 M=13，

序号 i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
关键字 k	52	11	15		43	70	84		99			37	25
冲突次数	0	3	0		0	0	0		0			0	0

  ▶ details

- 排序
- 一般情况下，N个对象的序列排序所需要的比较次数是$O(N\log N)$

  $O(N\log N)$是一般情况下排序算法可期待的最好的时间复杂度

  基本排序算法的平均情况下的时间复杂度是$O(N^2)$

  高级排序算法的平均情况下的时间复杂度是$O(N\log N)$

  - 冒泡排序：稳定，**效率与输入序列的特性无关**，跑$n-1$趟，执行的比较操作次数为$n(n-1)/2$，执行的交换操作次数最坏也为$n(n-1)/2$，时间复杂度 $O(n^2)$
  - 直接插入排序：**插入排序效率与输入特性有关**。需要$n-1$趟元素插入操作，最好下（已经有序）总的关键字比较次数为$n-1$，元素移动次数为$2(n-1)$。最坏下（反序）总的关键字比较次数和元素移动次数都是$n(n-1)/2$。平均下，关键字比较次数和元素移动次数约为$n^2/4$。因此直接插入排序的时间复杂度为$O(n^2)$
  - 折半插入排序：**效率与输入有关**。稳定，所需的关键字比较次数与初始排列无关，仅依赖于元素个数。需要进行的比较次数约为$n\log_2 n$。n较大时，总关键字比较次数比直接插入排序的最坏情况好得多，但比起最好写情况要差。折半插入排序的元素移动次数与直接插入相同，依赖于元素的初始排列。总时间复杂度仍为$O(n^2)$
  - 希尔排序：缩小增量排序，介于基本排序和高级排序之间（利用直接插入排序作为子序列）

如果序列中，间距为h的元素都有序，我们就称这个序列为h-排序的
设计步长序列$h_0, h_1, \cdots, h_n$，这是一个单调递增序列，$h_0 = 1$ ，第1步将序列变成h_n-排序,第2步
将序列变成h_{n-1}-排序，进行完第n步后，排序完成。

好处：用于优化简单插入排序，因为排序前期h_i较大，每个子序列中元素较少，排序较快。而后期由于前面工作基础导致子序列中大部分元素已经有序，接近于直接插入排序的最好情况。对于选择排序等性能与序列特性基本无关的没有明显优化作用。

- 直接选择排序：**不稳定**，关键字比较次数为$n(n-1)/2$，与初始排列无关。元素移动次数与初始排列有关，当初始序列已经有序的时候，元素交换次数为0.最坏时每一趟都要进行交换，总的交换操作次数为$n-1$。**选择排序的性能与序列特性基本无关。**$O(n^2)$

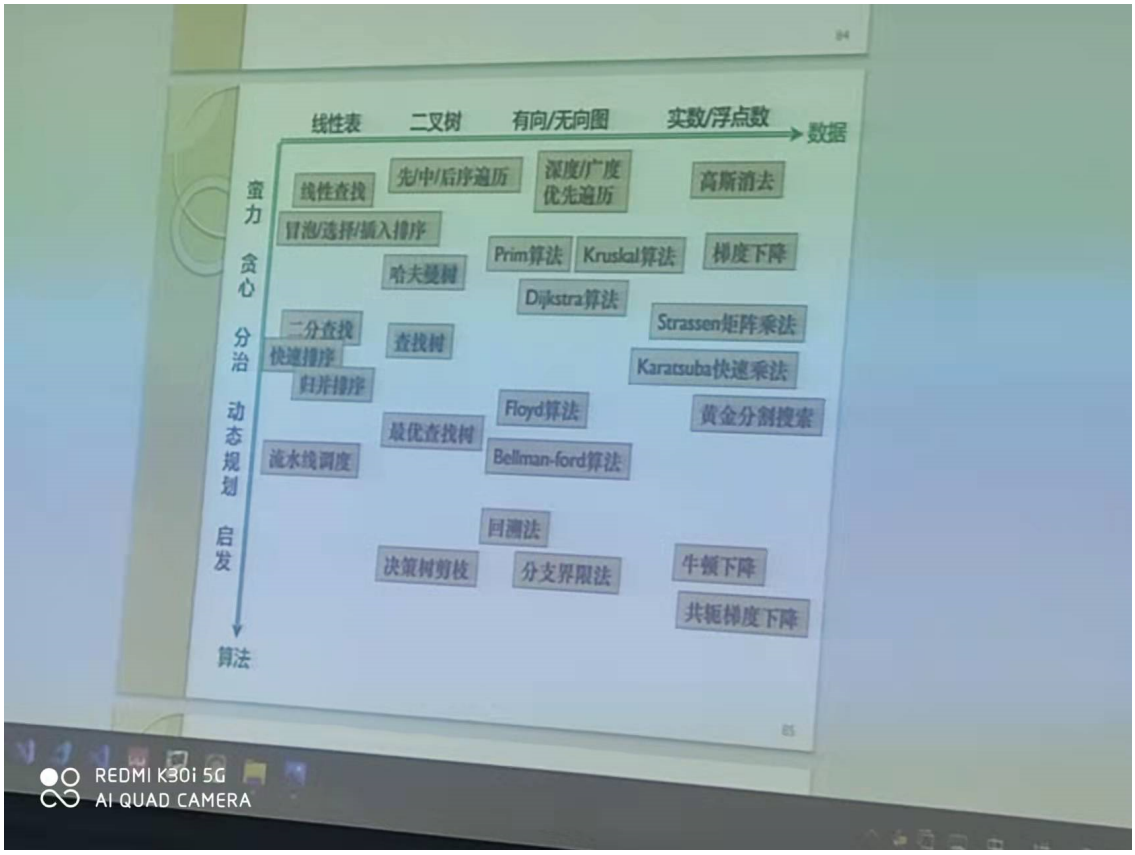
排序方法	比较次数		移动次数		稳定性
	最好情况	最坏情况	最好情况	最坏情况	
直接插入排序	$O(N)$	$O(N^2)$	0	$O(N^2)$	√
折半插入排序	$O(Nlog_2N)$		0	$O(N^2)$	√
冒泡排序	$O(N)$	$O(N^2)$	0	$O(N^2)$	√
选择排序	$O(N^2)$		0	$O(N)$	×

- 快速排序：**快排性能与输入序列有关**,**不稳定**，基于**分治**策略，总时间复杂度为 $O(n\log_2 n)$ 。最坏是序列已经有序时，为$O(n^2)$
- 归并排序：稳定，**归并排序性能与输入序列无关**，$O(n\log_2 n)$，但需要$O(n)$附加内存空间
- 堆排序：**不稳定**，**完全本地操作**，自底向上构造堆的时间复杂度为$O(n)$，删除堆顶元素并重新调整为新堆的时间复杂度为$O(2\log_2 n)$，所以对n个元素进行堆排序的时间复杂度为$O(n\log_2 n)$、

排序方法	比较次数		移动次数		稳定性	附加存储	
	最好情况	最坏情况	最好情况	最坏情况		最好	最差
直接插入排序	$O(N)$	$O(N^2)$	0	$O(N^2)$	√	1	
折半插入排序	$O(Nlog_2N)$		0	$O(N^2)$	√	1	
冒泡排序	$O(N)$	$O(N^2)$	0	$O(N^2)$	√	1	
选择排序	$O(N^2)$		0	$O(N)$	×	1	
快速排序	$O(Nlog_2N)$	$O(N^2)$	$O(Nlog_2N)$	$O(N^2)$	×	$O(log_2N)$	$O(N)$
归并排序	$O(Nlog_2N)$		$O(Nlog_2N)$		√	$O(N)$	

- 就平均性能来说，快速排序比归并排序速度快
- 归并排序是稳定的，快速排序是不稳定的，性能可能退化

- 算法设计思想



数值分析

- 误差
  - 绝对误差：已知x为真实值，而$\tilde{x}$是近似值，则定义$E(x)=x-\tilde{x}$为**绝对误差**
  - 相对误差：定义$E_r(x)=\frac{(x-\tilde{x})}{x}=\frac{E(x)}{x}$为**相对误差**
  - 函数值的绝对误差：$E[f(x)]=f'(x)E(x)$
  - 函数值的相对误差：$E_r[f(x)]=\frac{E[f(x)]}{f(x)}$
  - ▶ eg

  设需要计算$y=f(x)$，其中f做了近似（输入的x是确定的），近似结果为$\hat{y}$
  - 前向误差：$\Delta y=\hat{y}-y$
  - 后项误差：$\Delta x=\hat{x}-x$，其中$f(\hat{x})=\hat{y}$
  - 相对前向误差：$|\Delta y/y|$
  - 相对后向误差：$|\Delta x/x|$
  - ▶ eg

  - 有效数字：$|x-\tilde{x}|\leq\frac{1}{2}\times10^{-k}$，取符合要求的最大整数k，从小数点后第k位至左侧非零数字间的所有数字为有效数字。也即$k\leq-\lg(2|x-\tilde{x}|)$
  - ▶ eg

  - 有效数字与误差
  - ▶ eg（取有效数字之后运算精确结果所在范围？）

- 敏感性和病态性
  - 条件数：$cond = \frac{|\text{相对前向误差}|}{|\text{相对后向误差}|}$，条件数表示从输入数据的相对差异到解的相对差异的放大系数

    $cond \leq 1$表明问题不敏感，良态。$cond \gg 1$则病态。
  - 绝对条件数：$cond_{abs} = |f'(x)|$
  - 相对条件数：$cond = |\frac{xf'(x)}{f(z)}|$ （也即条件数）
  - 利用$E = \frac{\Delta x}{x} \times cond$得到算式计算之后的相对误差，$\Delta x$即为有效数字下一位的浮动
  - ▶ eg（取有效数字后计算结果的相对误差分析）

- 浮点数系统

参数	β	p	L	U
含义	基数	精度	指数值上界	指数值下界

任意一个浮点数x可以表示成如下形式：

$$x = \pm \left(d_0 + \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \beta^E$$

其中整数d_i满足$0 \leq d_i \leq \beta - 1$，一般称$p$位$\beta$进制数$d_0 d_1 \cdots d_{p-1}$为**尾数**，称$d_1 \cdots d_{p-1}$为尾数的小数部分，若要求$d_0 > 0$则称该浮点数系统是正规化的。整数$E$满足 $L \leq E \leq U$，称为**指数**

$IEEE$ 64位双精度浮点数：

$$x = (-1)^s (1 + f) 2^{c-1023}$$

$\beta = 2$，也即二进制。首位s用来表示浮点数，随后的11位c用来表示指数，剩余52位f用来表示小数的尾数部分（并且有$d_0 = 1$，也即$p = 52 + 1 = 53$）。定义$E = c - 1023$（为了保证非常接近0的数也能被表示）。另外，$IEEE$浮点数体系中将$c = 0$和$c = 2047$留给了特殊值表示，因此指数的下溢限为$L = E_{min} = 1 - 1023 = -1022$，上溢限为$U = E_{max} = 2046 - 1023 = 1023$。

eg：求IEEE双精度浮点数系统中下述机器数的十进制取值

$$x = \underbrace{0}_{s,1\text{位}} \ \underbrace{10000000001}_{c,11\text{位}} \ \underbrace{1001000100000001000000000000000000000000000000000000000000000000}_{f,52\text{位，其中1位于第1,4,8,16位置（从左往右数）}}$$

$s = 0$ 表示这是个正数

指数部分：$c = 2^{10} + 2^0 = 1025$

尾数的小数部分：$f = \frac{1}{2} + \frac{1}{2^4} + \frac{1}{2^8} + \frac{1}{2^{16}} = 0.5664215087890625$

综上：

$$x = (-1)^s (1 + f)^{c-1023} = 2^{1025-1023}(1 + f) = 6.26568603515625$$

一个正规浮点数系统可以表示的最小正数称为该系统的**下溢限（UFL）**

$$UFL = \beta^L$$

。。。。最大正数称为**上溢限（OFL）**

$$OFL = \beta^U(2-2^{-p+1})$$

例如：IEEE双精度浮点数系统上溢限和下溢限：

$$UFL = 2^{-1022} \approx 2.225 \times 10^{-308}$$
$$OFL = 2^{1023}(2-2^{-53+1}) \approx 1.798 \times 10^{308}$$

机器精度：

$$\epsilon_{mach} = \beta^{1-p}/2$$

eg：IEEE单精度浮点数字长32位，其中符号1位，指数部分8位，正规化的尾数的小数部分23位，求其下溢限、上溢限、机器精度

由于指数部分8位，因此取$E = c-(2^{8-1}-1) = c-127$，其中c=0和c=255不用

易得：$\beta = 2,\ p = 23+1 = 24,\ L = 1-127 = -126,\ U = 254-127 = 127$

因此有：

$$UFL = 2^{-126} \approx 1.175 \times 10^{-38}$$
$$OFL = 2^{127}(2-2^{-24+1}) \approx 3.403 \times 10^{38}$$
$$\epsilon_{mach} = 2^{-23}/2 \approx 5.96 \times 10^{-8}$$

- 一元方程求解
  - 二分法：

    求在$[a,b]$上的根，要求绝对误差不超过TOL，则迭代步数的期望值为

    $$k = \lceil \log_2((b-a)/TOL) \rceil$$

    ▶ eg
  - **不动点法（important）：**

    若对于函数g，存在x满足$g(x) = x$，则x 称为g的不动点

    记$f(x) = g(x) - x$，则g 的不动点就是$f(x) = 0$的解

    不动点法迭代就是利用迭代式$x_{k+1} = g(x_k)$来求$f(x) = 0$的解

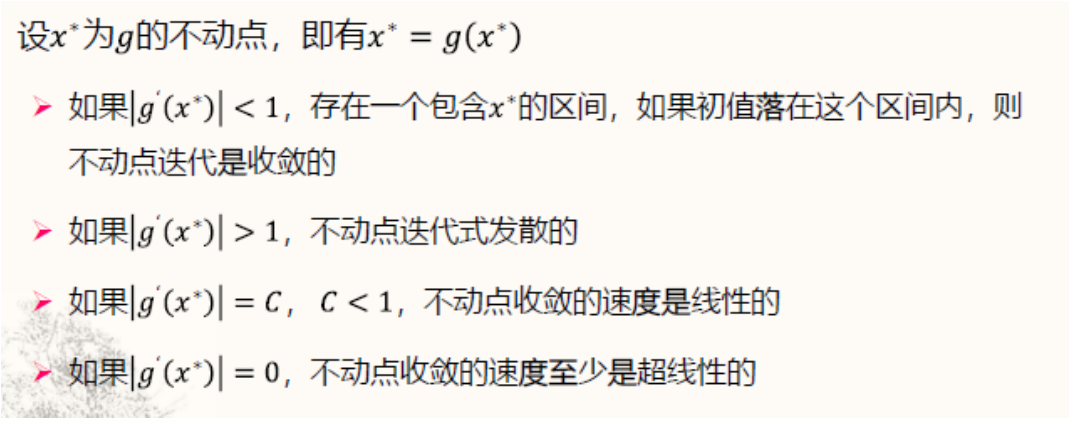
    对于$f(x) = 0$，对应的不动点函数可能有多种形式，如：

    $$f(x) = x^2 - x - 2 = 0$$

    对应的不动点函数有：

    $$x^2 - 2 = x \rightarrow g(x) = x^2 - 2 \tag{1}$$
    $$x^2 = x + 2 \rightarrow x = \sqrt{x+2} \rightarrow g(x) = \sqrt{x+2} \tag{2}$$
    $$x^2 = x + 2 \rightarrow x = 1 + \frac{2}{x} \rightarrow g(x) = 1 + \frac{2}{x} \tag{3}$$
    $$x(x-1) = 2 \rightarrow x = \frac{2}{x-1} \rightarrow g(x) = \frac{2}{x-1} \tag{4}$$

    收敛性：



因此，选择初值x_0 的时候，应该选择$|g'(x_0)| < 1$才能收敛。并且不动点函数需要满足：将$[a,b]$区间映射回本身，也即$\forall x \in [a,b]$，有$g(x) \in [a,b]$，才能收敛。

- 牛顿迭代法（平方收敛）

  不动点函数形式：

  $$g(x) = x - \frac{f(x)}{f'(x)}$$

  也即迭代式：

  $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

- 割线法：

  由导数定义：

  $$f'(x_{k-1}) = \lim_{x \to x_{k-1}} \frac{f(x) - f(x_{k-1})}{x - x_{k-1}}$$

  在牛顿迭代中，认为x_{k-2}非常接近x_{k-1}，则有：

  $$f'(x_{k-1}) \approx \frac{f(x_{k-2}) - f(x_{k-1})}{x_{k-2} - x_{k-1}}$$

  再代入牛顿迭代式即得割线法迭代式：

  $$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})} = x_{k-1} - \frac{f(x_{k-1})(x_{k-1} - x_{k-2})}{f(x_{k-1}) - f(x_{k-2})}$$

- 向量范数、矩阵范数以及矩阵病态性

  - 向量范数

    1-范数：$||x||_1 = \sum_{i=1}^{n} |x_i|$

    2-范数：$||x||_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$

    ∞-范数：$||x||_\infty = \max_{i=1}^{n} |x_i|$

  - 矩阵范数

    $||A||_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$，也即列绝对值和的最大值

    $||A||_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$，也即行绝对值和的最大值

  - 矩阵病态性

矩阵的条件数定义为其范数以及逆矩阵的范数的乘积$cond(A) = ||A^{-1}|| \cdot ||A||$，条件数在1附近的问题为良态问题，条件数高的问题称为病态问题。奇异矩阵条件数定义为$cond = \infty$

▶ eg

- 残差：

  验证方程组解的有效性

  残差：$||r|| = ||b - A\tilde{x}||$

  相对残差：$r = \frac{||b-A\tilde{x}||}{||A||\,||\tilde{x}||}$

  相对残差和解的相对误差之间的关系：

$$||\Delta x|| = ||x - \tilde{x}||$$
$$\frac{||\Delta x||}{x} \leq cond(A) \cdot r$$

- 线性方程组直接求解($O(n^3)$)
  - 初等消去阵和LU分解：

  即把原矩阵一步步化为上三角的操作

  下三角矩阵$M \in K^{n\times n}$称为初等消去阵，如果其对角线元素均为1，且除了对角线外仅有一列中存在非零元素。即：

$$M_k = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & m_n & 0 & \cdots & 1 \end{bmatrix}$$

令$\vec{m}_k = [0, \cdots, 0, m_{k+1}, \cdots, m_n]' \in \mathbb{R}^n$，则$M_k = I + \vec{m}_k \cdot \vec{e_k}'$

相应地，$M_k^{-1} = I - \vec{m}_k \cdot \vec{e_k}'$

至于m_k的具体求取，参照下例。

原矩阵A化成的上三角矩阵即为U

相应地，$L = I - \vec{m_1}\vec{e_1}' - \cdots - \vec{m_{n-1}}\vec{e_{n-1}}'$（也即各个初等消去阵对角线以下的元素取负值后合并）

▶ 一个LU分解的eg

⋆LU分解求解线性方程组：$Ax = b$

设$U \cdot x = y$，$L \cdot y = c$

然后先求出y，再代入前式求x。

▶ 一个LU分解求解线性方程组的eg

⋆ 列选主元求解线性方程组（等于是高斯消元求解线性方程组的一种）

主元：每步高斯消去时需要执行除法运算a_{kj}/a_{ii}，为了防止$|a_{ii}| \approx 0$，可以对系数矩阵进行行列变换，通过调整行列顺序选择绝对值大的元素作为主元。这称为选主元。

列选主元：选取对应列中对角线及以下元素中绝对值最大的元素作为主元。

eg：

$$\begin{bmatrix} 0.002000 & 88.71 \\ 7.129 & -4.330 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 88.73 \\ 66.96 \end{bmatrix}$$

显然$|a_{11}|$太小。因此调换顺序，改变主元：

$$\begin{bmatrix} 7.129 & -4.330 \\ 0.002000 & 88.71 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 66.96 \\ 88.73 \end{bmatrix}$$

再以7.129为主元进行操作

- 乔列斯基分解

  若A对称正定，则其LU分解结果更简单，即为：$A = LL'$

▶ eg

- 线性方程组的迭代解法(求解Ax=b)：
  - 矩阵的谱半径：$\rho(A) = \max(|\lambda|)$
  - 矩阵的收敛性：若$\lim\limits_{k\to\infty}(A^k)_{i,j} = 0$，或者$\rho(A) < 1$，则A是收敛的
  - **Jacobi迭代**

$$x[i]_k = \frac{b[i] - \sum\limits_{j\neq i} a_{ij}x[j]_{k-1}}{a_{ii}}$$

  - **高斯-赛德尔迭代**(收敛更快一点)

$$x[i]_k = \frac{b[i] - \sum\limits_{j<i} a[i][j]x[j]_k - \sum\limits_{j>i} a[i][j]x[j]_{k-1}}{a[i][i]}$$

其实是随手推的事，步骤见下例：

$$\begin{bmatrix} -5 & -1 & 2 \\ 2 & 6 & -3 \\ 2 & 1 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 32 \end{bmatrix}$$

step1：将线性方程组改写为如下分量形式：

$$\begin{gathered} -5x_1 - x_2 + 2x_3 = 1 \\ \Downarrow \\ x[1] = \frac{1 + x_2 - 2x_3}{-5} \end{gathered} \tag{1}$$

$$\begin{gathered} 2x_1 + 6x_2 - 3x_3 = 2 \\ \Downarrow \\ x[2] = \frac{2 - 2x_1 + 3x_3}{6} \end{gathered} \tag{2}$$

$$\begin{gathered} 2x_1 + x_2 + 7x_3 = 32 \\ \Downarrow \\ x[3] = \frac{32 - 2x_1 - x_2}{7} \end{gathered}$$

step2：

对于Jacobi迭代，则：把每个迭代式里边的x_i都代入上一轮的，如：

$$x[1]_1 = \frac{1 + x[2]_0 - 2x[3]_0}{-5} = \frac{1+0+0}{-5} = -0.2$$
$$x[2]_1 = \frac{2 - 0 + 0}{6} = 0.33$$
$$x[3]_1 = \frac{32 - 0 - 0}{7} = 4.57$$
$$\vdots$$

对于Gauss-Seidei迭代：比[i]靠前的元素带入这一轮的，比[i]靠后的元素带入上一轮的，如：

$$x[1]_1 = \frac{1 + x[2]_0 - 2x[3]_0}{-5} = \frac{1+0+0}{-5} = -0.2$$

$$x[2]_1 = \frac{2 - 2x[1]_1 + 3x[3]_0}{6} = \frac{2 + 2 \times 0.2 + 0}{6} = 0.4$$
$$x[3]_1 = \frac{32 - 2x[1]_1 - x[2]_1}{7} = \frac{32 + 2 \times 0.2 - 0.4}{7} = 4.57$$
$$\vdots$$

- 拟合与插值
  - 正规方程组（给出了求解线性最小二乘的一种方法）（$O(mn^2/2 + n^3/6)$）

    若$Ax = b$是超定的，则构造正规方程组：

    $$A'Ax = A'b$$

    从而解出$x = [x_1, x_2, x_3]$
  - 伪逆

    对非方阵A定义伪逆：

    $$A^{\dagger} = (A'A)^{-1}A'$$

    进而定义非方阵的条件数：$cond(A) = ||A||||A^{\dagger}||$
  - Householder变换（用其进行QR分解的时间复杂度：$O(mn^2 - n^3/3)$）

    ▶ eg

    利用Householder变换进行QR分解，进而求解最小二乘问题：

    $H_n \cdots H_2 H_1 Ax = \begin{bmatrix} R \\ O \end{bmatrix} x \cong H_n \cdots H_2 H_1 b$

    也即：利用H_i依次把系数矩阵各个列向量上三角化（把其尾部若干元素变成0）

    ▶ eg

  - 多项式插值（最小二乘拟合）

    n次多项式插值、牛顿插值、拉格朗日插值

    ▶ eg

- 无约束优化：

  定义：海森矩阵：

$$\{H_f(\mathbf{x})\} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

用于判断多元函数临界点（$\nabla f = 0$）的性质：

对于f的临界点$\mathbf{x}^*$，观察海森矩阵在这一点的性质：

若$H_f(\mathbf{x}^*)$正定，则$\mathbf{x}^*$是最小值点 （顺序主子式都>0）

负定，则是最大值点 （奇数阶顺序主子式<0，偶数阶顺序主子式>0，常见如1阶小于0二阶大于0）

不定，则是鞍点 （不符合以上两情况）

奇异，则问题是病态的

- ▶ eg1
- ▶ eg2
- ▶ eg3

- 约束优化问题

拉格朗日乘数法（偷工减料版）（以$\mathbf{x} = (x_1, x_2)$为例）

设目标函数：$f(\mathbf{x})$，约束条件为$h(\mathbf{x}) = 0$

以下例为载体解释这个流程：

求解等式约束优化问题：$f(\mathbf{x}) = x_1^2 + x_2^2,\ s.t.\ x_1 x_2^2 - 1 = 0$

拉格朗日函数：

$$\mathcal{L} = f(\mathbf{x}) + \lambda h(\mathbf{x})$$

本例中为：

$$\mathcal{L} = x_1^2 + x_2^2 + \lambda(x_1 x_2^2 - 1)$$

Jacobi：

$$\nabla \mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x_1} \\ \frac{\partial \mathcal{L}}{\partial x_2} \\ \frac{\partial \mathcal{L}}{\partial \lambda} \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathcal{J}_h(\mathbf{x})\lambda \\ h(\mathbf{x}) \end{bmatrix} = 0$$

本例中为：

$$\nabla \mathcal{L} = \begin{bmatrix} 2x_1 + \lambda x_2^2 \\ 2x_2 + 2\lambda x_1 x_2 \\ x_1 x_2^2 - 1 \end{bmatrix} = 0$$

解得：$\lambda = -\sqrt[3]{2},\ x_1 = 1/\sqrt[3]{2},\ x_2 = \sqrt[6]{2}$（实际上还有其他解，这里先用这组解分析）

其中$\mathcal{J}_h(\mathbf{x})$为约束雅克比。**它被要求是列满秩的，否则拉格朗日乘数法可能得不到约束最优点**

本例中为：

$$\mathcal{J}_h(\mathbf{x}) = \begin{bmatrix} x_2^2 \\ 2x_1 x_2 \end{bmatrix} = \begin{bmatrix} \sqrt[3]{2} \\ 2/\sqrt[6]{2} \end{bmatrix}$$

海森矩阵：为目标函数$f(x)$和约束函数$h(\mathbf{x})$海森矩阵的线性组合（$H_1 + \lambda H_2$），记为$B(\mathbf{x})$

本例中为：

$$B = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} + \lambda \begin{bmatrix} 0 & 2x_2 \\ 2x_2 & 2x_1 \end{bmatrix} = \begin{bmatrix} 2 & -2\sqrt{2} \\ -2\sqrt{2} & 0 \end{bmatrix}$$

约束雅克比零空间中的向量一定具有形式（两项分别取负倒数和倒数，这样点乘一定为0）：

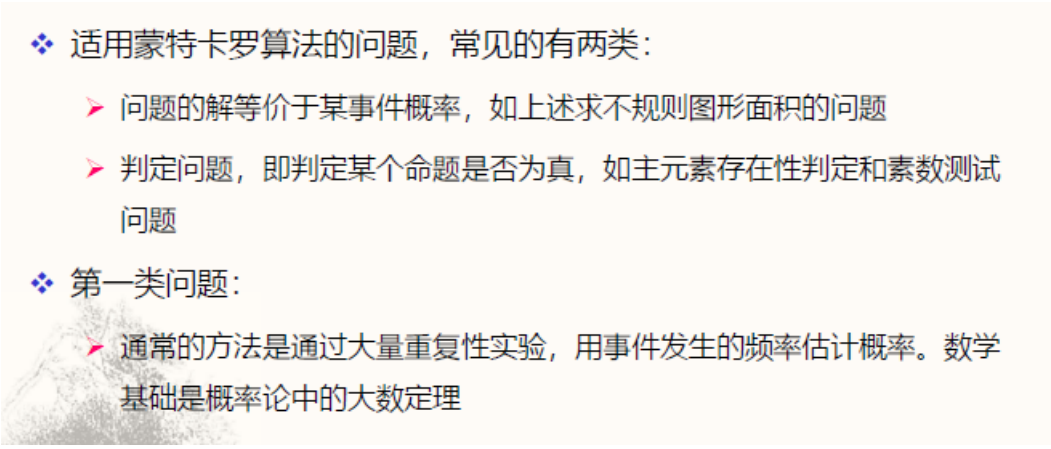
$$\mathbf{y} = k \begin{bmatrix} -1/\sqrt[3]{2} \\ \sqrt[6]{2}/2 \end{bmatrix}, \quad k \neq 0$$

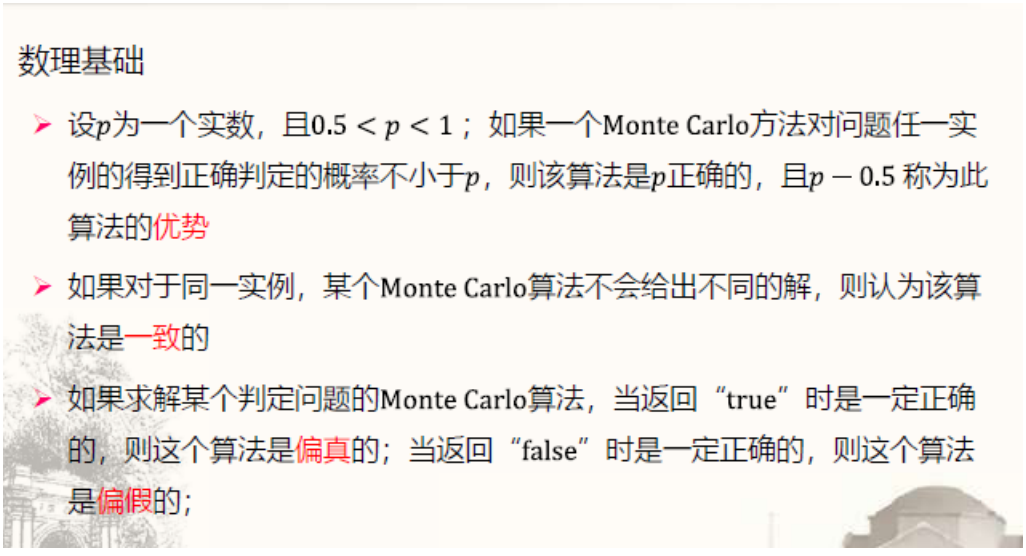
且可以验证：$y'By \approx 3.78k^2 > 0$，因此$(1/\sqrt[3]{2}, \sqrt[6]{2})$是优化问题的一个局部最优点

算法设计

- 随机算法
  - 舍伍德算法：适用于确定性算法在最坏条件下计算复杂度与其在平均情况下计算复杂度有着较大差异的情况。其基本思路就是引入随机性来减少计算问题好坏实例之间的差别，遵循这一思路的随机算法称为舍伍德算法。（只要输出一个解，就一定是正确的）
  - 拉斯维加斯算法：一些本质复杂度较高的计算问题可能不存在高效的确定性解法，针对这些问题在求解中加入一定随机性，可能会较快找到问题的解。执行一次可能得不到任何解，但一般会要求一个特定的拉斯维加斯算法对于任意可能的输入找到正确解的概率都要大于0（只要输出一个解，就一定是正确的）
  - 蒙特卡洛算法

    允许输出不精确、不正确的结果





▶ eg

算法	解的存在性	解的正确性	算法优势	设计要点
Sherwood算法	有	是	可能改善最坏情况	数据随机化
Las Vegas算法	不定	是	可能改善平均情况	与确定算法相结合
Monte Carlo算法	有	不保证	解决一些困难的问题	概率> 1/2 可多次执行

- 分治法

  将原始问题分解为子问题并递归求解，再从已经求解的子问题构建原始问题的解。至少包含两个递归调用的过程，并且递归求解的子问题互不重叠。

  例如：二叉树遍历（前序中序后序，$O(N)$）、快速排序（$O(N\log N)$）

- 减治法

  逐步减小原始问题规模，将原始问题转换为若干小规模问题

  例如：有序线性表折半查找、二分法求解非线性方程、小球称重问题

- 变治法

  将一个求解困难的问题转换为一个有已知解法的问题，并且这种转换的复杂度不炒股求解目标问题的算法复杂度。（不能减小问题规模）

  例如：无序线性表查找时，先排序，再用折半查找

- 贪心法

  例如：Prim和Kruskal计算最小生成树、Dijkstra计算最短路径树、Huffman求解Huffman编码、最速下降法求解优化问题

- 搜索算法：
  - 回溯法：

    如果能确定解空间树上某个结点对应的部分解是没有希望的，则可以分解对其后续分支的搜索。可以看做对解空间状态树的深度优先搜索

  - 分支定界法

    也在搜索解空间的过程中考虑某个分支是否可能发展为一个完整的最优解。与回溯法不同的是，分支界定法对于一棵解空间树的每一个节点所代表的部分解，都要计算出通过这个部分解能够扩展出的可行解的最佳值边界。利用最佳值边界确定对各分支的扩展和遍历的顺序，并确定最优解。

- 动态规划：

  与分治法不同的是，各个子问题都是相互有关系的。每个阶段做选择时，不仅要考虑当前状态，还要考虑该选择对后续选择的影响。

  需要满足的条件：**该问题的最优解应能分解出最优子结构**（不论过去的状态和决策如何，对前面的决策所形成的状态而言，余下部分的决策必定构成最优策略）

  还应满足无后效性条件（未来状态只取决于当前状态和决策）

eg1：给定一个序列，求最长递增子序列的长度（子序列的各元素之间可以跳过若干元素）

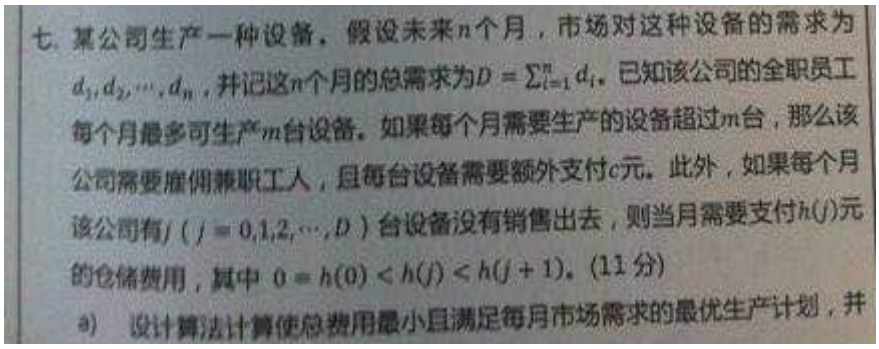
设$f(i)$是以第i个元素为结尾的最长递增子序列的长度。

则：

$$f(i)=\max_{\forall k\in\{k|a[k]<a[i]\}}\{f(k)+1\}$$

也即，若$a[k]<a[i]$，则$a[i]$可以加入以$a[k]$为结尾的最长递增子序列（长度$f(k)$）。若没有符合条件的k，则$f(i)=1$。最终结果即为遍历所有i ，也即：$\max\limits_{1<i<n} f(i)$

eg2：


七、某公司生产一种设备，假设未来n个月，市场对这种设备的需求为$d_1,d_2,\cdots,d_n$，并记这n个月的总需求为$D=\sum_{i=1}^{n}d_i$。已知该公司的全职员工每个月最多可生产m台设备。如果每个月需要生产的设备超过m台，那么该公司需要雇佣兼职工人，且每台设备需要额外支付c元。此外，如果每个月该公司有j（$j=0,1,2,\cdots,D$）台设备没有销售出去，则当月需要支付$h(j)$元的仓储费用，其中$0=h(0)<h(j)<h(j+1)$。(11分)

a) 设计算法计算使总费用最小且满足每月市场需求的最优生产计划，并

设$f(i,j)$是到第i个月为止，共生产j台机器所需要的最小钱数

$$f(i,j)=\min_{\forall k\in\{k|\sum_{\ell=1}^{i-1}d_\ell<k<D\}}\left\{f(i-1,k)+\underbrace{cost(j-k)}_{\text{这个月所花费钱数}}+\underbrace{keep(j)}_{\text{保养}}\right\}$$

其中：

k是第$i-1$个月时共生产的及其数量，$j-k$是这个月新生产的数量

其中：

$$cost(j-k)=\begin{cases}0, & j-k\le m\ \text{（不需要雇佣临时工）}\\(j-k-m)c, & j-k>m\end{cases}$$

$$keep(j)=h\left(j-\sum_{\ell=1}^{i}d_\ell\right)$$

eg3：

n件整权值礼物送给两人，总价值s，要求尽可能均分

设：$w(i)$是第i件礼物的价值，$F(i,j)$为前i件礼物（之中选一些），总价值不超过j元且离j最近的价值值

则可知，问题的解即为：$F(n,\lfloor\frac{s}{2}\rfloor)$

其中，F满足递推关系：

$$F(i,j)=\max_k\left\{F(i-1,k)+\begin{cases}0, & j-F(i-1,k)<w(i)\\w(i), & j-F(i-1,k)\ge w(i)\end{cases}\right\}$$

其中第二项即为决定第i件可不可以取