Chapter 3

2. 给出一个算法来探测一个给定的无向图是否包含圈.如果这个图包含圈,那么你的算法应该输出它.(它不应该输出图中所有的圈,仅仅其中的一个圈.)你的算法对于 n 个结点和 m 条边的图的运行时间应该是 $O(m+n)$.

解:

如果包含圈（环），则必定存在一个子图，是一个环路，环路中所有顶点的度>=2，故给出以下算法：

（1）删除所以度<=1 的顶点及相关的边，并将与这条边相关的顶点的度-1

（2）将度数变为 1 的顶点排入队列，并从该队列中取出一个顶点重复步骤（1）

（3）最后，如果还有未删除的顶点，则存在环，否则没有环

算法分析：若 m>=n，必存在圈；

若 m<n，上述算法删边和减度操作最多执行 m+n 次，则运行时间为 O(m+n)

算法如下：

```
V = vertex set
E = edge set
V = {G 中所有度数小于 2 的顶点}
E(V)为与 V 中的顶点相关的边

while V is not empty
    v = Next(V)
    delete v and E(v) from G
    update G;//改变和 v 点相连的顶点的度数
    将所有去掉 v 后度数变成<2 的顶点加入 V
end while

if G if empty
    没有回路
else
    有回路
```

6. 我们有一个连通图 $G=(V,E)$,以及一个指定的顶点 $u\in V$,假设我们计算一棵根在 u 的深度优先搜索树,并且得到一棵包含 G 的所有结点的树 T.假设接着我们计算一棵根在 u 的宽度优先搜索树,并且得到同一棵树 T.证明 $G=T$(换句话说,如果 T 既是根在 u 的一棵深度优先搜索树也是一棵宽度优先搜索树,那么 G 不可能包含任何不属于 T 的边.)

反证：假设存在边 e = (a,b)∈G 且 e 不属于 T

因为 T 是 DFS 树，所以 a,b 中的一个一定是另一个的祖先

因为 T 是 BFS 树,所以 u 到 a 的距离与 u 到 b 的距离最多差 1

因为 T 既是 DFS 树又是 BFS 树，那么 a,b 为父子关系

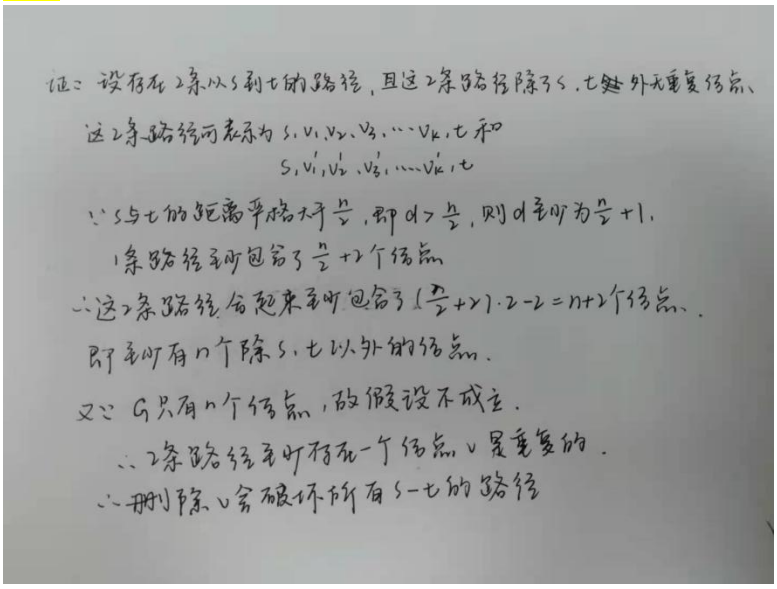
所以 a 与 b 在 F 中必存在一条边，即(a,b)∈T

与假设矛盾，故不存在这样的 e，故 G=T 得证

9. 存在一种本能的直觉，在一个通信网中两个远离的结点——由许多跳所分离——比起彼此接近的两个结点有着更为脆弱的连接. 某些算法的结果在某种程度上与精确表达这个概念的不同方式有关. 这里就是一个与路径对结点删除的敏感性相关的例子.

假设 n 个结点的无向图 $G=(V,E)$ 包含两个结点 s 和 t 并且使得 s 与 t 的距离严格大于 $n/2$. 证明一定存在某个结点 v，不等于 s 或 t，使得从 G 中删除 v 将破坏所有的 s-t 路径. (换句话说，从 G 通过删除 v 所得到的图不包含从 s 到 t 的路径.) 给出一个运行时间 $O(m+n)$ 的算法来找出这样一个结点 v.

答：



算法不会

Chapter 4

3. 你正在为一家货运公司做咨询，这家公司在纽约和波士顿之间有大量的货运生意. 生意量高得每天他们必须在这两个地点之间发出某些卡车. 卡车在允许携带的最大重量上有固定限制 W. 箱子一个接一个到达纽约站，并且每个包 i 有重量 w_i. 货运站是相当的小，每个时刻这个站至多只能容下一辆卡车. 公司的政策要求按照箱子到达的顺序运送它们；否则一个顾客看到在他后面到达的一个箱子反而先到波士顿会感到不舒服. 此刻，公司正在对装车使用一个简单的贪心算法：他们按照箱子到达的顺序装车，并且只要下一个箱子装不上，就让这辆卡车出发.

但是他们想知道他们是否可能用了过多的卡车，并且他们想问问你这种情况是否可以改进？他们正在想的是这样的事. 有时候发出一辆没装满的卡车，但允许后面的卡车装得更好，以这种方式也许可能减少所需要的卡车数.

对于一组给定的具有指定重量的箱子，证明目前正在用的贪心算法实际上使得需要的卡车数达到最小. 你的证明应该参照我们对**区间调度问题**用过的分析类型：应该通过确认一个量度来建立这个贪心装箱算法的最优性，而在这个量度下，它“领先于”所有其他的解.

证明：假设存在一种装箱方式，发出一辆未装满的卡车，允许后面的卡车装的更好，将未装满的卡车的后面辆卡车上顺序最靠前的箱子移动到未装满的卡车上，使其装满。重复这个操作，则除最后一辆车外没有未装满的车，这与已实施的贪心算法结果一致，调整过程也来增加或减少所需卡车数量，故假设方式并未优化已有贪心算法。

4. 你的某些朋友进入飞速发展的时间序列数据挖掘的领域，在这种挖掘中人们在某段时间出现的事件序列中寻找模式. 在股票交易中的购买——什么是正被买的——就是具有时间自然顺序的一种数据资源. 给定这类事件的一个长序列 S，你的朋友想要一种有效的方式来发现其中的某些模式——例如，他们很可能想知道是否下面这四个事件出现在这个序列 S 中，按顺序出现，但是不一定连续出现.

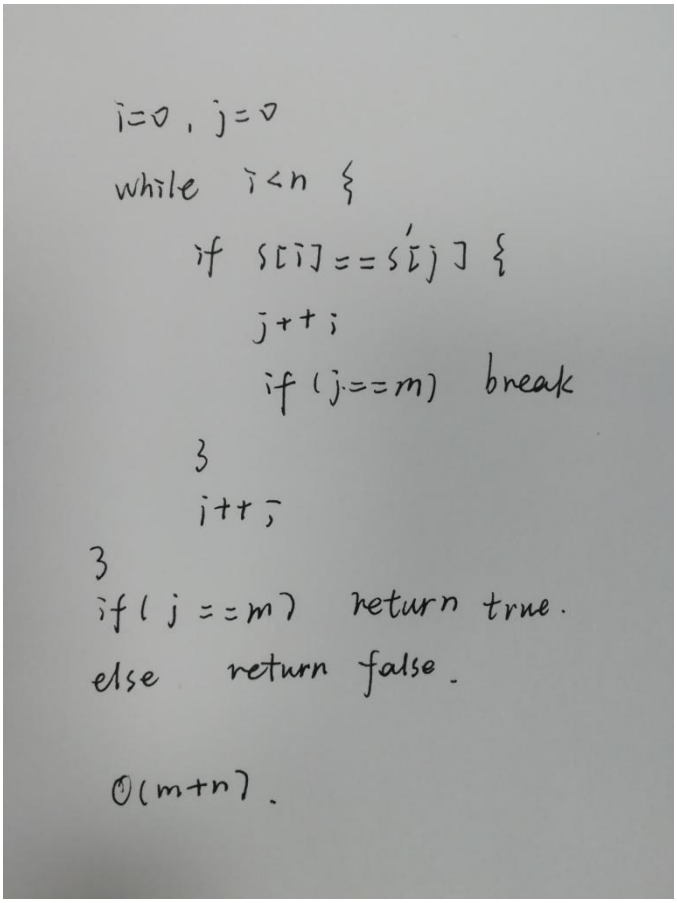
买 Yahoo 股票，买 eBay 股票，买 Yahoo 股票，买 Oracle 股票

他们从一组可能的事件（即可能的交易）和 n 个这种事件的序列 S 开始. 一个给定的事件在 S 中可能出现多次（即在一个序列 S 中 Yahoo 股票可能被买多次）. 如果存在一种方式从 S 中删除某些事件以使得留下的事件按照顺序等于序列 S'，我们就说序列 S' 是 S 的子序列. 例如，上述四个事件的序列因此是下面序列的子序列：

买 Amazon 股票，买 Yahoo 股票，买 eBay 股票，买 Yahoo 股票，
买 Yahoo 股票，买 Oracle 股票

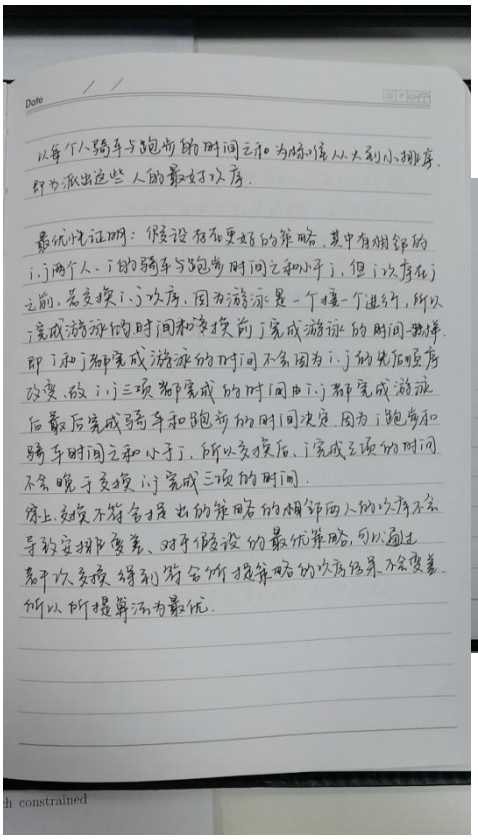
他们的目标是能够构造短序列并且快速检测它们是否是 S 的子序列. 因此这是他们向你提出的问题：对于两个事件序列——长度为 m 的序列 S' 与长度为 n 的序列 S，每个可能包含多于一次的事件——给出一个算法在 $O(m+n)$ 的时间确定 S' 是否为 S 的子序列.

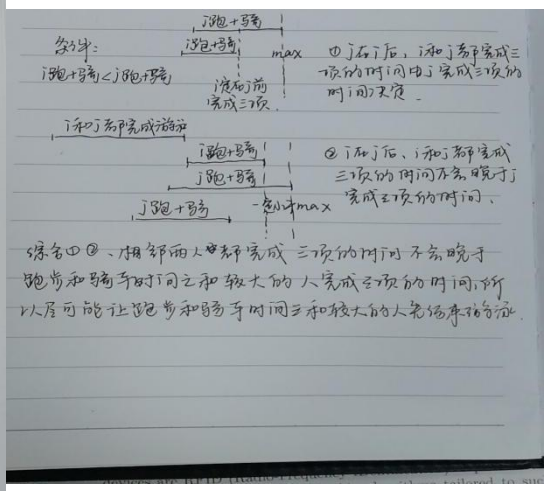
答：



6. 你朋友的工作是夏令营顾问，他正负责为一组中学年龄的露营者组织活动. 他的一个计划是遵循最小-三项全能训练：每个竞争者必须在泳池中游 20 圈，接着骑 10 英里自行车，然后跑 3 英里. 这个计划通过以下规则以错开的方式派出这些竞争者：这些竞争者每次必须一个人使用游泳池. 换句话说，第一个竞争者游 20 圈，出去，并且开始骑车. 只要第一个人出了这个游泳池，第二个竞争者开始游 20 圈；只要他或她出去并且开始骑自行车，第三个竞争者开始游泳……，等等.

每个竞争者有一个计划游泳时间（他或她完成 20 圈的预期时间），一个计划骑车时间（他或她完成 10 英里骑车的预期时间），以及一个计划跑步时间（他或她完成 3 英里跑步的预期时间）. 你的朋友想要对这个三项全能计划确定一个安排：竞争者开始先后的一个顺序. 假设他们每个人在三个部分恰好用了他们计划的游泳、骑车、跑步时间，一个安排的完成时间是所有竞争者结束三项全能所有三段的最早时间（此外，注意参与者们可以同时骑车和跑步，但是每个时刻至多可以一个人在游泳池里）. 如果一个人想使得整个竞赛尽可能早结束，什么是派出这些人的最好次序？更准确地说，给出一个算法产生一个最小完成时间的安排.





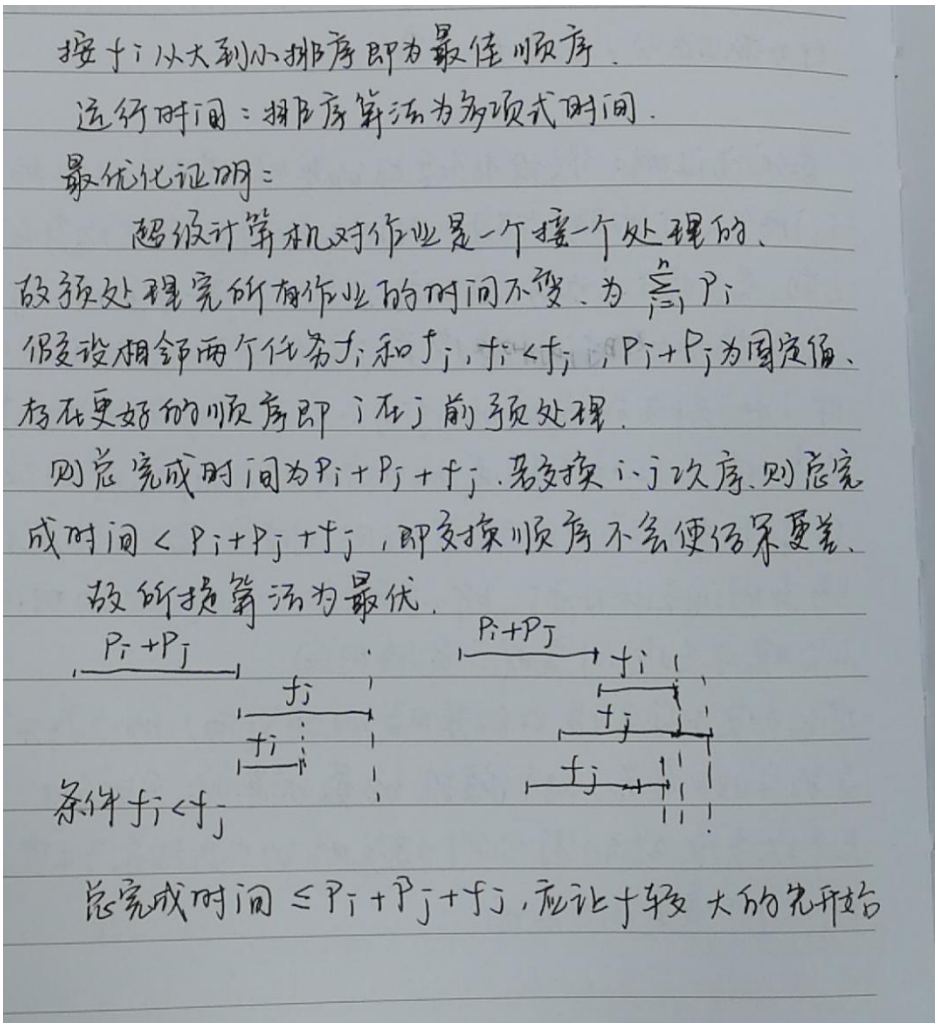
7. 广泛流行的西班牙语搜索引擎 EIGoog，每当它重新编译它的索引时需要做一系列的计算. 幸运的是，这家公司在它的配置上有一台大的超级计算机以及基本上无限供给的高端 PC 机.

他们把全部计算分成 n 个不同的作业，标记为 $J_1, J_2, \cdots, J_n$，这些作业可以相互完全独立地被执行. 每个作业由两步组成：首先需要在超级计算机上对它进行预处理，然后需要在这些 PC 机中的某一台结束作业. 我们说作业 J_i 在超级计算机上需要 p_i 秒时间，接着在一台 PC 上需要 f_i 秒时间.

由于前提中至少 n 台 PC 是有效的，作业的结束可以完全并行执行——所有的作业可以同时处理. 但是，超级计算机每个时刻只能在一个作业上工作，因此系统管理员需要提出一个在超级计算机上装入作业的顺序. 按次序，只要第一个作业在超级计算机上做完，它就被手动切断到一台 PC 上做结束工作；就在此刻，第二个作业可以被装入超级计算机；当第二个作业在超级计算机上做完的时候，可以将它继续放到一台 PC 上，不管第一个作业是否做完(因为 PC 是并行工作的)；同样地继续下去.

让我们说一个调度是一个关于超级计算机的作业次序，并且这个调度的完成时间是所有的作业在 PC 上都结束处理的最早时间. 这是极小化的一个重要的量，因为它决定了 EIgoog 多快可以产生一个新的索引.

给出一个多项式时间的算法找到一个具有最小完成时间的调度.



Chapter 5

2. 回顾求逆序个数的问题. 如本书中一样，给定 n 个数的序列 $a_1, a_2, \cdots, a_n$，其中我们假定所有的数都不相同，定义一个逆序是一对 $i<j$ 使得 $a_i>a_j$.

我们引入逆序计算问题作为考察两个排序有多大差别的一个好的度量指标. 但是，人们可能感觉这个量度太敏感了. 如果 $i<j$ 且 $a_i>2a_j$，我们把这对 i,j 叫做重要的逆序. 给出一个 $O(n\log n)$ 的算法计数在两个排序中的重要的逆序个数.

两趟归并，一趟求重要逆序数对数，一趟将两个数组进行合并

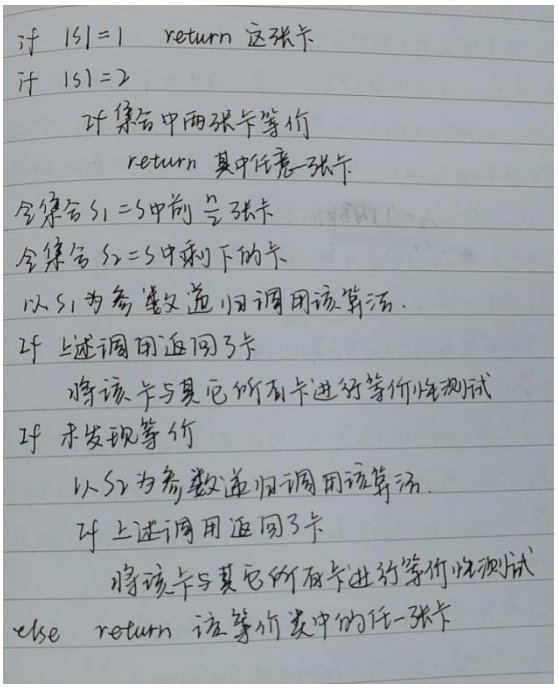
```
int merge(int a[], int b[], int l, int m, int r){
   int num1 = 0;
   int i=l;
   int j=m+1;
   int k = l;
   int t = m-i+1;
   while(i!=m+1&& j!=r+1){
      if(a[i]>2*a[j]){
         num1+=t;
         j++;
      }else{
         i++;
      }
      t = m-i+1;
   }
   while(i!=m+1){
      if(a[i]>2*a[j-1]){
         num1+=t;
         break;
      }else{
         i++;
      }
      t = =m-i+1;
   }
   i=l;
   j=m+1;
   k=l;
   while(i!=m+1&&j!=r+1){
      if(a[i]>a[j]) b[k++]=a[j++];
      else b[k++] = a[i++];
      while(i!=m+1) b[k++] = a[i++];
      while(j!=r+1) b[k++]=a[j++];
      for(i=l;i<=r;i++) a[i] = a[j]
      return num1;
   }
}
int merging(int a[], int b[], int l, int r){
   int m;
```

```
    int num1 =0,num2=0,num3=0;
    if(l<r){
      m = l+(r-l)/2;
      num1 = merging(a,b,l,m);
      num2 = merging(a,b,m+1,r);
      num3 = merge(a,b,l,m,r);
    }
    return num1+num2+num3;
}
```

3. 假设你正在为一家银行做咨询，这家银行正进行一项诈骗检测，他们向你提出下面的问题. 他们有一组 n 张被没收的银行卡，怀疑它们被用于诈骗. 每张银行卡是一个小塑料卡，包含一个带有某些加密数据的磁条，并且它与银行中唯一的账号对应. 每个账号可以有许多银行卡与之对应，如果两张银行卡对应了同一个账号，我们将说它们是等价的.

从一个银行卡直接读出账号是非常困难的，但是银行有一个高技术的“等价性测试器”，它拿两张银行卡并且在执行某些计算之后，确定是否它们是等价的.

他们的问题如下所述：在一组 n 张卡中，是否有比 $n/2$ 张还要多的卡全都是相互等价的？假定你对这些卡能做的唯一可行的操作是取两张卡并且把它们插入等价性测试器. 若只能启动 $O(n\log n)$ 次等价性测试器，显示怎样来确定对他们问题的答案.



7. 假设现在给定一个 $n\times n$ 的网格图 G(一个 $n\times n$ 的网格图就是一个 $n\times n$ 的棋盘的邻接图. 完全精确地说,它是一个图,它的结点集合是所有自然数的有序对 (i,j) 的集合,其中 $1\leqslant i\leqslant n$ 且 $1\leqslant j\leqslant n$;结点 (i,j) 与结点 (k,l) 被一条边相交当且仅当 $|i-k|+|j-l|=1$).

我们使用前面问题的某些术语. 每个结点 v 被一个实数 x_v 标记;你可以假设所有这些标记都不相同. 显示怎样对 G 的结点只用 $O(n)$ 次探查来找到 G 的一个局部最小结点(注意 G 有 n^2 个结点).

令B表示网格G中边界上的结点,如果网格G至少有一个局部最小值不发生在边界上,则说其满足属性*

若网格G满足属性*,则令V中B为一个与B中的结点相邻且比B中所有结点小的结点,令C为G的最中间的行和列上,但不在边界上的结点.

令S=B∪C,从G中删除S,使之成为4个子网格.

令T为与S相邻的所有结点,使用O(n)次探测,找到最小的结点 $u\in S\cup T$,由属性*知 $u\notin B$,则有两种情况:

① $u\in C$,则u为一个内部局部最小值.

② $u\in T$,令G'为包含u的子网格以及其边界上的S上的结点,则G'满足属性*.

故G'具有内部局部最小值,该值也是G的一个内部局部最小值.

在G'上递归调用该算法,可以找到这样的内部局部最小值,可知该算法有 $T(n)=O(n)+T(\frac{n}{2})$ 即 $T(n)=O(n)$

考虑所有G的情况先进行O(n)次探测,可以发现B中的最小值v

如果v是角落上的结点,则v即为所求.

否则,v有以往一的一个不属于B的相邻结点u.

若v的值小于u,则v为所求否则G满足属性*.调用上述算法进行求解. 整个过程共使用O(n)次探测.