算法设计练习题

一. 计算复杂性

1. P184 10.3 设计一个多项式时间的算法判断一个无向图 **G** 是否是 **2** 可着色的

算法：2-COLORING

输入：无向图 G(V, E)

输出：该图是 2 可着色的，则输出 yes；否则，输出 no.

1.取任一节点，标记为白色

2.所有与它邻接的节点标记为黑色

3.对任意已标记的节点 v，将所有与 v 邻接且未标记的节点标记为与 v 相反的颜色

4.重复步骤 3，直到不存在与已标记节点邻接且还未标记的节点

5.如果图中还有未标记的节点，那么这些节点一定在一个新的连通分量中，再选择其中一个节点标记为白色，转到步骤 3

6.如果得到的图中，所有邻接的节点都标记为不同的颜色，则输出 yes；否则，输出 no.

End 2-COLORING

时间复杂度分析：在 n 个顶点和 m 条边的图上进行分析，算法的运行时间是 $\theta(m+n)$

2. P185-186 10.16 团集问题的 **NP** 完全性是由可满足性问题归约到它证明的，给出一个从顶点覆盖到团集的较简单的归约

法 1：规约方法：设 G=（V，E）是连通无向图，$S \subseteq V$ 是一个团集当且仅当 $V-S$ 是 $\overline{G}$ 中的一个顶点覆盖。

证明：设 e=(u,v)是 G 中的任意边，$S \subseteq V$ 是一个团集当且仅当 u 和 v 都在 S 中，即 $V-S$ 是 $\overline{G}$ 中的一个顶点覆盖。

法 2：证明：设 G=（V，E）是连通无向图，$S \subseteq V$ 是 G 中的一个独立集，则 S 是 $\overline{G}$ 的一个团集，独立集 $\propto_{poly}$ 团集。因为顶点覆盖 $\propto_{poly}$ 独立集，根据定理 10.3，顶点覆盖 $\propto_{poly}$ 团集。

3. P185-186 10.26 用顶点覆盖问题规约到集合覆盖问题，证明集合覆盖问题是 **NP** 完全问题。

证明：第一步是说明集合覆盖问题是 NP 的。因为一个不确定性算法可以从猜测一个集合 X 的子集族 F 开始，然后验证是否存在 F 中的 k 个子集的并是集合 X。

第二步证明顶点覆盖问题可以在多项式时间内规约到集合覆盖问题。

设任意连通无向图 G=(V,E)，集合 X 为 G 中所有与边相邻的顶点的集合，其子集族则是每个顶点与其相邻的顶点构成的子集。集合 X 是满足集合覆盖的当且仅当 X 的子集族中存在 k 个子集的并是 X，当且仅当 G 中存在大小为 k 的顶点覆盖。

4. P215 12.16

证明：设$\{x_1,x_2,\ldots,x_n\}$是一个正实数集合。对于每一个实数 x_i，我们使它和二维平面中的点$\{(x_1,1),(x_j,0) \mid j \in 2,\ldots,n\}$相联系，这样，所构造的 n 个点都位于三角形边上。如果我们用 TRIANGULATION 问题的任何算法求解构造的实例，输出将是根据它们的 x 坐标排序的构造点的表，遍历表并读出每点的第一个坐标，结果是排序好的

数。所以，排序问题规约到 TRIANGULATION 问题，排序问题是 Ω(n logn)，TRIANGULATION 问题也是 Ω(n logn)。

5. P215 12.17

证明：设$\{x_1,x_2,\ldots,x_n\}$是一个升序排列的正实数集合，及实数 x。对于实数 x 及每一个实数 x_i，我们使它和二维平面中的点$\{(x,0),(x_i,0) \mid i\in 1,\ldots,n\}$相联系，这样，所构造的点都位于 x 轴上。如果我们用 NEAREST POINT 问题的任何算法求解，输出就是二分搜索要查找的数。所以，二分搜索问题规约到 NEAREST POINT 问题，二分搜索问题是 Ω(logn)，NEAREST POINT 问题也是 Ω(logn)。

6. P215 12.18

证明：设$\{x_1,x_2,\ldots,x_n\}$是一个正实数集合，对于每一个实数 x_i，我们构造点$(x_i,0)$与之对应，于是这些点在 x 轴上。如果我们用 ALL NEAREST POINT 问题的任何算法求解，输出将是每个点$(x_i,0)$对应的最近点对$(x_i,0)$，$(x_j,0)$。所以，CLOSEST-PAIR 问题规约到 ALL NEAREST POINT 问题，CLOSEST-PAIR 问题是 Ω(n logn)，ALL NEAREST POINT 问题也是 Ω(n logn)。

二. 随机算法

1. P241 14.2 假定你有一枚硬币，请设计一个有效的随机算法用来生成整数 **1,2...n** 的随机排列，**n** 为正整数，分析你的算法的时间复杂性。

基本思想：从空序列开始，逐个向序列添加 1, 2, …, n。根据二分搜索的思想，并利用多次抛硬币，来随机确定每个添加的数在序列中的位置。

```
算法 RANDOMIZE2
  输入：正整数 n
  输出：1, 2, …, n 的一个随机排列。
        A[1]=1
        for i=2 to n
          j=randombisearch(1, i-1)//在数组 A 中随机确定 i 的插入位置。
          insert(A , j, i)   //在数组 A 的 j 位置上插入值 i。
        end for
        output A[1..n]
  end RONDOMIZE2
  过程 randombisearch(low, high)
        // 利用二分搜索在 A[low..high+1]中随机确定值 i 的插入位置，
        // 并返回该位置。
        if low>high then
          return low
        else
          mid=⌊(low+high)/2⌋
          k=random(1,2) //抛一次硬币。
          if k=1 then high=mid-1 //插入位置在 A[low..mid]中。
          else low=mid+1 //插入位置在 A[mid+1..high+1]中。
```

```
            return randombisearch(low, high)
        end if
    end randombisearch
```

时间复杂性：$\Theta(n^2)$

2. **P241 14.5** 在算法 **RANDOMIZEDQUICKSORT** 的讨论中曾说过，为算法 **QUICKSORT** 得到一个 **O**（**log n**）期望时间的一种可能性，是通过排列输入元素使它们的次序变成随机的来获得的。描述一个 **O**（**n**）时间算法，先随机排列下

算法思想：对预排序的数组进行随机排列，使该数组与原先相比显得无序。尽量避免 QUICKSORT 算法最坏情况的发生即 n^2 时间，使之更趋近于最佳情况 nlogn 时间。

```
算法 PRE_DISPOSE
输入：n 个元素的数组 a[1…n]
输出：随机排列的数组 a
    for  i=1 to n
        P =random(n)//随机选择小于 n 的数
        Q=random(n) //随机选择小于 n 的数
        互换 a[P]和 a[Q]
    end for
end PRE_DISPOSE
```

3. **P241 14.7** 考虑对算法 **BINARYSERCH** 做如下修改见 **1.3** 节，在每次迭代中，随机的选择剩下的位置来代替搜索区间减半

设元素存储在一维数组 C 中，第 0 个位置不放元素，若每次生成的随机数都是要查找的剩余元素的第一个且未找到要搜索的数，则时间复杂度计算公式如下：

$$\begin{cases} C(n)=\dfrac{1}{n}\sum_{i=0}^{n-1}C(i)+1 \\ C(0)=0 \end{cases}$$

计算得到时间复杂度为 $O(\log n)$

4. 写出 **n** 皇后问题的如下随机算法：先在棋盘上随机放置 **m(m<n)**个互不冲突的皇后，然后用回溯法搜索其余皇后的位置。

```
算法 NQUEENS_RAN_ACCU
输入：正整数 n,m，其中 n 表示棋盘纬度，m 表示随机算法和回溯算法的处理的划分， m<n。
输出：若找到解，则输出 n 皇后问题的一个解 x[1..n],否则输出无解
Flag_Random=true //随机查找时是否有解得标记
Flag_Accu=false//精确查找时是否有解得标记
k=1
x[m+1]=0//精确查找初始化
while k<=n and Flag_Random
    if  k<=m//随机算法
```

```
j=0
  for i=1 to n //寻找第 k 行所有可放置皇后的位置。
      if place(k) then   //若第 k 行的位置 i 可放置皇后,
        j++; temp[j]=i; //则存储该位置。
      end if
  end for
  if j>0 then x[k]=temp[random(1, j)] //随机选取一个位置放皇后
  else Flag_Random =false//表示找不到解
  end if
  k++
else if   k>m//回溯算法
  while k>m and not Flag_Accu
      while x[k]< n and not Flag_Accu
        x[k]=x[k]+1 //试将第 k 行的皇后移到下一个位置。
        if place(k) then //第 k 行的当前位置可放置皇后。
          if k=n then Flag_Accu =true   //x[m+1..n]是一个解
           else   //x[m+1..k]是精确解答时的部分解
            k=k+1   //前进到下一行
            x[k]=0
          end if
        end if   //否则，剪枝
      end while
      k=k-1      //回溯
  end while
  if   k =m
       break; //退出整个循环
  end if
end while
if Flag_Random and Flag_Accu   then output x     //输出一个解
   else output "No solution"   //输出无解
```

三. 近似算法

1. 对于装箱问题，分别写出近似算法 **FF** 和 **BF**。

思路:

1.最先适配法(FF)：箱子编号为 1, 2, …, n，初始时各箱子为空。各项按 u1, u2, …, un 的顺序装箱，装项 ui 时，将其装入序号最小的可容得下该项的箱子中（即装入满足 *l*<= 1-si 的序号最小的箱子中，其中 *l* 表示箱子的已填充容量）。

2.最佳适配法(BF)：箱子编号为 1, 2, …, n，初始时各箱子为空。各项按 u1, u2, …, un 的顺序装箱，装项 ui 时，将其装入满足 l<= 1-si 并且使 l 值最大的箱子中。

算法 FF

输入: n 个项的集合 u[1…n], n 个箱子的容量 l[1…n]，其中 0<=u[i]<=1，l[i]=1.

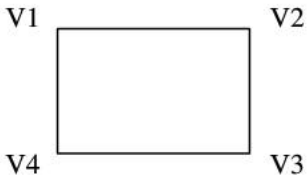
输出: 装这 n 个项的最少箱子的个数 k

```
k=1;//箱子的个数
```

```
    for i=1 to n//装项 ui 时，将其装入序号最小的可容得下该项的箱子中
      j=1
      flag=false;
      while j<=k and not flag//从序号最小的开始查找
        if l[j]>u[i] then//找到可以放进去的箱子
            l[j]=l[j]-u[i]
          flag=true;
        else j++//继续寻找
        end if
      end while
        if not flag then//没有找到可以放进去的箱子
         k++//开启新的箱子
         l[k]=l[k]-u[i];
        end if
    end for
    return k
end FF
算法 BF
 输入: n 个项的集合 u[1…n], n 个箱子的容量 l[1…n]，其中 0<=u[i]<=1，l[i]=1.
 输出:装这 n 个项的最少箱子的个数 k
    k=1;//箱子的个数
    for i=1 to n//装项 ui 时，将其装入满足 l<= 1-si 的箱子中使 l 值最大的箱子中
      if l[j]>u[i] then//找到 l 值最大的可以放进去的箱子
            l[j]=l[j]-u[i]
      else
          k++//开启新的箱子
          l[k]=l[k]-u[i];
      end if
     sort(l[1…k])//对前 k 个箱子的 l 值从大到小排序
     end for
     return k
end BF
```

2. P255 15.10

如图：



按照算法，先得到顶点的度数降序排列：V1、V2、V3、V4（度数均为 2），先将 V1 添加到覆盖中，删除边{ V1 ，V4}和{ V1，V2}，接着添加 V2，删除边{ V2，V3}，添加 V3，删除边{ V3，V4}。故得覆盖{ V1，V2，V3}，而最佳覆盖为{ V1，V3}或{ V2，V4}。

3. P256 15.14 15.15（这两题的答案是学长给的）

15.14 考虑在给定的图 G 中找出最大团集问题的近似算法，基本步骤：

1、 C= { }

2、 向 C 中添加一个顶点（该顶点不在 C 中，与 C 中每个顶点相连接）

3、 重复步骤 2，直到 C=G 或者 G-C 中任一点 x 与 C 中点都不是全部相连

Solution:

这里 $D_{II}=\{graph \mid graph是无向图\}$

取 $g\in D_{II}$,则 $S_{II}(g)$ 为实例 g 的侯选解集,设 σ 是由题意给出的算法。则

$\sigma\in S_{II}(g)$,定义 $f_{II}(\sigma)=k$,其中 k 为算法 σ 作用于实例 g 而找到团集的顶点个数

∵题目算法寻求最大团集，这是最大化问题，∴近似度 $R_\sigma(g)=\dfrac{OPT(g)}{\sigma(g)}\geq 1$

分三种情况讨论：

$i)$：若 $g\in D_{II}$, g 是一个具有 $m\geq 1$ 个顶点的无向完全图，则 $R_\sigma(g)=1$

$ii)$：若 g 是由 n 个无向完全图：$g_1,g_2,\cdots,g_n$ 之简单并，$g=\bigcup_{i=1}^{n}g_i$，

用 $\eta(g_i)$ 表示 g_i 的顶点个数，则 $R_\sigma(g)=\dfrac{\max\{\eta(g_1),\eta(g_2),\cdots,\eta(g_n)\}}{\eta(g_i)}=\dfrac{OPT(g)}{M}$

$M\in Z^+$。且 $OPT(g)\geq M$。上式分母取 $\eta(g_i)$ 示意算法 σ 输出的是具有 $\eta(g_i)$ 个顶点的团集 $g_i, i=1,2,\cdots,n$

$iii)$：对于 $\forall g\in D_{II}$, g 不是由 $i)$、$ii)$ 讨论的无向图，则可以去掉一些边，这些边及跟这些边相连的顶点不构成图 g 的完全子图，最后图 g 分解成独立的“较小”的无向图（即团集）的简单并。转为 $ii)$ 讨论的情形。

结论：这个近似算法可能达到的近似度具有如下形式：$\forall g\in D_{II}$，

$$R_\sigma(g)=\frac{\max\{\eta(g_1),\eta(g_2),\cdots,\eta(g_n)\}}{\eta(g_i)}=\frac{OPT(g)}{M},\quad M\in Z^+,且OPT(g)\geq M。$$

15．15

证明练习 15.14 中给出的查找最大团集问题的启发式算法的近似度是无界的。

证明：反证法：

假定该启发式算法的近似度是有界的，

则 $\exists A>0, A为实数，$ 取 $M=[A]+1, \forall g\in D_{II}, |R_\sigma(g)|\leq A<M<+\infty$

由于 $g \in D_{II}$ 的任意性，构造 g 如下：设 g_1 是一个具有 $6M$ 个顶点的无向完全图，g_2 是具有 3 个顶点的无向完全图，取 $g = g_1 \cup g_2$,显然 $g \in D_{II}$，且由于 σ 是近似算法，∴不可能输出的都是具有 $OPT(g)$ 个顶点的团集,故而：

$$R_\sigma(g) = \frac{6M}{3} = 2M < M$$

导出矛盾！

4．P256　15.16 给出着色问题的一个近似算法:找出给一个无向图着色,使得相邻顶点着不同颜色的最少颜色数。证明或否定该算法的近似度是有界的

算法思想：对无向图进行分类讨论

算法：COLORING

输入：无向图 G(V,E)

输出：着色的颜色数

```
if V=0 then   return   0   end if
if E=0 then   return   1   end if
if   G 是二分图  return   2   end if
else   return 4
end if
end   COLORING
```

证明：在上述算法 A 中，

在 V=0，E=0，　G 是二分图情况都属于最优解，所以其 RA(I)=1;

而只有 else 语句不是最优解，因为在 else 语句中出现的情况不是 3 可着色就是 4 可着色的，任何一个图形并不都满足 3 可着色，3 着色是 NP 完全问题,但是任何一个平面图 G 都是 4 可着色的，所以这时我们返回 4，即

A(I)=4,　　OPT(I)=3

RA(I)=A(I)/OPT(I)=4/3

所以 1<=RA(I)<=4/3

该算法的近似度是有界的得证

5．P256　15.21

证明只需找出一个反例即可。

例 X = { 1, 2, 3, 4, 5, 6, 7, 8}

子集族 F = {　{ 1, 2 }　{ 2, 3, 4, 5, 6, }　{ 6, 7, 8 }　{ 1, 2, 3, 4, }　{ 5, 6 , 7, 8, }　}

已知最小覆盖只需两个即　{ 1, 2, 3, 4, } { 5, 6 , 7, 8, }　。但按照算法则先选择交集个数最大的子集{ 2, 3, 4, 5, 6, }，然后再在其余的选择两个比如{ 1, 2 }，{ 6, 7, 8 } 或者{ 1, 2, 3, 4, } { 5, 6 , 7, 8, }。该题给出的算法对例子的最小覆盖个数是 3 个。所以不总是产生最小覆盖。