第一章

分析 1-1

$3n^2+10n=O\left(n^2\right)$

$\frac{n^2}{10}+2^n=O\left(2^n\right)$

$21+\frac{1}{n}=O(1)$

$\log n^3=3\log n=O(\log n)$

$10\log 3^n=n\cdot 10\log 3=O(n)$

分析 1-4

(1) 设新输机器能解决规模为n'，则有$3\cdot 2^{n'}=64\cdot 3\cdot 2^n$，所以$n'=n+6$

(2) 设新输机器能解决规模为n'，则有$(n')^2=64\cdot n^2$，所以$n'=8n$

(3) 能解任意规模问题。

分析 1-5

设新输机器能解决规模为n'：

(1) 复杂性为n 时，$n'=100n$

(2) 复杂性为n^2 时，$n'=10n$

(3) 复杂性为n^3 时，$n'=\sqrt[3]{100}n$

(4) 复杂性为$n!$ 时，$n'\approx n$

分析 1-6

(2) $f(n)=O(g(n))$

(4) $f(n)=\Omega(g(n))$

(6) $f(n)=\theta(g(n))$

(8) $f(n)=O(g(n))$

实现 1-2

考虑采用递归的方法枚举。开一个 visit 数组代表当前字符是否被使用过。递归需要的参数是当前深度，最大深度，当前深度是否能够达到最大深度（字典序已经更大时不允许超出）。使用递归的方式统计有多少字符串满足条件。该做法可能不够快速有效。

实现 1-3

```
int check(int x){
    int cnt=0;
    for (int i=2;i*i<=x;++i){
        while (x%i==0){
            ++cnt;
            x/=i;
        }
    }
    if (x!=1) ++cnt;
    return cnt;
}

int main(){
    int ans=0,ansx=-1,ret;
    for (int i=a;i<=b;++i){
        ret=check(i);
        if (ret>ans){
            ans=ret;
            ansx=i;
        }
    }
}
```

第二章

分析 2-2

(1) 错误，存在死循环
(2) 错误，存在死循环
(3) 错误，答案错误
(4) 错误，死循环
(5) 正确。等价于 C++中的 upper_bound
(6) 错误，答案错误
(7) 错误，存在死循环

分析 2-3

```
while (l<=r){
    mid=(l+r)/2;
    if (x==mid){
        l=r=mid;
        break;
    }
    if (x<mid) l=mid+1;
    else r=mid-1;
}
```

分析 2-4

考虑将问题化归成对等规模的两数分治乘问题。将 n 分成每段长为 m，共 $\frac{n}{m}$ 段，则每段乘法复杂度为 $O\left(m^{\log 3}\right)$ ，总复杂度为：

$$O\left(\frac{n}{m}\cdot m^{\log 3}\right)=O\left(n\cdot m^{\log 3-1}\right)=O\left(nm^{\log\frac{3}{2}}\right)$$

分析 2-6

考虑将问题划归成 Strassen 算法能解决的问题。具体方法题干中已经自己说明白了，将一个 $n\cdot n$ 的矩阵转化为 $m\cdot m$ 个 $2^k\cdot 2^k$ 的子矩阵，则需要计算 m^3 个子矩

阵问题，每个子矩阵问题复杂度为 Strassen 的 $O\left(\left(2^k\right)^{\log 7}\right)=O\left(7^k\right)$ ，则总复杂度为 $O\left(m^3\cdot 7^k\right)$ 。

分析 2-7

$P(x)=\prod_{i=1}^{d}x-n_i$ ，采用分治法解决问题，则 $T(d)=\begin{cases}O(1),d=1\\2T\left(\frac{d}{2}\right)+\frac{d}{2}\log\left(\frac{d}{2}\right),d>1\end{cases}$

令 $2^k=d$ ，有 $\frac{T\left(2^k\right)}{2^k}=\frac{T\left(2^{k-1}\right)}{2^{k-1}}+\frac{k-1}{2}$ ，相消有 $T\left(2^k\right)=2^{k-2}k(k-1)$

则 $T(d)=O\left(d\log^2 d\right)$ 。

分析 2-16

(1) 能，代码见下
(2) 能，代码见下
(3) 不能

```
int cnt;
char pre[maxn];
char in[maxn];
char post[maxn];

void solve_post(int in_L,int in_R,int pre_L){
    if (in_L==in_R){
        post[cnt++]=in[in_L];
        return;
    }
    int mid=search_in(in_L,pre[pre_L],in_R);
    if (mid!=in_L){
        solve_post(in_L,mid-1,pre_L+1);
    }
    if (mid!=in_R){
        solve_post(mid+1,in_R,pre_L+mid-in_L+1);
    }
    post[cnt++]=in[mid];
}

```

```
void solve_pre(int in_L,int in_R,int post_L){
    if (in_L==in_R){
        post[cnt++]=in[in_L];
        return;
    }
    int mid=search_in(in_L,pre[pre_L],in_R);
    post[cnt++]=in[mid];
    if (mid!=in_L){
        solve_pre(in_L,mid-1,post_L);
    }
    if (mid!=in_R){
        solve_pre(mid+1,in_R,post_L+mid-in_L);
    }
}
```

实现 2-1

仔细观察书上代码，使用分治的思想解决问题。发现实现正确的前提是多重集合有序。既然如此，则有更简单的做法解决，$O(n)$ 扫一遍，记录当前数和当前数出现的次数即可。

反之，若不保证有序，则可用$O(n)$时间复杂度，$O(1)$ 辅助空间找到绝对众数。所谓绝对众数是指出现频率高于$\frac{1}{2}$ 的数。算法使用的是打擂台的思想。这是 leetcode 经典题，刷过一两道的都做过。

实现 2-2

考虑采用分治的思想解决问题。如果能够构造出一种起点和终点在旋转 90 度后能相互可达的方法，就可以使用分治的思想构造出全部的路径。时间复杂度为$O(n^2)$ 。

实现 2-3

根据题意，有 $f(n)=\begin{cases}1,n=1\\ 1+\sum_{i=1}^{\frac{n}{2}} f(i),n>1\end{cases}$，考虑采用记忆化搜索方法解决。

```
long long vis[maxn];
```

```
2
3  void solve(int n){
4      if (vis[n]!=-1) return vis[n];
5      long long ans=0;
6      for (int i=1;i<=n/2;++i){
7          ans+=solve(i);
8      }
9      return vis[n]=ans;
10 }
```

实现 2-11

书上代码的时间复杂度好到解决题目所给问题规模数据。取:

$$n=2^3\cdot 3\cdot 5\cdot 7\cdot 11\cdot 13\cdot 17\cdot 19\cdot 23=892371480$$

或

$$n=2^{30}=1073741824$$

则实测代码无法在可以忍受的时间复杂度内完成。考虑采用数学方法计算此问题。

将 n 进行质因数分解，统计每一个质因数的总数，则总答案只与这些总数有关，无序关心这些数的具体数值。且答案为 2 的幂次乘组合数的积。

补充 2-1

```
1  int L=0,R=n-1,mid,ans=-1;
2  while (L<=R){
3      mid=(L+R)/2;
4      if (T[mid]==mid){
5          ans=mid;
6          break;
7      }
8      if (T[mid]<mid){
9          L=mid+1;
10     }
11     else{
12         R=mid-1;
13     }
14 }
```

补充 2-2

没想到最坏 3n/2-2 的做法，这是最坏 2n-2，平均 3n/2-2 的做法。

```
int main(){
    int ma=a[0],mi=a[0];
    for (int i=1;i<n;++i){
        if (a[i]>ma) ma=a[i];
        else if (a[i]<mi) mi=a[i];
    }
}
```

补充 3

考虑使用分治的方法在 $n+\lceil \log_2 n \rceil$ 次比较内找到第二大数，根据 Randomized Select 的性质可知第一大数也已经被确定。该复杂度同样为平均值，并非最坏值。

第三章

补充 1（part 1）

$$LCS\left(\{b,a,a,b,a,b,a,b\},\{a,b,a,b,b,a,b,b,a\}\right)=\{a,a,b,a,b,a\}$$

补充 2（part 1）

$$LCS\left(X,Y\right)=\{B,C,A,B\}$$

分析 3-3

题干没有明确 x 是正整数还是正实数。若考虑正实数这显然是一个贪心问题。若为正整数，则这是一道典型的完全背包问题，代码如下：

```
for (int i=0;i<=n;++i){
    for (int j=0;j<=b;++j){
        dp[i][j]=0;
    }
}
for (int i=1;i<=n;++i){
    for (int j=a[i];j<=b;++j){
        dp[i][j]=max(dp[i][j],dp[i-1][j-a[i]]+val[i]);
    }
}
```

分析 3-4

这是一个有多重限制因素的 0-1 背包问题，代码如下：

```
for (int i=1;i<=n;++i){
    for (int j=w1[i];j<=max_w1;++j){
        for (int k=w2[i];j<=max_w2;++k){
            dp[i][j][k]=max(dp[i][j][k],dp[i-1][j-w1[i]][k-w2[i]]+val[i]);
        }
    }
}
```

实现 3-1

Comment：不说数据范围的题目都是在耍流氓。对于标答做法，如果给予很大的 $\max\{\max\{a_i\},\max\{b_i\}\}$ ，而 n 很小，则显然是不合适的。甚至复杂度效果还不如二进制枚举。

```
for (int i=1;i<=n;++i){
    for (int j=0;j<=sum;++j){
        for (int k=0;k<=sum;++k){
            if (j-a[i]>=0&&dp[i-1][j-a[i]][k]){
                dp[i][j][k]=true;
            }
            else if (k-b[i]>=0&&dp[i-1][j][k-b[i]]){
                dp[i][j][k]=true;
            }
            else{
                dp[i][j][k]=false;
            }
        }
    }
}
```

补充 3-1-1（part 2）

详见实验报告。

补充 3-1-2（part 2）

```
int main(){
    int n,ans;
    while (scanf("%d",&n),n){
        printf("%d\nTimes:",n);
        ans=1;
        while(n>3){
            n=(int)(ceil(1.0*n/3.0));
            ++ans;
        }
        printf("%d\n",ans);
    }
}
```

补充 2-2（part 2）

$N_1=\{J_1,J_3,J_4,J_6\},N_2=\{J_2,J_5\}$

根据 Johnson 调度法则，最优调度为 J_1,J_4,J_6,J_3,J_2,J_5

算得最少完成时间为 41 分钟。

补充 2-3（part 2）

n\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	3	3	3	3	3	3	3	3
2	0	0	5	5	5	8	8	8	8	8	8
3	0	0	5	5	5	8	8	8	8	12	12

第四章

分析 4-3

一般的，若 $a_n=\begin{cases}1, n\le 1\\ a_{n-1}+a_{n-2}, n>1\end{cases}=\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n-\left(\frac{1-\sqrt{5}}{2}\right)^n\right)$ ，

则有： $a_n\le\sum_{i=0}^{n-1}a_i\le a_{n+1}$

由哈夫曼贪心选择性质可知，每次哈弗曼树合并都会将 a_n 和包含 $\{a_0,a_1,\cdots,a_{n-1}\}$ 的子树进行合并。

由此可得一种哈夫曼编码为：$a_n=\begin{cases}0000\cdots0 & ,n=0\\ 0000\cdots1 & ,n>0\end{cases}$ ，且所有数字对应哈夫曼编码中前导 0 的个数恰为 $N-n$ 个。

实现 4-6

贪心选择性质：

设 $\{g_1,g_2,\cdots,g_{u-1},g_u,\cdots,g_n\}$ 是一个最优服务次序，其中 $g_u=\min\{g_1,g_2,\cdots,g_n\}$，则 $\{g_u,g_2,\cdots,g_{u-1},g_1,\cdots,g_n\}$ 同样是一个最优服务次序。

证明：

$u=1$ 时，最优服务次序未发生改变。

$u\ne1$ 时，$T=\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i}g_j=\frac{1}{n}\sum_{i=1}^{n}(n-i+1)\cdot g_i=\frac{1}{n}\left(n\cdot g_1+(n-1)\cdot g_2+\cdots+g_n\right)$

$$T-T'=\frac{1}{n}\left(n\cdot g_1+(u-1)g_u-n\cdot g_u-(u-1)g_1\right)=\left(\frac{n-u+1}{n}\right)(g_1-g_u)\ge0$$

此时可知 $\{g_u,g_2,\cdots,g_{u-1},g_1,\cdots,g_n\}$ 同样是一个最优服务次序。

最优子结构性质：

设 $\{g_1,g_2,g_3\cdots g_n\}$ 是一个 n 个顾客的最优服务次序，则 $\{g_2,g_3\cdots g_n\}$ 是一个除去 g1 顾客的 n-1 个顾客的最优服务次序。

证明（反证法）：

设$\{g_2, g_3 \cdots g_n\}$不是一个 n-1 个顾客的最优服务次序，即存在至少一个不一样的序列$\{u_2, u_3 \cdots u_n\}$，满足该序列是 g 序列的重排，且$T_u' > T_g'$ 。

构造新的服务次序$\{g_1, u_2, u_3 \cdots u_n\}$，计算其与原 n 个顾客最优服务次序：$T_u = n \cdot g_1 + T_u' < n \cdot g_1 + T_g' = T_g$。

由此可知，$\{g_1, g_2, g_3 \cdots g_n\}$不是 n 个顾客的最优服务次序，与已知矛盾。

因此，$\{g_2, g_3 \cdots g_n\}$必然是一个 n-1 个顾客的最优服务次序。

实现 4-9

贪心选择性质：

设$\{g_1, g_2, g_3 \cdots g_m\}$是一种加油次数最少的加油方式（已排序），那么必然有$g_1 \le n$且$g_2 - g_1 \le n$ 。

若g_1已经是最远不加油可到达的加油站，那么已经符合贪心选择性质。

若g_1不是最远不加油可到达的加油站，即存在g_u满足$g_1 < g_u \le n$，则必然有$g_u \le n$且$g_2 - g_u \le n$，则贪心选择结果$\{g_u, g_2, g_3 \cdots g_m\}$也是一种加油次数最少的加油方式。

最优子结构性质：

设$\{g_1, g_2, g_3 \cdots g_m\}$是一个从原点出发的最少加油序列（已排序），则$\{g_2 - g_1, g_3 - g_1 \cdots g_m - g_1\}$是一个以原点为起点的最少加油序列。

证明（反证法）

设存在$\{u_2 - g_1, u_3 - g_1 \cdots u_{m'} - g_1\}$是一个从原点出发加油次数为$m'-1$次的更优的加油序列，其中$m' < m$。

由于$0 \le u_2 - g_1 \le n$，构造序列$\{g_1, u_2, u_3 \cdots u_{m'}\}$是一个从原点出发加油次数为$m'$次的加油方法。与已知为最少加油次数为 m 矛盾。

因此$\{g_2 - g_1, g_3 - g_1 \cdots g_m - g_1\}$必然是一个以原点为起点的最少加油序列。

补充 2

这是一个很有意思的问题，如果每个任务的完成时间不是定长 1，则这个问题将转化为动态规划问题进行求解。但如果每个任务都只需要一个单位长度即可完成，则可用简单的贪心算法实现。

```
vector<int> a[maxm];

int main(){
    int n,t1,t2,m=-1,ans=0;
    scanf("%d",&n);
    for (int i=0;i<n;++i){
        scanf("%d%d",&t1,&t2);
        a[t1].push_back(t2);
        m=max(m,t2);
    }
    priority_queue<int> que;
    for (int i=0;i<m;++i){
        if (!que.empty()) que.pop();
        for (auto &elem:a[i]){
            que.push(elem);
        }
    }
    while (!que.empty()){
        ans+=que.top();
        que.pop();
    }
    printf("%d\n",ans);
}
```

第五章

实现 5-4

```
int n,ans;
bool vis[maxn];
int P[maxn][maxn],Q[maxn][maxn];

void dfs(int p,int sum){
    if (p==n){
        ans=max(ans,sum);
        return;
    }
    for (int i=0;i<n;++i){
        if (!vis[i]){
            vis[i]=true;
            dfs(p+1,sum+P[p][i]*Q[i][p]);
            vis[i]=false;
        }
    }
}

int main(){
    scanf("%d",&n);
    for (int i=0;i<n;++i){
        for (int j=0;j<n;++j){
            scanf("%d",&P[i][j]);
        }
    }
    for (int i=0;i<n;++i){
        for (int j=0;j<n;++j){
            scanf("%d",&Q[i][j]);
        }
    }
    ans=0;
    memset(vis,false,sizeof(vis));
    dfs(0,0);
    printf("%d\n",ans);
    return 0;
}
```

实现 5-6

```cpp
int n,ansx;
set<int> cur[25],ans[25];

void dfs(int x){
    if (x-1>ansx){
        ansx=x-1;
        for (int i=0;i<n;++i){
            ans[i]=cur[i];
        }
    }
    for (int i=0;i<n;++i){
        bool ok=true;
        for (auto &elem:cur[i]){
            if (elem*2==x) continue;
            if (cur[i].find(x-elem)!=cur[i].end()){
                ok=false;
                break;
            }
        }
        if (ok){
            cur[i].insert(x);
            dfs(x+1);
            cur[i].erase(x);
        }
    }
}

int main(){
    scanf("%d",&n);
    ansx=0;
    dfs(1);
    printf("%d\n",ansx);
    for (int i=0;i<n;++i){
        for (auto &elem:ans[i]){
            printf("%d ",elem);
        }
        putchar('\n');
    }
    return 0;
}
```

复杂度超高，最多支持 n=3 的规模。n=4 时答案不小于 58。

实现 5-21 5-22 5-23

Comment：你听听这题面描述在说什么鬼话：“*函数的参数包括节点可行性判定函数和上界函数等必要函数*”，什么是参数？不过，除了严重的题面描述问题，书上代码写得不错，但对面向对象程序设计要求较高，实用性着实不强。圆排列问题蛮有意思的，重写一下练习练习。

```
1  const int maxn=20;
2  const int inf=0x3f3f3f3f;
3  double r[maxn];
4  bool vis[maxn];
5  int n;
6  double ans;
7
8  void dfs(int p,double x,double h){
9      if (p==n){
10         ans=min(ans,x+h);
11         return;
12     }
13     for (int i=0;i<n;++i){
14         if (vis[i]) continue;
15         vis[i]=true;
16         dfs(p+1,x+2*sqrt(h*r[i]),r[i]);
17         vis[i]=false;
18     }
19 }
20
21 int main(){
22     scanf("%d",&n);
23     for (int i=0;i<n;++i){
24         scanf("%lf",&r[i]);
25     }
26     ans=1.0*inf;
27     memset(vis,false,sizeof(vis));
28     for (int i=0;i<n;++i){
29         vis[i]=true;
30         dfs(1,r[i],r[i]);
31         vis[i]=false;
32     }
33     printf("%f\n",ans);
34     return 0;
35 }
```

其中第 16 行结论是简单几何推导得到。

第六章

分析 6-1

分支限界法类似于回溯法，也是一种在问题的解空间树 T 上搜索问题解的算法。但在一般情况下，分支限界法与回溯法的求解目标不同。回溯法的求解目标是找出 T 中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解，即在某种意义下的最优解。

Comment：书上的代码到第六章的时候突然变烂了，可读性极差，恕不能在这段时间内理解透彻。
垃圾的算法=缩进错误+变量名乱取+封装水平不如不会封装+除了有符号数字外没有任何兼容性的模板类+多此一举无法体现算法优越性的实现方法。

——17122490 秦敏浩