第四次作业

算法设计题

4-1 会场安排问题

问题表达

如果按照图着色问题处理，则最开始要将k个任务根据是否相容构建成一个图，构建此图需要$O(n^2)$复杂度，既不适合将其处理，将每个活动令为$[s_i,e_i]$表示第i个活动的开始时间与结束时间。由于k个活动最多需要k个会场，令$p[k][2]$表示第k个会场最后一个活动的开始结束时间，则对于任务集合S包含n个任务，去除第j个任务，各会场分配信息为p，令计算会场分配个数的函数为$Count$，对于此问题，即求$Count(S,p)$的值，其递推式为:

$$\min_{j\in S}\{Count(S-\{j\},p)\},p[i][0]=s_j,\ p[i][1]=e_j,$$

$$约束条件:i=\min\{k,p[k][1]\le s_j\}$$

即每次取得数组p符合相容条件，经过递推式集合S规模减一，变成$S-\{j\}$。最开始初始化$p[k][2]=0$，$S=\{1,2,3,\dots,k\}$，即可求出最后最小会场数。

对于边界条件，即S为空集，则计算p中第二维度有多少不为空即为其会场值。

贪心选择

由于每次对于S集合的任务选择为其动态性来源，为了减除动态性，考虑到任务越先完成，越先能够尽可能多地完成后续任务，于是对于每次活动的选择，总是贪心地选取结束时间最早的活动，并将其安排在相容的会场中，对于相容的会场，选取序号靠前的相容会场，即$i=\min\{k,p[k][1]\le s_j\}$，即可最终求出所需最少的会场。

最优子结构

采取归纳法证明，对于S集合满足最优选择，则对新添加一个活动k，若满足最优结构，若是k活动满足与其他安排的活动相容条件，则会场数维持原样且k活动使用的会场结束时间更新，不满足则需要新添加一个会场，即得到最优解，即此问题满足最优子结构。

时间复杂度

由于贪心选择只需要遍历排好序的各个活动的开始结束时间，时间复杂度为$O(n)$，所以时间复杂度主要在排序算法上，由于排序算法最快的时间复杂度为$O(n\log n)$，即最终时间复杂度为$O(n\log n)$。

伪代码

```
int venue_arrangement(){
    int p[][2];//记录会场活动开始结束时间
    int t;//记录最后一个正在进行的活动
    int count=0;//记录用了几个会场

    t=0;//第一个活动进行，且为最后一个
    count++;
    Read(p);//读取输入数据，O(n)
    Sort(p);//排序，复杂度O(n*log(n))

    //贪心选择，复杂度O(n)
    for(int i=1;i<n;i++){
        if(p[t][1]>p[i][0]){
```

```
            count++;
        }else{
            t++;
        }
    }

    Write(count);//写入，复杂度O(1)
    return count;
}
```

4-2 最优合并问题

问题表达

对于此问题设集合$S=\{s_1,s_2,s_3,\ldots,s_k\}$表示为$k$个序列集合，$l_i$表示为$s_i$序列的长度，$Num_min(S)$表示$S$序列合并所需比较最小值，$Num_max(S)$表示$S$序列合并所需比较最大值。则其递推方程为

$$Num\_min(S)=\min_{s_i,s_j\in S}\{l_i+l_j-1+Num\_min(S-\{s_i,s_j\}+\{s_i+s_j\})\}$$
$$Num\_max(S)=\max_{s_i,s_j\in S}\{l_i+l_j-1+Num\_max(S-\{s_i,s_j\}+\{s_i+s_j\})\}$$

边界条件为$Num_min(\{s_i,s_j\})=Num_max(\{s_i,s_j\})=l_i+l_j-1$。每次选取两个序列组成一个新序列并将合并两个新序列比较个数加上，即可得出最大最小值。此时的动态性为每次选择的合并序列。

贪心选择

对于每次选择的待合并的序列，为了消除动态性，即使得i,j固定，则对于总体比较次数更少，则贪心选择l_i+l_j-1最小的序列进行合并，同理为了选取总体比较次数最多，则贪心选择l_i+l_j-1值最大的序列进行合并。即对序列的长度进行由小到大排序，对于比较次数最少的，每次选取序列长度最短的两个序列进行合并，对于比较次数最多的，每次选取序列长度最长的两个序列进行合并。

最优子结构

对于$Num_min(S-\{s_i,s_j\}+\{s_i+s_j\})\}$，若其满足最小值，由于$l_i+l_j-1$最小，则其相加亦为最小值，即加法满足最优子结构，即此问题，满足最优子结构，同理对于求最大值亦然。

时间复杂度

由于对序列长度由小到大排序时间复杂度需要$O(n\log n)$，对于排好序的数组的存储，用完全二叉树存储，对于合并后序列长为i，为了维持其升序与降序，由于原序列已经排好序，并将新合并的节点插入，对于二叉树，则需要$\log i$复杂度。由于找出最大最小值要遍历n遍，即总共需要$O(n\log n+\sum_{i=1}^{n}\log i)$时间复杂度，化简得时间复杂度为$O(n\log n)$。

伪代码

```
int num_select()
{//次数选择最优情况与最差情况
    int min=0;
    int max=0;
    int *a, *b;
    int n;
    n = Read(a);//读取数据传入a，并将序列长度设为n；时间复杂度为O(n)
    Sort(a);//从小到大排序，时间复杂度为O(nlogn)，并将a用树状结构保存
    b = Reverse(a, n);//将a倒序即将数组从大到小排列并设为b，时间复杂度为O(n)
    for(m=0;m<n-1;m++) { //n个数，需要排序n-1次
```

```
        min += binary_insert_min(a);//将最小的两个值相加合并成一个新的节点并插入此树，
                                    //最后返回最小两个值相加减一的值,时间复杂度为log n
        max += binary_insert_max(b);//将最大的两个值相加合并成一个新的节点并插入此树，
                                    //最后返回最大两个值相加减一的值,时间复杂度为log n
    }
    Write(max, min);//写出数据
    return 0;
}

```

4-9 汽车加油问题

问题表达

令$num(n,i)$表示汽车在第i个加油站还能行驶$n$km所需最小加油次数，令起点与终点距离为diskm，第i个加油站距第$i+1$个加油站为$a[i+1]$。由于汽车到达第$i+1$个加油站可选择加油与不加油，对于加油，其还能行驶n_{max}km，不加油则只能行驶$n-a[i+1]$km。则其递推式为

$$num(n,i)=\min\{num(n-a[i+1],i+1),num(n_{max},i+1)+1\},i\in\{0,1,2,3,\dots,k\}$$

对于边界条件，有两种，即不可达与达到目的地，如果不可达，即$n<a[i+1]$，此时令$num(n,i)=+\infty$，如果到达目的地，即对于到达最后一个加油站k，其能最终行驶到终点，即$n\geq a[k+1]$，此时$num(n,k)=0$。

贪心选择

当汽车到达第i个加油站时，如果剩下的油够行驶到第$i+1$个加油站，为了使得加油次数最小，则贪心选择不加油，如果不够则进行加油。即上述递推式改为

$$num(n,i)=\begin{cases}num(n-a[i+1],i+1), & if\ n-a[i+1]\geq a[i+2]\\ num(n_{max},i+1)+1, & if\ n-a[i+1]<a[i+2]\end{cases}$$

最优子结构

对于前i个加油站，如果其为最优，则对与第$i+1$个加油站，如果行驶到第$i+2$个加油站，则不需要加油，否则必须加油，即上诉贪心选择符合最优子结构。

贪心选择正确性

如果到达第i个加油站，其剩余油量不够到第$i+1$个加油站，即$n-a[i]<a[i+1]$必须加油，贪心选择正确。假设剩余油量够行驶到第$j-1$个加油站，但不够行驶到第j个加油站，即$n-a[i]<a[j]$，如果在第i到$j-1$个加油站加了油，则至少加一次油，如果是贪心选择，则只会在第$j-1$个加油站加油，只加一次油，即此时贪心选择达到最优解。

时间复杂度

由于此时只需对各加油站与上一个加油站的距离遍历一遍，就能得到最优解，即时间复杂度为$O(n)$。

伪代码

```
int main()
{
    int n_max,n;
    int a[100];
    int num=0,s=n_max;
```

```
    Read(a, n_max, n);//读取数据，时间复杂度O(n)
    for(int i=0;i<=n;i++){//循环n次，时间复杂度为O(n)
        if(a[i]>n_max){//无法到达下一个加油站
             cout<<"No Solution";
             return 0;
         }
         if(s-a[i]>=0){
             s-=a[i];//剩余行驶距离
         }
         else{//剩余距离不够行驶到下一个加油站则加油
             num++;
             s=n_max-a[i];
         }
    }
    Write(num);//写入数据
    return 0;
}
```

算法实现题

4-4 磁盘文件最优存储问题

问题表达

此问题即可表达为找出一个排列，使得检索n个文件的期望值最小，即

$$\min\{\sum_{1\le i\le j\le n} p_i p_j d(i,j)\}$$

贪心选择

由于文件的不同位置为其动态性来源，则对于其中一项$p_i p_j d(i,j)$为了使其更小，由于$p_i * p_j$值固定，即只能改$d(i,j)$的值，由于被检索到概率大的文件需要使得$d(i,j)$更可能小，即被检索概率较大的文件需要离所有文件都较近，即尽量排在所有文件中间，对于概率小的则尽可能排在两侧。具体步骤为将文件按照检索概率进行排序，概率最大的排在最中间的磁道，概率奇数大的按照概率由小到大排在中间位置的左侧，概率偶数大的按照概率由大到小排列在中间位置右侧，保证由中间到边缘检索概率由大到小排列。

贪心选择正确性

令序号为k的文件距离中心磁道为d_k，对于距离中心为d_i，d_j的磁道，即$d_i < d_j$，$p_i > p_j$，则总时间为$t(i,j) = \sum_{1\le i\le j\le n} p_i p_j d(i,j)$。若将$i,j$交换位置，则总时间为$t(j,i)$，由于其他位置没有改变，其值亦未变，改变的是与$i,j$相关的值。即

$$t(j,i) - t(i,j) = \sum_{m,m\neq i,j} p_j p_m d'(j,m) + \sum_{m,m\neq i,j} p_i p_m d'(i,m) - \sum_{m,m\neq i,j} p_i p_m d(i,m) - \sum_{m,m\neq i,j} p_j p_m d(j,m)$$

由于i,j交换位置，即$d'(j,m) = d(i,m), d'(i,m) = d(j,m)$，即上式化简为

$$t(j,i) - t(i,j) = (p_i - p_j) \sum_{m,m\neq i,j} p_m (d(j,m) - d(i,m))$$

由于$p_i > p_j$，考虑到$\sum_{m,m\neq i,j} p_m (d(j,m) - d(i,m))$，可将其分成三个集合，即$m_1, m_2, m_3$，分别对应$d_m < d_i, d_i < d_m < d_j, d_j < d_m$。则上式可化为

$$\sum_{m_1} p_m (d_j - d_i) + \sum_{m_2} p_m (|d_j + d_i - 2d_m|) + \sum_{m_3} p_m (d_j - d_i)$$
$$= (d_j - d_i) - \sum_{m_2} p_m (d_j - d_i - |d_j + d_i - 2d_m|)$$

由于当$m \in m_2$时，$\max\{d_j - d_i - |d_j + d_i - 2d_m|\} = d_j - d_i$，且$\sum_{m_2} p_m \leq 1$，即$\sum_{m_2} p_m(d_j - d_i - |d_j + d_i - 2d_m|) \leq d_j - d_i$，即交换后$t(j,i) - t(i,j) \geq 0$，时间至少不会变短。此贪心选择能够得到最优解。

时间复杂度

由于此算法排序平均时间复杂度需要$O(n \log n)$，遍历需要$O(n)$，计算总时间期望需要$O(n^2)$，即总平均时间复杂度为$O(n^2)$。

代码

见附件，直接在附件文件夹运行$greedy_disk_storage.exe$即可。