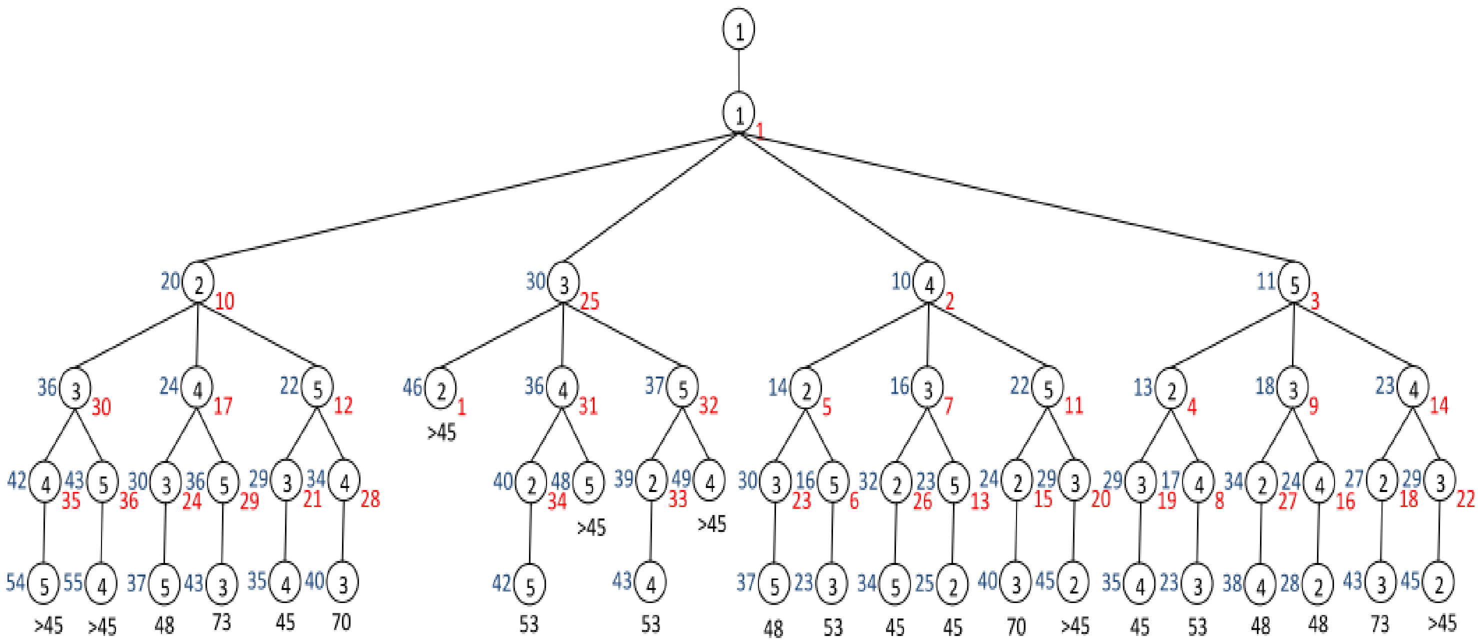
一、 习题七

1.解：（往届同学解答）

1. 假设旅行商问题的邻接矩阵如图 1 所示，试用优先队列式分枝限界算法给出最短环游。画出解空间树的搜索图，并说明搜索过程。
   解空间树搜索图如下



为了方便看清路径，圆圈内为顶点序号，未标结点号。左侧蓝色数字为该路径到该结点的总路程，右下红色数字为该结点的搜索序数。从顶点 1 开始搜索，将其四个儿子结点放入队列。然后优先访问队列中当前路程最小的结点，再将它的儿子放入队列。然后重复上述过程，优先访问队列中当前路程最小的结点，将其儿子放入队列。

第一个被访问到的叶子结点为 1-4-2-5-3-1 这条路径，其总路程为 53，记录为当前最短路程。之后若某结点的路程大于最短路程，则不再搜索该结点的子树。若某叶子结点的总路程小于当前最短路程，则更新之。如此搜索，当访问到 1-4-3-5-2-1 这条路径的叶子结点时，其总路程为 45<53，将 45 更新为当前最短路径，继续搜索。

最后得最短环游为 1-2-5-3-4-1、1-4-3-2-5-1、1-4-3-5-2-1、1-5-2-3-4-1，总路程均为 45。

2.解：见讲义第七章第 2 节，要求上机实现。

3 解：往届同学答案，可用贪心算法改善 f 初值;亦可以根据 X.t[]用贪心算法估计在任务 1,2,...，X.length 分派后，该路径总执行时间的上界，与 f 比较来剪枝(程序中用 X.Time 与之比较，并未实现下述算法思想中的限界函数。)

限界函数：将n个任务按照所需时间非递减排序，得到任务序列1，2，3，4……,n，满足时间关系t[1]<t[2]<……<t[n]。将n个任务中的前k个任务分配给当前k个机器，然后将第k+1个任务分配给最早完成已分配任务的机器，依次进行，最后找出这些机器最终分配任务所

需时间最长的，此时间作为分支限界函数，如果一个扩展节点所需的时间超过这个已知的最优值，则删掉以此节点为根的子树。否则更新最优值。

优先级：哪台机器完成当前任务的时间越早，也就是所有机器中最终停机时间越早， 优先级就越高，即被选作最小堆中的堆顶，作为扩展节点。

设task[n]用来记录最优的调度顺序

每个节点具有信息：

{Parent：父亲节点，**Level**：节点所在深度加1，**Ctime**：运行到当前节点所用时间**}**

当**level<=k**时（不能用界限函数来剪枝，也不需要判断优先级），由于机器还未装满（即前面的k个任务其实是同时加入的），可以令Ctime=0，

当**level>k**时（需要界限函数来剪枝，需要判断优先级），Ctime就是运行到当前状态所用的总时间，Ctime作为优先级函数，即从最小堆中选取Ctime最小的节点优先。当找出第一个解节点时，记录此时的Ctime作为目标函数值，以后生成的节点的Ctime大于该目标函数值时，就可以剪掉该节点，如此下去一直到最小堆为空为止。

上述就是最佳调度问题的分支限界算法。

解空间树的节点包括以下信息:

```
Node{
   int Path[n]; //节点对应的解空间树的路径，即到该节点为止的策略记录
   int T[k];//在本策略下的每台机器的运行时间
   int Time; //本策略的总执行时间，为每台机器运行时间的最大值
   int length;//本节点的深度,即当前处理的作业
}
```

算法实现：

```
Proc BestDispatch(int n,int k,int t[]){
  Node Boot,X,P,result; //Boot为根节点，result保存最优解
  int f;//记录当前最优解的执行时间
  f=n*max(t[]); //初始化
  Boot.T[n]={0};
  Boot.Time=0;
  Boot.Path[n]={0};
  Boot.length=0; //初始化根节点
  AddHeap(Boot);//根节点加入堆中，堆中元素按照Time值由小到大排序
  While (!Heap.empty()) do{
    P=DeleteMinHeap();//P为当前优先级最高的点
    for i=1 to k do//扩展P的k个子节点
      X=Newnode(P.Path[],P.T[],P.length+1);
      X.Path[X.length]=i;
      X.T[i]=X.T[i]+t[X.length];
      X.Time=max(X.T[]);
      if X.length==n then//X为叶节点
        if X.Time<f    then//X的执行时间小于已知最优解
          f=X.Time;//将X设为最优解
          result=X;
        end{if}
      else //X为中间节点
        if X.Time<f then
          AddHeap(X);
        end{if}//X的当前执行时间小于已知最优解则加入堆中，否则剪去
      end{if}
    end{for}
  end{while}
end{ BestDispatch }
```

二、

1.解：

设W={$s_1,s_2,..,s_m$}，求W满足条件$\bigcup_{s_i\in W} s_i = A$的子集且是$s_i$不相交。用回溯算法，解向量为<$x_1,x_2,\ldots,x_m$>，$x_i$=0,1，其中$x_i$=1当且仅当是$s_i$∈W。部分向量<$x_1,x_2,\ldots,x_k$>表示已经考虑了对$s_1,s_2,\ldots s_k$的选择。U为当前选择的集合的并集。

```
Eaxtsetcover(U,k)

  {   if k=m+1 then return
```

```
If |U+w(k)|=|U|+|W(k)|   then   ; W(k)与U不相交
    X(k)=1   ;左儿子可行
    If |S+W(k)|=|A|   then
      Print X[]
      Return
    Else
      Eaxtsetcover(U+W(k),k+1)
    End if
  End if
  X(k)=0    ;生成右儿子
  Eaxtsetcover(U,k+1)
}
```

2.解：

设解向量为x[4][4]，约束条件：同行、同列不能相等。回溯算法：

```
Bool Latincheck(int x[4][4],int k,int row,int line)
   {
     for (i=0,i<row,i++) if(x[i][line]==k)return false
     for(i=0,i<line,i++) if(x[row][i]==k)return false
     Retuen true
   }
 Void LatinMartix(int x[4][4],int row,int line)
```

```
{
 if (row==3) && (line==3)
   {for i=0,i<4,i++)
    for(j=0,j<4,j++)
      Cout<<x[i][j]<<" ";
   Return;}
 else
  for (color=1,color<=4,color++)
  {
   if (latinCheck(x,color,row,line))
   {x[row][line]=color;
    if (line<3)latinMartix(x,row,line+1);
    else if(line==3)&&(row!=3)LatinMartix(x,row+1,0);
   }
 }
```

3．解：

设n个人的集合是{1,2,..,n}，n项工作的集合是{1,2,…,n}，每个人恰好1项工作。令解向量为X=<$x_1,x_2,\ldots,x_n$>，x_i=j表示把工作j分配给i人，那么分配成本是C(X)=$\sum_{i=1}^{n} C(i,x_i)$，搜索空间是排列树。部分向量<$x_1,x_2,\ldots,x_k$>表示已经考虑了对人1,2,…,k的工作分配，结点分支的约束条件为：$x_{k+1}\in\{1,2,\ldots,n\}-\{x_1,x_2,\ldots,x_k\}$。可以设立代价函数：

F($x_1,x_2,\ldots,x_k$)=$\sum_{i=1}^{k} C(i,x_i)+\sum_{i=k+1}^{n}\min\{C(i,t)\mid t\in\{1,2,\ldots,n\}-\{x_1,x_2,\ldots,x_k\}\}$，界B是已经得到的最好

可行解的分配成本，如果代价函数大于界，则剪枝。可用优先队列分支限界算法，以F最小优先扩展。根节点有n个儿子x_1=1,2,..n，儿子节点有n-1个儿子x_2=2,3,…n；x_2=1,3,4,…n;….。

算法可参考一.3，节点需增加一个数组work[n],标识已被分配的工作，树高length表示人员编号，队列中每个节点扩展，for i=1 to k 改为for i=1 to n ,但i工作是否可以分配给p.length+1人，需要检查work[i]是否为0，并及时标记已分配。基本照猫画虎就可以了。