目录

[作业1：算法基础与递归 1](#_Toc136981009)

[作业2：分治法作业 2](#_Toc136981010)

[作业3：DFS与回溯法 6](#_Toc136981011)

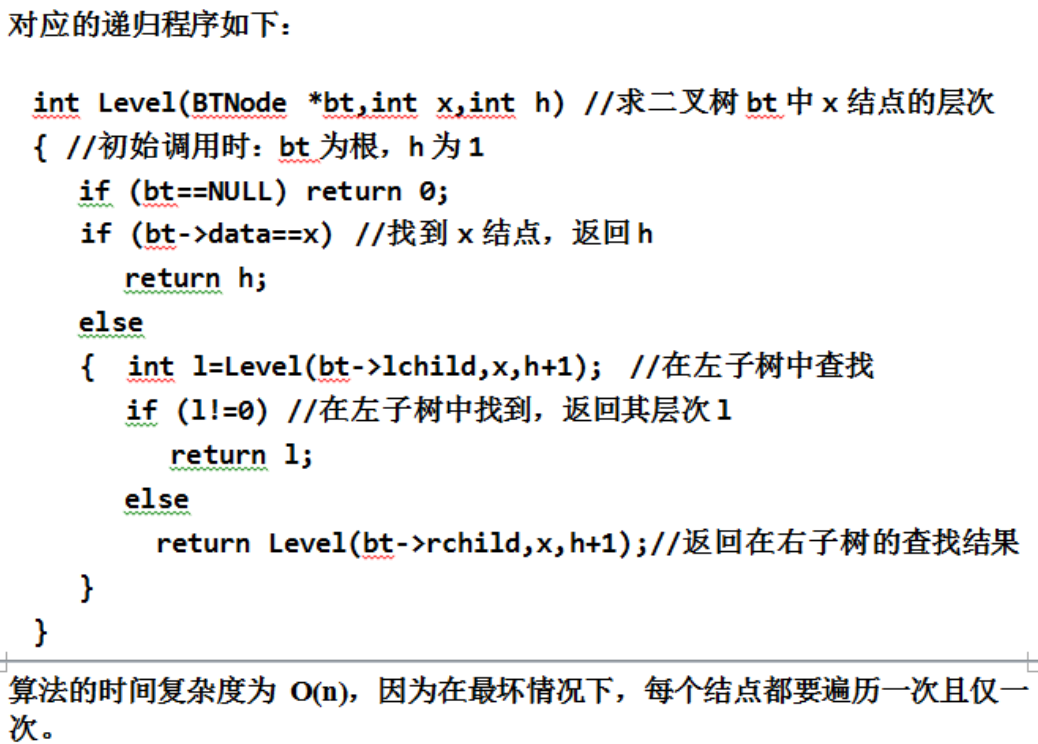
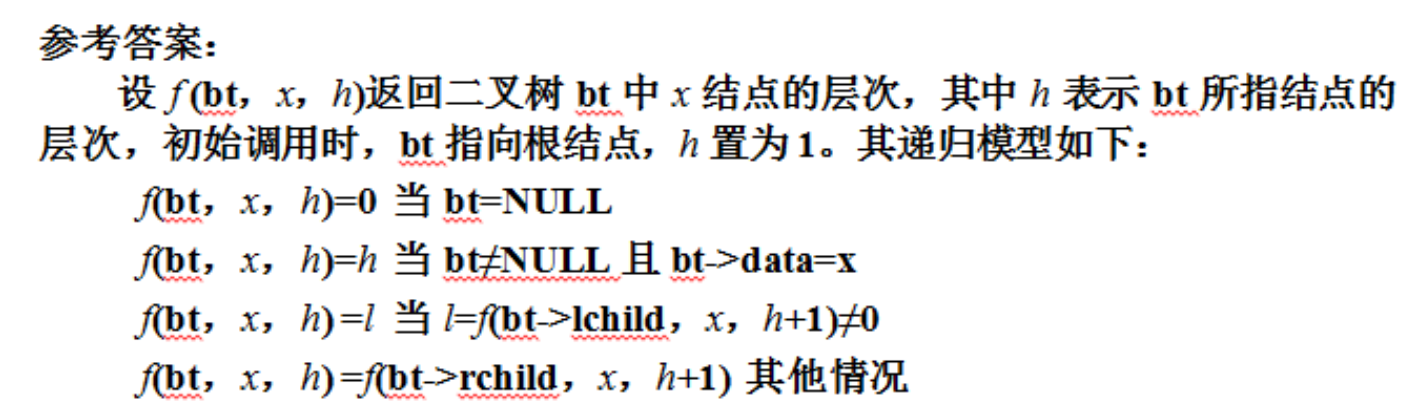
[作业4：BFS与分支限界法 10](#_Toc136981012)

[作业5：贪心法 14](#_Toc136981013)

[作业6：动态规划法 17](#_Toc136981014)

## 作业1：算法基础与递归

### 19. (简答题, 10分)假设二叉树采用二叉链作为存储结构，所有结点值均不相同，设计一个递归算法求值为x的结点的层次（根结点的层次为1），没有找到这样的结点时返回0。分析该算法的时间复杂度（假设二叉树的节点数为n）。



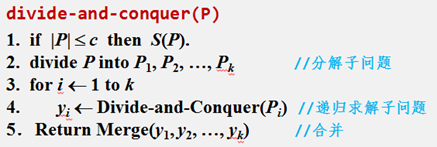
## 作业2：分治法作业

### 6. (简答题, 8分)描述分治算法的一般框架，并说明如何保证分治算法的正确性。

**正确答案：**

参考答案：

分治法包含分解、求解子问题、合并三个步骤，算法框架如下：



正确性保证在于：P的解y可以由y1、y2、…、yk合并得出。

### 7. (简答题, 8分)

### 归并排序算法 mergeSort 的最坏情况时间复杂度是 O(nlogn)，它的最好情况时间复杂度是多少？能说归并排序算法的时间复杂度是 Θ(nlogn) 吗？

**正确答案：**

参考答案：

（1）在归并排序中，不论输入什么实例数组，归并的轮数不会发生变化，都是 logn 轮。归并中比较的次数会发生变化（即两个长n/2的有序序列归并）：

    最坏情况是两个序列是大小交错的，这时需要比较n-1次；

    最好情况是一个序列完全大于（或小于）另一个序列，这时只需要比较n/2次。

差异都是线性的，不改变复杂性的阶。因此最坏情况时间复杂度是 O(nlogn)，最好情况时间复杂度也是O(nlogn)。

（2）根据上述分析，可以说归并排序算法的时间复杂度是 Θ(nlogn)。

### 8. (论述题, 18分)设n个不同的整数按升序存于数组A[1..n]中，设计分治算法求使得A[i]=i的下标i，并分析时间复杂度。

**正确答案：**

算法思路：

由于数组A[i] 是不同整数的升序排列，如果存在A[i] = i，则当k < i 时，A[k] < k; 当k > i 时，A[k] > k. 因此可用折半查找方法。

（1）分解：将原序列A[left…right]一分为二分解成两个子序列A[left…i-1]和A[i+1…right]，其中中位下标i为划分的基准位置，即将整个问题分解为两个子问题。

（2）求解子问题：判断A[i]与i的大小，若A[i]>i，在左边查找；若A[i]<i，则在右边查找；若A[i]=i，则找到并输出i.

（3）合并：由于整个有序序列放在数组中，合并步骤不需要执行任何操作。

算法描述：

int  Search（left，right，A[]）

输入：给定数组A[1...n]及范围left … right, 初始值为1...n.

输出：满足A[i] = i的下标i，或不存在返回－1.

1．  if left > right then return – 1 ; [//未找到](file:///\\xn--rcr501brpd\)。

2．  if left = right and A[mid]!= mid then return – 1;

3．  令mid←⎣(left+right)/2⎦; //取数组中间元素

4．  if A[mid] = mid then  return mid; [//已找到](file:///\\xn--rcr89z1sd\)。

5．  if A[mid] < mid then  return Search（mid+1，right，A）;//在数组右边查找。

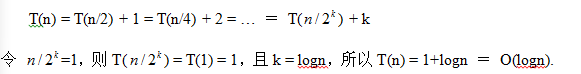
6．  if A[mid] > mid then  return Search（left，mid－1，A）;//在数组左边查找。

算法分析：

设n个元素数组查找的时间为T(n)，根据算法得到时间递推式为：

T(1) = 1， T(n)=T(n/2) + 1

用递推法可解得：



即算法时间复杂度为O(logn).

### 9. (论述题, 18分)**在一个数组中出现次数最多的数称为众数。用分治法设计求解众数问题的算法，并分析时间复杂度。**

**正确答案：**

算法思路：

首先对数组进行快速排序，花费时间O(nlogn)，然后在已排序的数组中查找众数。对于数组A[left, right]应用分治法：

（1）分解：取数组中间位置的元素，然后从中间往左边找到第一个与中间元素不相同元素的位置mid1；同样从中间往右边找到第一个与中间元素不相同元素的位置mid2；得到两个子区间[1…mid1], [mid2…right].

（2）求解子问题：若子序列的长度不超过中间元素的重数，则放弃；否则在子区间上递归查找。

（3）合并：对于两个子区间得到的准众数和中间元素的重数比较，得到重数最大的众数。

算法描述：

int  SearchP（left，right，A[]）

输入：给定已排序的数组A[1...n]及范围left … right, 初始值为1...n.

输出：最大的众数p及它的重数num。.

1．取数组给定区间内的中间位置的元素：midp=A[(left+right)/2]；

2．从中间往左边找到第一个与midp不相同数字的位置mid1；

3．从中间往右边找到第一个与midp不相同数字的位置mid2；

4．统计midp的重数midnum = mid2 – mid1 – 1;

5．若mid1 > midnum，则(lnum,lp)= SearchP（left，mid1，A）；//在左边查找。

6．若right-mid1+1 > midnum，则(rnum,rp)= SearchP（mid2，right, A）;//在右边查找。

7．num = max{lnum, rnum, midnum}，p=num对应的元素;

8．return (p,num).

算法分析：

快速排序算法的时间复杂度为0(nlogn)，设众数查找时间为T（n），它包括分解、子问题和合并三部分的时间。根据算法得到递推式：

              T(1) = 1，T(n) = 2T(n/2) + n

用递推法容易解得T(n) = O(nlogn). 结合排序时间，所以算法总的时间为O(nlogn).

### 10. (论述题, 18分)**设M是一个n×n的整数矩阵，其中每一行（从左到右）和每一列（从上到下）的元素都按升序排列，确定一个给定的整数x是否在M中。设计分治算法求解该问题，并分析时间复杂度。**

**正确答案：**

算法思路：

设矩阵区间表示为四元组（left, right, top, bottom），即矩阵的最左列left，最右列right，最顶行top，最底行bottom。考虑矩阵的右上角元素M(top,right),根据其与x的大小关系缩小查找区间，即用分治法：

（1）若M(top,right)>x，则说明M的最右列元素都比x大，因此排除最右列,在子矩阵区间（left，top，right-1，bottom）内继续查找;

（2）若M(top,right)<x，则说明M的最上一行元素都比x小，因此排除第一行,在子矩阵区间（left，top+1，right，bottom）内继续查找;

（3）若M(top,right)=x，则找到，输出结果。

算法描述：

Bool  SearchM（top,right, M[], x）

输入：给定已排序的矩阵M[1...n，1...n] 及矩阵区间（left，right, top，bottom），初始值为(1,n,1,n). 要查找的数x

输出：若x在M中返回True，否则返回False.

1．若 top=bottom and right=left and M(top,right)!=x则返回False；

2．If M(top，right)=x 则返回True；

3．If M(top，right)>x,则返回SearchM(top, right-1, M, x);

4．Else 返回SearchM(top+1, right，M, x).

算法分析：

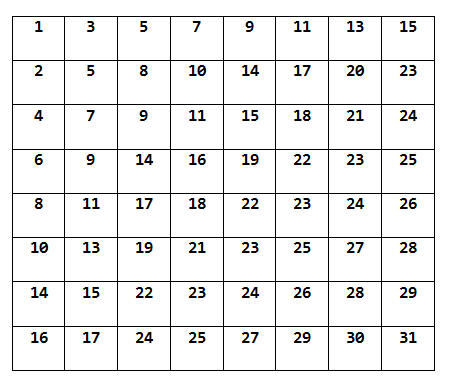
上述算法每次迭代都要减少一行或一列，因此该算法最多迭代2n次，设时间复杂度是T(2n),可得时间递推式：

T(1)=1，T(2n) = T(2n-1)+1 = … = 2n = O(n)

**答案解析：**

举例如下：

查找示例x＝12或x＝20



## 作业3：DFS与回溯法

### 11. (简答题, 10分)**写出深度优先遍历的算法框架。**

**正确答案：**

参考答案：

主程序框架：

•      Input: Graph g（邻接矩阵或邻接表）

•      Output: Return value depends on application.

•      int dfsSweep(graph g,int n, …)

•          int ans;

•          Color all node WHITE;

•          For each v in g.V

•             if  ( v.color == WHITE )

•                int vAns = dfs(g, v, …);

•                <Process vAns>

•          return ans;

DFS算法：

void DFS(Graph g,int v)  //邻接矩阵的DFS算法

{  v.color = GRAY;    //进入访问

   <Preorder processing of node v>;

   foreach w in g.neighbor(v)  //找顶点v的所有相邻点

       if (w.color == WHITE)

       <Exploratory processing of edge vw>;

           DFS(g,w); //找顶点v的未访问过的相邻点w

           <Backtrack processing of edge vw>;

       else

       <Checking edge vw>;

   <Postorder processing of node v >

   v.color = BLACK;

}

### 12. (简答题, 10分)**使用回溯法解0/1背包问题：n=3，C=9，V={6,10,3}，W={3,4,4},其解空间由长度为3的0-1向量组成，要求用一棵完全二叉树表示其解空间（从根出发，左1右0），并画出其解空间树，计算其最优值及最优解。**

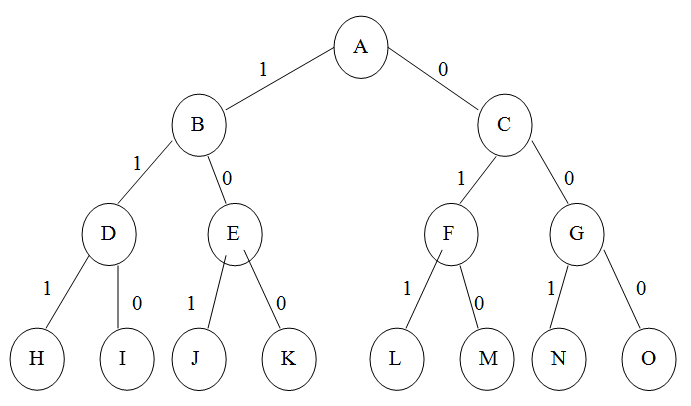
**正确答案：**

参考答案：

解空间为{(0,0,0),(0,1,0),(0,0,1),(1,0,0),(0,1,1),(1,0,1),

(1,1,0),(1,1,1)}。

解空间树为：



该问题的最优值为：16     最优解为：（1，1，0）

### 13. (简答题, 10分)

**n后问题回溯算法**

**(1)用二维数组A[N][N]存储皇后位置,若第i行第j列放有皇后,则A[i][j]为非0值,否则值为0。**

**(2)分别用一维数组M[N]、L[2\*N-1]、R[2\*N-1]表示竖列、左斜线、右斜线是否放有皇后，有则值为1,否则值为0。**

**try(i,M,L,R,A)**

**for(j=0;j<N;j++)**

**if(  1     )  /\*安全检查\*/**

**{ A[i][j]=i+1;    /\*放皇后\*/**

**2     ；**

**if(i==N-1)   输出结果；**

**else    3    ；;  /\*试探下一行\*/**

**4    ；      /\*去皇后\*/**

**5    ；;**

**}**

**正确答案：**

参考答案：

(1) !M[j]&&!L[i+j]&&!R[i-j+N]

(2) M[j]=L[i+j]=R[i-j+N]=1;

(3) try(i+1,M,L,R,A)

(4) A[i][j]=0

(5) M[j]=L[i+j]=R[i-j+N]=0

### 14. (论述题, 15分)

**用回溯法设计求解装载问题的算法，并分析时间复杂度。**

**装载问题：有一批共n个集装箱要装上2艘载重量分别为c1和c2的轮船，其中集装箱i的重量为wi，且w1+w2+…+wn<=c1+c2。装载问题要求确定是否有一个合理的装载方案可将这n个集装箱装上这2艘轮船。如果有，找出一种装载方案。**

**正确答案：**

参考答案（下面两种解法之一即可）：

void backtrack (int i，x, cw, r)   {// 搜索第i层结点

if (i > n)  // 到达叶结点，更新最优解bestx,bestw;return;

r -= w[i];

if (cw + w[i] <= c1) {// 搜索左子树

x[i] = 1;

cw += w[i];

backtrack(i+1,x,cw,r);

cw -= w[i];     }

if (cw + r > bestw)  {  // 搜索右子树

x[i] = 0;

backtrack(i+1,x,cw,r);      }

}

另解：

bool load(i,x,cw1,cw2)

{  if (i>n ) ｛输出解x；return true;｝

if (cw1+w[i]>C1 && cw2+w[i]>C2) return false;

if (cw1+w[i] <= c1) {// 集装箱i装到1号船

x[i] = 1;

cw1 += w[i];

load(i+1,x,cw1,cw2);

cw1 -= w[i];  }

       if (cw2 + w[i] <= c2) {//集装箱i装到2号船

x[i] = 2;

load(i+1,x,cw1,cw2);

}

}

本问题是子集树问题，时间复杂度为O(2^n)。

### 15. (论述题, 15分)

**用回溯法设计求解整数变换问题的算法，并分析时间复杂度。**

**整数变换问题：关于整数**i**的变换**f**和**g**定义如下：**f**(**i**)=3**i**,**g**(**i**)=int(**i**/2). 对于给定的两个整数**n**和**m**，要求用最少的变换**f**和**g**变换次数将**n**变为**m**.**

**正确答案：**

参考答案：

算法思路：

  设n和m不同，则需要至少一步将n变换成m，每一步可以选择变换f或g, 先试用f，再回溯试用变换g. 设定最大步数为maxdep，解向量s[maxdep], s[i]=0表示没有变换，1表示f，2表示g.

用全局变量mindep表示最少变换数，初值为maxdep或无穷大；minS表示最少次数对应的解向量。

算法描述：

Bool  search(int dep, int n, s[])

输入：当前深度dep（初值为0），输入n要变换成m，s是解向量。

输出：若变换成功返回True，且输出最少变换数及解向量，否则返回false.

1.  if  n = m then

2.     if dep < mindep then  mindep ＝ dep，minS ＝ s.copy ;

3.     Return True;

4.  dep ＝ dep + 1;

5.  If dep > {maxdep, mindep}  return  false;

//超过最大深度（不成功）或超过最小步数(剪枝)。

6.  s[dep] ＝ 1;  //采用变换f(i)=3i.

7.  flag1 ＝ search(dep,  3\*n,  s);

8.  s[dep] ＝ 2;  //回溯

9.  flag2 ＝ search(dep,  ën/2û, s);

10. return  flag1 or flag2.

算法分析：

最坏情况下，时间复杂度为O(2^maxdep).

### 16. (论述题, 15分)

**用回溯法设计求解马的周游问题的算法，并分析时间复杂度。**

**马的周游问题：给出一个**n**´**n**棋盘，已知一个中国象棋马在棋盘上的某个起点位置（**x**0,**y**0），求一条访问每个棋盘格点恰好一次，最后回到起点的周游路线。（设马走日字）**

**正确答案：**

参考答案：

算法思路：

马走日字，从棋盘上的某个格点出发有8走法，相对水平位移和垂直位移用两个数组表示：

dx[8]={2, 1, -1, -2, -2, -1, 1, 2}，

dy[8]={1, 2, 2, 1, -1, -2, -2, -1}.

用一个数组tag[n][n]标记被马访问过的格点，tag[x][y]=0表示格点（x, y）没有被访问，tag[x][y]=1表示格点（x, y）已被访问。马从起始点出发遍历棋盘每个格点一次，最后回到起点得到一条周游路线。

   解向量用数组path[n\*n][2], 每个元素是所能到达的格点坐标（x, y）；若问题无解，则返回false。

    用回溯法，马在每个位置都有8种走法，如果越界（合法范围：1<= x, y <= n.）或者目标点被访问，则剪枝，否则分支再回溯，即最多可回溯7次。

算法描述：

Bool  HorseTravel(dep, x, y, tag[][], path[][] )

输入：正整数dep(初值=1)表示步数，位置(x, y), 初始为起点(x0, y0).

输出：周游路线path；若问题无解，则返回false.

1.  if (x, y)=(x0,y0)  and  dep = n\*n + 1  then

2.    输出path;

3.    return True；

4.  If dep > n\*n or (x,y)越界or tag[x, y] then return False；

5.  tag[x][y] = 1;

6.  path[dep] = (x, y);

7.  flag = false；

8.  for k = 1 to 8

9.    flag = flag OR HorseTravel(dep+1, x+dx[k], y+dy[k], tag, path );

10. Return flag.

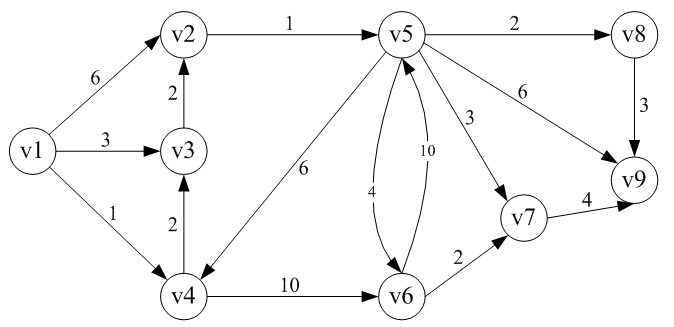
算法分析：解空间搜索树每步最多8个分支，完全遍历需要n\*n步，所以时间复杂度是O(8^(n\*n)).

## 作业4：BFS与分支限界法

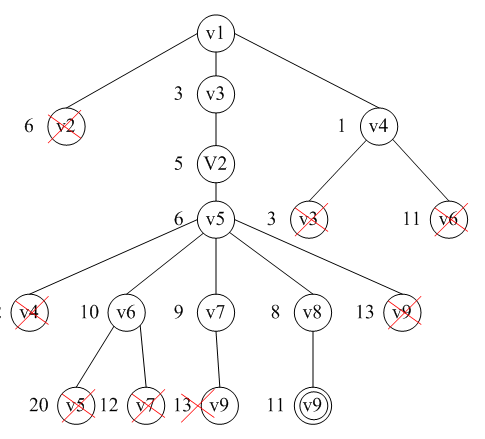
### 6. (简答题, 10分)**写出基于广度优先搜索的分枝限界法算法框架。**

### 7. (简答题, 10分)

**根据优先队列式分支限界法，求下图中从v1点到v9点的单源最短路径，请画出求得最优解的解空间树。要求中间被舍弃的结点用×标记，获得中间解的结点用单圆圈O框起，最优解用双圆圈◎框起。**



**正确答案：**



### 8. (简答题, 10分)**用分支限界法求解**n**皇后问题，请设计队列中的结点结构，并写出在处理当前结点时可以添加新结点的条件。**

**正确答案：**

结点结构：

struct NodeType｛

   int no；   //结点编号

   int row；  //结点行号

   vector<int> cols  //前row－1个皇后放置的列号。

}

设当前结点为e，k=0 .. e.row-1, col=0..n –1, 则在（e.row，col）格子中能够放置皇后的条件：

For all k, e.cols[k]!=col  && abs(e.cols[k]-col) != abs(k – e.row)

### 9. (论述题, 20分)

**请用分枝限界法求解下列问题，确定限界函数，规范描述算法，并分析算法时间复杂度。**

**装载问题：有一艘载重量为**C**的轮船，一批共**n**个集装箱，集装箱**i**的重量为**wi**，设计一个合理的装载方案使得尽可能多的集装箱装上这艘轮船。**

**正确答案：**

算法思路：

（1）问题转化为：先把C1尽可能装满，剩余的装到C2，不影响达到最优。输入集装箱重量：w[n]={ w1, w2, …, wn}, 总重量为W= w1+ w2+ …+ wn. 输出最优解向量x[n]和目标值（最大且不超过C1的重量）maxW<= C1.

（2）广度优先搜索（BFS），搜索结点队列为qu，队列结点结构包括：深度i，当前装入总重量cw，结点处的解向量x[n], 上界ub.

（3）分枝限界函数ub的计算：当前结点i之后的所有集装箱重量之和。

（4）处理结点i：

* 当且仅当i < n且cw< C1且cw＋ub >= maxW 时扩展新的搜索结点；否则剪枝。
* 检验结点重量是否达到最大重量，当maxW<=C1且W – maxW<=C2时，返回最优解向量和目标值，结束循环。

（5）队列为空时结束循环并返回失败标志。

算法描述：

Bool BFSLoad（w[n]，C1，C2）

输入：集装箱重量数组w[n]，载重量分别为C1和C2；

输出：最优解向量x[n]，最大重量maxW，若无解返回false

1.  maxW¬0; x[n] ¬0;

2.  pNode¬{0, 0, x, W }; //封装初始结点。

3.  qu ¬ pNode； //初始结点加入队列。

4.  while （队列qu不空）do

5.  {     p ¬ 取出队列qu的首结点；

6.        If (p.i==n && p.cw > maxW && p.cw <=C1) {

7.           maxW ¬ p.cw;

8.           x ¬ 复制p.x；}

9.        If  maxW <= C1 and W – maxW<=C2 then  retrun true；

10.       If (p.i<n && p.cw<C1) {

11.          p.x[p.i+1] ¬ 1;

12.          qu ¬ {p.i+1, p.cw+w[p.i+1],  p.x,  p.ub –w[p.i+1] }；

13.       If (p.i<n and p.cw＋p.ub >= maxW) {

14.          p.x[p.i+1] ¬0;

15.          qu ¬ {p.i+1, p.cw,  p.x, p.ub –w[p.i+1] }；

16. }

17. Return false.

算法分析：

   广度优先，最大搜索深度为n，最坏情况下时间复杂度为O(2n).

### 10. (论述题, 20分)

**用分枝限界法设计求解下列问题的算法，请用伪代码规范描述，并分析时间复杂度。**

**子集和问题：给定有**n**个不同正整数的集合**A**={**a**1,**a**2, …,**an**}和一个整数**W**，要求找出**A**的一个子集**S**，使得**S**中的所有元素之和为**W**.**

**正确答案：**

算法思路：

（1）输入集合记为一维数组A[n]，所有元素的和为T，整数W。如果有解输出解向量x[n]，x[i]＝1表示子集S中包含A[i]，x[i]＝0表示子集S中不包含A[i]. 如果无解返回false

（2）广度优先搜索采用队列qu，队列结点结构定义为：深度cdep，当前子集和cs，当前解向量x[n], 上界ub.

（3）限界函数ub定义为：当前剩余元素的和。

（4）处理当前结点p：

* 若当前子集和p.cs＝W，则返回解向量p.x；
* 添加新搜索结点的条件：p.cdep<n且p.cs < W且 p.cs+p.ub > W. 否则剪枝。

（5）若遍历所有结点（即队列为空）没有求得解则返回false。

算法描述：

bool BFSsubSetSum（A[n]，W ）

输入：整数集合一维数组A[n]，整数W；

输出：解向量x[n]或false.

1.  x[n] ¬0;

2.  pNode¬{0, 0, x, T }; //封装初始结点。

3.  qu ¬ pNode； //初始结点加入队列。

4.  while （队列qu不空）do

5.  ｛   p ¬ 取出队列qu的首结点；

6.       If  p.cs=W  then ｛ x ¬ p.x；retrun true；｝

7.       If  (p.cdep < n && p.cs<W && p.cs＋p.ub >= W)

8.       ｛   dep ¬ p.cdep +1;

9.            x ¬ p.x;

10.           x[dep] ¬ 1;

11.           qu ¬ {dep, p.cs+A[dep], x, p.ub –A[dep] ；

12.           x[dep] ¬0;

13.           qu ¬ {dep, p.cs, x, p.ub –A[dep] }；

14.      }

15. ｝

16. Return false.

算法分析：

   广度优先，最大搜索深度为n，最坏情况下时间复杂度为O(2n).

## 作业5：贪心法

### 7. (简答题, 14分)**已知有7种面值分别是1、2、5、10、20、50、100的纸币，现需要找退价值为Y的钱。设计一个贪心算法求解所需纸币张数最少的找零方案（只要求阐述算法思路），并证明该问题具有最优子结构性质。**

**正确答案：**

参考答案：

先用大面值的纸币找零，剩下的用次大的纸币找零，依次类推，直至余值为零。令Y的找零方案对应面值1、2、5、10、20、50、100的张数分别是c1, c2, c3, c4, c5, c6, c7。则c7 = Y/100; Y= Y%100; c6 = Y/50; Y= Y%50; …; c2 = Y/2; Y= Y%2; c1 = Y;

用a1, a2, a3, a4, a5, a6, a7分别表示1、2、5、10、20、50、100，假设c1, c2, c3, c4, c5, c6, c7为Y的找零方案的最优解，容易用反证法证明c1, c2, c3, c4, c5, c6, 0是子问题B=Y-c7\*100的最优解。如果c1, c2, c3, c4, c5, c6, 0是不是子问题B的最优解，则存在B的一个解b1, b2, b3, b4, b5, b6, 0，使得a1\*b1+ a2\*b2+a3\*b3+a4\*b4+a5\*b5+a6\*b6=B且a1\*b1 + a2\*b2 + a3\*b3 + a4\*b4 + a5\*b5 + a6\*b6 < a1\*c1 + a2\*c2 + a3\*c3 + a4\*c4 + a5\*c5 + a6\*c6   , 则c1, c2, c3, c4, c5, c6, c7不为Y的找零方案的最优解，因为b1, b2, b3, b4, b5, b6, c7是Y的一个找零方案，且b1+ b2+ b3+ b4+ b5+ b6+c7 < c1+ c2+ c3+ c4+ c5+ c6+c7，矛盾。

### 8. (简答题, 14分)**计划组织一个独木舟旅行。租用的独木舟都是一样的，最多乘坐2人，而且载重有个限度，为**W**。假设参加旅行的人数为**n**，其总量为{**w**1, …,**wn**}, 设计一个贪心算法求所需的最少独木舟数（只要求阐述算法思路）。**

**正确答案：**

参考答案：

Step 1. 对n个人的重量从大到小排序，假设排序结果是{w1, …, wn};

Step 2. 令i = 1, j = n, c = 0， 然后从wi开始，如果wi > W, 则算法终止，没有可行的乘坐独木舟过河方案;

Step 3. 如果i < j且wi+ wj <= W, 则输出“重量为wi和 wj的两人乘坐一个独木舟”，j 减一；否则输出“重量为wi 的一个人乘坐一个独木舟”;

Step 4. C加一，i加一，如果i<=j, 转Step 3 ;

Step 5. 输出“所需的独木舟的个数为”C.

### 9. (简答题, 14分)**一辆汽车加满油后从A地出发到达B地，中途有若干个加油站可以加油，要求加油次数最少。给出贪心选择策略，每次应该选择在哪个加油站加油？说明这种贪心选择策略具有最优子结构性质。**

**正确答案：**

参考答案：

贪心选择策略：每次选择能跑到的且是最远的一个加油站加油。

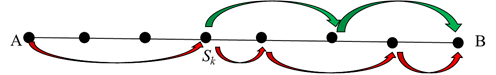
最优子结构：即问题的最优解包含其子问题的最优解。证明如下：

设从A地到B地有n个加油站S1，S2，…, Sn，汽车在任意Si加油站加油后必然可以顺利达到Si＋1，汽车出发时邮箱是满的，司机在任意加油站能判定是否必须加油才能跑到下一加油站。

首先第一次贪心选择是合理的，即第一次加油站的选取可以得到一个最优加油方案。不妨设第一次的加油站是Sk，若最优加油方案第一次不在Sk加油，则必然在Sk之前某个加油站Si（i < k）加油。设最优方案第二次加油在Sj（j ³ k），显然j > k，则在Sk加油后也能到达Sj（因为距离比从Si到达Sj更近），而且加油次数不变。因此第一次在Sk加油也是一个最优方案。



考虑从Sk加满油出发到达B地加油次数最少的方案，这是一个子问题（形式同原问题一致且规模更小）。该子问题的最优解与贪心选择Sk一起构成原问题的最优解。如若不然，如下图所示，红色箭头线表示贪心选择Sk和子问题的最优解。若它不是原问题的最优解，由于原问题一定有包含贪心选择Sk的最优解，不妨设为图中绿色箭头线，即子问题中还有次数更少的解，这与假设红色箭头线是子问题的最优解矛盾。



因此，本问题具有最优子结构性质。

### 10. (简答题, 14分)**有n个物品，已知n=7, 利润为P=(10,5,15,7,6,18,3)，重量W=(2,3,5,7,1,4,1)，背包容积M=15,物品只能选择全部装入背包或不装入背包，设计贪心算法，并讨论是否可获最优解。**

**正确答案：**

参考答案：

定义结构体数组G，将物品编号、利润、重量作为一个结构体：例如G[k]={1,10,2}

求最优解，按利润/重量的递减序，有

 {5,6,1,6} {1,10,2,5}{6,18,4,9/2} {3,15,5,3} {7,3,1,3}{2,5,3,5/3}  {4,7,7,1}

   算法

   procedure  KNAPSACK(P，W，M，X，n)

    //P(1：n)和W(1；n)分别含有按

      P(i)/W(i)≥P(i+1)/W(i+1)排序的n件物品的效益值

      和重量。M是背包的容量大小，而x(1：n)是解向量//

    real P(1：n)，W(1：n)，X(1：n)，M，cu；

    integer i，n；

    X←0  //将解向量初始化为零//

    cu←M  //cu是背包剩余容量//

    for i←1 to n do

      if  W(i)>cu  then  exit  endif

      X(i) ←1

       cu←cu-W(i)

    repeat

    end GREEDY-KNAPSACK

  根据算法得出的解：

  X=（1,1,1,1,1,0,0）获利润52， 而解

（1,1,1,1, 0, 1,0）可获利润54

因此贪心法不一定获得最优解。

11. (简答题, 14分)

通过键盘输入一个高精度的正整数n(n的有效位数≤240)，去掉其中任意s个数字后，剩下的数字按原左右次序将组成一个新的正整数。编程对给定的n 和s，寻找一种方案，使得剩下的数字组成的新数最小。

如输入n=178543,s=4,输出：13

**正确答案：**

参考答案：

为了尽可能地逼近目标，我们选取的贪心策略为：每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字，否则删除第一个递减区间的首字符。然后回到串首，按上述规则再删除下一个数字。重复以上过程s次，剩下的数字串便是问题的解了。

具体算法如下：

输入s, n;

while（ s > 0 ）

{  i=1;  //从串首开始找

while (i < length(n)) && (n[i]<n[i+1])

{i++;}

delete(n,i,1); //删除字符串n的第i个字符

s--;

}

while (length(n)>1)&& (n[1]=‘0’)

delete(n,1,1); //删去串首可能产生的无用零

输出n;

## 作业6：动态规划法

### 9. (简答题, 10分)**以换零钱为例说明动态规划法求解的基本步骤。**

**正确答案：**

参考答案：

1、划分子问题，分析最优子结构；

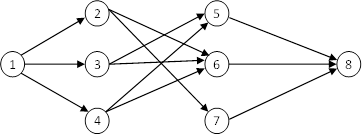
2、设置最优函数，建立递推关系；

3、计算最优值；

4、构造最优解。

### 10. (简答题, 10分)

**写出多段图最短路经动态规划算法求解下列实例的过程，并求出最优值。**



**各边的代价如下：**

**C(1,2)=3， C(1,3)=5 ，C(1,4)=2**

**C(2,6)=8 ，C(2,7)=4 ，C(3,5)=5 ，C(3,6)=4， C(4,5)=2，C(4,6)=1**

**C(5,8)=4， C(6,8)=5 ，C(7,8)=6**

**正确答案：**

参考答案：

设Cost(i)表示i号结点到达终点8的最小代价，D[i]记录i号结点最小代价路径上的下一结点。

Cost(8)=0

Cost(7)= C(7,8)+0=6, D[7]=8

Cost(6)= C(6,8)+0=5, D[6]=8

Cost(5)= C(5,8)+0=4, D[5]=8

Cost(4)= min{C(4,6)+Cost(6),C(4,5)+Cost(5)}= min{1+5,2+4}=6，

D[4]=6

Cost(3)= min{C(3,5)+Cost(5),C(3,6)+Cost(6)}= min{4+5,5+4}=9

D[3]=5

Cost(2)= min{C(2,6)+Cost(6),C(2,7)+Cost(7)}= min{8+5,4+6}=10

D[2]=7

Cost(1)=min{C(1,2)+Cost(2),C(1,3)+Cost(3),C(1,4)+Cost(4)}

         = min{3+10,5+9,2+6}= 8

D[1]=4

所以最优解为Cost(1)=8，路径：1→4→6→8。

### 11. (简答题, 10分)证明：任何可以用动态规划法求解的问题，一定可以用循环实现（不需要递归）。

**正确答案：**

答案要点：

1、子问题之间的依赖关系一定是有向无环图（DAG）。

2、有向无环图（DAG）一定存在拓扑排序。

3、按逆拓扑序求解一定能完成所有子问题的求解。

### 12. (论述题, 15分)

**用动态规划法设计求解子集和问题的算法，并分析算法时间和空间复杂度。**

**子集和问题：给定有n个不同正整数的集合A={a1, a2, …, an}和一个整数W，要求找出A的一个子集S，使得S中的所有元素之和为W.**

### 13. (论述题, 15分)

**用动态规划法求解完全背包问题，并分析时间和空间复杂度。**

**完全背包问题：有**n **种重量和价值分别为w[i]、v[i]（1≤i≤n）的物品，从这些物品中挑选总重量不超过W的物品，求出挑选物品价值总和最大的挑选方案，这里每种物品可以挑选任意多件。**

**正确答案：**

参考答案：

（1）设二维数组dp，dp[i][j] 表示从前i个物品中选出重量不超过j（或者剩余容量为j）的物品的最大总价值。则dp[n][W]为所求问题的最优价值。显然存在k，使得

      dp[i][j]=MAX{dp[i-1][j-k\*w[i]]+k\*v[i]：k\*w[i]≤j}

即具有最优子结构。设二维数组fk[i][j]存放dp[i][j]得到最大值时物品i挑选的件数。

（2）状态转移函数：

dp[i][0]=0（背包不能装入任何物品时，总价值为0），

dp[0][j]=0（没有任何物品可装入时，总价值为0），

dp[i][j]=MAX{dp[i][j]，dp[i-1][j-k\*w[i]]+k\*v[i]：k\*w[i]≤j}，

 i=1,2,..,n,  j=1,2,..., n

fk[i][j]=k;  物品i取k件。

算法描述：

int solve()  //求解完全（多重）背包问题

输入：n, W; w[], v[];

输出：dp[][], fk[][];

for (i=1;i<=n;i++)

     for (j=1;j<=W;j++)

        for (k=0; k\*w[i]<=j; k++)

          if (dp[i][j]<dp[i-1][j-k\*w[i]]+k\*v[i])

           {  dp[i][j]=dp[i-1][j-k\*w[i]]+k\*v[i];

              fk[i][j]=k;//物品i取k件

           }

根据fk[][]由此容易反求出原问题的解向量x[n]：

x[n] = fk[n][W];

r = W;

for i= n-1 to 1

     r= r - x[i+1]\*w[i];

    x[i] = fk[i][r];

算法分析：

solve算法有三重循环，k的循环最坏可能从0到W，所以算法的时间复杂度为O(nW^2)。

算法的空间复杂度是两个二维数组的存储量，即为O(nW)。