1. 给出渐进上界符号 O 的定义

解答:

2. 给出 8 种复杂性函数

解答:

3. 递归函数的两个要素?

解答: 边界条件、递归方程

4. 写出产生全排列的递归程序

解答:

5. 已知递归算法的复杂度递归方程，求解 T(n)

解答:

6. 分治法的适应条件

解答:

7. 最优子结构的定义

解答:

8. 矩阵连乘问题：

（1）最优值的定义

（2）动态规划求解程序

解答： (2)

```
#include<iostream>
using namespace std;
const int N = 100;
int A[N];//矩阵规模
int m[N][N];//最优解
int s[N][N];
void MatrixChain(int n)
{
    int r, i, j, k;
    for (i = 0; i <= n; i++)//初始化对角线
    {
        m[i][i] = 0;
    }
    for (r = 2; r <= n; r++)//r个矩阵连乘
    {
        for (i = 1; i <= n - r + 1; i++)//r个矩阵的r-1个空隙中依次测试最优点
        {
            j = i + r - 1;
            m[i][j] = m[i][i]+m[i + 1][j] + A[i - 1] * A[i] * A[j];
            s[i][j] = i;
            for (k = i + 1; k < j; k++)//变换分隔位置，逐一测试
            {
                int t = m[i][k] + m[k + 1][j] + A[i - 1] * A[k] * A[j];
```

```
                if (t < m[i][j])//如果变换后的位置更优，则替换
                    原来的分隔方法。
                {
                    m[i][j] = t;
                    s[i][j] = k;
                }
            }
        }
    }
}
void print(int i, int j)
{
    if (i == j)
    {
        cout << "A[" << i << "]";
        return;
    }
    cout << "(";
    print(i, s[i][j]);
    print(s[i][j] + 1, j);//递归1到s[1][j]
    cout << ")";
}
int main()
{
    int n;//n个矩阵
    cin >> n;
    int i, j;
    for (i = 0; i <= n; i++)
    {
        cin >> A[i];
    }
    MatrixChain(n);
    cout << "最佳添加括号的方式为: ";
```

```
    print(1, n);
    cout << "\n最小计算量的值为: " << m[1][n] << endl;
    return 0;
}
```

9. 求解活动安排问题的贪心算法程序，并证明其正确性

解答：

（1）

```
int greedySelector(int s[],int f[],int a[])
{
    int n = sizeof(s) / sizeof(s[0]);
    a[0] = 1; // The first activity is always selected
    int j=1;
    int count = 1;
    for (int i = 1; i < n; i++)
    {
        if (s[i] >= f[j])// If the start time of the current
           activity is greater than or equal to the finish
           time of the last selected activity
        {
            a[count] = i + 1; // Store the index of the
               selected activity
            j = i; // Update the last selected activity
            count++;
        }
        else
        {
            a[i] = 0; // If the activity is not selected,
               store 0
        }
    }
    return count;
}
```

（2）正确性证明：

10. 写出回溯法搜索子集树和排序树的一般算法代码

解答:

```
1  （1）子集树
2
3
4
5
6
7
8
9
10
11
12
13
14 （2）算法树
```

11. 写出 0-1 背包的（1）队列式（2）优先队列式的分支界限程序

解答:

```
（1）
//分支限界算法 - 01背包问题      FIFO队列
int w[] = { 16,15,15 };//物品的重量
int v[] = { 45, 25, 25 };//物品的价值
int c = 30;//背包的容量
const int n = sizeof(w) / sizeof(w[0]);//物品的个数
int cw = 0;//已选择物品的重量
int cv = 0;//已选择物品的价值
int bestv = 0;//装入背包的物品的最优价值

//描述节点类型
struct Node
{
    Node(int w, int v, int l, Node *p, bool left)
    {
        weight = w;
        value = v;
        level = l;
        parent = p;
        isleft = left;
    }
    int weight;//已选择物品的总重量
    int value;//已选择物品的总价值
    int level;//节点所在的层数
    Node *parent;//记录父节点
    bool isleft;//节点是否被选择
};
Node *bestnode = nullptr;//记录最优解的叶子节点
queue<Node*> que;//广度遍历需要的FIFO队列
```

```
30
31  void addLiveNode(int w, int v, int l, Node *parent, bool
        isleft)
32  {
33      Node *node = new Node(w, v, l, parent, isleft);
34      que.push(node);
35
36      if (l == n && v == bestv)
37      {
38          bestnode = node;
39      }
40  }
41  //求价值上界
42  int maxBound(int i)
43  {
44      int bound = 0;
45      for (int level = i + 1; level < n; ++level)
46      {
47          bound += v[level];
48      }
49      return bound;
50  }
51  int main()
52  {
53      int i = 0;//起始的层数
54      Node *node = nullptr;//记录父节点
55      while (i < n)
56      {
57          //选择物品i
58          int wt = cw + w[i];
59          if (wt <= c)
60          {
61              if (cv + v[i] > bestv)
```

```
62              {
63                  bestv = cv + v[i];
64              }
65
66              //把左孩子加入活结点队列当中
67              addLiveNode(cw+w[i], cv+v[i], i+1, node, true);
68          }
69
70          //不选择物品i
71          int upbound = maxBound(i);
72          if (cv + upbound >= bestv)
73          {
74              addLiveNode(cw, cv, i + 1, node, false);
75          }
76
77          node = que.front();
78          que.pop();
79          i = node->level;
80          cw = node->weight;
81          cv = node->value;
82      }
83
84      cout << bestv << endl;
85      int x[n] = { 0 };
86      for (int j = n - 1; j >= 0; --j)
87      {
88          x[j] = bestnode->isleft ? 1 : 0;
89          bestnode = bestnode->parent;
90      }
91
92      for (int v : x)
93      {
94          cout << v << " ";
```

```
    }
    cout << endl;
    return 0;
}
 (2)
#include <iostream>
#include <queue>
#include <functional>
#include <vector>
using namespace std;

//分支限界算法 - 01背包问题  优先级队列
int w[] = { 16,15,15 };//物品的重量
int v[] = { 45, 25, 25 };//物品的价值
int c = 31;//背包的容量
const int n = sizeof(w) / sizeof(w[0]);//物品的个数
int cw = 0;//已选择物品的重量
int cv = 0;//已选择物品的价值
int bestv = 0;//装入背包的物品的最优价值

//描述节点类型
struct Node
{
    Node(int w, int v, int l, int up, Node *p, bool left)
    {
        weight = w;
        value = v;
        level = l;
        parent = p;
        isleft = left;
        upbound = up;
    }
    int weight;//已选择物品的总重量
```

```
128     int value;//已选择物品的总价值
129     int level;//节点所在的层数
130     Node *parent;//记录父节点
131     bool isleft;//节点是否被选择
132     int upbound;//节点的价值上界，从这个节点往下，最多能选择的物品产生的总价值
133 };
134 
135 //queue<Node*> que; //广度遍历需要的FIFO队列
136 priority_queue<Node*, vector<Node*>, function<bool(Node*, Node*)>> que([](Node*n1, Node*n2)->bool //默认是大根堆
137 {
138     return n1->upbound < n2->upbound;
139 });
140 
141 void addLiveNode(int w, int v, int l, int up, Node *parent, bool isleft)
142 {
143     Node *node = new Node(w, v, l, up, parent, isleft);
144     que.push(node);
145 
146     //用优先级队列就不用标记产生最优解的叶子节点了，因为优先级队列到达某一个叶子节点时，最优值就产生了
147     /*if (l == n && v == bestv)
148     {
149         bestnode = node;
150     }*/
151 }
152 
153 //求价值上界
154 int maxBound(int i)
155 {
156     int bound = cv;
```

```
157     for (int level = i; level < n; ++level)
158     {
159         bound += v[level];
160     }
161     return bound;
162 }
163 int main()
164 {
165     int i = 0;//起始的层数
166     Node *node = nullptr;//记录父节点
167     int upbound = maxBound(0);
168     while (i < n)
169     {
170         //选择物品i
171         int wt = cw + w[i];
172         if (wt <= c) {
173             if (cv + v[i] > bestv)
174             {
175                 bestv = cv + v[i];
176             }
177 
178             //把左孩子加入活结点队列当中
179             addLiveNode(cw + w[i], cv + v[i], i + 1, upbound,
                    node, true);
180         }
181 
182         //不选择物品i
183         upbound = maxBound(i+1);//i+1表示第一个未被处理的物品
                的数组下标
184         if (upbound >= bestv)
185         {
186             addLiveNode(cw, cv, i + 1, upbound, node, false);
187         }
```

```
188
189         node = que.top();
190         que.pop();
191         i = node->level;
192         cw = node->weight;
193         cv = node->value;
194         upbound = node->upbound;
195     }
196
197     cout << bestv << endl;
198     int x[n] = { 0 };
199     for (int j = n - 1; j >= 0; --j)
200     {
201         x[j] = node->isleft ? 1 : 0;
202         node = node->parent;
203     }
204
205     for (int v : x)
206     {
207         cout << v << " ";
208     }
209     cout << endl;
210     return 0;
211 }
```

12. 写出旅行商问题的（1）队列式（2）优先队列式的分支界限程序

解答:

13. 给出计算时间下界及上界规约

解答:

14. 什么是（1）P 类问题（2）NP 类问题（3）NP——hard 问题（4）NP 完全问题

解答:

15. 已知团问题属于 NPC，证明定点覆盖问题也是 NPC

解答：

16. 设计定点覆盖问题的近似算法，并计算其性能

解答：