16.1-3 首先将原问题转化为区间图着色问题，将两两不相邻的顶点集合看成一个等价类，采用贪心策略如下：每次任意选取一个顶点，将其等价类纳入到一个集合 Ai 中，并对原图删除 Ai 中的所有顶点。如此重复以上操作，最后每一个集合中的顶点属于同一颜色。不同集合属于不同颜色，则得到最优解。

对于该算法操作，我们容易知道无论从哪一个顶点开始，所得到的集合个数均相同。（因为在一个集合的顶点都是一个等价类）

证明其最优性：

a. 对于只有原图关于 A1 的子图，那么明显最优是全部染成一种颜色。
b. 假设对 G 有 k+1 个等价类，首先去除图 G 满足算法的 A1 顶点集，剩下的 k 个集合 A2…Ak+1 组成的点集，所以可以得到最少的染色至多为 k+1 种。假设对于图 G，最优解法小于等于 k，不妨为 s<=k，将每一种颜色的点纳入一个集合，总共 B1…Bs，则会有 B1 中的顶点互不相邻，所以得到等价类的个数为 s，这与假设中等价类个数为 k+1 矛盾，所以最少的染色恰好为 k+1 种。

所以对等价类个数为 n 的图 G 最优染色一定是 n 种，即算法得到的染色最佳。

16.2-5 算法：将顶点进行从小到大的排序。每一次以点集最左边的那个为左端点，画出一个单位闭区间，将落入闭区间中的顶点全部删除，再次循环考虑即可。每一次得到的闭区间的总集合为最优解。

证明：考虑最优的集合，则考虑其最左的区间记为 a1，a1 覆盖最左的端点。即区间 a1 的左端点不大于算法中最左区间的左端点，算法给出的区间两两没有交集，故区间 a1 在算法除第一个区间的其他区间的左侧。去掉第一个区间中的点后再次进行上述分析可以得出每一次去掉的一个算法得出的区间，都会有 ak 区间相对应，所以最优的集合中元素个数一定不小于算法得到集合的个数。故只能相等，所以即为最优。

16.2-3 算法：从价值最大开始按照价值递减（也是重量递增）的物品一个一个放进入背包，直到背包不能再放入物品即可。

证明最优性：

首先价值最大的物品一定在内，否则，只需要将背包中任何一个物品替换成价值最大的物品后仍能够保持总价值增大，但不超过限制。设物品按价值递减排序后为 a1,a2…an，再证明最优解中背包中的物品按照排序后一定为 a1,a2…ak，否则不成立，则存在一个 am，使得背包中排序后 a1,a2…a(m-1),a(m+k)…,(k>0)，此时将 a(m+k)替换成 am 仍然背包限制满足且总价值不减。故综上最优解背包中存在的物品一定满足：a1,a2…ak，故算法成立。

16.3-3 对{an}为斐波那契数列，易知满足 a1+a2+…+an<=a(n+2)，所以每一次对于 a1+a2+…+ak 的结点而言只会和 a(k+1)互为兄弟结点，即生成了一个除根结点层外每一层都只有两个结点的整体成斜趋势的哈夫曼树，该树高为 n。哈夫曼编码表现为 0，10，110，…，111…10，111…11 (0，1 可以互换，共 n 个)。对于 n 对于 8，只需要代入 n 即可。

2-1

伪代码如下：

cost 是权重记录，work 是 n 维数组，assign 为当前分配记录，assign_end 为最终选取分配

```
void dfs(csum, idx) ：
    if csum >min ：
        return;
    if idx>=num :
        Min = csum;
        For   i=0 to i<num :
            assign_end[i] = assign[i];
        return 0;
    For   i = 0 to i<num :
        If work [i]:   continue;
        work [i] =1;
        assign[t] = i;
        csum +=cost[i][t];
        dfs(csum,t+1);
        csum -=cost[i][t];
        work [i] =0;
```

直接调取 dfs(0,0)即可。

2-2

伪代码如下：

max 初值为 0, assign 为当前分配记录，assign_end 为最终选取分配

```
void dfs(idx,t) ：
    if idx>=num ：
        Min = csum;
        For   i=0 to i<num :
            For   j = 0 to j<size(assign[i]) :
                assign_end[j] = assign[j];
        return ;
    For   i = 0 to i<k :
        if csum[i] + cost[idx]>Min : return;
        csum[i]+=cost[idx];
        temp = max;
        max = max(csum[i], max);
        max = temp;
        assign[i].push (idx);
        dfs(csum,t+1);
        assign[i].pop();
```

直接调用 dfs(0,0)即可。