hw8

> 16.1-3；16.2-3，16.2-5；16.3-3；上机题

16.1-3

16. 1-3 对于活动选择问题，并不是所有贪心方法都能得到最大兼容活动子集。请举例说明，在剩余兼容活动中选择持续时间最短者不能得到最大集。类似地，说明在剩余兼容活动中选择与其他剩余活动重叠最少者，以及选择最早开始者均不能得到最优解。

给定的算法只会愚蠢地计算O(n)个数中的最小值，或者根据Sij的大小返回零。由于我们选择i和j使得1≤i≤j≤n，所以可能的子问题数量是O(n2)。因此，运行时间是O(n3)。

16.2-3

16. 2-3 假定在 0-1 背包问题中，商品的重量递增序与价值递减序完全一样。设计一个高效算法求此背包问题的变形的最优解，证明你的算法是正确的。

Algorithm 1 0-1 Knapsack(n,W)

```
Initialize an n + 1 by W + 1 table K
for j = 1 to W do
    K[0, j] = 0
end for
for i = 1 to n do
    K[i, 0] = 0
end for
for i = 1 to n do
    for j = 1 to W do
        if j < i.weight then
            K[i, j] = K[i − 1, j]
        end if
        K[i, j] = max(K[i − 1, j], K[i − 1, j − i.weight] + i.value)
    end for
end for
```

每一步只选择最轻的（也是最有价值的）物品放入背包。要证明这种解决方法是最优的，假设我们选择了某个物品 j，但是没有选择另一个更轻更有价值的物品 i。那么，我们可以用物品 i 替换背包中的物品 j。这样做肯定能放得下，因为 i 更轻，而且也会增加总价值，因为 i 更有价值。

16.2-5

16. 2-5 设计一个高效算法，对实数线上给定的一个点集$\{x_1, x_2, \cdots, x_n\}$，求一个单位长度闭区间的集合，包含所有给定的点，并要求此集合最小。证明你的算法是正确的。

考虑最左边的区间。如果它向左延伸超过最左边的点，那么它没有任何好处，但是，我们知道它必须包含最左边的点。所以，我们知道它的左端点就是最左边的点。所以，我们就删除任何在最左边的点的单位距离内的点，因为它们都被这个单一的区间覆盖了。然后我们就重复这个过程，直到所有的点都被覆盖。由于每一步都有一个明显的最优选择，即把最左边的区间放在哪里，所以这个最终的解决方案是最优的。

16.3-3

16.3-3 如下所示，8 个字符对应的出现频率是斐波那契数列的前 8 个数，此频率集合的赫夫曼编码是怎样的？

a：1　b：1　c：2　d：3　e：5　f：8　g：13　h：21

你能否推广你的结论，求频率集为前 n 个斐波那契数的最优前缀码？

一个最优的霍夫曼编码是

$$0000000 \to a\ 0000001 \quad \to b\ 000001 \to c\ 00001 \quad \to d\ 0001 \to e\ 001 \quad \to f\ 01 \to g\ 1 \quad \to$$

这可以推广到将前 n 个斐波那契数作为频率，即第 $k < n$ 个最频繁的字母的编码为 $0^{k-1}1$，而第 n 个最频繁的字母的编码为 0^{n-1}。为了证明这一点，我们将证明递推关系

$$\sum_{i=0}^{n-1} F(i) = F(n+1) - 1$$

这将表明我们应该将频率为 $F(n)$ 的字母与将较小频率的字母合并在一起的结果合并在一起。我们将用归纳法证明它。对于 $n = 1$，这是显而易见的。现在，假设我们有 $n - 1 \geq 1$，那么，

$$F(n+1) - 1 = F(n) + F(n-1) - 1 = F(n-1) + \sum_{i=0}^{n-2} F(i) = \sum_{i=0}^{n-1} F(i)$$

为了利用这一事实，来证明所需的霍夫曼编码是最优的，我们声称当我们贪婪地合并节点时，每个阶段我们都可以维持一个包含所有最不频繁的字母的节点。最初，这只包含最不频繁的字母。在第 k 阶段，归纳地，我们假设它包含了 k 个最不频繁的字母。这意味着它的权重为 $\sum_{i=0}^{k-1} F(i) = F(k-1) - 1$。这个阶段的所有其他节点的权重为 $F(k), F(k+1), \ldots F(n-1)$。所以，显然，两个最低权重的节点是包含 k 个最不频繁的字母的节点和包含第 k 个最不频繁的字母的节点。因此，贪婪算法告诉我们，我们应该将这些节点合并在一起，留下一个包含 $k+1$ 个最不频繁的字母的节点，完成归纳。

Sch2-1：工作分配问题

设有n件工作要分配给n个人去完成，将工作i分配给第j个人所需费用为c_{ij}。试设计一个算法，为每个人分配1件不同的工作，并使总费用达到最小。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

const int MAXN = 100;
const int INF = 1e9;
```

```
int cost[MAXN][MAXN];
int lx[MAXN], ly[MAXN];
int match[MAXN];
bool vx[MAXN], vy[MAXN];
int slack[MAXN];
int n;

bool hungarian(int x) {
    vx[x] = true;
    for (int y = 0; y < n; y++) {
        if (vy[y]) continue;
        if (lx[x] + ly[y] == cost[x][y]) {
            vy[y] = true;
            if (match[y] == -1 || hungarian(match[y])) {
                match[y] = x;
                return true;
            }
        } else {
            slack[y] = min(slack[y], lx[x] + ly[y] - cost[x][y]);
        }
    }
    return false;
}

void update() {
    int a = INF;
    for (int y = 0; y < n; y++) {
        if (!vy[y]) a = min(a, slack[y]);
    }
    for (int x = 0; x < n; x++) {
        if (vx[x]) lx[x] -= a;
    }
    for (int y = 0; y < n; y++) {
        if (vy[y]) ly[y] += a;
        else slack[y] -= a;
    }
}

void KM() {
    for (int x = 0; x < n; x++) {
        lx[x] = ly[x] = 0;
        match[x] = -1;
        for (int y = 0; y < n; y++) {
            lx[x] = max(lx[x], cost[x][y]);
        }
    }
    for (int x = 0; x < n; x++) {
        for (int y = 0; y < n; y++) {
            slack[y] = INF;
        }
        while (true) {
            for (int i = 0; i < n; i++) {
                vx[i] = vy[i] = false;
            }
            if (hungarian(x)) break;
            else update();
        }
```

```
    }
}

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> cost[i][j];
        }
    }
    KM();
    int ans = 0;
    for (int y = 0; y < n; y++) {
        if (match[y] != -1) {
            ans += cost[match[y]][y];
        }
    }
    cout << ans << endl;
    return 0;
}
```

Sch2-2：最佳调度问题

设有n个任务由k个可并行工作的机器来完成，完成任务i需要时间为t_i。试设计一个算法找出完成这n个任务的最佳调度，使完成全部任务的时间最早。（要求给出调度方案）

```
#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

struct Machine {
    int id;
    int end_time;
    bool operator<(const Machine& rhs) const {
        return end_time > rhs.end_time || (end_time == rhs.end_time && id
> rhs.id);
    }
};

std::vector<int> schedule(const std::vector<int>& tasks, int k) {
    std::priority_queue<Machine> machines;
    for (int i = 0; i < k; ++i) {
        machines.push({i, 0});
    }
    std::vector<int> task_order(tasks.size());
    std::vector<int> sorted_tasks = tasks;
    std::sort(sorted_tasks.begin(), sorted_tasks.end(), std::greater<int>
());
    for (int i = 0; i < sorted_tasks.size(); ++i) {
        Machine m = machines.top();
        machines.pop();
        task_order[i] = m.id;
        m.end_time += sorted_tasks[i];
        machines.push(m);
```

```
    }
    return task_order;
}

int main() {
    std::vector<int> tasks = {10, 20, 30, 40, 50, 60, 70, 80, 90};
    int k = 3;
    std::vector<int> result = schedule(tasks, k);
    for (int i = 0; i < result.size(); ++i) {
        std::cout << "Task " << tasks[i] << " is assigned to machine " <<
result[i] << std::endl;
    }
    return 0;
}
```