Solution: Homework 3, 4

Minimum Spanning Tree

采用 Kruskal 或者堆优化的 Prim 或者 Borůvka 即可。

Prim 的堆优化实现和 Dijkstra 几乎一致。

Holiday scheduling: episode 1

本质是选择不相交的区间。

Holiday scheduling: episode 2

`f[i][j]` 表示第 i 天结束时，累计玩了 j 天，能积累下最多多少钱。一共三种转移：

- 发呆：`f[i][j] -> f[i + 1][j]`
- 玩：`f[i][j] >= x` 时，`f[i][j] - x -> f[i + 1][j]`
- 工作：存在以 $i+1$ 开始 k 结束的价值为 w 的任务时，`f[i][j] + w -> f[k][j]`。任务可以用存图的方式存下来。

初始条件为 `f[0][0] = 0`，其余为 $-\infty$ 。最后找到最大的 k 满足 `f[n][k] >= 0` 即可。

总复杂度为 $O(n^2+nm)$。

Knapsack 3 in 1

将每个物品拆成 $1, 2, 4, 8, \cdots$ 直到拆不动为止，然后做零一背包。

总复杂度 $O(nW \log C)$。

Colorful inversion

本题来源是 ICPC Kunming 2020 L。

为什么就是最长下降子序列长度：因为每个下降子序列是一个团，颜色必须互不相同，因此染为最长下降子序列长度是必要的。同时因为最长下降子序列长度满足条件，所以亦是充分的。

Bo

本题来源是 Codeforces 867E Buy Low Sell High，当然也是一个经典题。

一个直观的想法是，我能赚钱就卖。也就是说，从前到后考虑，如果当前这天的价格比起之前价格的最小值高，那么我就 "buy low" 并且在当天卖出。

这么做有时不是很优： $a=[1,2,3]$，如果买了 1 卖 2 只能赚一块钱，而买了 1 卖 3 能赚两块钱。

观察发现，买 1 卖 3 可以看做：先买 1 再卖 2，然后买 2 再卖 3。这里第二次的买 2 并不是买入了某只股票，而是可以看做：我撤销之前的某次失误操作，换成了一个正确的操作。由于买和卖是相反的操作，撤销卖出的本质就是买入。

因此，本题解决的核心思路就是利用可靠的撤销操作，利用贪心完成本题。这种套路有时被称为反悔贪心。

具体实现时，我们每当完成一次交易时，将交易卖出的价格作为一个可买入的价格放入集合里，而以后购买这个价格代表的是：撤销这一次交易，并将卖出价格更新为当前的。

例如，错误的贪心可能是这样的：

```
for i = 1 to n:
    if prices.min() < a[i]:
        ans += a[i] - prices.min()
        prices.erase_min()
    else:
        prices.add(a[i])
```

而带反悔的贪心是这样的：

```
for i = 1 to n:
    if prices.min() < a[i]:
        ans += a[i] - prices.min()
        prices.erase_min()
        prices.add(a[i])
    prices.add(a[i])
```

可以注意到的是，如果某次卖出成功了，会有两个 `a[i]` 放进集合里，含义其实分别是撤销一次交易和买入撤销后没有操作的 `a[i]`。更抽象地来说，把本题写成整数规划的形式，相当于把 $x_i = 1$ 变为了 0 和 -1，从堆里能够取出一次则值减一。

Fair division

把拆金块的过程看成合并金块的过程，就是哈夫曼树了。因此每次选两个最小的合并就好。

Bonus: 如果每次分割支付的是金块的一部分（而不是另一种货币），也即：每次切金块就会有 p 的损耗，剩余任意分割。现在给定一个 w （不必再满足 $\sum_i a_i = w$ 的限制），判断是否存在一种可行的切割方案。

如果直接套用哈夫曼树的算法，那么每次合并完后相当于金块“膨胀”了 $p/(1-p)$，这样的情况下哈夫曼树算法还对吗？为什么？