HW5+HW8+样卷简答附加

作业详细答案已经发群里了，所以这里只是稍微补充一些值得注意的地方

HW5 Q1 删数字

- 贪心往往要选择“当前最……的”，因此贪心的实现往往与一些能够维护序关系的数据结构或算法相结合，例如排序、优先队列、红黑树等，本题就可以用优先队列维护
- 想要实现数组元素快速增删可以用链表，如果你已经持有了元素的指针，则可以在常数时间内完成
- 答案中给出的用数组模拟链表的方式值得注意，典型的能够体现类似思想的数据结构是二叉堆（将“树”建在一个数组上）
- 附加小题中使用了一种常用的对于“决策安全性”的证明方法，即：对于任意一个决策，都存在一个至少不更劣的贪心决策与之对应

HW5 Q2 收电脑

- 对于贪心正确性的另一种常见证明方法：先给出一个答案的上/下界，然后指出贪心算法恰好能够达到这个界
- 题目用到的滑动窗口法值得注意（尤其是面试写力扣算法题）

HW5 Q3 排列优化

- 一类典型的使用贪心能够解决的问题
- 第一问类似于数学中的“排序不等式”，将较大数与较大数相乘，这样更能发挥较大数的作用
- 第二问类似于“田忌赛马”：反正下等马肯定输，不如用来消耗掉对手的上等马

HW5 Q4 括号匹配

- 主要在于：怎么维护“最近未匹配的括号”，联想到栈的LIFO特性，恰好可以实现最近未匹配括号的快速查找。
- 某种程度上栈可以看成一个特殊的优先队列（堆顶是最新加入的元素）

HW8 Q1 变种最短路

- 第一问中提到的SPFA算法可以看成Bellman-Ford的高替，用于处理带负权图的单源最短路，又快又好写，但正权图还是Dijkstra更好
- 第二问的变种Dijkstra非常危险，最坏情况下可能具有指数级运行时间，切记不要使用

HW8 Q2 塔扬算法

- 图论中各种名词概念要记清
- Low值的计算本质上类似于树形dp

HW8 Q3 最小环

- 运行一次Dijkstra就可以把从源点到其他所有顶点的最短路算出来，而不是只能算一个点对之间的最短路

HW8 Q4 SAT

- 有趣的事实： 2SAT只是个P问题，但3SAT的难度却能达到NPH。

1. 分治策略（Divide-and-Conquer）是我们在算法设计中经常用到的方法。同时，递归式与分治方法紧密相关，它可以用来刻画分治算法的运行时间。请说明何为分治策略以及你所知道的求解递归式的方法。

2.3.1 分治法

许多有用的算法在结构上是**递归的**：为了解决一个给定的问题，算法一次或多次递归地调用其自身以解决紧密相关的若干子问题。这些算法典型地遵循**分治法**的思想：将原问题分解为几个规模较小但类似于原问题的子问题，递归地求解这些子问题，然后再合并这些子问题的解来建立原问题的解。

- **代入法** 我们猜测一个界，然后用数学归纳法证明这个界是正确的。
- **递归树法** 将递归式转换为一棵树，其结点表示不同层次的递归调用产生的代价。然后采用边界和技术来求解递归式。
- **主方法** 可求解形如下面公式的递归式的界：

$$T(n) = aT(n/b) + f(n) \tag{4.2}$$

其中 $a \geqslant 1$，$b > 1$，$f(n)$是一个给定的函数。这种形式的递归式很常见，它刻画了这样一个分治算法：生成 a 个子问题，每个子问题的规模是原问题规模的 $1/b$，分解和合并步骤总共花费时间为 $f(n)$。

- 递归树法一般不能用于证明

2. 数据库中存储了大小为 n, 取值范围在 0 到 750 区间的整数数组，要求数据库对该数组做某种线性时间的预处理，使得对于任意的统计某个区间 $[a,b], a,b \in [0,k]$ 元素个数的查询需求，该数据库可以在 $O(1)$ 时间内返回结果。

- 先用类似计数排序的方法得到0-750之间每个值的元素个数，然后O(n)时间预处理出前缀和，之后每次区间求和就只需在前缀和上做差即可

数据组织形式	单点查询	单点修改	区间求和	区间加减
原数组	O(1)	O(1)	O(n)	O(n)
前缀和数组	O(1)	O(n)	O(1)	O(n)
差分数组	O(n)	O(1)	O(n)	O(1)

- 原数组、前缀和数组、差分数组之间的转换需要O(n)时间
- 有没有折衷的方法：线段树（不要求）

3. 对于 n 件物品，背包容量为 W 的 0/1 背包问题，其中第 i 件物品的价值为 v_i，重量为 w_i。请写出用动态规划求解该问题的时间复杂度，并解释为什么该算法被称为伪多项式时间算法。

Pseudo-polynomial time:

a numeric algorithm runs in pseudo-polynomial time if its running time is a polynomial in **the numeric value of the input** but not necessarily in **the length of the input** (the number of bits required to represent it)

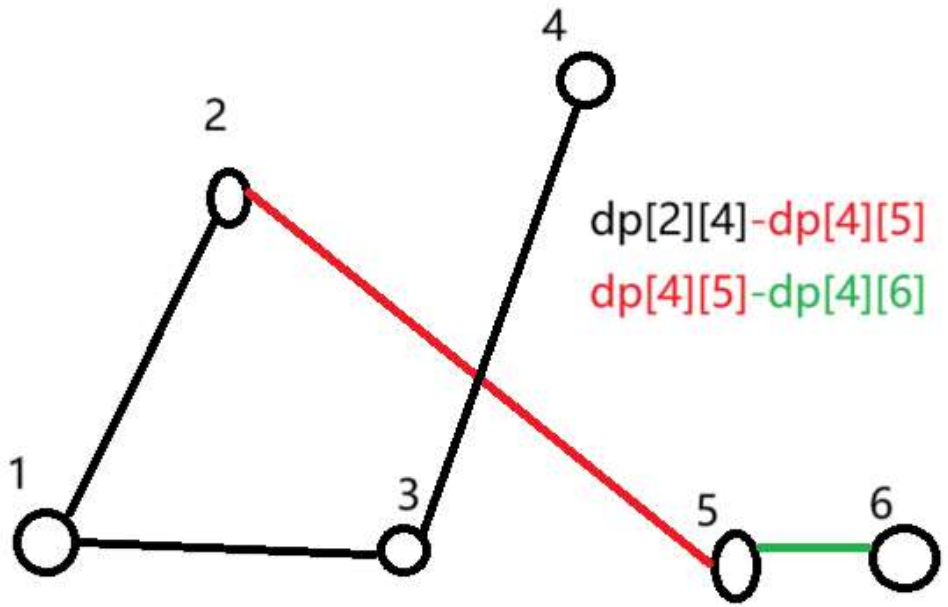
- The Running time of dynamic programming algorithm on 0-1 Knapsack problem is $O(W * n)$, the number W needs $\log W$ bits to describe, so it is **pseudo-polynomial**.
- Other pseudo-polynomial algorithm: Primality testing

- 思考1：为什么朴素Ford-Fulkerson算法是伪多项式时间的。
- 思考2：为什么用二分搜索找完全平方数的平方根是多项式时间的。

4. 计算 KMP 算法中对应于模式 $P = ababbabbabbababbabb$ 的前缀函数 π.

- 根据前缀函数的定义肉眼一个位置一个位置找最长公共真前后缀就行了
- 利用性质$\pi[i+1] \leq \pi[i]+1$在这题中会稍微好找一点
- 答案是0,0,1,2,0,1,2,0,1,2,0,1,2,3,4,5,6,7,8

- 将所有点从左到右排序，记为$v_1, v_2, \ldots, v_n$，定义$dp[i][j](i < j)$为：从v_1出发沿两条不同的路向右走，一条走到v_i结束，一条走到v_j结束，两条路径中间不能有相同的点，每个点必须属于两条路径之一
- 对于$dp[i][j]$，如果$j \neq i+1$，v_j的前驱一定是v_{j-1}（否则另一条路径的终点就不是i而是$j-1$了），因此$dp[i][j] = dp[i][j-1] + d[j-1][j]$
- 如果$j = i+1$，那么v_j可以选择任意$v_1 \sim v_{i-1}$作为前驱，因此$dp[i][j] = \min_{1 \leq k < i}(dp[k][i] + d[k][j])$



- 算出dp数组后，对$dp[i][n] + d[i][n]$取最小值即为结果。