12.1. Монетки.

Алгоритм простой: берем наибольшее возможное число монет б**о**льшего достоинства, прежде чем взять монеты меньшего достоинства.

Понять почему он работате тоже просто: предположим, что мы нашли другой, оптимальный, набор из меньшего числа монет. Может быть два случая:

- в двух наборах одинаковое число монет старшего достоинства c^k

Этот случай не очень интересен, ибо он сводится (вычитанием) к следующему:

- в двух наборах разное число монет старшего достоинства c^k.

В этом случае новый набор содержит (как минимум) на с монет достоинства c^{k-1} больше, чем первый за каждую «недостающую» монету достоинства c^k. Тогда можно эти монеты заменить на одну c^k и получим более оптимальный набор. Противоречие.

Также, на задачу можно смотреть как на разложение числа А в с-ичной системе счисления. Причем «цифры» в этом числе будут отвечать количеству соответствующих монеток. Причем рассуждения будут аналогичны (не может быть больше с-1 монетки какого-либо достоинства (кроме самого большого), ибо в ином случае их можно заменить на одну постарше и т.д.)

12.2. Хаффман.

Действительно, оптимальным деревом Хаффмана будет полное сбалансированное бинарное дерево с 256 листами. Каждому листу будет соответствовать кодовое слово длиной 8. А значит текст, закодированный Хаффманом по длине не будет выигрывать у текста, закодированного с использованием обычного восьмибитного кода.

Осталось понять, почему дерево Хаффмана будет именно таким. А предположим, это не так, то есть оптимальное дерево иное, то есть имеется лист на высоте семь (нашлось кодовое слово длиной семь), значит существует 2 листа на высоте 9. Ясно, что лист на высоте семь есть лист, построенный по самой большой частоте f_{max}, а два листа на высоте 9 – листы, построенные по минимальным частотам.

Длина закодированного текста полным деревом была

$$L = \sum_{i=1}^{256} 8 \cdot f_i$$

а длина текста, закодированного «новым, оптимальным» деревом стала:

$$L' = 9 \cdot f_{min} + 9 \cdot f_{next\_min} + \sum(8 * f_i) + 7 \cdot f_{max}$$

Разница: $f_1 + f_2 - f_{max} > \frac{f_{max}}{2} + \frac{f_{max}}{2} - f_{max} = 0$, т.к. $f_i \le f_{max} < 2 \cdot f_i$

Получили противоречие с тем, что дерево оптимальное.

12.3. Солдатики.

Посмотрим на самого сильного солдатика Васи. Он может победить любого из солдатиков Пети (тут необходимо заметить, что можно просто не рассматривать солдатиков Пети, которые сильнее любого из солдатиков Васи: их нельзя победить). Значти этого солдатика надо ставить напротив самого сильного из солдатиков Пети. Ибо если его поставить не на том месте, то его можно всегда будет передвинуть на указанное место и выигрыш не уменьшится. Рассмотрим второго солдатика: его надо ставить напротив самого сильного из солдатиков противника, из числа тех, кого он может победить. Объяснение аналогичное: если он стоит не там, то его можно переставить.

Алгоритм вырисовывается простой, действуем жадно: делаем сортировку солдатиков обоих мальчиков и одним проходом ставим солдатов на первое место, на котором противник стал слабее. Сложность: сортировка + один проход.

12.4. Игрушки.

Очень долго решал эту задачу, скорее всего я не смогу объяснить корректность жадного подхода, но алгоритм нарисовался такой.

Исходя из условия, время заказа составляет как минимум время покраски всех изделий плюс время на ожидание пока первый мастер сделает первую форму. Хочется, чтобы это время было минимальным, то есть чтобы второй ждал как можно меньше.

Дальше я пытался просто расположить все времена по убыванию/возрастанию одного из времен: либо время на формирование, либо покраску; однако каждый раз находил контрпример.

Последний алгоритм вроде работает: будем действовать жадно ☺.

Введем такое понятие как «опережение». Представим, что первый мастер начал делать какую-то игрушку. Как только он закончит, второй мастер может начать ее раскрашивать. Как только второй закончит, хорошо бы, если б к этому моменту первый сделал ему что-нибудь. Назовем время первого мастера, когда он готовит «следующие» игрушки для второго – опережением.

Ясно, что время первого опрежения есть b1.

Все игрушки можно разделить на 2 класса: те, которые увеличивают (неуменьшают) время опережения (это те, где b-a >= 0, то есть их красить дольше, чем лепить) и те которые уменьшают это время (где a-b < 0).

Жадность здесь будет такой: будем сначала делать игрушки, которые увеличивают время «опережения» (причем в порядке возрастания времени формовки a), а затем игрушки, которые это бонусное время уменьшают (теперь уже в порядке умеьншения времени формовки a): это все нужно, что избавить второго мастера от тормозов в работе.

Это продолжаться будет до тех пор, пока второму все-таки придется затормозить: тогад алгоритм начинается сначала (ведь в самом начале второй ждет первого): выбираем минимальное время ожидания второго мастера и начинаем сначала.

Сложность: асимптотически не больше пары сортировок (по временам формовки и покраски).

Строгого доказательства почему это работает (или когда это сломается) я привести не могу =(

Нет ли этой задачи в каком-нибудь сборнике с автоматическими тестами? Хочется ее написать…

12.5. «Плоский мир» Терри Пратчетта.

Сначала поймем, что от нас требуется.

Нам нужно максимизирвоать матожидание. Для этого хочется посчитать матожидание такой прибыли: что если в гильдию i придет ровно j человек? (j от 0 до 10). Это на самом деле легко посчитать: известна вероятность этого события, известна прибыль от такого события (формула приведена в задаче).

Просчет такой таблицы занимает: каждая гильдия обсчитывается за O(11*11) = O(1). Всего гильдий t, значит такая таблица счиается за O(t).

Итак, задача свелась к такой: дана таблица размером t строк на 11 столбцов, клетка p[i, j] означает матожидание прибыли, если в гильдию i придет ровно j волонтеров.

p0	p1	p2	p3	p4	p5	p6	p7	p8	p9	p10
pr[1,0]	pr[1,1]	…								

Задача: выбрать такие ячейки, чтобы сумма была максимальной, а сумма индексов столбцов в точности равнялась n. Причем в каждой строке была выбрана ровно одна ячейка.

Решается это жадно: выбираем максимальную ячейку, сводим задачу к предыдущей…

Однако, начинаются проблемы: значения в строке в таблице не монотонны. Удалось доказать, что они даже не унимодальны. (бинарный поиск не применить).

Еще пробелма в том, что мы можем распределить не всех волонтеров… Тогда придется часть ячеек сдвигать на менее выгодные позиции, увеличивая индексы столбцов.

Это хотя бы верное направление решения?

Единственное что приходит на ум:

Отсортируем вообще все значения в таблице (их O(t)), будем выбирать жадно самые максимальные, пока не выберем по одному числу в каждой строке. Затем, если еще недостаточно, будем выбирать «менее» максимальные числа, пока не распределим всех n волонтеров.

*Но кажется, это не сработает, а уж тем более за n*log(t) и вообще попахивает динамикой…*