Вологодский государственный университет

РАЗБОР ЗАДАЧ XVII МЕЖРЕГИОНАЛЬНОЙ ОЛИМПИАДЫ ПО ПРОГРАММИРОВАНИЮ (Вологда, 12 апреля 2025)

Задача A. Космические рисунки

Идея решения – **формула включений-исключений**.

Если рисунок сделан на каждой k-й доске, то всего таких досок n // k (где // - деление нацело). Просуммируем число досок с каждым видом рисунка:

$$S = N // 4 + N // 5 + N // 6$$

Но у нас посчитались и доски с двумя рисунками, причём дважды. Вычтем их:

$$S = N // 4 + N // 5 + N // 6 – N // 20*2 – N // 12*2 – N // 30*2$$

Пояснение: например, идя с шагом 4 и с шагом 6, мы будем встречаться в позициях 12, 24 и так далее, то есть с шагом НОК(4, 6)=12. Таких досок будет N // 12. Для других пар шагов – аналогично.

Ещё есть доски с тремя рисунками. Каждая вначале 3 раза добавлялась, потом 6 раз вычиталась. Значит, надо ещё 3 раза добавить. Окончательно получаем:

$$\mathbf{S = N // 4 + N // 5 + N // 6 – N // 20 * 2 – N // 12 * 2 – N // 30 * 2 + N // 60 * 3}$$

Задача B. Расход топлива

Идея решения – **система линейных уравнений**.

Пусть x – расход на 100 км по городу, y - по трассе. Имеем систему уравнений:

$$\begin{cases} x * A + y * B = 100*P \\ x * C + y * D = 100*Q \end{cases}$$

Решим её методом Крамера. Сосчитаем определители:

$$d = A*D - C*B$$
$$dx = 100*P*D - 100*Q*B,$$
$$dy = A*100*Q - C*100*P$$

- Если d = 0, dx = 0 и dy = 0, то решений бесконечно много – ответ «**Ambiguity**»
- Иначе если d = 0, dx ≠ 0 или dy ≠ 0, то решений нет – ответ «**Contradiction**»
- Иначе x = dx / d, y=dy / d
- Если x < 0 или y < 0, то ответ «**Not positive**»
- Иначе выводим «**Success**», затем корни x и y

Задача C. Путешествие на машине

Идея решения – **нестандартный граф, алгоритм Дейкстры.**

Изменим формулировку задачи. Пусть, приехав на заправку i с ценой бензина c[i], водитель получает сертификат, дающий право везде заправляться по цене не выше c[i]. Также пусть водитель каждый раз заправляет бензина минимальное количество, которого хватит до следующей заправки.

Такая формулировка эквивалентна исходной (заправиться по сертификату – это то же самое, что заправиться впрок на заправке, которая выдала этот сертификат).

Заметим, что есть не более 100 вариантов стоимости бензина. По входному графу построим новый граф следующим образом. Каждую вершину u заменим на 100 новых вершин: (u, 1), (u, 2), … (u, 100), где второй элемент пары – цена в сертификате.

Проведём рёбра. Рассмотрим вершину (u, x). Если в исходном графе было ребро u-v, то в новом графе проведём направленное ребро (u, x) → (v, min(x, c[v])). То есть, если на заправку u мы приехали с сертификатом с ценой x, и поехали на заправку v, то там у нас окажется сертификат с ценой min(x, c[v]).

Задача C. Путешествие на машине (продолжение)

В новом графе мы можем считать, что покупаем в каждой вершине на пути следования ровно столько бензина, чтобы хватило доехать до следующей вершины. Причём, выезжая из вершины (u, x), мы покупаем бензин по цене x (помним, что на самом деле мы его купили раньше – когда проезжали через вершину с ценой x).

Тогда веса рёбер будут такие: для ребра (u,x)→(v,y) с расходом f_{uv} вес равен $x \cdot f_{uv}$.

В итоге задача свелась к поиску путей минимального суммарного веса от вершины (1,c[1]) до всех вершин вида (n,?). Пути можно найти алгоритмом Дейкстры. Среди них выбираем минимальный.

Двигаясь по найденному пути в обратную сторону, восстанавливаем места заправок.

Для этого храним в переменной L, сколько литров надо докупить. Вначале L = 0.

Для очередного ребра (u,x)→(v,y) вначале добавляем к L расход по этому ребру.

Затем смотрим: если c[u] = x, то покупаем L литров в вершине u, а иначе откладываем покупку (так как ближе к началу пути встретится цена x, которая меньше c[u]).

Задача D. Мины

Идея решения – **два указателя.**

Пусть указатель i указывает на текущую мину слева, и он будет идти влево.
Указатель j указывает на текущую мину справа, и он будет идти вправо.
Также поддерживаем границы зоны взрывов: левая граница l и правая граница r.

Пока $x[i]$ или $x[j]$ лежит в интервале от l до r :

Если $x[i]$ лежит в интервале от l до r, то:

$l = \min(l, x[i] - d[i])$

$r = \max(r, x[i] + d[i])$

$i = i - 1$

Если $x[j]$ лежит в интервале от l до r, то:

$l = \min(l, x[j] - d[j])$

$r = \max(r, x[j] + d[j])$

$j = j + 1$

Задача E. Мессенджер

Первый способ решения – **модифицированный способ кодирования Хэмминга.**

Заменим символы от A до Z на числа от 0 до 25 соответственно. В ответе поставим их в позиции, номера которых – не степени числа 2. Пример для строки “ABAC”:

```
   1 2 3 4 5 6 7
s: ? ? 0 ? 1 0 2
```

В позициях ‘?’ будут контрольные коды. Контрольный код в позиции 2^i вычисляется как сумма по модулю 26 тех элементов, у которых номера позиций в двоичном представлении содержат единицу в бите i. Для нашего примера:

Код в позиции 2^0 равен (s[3]+s[5]+s[7]) % 26 = (0+1+2) % 26 = 3.

Код в позиции 2^1 равен (s[3]+s[6] +s[7]) % 26 = (0 + 0 + 2) % 26 = 2.

Код в позиции 2^2 равен (s[5]+s[6]+s[7]) % 26 = (1 + 0 + 2) % 26 = 3.

Результат кодирования: `3 2 0 3 1 0 2,` то есть “DCADBAC”

Задача E. Мессенджер (продолжение)

Декодирование. Вычислим контрольные коды и сравним с кодами в сообщении.

Если все они совпали, то ошибок нет.

Если испортился элемент номер i, то испортятся все контрольные коды, соответствующие единичным битам в двоичной записи i. То есть, если в числе i бит j равен 1, то испортится контрольный код 2^j.

Следствие: сумма номеров не совпавших контрольных кодов даёт позицию испорченного символа. Например, если не совпали контрольные коды 1 и 4, то испорчен элемент в позиции 5.

Осталось понять, какая буква была в этой позиции. Возьмём любой не совпавший контрольный код. Он вычисляется как сумма элементов, включая испорченный. Тогда разность между принятым и вычисленным контрольным кодом (по модулю 26) равна расстоянию между заменённой и оригинальной буквой (по модулю 26).

Задача E. Мессенджер (продолжение)

Второй способ решения – **хеширование строк.**

Кодирование. Сосчитаем хеш-функцию над исходной строкой, закодируем её девятью буквами от A до Z и допишем к исходной строке. Также ещё вычислим контрольную сумму для хеша (можно просто сумму его символов по модулю 26), чтобы знать, не испортился ли сам хеш.

Декодирование. Проверяем контрольную сумму хеша. Если не совпало, то хеш испорчен, значит, основная строка цела.

Иначе пробуем заменять каждую букву на все возможные и пересчитывать хеш заново (пересчёт можно делать за O(1)). В какой-то момент хеши совпадут – это значит, что ответ найден.

Задача F. Сортировка обменом

Идея решения – алгоритм на основе несложных соображений:

1. Найдём первый отрезок слева из последовательных чисел, стоящих не на своём месте. Например, для массива «1 2 7 8 6 3 4 5 9» это отрезок «7 8». Если не нашли, решения нет.

2. Найдём первый отрезок справа из последовательных чисел, стоящих не на своём месте. Например, для массива «1 2 7 8 6 3 4 5 9» это отрезок «3 4 5». Если не нашли или нашли отрезок из пункта 1, то решения нет.

3. Меняем отрезки местами и проверяем, получилось ли отсортировать. Если не получилось, то решения нет.

Задача G. Удаление скобок

Идея решения – **динамическое программирование на отрезках.**

Как частный случай, попробуем удалить вообще все скобки. Затем переберём позицию первой не удалённой открывающей скобки (открывающие и закрывающие перед ней удаляем).

Зафиксировав её, переберём позицию парной к ней закрывающей:

..... ±(…….)……

Сосчитаем выражение слева от ‘(‘. В нём все скобки удалены, поэтому оно вычисляется однозначно.

И у нас получились две подзадачи:
1). Найти минимальное либо максимальное значение выражения в скобках (в зависимости от знака перед ним).
2). Найти максимальное значение выражения после скобок.

Возможный подвох: удаляются не обязательно парные скобки, например:

1-(2-3)+(4-5) ⇨ 1-(2-3+4-5)

Задача H. Пара соседей

Идея решения – **псевдодвухмерное динамическое программирование**.

Пусть f[i] – количество подмножеств из первых i элементов, где нет двух соседних элементов, g[i] – где есть ровно одна пара соседей.

Начальные значения: f[0] = 1, f[1] = 2, f[2] = 3, g[0] = 0, g[1] = 0, g[2] = 1.

Элемент i мы можем брать либо не брать. Если не берём, то у нас остаются все множества из i-1 элементов. Если берём, то можем добавить его в каждое подмножество из i-2 элементов (так как элемент i-1 не берём). Отсюда:

$$f[i] = f[i - 1] + f[i - 2]$$

Аналогично для g. Мы можем не брать i-й элемент, тогда остаются все подмножества из i-1 элементов с одной парой соседей. Также мы можем взять i-й элемент, тогда у нас два случая. Во-первых, пара соседей может образоваться раньше, тогда мы не берём элемент i-1. Во-вторых, пара соседей может образоваться прямо сейчас, тогда не берём элемент i-2. Отсюда:

$$g[i] = g[i - 1] + g[i - 2] + f[i - 3]$$

Задача H. Пара соседей (продолжение)

Другой способ решения – **комбинаторика + ДП**.

Переберём позицию, где встретилась пара соседей. Пусть слева от неё находятся L элементов, справа – R.

Пусть f(X) – количество подмножеств из X, в которых нет соседних элементов. Тогда к ответу добавляем f(L-1) × f(R-1). Единицы вычитаются потому что соседние к паре элементы брать нельзя.

Вычисление функции f мы видели на предыдущем слайде, напомним:

$$f[0] = 1, f[1] = 2, f[i] = f[i - 1] + f[i - 2]$$

Задача I. Казино

Идея решения – **конструктивный алгоритм**. Например, вот такой:

- Поставим 8 фунтов. Если выиграли, то повторим. Если всё время будем выигрывать, то за 100 раундов заработаем 800 фунтов, и будет искомая тысяча.
- Если в первый раз проиграли, то в следующий раз поставим 24 фунта. Тогда, если выиграли, то наша сумма за два раунда увеличится на 16 фунтов. Если такое будет происходить постоянно, то за 50 раз по два раунда мы получим 800 фунтов.
- Но мы можем и второй раз проиграть. Тогда в третий раз поставим 56 фунтов. Если выиграли, то наша сумма за три раунда увеличится на 56-32=24, и за 99 раундов мы получим 792 фунта. Итоговая сумма 210+792=1002.
- Если и в третий раз мы проиграли, то в следующий раз обязательно выиграем. При этом у нас останется на ставку не менее 210-8-24-56=122 фунта. Итого мы заработаем 122-(8+24+56)=34 фунта за 4 раунда, или 850 фунтов за 100 раундов. Итоговая сумма 210+850=1060.

После каждого выигрыша повторяем описанный процесс с начала.

Задача J. Сумма квадратов

Идея решения – **неочевидным образом применить знания школьной математики**.

- Возьмём числа 2, 4, 6, …, 2·n-4 – всего n-2 числа.
- Добавим к ним число 1 – итого n-1 число.
- Вычислим сумму их квадратов. Она будет нечётной, то есть число вида 2p+1.
- Вычислим значение p и добавим в наш список – итого стало n чисел.

Утверждение. Сумма квадратов этих n чисел равна $(p+1)^2$.

Действительно, $(p+1)^2 = p^2+2p+1$, где 2p+1 – это сумма квадратов первых n-1 чисел выше.

То есть, для n > 2 ответ равен: 2, 4, 6, …, 2n-4, 1, p, p+1.

Для n=1 ответом будет 1 1, для n=2 ответ дан в примере из условия.

Задача K. Секретный уровень

Идея решения – **перебор и битовый параллелизм**.

Эта задача – усложненная версия задачи № 820 на acmp.ru.

Заметим, что цвет каждой клетки определяется количеством нажатий на неё и соседние клетки. Поэтому порядок нажатий неважен. Также нет смысла ни на какую клетку нажимать дважды, поскольку два нажатия – то же самое, что ноль.

Будем отдельно искать способ покраски поля в белый цвет и в чёрный. Для удобства во втором случае можно инвертировать исходные цвета, тогда второй раз тоже будем искать способ покраски в белый (можно дважды вызвать одну и ту же функцию).

Алгоритм:

- Перебираем, какие будут нажатия в первой строке.
- Чтобы при этом первая строка стала белой, однозначно определятся нажатия во второй строке.
- Чтобы вторая строка стала белой, однозначно определятся нажатия в третьей, и т.д.
- Если последняя строка стала белой, то мы получили кандидата в ответ.

Задача K. Секретный уровень (продолжение)

Простая реализация алгоритма работает за O(n·n·2^n)).

Используя битовый параллелизм, уменьшим сложность до O(n·2^n).

Представим поле как массив целых чисел field, где биты в field[i] – это цвета в строке i (бит 1 – черный, бит 0 – белый).

Пусть int p – закодированные нажатия в строке i-2 (1 – нажато, 0 – не нажато).

Пусть int p2 – аналогично нажатия в строке i-1, при которых вся строка i-2 – белая.

Сосчитаем для каждой клетки в строке i-1 сумму по модулю 2 нажатий её и соседних клеток. Для этого достаточно сделать xor с предыдущей строкой, а также с этой строкой, сдвинутой на 1 влево и на 1 вправо (в конце ещё отрежем лишний бит из-за сдвига вправо):

```
int presses = p2 ^ p ^ (p2 >> 1) ^ (p2 << 1);
presses &= (1 << n) - 1;
```

Теперь в presses бит с номером j равен сумме (по модулю два) нажатий на клетку (i-1, j), а также на клетки сверху, слева и справа.

Задача K. Секретный уровень (продолжение)

Итак, в переменной presses бит с номером j равен сумме (по модулю два) нажатий на клетку (i-1, j), а также на клетки сверху, слева и справа.

Заметим, что если эта сумма четна, то цвет клетки не изменится, иначе – инвертируется. То есть, цвета в строке i-1 можно получить ещё одной операцией xor с начальными цветами строки i-1:

```
int p3 = field[i-1] ^ presses;
```

Осталось заметить, что если какой-то бит j в p3 равен 1 (черный цвет), то в строке i в позиции j надо сделать нажатие, чтобы заменить цвет на белый. То есть, p3 – *это и есть битовое представление нужных нажатий в строке i, чтобы строка i-1 стала белой.*

Далее меняем пару (p, p2) на пару (p2, p3), и продолжаем аналогично.

Ещё нам здесь нужно уметь быстро считать количество единичных битов в числе. В C++ для этого есть готовая функция __builtin_popcount.

Сложность такого решения - $O(n \cdot 2^n)$.

СПАСИБО
ЗА ВНИМАНИЕ!

Авторы задач:

Андрианов Игорь Александрович,
Меньшиков Фёдор Владимирович,
Метляхин Александр Игоревич,
Метляхина Валентина Станиславовна