Региональная студенческая (межвузовская) олимпиада по программированию
21 апреля 2018 г.

Разбор задач

Андрианов И. А.

Вологодский государственный университет

Вологда
2018

Задача 1 - «Степень»

При $n \geq 1$ ответ равен 5 ($5^1 = 5$, $5^2 = 25$…)

При $n = 0$ ответ равен 1 ($5^0 = 1$)

При $n < 0$ ответы чередуются с периодом 4:

$5^{-1} = 0.\underline{2}$

$5^{-2} = 0.0\underline{4}$

$5^{-3} = 0.00\underline{8}$

$5^{-4} = 0.001\underline{6}$

$5^{-5} = 0.0003\underline{2}$

$5^{-6} = 0.00006\underline{4}$

…

Задача 2 - «Массивы»

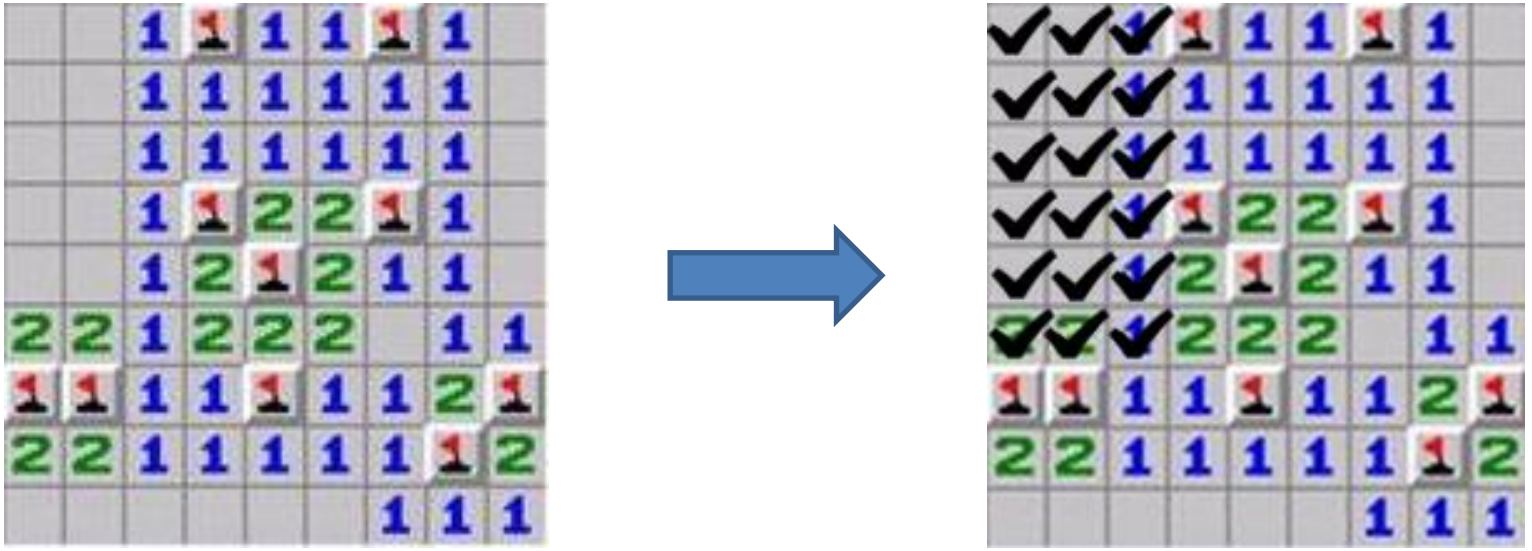
Решение «в лоб» не проходит по времени.

Можно понять, что не обязательно перемещать числа в памяти. Достаточно хранить массивы в виде односвязных списков и их объединение производить за O(1). В C++ это делается методом **splice** класса **std::list**.

Вариант решения с явным перемещением элементов: на каждом шаге меньший массив прикреплять к большему. При этом придётся иногда присоединять к началу, поэтому для хранения элементов удобно использовать деки (в C++ – класс **std::deque**). Также пригодится метод **swap** – поменять содержимое двух деков за O(1).

Задача 3 - «Miner»

Сосчитаем и запишем в каждую клетку, сколько мин находится рядом с ней. Пока есть клетки с нулём, берём очередную такую клетку и рекурсивно обходим область нулевых клеток, включая границу этой области из ненулевых клеток



Каждая такая область даёт +1 к ответу. В конце добавим к ответу количество непосещённых клеток.

Задача 4 - «Криптография»

Основные идеи: длинная арифметика, бинарный поиск

Делаем бинарный поиск по ответу. Пусть [**left**, **right**] – диапазон, в котором лежит ответ, **mid = (left + right) div 2**
Если $\mathbf{mid}^{\mathbf{n}} = \mathbf{p}$, то ответ найден
Если $\mathbf{mid}^{\mathbf{n}} < \mathbf{p}$, то корень лежит в интервале **(mid+1, right)**
Если $\mathbf{mid}^{\mathbf{n}} > \mathbf{p}$, то корень лежит в интервале **(left, mid-1)**

Важный момент: нужно правильно подобрать правую границу. Если взять **right** = $\mathbf{10^9}$, то это даст предел времени для тестов наподобие $\mathbf{2^{32766}}$

Можно подобрать правую границу отдельным циклом (перед бинарным поиском): пробовать **right** = 1, 2, 4, 8 …, пока $\mathbf{right}^{\mathbf{n}}$ не превысит **p**.

Задача 5 - «Обработка видео»

Для удобства будем нумеровать карты и кадры с нуля. Можно считать, что видеокарты обрабатывают кадры в порядке циклической очереди (round robin):

0, 1, …, N-1, 0, 1, … N-1, 0, 1…

Тогда кадр ***i*** будет обработан картой ***(i mod n)***.

Создадим массив **cam**, где ***cam[j]*** – время, когда освобождается ***j***-я карта. Прочитав ***i***-й кадр со временем ***x***, легко определить, когда он обработается картой:

cam[i % n] = (cam[i % n] > x) ? cam[i % n] + t : x + t;

Пусть **res** – время освобождения очереди к клиенту. Тогда ответ для ***i***-го кадра (и новое значение **res**) найдётся так:

res = (res > cam[i % n]) ? res + v : cam[i % n] + v;

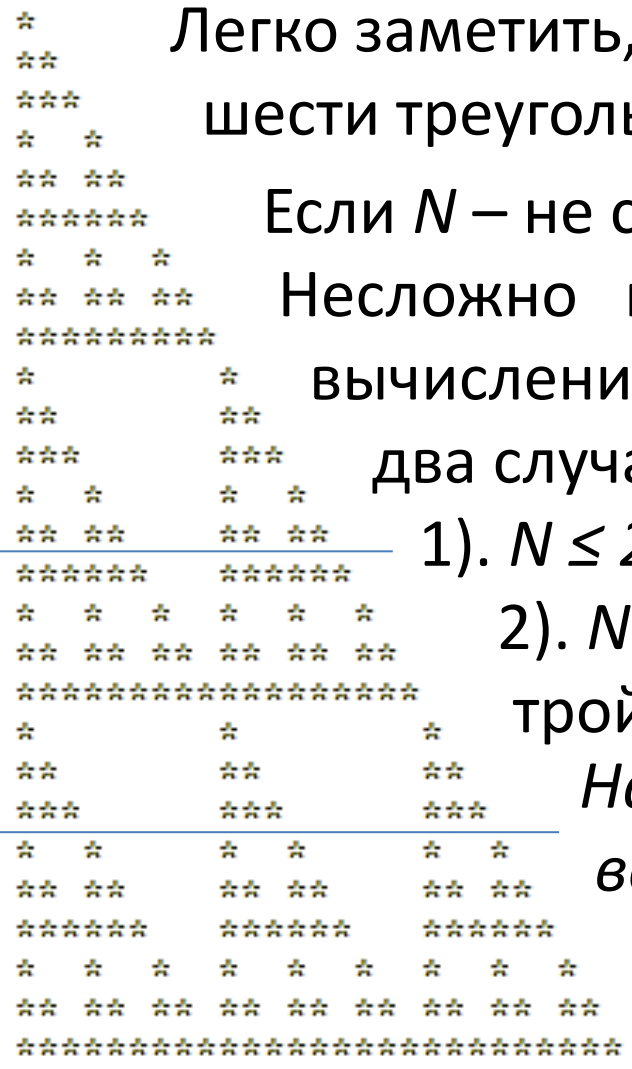
Пример решения задачи 5 на языке C++:

```cpp
#include <stdio.h>
#include <bits/stdc++.h>

int main() {
  int f, n, t, v;
  scanf("%d %d %d %d", &f, &n, &t, &v);
  std::vector<int> cam(n);
  int res = 0;
  for (int i = 0; i < f; i++) {
    int x;
    scanf("%d", &x);
    int c = i % n;
    cam[c] = (cam[c] > x) ? cam[c] + t : x + t;
    res = (res > cam[c]) ? res + v : cam[c] + v;
    printf("%d\n", res);
    fflush(stdout);
  }
}
```

Задача 6 - «Треугольник Паскаля»

Метод решения – исследование: построить треугольники для небольших *N* и поискать закономерности.



Легко заметить, что для $N=3^x$ треугольник составлен из шести треугольников со стороной 3^{x-1} (т.е. это фрактал)

Если *N* – не степень 3, то низ обрезан.

Несложно построить рекурсивную функцию для вычисления ответа. Можно отдельно рассмотреть два случая:

1). $N \le 2/3p$

2). $N > 2/3p$ (где *p* – ближайшая степень тройки, большая *N*).

На рисунке показаны два вида отреза: верхняя линия отсекает один целый треугольник и два нецелых, нижняя – три целых, три нецелых

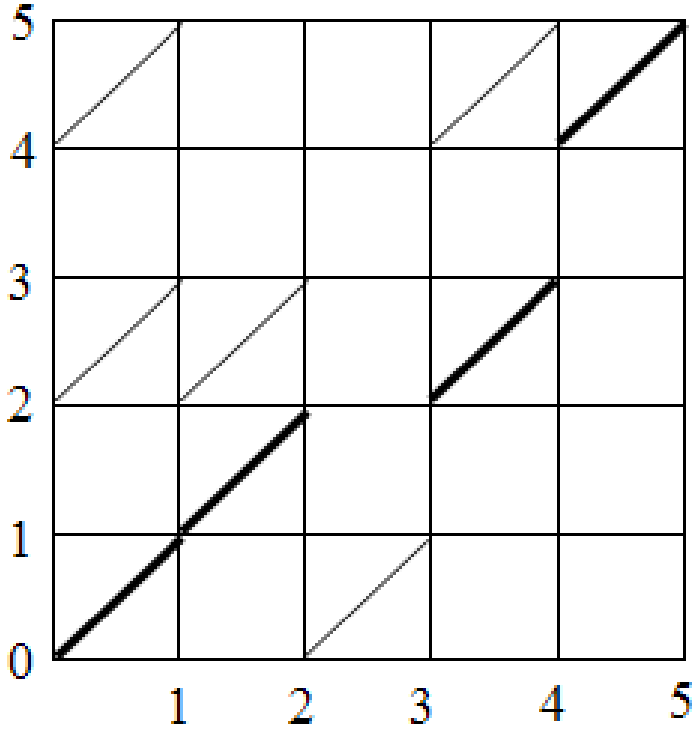
Задача 7 - «Кратчайший путь»

Усложнённая версия задачи с Тимуса:
http://acm.timus.ru/problem.aspx?space=1&num=1119
Оригинальная задача: N, M ≤ 1000, K ≤ 100
Новая задача: N, M ≤ 10^9, K ≤ 10^5.

Утверждение 1: есть смысл ходить только на север, восток и северо-восток.
Утверждение 2: чем больше диагональных кварталов на пути, тем лучше.

Отсортируем все диагональные кварталы по возрастанию ***x***, а для одинаковых ***x*** – по убыванию **y**. Выпишем последовательность ***y***-координат кварталов.
Задача сводится к следующей: найти в ней наибольшую возрастающую подпоследовательность. Это можно сделать за $O(k \cdot \log(k))$

Задача 7 – пример, поясняющий решение



Последовательность ***y***-координат (проходим диагональные кварталы слева направо сверху вниз):

5 3 1 3 2 1 5 3 5

Наибольшая возрастающая подпоследовательность:

1 2 3 5, то есть можно пройти по диагонали 4 квартала

Альтернативное решение:

Пусть *f(i)* – максимум, который можно достигнуть, дойдя до *i*-го квартала и пройдя его наискосок.

$$f(i) = 1 + \max(f(j)),$$

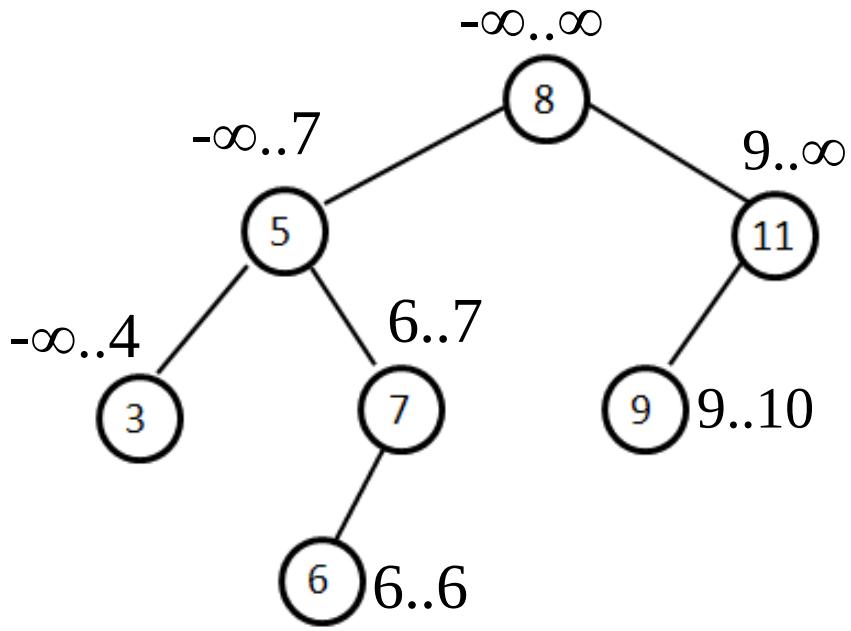
где *j* пробегает по всем диагональным кварталам ниже и левее i.

Для эффективного поиска максимума используем дерево отрезков.

Сложность – также $O(k \cdot \log(k))$

Задача 8 - «Двоичное дерево»

Каждый узел дерева задаёт интервал элементов, которые будут проходить через этот узел при последующих вставках. Интервалы для дочерних узлов являются подынтервалами родительских.



Куда будет подвешен новый лист с ключом ***x***?

К созданному как можно позднее узлу, интервал которого включает ***x***. Например, при вставке числа 10 лист будет подвешен к узлу с интервалом 9..10.

Для эффективного поиска применим ***дерево отрезков***.

Суть решения: поскольку новый узел задаёт новый отрезок, то и будем присваивать номер узла этому отрезку. *Подробнее*:

1. Выполним сжатие координат

2. Создадим дерево отрезков с операцией присвоения на отрезке. Идём по входным числам, берём очередное число *x*.

- Чтобы быстро найти узел в двоичном дереве, к которому будет подвешен новый лист с ключом *x*, достаточно запросить в дереве отрезков значение в позиции *x*

- Подвесив к найденному узлу новый лист с номером *i*, соответствующий интервалу [*left*, *right*], нужно также в дереве отрезков всему отрезку [*left*, *right*] присвоить значение *i*.

Максимальную высоту можно поддерживать в ходе работы либо найти её в конце обходом полученного дерева.

Задача 9 - «Гири»

Методы решения – ДП, битовый параллелизм.

Пусть $w[0], \ldots, w[n-1]$ – веса гирь, $M = w[0] + \ldots + w[n-1]$.
Создадим массивы $can[0..M]$ и $dup[0..M]$ из 64-битных целых чисел, где:
$can[i][бит\_j] = 1$, если можно набрать вес i, используя ровно j гирь
$dup[i][бит\_j] = 1$, если вес i, используя j гирь, можно набрать несколькими способами.

Внешним циклом по i будем перебирать гири от 0 до $n-1$.
Внутренним циклом по j будем перебирать по убыванию возможные веса (от максимального набранного на предыдущих шагах до 0).
Значение can[j] – это набор битов, означающий набор разных количеств гирь, которыми можно набрать вес j.
Тогда (can[j] << 1) – это набор количеств гирь, которыми можно набрать вес j+w[i] (добавив ещё гирю номер i).

Если какие-то количества гирь давали дубликаты для *j*, то на единицу большие количества дадут дубликаты и для j + w[i]:

$$dup[j + w[i]] \mid= dup[j] << 1;$$

Если какое-то количество гирь давало ранее вес j+w[i], и мы снова получили этот вес с таким же количеством, то это дубликат:

$$dup[j + w[i]] \mid= can[j + w[i]] \,\&\, (can[j] << 1);$$

Отметим, какими ещё количествами можно набрать вес j+w[i]:

$$can[j + w[i]] \mid= can[j] << 1;$$

Количество операций = $O(n \cdot M)$. При максимальных входных данных оценка сверху составляет $50 \cdot 50 \cdot 10^5 = 250$ млн. действий. Более точный анализ даёт в 2 раза меньше – около 125 млн. действий.

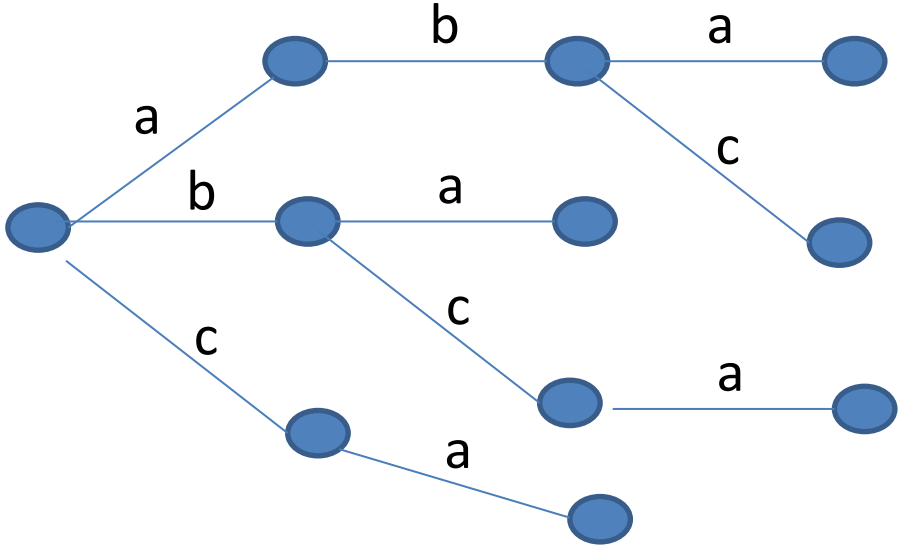
Задача 10 - «Поисковый индекс»

По мотивам статьи http://wonconsulting.com/~cho/papers/cho-regex.pdf

Идём по документам и с каждой позиции добавляем до ***L*** символов в бор, увеличивая счётчик документов в каждом узле бора на пути.

Чтобы не сосчитать один документ дважды, будем хранить также в каждом узле номер последнего документа. Счётчик увеличиваем, только если номер текущего документа не совпадает с номером в узле.

Пример бора для строк ‘aba’, ‘abca’ при L=3:



Обходим бор поиском в глубину, перебирая переходы по возрастанию символов. Тогда строки будут получаться в лексикографическом порядке.

Если счётчик в текущем узле <= ***T***, то в потомки смысла идти нет. Однако, текущая строка может не быть ответом, так как ответом может быть её суффикс (вместо *aXXX* ответом может быть *XXX*).

Чтобы это проверить, поищем текущую строку без первого символа в боре. Если при этом мы дошли до узла со счётчиком <= T, то исходная строка ответом не является, а иначе – её выводим.

Оценка сложности. Построение бора выполняется за $O(L \cdot S)$, где S – сумма длин всех строк.

Обход бора с проверкой суффиксов выполняется за $O(L^2 \cdot S)$.

Макс. размер бора – порядка $28*4*10*10^6 \approx 1$ гигабайт.

Задача 10 – альтернативное решение

Вначале построим *префиксно-свободное множество* полезных подстрок ***P***, в котором ни одна подстрока не является префиксом другой.

Возьмём все подстроки из одного символа (‘a’, ’b’…), для каждой из них построим ***список вхождений*** – то есть в каких документах и в каких позициях она встречается. Это делается за один проход по документам.

Подстроки, которые встречаются менее чем в T документах, сохраним в ***P*** *(в качестве* ***P*** *используем вектор)*.

Остальные подстроки необходимо ***расширить*** – приписать к ним справа один символ. Поскольку нам известно, где встречается каждая подстрока, то это делается за один проход по спискам вхождений подстрок.

Так повторяем до достижения длины ***L***.

Утверждение: в префиксно-свободном множестве ***P*** не может быть двух строк вида ***q*** и ***pqr***.

Доказательство: поскольку строка ***q*** содержится в ***P***, то он встречается не боле чем в ***T*** документах. Тогда строка ***pq*** – и тем более. Отсюда следует, что в ходе построения строка ***pq*** не могла расшириться до **pqr,** так как сразу попала в **P**.

Следствие. “Лишние” элементы в ***P*** – только такие строки, суффиксы которых тоже лежат в ***P***. Отбросить их можно так:
-перевернём все строки из ***P***, отсортируем их
-добавим первую в ответ
-пройдёмся по всем остальным. Если последняя добавленная в ответ строка не является префиксом текущей строки, добавляем её к ответу. В конце ещё раз перевернём строки и отсортируем.

Оценка сложности: сумма длина списков вхождений не превышает *S*, выполняется максимум *L* проходов – итого O(*S·L)* действий. Затем сортировка вектора *P* даёт O(*S·L·log(S·L)).*