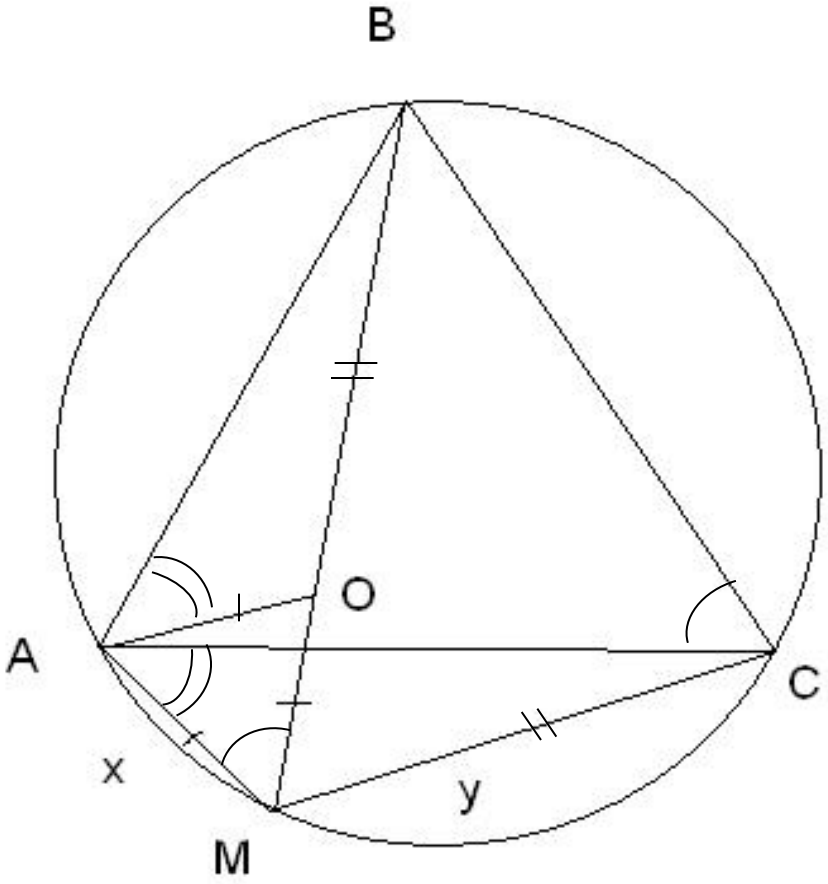
Задача X - «Длина хорды» (с пробного тура)

Решение



На отрезке *MB* от точки *M* отложим отрезок, равный *x*. Поскольку углы *ACB* и *AMB* опираются на общую хорду *AB*, они равны.

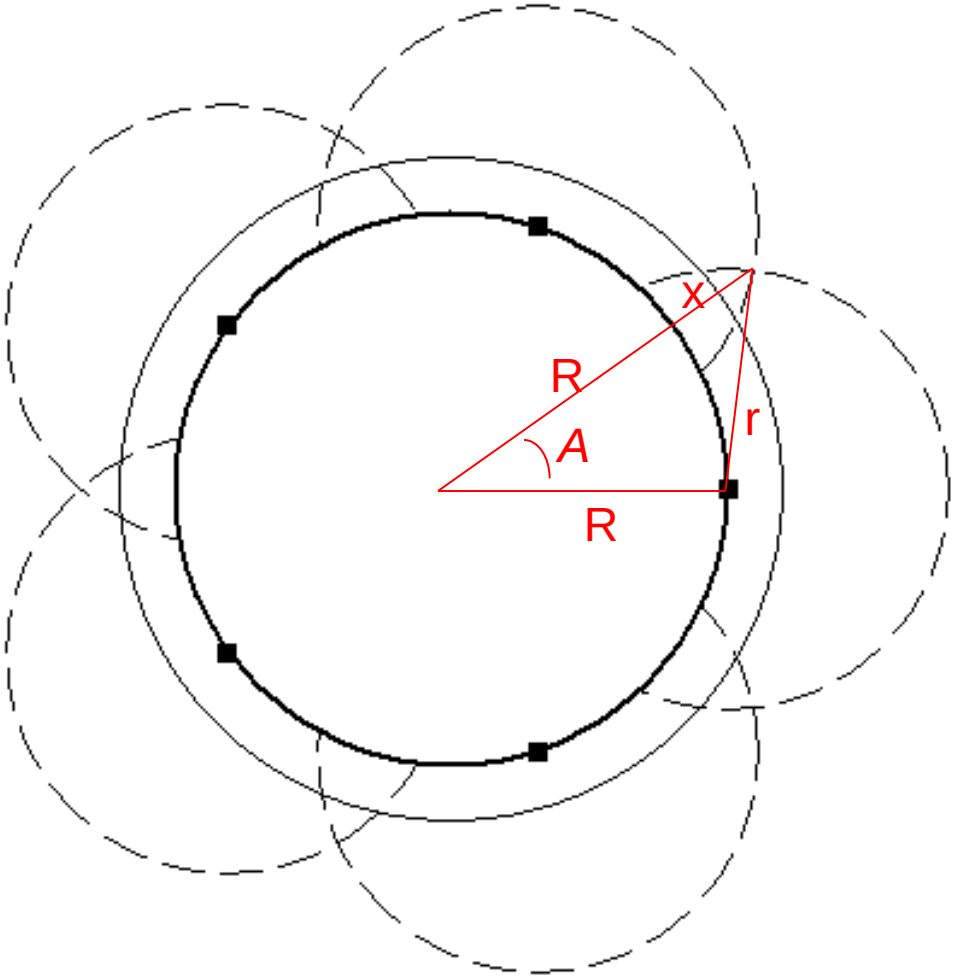
Треугольник *AMO* равнобедренный с углом 60 градусов, то есть он равносторонний. Значит, длина отрезка *AO* тоже равна *x*.

Углы *MAC* и *OAB* равны, так как они дополняются до 60 градусов одним и тем же углом *CAO*.

Треугольники *OAB* и *OMC* равны (*OM=OA=x*, *AB=BC*, углы *MAC* и *OAB* тоже равны). Значит, *OB=MC=y*, то есть *MB=x+y*.

Вопросы?

Задача A - «Датчики»



Пусть N – количество датчиков

Угол $A = \pi/N$

По теореме косинусов:
$r^2 = (R+x)^2 + R^2 - 2\cdot(R+x)\cdot R\cdot\cos(\pi/N)$

Нужно найти минимальное N, при котором $x \geq d$.
Подставим в формулу d вместо x, выразим N, округлим результат вверх:

$$N = \text{ceil}(\pi / \arccos(R^2 + (R+d)^2 - r^2) / (2.0\cdot R\cdot(R + d))))$$

Вопросы?

Задача B - «Системы счисления»

При N =1 ответ равен 2 – частный случай.

Пусть k – искомое основание системы счисления.

Запись *100…0* означает, что $N = k^R$, где R – количество нулей.

Переберём все возможные значения R (не больше 60, поскольку даже при наименьшем возможном k=2 имеем $2^{60} > 10^{18}$).

Для каждого p найдём k *по формуле* $k = \log_R(N)$ и проверим его на целочисленность.

```
if (n == 1) {
  std::cout << 2;
  return 0;
}
long long answer = n;
for (int r = 2; (1LL << r) <= n; r++) {
  long long k = exp(log(n) / r) + 0.5;
  if (lpow(k, r) == n) // длинная целая степень
    answer = k;
}
```

Решение другим способом

Наша задача состоит в том, чтобы найти максимальное R

Разложим N на простые множители:

$$N=p_1^{s1}*p_2^{s2}*\ldots*p_m^{sm}$$

Будем искать делители до 10^6 (а не до 10^9). Если при этом N полностью не будет разложено на множители, то **R<=2**, то есть N может быть только **полным квадратом**. Проверим этот факт непосредственно.

Далее найдем число R:

$$R=GCD(s_1, s_2, \ldots, s_m).$$

А потом число K (которое и будет ответом):

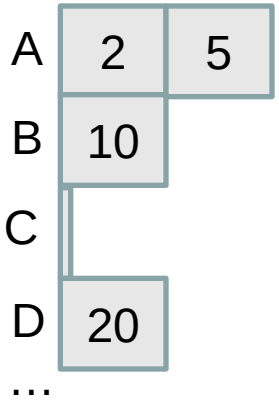
$$K=p_1^{s1/R}*p_2^{s2/R}*\ldots*p_m^{sm/R}$$

Вопросы?

Задача C – «Контест»

Задача решается методом динамического программирования

Структура данных. Оставим от названий задач только первые буквы. Для каждой буквы будем хранить сложности всех задач, которые на эту букву начинались. Создадим двухмерный массив *problems*, где строки могут иметь разную длину (массив из векторов). Строки отсортируем по неубыванию:



Пусть *count*[*i*][*j*] – количество контестов, которые заканчиваются на букву *i* и содержат самую сложную задачу со сложностью *problems*[*i*][*j*].

Переход ДП:
count[i][j] = *count*[i-1][1] + *count*[*i*-1][2]+…+*count*[*i*-1][*k*], где *k* – максимальный индекс, при котором сложность предыдущей задачи не превышает сложности текущей, т.е. *problems*[*i*-1][*k*]<=*problems*[*i*][*j*]

Для эффективного вычисления *count* заметим, что сумму можно не считать каждый раз заново, а сохранять предыдущую и просто продолжать её увеличивать. Тогда каждая из 26 строк вычислится за линейное время.

```
std::vector<long long> problems[26], count[26];
//...
std::sort(problems[0].begin(), problems[0].end());
count[0].resize(problems[0].size(), 1);
for (i = 1; i < 26; i++) {
  std::sort(problems[i].begin(), problems[i].end());
  if (problems[i].size() == 0) break;
  count[i].resize(problems[i].size());
  long long sum = 0; // накапливаемая сумма
  int prev = 0; // текущее положение в предыдущей строке
  for (int j = 0; j < (int)problems[i].size(); j++) {
     while (prev < (int)problems[i - 1].size()
     && problems[i - 1][prev] <= problems[i][j]) {
       sum += count[i - 1][prev];
       prev++;
     }
     count[i][j] = sum;
  }
  if (sum == 0) break;
}
long long answer = 0;
for (int j = 0; j < (int)count[i - 1].size(); j++)
  answer += count[i - 1][j];
```

Вопросы?

Задача D – «Деревья»

Способ решения: попытаться подобрать формулу ;)

1). Напишем несложное переборное решение, которое нам даст ответы при *N* примерно до 12. По ним будем проверять правильность подбираемой формулы.

2). Заметим, что ответ зависит только от *a* и *b*, где *a* – количество четных чисел от 1 до *N*, *b* – количество нечётных:

$$a = N \text{ div } 2,\ b = N - a$$

3). Заметим, что формула симметрична относительно *a* и *b* (если заменить *a* на *b* и *b* на *a*, то формула не изменится).

4). В условии задачи есть подсказка – формула Кэли: N^{N-2}. Можно предположить, что искомая формула на неё чем-то похожа.

Ответ:

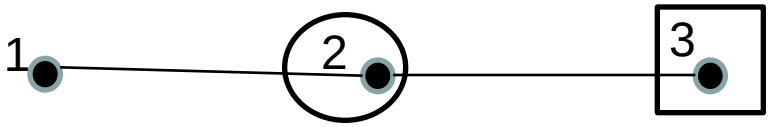
$$a^{b-1} \times b^{a-1}$$

Вывод формулы

Дальнейшие рассуждения являются развитием идеи, изложенной здесь: www.mccme.ru/nir/uir/gorbunov.doc

Определение. Помеченным деревом называется дерево, в котором выделены две вершины – чётная и нечётная. Первую будем называть левым концом, вторую – правым. Левый конец будем далее отмечать кружочком, правый – квадратом.

Пример помеченного дерева на 3 вершинах:



Пусть F_n – количество помеченных деревьев на n вершинах, а K_n- количество деревьев на n вершинах графа.

Лемма 1. $F_n = K_n \cdot a \cdot b$. Действительно, в каждом дереве мы левый конец можем выбрать a способами, правый – b способами.

Теорема 1. Количество деревьев, которые можно построить на a четных и b нечётных вершинах, где каждое ребро соединяет вершины разной четности, равно $a^{b-1} \times b^{a-1}$

По лемме 1 $F_n = K_n \cdot a \cdot b$, то есть нужно доказать, что $F_n = a^b \times b^a$.

Рассмотрим множество помеченных деревьев на *a* четных и *b* нечётных вершинах и множество ориентированных графов, у которых:

- из каждой вершины идет ровно одно ребро;
- между двумя вершинами не более двух ребер;
- каждое ребро соединяет вершины разной чётности.

Количество способов построить такой ориентированный граф – $a^b \times b^a$, поскольку из каждой четной вершины можно провести ребро *b* способами, а из каждой нечётной – *a* способами.

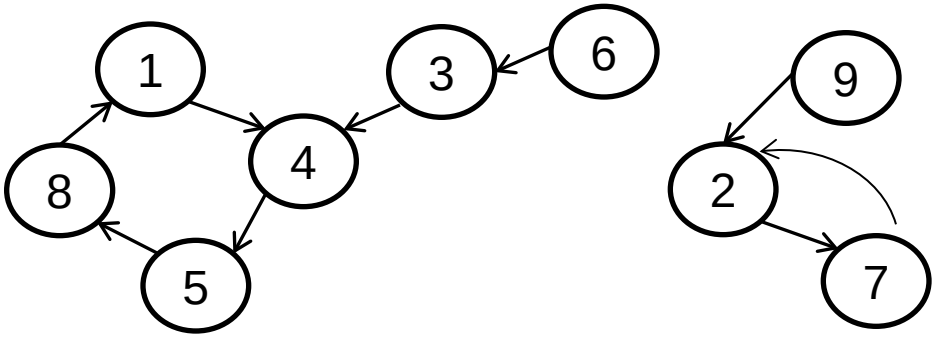
Докажем, что эти множества равномощны. Для это установим взаимно однозначное соответствие между такими графами и помеченными деревьями, в а для этого построим:

1) алгоритм превращения графа в помеченное дерево
2) алгоритм превращения помеченного дерева в граф

Применение двух этих алгоритмов подряд должно восстанавливать исходный объект.

1. Как превратить граф в помеченное дерево?

По сути, каждый граф рассматриваемого вида состоит из нескольких изолированных циклов, к которым еще могут присоединяться деревья

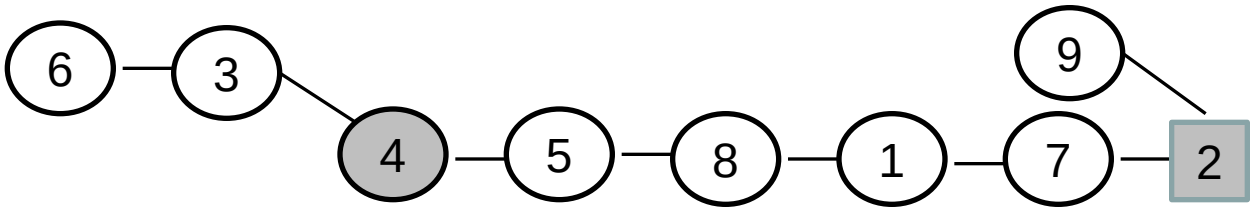


Упорядочим все циклы графа по нечётной вершине с наименьшим номером, входящей в цикл. Выпишем для каждого цикла порядок обхода его вершин, начиная с вершины, следующей за наименьшей нечётной (тогда наименьшая нечетная станет последней). В примере получатся следующие две последовательности:

4 5 8 1 и 7 2

Сольём все последовательности в одну общую: 4 5 8 1 7 2

Теперь каждые соседние члены соединим ребром, первый член сделаем левым концом, правый член – правый концом. Все остальные вершины соединим точно так же, как в графе. Получим помеченное дерево:



Почему гарантируется отсутствие циклов, т.е. получится именно дерево? Поскольку мы выбросили по одному ребру из каждого цикла.

Почему в полученном дереве все рёбра будут соединять только чётные вершины с нечетными? Поскольку в графе все циклы могут иметь только чётное число вершин, и их четность чередуется, то и теперь чередование выполняется.

2). Как из помеченного дерева восстановить исходный граф?

Для этого рассмотрим в дереве путь от левого конца до правого. Ищем на этом пути наименьшую нечетную вершину. Всё, что до неё, даёт нам первый цикл. Затем ищем следующую наименьшую нечетную вершину – получаем следующий цикл, и так далее. Тем самым исходный граф восстановится однозначно.

Таким образом, взаимно однозначное соответствие между графами такого вида и помеченными деревьями установлено, теорема доказана.

Можно ли эту задачу решить методом ДП? Открытый вопрос.

Вопросы?

Задача E – «Вычитание квадратов»

Идея: выигрышные и проигрышные позиции

Выигрышная позиция – позиция, находясь в которой, игрок, кому сейчас нужно делать ход, может выиграть одним ходом или попасть в проигрышную позицию.
Проигрышная позиция – позиция, из которой можно сходить только в выигрышную позицию.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	2	1	1	2	1	2	1	1	2	1	2	1	1	2	1	2	1	1	2

Детали реализации

Если действовать по определению, то при $n > 10^6$ можно получить TL.

```
win[0] = false;
win[1] = true;
win[2] = false;
win[3] = true;
for (int i = 4; i <= n; i++) {
    int q = sqrt(i);
    while (q > 0) {
        if (i - q * q >= 0) {
            if (!win[i - q * q]) {
                win[i] = true;
                q = -1;
            }
            else {
                q--;
            }
        }
    }
}
```

Заметим, что проигрышных позиций значительно меньше, чем выигрышных.

Целесообразно для каждой проигрышной позиции строить все выигрышные, из которых в неё можно попасть.

```
for (int i = 0; i <= n; i++) {
    if (!win[i]) {
        for (int q = 1; i + q * q <= n; q++) {
            win[i + q * q] = true;
        }
    }
}
int z = 0;
for (int i = 1; i <= n; i++) {
    if (!win[i]) {
        z++;
    }
}
```

Использование преподсчёта

Исследование показало, что при подавляющем большинстве входных данных, если `n mod 5 = 1` или `n mod 5 = 3`, то выигрывает первый игрок. Соответственно, можно явно прописать исключения и идти с бо́льшим шагом и прокручивать меньше витков внутреннего цикла.

Но для полного решения задачи это необязательно.

Вопросы?

Задача F – «Разложение многочлена на множители»

Краткие теоретические сведения о круговых многочленах

Круговой многочлен, или многочлен деления круга -

многочлен вида $$\Phi_n(x) = \prod_k \left(x - \xi_n^k\right)$$

где ξ_i - корни степени n из единицы,

а произведение берётся по всем натуральным числам k, меньшим n и взаимно простым с n.

Свойства

- Степень кругового многочлена

$$\deg \Phi_n(x) = \varphi(n)$$

где $\varphi(n)$ - функция Эйлера (количество чисел меньше n и взаимно простых с ним).

- Круговой многочлен удовлетворяет соотношению

$$\prod_{d/n} \Phi_d(x) = x^n - 1$$

Примеры

$$\Phi_1(x) = x - 1$$
$$\Phi_2(x) = x + 1$$
$$\Phi_3(x) = x^2 + x + 1$$
$$\Phi_4(x) = x^2 + 1$$
$$\Phi_8(x) = x^4 + 1$$
$$\Phi_{12}(x) = x^4 - x^2 + 1$$

Алгоритм получения коэффициентов многочлена

Рассмотрим нахождение $\Phi_n(x)$ при небольших значениях *x* и обобщим выявленные закономерности.

$\Phi_1(x) = x - 1$

Поскольку $x^2 - 1 = \Phi_1(x) \cdot \Phi_2(x)$, $\Phi_2(x) = \dfrac{x^2 - 1}{x - 1} = x + 1$

Поскольку $x^3 - 1 = \Phi_1(x) \cdot \Phi_3(x)$, $\Phi_3(x) = \dfrac{x^3 - 1}{x - 1} = x^2 + x + 1$

То есть для нахождения $\Phi_n(x)$

мы многочлен $x^n - 1$ делим на все $\Phi_d(x)$,

где *d* - положительный делитель числа *n*, включая единицу, но не включая *n*

Например, $\Phi_6(x) = (x^6 - 1) / (\Phi_1(x) \cdot \Phi_2(x) \cdot \Phi_3(x))$

Реализация: структура

```
struct Polynom {

    std::vector<int> coef;

    Polynom(const std::vector<int> & coef)
        : coef(coef)
    {
        assert(coef.size() > 0);
        assert(coef.back() > 0);
    }

    Polynom(){}

    static Polynom xpnm1(int n) {
        assert(n > 0);
        std::vector<int> coef(1 + n, 0);
        coef[0] = -1;
        coef[n] = 1;
        return Polynom(coef);
    }

    void println() const {
        for (int i = (int)coef.size() - 1; i >= 0; i--) {
            printf("%d", coef[i]);
            if (i > 0) {
                printf(" ");
            }
        }
        printf("\n");
    }
};
```

Реализация: оператор деления

```
Polynom operator / (const Polynom& left, const Polynom& right) {
    assert(left.coef.size() > right.coef.size());
    std::vector<int> dividend = left.coef;
    std::vector<int> res(left.coef.size() - right.coef.size() + 1, 0);
    for (int i = (int)res.size() - 1; i >= 0; i--) {
        assert(dividend[i + (int)right.coef.size() - 1] % right.coef.back() == 0);
        res[i] = dividend[i + (int)right.coef.size() - 1] / right.coef.back();
        assert(-17826 <= res[i] && res[i] <= 17826);
        for (int j = 0; j < (int)right.coef.size(); j++) {
            dividend[i + j] -= res[i] * right.coef[j];
        }
        assert(dividend[i + (int)right.coef.size() - 1] == 0);
    }
    return Polynom(res);
}
```

Вопросы?

Задача G – «Ботанический сад»

1. Простое решение с пределом времени.

Заметим, что «семейных» билетов типа C и D нужно купить как можно больше (так как покупка такого билета заведомо не хуже, чем покупка - отдельных билетов на троих человек).

Отсюда решение:

- перебираем количество билетов типа C.
- для оставшихся людей берём как можно больше билетов типа D.
- наконец, на оставшихся людей берём индивидуальные билеты

Пусть xc – количество билетов типа C. Тогда количество билетов остальных видов определяется так:

xd = min((m - xc) / 2, n - 2*xc),

xa = m - xc - 2*xd,

xb = n - 2*xc - xd

Сложность: линейная. При ограничении 10^9 - слишком медленно.

2. Преобразуем предыдущее решение в эффективное

Запишем функцию итоговой стоимости от количества билетов типа C:

$$cost(xc) = c * xc + d * F(xc) + (m - xc - 2*F(xc))*a + (n - 2*xc - F(xc)) * b,$$

где $F(xc) = \min((m - xc) \text{ div } 2, n - 2*xc)$

Что собой представляет график функции cost?
При увеличении xc на единицу первый член увеличивается на некоторую величину, остальные – уменьшаются. То есть функция либо возрастает, либо убывает. Такое поведение функции нарушается только в одном месте – когда $(m - xc) \text{ div } 2 = n - 2*xc$ – в этот момент может произойти скачок. При этом $xc \approx (2n - m) \text{ div } 3$.

Таким образом, ответ стоит искать только на концах допустимого интервала значений xc, а также в окрестности точки $x = (2n - m) \text{ div } 3$.

Реализация

```
int left  = 0;
int right = std::min(m, n / 2);
solve(left);
solve(right);
if (2 * n > m) {
  int mid = (2 * n - m) / 3;
  for (int i = mid - 1; i <= mid + 1; i++) {
    if (i > left && i < right)
      solve(i);
  }
}

void solve(int xc) {
  // добираем по максимуму билетами "2 взрослых, 1 ребёнок"
  // и одиночных - сколько уж останется
  int xd = std::min((m - xc) / 2, n - 2*xc);
  long long cost = c*xc + d*xd + (m - xc - 2*xd)*a + (n - 2*xc - xd)*b;
  minCost = std::min(minCost, cost);
}
```

Вопросы?

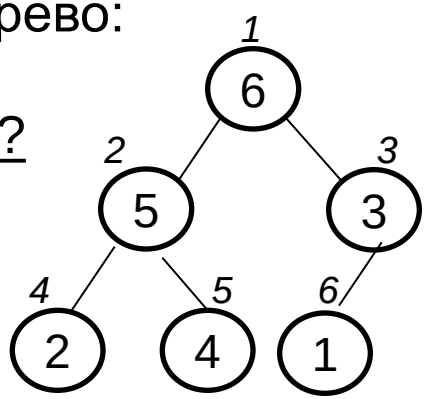
Задача H – «Пирамиды»

Решение: ДП + комбинаторика

Любую пирамиду можно представить в виде ***почти полного бинарного дерева***, в котором:

- каждый уровень дерева заполнен полностью кроме, возможно, последнего уровня, который заполнен без пропусков слева направо;
- значение в любой вершине не меньше любого из своих детей;
- элементы нумеруются по уровням (сверху вниз слева направо).

Пример: перестановке 6 5 3 2 4 1 соответствует дерево:



<u>Сколько будет пирамид из *N* различных элементов?</u>

Самый большой элемент становится корнем.

Вычислим размер левого и правого поддерева (можно отдельно рассмотреть три случая:

- полное дерево (2^x-1 элементов)
- левое поддерево полное, правое неполное
- левое поддерево неполное, в правом – весь нижний уровень пуст

Пусть size1 и size2 – размеры поддеревьев.
Обозначим через *f*(*i*) количество пирамид размера *i*.

Переход ДП:

f(1+size1+size2) = f(size1) * f(size2) * C(size1+size2, size1),
где C(x, y) означает количество сочетаний из x по y.

Пояснение: C(size1+size2, size1) – это количество способов выбрать элементы для помещения в левое поддерево (а все остальные тогда пойдут в правое). И для каждого варианта перемножаем количество способов построить левое поддерево и правое.

Оценка сложности:
Матрицу для значений C(i, j) можно построить по свойству треугольника Паскаля c(i, j) = c(i – 1,j - 1) + c(i – 1,j).
Время построения – $O(N^2)$.
После этого ДП работает за линейное время.
Итого сложность – $O(N^2)$.

Вопросы?

Задача I –
«Цепные дроби»

Определение

Цепная дробь – это дробь вида

$$[a_0;\ a_1, a_2, a_3, \dots, a_n, \dots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 \dots}}}$$

Теорема Лагранжа

Если цепная дробь периодична, начиная с некоторого места, то число, ей соответствующее – квадратичная иррациональность,
то есть является корнем квадратного уравнения с целыми коэффициентами.
Обратное утверждение: для любой квадратичной иррациональности цепная дробь периодична, начиная с некоторого места.

Пример, поясняющий суть решения

$$\sqrt{7} = 2 + \sqrt{7} - 2 = 2 + \frac{1}{\frac{1}{\sqrt{7}-2}} = 2 + \frac{1}{\frac{\sqrt{7}+2}{(\sqrt{7}+2)(\sqrt{7}-2)}} = 2 + \frac{1}{\frac{\sqrt{7}+2}{3}} =$$

$$= 2 + \frac{1}{1 + \frac{\sqrt{7}-1}{3}} = 2 + \frac{1}{1 + \frac{1}{\frac{3}{\sqrt{7}-1}}} = 2 + \frac{1}{1 + \frac{1}{\frac{\sqrt{7}+1}{2}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{\sqrt{7}-1}{2}}} =$$

$$= 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\frac{2}{\sqrt{7}-1}}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\frac{\sqrt{7}+1}{3}}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{\sqrt{7}-2}{3}}}} =$$

$$= 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\frac{3}{\sqrt{7}-2}}}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\frac{\sqrt{7}+2}{1}}}}} = 2 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{4 + \cdots}}}} =$$

$$= [2; 1, 1, 1, 4 \ \ldots]$$

Когда снова получим числитель и знаменатель, которые были на первом шаге, то период определён

Свойства периодических цепных дробей

1. У цепных дробей для чисел вида $\sqrt{n}$ нет предпериода.
2. Последнее число в периоде равно удвоенной целой части числа $\sqrt{n}$.
3. Последовательность чисел до удвоенной целой части числа $\sqrt{n}$ симметрична относительно своего центра.

Реализация

```
var
  a, b, c: array[0..100000] of longint;
  w, i, j: longint;

begin
   readln(w);
   i := 0;
   a[0] := 0;
   while (a[0] + 1) * (a[0] + 1) <= w do
      inc(a[0]);
   if w > a[0] * a[0] then begin
      b[0] := a[0];
      c[0] := w - a[0] * a[0];
      a[1] := trunc((a[0] + b[0]) / c[0]);
      i := 1;
      while a[i] < 2 * a[0] do begin
         b[i] := c[i - 1] * a[i] - b[i - 1];
         c[i] := round((w - b[i] * b[i]) / c[i - 1]);
         i := i + 1;
         a[i] := trunc(round(a[0] + b[i - 1]) / c[i - 1]);
      end;
      write('sqrt(', w, ') = [', a[0]);
      for j := 1 to i do
        if j = 1 then
            write('; ', a[j])
         else
            write(', ', a[j]);
      writeln(']');
   end
   else
      writeln('sqrt(', w, ') = [', a[0], ']');
end.
```

Альтернативное решение с помощью длинной арифметики

Элементы цепной дроби можно получать так:

$x = \sqrt{n}$

$$a_0 = \lfloor x \rfloor, x_0 = x - a_0,$$

$$a_1 = \left\lfloor \frac{1}{x_0} \right\rfloor, x_1 = \frac{1}{x_0} - a_1,$$

$$\dots$$

$$a_n = \left\lfloor \frac{1}{x_{n-1}} \right\rfloor, x_n = \frac{1}{x_{n-1}} - a_n,$$

Вычисления заканчиваются, когда мы снова получили x_0.

Сложность 1: нужна высокая точность – в районе 800 знаков после десятичной точки. В Java можно использовать класс BigDecimal.

Сложность 2: у класса BigDecimal нет метода sqrt – нужно писать вручную (можно использовать метод Ньютона).

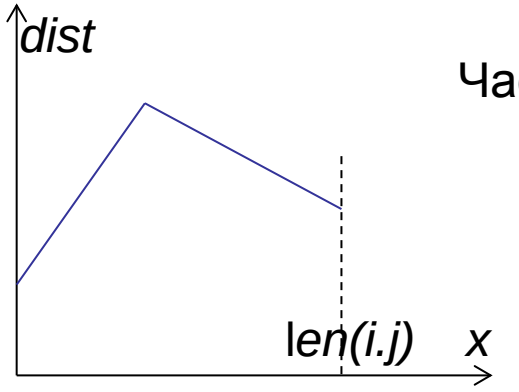
Вопросы?

Задача J – «Склад»

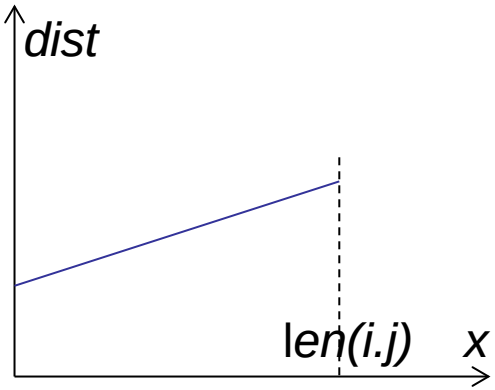
Решение.
Переберём все рёбра. Определим наилучшее положение склада на каждом ребре, из них выберем лучшее.

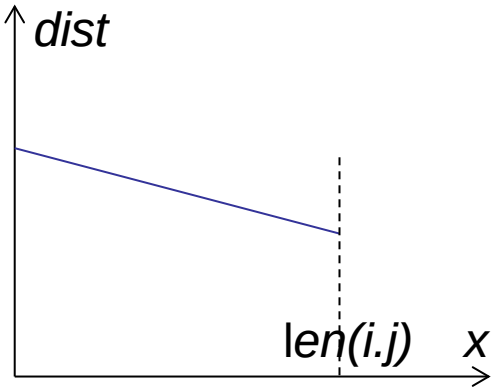
Пусть взято очередное ребро *i-j*. Временно вычеркнем ребро из графа. Запустим алгоритм Дейкстры от вершины *i* и от вершины *j* – получим массивы расстояний *di* и *dj* от вершин *i* и *j* до остальных вершин.

Пусть x – искомое расстояние на ребре от точки *i* до точки *j*. Для каждой вершины *w* определим, как меняется расстояние до неё от точки внутри ребра при изменении *x* от *0* до длины ребра len(*i*,*j*).
При *x* = 0 dist(x, w) = min(di[w], len(i, j) + dj[w])
При *x* = len(i, j) dist(x, w) = min(dj[w], len(i,j) + di[w])
При изменении *x* от 0 до len(i, j) функция dist(x,w) какое-то время растёт, а потом начинает убывать. График имеет вид:



Частные случаи:



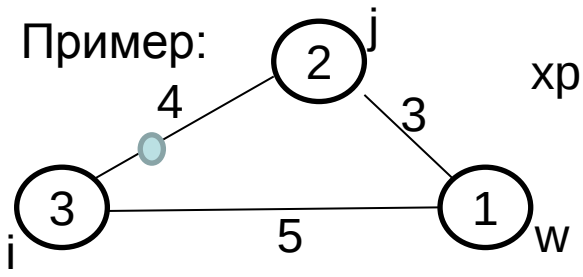


Точка перелома графика функции – это точка, из которой одинаковые расстояниы до вершины *w*, идём ли мы в неё через *i*, или через *j*:

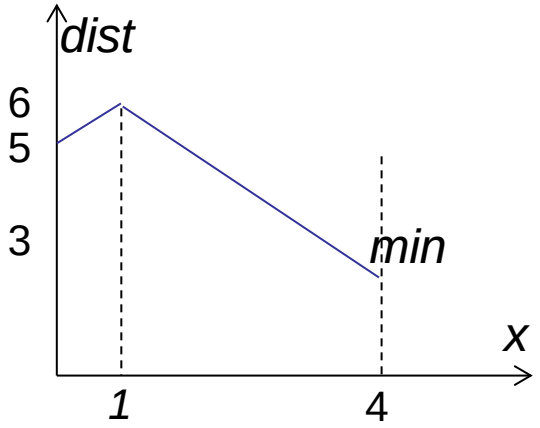
xp[w] = (dj[w] - di[w] + len(i, j)) / 2

y[w] = x[w] + di[w]

Пример:



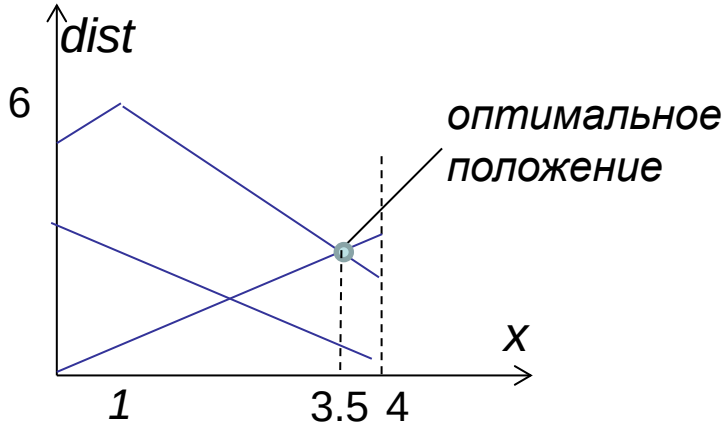
xp[1]=(3-5+4)/2=1



Добавим к этому примеру графики для остальных вершин (w=3 и w=2):

xp[3] = (4-0+4)/2 = 4

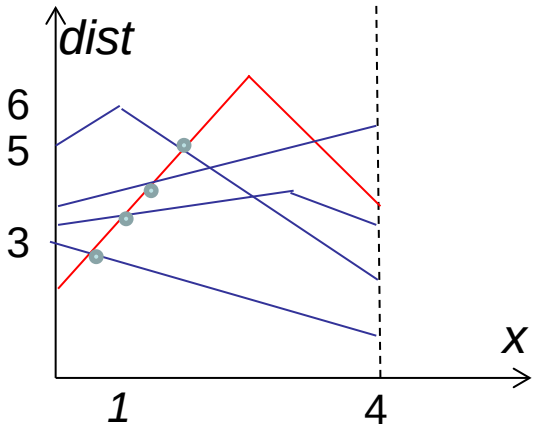
xp[2] = (0-4+4)/2 = 0



То есть нам нужно найти самую низкую точку, выше которой ничего нет

Как это сделать эффективно?

Поиск оптимальной точки за квадратичное время



Перебираем каждый из графиков, берём в нём первую часть (линию, идущую снизу вверх).

Теперь для него перебираем оставшиеся графики. В каждом из них берём вторую часть (линию, идущую сверху вниз) и ищем точки пересечения с выбранным графиком. Самая правая точка – возможный ответ (только нужно убедиться, что выше неё ничего нет).

Поскольку Дейкстра работает за $O(N^2)$, то итоговая сложность решения – $O(N^4)$.

Вопросы?