Сбалансированные деревья. AVL-дерево

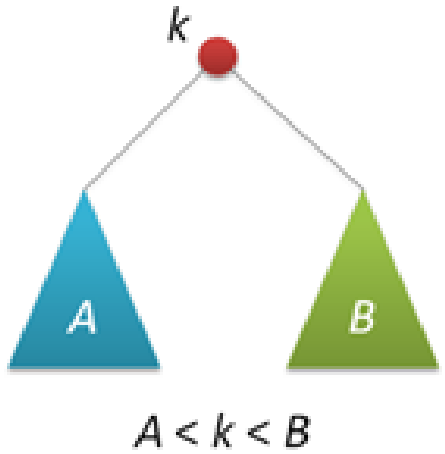
Булгаков Илья, Гусев Илья

Московский физико-технический институт

Москва, 2023

Содержание

Деревья поиска



Основные методы

1. insert
2. search
3. delete

Асимптотика: $O(h)$

Деревья поиска

Виды самобалансирующихся деревьев

- AVL-дерево (рассматривали прошлый раз)
- B-дерево
- Splay-дерево
- Красно-черное дерево

AVL-деревья

Что такое AVL-дерево?

Один из первых видов сбалансированных двоичных деревьев поиска, придуманное в 1962 году советскими учеными Адельсон-Вельским и Ландисом.
Свойства:

- Является двоичным деревом поиска. Поиск работает традиционно.
- Для любого узла дерева высота его правого поддерева отличается от высоты левого поддерева не более чем на единицу. Доказано, что в этом случае высота дерева логарифмически зависит от числа его узлов.

AVL-деревья

Структура узла

В узел добавляется дополнительная переменная, которая отвечает за баланс дерева. В традиционной реализации это balance factor, который показывает, насколько отличаются высоты детей: -1, 0 и 1. Можно вместо этого хранить высоту дерева.

```
struct CNode // Узел AVL-дерева
{
        int Key;
        unsigned char Height; // Высота поддерева с корнем в данном узле
        CNode* Left;
        CNode* Right;
        CNode(int k): Key(k), Left(0), Right(0), Height(1) {}
};
```

AVL-деревья

Повороты

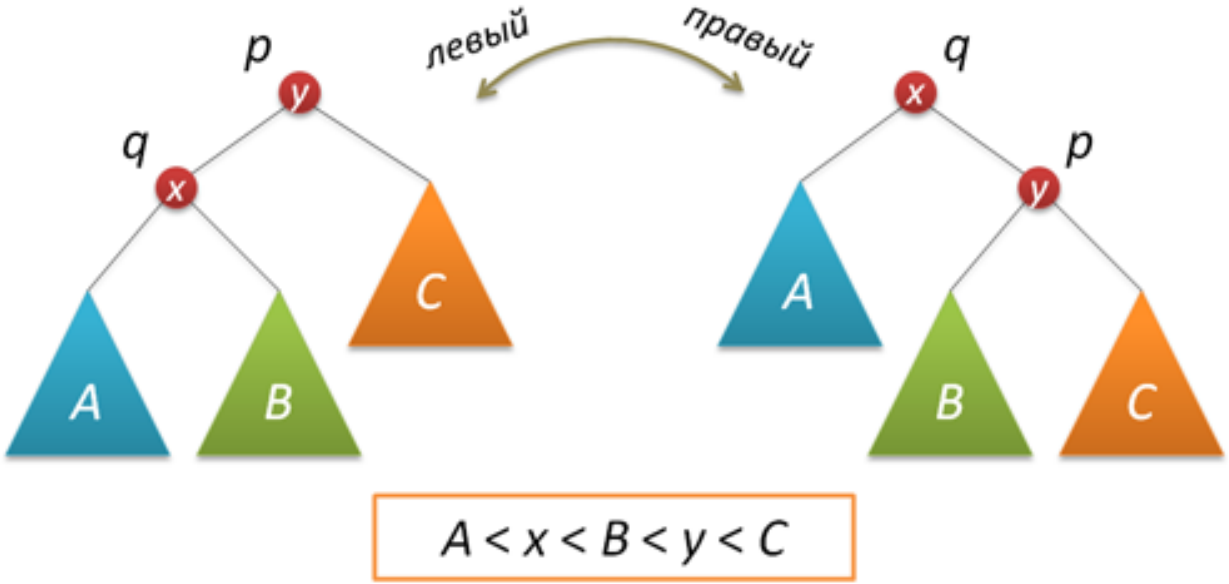
Введем некоторые типовые операции над деревьями - повороты. Они применяются не только в AVL деревьях, но и в других деревьях.
Цель поворотов - исправление ситуаций возникновения дисбаланса в высоте узлов.
Рассмотрим два вида поворотов. Левые и правые повороты симметричны, поэтому их можно рассматривать как один вид.

- Обычный левый/правый поворот
- Большой левый/правый поворот

AVL-деревья

Обычный левый/правый поворот

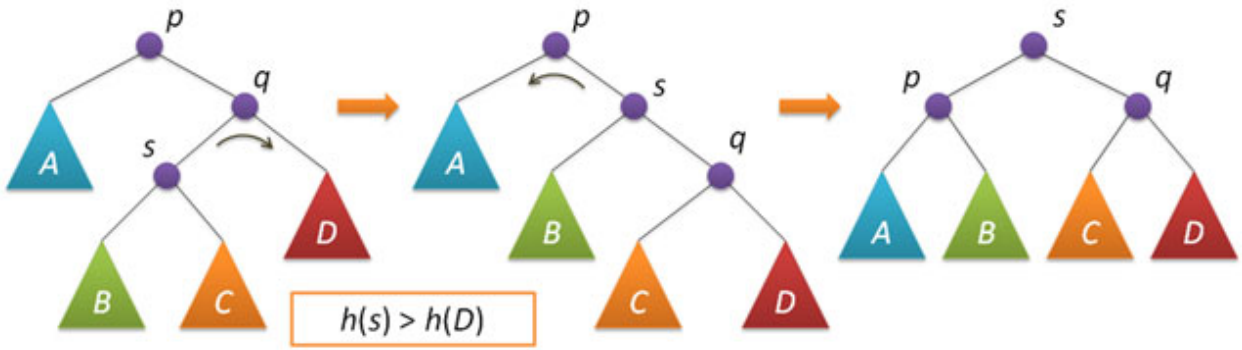
На примере правого поворота: поворот относительно узла p. Узел q идет наверх и становится родителем p. Правый ребенок q становится левым ребенком для p.
Обратите внимание, что свойства дерева поиска сохраняются!



AVL-деревья

Большой левый поворот

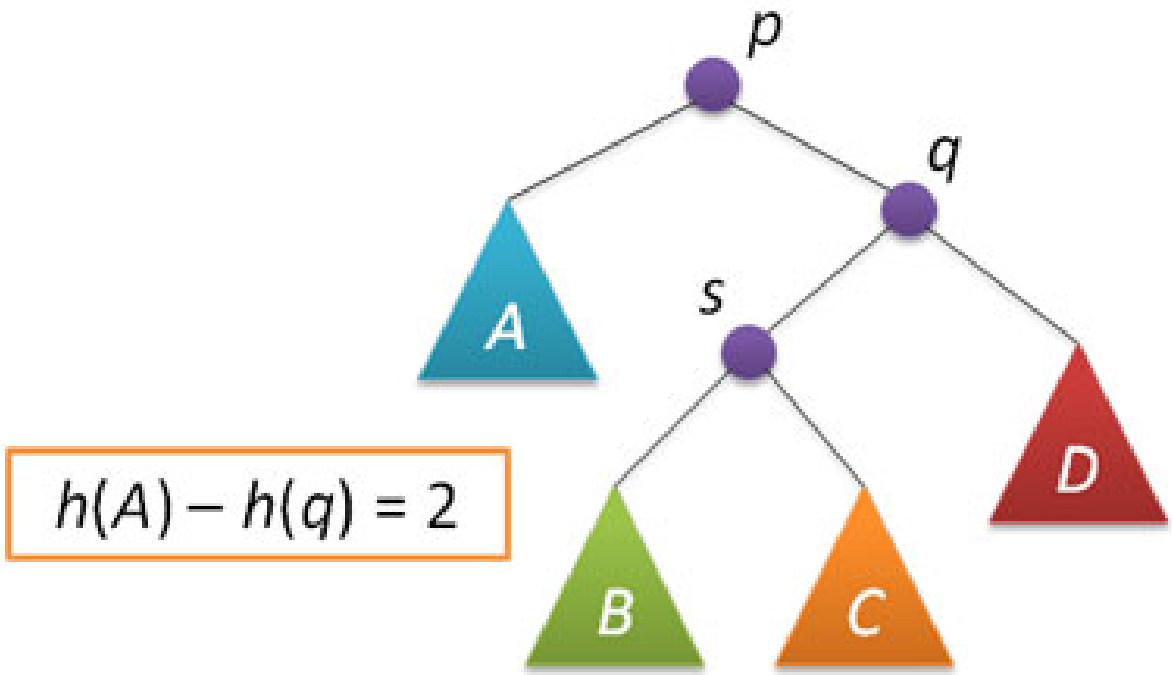
Большой левый поворот является комбинацией из двух поворотов - обычного правого и обычного левого.
Сначала делаем поворот относительно узла q (правый), а потом относительно узла p (левый).



AVL-деревья

Алгоритм

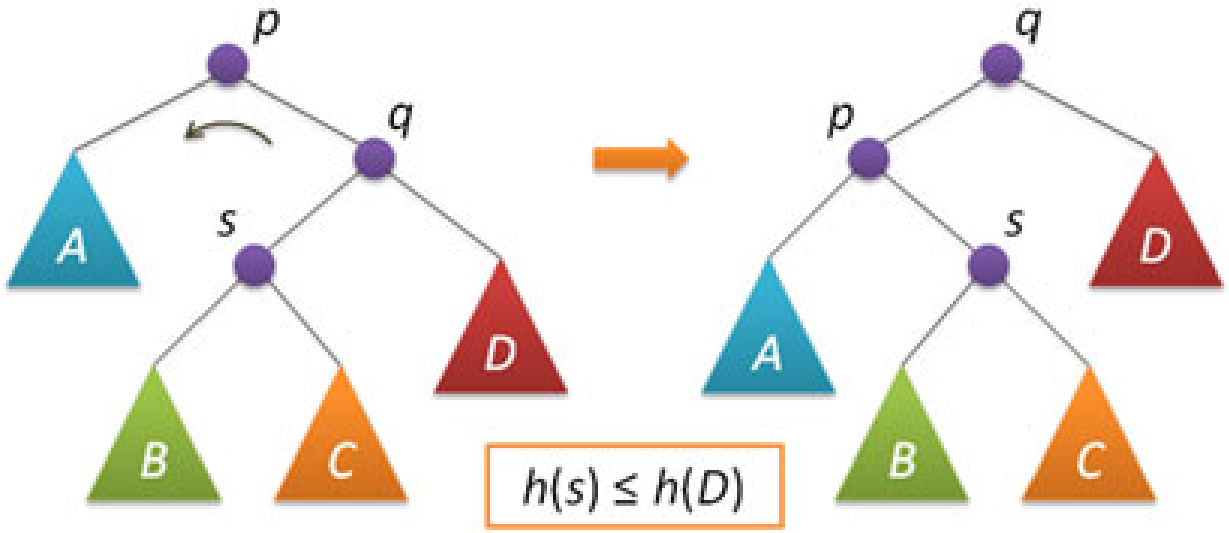
Идея алгоритма заключается в том, что если в дереве возникает дисбаланс и ограничение на высоты нарушается, то нужно рассмотреть несколько случаев и аккуратно выполнить повороты в дереве, которые восстанавливают баланс. Пусть дисбаланс в высотах A и q.



AVL-деревья

Алгоритм

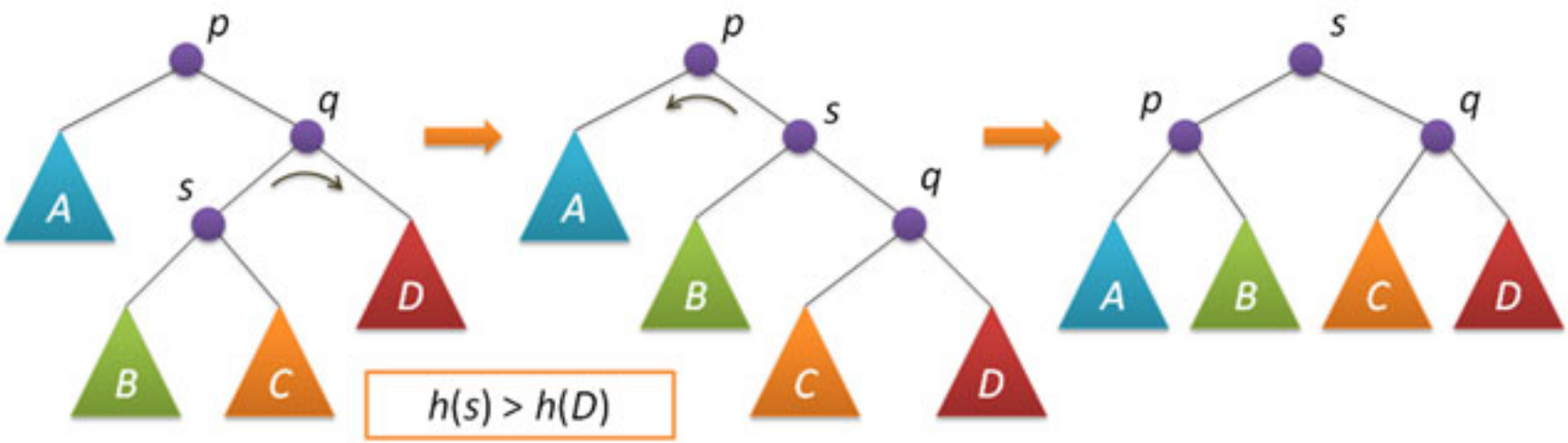
Основная вилка - понять, как соотносятся узлы S и D.
Вариант 1. h(S) <= h(D). Делаем простой поворот



AVL-деревья

Алгоритм

Вариант 2. h(S) > h(D). Делаем большой поворот



AVL-деревья

Операции: поиск, добавление

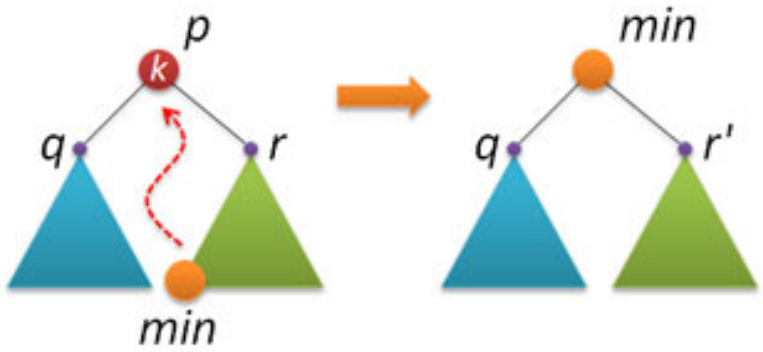
- Поиск: не отличается от обычного дерева поиска
- Добавление: вставляем, как в обычное бинарное дерево, запускаем балансировку при возвращении из рекурсии

```
CNode* insert(CNode* p, int k) // вставка ключа k в дерево с корнем p
{
        if( !p ) return new CNode(k);
        if( k<p->key ) {
                p->left = insert(p->left,k);
        } else {
                p->right = insert(p->right,k);
        }
        return balance(p); // балансировка при возвращении из рекурсии
}
```

AVL-деревья

Операции: удаление

- Удаление: та же методика, что и в обычном дереве поиска. Запускаем балансировку при возвращении из рекурсии



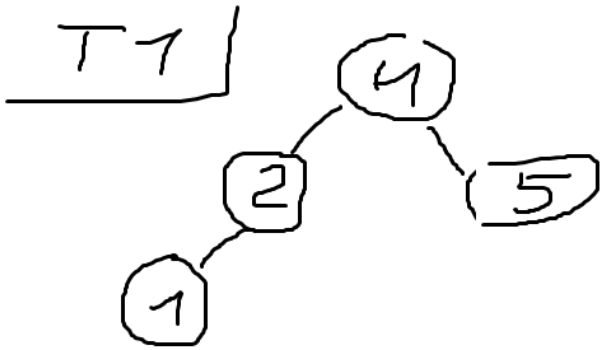
AVL-деревья

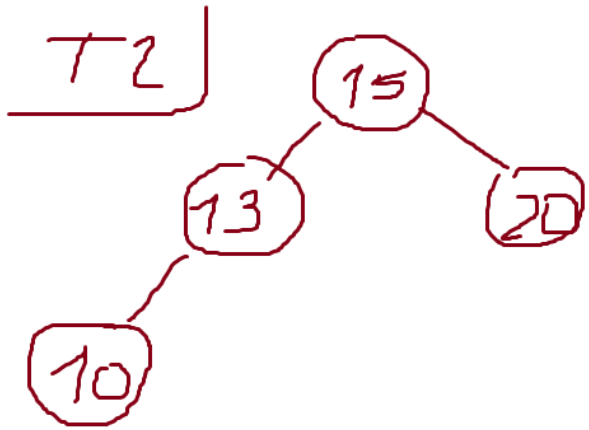
Задачка

6. Пусть даны два AVL-дерева T_1 и T_2, причём все ключи T_1 из них строго меньше всех ключей T_2. Предложите алгоритм построения AVL-дерева, множество ключей которого совпадает с объединением множеств ключей T_1 и T_2, за время $O(\log(|T_1| + |T_2|))$.

AVL-деревья

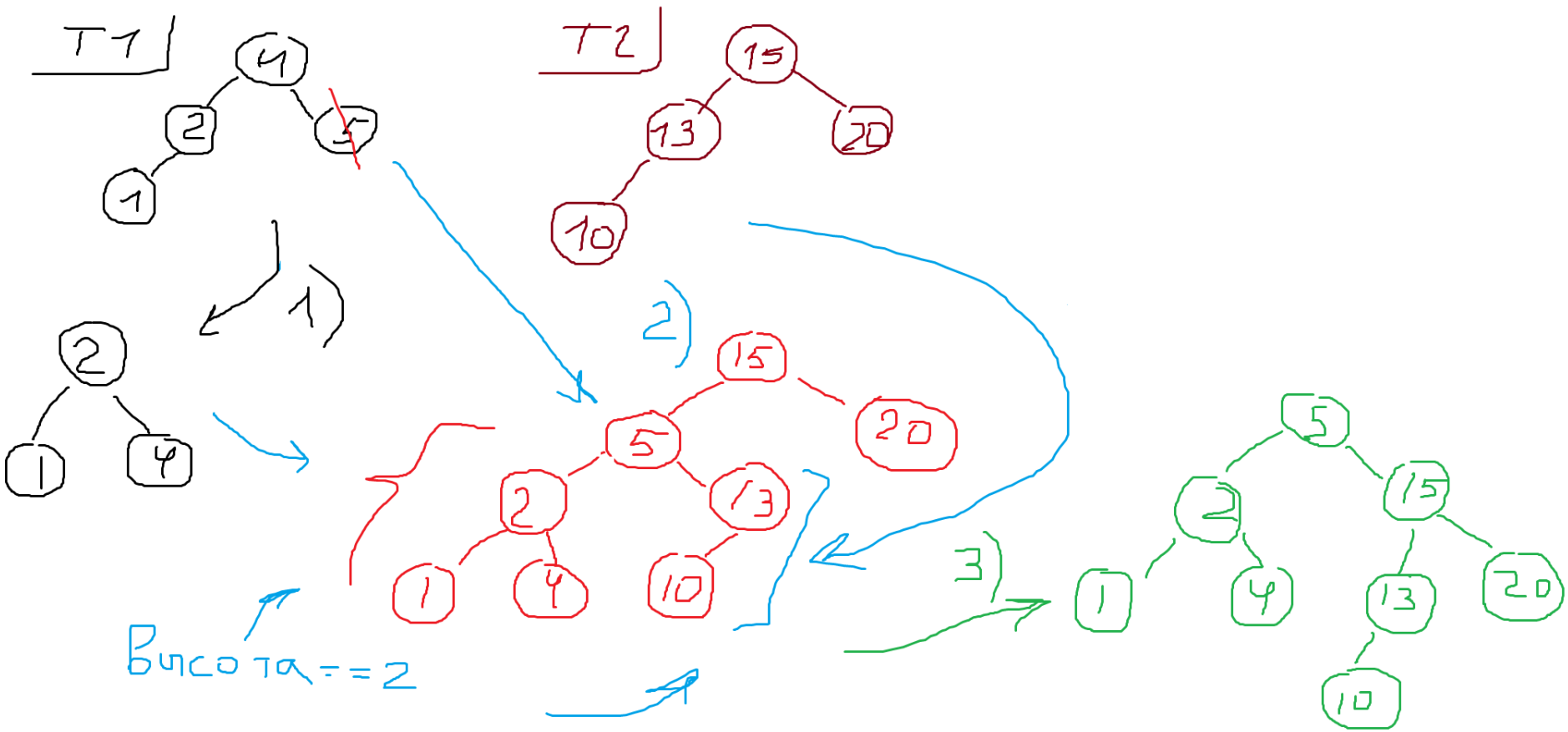
Задачка





AVL-деревья

Задачка



AVL-деревья

Задачка 2

Есть два множества отрезков на прямой, в них поступают запросы добавления. После каждого нужно сказать: сколько существует пар (отрезок из первого множества, отрезок из второго множества) таких, что они пересекаются.

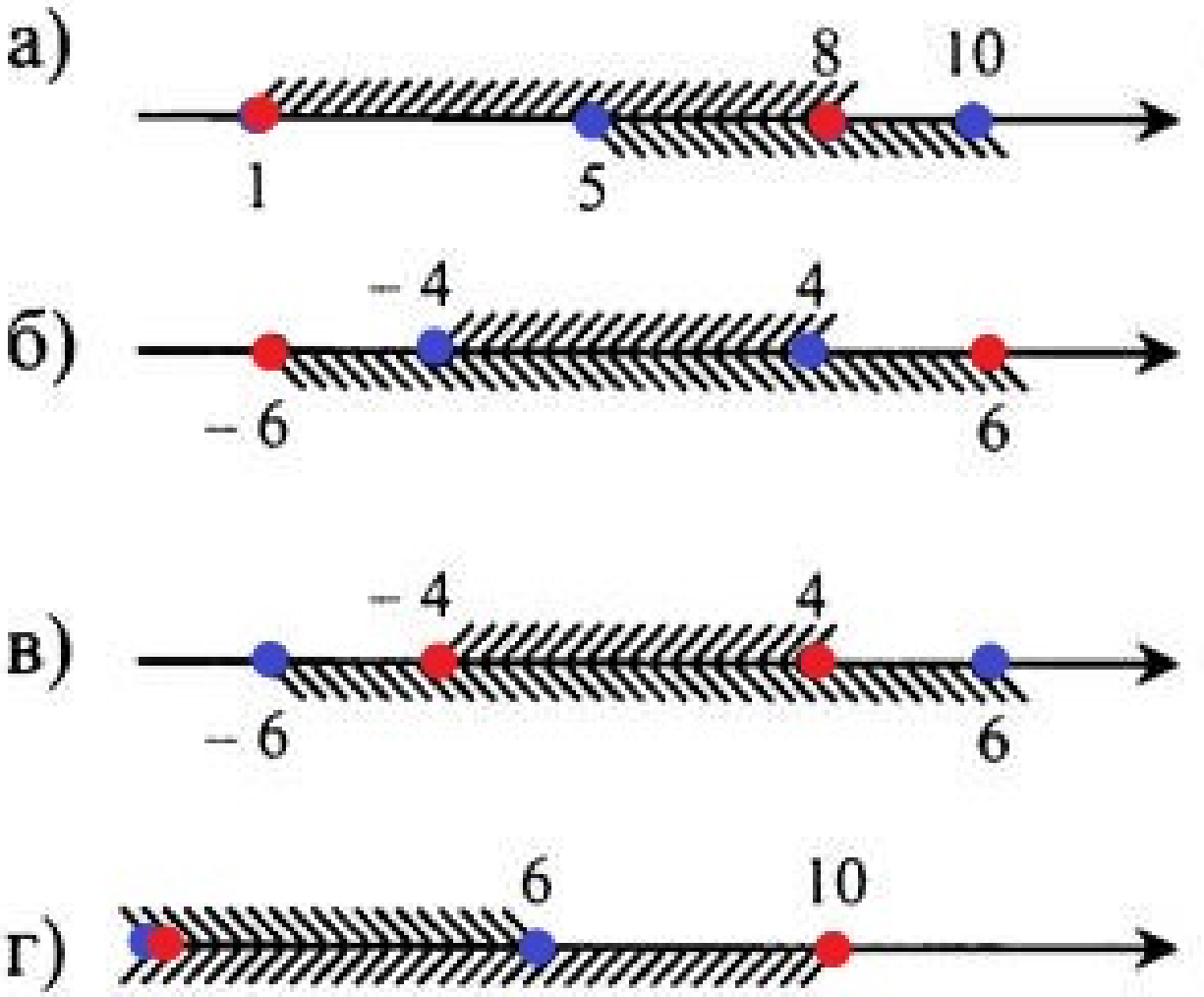
AVL-деревья

Задачка 2

```
struct SegmentSet // Узел AVL-дерева
{
        int size;
        CNode* leftEnd;
        CNode* rightEnd;
        SegmentSet* pairedSet;
};
struct Segment
{
    int left;
    int right;
}
```

AVL-деревья

Задачка 2



AVL-деревья

Задачка 2

```
struct CNode // Узел AVL-дерева
{
        int key;
        unsigned char height; // Высота поддерева с корнем в данном узле
        CNode* left;
        unsigned char leftSize;
        CNode* right;
        unsigned char rightSize;
        CNode(int k): key(k), left(0), leftSize(0), right(0),
            rightSize(0), height(1) {}
};
```

AVL-деревья

Задачка 2

```
CNode* insert(CNode* p, int k) // вставка ключа k в дерево с корнем p
{
        if( !p ) return new CNode(k);
        if( k<p->key ) {
                p->left = insert(p->left,k);
                p->leftSize++;
        } else {
                p->right = insert(p->right,k);
                p->rightSize++;
        }
        return balance(p); // балансировка при возвращении из рекурсии
}
```

AVL-деревья

Задачка 2

```
int SegmentSet::insert(Segment s) // вставка отрезка в множество
{
        leftEnd = insert(leftEnd, s.left);
        rightEnd = insert(rightEnd, s.right);
        int leftEnd_less = less(pairedSet->leftEnd, s.left) -
                less(pairedSet->rightEnd, s.left); // а + б
        int leftEnd_greater = greater(pairedSet->leftEnd, s.left) -
                greater(pairedSet->lefttEnd, s.right); // в + г
        return leftEnd_less + leftEnd_greater;
}
```

AVL-деревья

Задачка 2

```
int less(CNode* p, int k) // вставка ключа k в дерево с корнем p
{
        if( !p ) return 0;
        if( k < p->key ) {
                return less(p->left, k);
        } else {
                return less(p->right, k) + p->leftSize + 1;
        }
}
```

Полезные ссылки I

Про AVL-деревья на Хабре
https://habr.com/ru/post/150732/