ITIVITI

LET IT TRADE

Алгоритмы и структуры данных

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

УНИВЕРСИТЕТ ИТМО

Содержание курса

- Введение в теорию алгоритмов
- Алгоритмы сортировок
- **Структуры данных**
  - Линейные структуры
  - **Бинарные деревья поиска**
  - Хеши и хеш-функции
- Алгоритмы на графах
  - Обходы графов в ширину и глубину
  - Минимальные остовные деревья
  - Поиск кратчайших путей в графе

АТД «Таблица символов»

- Таблица символов (Symbol table) – структура данных для хранения пар вида <ключ, значение>
- Основные поддерживаемые операции:
  - Вставка нового элемента
  - Поиск элемента с заданным ключом
  - Удаление указанного элемента
- Что значит «указанного элемента»?
  - Указатель (handle): структуре данных нельзя свободно перемещать / удалять элемент
  - Ключ: операции с элементом реализуются через его поиск в структуре данных

АТД «Таблица символов»

ITIVITI

- Поддержка элементов с одинаковыми ключами
- Структура данных не поддерживает дубликаты:
  - Храним <ключ, список значений>
  - Поиск: возвращаем все элементы
  - Удаление: удаляем все элементы
  - Дубликатами управляет клиент
- Поддержка дубликатов в структуре данных:
  - Поиск: возвращаем произвольный элемент
  - Удаление: удаляем элемент по указателю (handle)
  - Механизм возврата всех дубликатов

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Strict Weak Ordering

ITIVITI

- Иррефлексивность: для всех элементов $!(a < a)$
- Транзитивность: если $(a < b)$ && $(b < c)$, то $(a < c)$
- Нарушение транзитивности: «камень, ножницы, бумага»
- Асимметричность: если $(a < b)$, то $!(b < a)$
  - Следствие транзитивности и иррефлексивности
  - Действительно: если одновременно $(a < b)$ && $(b < a)$, то по транзитивности должно быть $(a < a)$
- Элементы a и b несравнимы, когда $!(a < b)$ && $!(b < a)$
- Если несравнимость транзитивна, то получим эквивалентность $a \sim b$
  - Рефлексивность и симметричность по построению

Strict Weak Ordering

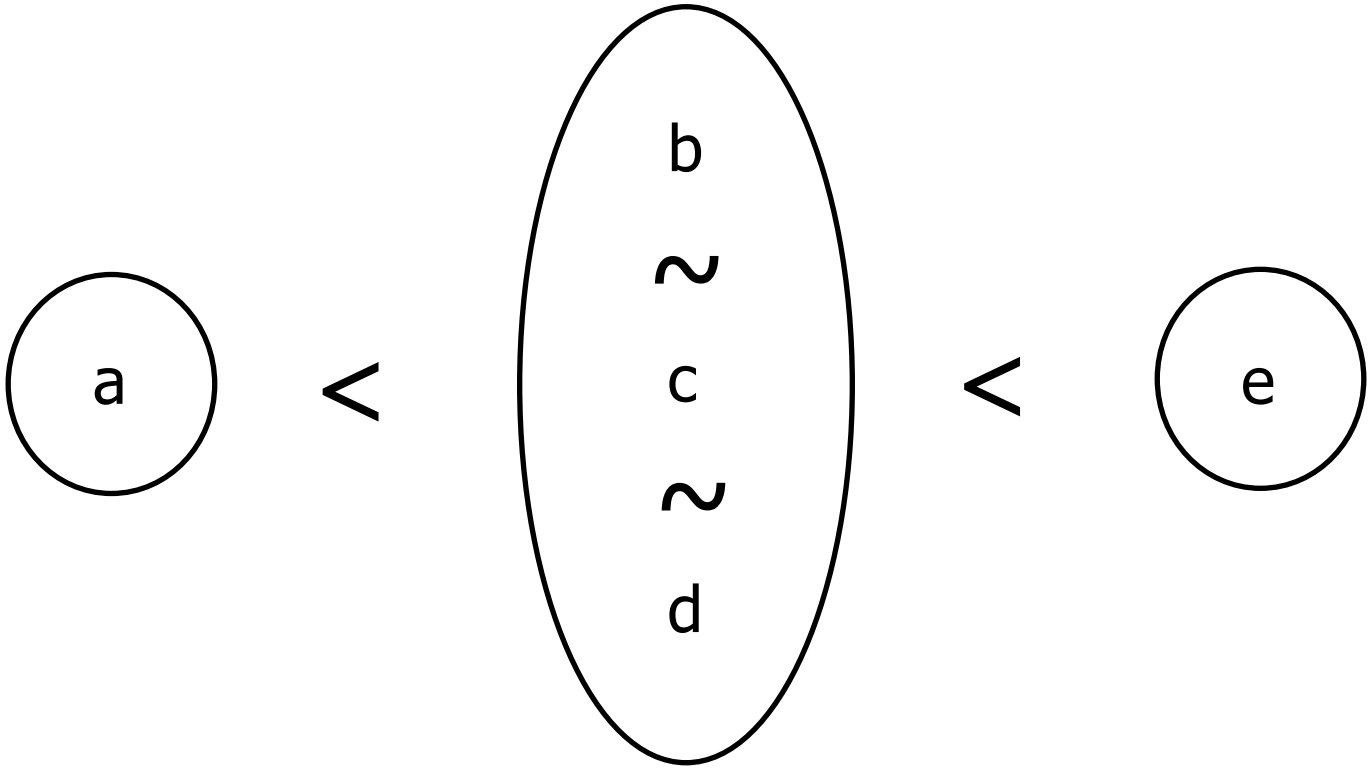
ITIVITI

- Зачем нужна транзитивность несравнимости?
- По транзитивности < и транзитивности ~ получаем:
- Если *a* и *b* несравнимы, тогда для всех $c \neq a$, $c \neq b$ если $c < a$, то и $c < b$:
  - *b* не может быть < *c*, т.к. иначе $b < c < a$
  - *b* не может быть ~ *c*, т.к. иначе $a \sim b \sim c$
- ... если $a < c$, то и $b < c$
- ... если $c \sim a$, то и $c \sim b$
- Т.о. strict weak ordering определяет *strict total ordering* на классах эквивалентности ~

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Strict Weak Ordering: классы эквивалентности



Strict Weak Ordering

```
1. class Operation {
2.     int start_time, end_time;
3.     bool operator==(const Operation& o) const
4.     { return start_time == o.start_time &&
5.              end_time == o.end_time; }
6.     ...
7.     bool started_before(const Operation& o) const
8.     { return start_time < o.start_time; }
9.     bool happened_before(const Operation& o) const
10.    { return end_time < o.start_time; }
11.};
```

- started_before
  - Транзитивно? Несравнимость транзитивна?
- happened_before
  - Транзитивно? Несравнимость транзитивна?

Strict Weak Ordering

- Без фокусов

```
1. bool operator< (const Operation& o) const
2. { return started_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(10, 15));
7. mset.insert(Operation(10, 15));
8. mset.insert(Operation(10, 15));
9.
10.cout << mset.size() << " ";
11.// ищем несравнимый элемент
12.cout << mset_contains(Operation(10, 15)) << endl;
```

Strict Weak Ordering

ITIVITI

- Без фокусов

```
1. bool operator< (const Operation& o) const
2. { return started_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(10, 15));
7. mset.insert(Operation(10, 15));
8. mset.insert(Operation(10, 15));
9.
10.cout << mset.size() << " ";                          // 3
11.// ищем несравнимый элемент
12.cout << mset_contains(Operation(10, 15)) << endl;    // true

13.// [10,15] [10,15] [10,15]
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №1: нарушение асимметричности

```
1. bool operator< (const Operation& o) const
2. { return started_before(o); }
3. ...
4. multiset<Operation, std::less_equal<Operation>> mset;
5.
6. mset.insert(Operation(10, 15));
7. mset.insert(Operation(10, 15));
8. mset.insert(Operation(10, 15));
9.
10.cout << mset.size() << " ";
11.// ищем несравнимый элемент
12.cout << mset_contains(Operation(10, 15)) << endl;
```

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №1: нарушение асимметричности

```
1. bool operator< (const Operation& o) const
2. { return started_before(o); }
3. ...
4. multiset<Operation, std::less_equal<Operation>> mset;
5.
6. mset.insert(Operation(10, 15));
7. mset.insert(Operation(10, 15));
8. mset.insert(Operation(10, 15));
9.
10.cout << mset.size() << " ";                          // 3
11.// ищем несравнимый элемент
12.cout << mset_contains(Operation(10, 15)) << endl;    // false

13.// [10,15] [10,15] [10,15]
```

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(14, 21));
7. // ищем несравнимый элемент
8. cout << mset_contains(Operation(17, 18)) << " ";
```

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(14, 21));
7. // ищем несравнимый элемент
8. cout << mset_contains(Operation(17, 18)) << " ";    // true
```

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(14, 21));
7. // ищем несравнимый элемент
8. cout << mset_contains(Operation(17, 18)) << " ";    // true
9. // добавим элементы меньше и больше искомого (17, 18)
10.mset.insert(Operation(10, 15));
11.mset.insert(Operation(20, 25));
12.// элемент эквивалентный (17, 18) еще в контейнере?
13.cout << mset_contains(Operation(17,18)) << endl;
```

Strict Weak Ordering: Фокусы с исчезновением элементов

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. multiset<Operation> mset;
5.
6. mset.insert(Operation(14, 21));
7. // ищем несравнимый элемент
8. cout << mset_contains(Operation(17, 18)) << " ";    // true
9. // добавим элементы меньше и больше искомого (17, 18)
10.mset.insert(Operation(10, 15));
11.mset.insert(Operation(20, 25));
12.// элемент эквивалентный (17, 18) еще в контейнере?
13.cout << mset_contains(Operation(17,18)) << endl;    // false

14.// [14,21] [10,15] [20,25]
```

Strict Weak Ordering: Чудеса сортировки

ITIVITI

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. vector<Operation> vec;
5. vec.push_back(Operation(20, 25));
6. vec.push_back(Operation(14, 21));
7. vec.push_back(Operation(10, 15));
8.
9. stable_sort(vec.begin(), vec.end());
10.// как отсортированы элементы?
11.
12.vec.erase(++vec.begin());
13.stable_sort(vec.begin(), vec.end());
14.// как отсортированы элементы?
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Strict Weak Ordering: Чудеса сортировки

ITIVITI

- Фокус №2: нарушение транзитивности несравнимости

```
1. bool operator< (const Operation& o) const
2. { return happened_before(o); }
3. ...
4. vector<Operation> vec;
5. vec.push_back(Operation(20, 25));
6. vec.push_back(Operation(14, 21));
7. vec.push_back(Operation(10, 15));
8.
9. stable_sort(vec.begin(), vec.end());
10.// [20,25] [14,21] [10,15]
11.
12.vec.erase(++vec.begin());
13.stable_sort(vec.begin(), vec.end());
14.// [10,15] [20,25]
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Интерфейсы ключа и элемента

ITIVITI

LET IT TRADE

```
1. class Key {
2. public:
3.     ...
4.     // Упорядоченное множество (strict total order)
5.     // Не совсем total, т.к. a < a is never true (трихотомия)
6.     friend bool operator<  (const Key& k1, const Key& k2);
7.     // Equality, not equivalence
8.     friend bool operator== (const Key& k1, const Key& k2);
9. };
10.
11. class Item {
12. public:
13.     Key key() const;
14.     bool null() const;
15.     void show(ostream& out = cout) const;
16.     ...
17. };
```

Равенство vs Эквивалентность

ITIVITI

- Case insensitive set of strings

```
1. friend bool operator<  (const Key& k1, const Key& k2) {
2.    if (k1.m_key.size() < k2.m_key.size()) return true;
3.    ...
```

- "ab" < "acd", тогда lower_bound("ac") == "acd"
- "ab" < "ef" < "acd", тогда lower_bound("ac") == "ef"

```
1. friend bool operator<  (const Key& k1, const Key& k2) {
2.
3.   return lexicographical_compare(k1.m_key.begin(),
4.                                   k1.m_key.end(),
5.                                   k2.m_key.begin(),
6.                                   k2.m_key.end(),
7.   [](char c1, char c2) { return tolower(c1)<tolower(c2); });
8.
9. }
```

LET IT TRADE

Равенство vs Эквивалентность

- Вариант 1 "Equality"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return k1.m_key == k2.m_key;
3. }
```

```
1. Key k1("Hello"), k2("hello");
2. cout << "k1 equals to k2: " << (k1 == k2) << endl; // false
3.
4. set<Key> sset;
5. sset.insert(k1);
6. if (k2 != k1) { // добавим другой (неравный) ключ
7.    cout << sset.insert(k2).second << " ";
8.    cout << sset.size() << " ";
9.    cout << sset_find(k2) << " "; // find(begin(), end(), k)
10.   cout << sset_contains(k2) << endl; // sset.find(k)
11.}
```

УНИВЕРСИТЕТ ИТМО

Равенство vs Эквивалентность

ITIVITI

- Вариант 1 "Equality"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return k1.m_key == k2.m_key;
3. }
```

```
1. Key k1("Hello"), k2("hello");
2. cout << "k1 equals to k2: " << (k1 == k2) << endl; // false
3.
4. set<Key> sset;
5. sset.insert(k1);
6. if (k2 != k1) {
7.    cout << sset.insert(k2).second << " ";          // false!
8.    cout << sset.size() << " ";                     // 1
9.    cout << sset_find(k2) << " ";                   // false!
10.   cout << sset_contains(k2) << endl;              // true
11.}
```

LET IT TRADE

Равенство vs Эквивалентность

- Вариант 2 "Equivalence"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return !(k1 < k2) && !(k2 < k1);
3. }
```

```
1. Key k1("Hello"), k2("hello");
2. cout << "k1 equals to k2: " << (k1 == k2) << endl; // true
3.
4. set<Key> sset;
5. sset.insert(k1);
6.
7. cout << sset.insert(k2).second << " ";              // false
8. cout << sset.size() << " ";                         // 1
9. cout << sset_find(k2) << " ";                       // true
10.cout << sset_contains(k2) << endl;                  // true
```

- Победа?

Равенство vs Эквивалентность

- Вариант 2 "Equivalence"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return !(k1 < k2) && !(k2 < k1);
3. }
```

```
1. Key k1("Hello"), k2("hello");
2. cout << "k1 equals to k2: " << (k1 == k2) << endl; // true
3.
4. set<Key> sset;
5. sset.insert(k1); sset.insert(k2);
6.
7. cout << *sset.find(k1) << " ";
8. cout << *sset.find(k2) << " ";
9. cout << (k2 == *sset.find(k2)) << " ";
10.cout << (k2[0] == (*sset.find(k2))[0]) << endl;
```

Равенство vs Эквивалентность

- Вариант 2 "Equivalence"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return !(k1 < k2) && !(k2 < k1);
3. }
```

```
1. Key k1("Hello"), k2("hello");
2. cout << "k1 equals to k2: " << (k1 == k2) << endl; // true
3.
4. set<Key> sset;
5. sset.insert(k1); sset.insert(k2);
6.
7. cout << *sset.find(k1) << " ";                    // Hello
8. cout << *sset.find(k2) << " ";                    // Hello
9. cout << (k2 == *sset.find(k2)) << " ";            // true
10.cout << (k2[0] == (*sset.find(k2))[0]) << endl;   // false!
```

Равенство vs Эквивалентность

- Вариант 2 "Equivalence"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return !(k1 < k2) && !(k2 < k1);
3. }
```

```
1. unordered_set<Key, hash<string>> uset;
2. uset.insert(k1);
3. // а есть ли только что добавленный элемент в set-е?
4. if (k1 == k2) { cout << uset_contains(k2) << " ";
5. // добавим такой же элемент еще раз
6.   cout << uset.insert(k2).second << " ";
7.   cout << uset.size() << " ";
8.   cout << (k1 == *uset.find(k2)) << " ";
9.   cout << *uset.find(k1) << " ";
10.  cout << *uset.find(k2) << " ";
11.}
```

Равенство vs Эквивалентность

ITIVITI

- Вариант 2 "Equivalence"

```
1. friend bool operator== (const Key& k1, const Key& k2) {
2.     return !(k1 < k2) && !(k2 < k1);
3. }
```

```
1. unordered_set<Key, hash<string>> uset;
2. uset.insert(k1);
3. // а есть ли только что добавленный элемент в set-е?
4. if (k1 == k2) { cout << uset_contains(k2) << " ";   // false!
5. // добавим такой же элемент еще раз
6.   cout << uset.insert(k2).second << " ";            // true!
7.   cout << uset.size() << " ";                       // 2 !!!
8.   cout << (k1 == *uset.find(k2)) << " ";            // true
9.   cout << *uset.find(k1) << " ";                    // Hello
10.  cout << *uset.find(k2) << " ";                    // hello
11.}
12.hash<string>{}(k1) == hash<string>{}(k2) ?
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Интерфейс «Таблицы символов»


```
1. // Не возвращаем указатели (handles) на элементы
2. class SymbolTable {
3. public:
4.     SymbolTable();
5.     // Число элементов (считаем лениво или агрессивно)
6.     size_t count();
7.     // Поиск элемента по ключу
8.     Item search(const Key& k);
9.     // Вставка элемента с заданным ключом; без дубликатов
10.    void insert(Item x);
11.    // Удаление элемента с ключом, равным ключу x
12.    // Выполняется поиск удаляемого элемента
13.    void remove(const Item& x);
14.    // Ноль элементов меньше минимального
15.    Item min();
16.    // N-1 элемент меньше максимального
17.    Item max();
```

Интерфейс «Таблицы символов»

```
18.    // Выбор i-го по порядку элемента, где i от 0 до N-1
19.    // Поиск элемента ранга i:
20.    // в точности i других элементов меньше искомого
21.    Item select(size_t i);
22.
23.    // Ранг элемента x: количество элементов с ключами,
24.    // меньшими ключа x
25.    size_t rank(const Item& x);
26.
27.    // Предыдущий элемент
28.    Item pred(const Key& k);
29.    // Следующий элемент
30.    Item succ(const Key& k);
31.    // Вывод в отсортированном виде
32.    void show(ostream& out = cout);
33.    ...
34. };
```

АТД «Таблица символов»

ITIVITI

- Альтернативные названия – ассоциативный массив (associative array, map), словарь (dictionary)
- Реализация на (не)отсортированном списке:
  - хорошо вставлять / удалять, плохо искать
  - удобна для очень небольших таблиц
- Реализация на отсортированном массиве:
  - квадратичное время построения
  - удобна для статических таблиц (запрещены операции вставки): заполнили и отсортировали для бинарного поиска
- Что делать, если операции поиска и вставки / удаления идут вперемешку и размер таблицы большой?

LET IT TRADE

ITIVITI

LET IT TRADE

Структура данных для работы с упорядоченными множествами

- Сочетает в себе гибкость вставки / удаления в связный список с эффективностью поиска в упорядоченном массиве
- «Динамическая версия» упорядоченного массива
- Такой же набор поддерживаемых операций

Упорядоченные массивы: поддерживаемые операции

ITIVITI

Операция	**Оценка сложности**
Поиск по ключу	$O(\log N)$
Порядковая статистика (i-ый по величине ключ)	$O(1)$
Наименьший / наибольший ключ	$O(1)$
Предыдущий / следующий ключ	$O(1)$
Ранг (количество ключей, меньших либо равных заданному)	$O(\log N)$
Вывод в отсортированном виде	$O(N)$
Вставка	$O(N)$
Удаление	$O(N)$

LET IT TRADE

Сбалансированные деревья поиска

ITIVITI

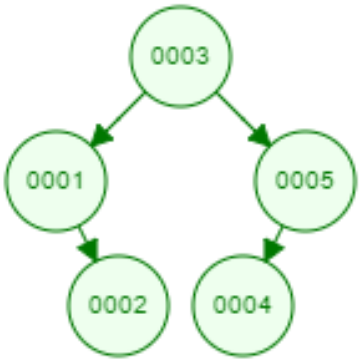
Операция	**Оценка сложности**
Поиск по ключу	$O(\log N)$
Порядковая статистика (i-ый по величине ключ)	$O(\log N)$
Наименьший / наибольший ключ	$O(\log N)$
Предыдущий / следующий ключ	$O(\log N)$
Ранг (количество ключей, меньших либо равных заданному)	$O(\log N)$
Вывод в отсортированном виде	$O(N)$
Вставка	$O(\log N)$
Удаление	$O(\log N)$

- ... сравнить с пирамидой

LET IT TRADE

Бинарные деревья поиска

- Бинарное дерево, в котором
  - каждый узел содержит ключ (и связанное с ним значение)
  - ключи всех узлов левого поддерева меньше (либо равны) ключа родительского узла
  - ключи всех узлов правого поддерева больше (либо равны) ключа родительского узла
- Структура БДП аналогична разбиению при быстрой сортировке



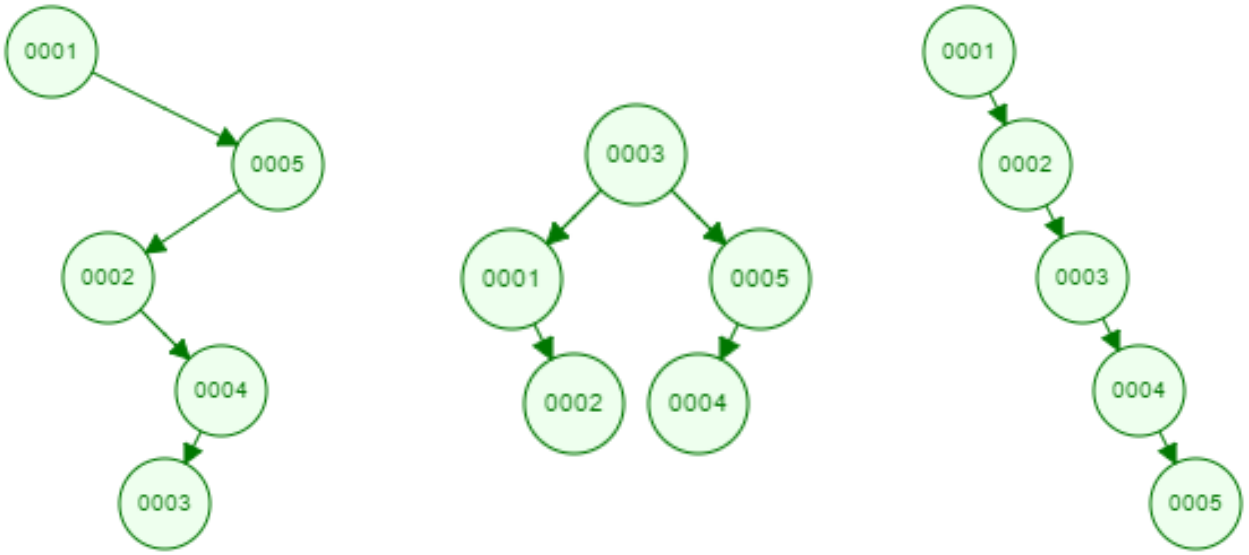
https://www.cs.usfca.edu/~galles/visualization/BST.html

Бинарные деревья поиска

```
1. class SymbolTable {
2. ...
3. private:
4. struct Node {
5.     Item   item;
6.     Node*  left;
7.     Node*  right;
8.     size_t count; // кол-во узлов в поддереве с этим корнем
9.     Node(Item x)  {
10.      item = x; count = 1;
11.      left = NULL; right = NULL;} // dummy nodes?
12. };
13.
14. Node*  m_root;
15. Item   nullItem;
16. ...
17. };
```

Высота БДП

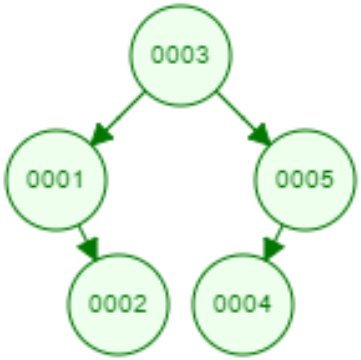
- Для заданного множества ключей можно построить различные БДП
- Высота БДП лежит в пределах от $\log_2 N$ (наилучший случай) до N (наихудший случай)



https://www.cs.usfca.edu/~galles/visualization/BST.html

Поиск в БДП

- Если дерево пусто, то имеем промах при поиске
- Сравниваем ключ корня с искомым
  - Если ключи совпали, то элемент найден
  - Если нет, то сразу понятно, в каком поддереве искать ключ
- На каждом шаге гарантируется, что в исключенных из рассмотрения поддеревьях нет искомого ключа
- Действуем аналогично бинарному поиску



https://www.cs.usfca.edu/~galles/visualization/BST.html

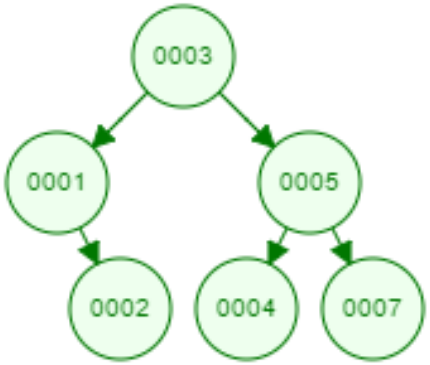
Поиск в БДП

```
1. public:
2. Item search(const Key& k) {
3.     return do_search(m_root, k);
4. }
5. ...
6. private:
7. Item do_search(Node* node, const Key& k) {
8.     if (node == NULL) return nullItem;
9.     Key t = node->item.key();
10.    if (k == t) return node->item;
11.    if (k < t)
12.        return do_search(node->left, k);
13.    else
14.        return do_search(node->right, k);
15.}
```

- Что если искомого ключа нет в БДП?

Вставка в БДП

- Поиск отсутствующего ключа заканчивается на пустом поддереве
- Заменим пустое поддерево вставляемым узлом
- Если вставляемый ключ уже существует в дереве, то ничего не делаем



https://www.cs.usfca.edu/~galles/visualization/BST.html

Вставка в БДП

```
1. void do_insert(Node* & node, Item x) { // NB Node* &
2.     if (node == NULL) { node = new Node(x); return; }
3.     Key t = node->item.key();
4.     if (x.key() == t) return;
5.     if (x.key() < t)
6.         do_insert(node->left, x);
7.     else
8.         do_insert(node->right, x);
9.     // node->count++; ???
10.     node->count = 1+do_count(node->left)+\
11.     do_count(node->right); // или node->left->count ???
12.}
```

- Node* передаем по ссылке
- Полезно сравнить с do_search()

Поиск и вставка в БДП

- Какое наихудшее время выполнения операций поиска / вставки в БДП, содержащее N ключей и имеющее высоту *height*?
  - $\Theta(1)$
  - $\Theta(\log N)$
  - $\Theta(N)$
  - $\Theta(height)$

Поиск и вставка в БДП

- Какое наихудшее время выполнения операций поиска / вставки в БДП, содержащее N ключей и имеющее высоту $height$?
  - $\Theta(1)$
  - $\Theta(\log N)$
  - $\Theta(N)$
  - $\Theta(height)$
- Т.о. недостаточно знать общее число N ключей, для оценки времени поиска / вставки
- Еще необходимо знать структуру дерева

Обход дерева

- Прямой обход (preorder): узел, затем левое и правое поддеревья
- Поперечный обход (inorder): левое поддерево, узел, правое поддерево
- Обратный обход (postorder): сначала левое и правое поддеревья, а затем узел
- Обход для вывода БДП в отсортированном виде?

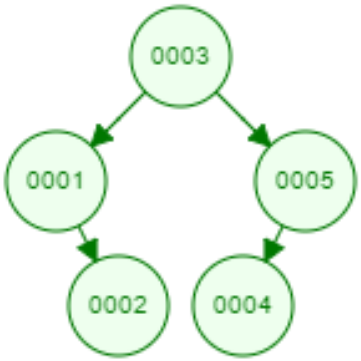
Обход дерева

- Прямой обход (preorder): узел, затем левое и правое поддеревья
- Поперечный обход (inorder): левое поддерево, узел, правое поддерево
- Обратный обход (postorder): сначала левое и правое поддеревья, а затем узел
- Обход для вывода БДП в отсортированном виде?

```
1. void do_show(Node* node, ostream& out) {
2.     if (node == NULL) return;
3.     do_show(node->left, out);
4.     node->item.show(out);
5.     do_show(node->right, out);
6. }
```

Поиск наименьшего ключа

- Если левое поддерево пусто, то наименьший ключ находится в корне
- Если левое поддерево не пусто, то наименьший элемент находится в нем – перейдем туда
- Эквивалентен поиску -∞



https://www.cs.usfca.edu/~galles/visualization/BST.html

Поиск наименьшего ключа

ITIVITI

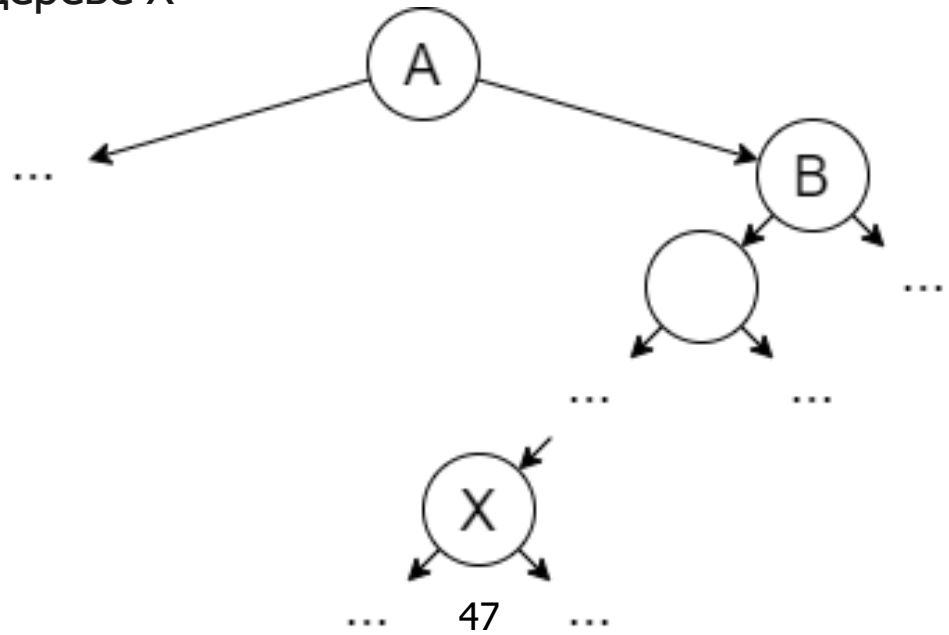
LET IT TRADE

```
1. public:
2. Item min() {
3.     return do_min(m_root);
4. }
5. ...
6. private:
7. Item do_min(Node* node) {
8.     if (node->left == NULL) return node->item;
9.     return do_min(node->left);
10.}
```

- Аналогично осуществляется поиск наибольшего ключа

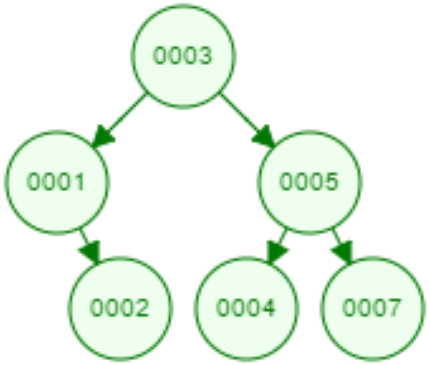
Поиск предыдущего элемента

- … в отсортированной последовательности (при поперечном обходе)
- Пусть A – «наименьший» предок X, у которого X лежит в правом поддереве: A < X < B, где B – правый ребенок A
  - Элементы, которые могут находиться между A и X, – в левом поддереве X



Поиск предыдущего элемента

- Если левое поддерево не пусто, то возвращаем max() оттуда
- В противном случае подымаемся наверх до первого поворота налево при подъеме
- Если не встретим поворота налево, то значит предыдущего элемента нет (стартовали из наименьшего элемента)
- Структура БДП позволяет найти предыдущий элемент не выполняя сравнений!



https://www.cs.usfca.edu/~galles/visualization/BST.html

Поиск предыдущего элемента

```
1. Item do_pred(Node* node, const Key& k) {
2.     if (node == NULL) return nullItem;
3.     Key t = node->item.key();
4.     if (k == t) {
5.         if (node->left != NULL) return do_max(node->left);
6.         else return nullItem;
7.     }
8.     if (k < t)
9.         return do_pred(node->left, k);
10.    else {
11.        Item x = do_pred(node->right, k);
12.        if (x.null()) return node->item;
13.        else return x;
14.    }
15.}
```

- Какой элемент вернем, если ключа k нет в БДП?

Выборка в БДП

- Нахождение *i*-го по порядку ключа (*i* от 0 до *N*-1)
- Поиск элемента ранга *i*: в точности *i* других элементов меньше искомого
- Аналогично выборке на основе «разбиения» при быстрой сортировке
  - Проверяем число элементов в левом поддереве
  - Для этого каждый узел поддерживает `count` – количество узлов в поддереве с этим корнем (информация о самой структуре дерева)
  - `count` необходимо обновлять при изменении дерева: вставка, удаление, повороты
  - ```
    node->count = 1 + do_count(node->left) +
    do_count(node->right);
    ```
    ```

Выборка в БДП

```
1. Item do_select(Node* node, size_t i) {
2.     if (node == NULL) return nullItem;
3.
4.     size_t c = 0;
5.     if (node->left != NULL) c = node->left->count;
6.
7.     if (c == i) return node->item;
8.     if (c > i)
9.         return do_select(node->left, i);
10.     else
11.        return do_select(node->right, i-c-1);
12.}
```

- Какой элемент вернем, если будем искать элемент ранга *N* и выше?

Определение ранга ключа

- Операция, обратная выборке: задан ключ, требуется найти его ранг

```
1. size_t do_rank(Node* node, const Key& k) {
2.   if (node == NULL) return 0;
3.   Key t = node->item.key();
4.   if (k == t) return do_count(node->left);
5.   if (k < t)
6.      return do_rank(node->left, k);
7.   else
8.      return 1+do_count(node->left)+do_rank(node->right, k);
9.}
```

- Какой ранг вернем, если ключа k нет в БДП?
  - Ранг элемента <6,x>? <8,y>?

Удаление элемента

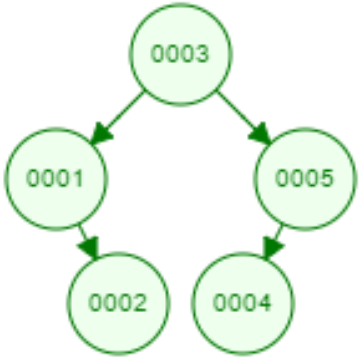
ITIVITI

- Удаление по указателю (handle)
  - Аналогично удалению в односвязном списке
  - Необходимо поменять указатель родительского узла **на** удаляемый узел
  - В узел добавляем указатель на родителя
  - Ленивое удаление
- Удаление через поиск по ключу
  - Находим удаляемый элемент с заданным ключом (и возможно с заданным handle)
  - В процессе поиска получаем указатель родительского узла на удаляемый

LET IT TRADE

Удаление элемента

- Вариант 1: удаляемый узел не имеет потомков…
- Вариант 2: удаляемый узел имеет одного потомка…
- Вариант 3: удаляемый узел имеет двух потомков…
  - Заменим на следующий элемент
  - У следующего элемента нет левого потомка – переходим к варианту 2
  - Предыдущий вместо следующего? случайный выбор?



https://www.cs.usfca.edu/~galles/visualization/BST.html

Удаление элемента

ITIVITI

```
1. void do_remove(Node* & node, const Key& k) { // NB Node* &
2.     if (node == NULL) return;
3.     Key t = node->item.key();
4.     if (k == t) {
5.       // вариант 1: удаляемый узел не имеет потомков
6.       if(node->left == NULL && node->right == NULL) {
7.           delete node;
8.           node = NULL;
9.           return; // обновить node->count ???
10.      }
11.      // вариант 2: нет левого поддерева
12.      else if(node->left == NULL) {
13.          Node* tmp = node;
14.          node = node->right;
15.          delete tmp;
16.          return; // обновить node->count ???
17.      }
18.      ...
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Удаление элемента

```
19.       // вариант 2: нет правого поддерева
20.       else if(node->right == NULL) { ...
21.       }
22.       // вариант 3: есть левое и правое поддеревья
23.       else {
24.          Node* tmp = node->right; // идем в правое поддерево
25.          while (tmp->left != NULL)
26.              tmp = tmp->left;     // и ищем там минимум
27.          node->item = tmp->item;  // теперь надо удалить tmp
28.          do_remove(node->right, tmp->item.key()); // почему ?
29.       }
30.    }
31.    if (k < t) do_remove(node->left, k);
32.    // else ???
33.    if (t < k) do_remove(node->right, k);
34.    node->count = 1+do_count(node->left)+\
35.    do_count(node->right);
36.}
```

ITIVITI

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО