БК №536
КУРС «ПРОГРАММИРОВАНИЕ В ЗАДАЧАХ РАДИОЛОКАЦИИ»
РАЗДЕЛ IV «СТРУКТУРЫ ДАННЫХ»

ЛЕКЦИЯ №7

«ОБЗОР СТРУКТУР ДАННЫХ»

Tuple

Кортеж — упорядоченный набор фиксированной длины

- гетерогенный;
- аналог структур с безымянными полями (доступ по индексу).

$$T = S_1 \times S_2 \times \cdots \times S_n = \{(x_1, x_2, \ldots, x_n) | \; x_i \; \in S_i, i \; \in [1, n]\}$$

Явно или неявно представлен в большинстве ЯП:

- константные кортежи в Python используются для возврата нескольких значений из функции;
- запись в реляционной таблице — кортеж;
- std::initializer_list в C++ по своей сути кортеж, также существует std::tuple

Tuple

Кортеж (std::tuple) в C++:

- определен в tuple;
- compile-time;
- в рантайме доступ к элементам за O(1);
- изменяемый;
- повышает гибкость и читаемость.

Tuple

До C++ 11

```
enum Error {
    Ok,
    Warning,
    Critical
};

Error checkSystem(std::string& description){
    // some code
    code = getErrorCode();
    description = getDescription();
    return code;
}
// ...
std::string descr;
Error code = checkSystem(descr);
```

```
enum Error {
    Ok,
    Warning,
    Critical
};
struct CheckSystemOutput {
    Error errorCode;
    std::string description;
};
CheckSystemOutput checkSystem(){
    // some code
    CheckSystemOutput result;
    result.errorCode = getErrorCode();
    result.description = getDescription();
    return result;
}
// ...
CheckSystemOutput output = checkSystem();
```

Tuple

После C++ 11

```
enum Error {
    Ok,
    Warning,
    Critical
};

std::tuple<Error, std::string> checkSystem(){
    // some code
    return std::make_tuple(getErrorCode(), getDescription());
}
// ...
auto result = checkSystem();  // get<0>(result) - code, get<1>(result) - descr

Error code; std::string descr;
std::tie(code, descr) = checkSystem;
```

Tuple

После C++ 14

```
enum Error {
    Ok,
    Warning,
    Critical
};

auto checkSystem(){
    // some code
    return std::make_tuple(getErrorCode(), getDescription());
}
// ...
auto result = checkSystem();  // get<0>(result) - code, get<1>(result) - descr

Error code; std::string descr;
std::tie(code, descr) = checkSystem;
```

Tuple

После C++ 14

```
enum Error {
    Ok,
    Warning,
    Critical
};

auto checkSystem(){
    // some code
    return std::make_tuple(getErrorCode(), getDescription());
}
// ...
auto [code, descr] = checkSystem();
```

Priority queue

Очередь с приоритетами — абстрактный тип данных, элементы которой помимо значения имеют приоритет.

Операции:

- добавить элемент;
- получить приоритетную задачу (найти максимум/минимум);
- удалить приоритетную задачу (удалить максимум/минимум);
- уменьшение/увеличение приоритета;
- слияние очередей.

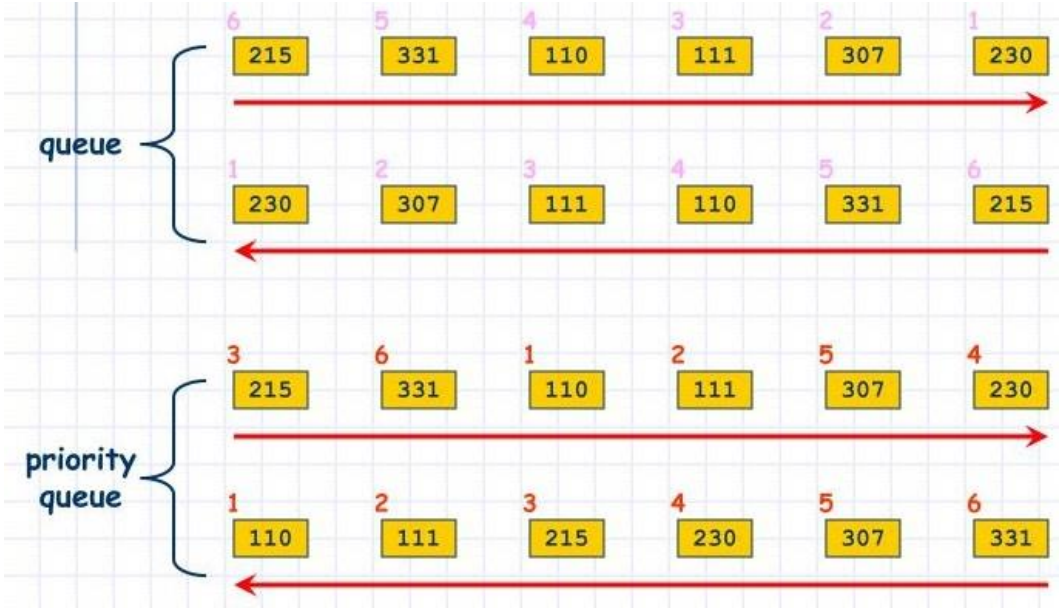
Использование:

- системы массового обслуживания.

Priority queue

Предпочтительные способы реализации: кучи

В C++ представлена классом std::priority_queue (заголовочный файл queue).



Heap

Куча — дерево или набор деревьев, удовлетворяющий свойству:

$\forall$ A, B — узлы дерева : B — потомок A =>

- B.key $\geq A.key\ (min - heap)$
- B.key $\leq A.key\ (max - heap)$

Операции:

- максимум/минимум;
- удалить максимум/минимум;
- обновить ключ;
- добавить элемент;
- слияние.

Heap

Применение:

- очередь с приоритетами;
- алгоритмы на графах;
- пирамидальная сортировка (HeapSort).

Виды куч:

- бинарная куча;
- фибонаначчиева куча;
- 2-3 куча;
- куча Бродала;
- и еще куча куч.

Binary heap

Бинарная куча — бинарное дерево, для которого выполнены условия:

- значения ключей (приоритетов) в любой вершине не меньше (не больше), чем в любом из ее потомков;
- длина простых путей отличается не больше, чем на 1;
- заполняется слой за слоем слева направо.

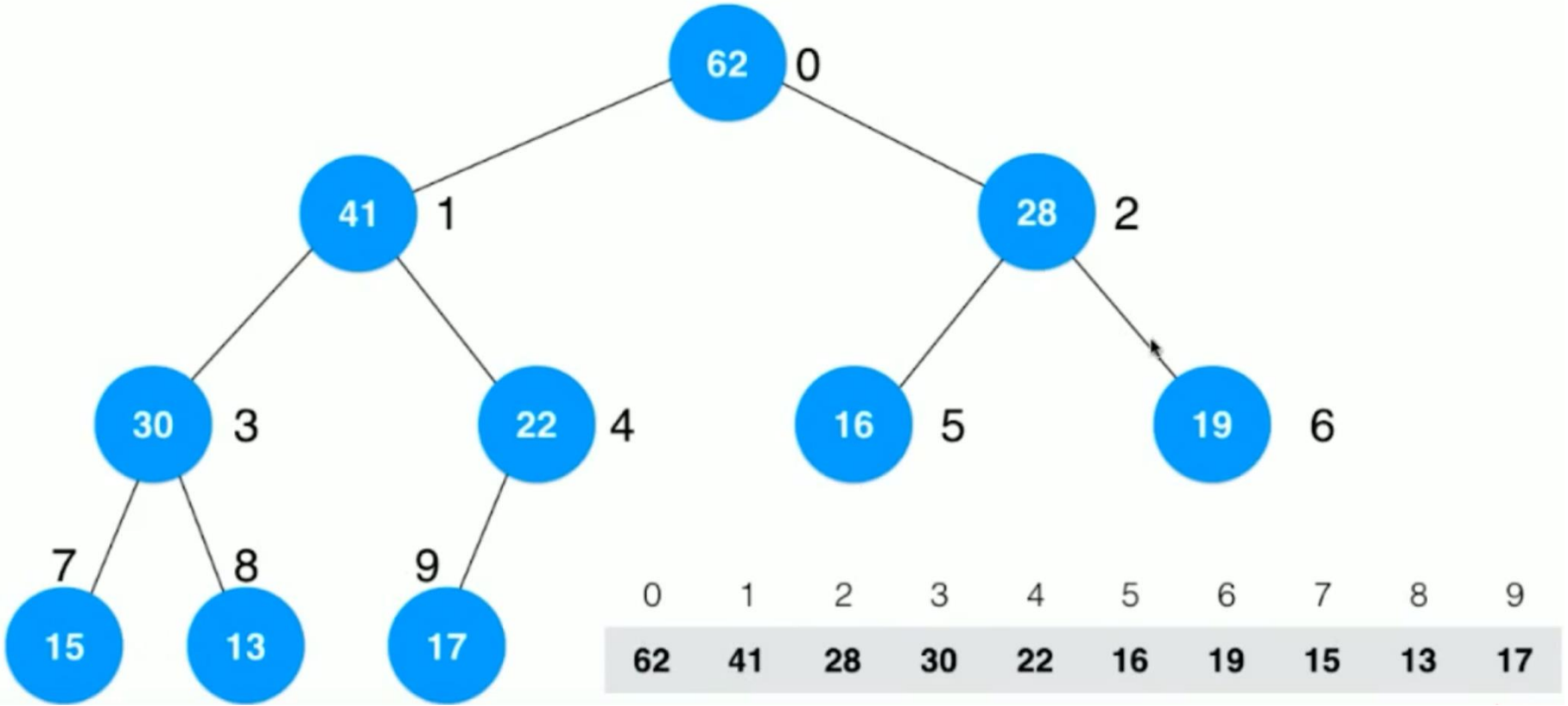
Распространена реализация через вектор.

для i-го элемента:

- левым ребенком будет элемент 2i + 1 (2i в случае нумерации вектора с 1);
- правым ребенком будет элемент 2i + 2 (или 2i + 1).

При модификациях кучи производится восстановление свойств бинарной кучи.

Binary heap



0	1	2	3	4	5	6	7	8	9
62	41	28	30	22	16	19	15	13	17

Heap

Операция	Двоичная	Биномиальная	Фибоначчиева	Спаренная[2]	Бродал
найти минимум	Θ(1)	Θ(log *n*) or Θ(1)	Θ(1)[1]	Θ(1)	Θ(1)
удалить минимум	Θ(log *n*)	Θ(log *n*)	O(log *n*)*	O(log *n*)*	O(log *n*)
добавить	Θ(log *n*)	O(log *n*)	Θ(1)	O(1)*	Θ(1)
уменьшить ключ	Θ(log *n*)	Θ(log *n*)	Θ(1)*	O(log *n*)*	Θ(1)
слияние	Θ(*n*)	O(log *n*)**	Θ(1)	O(1)*	Θ(1)

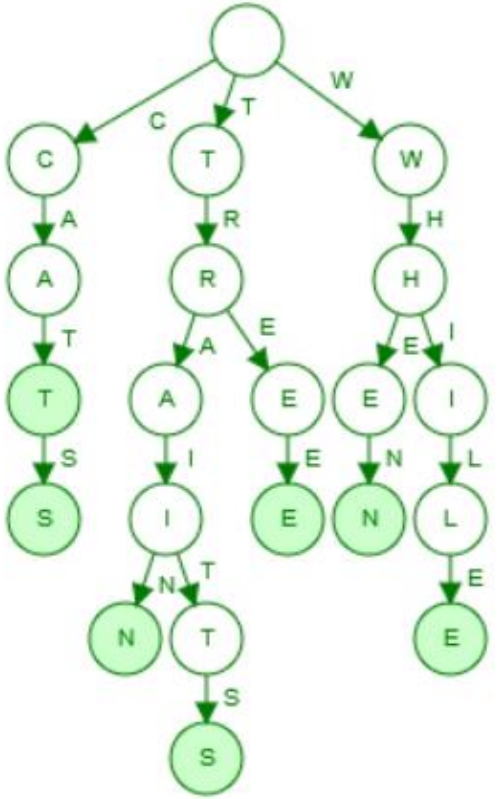
(*) Амортизационное время

(**) Где n — размер наибольшей кучи

Trie

Префиксное дерево — ассоциативный массив, в котором ключ посимвольно или побайтово распределен по связанным узлам или ребрам дерева.

- ключи чаще всего строки;
- узел состоит из части ключа, данных и метки терминальности;
- путь от корня до терминальной вершины — существующий ключ;
- сложность основных операций зависит от мощности алфавита и длины ключа и равна $O(|A||key|) = O(|key|)$(грубая)

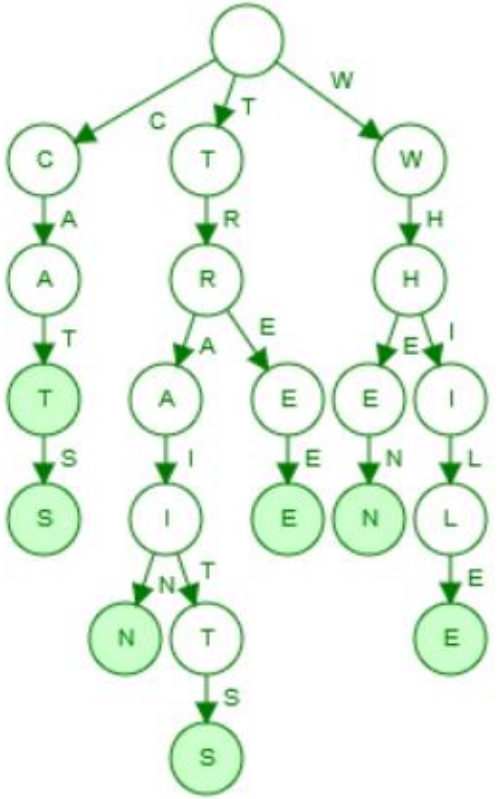


Trie

Применение:

- predictive text input;
- подсказки в IDE;
- сжатие данных без потерь (LZW);
- алгоритм поиска в тексте (Ахо-Корасик);
- алгоритмы расстановки переносов.

Модификация Modified Merkle Patricia Trie используется в Ethereum.



Radix Trie