Лекция 11. Неблокирующие алгоритмы.

Содержание

- Важные подалгоритмы. RCU. Hazard Pointers. Marked References.
- Оптимистический подход.
- Неблокирующий стек.
- Неблокирующая очередь.
- Неблокирующие списки.
- Skiplist.

Важные подалгоритмы. RCU. Hazard Pointers. Marked References.

Hazard Pointers

- Каждый поток сохраняет список опасных указателей, которые он, возможно, изменит.
- Указатели из этого списка не должны модифицироваться или освобождаться другими потоками.
- Каждый поток чтения владеет разделяемым указателем `HP` в режиме один писатель/много читателей.
- Когда поток-читатель назначает адрес в `HP`, он тем самым извещает остальные потоки: я читаю, если вы хотите изменить - изменяйте, но не изменяйте содержимое по адресу и не удаляйте его.
- Если поток хочет удалить указатель, он помещает его в список «удалить позднее». Это произойдёт, когда адрес уйдёт из всех списков потоков, поэтому требуется счётчик использования.

Marked References

MarkedReference

- Существует объект `MarkedReference`, содержащий пару: указатель и счётчик.
- Счётчик глобален для всех создаваемых объектов.
- Счётчик увеличивается каждый раз при создании объекта.
- Все операции с потенциальными `ABA` указателями должны проходить через методы класса `MarkedReference`.

Оптимистический подход.

- Поток: я хочу поработать с данными.
- Я делаю снимок текущего состояния.
- Я делаю, что хочу.
- Перед записью я проверяю, не изменилось ли текущее состояние.
- Если изменилось (я оптимист, и надеюсь, что нет), то повторяю всю работу заново.
- Проблема. ПРОБЛЕМА. ПРОБЛЕМА. АВА.

Неблокирующий стек.

Неблокирующий стек

```
class LockFreeStack {
  volatile T* top;
  T* pop() {
    do  {
      T* ret = top;
      if (ret == NULL) return NULL;
      T* next = ret->next;
    } while (!CAS(top, ret, next));
    return ret;
  }
  void push(T* obj) {
    do {
      T* next_ptr = top;
      obj->next = next;
    } while (!CAS(top, next, obj));
  };
```

Неблокирующие списки

Списки

Список — структура данных, которая реализует абстракции:

- *insertToFront* — добавление элемента в начало списка.
- *insertToBack* — добавление элемента в конец списка.
- *insertBefore* — добавление элемента перед заданным.
- *insertAfter* — добавление элемента после заданного.
- *find* — поиск элемента
- *size* — определение количества элементов

Списки: реализация

Для реализации списков требуются явное использование указателей.

```
struct linkedListNode { // Одна из реализаций
    someType *data;         //
    linkedListNode *next; // Связь со следующим
};
```

Внутренние операции создания элементов — через `malloc`, `calloc`, `new`, `MR`.

```
...
linkedListNode *item = new linkedListNode();
item->data = myData;
...
```

Списки: сложность

Сложность основных операций:

- Вставка элемента в голову списка — $O(1)$
- Вставка элемента в хвост списка — $O(N)$
- Поиск элемента — $O(N)$
- Удаление известного элемента — $O(1)$
- Вставка элемента ЗА известным — $O(1)$
- Вставка элемента ПЕРЕД известным — $O(N)$

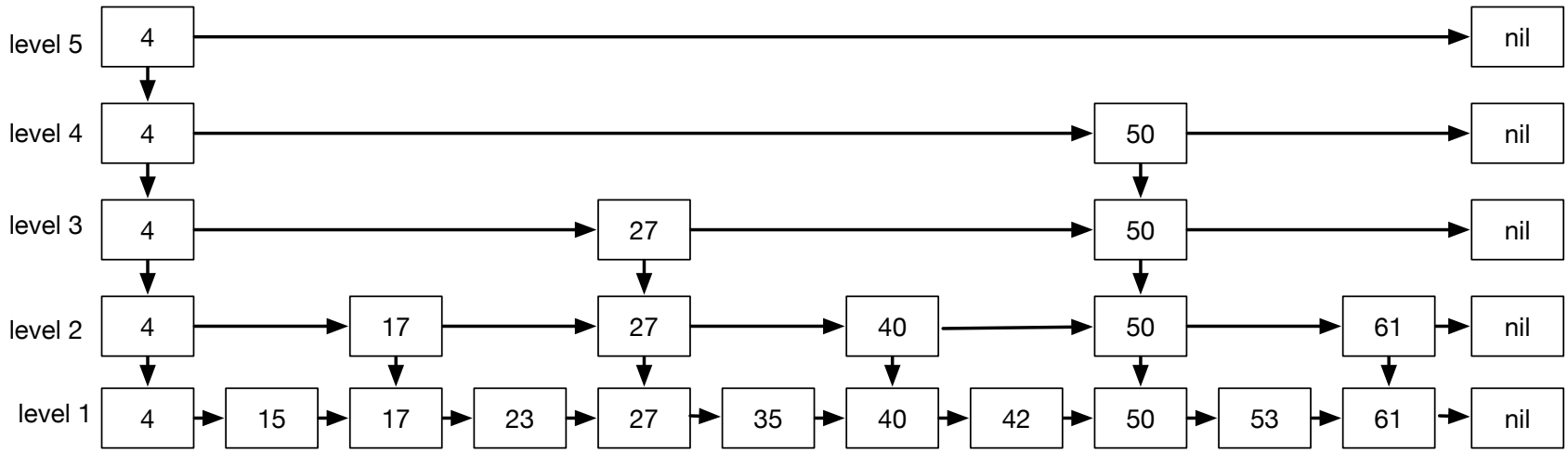
Неблокирующие списки

- Если не в начало и не в конец — то через `RCU` или `HP`.

Skiplists

Списки с пропусками

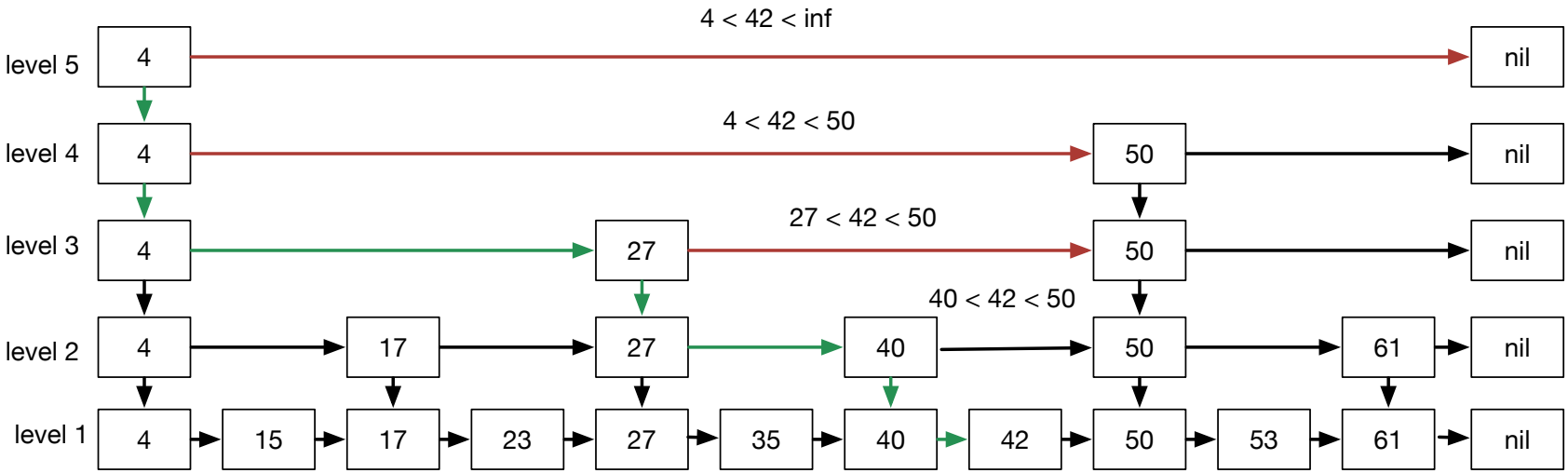
Рассмотрим следующую структуру данных:



- Она представляет из себя несколько списков, организованных в виде списков.
- Каждый следующий список примерно в два раза короче предыдущего и он пропускает примерно половину элементов предыдущего.

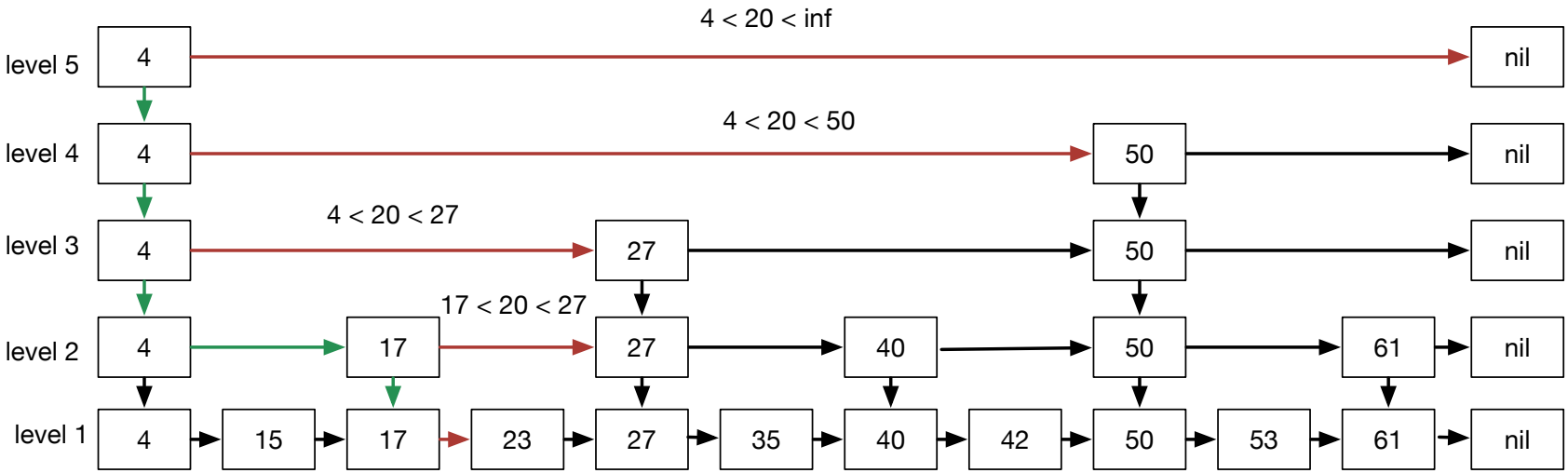
Списки с пропусками

- Поиск существующего элемента.



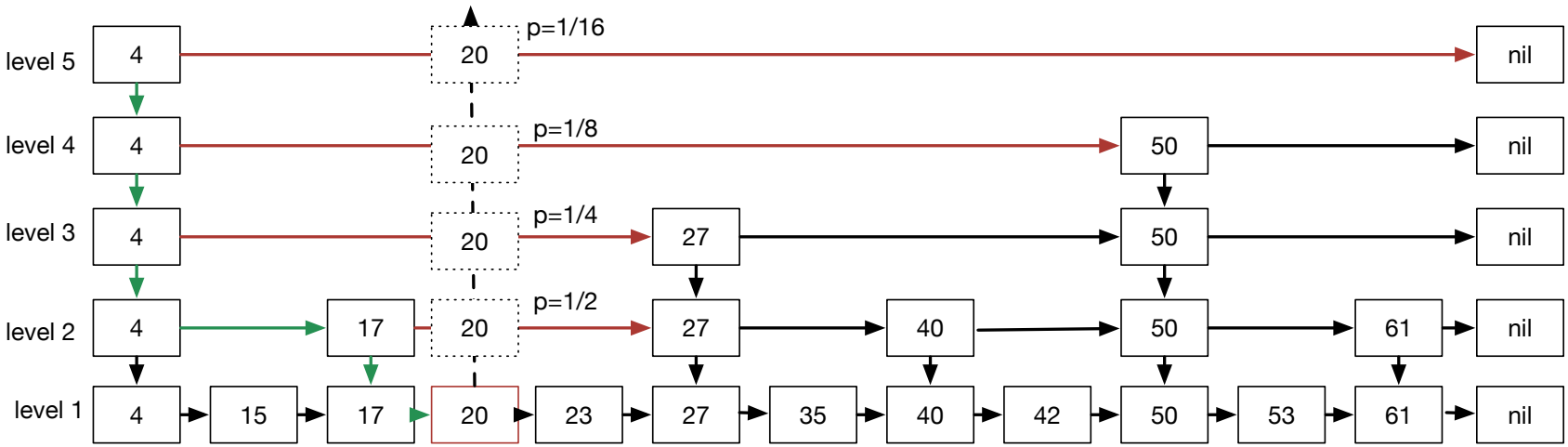
Списки с пропусками

- Поиск несуществующего элемента.



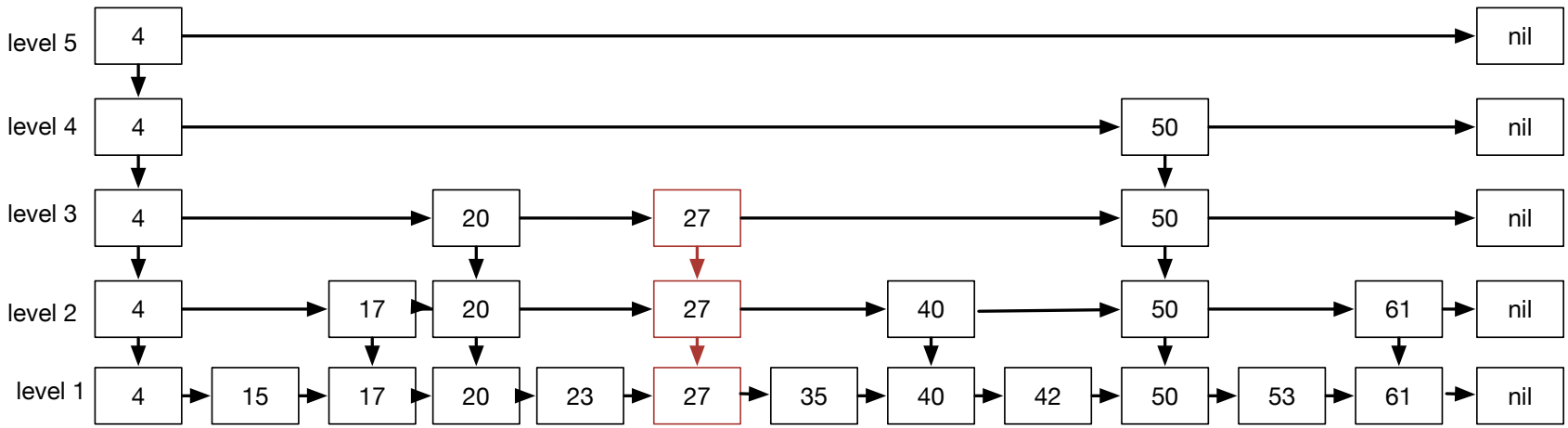
Списки с пропусками

- Вставка элемента.



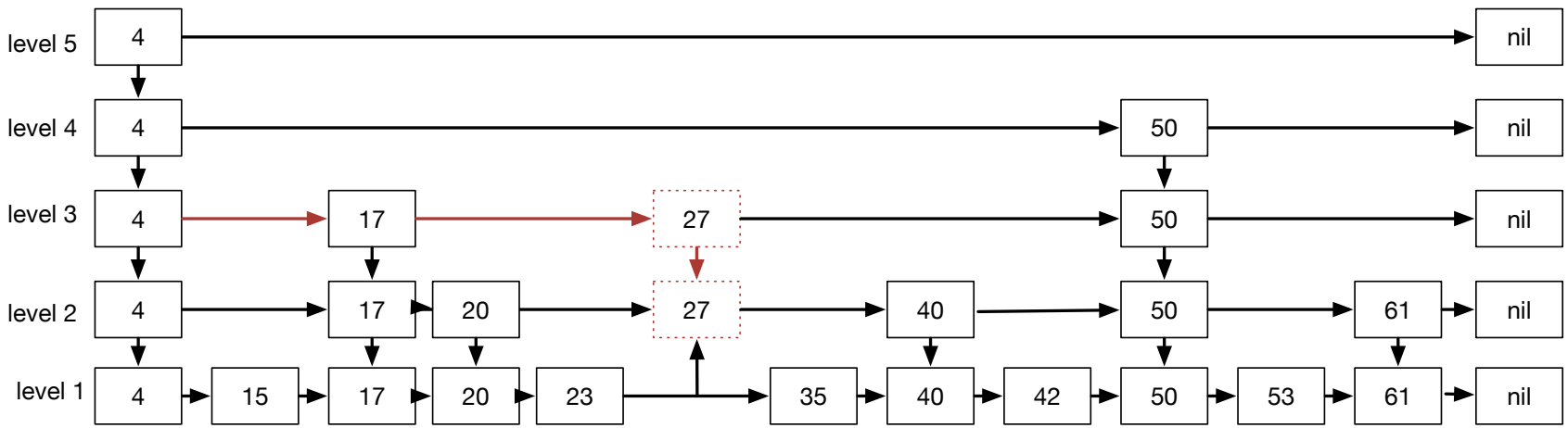
Списки с пропусками

- Удаление элемента. Поиск и пометка столбца.



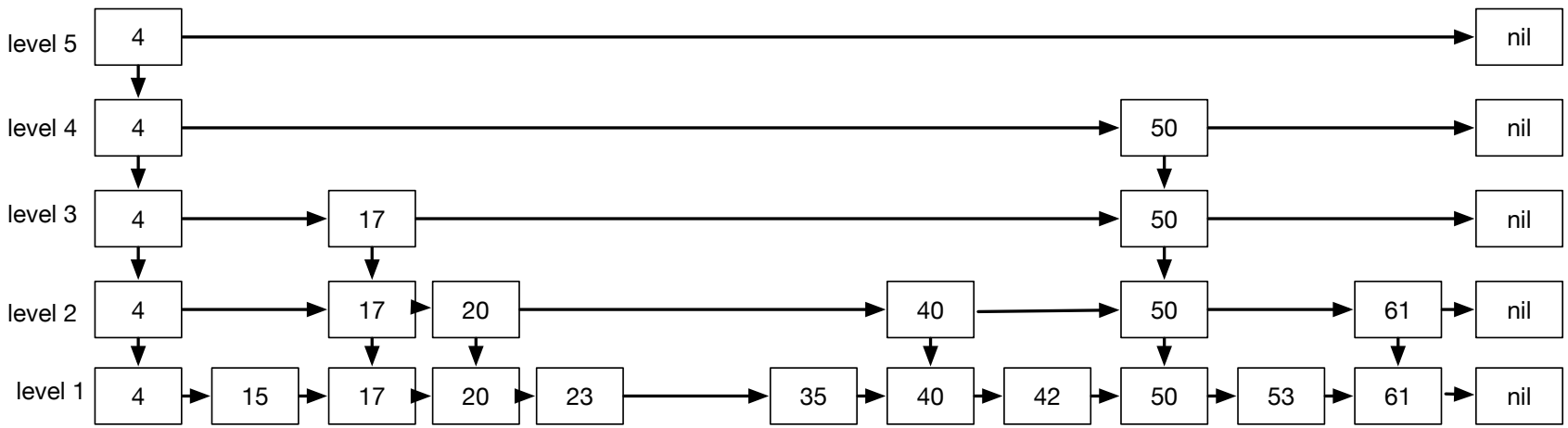
Списки с пропусками

- Удаление элемента. Удаление из строк.



Списки с пропусками

- Удаление элемента. Заключительное удаление.



Списки с пропусками

- Вставка 10^6 элементов в структуру данных.

Вставка	Array	AVL	SkipList
Случайно	130120	1230	1714
По возрастанию	110	495	543
По убыванию	264220	377	403

Списки с пропусками

Амортизационная сложность списков с пропусками:

- Вставка — $T(N) = O(\log N)$
- Поиск — $T(N) = O(\log N)$
- Удаление — $T(N) = O(\log N)$

Быстрый генератор случайных чисел

```
class FastRandom {
private:
    unsigned long long rnd;
public:
    FastRandom(unsigned long long seed) {
        rnd = seed;
    }
    unsigned long long rand() {
        rnd ^= rnd << 21;
        rnd ^= rnd >> 35;
        rnd ^= rnd << 4;
        return rnd;
    }
}
```

Спасибо за внимание.

Следующая тема — конкуренция потоков. Пулы потоков.