



细谈

Laziness & Persistence

刘家财

<http://liujiacai.net/>

提纲

- ◆ Laziness
 - ◆ 按需求**值**、delay/force/realized?/lazy-seq
 - ◆ 无**穷**序列 (infinite sequences)
 - ◆ Chunked Sequence
- ◆ Persistence
 - ◆ []: Persistent bit-partitioned vector trie
 - ◆ {}: Array-mapped hash trie

Laziness

Laziness—按需求值

- Call by name vs Call by value

```
(defn do-some-magic [cheap expensive]  
  (if (some-condition)  
      (force expensive)  
      cheap))
```

```
(do-some-magic  
  "tom"  
  (delay (Thread/sleep 10) "jerry"))
```

Laziness—无穷序列

```
(take 5 (range))  
;;-> (0 1 2 3 4)
```

```
(def powers-of-two (iterate (partial * 2) 1))
```

```
(take 10 powers-of-two)  
;;-> (1 2 4 8 16 32 64 128 256 512)
```

```
(nth powers-of-two 11)  
;;-> 1024
```

Laziness—Chunked Seq

```
(def gimme #(do (print \.) %))  
(take 1 (map gimme (range 32)))  
  
;; -> (0).....
```

只取第一个元素，为啥打印了32个.？

每次求32个元素的值

是 cache 与 laziness 折中的一种方法

Laziness—Chunked Seq

```
(def gimme #(do (print \.) %))  
(def foo (map gimme (range 64)))
```

```
(take 4 foo)  
;; -> (0 1 2 3).....
```

```
(take 5 foo)  
;; -> (0 1 2 3 5)
```

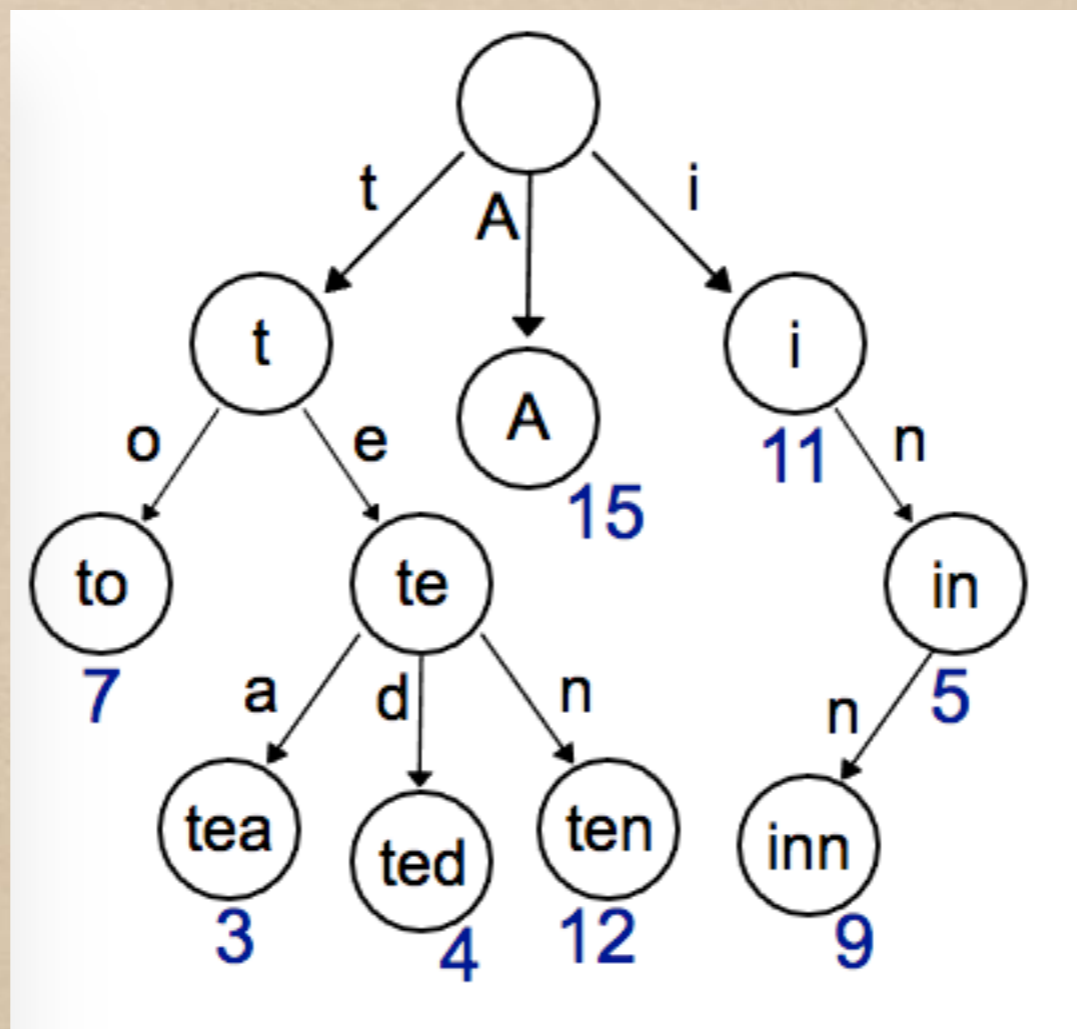
```
(nth foo 32) ;; 求第33个元素时才再次求值  
;; -> .....32
```

Persistence

Persistent bit-partitioned vector trie

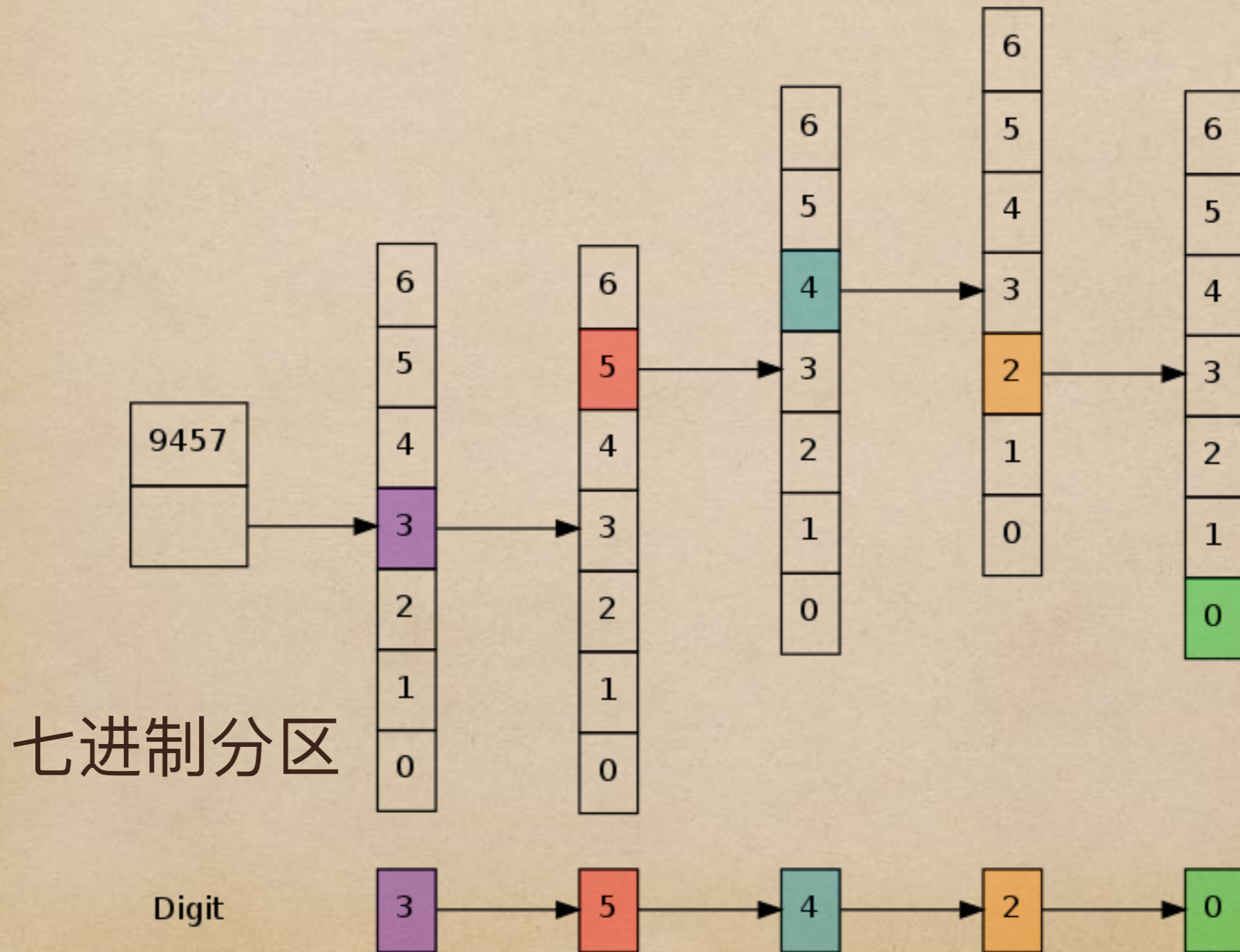
- ◆ Persistent: 所有版本的数据均**可见**
- ◆ vector **数组**, 一小段**连**在一起的内存
- ◆ trie **前缀树**

Trie



前缀树：词频统计、前缀匹配

Digit Partitioning



```
public class DigitTrie {
    public static final int RADIX = 7;

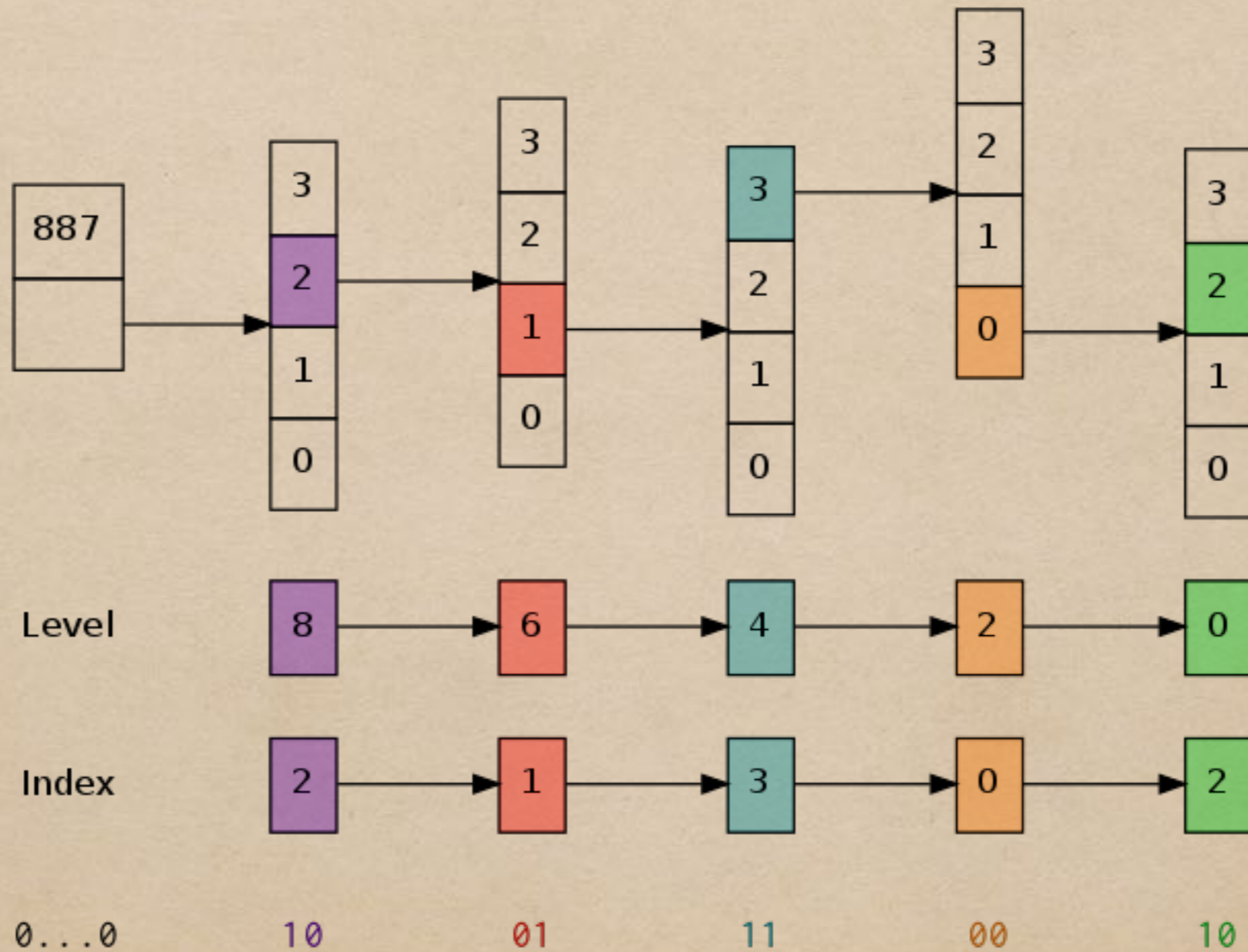
    // Array of objects. Can itself contain an array of objects.
    Object[] root;
    // The maximal size/length of a child node (1 if leaf node)
    int rDepth; // equivalent to RADIX ** (depth - 1)

    public Object lookup(int key) {
        Object[] node = this.root;

        // perform branching on internal nodes here
        for (int size = this.rDepth; size > 1; size /= RADIX) {
            node = (Object[]) node[(key / size) % RADIX];
            // If node may not exist, check if it is null here
        }

        // Last element is the value we want to lookup, return it.
        return node[key % RADIX];
    }
}
```

Bit Partitioning



```
public class BitTrie {
    public static final int BITS = 5,
                          WIDTH = 1 << BITS, // 2^5 = 32
                          MASK = WIDTH - 1; // 31, or 0x1f

    // Array of objects. Can itself contain an array of objects.
    Object[] root;
    // BITS times (the depth of this trie minus one).
    int shift;

    public Object lookup(int key) {
        Object[] node = this.root;

        // perform branching on internal nodes here
        for (int level = this.shift; level > 0; level -= BITS) {
            node = (Object[]) node[(key >>> level) & MASK];
            // If node may not exist, check if it is null here
        }

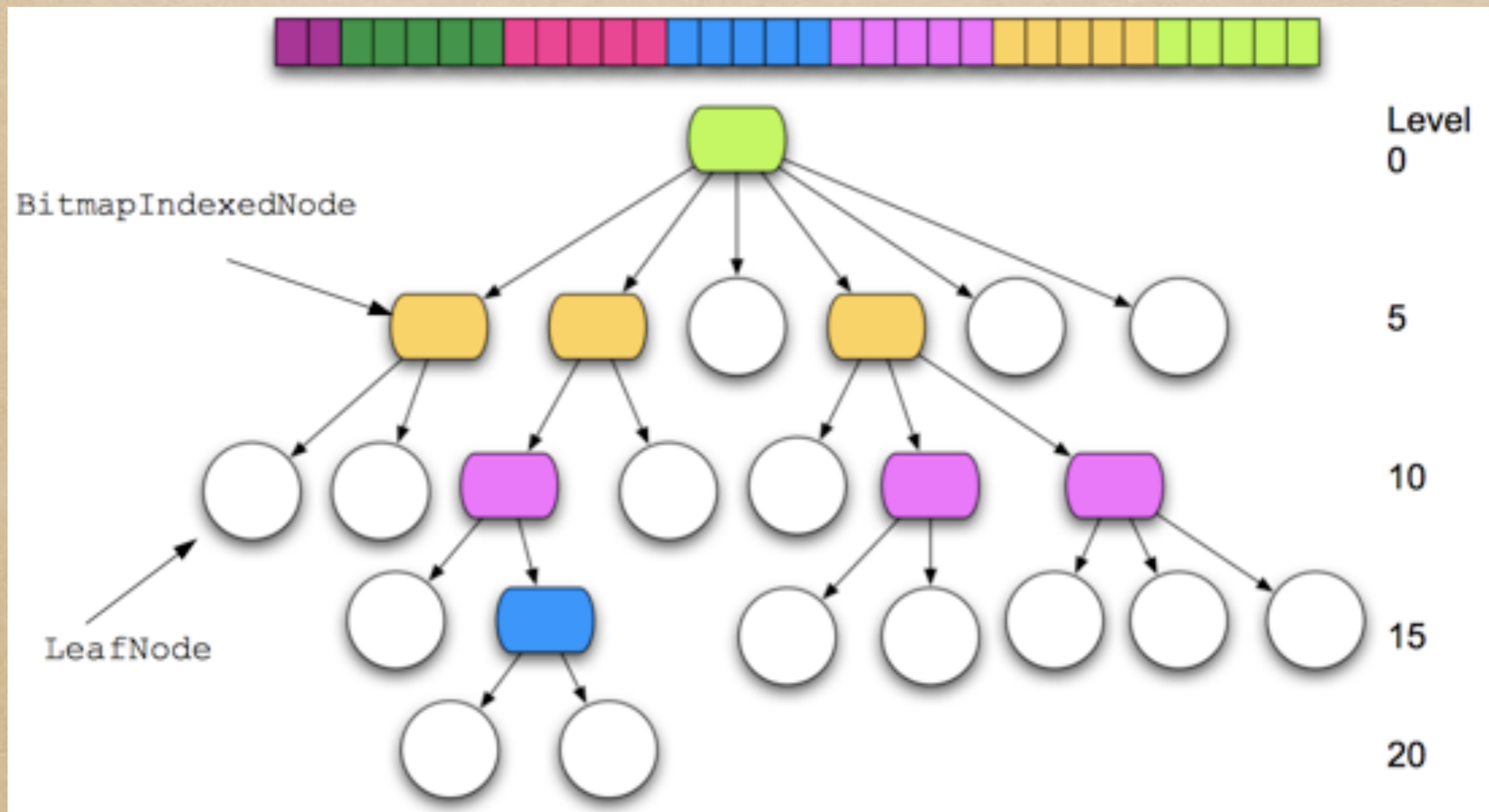
        // Last element is the value we want to lookup, return it.
        return node[key & MASK];
    }
}
```

PersistentVector

```
public Object[] arrayFor(int i){
    if(i >= 0 && i < cnt)
    {
        if(i >= tailoff())
            return tail;
        Node node = root;
        for(int level = shift; level > 0; level -= 5)
            node = (Node) node.array[(i >>> level) & 0x01f];
        return node.array;
    }
    throw new IndexOutOfBoundsException();
}

public Object nth(int i){
    Object[] node = arrayFor(i);
    return node[i & 0x01f];
}
```

PersistentHashMap



Bit-partitioning 的问题

- ◆ `map.put(key, val)`
- ◆ `key` 的 `hash` 值不是连续的整数，如果每个节点存32个太浪费空间，如何解决？
- ◆ 如果只存N个，如何保证bit与index的对应关系

解决方法

- ◆ 解决方法增加一个函数: $[0, 31] \rightarrow [0, n)$
- ◆ 要求: 查询快 $O(1)$
- ◆ bitpos & index 的组合

bitpos

- $\{10^n \mid n \geq 0\}$. 1 后面 n 个 0
- mask 求出当前节点的 index

```
int mask(int hash, int shift){  
    return (hash >>> shift) & 0x01f;  
}
```

```
int bitpos(int hash, int shift){  
    return 1 << mask(hash, shift);  
}
```

index


- bitmap 一个32位的int，每位代表相应的子节点是否存在元素，1 存在，0 不存在。

```
final int index(int bit){  
    return Integer.bitCount(bitmap & (bit - 1));  
}
```

BitmapIndexedNode#find

```
public IMapEntry find(int shift, int hash, Object key){
    int bit = bitpos(hash, shift);
    if((bitmap & bit) == 0)
        return null;
    int idx = index(bit);
    Object keyOrNull = array[2*idx];
    Object valOrNull = array[2*idx+1];
    if(keyOrNull == null)
        return ((INode) valOrNull).find(shift + 5, hash, key);
    if(Util.equiv(key, keyOrNull))
        return (IMapEntry) MapEntry.create(keyOrNull, valOrNull);
    return null;
}
```

解决 Hash 冲突

- ◆ 链表法 
- ◆ 开放地址法
- ◆ 再哈希法

扩展阅读

- ◆ <http://hypirion.com/musings/understanding-persistent-vector-pt-1>
- ◆ <http://blog.higher-order.net/2009/09/08/understanding-clojures-persistent-hashmap-deftwice>

Thank You.



群名称: SICP读书群
群号: 119845407



公众号