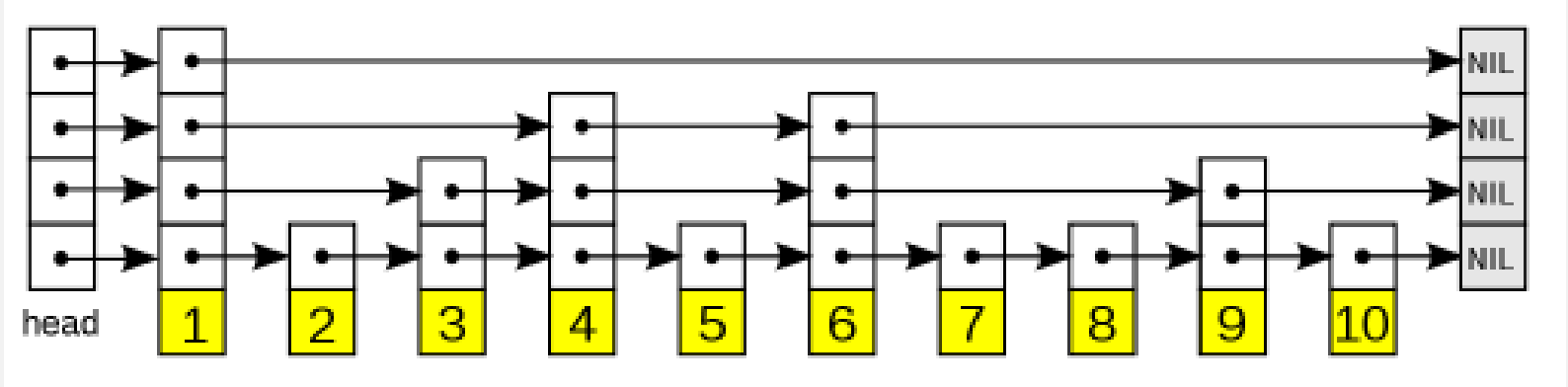
SkipList

练习

DaSE
Data Science
&Engineering

跳表简介




```
  enum { kMaxHeight = 12 };//定义skiplist链表最高节点
// Immutable after construction
Comparator const compare_;//比较器有最顶层的options通过一层一层传递下来，用于///链表排序
Arena* const arena_;    // leveldb内存池，从memtable传过来

Node* const head_;//skiplist头节点

// Modified only by Insert().  Read racily by readers, but stale
// values are ok.
port::AtomicPointer max_height_;   // skiplist目前的最高高度
  // Read/written only by Insert().
Random rnd_;//随机类，用于随机化一个节点高度
```

代码简介

```
1   template<typename Key, class Comparator>
2   SkipList<Key,Comparator>::SkipList(Comparator cmp, Arena* arena)
3       : compare_(cmp),
4         arena_(arena),
5         head_(NewNode(0 /* any key will do */, kMaxHeight)),
6         max_height_(reinterpret_cast<void*>(1)),
7         rnd_(0xdeadbeef) {
8     for (int i = 0; i < kMaxHeight; i++) {
9       head_->SetNext(i, NULL);
10    }
11  }
```

代码简介

```
template<typename Key, class Comparator>
typename SkipList<Key,Comparator>::Node*
SkipList<Key,Comparator>::NewNode(const Key& key, int height) {
  char* mem = arena_->AllocateAligned(
      sizeof(Node) + sizeof(port::AtomicPointer) * (height - 1));//从内存池里面分配
//足够的内存，用于存储新节点。
  return new (mem) Node(key);//返回这个节点。
}
```

代码简介

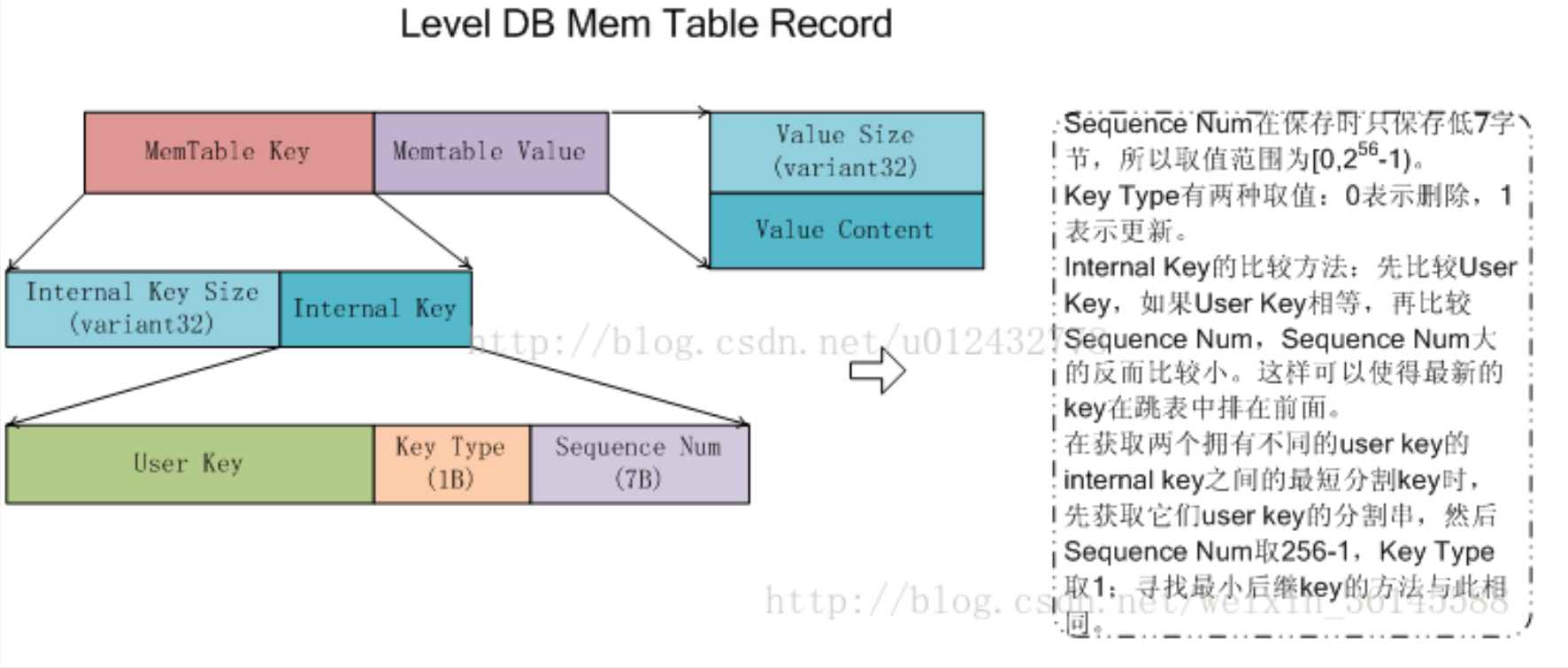
```
template<typename Key, class Comparator>
void SkipList<Key,Comparator>::Insert(const Key& key) {
  Node* prev[kMaxHeight];//kMaxHeight个前节点，因为高度还未知，所以先设为最大值
  Node* x = FindGreaterOrEqual(key, prev);//查找key值节点前GetMaxHeight()个前节点。

  assert(x == NULL || !Equal(key, x->key));

  int height = RandomHeight();//随机化一个节点高度
  if (height > GetMaxHeight()) {//如果当前节点的高度大于最高节点，则高出部分的的前节
//点都是头节点。
    for (int i = GetMaxHeight(); i < height; i++) {
      prev[i] = head_;
    }
    max_height_.NoBarrier_Store(reinterpret_cast<void*>(height));
  }

  x = NewNode(key, height);//新建节点
  for (int i = 0; i < height; i++) {
    x->NoBarrier_SetNext(i, prev[i]->NoBarrier_Next(i))//设立当前节点的后节点;
    prev[i]->SetNext(i, x);//设立当前节点的前节点。
  }
}
```

迭代器简介

Valid(): //判断迭代器当前节点是否有效
Key(): //返回当前节点的key值
Next(): //跳跃链表的第0层就是单链表，所以可以直接指向下一个节点
Prev(): //查找当前节点的上一个节点。
Seek(): //查找某个特定的key值的节点。
SeekToFirst(),
SeekToLast(): //查找第一个和最后一个节点

Kv格式

1.UserKey：用户提供的键值
2.InternalKey：UserKey + KeyType(1)+SequenceNumber(7)
3.LookupKey：EncodeString(InternalKey.size()) + InternalKey



Task 2

输出索引-要求：

从跳表的首指针开始，把每一层的节点信息都打印出来，每次插入都可以看到跳表的变化

修改文件skiplist.h(339+行insert方法后，有一个PrintTable方法，将其完善)

```
/home/rui/Git/courseForLeveldb/cmake-build-debug/db_test
k1:v1
k1:v1
------------PRINT------------
k1:v1
k1:v1   k2:v2
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3   k4:v4
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3   k4:v4   k5:v5
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3   k4:v4   k5:v5   k6:v6
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3   k4:v4   k5:v5   k6:v6   k7:v7
------------PRINT------------
k1:v1
k1:v1   k2:v2   k3:v3   k4:v4   k5:v5   k6:v6   k7:v7   k8:v8
------------PRINT------------
k1:v1   k9:v9
k1:v1   k2:v2   k3:v3   k4:v4   k5:v5   k6:v6   k7:v7   k8:v8   k9:v9
------------PRINT------------
k1:v1   k9:v9
k1:v1   k10:v10 k2:v2   k3:v3   k4:v4   k5:v5   k6:v6   k7:v7   k8:v8   k9:v9
------------PRINT------------

Process finished with exit code 0
```