**山东大学计算机科学与技术学院  
《数据结构与算法》课程设计报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 上机学时：4 | | 日期： 2019.3.21 | |
| 课程设计题目：基本数据结构及其应用之跳表 | | | |
| 软件开发环境：  Microsoft Visual Studio 2017 | | | |
| 报告内容：  1.需求描述  1.1 问题描述  实现并分析跳表结构  1.2 基本要求  （1）构造并实现跳表ADT，跳表ADT 中应包括初始化、查找、插入、删除指定关键字的元素、删除关键字最小的元素、删除关键字最大的元素等基本操作。  （2）分析各基本操作的时间复杂性。  （3）针对一个实例实现跳表的动态演示。  1.3 输入说明  无  1.4 输出说明  无  2.设计  2.1 系统结构设计  (1) 在main.cpp中设计简单的交互提示界面来提示用户如何操作  (2) 跳表实现存放在skipList.h文件中，与main.cpp独立方便维护。使用时仅需将skipList.h文件引入即可。  (3) 跳表节点结构体单独存放在skipNode.h中，并在skipList.h中引入即可。  2.2 设计思路  跳表具有如下性质：  (1) 多层结构。  (2) 每一层都是一个不降序的链表。  (3) 最底层(0)的链表包含所有元素。  (4) 如果一个元素出现在级数i的链表中，则它也会在级数i以下的链表中出现。  (5) 查找平均复杂度为O(log),最优复杂度为O(1),最坏复杂度为O(n)。  2.3 数据及数据类(型)定义  定义每个链表的节点为一个结构体，结构体中包含三个元素element和next指针数组以及整形size，前者用来保存当前节点的键值对信息，后者用来保存每一层链表中下一个节点的地址，size用来记录该节点的层高。  C++语言描述为  struct skipNode {  std::pair<const K, E> element;  skipNode<K, E> \*\*next;  int size;  }  2.4.算法设计及分析  Find: 从最上面一层，头指针开始查找，在每一层查找的时候不停地找下一个节点，直到节点的元素值比要查找的值大为止，然后跳到下一层查找。然后判断当前节点的下一个节点是否等于要查找的值即可。  Insert: 先查找当前要插入的节点是否在跳表中，如果存在则直接修改value，否则就随机一个level值，将它插入到这一层里。如果随机出的level值比当前最大的level值还要大，更新最大的level值。然后新建一个节点，把它插入到从第0层到第level层中每一层的last[i]的后面。  Erase: 先调用search函数查找要删除的节点，如果这个节点在跳表里的话，将每一层的last[i]之后的节点删除掉。  Output：从最大level值枚举层数，然后遍历链表中的所有节点，如果节点的最大层数小于当前层数则说面该节点不会在该层中出现，跳过即可，否则输出对应元素。  3. 测试结果    4. 分析与探讨  跳表的理想复杂度是优秀的O(log)级别，但这是建立在跳表理想的结构之上的，如果随着元素的插入删除，其理想化结构会被破坏，导致查询的复杂度退化为O(n)。此是应对跳表做整理使其继续保持理想的结构来避免复杂度的退化。  5. 附录：实现源代码  /\*main.cpp\*/  #include <iostream>  #include "skipList.h"  #include <cstdlib>  #include <ctime>  #include <cstdlib>  #include <Windows.h>  using namespace std;  void display() {  std::cout << "======================================" << std::endl;  std::cout << "|| 1. insert key value ||" << std::endl;  std::cout << "|| 2. find key ||" << std::endl;  std::cout << "|| 3. erase key ||" << std::endl;  std::cout << "|| 4. erase maxKey ||" << std::endl;  std::cout << "|| 5. erase minKey ||" << std::endl;  std::cout << "|| 6. display ||" << std::endl;  std::cout << "======================================" << std::endl;  return;  }  int main() {  srand(time(0));  skipList<int, int> sl(-100000000, 100000000, 100000, 0.7);  std::pair<const int, int>\* res;  display();  while (true) {  int op, key, value;  std::cin >> op;  switch (op) {  case 1:  std::cin >> key >> value;  sl.insert(key, value);  break;  case 2:  std::cin >> key;  res = sl.find(key);  if (res != nullptr) std::cout << "(" << res->first << ", " << res->second << ")" << std::endl;  else std::cout << "No such element" << std::endl;  break;  case 3:  std::cin >> key;  sl.erase(key);  std::cout << "ok" << std::endl;  break;  case 4:  sl.eraseMaxKey();  std::cout << "ok" << std::endl;  break;  case 5:  sl.eraseMinKey();  std::cout << "ok" << std::endl;  break;  case 6:  sl.output();  break;  default:  return 0;  }  }  return 0;  }  /\*skipNode.h\*/  #ifndef skipNode\_  #define skipNode\_  #include <utility>  #include <string>  template<typename K, typename E>  struct skipNode {  std::pair<const K, E> element;  skipNode<K, E> \*\*next;  int size;  skipNode(const std::pair<const K, E>& thePair, int size) :element(thePair), size(size) {  next = new skipNode<K, E>\*[size];  }  std::string to\_str() const {  return std::string("(") + std::to\_string(element.first) + std::string(", ") + std::to\_string(element.second) + std::string(")");  }  };  #endif  /\*skipList.h\*/  #ifndef skipList\_  #define skipList\_  #include <utility>  #include <cstdlib>  #include <iostream>  #include <cmath>  #include "skipNode.h"  template<typename K, typename E>  class skipList {  public:  skipList(const K, const K, int = 100000, float = 0.5);  ~skipList();  bool empty() const { return dSize == 0; }  int size() const { return dSize; }  std::pair<const K, E>\* find(const K&) const;  void insert(const K&, const E&);  void erase(const K&);  void eraseMinKey();  void eraseMaxKey();  void output(std::ostream& = std::cout);  protected:  float prob; // 级分配的可能  int dSize; // 节点个数  int maxLevel; // 最大层数  int level; // 当前层数  K tailKey; // 最大键值  K headKey; // 最小键值  skipNode<K, E>\* headNode; // 头节点  skipNode<K, E>\* tailNode; // 尾节点 不需要指针域  skipNode<K, E>\*\* last; // 记录查询到的最后一个指针  };  template<typename K, typename E>  skipList<K, E>::skipList(const K headKey, const K tailKey, int maxPaires, float prob) {  this->prob = prob \* RAND\_MAX;  this->maxLevel = ceil(log(maxPaires) / log(1 / prob));  this->dSize = this->level = 0;  this->tailKey = tailKey;  this->headKey = headKey;  std::pair<K, E> tailPair;  tailPair.first = headKey;  headNode = new skipNode<K, E>(tailPair, maxLevel);  tailPair.first = tailKey;  tailNode = new skipNode<K, E>(tailPair, 0);  last = new skipNode<K, E>\*[maxLevel];  for (int i = 0; i < maxLevel; ++i) headNode->next[i] = tailNode;  }  template<typename K, typename E>  skipList<K, E>::~skipList() {  while (headNode != tailNode) {  skipNode<K, E>\* nextNode = headNode->next[0];  delete headNode;  headNode = nextNode;  }  delete tailNode;  delete[] last;  }  template<typename K, typename E>  std::pair<const K, E>\* skipList<K, E>::find(const K& theKey) const {  if (theKey >= tailKey || theKey <= headKey) return nullptr;  int cnt = 1;  skipNode<K, E>\* nextNode = headNode;  for (int i = level; i >= 0; --i) {  while (nextNode->next[i]->element.first < theKey) {  nextNode = nextNode->next[i];  cnt++;  }  }  std::cout << "find: " << cnt << std::endl;  if (nextNode->next[0]->element.first == theKey) return &nextNode->next[0]->element;  return nullptr;  }  template<typename K, typename E>  void skipList<K, E>::insert(const K& theKey, const E& theElement) {  if (theKey >= tailKey || theKey <= headKey) {  return;  }  int cnt = 0;  skipNode<K, E>\* nextNode = headNode;  for (int i = level; i >= 0; --i) {  while (nextNode->next[i]->element.first < theKey) {  nextNode = nextNode->next[i];  cnt++;  }  last[i] = nextNode;  }  nextNode = nextNode->next[0];  if (nextNode->element.first == theKey) {  nextNode->element.second = theElement;  std::cout << "insert: " << cnt << std::endl;  return;  }  int lev = 0;  while (rand() <= prob) lev++;  if (lev >= maxLevel) lev = maxLevel - 1;  if (lev > level) {  lev = ++level;  last[lev] = headNode;  }  skipNode<K, E>\* newNode = new skipNode<K, E>(std::make\_pair(theKey, theElement), lev + 1);  for (int i = 0; i <= lev; ++i) {  newNode->next[i] = last[i]->next[i];  last[i]->next[i] = newNode;  cnt += 2;  }  std::cout << "insert: " << cnt << std::endl;  dSize++;  }  template<typename K, typename E>  void skipList<K, E>::erase(const K& theKey) {  if (theKey >= tailKey || theKey <= headKey) {  return;  }  int cnt = 0;  skipNode<K, E>\* nextNode = headNode;  for (int i = level; i >= 0; --i) {  while (nextNode->next[i]->element.first < theKey) {  nextNode = nextNode->next[i];  cnt++;  }  last[i] = nextNode;  }  nextNode = nextNode->next[0];  if (nextNode->element.first != theKey) return;  for (int i = 0; i <= level; ++i) {  if (last[i]->next[i] != nextNode) break;  last[i]->next[i] = nextNode->next[i];  cnt++;  }  std::cout << "erase: " << cnt << std::endl;  while (level > 0 && headNode->next[level] == tailNode) level--;  delete nextNode;  dSize--;  }  template<typename K, typename E>  void skipList<K, E>::eraseMinKey() {  skipNode<K, E>\* theNode = headNode->next[0];  if (theNode == tailNode) return;  erase(theNode->element.first);  }  template<typename K, typename E>  void skipList<K, E>::eraseMaxKey() {  skipNode<K, E>\* nextNode = headNode->next[0];  int maxKey = headKey;  while (nextNode != tailNode) {  maxKey = nextNode->element.first;  nextNode = nextNode->next[0];  }  if (maxKey == headKey) return;  this->erase(maxKey);  }  template<typename K, typename E>  void skipList<K, E>::output(std::ostream& out) {  skipNode<K, E> \*theNode;  for (int i = level; i >= 0; --i) {  out << headNode->to\_str() << ' ';  theNode = headNode->next[0];  while (theNode != tailNode) {  if (theNode->size < i) for (int i = 0; i < theNode->to\_str().length() + 4; ++i) out << '-';  else out << "-> " << theNode->to\_str() << ' ';  theNode = theNode->next[0];  }  out << "-> " << tailNode->to\_str() << std::endl;  }  out << std::endl;  }  #endif | | | |
|  | | | |