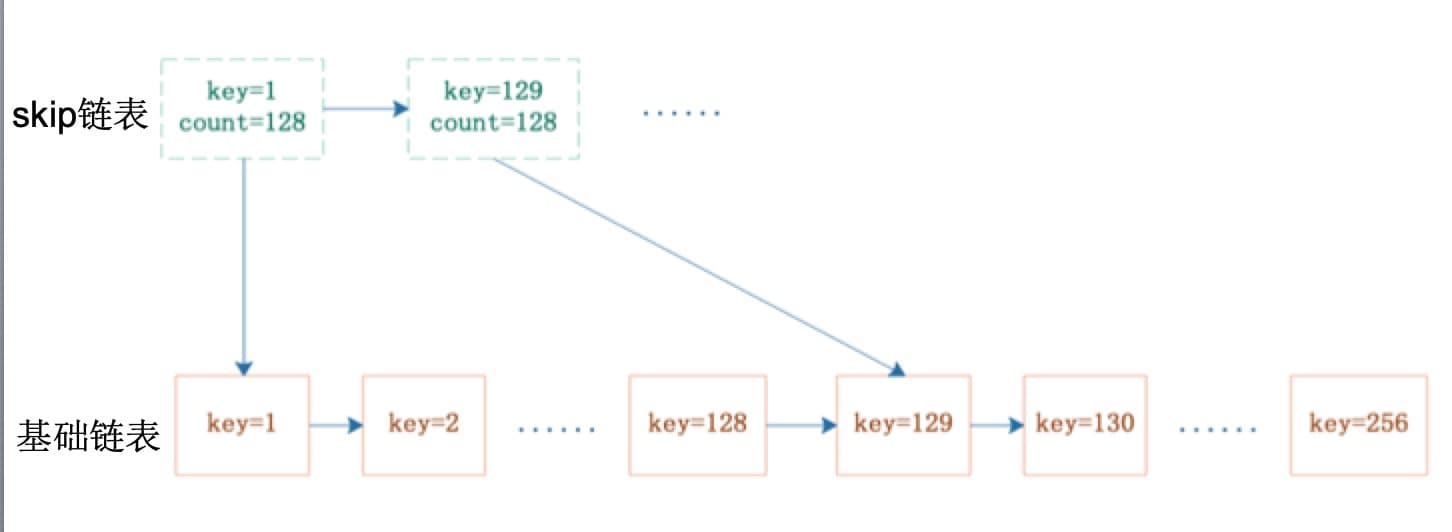
有序链表结构在离线数据处理场景下是一种可选数据容器，按照key从小到大的顺序组织成单向链表，其具体要求如下：

（1) 插入节点：遍历链表，按照key有序的原则找到位置后进行节点的插入，如果发现存在重复key的节点则进行value值替换

（2) 查询节点：遍历链表，找到目标key的节点，返回value值

单向链表随着插入节点的规模增长，插入和查询性能会遇到严重的性能下降问题。

一种优化思路是：在有序链表基础上，添加一种基于链表结构的逻辑跳表来加速定位过程，原理如下图：



编码要求：

1. 实现基于基础链表，添加单层逻辑跳表（skip链表结构）的优化版本有序链表，不考虑多级跳表优化（跳表之上也添加优化跳表结构）

2. 假设逻辑跳表中的单个节点最多对应128个基础链表的LinkNode结构，超过则需要添加新的跳表节点

请编码实现带有逻辑跳表的有序链表的以下操作接口

1. 指定key插入value值

2. 指定key查询value

**编译器版本:** gcc 4.8.4

请使用标准输入输出(stdin，stdout) ；请把所有程序写在一个文件里，勿使用已禁用图形、文件、网络、系统相关的头文件和操作，如sys/stat.h , unistd.h , curl/curl.h , process.h

**时间限制:** 3S (C/C++以外的语言为: 5 S)   **内存限制:** 128M (C/C++以外的语言为: 640 M)

**输入:**

输入为单行字符串输入：存在两种输入模式 1. 输入格式为list:key,value;key,value;... list表示后续为实际insert的kv列表 key,value分别描述要添加的key和value值（都是uint64类型整数） 2. 输入格式为count:N（N是整数类型） count描述要添加的节点个数，":"后指定具体的个数值。测试过程会随机构造N个乱序的key/value进行插入

**输出:**

输出bool值描述校验结果 PASS：符合预期 FAIL：不符合预期

**输入范例:**

输入范例： list:8,2;4,5;7;7 count:10

**输出范例:**

PASS 或 FAIL

#include <iostream>

#include <vector>

#include <numeric>

#include <limits>

#include <algorithm>

#include <stdint.h>

#include <string>

using namespace std;

/\*

编译器版本: gcc 4.8.4

请使用标准输入输出(stdin，stdout) ；请把所有程序写在一个文件里，勿使用已禁用图形、文件、网络、系统相关的头文件和操作，如sys/stat.h , unistd.h , curl/curl.h , process.h

时间限制: 3S (C/C++以外的语言为: 5 S) 内存限制: 128M (C/C++以外的语言为: 640 M)

输入:

输入为单行字符串输入：存在两种输入模式

1. 输入格式为list:key,value;key,value;...

list表示后续为实际insert的kv列表

key,value分别描述要添加的key和value值（都是uint64类型整数）

2. 输入格式为count:N（N是整数类型）

count描述要添加的节点个数，":"后指定具体的个数值。测试过程会随机构造N个乱序的key/value进行插入

输出:

输出bool值描述校验结果

PASS：符合预期

FAIL：不符合预期

输入范例:

输入范例：

list:8,2;4,5;7;7

count:10

输出范例:

PASS 或 FAIL

\*/

struct LinkNode

{

uint64\_t key;

uint64\_t value;

LinkNode\* next = NULL;

};

struct SkipLinkNode

{

uint64\_t key;

int count;

LinkNode\* pointer = NULL;

SkipLinkNode\* next = NULL;

};

class SkipLinkList

{

public:

SkipLinkList(int skipStep = 128);

~SkipLinkList();

bool simpleCheck();

public:

void insert(uint64\_t key, uint64\_t value);

bool find(uint64\_t key, uint64\_t &value);

private:

LinkNode\* \_listHeader = NULL;

SkipLinkNode\* \_skipListHeader = NULL;

int \_skipStep;

int \_itemCount;

};

SkipLinkList::SkipLinkList(int skipStep)

: \_skipStep(skipStep)

, \_listHeader(NULL)

, \_skipListHeader(NULL)

, \_itemCount(0)

{}

SkipLinkList::~SkipLinkList()

{

LinkNode\* node = \_listHeader;

while (node != NULL) {

LinkNode\* toDelNode = node;

node = node->next;

delete toDelNode;

}

SkipLinkNode\* skipNode = \_skipListHeader;

while (skipNode != NULL) {

SkipLinkNode\* toDelNode = skipNode;

skipNode = skipNode->next;

delete toDelNode;

}

}

bool SkipLinkList::simpleCheck()

{

uint64\_t lastKey = 0;

int nodeCount = 0;

LinkNode\* node = \_listHeader;

while (node != NULL)

{

nodeCount++;

if (node->key < lastKey)

{

return false;

}

lastKey = node->key;

node = node->next;

}

if (nodeCount < \_skipStep)

{

return (\_skipListHeader == NULL);

}

lastKey = 0;

int skipNodeCount = 0;

int expectNodeCount = 0;

SkipLinkNode\* skipNode = \_skipListHeader;

while (skipNode != NULL)

{

skipNodeCount++;

if (skipNode->key < lastKey)

{

return false;

}

lastKey = skipNode->key;

expectNodeCount += skipNode->count;

skipNode = skipNode->next;

}

return nodeCount == expectNodeCount &&

skipNodeCount >= ((nodeCount + \_skipStep - 1) / \_skipStep);

}

/\*\* 请完成下面这个函数，实现题目要求的功能 \*\*/

void SkipLinkList::insert(uint64\_t key, uint64\_t value)

{

// TODO:

if (\_listHeader == NULL) {

\_listHeader = new LinkNode;

\_listHeader->key = key;

\_listHeader->value = value;

\_skipListHeader = new SkipLinkNode;

\_skipListHeader->key = key;

\_skipListHeader->pointer = \_listHeader;

\_skipListHeader->next = NULL;

\_skipListHeader->count = 1;

\_skipStep = 128;

\_itemCount = 1;

}

else {

uint64\_t value1;

if (!find(key, value1)) {

LinkNode\* newLinkNode = new LinkNode;

newLinkNode->key = key;

newLinkNode->value = value;

LinkNode \*p = \_listHeader;

for (int i = 0; i < \_itemCount; i++)

p = p->next;

p->next = newLinkNode;

\_itemCount++;

\_itemCount %= 128;

if (\_itemCount == 1) {

SkipLinkNode\* newSkipLinkNode = new SkipLinkNode;

newSkipLinkNode->count = 1;

newSkipLinkNode->key = key;

newSkipLinkNode->next = NULL;

SkipLinkNode\* sp = \_skipListHeader;

while (sp->next) {

sp = sp->next;

}

sp->next = newSkipLinkNode;

newSkipLinkNode->pointer = newLinkNode;

}

else {

SkipLinkNode\* sp = \_skipListHeader;

while (sp->next) {

sp = sp->next;

}

sp->count++;

}

}

else {

if (value != value1) {

int skey = key / 128;

SkipLinkNode \*sp = \_skipListHeader;

while (sp && sp->next && skey > 0) {

sp = sp->next;

skey--;

}

LinkNode \* p = sp->pointer;

while (p && p->key < key) {

p = p->next;

}

if (p->key == key) {

p->value = value;

}

}

}

}

}

bool SkipLinkList::find(uint64\_t key, uint64\_t &value)

{

// TODO:

int skey = key / 128;

SkipLinkNode \*sp = \_skipListHeader;

while (sp && sp->next && skey > 0) {

sp = sp->next;

skey--;

}

LinkNode \* p = sp->pointer;

while(p && p->key<key) {

p = p->next;

}

if (p->key == key) {

value = p->value;

return true;

}

else

return false;

}

/\*\* 请完成上面的函数，实现题目要求的功能 \*\*/

vector<string> splitString(const string& text, const string &sepStr) {

vector<string> vec;

string str(text);

string sep(sepStr);

size\_t n = 0, old = 0;

while (n != string::npos)

{

n = str.find(sep, n);

if (n != string::npos)

{

vec.push\_back(str.substr(old, n - old));

n += sep.length();

old = n;

}

}

vec.push\_back(str.substr(old, str.length() - old));

return vec;

}

template <typename T>

bool stringToInteger(const string& text, T& value) {

const char\* str = text.c\_str();

char\* endPtr = NULL;

value = (T)strtol(str, &endPtr, 10);

return (endPtr && \*endPtr == 0);

}

bool testSkipLinkList(string inputParam) {

vector<string> inputVec = splitString(inputParam, ":");

if (inputVec.size() != 2 || (inputVec[0] != "list" && inputVec[0] != "count"))

{

cout << "input format error!" << endl;

return false;

}

// prepare data

vector<pair<uint64\_t, uint64\_t> > keyValueVec;

if (inputVec[0] == "list") {

vector<string> kvVec = splitString(inputVec[1], ";");

for (size\_t i = 0; i < kvVec.size(); i++) {

vector<string> kvStr = splitString(kvVec[i], ",");

uint64\_t key, value;

if (kvStr.size() != 2 ||

!stringToInteger<uint64\_t>(kvStr[0], key) ||

!stringToInteger<uint64\_t>(kvStr[1], value))

{

cout << "key-value list format error!" << endl;

return false;

}

keyValueVec.push\_back(make\_pair(key, value));

}

}

if (inputVec[0] == "count") {

uint64\_t count;

if (!stringToInteger<uint64\_t>(inputVec[1], count)) {

cout << "count format error!" << endl;

return false;

}

for (uint64\_t i = 0; i < count; i++) {

keyValueVec.push\_back(make\_pair(i, i));

}

random\_shuffle(keyValueVec.begin(), keyValueVec.end());

}

uint64\_t maxKey = 0;

uint64\_t value;

SkipLinkList list(4);

for (size\_t i = 0; i < keyValueVec.size(); i++)

{

list.insert(keyValueVec[i].first, keyValueVec[i].second);

// test find

bool ret = list.find(keyValueVec[i].first, value);

if (!ret || value != keyValueVec[i].second)

{

cout << "test find error!" << endl;

return false;

}

// test replace

list.insert(keyValueVec[i].first, keyValueVec[i].second + 1);

ret = list.find(keyValueVec[i].first, value);

if (!ret || value != keyValueVec[i].second + 1)

{

cout << "test replace error!" << endl;

return false;

}

maxKey = max(maxKey, keyValueVec[i].first);

}

// test not exist key

if (list.find(maxKey + 1, value))

{

cout << "find not exist error!" << endl;

return false;

}

return list.simpleCheck();

}

int main() {

bool res;

string \_inputParam;

getline(cin, \_inputParam);

res = testSkipLinkList(\_inputParam);

if (res)

{

cout << "PASS" << endl;

}

else

{

cout << "FAIL" << endl;

}

return 0;

}

飞鸽传书

古代人用飞鸽传书来传递情报，但其安全性存在很大的隐患，当信鸽被敌人发现后会造成信息泄露或者信息被篡改，造成重大损失。于是有个聪明人，教会鸽子可以发出2种声音“咕，呱”，但每个鸽子最多可以按顺序记住6个发声，比如“咕咕呱咕呱呱”；此人养了20只这种鸽子，现在想用这些鸽子发送情报。假设鸽子都能飞达目的地，但鸽子到达的顺序和鸽子放飞的顺序可能并不一致。鸽子之间只能通过叫声区分，不能通过外表或者其他标记区分。

现在需要用这些鸽子来传递信息，每条信息由26个大写英文字母，加上$%^&\*五个符号，共计31个字符组成。每条信息不超过1023个字符。

现在需要对每条信息输出一个传输方案，传输方案是由两个数组成（2^N，M），2^N是本次单条信息单批次放飞的鸽子数目，M是需要的放飞批次（往返一次算一次放飞批次）。 输出最佳方案的数值，最佳方案定义为M最小情况下的N最小值。

**输入信息：**

**ABC  
ABCD  
YYAB^**

**输出信息：**

**8 1  
16 1  
16 1**

**编译器版本:** gcc 4.8.4

请使用标准输入输出(stdin，stdout) ；请把所有程序写在一个文件里，勿使用已禁用图形、文件、网络、系统相关的头文件和操作，如sys/stat.h , unistd.h , curl/curl.h , process.h

**时间限制:** 3S (C/C++以外的语言为: 5 S)   **内存限制:** 128M (C/C++以外的语言为: 640 M)

**输入:**

输入数据包含多行，每行表示需要传递的信息。

**输出:**

输出数据如题干要求：

**输入范例:**

AA AA&^ AAAAA

**输出范例:**

4 1 16 1 16 1