[如何用B+树设计数据库中的索引文件](http://blog.csdn.net/tyronerenekton/article/details/53676527)

版权声明：本文为博主原创文章，未经博主允许不得转载。

**声明：**   
1、B+树的代码不是我写的，是网上的，关于[**Java**](http://lib.csdn.net/base/java)写的B+树的都是这个代码，我也不知道怎么写原作者。   
2、如果不懂B+树是肯定看不懂这篇blog的。   
3、我在原有代码上简单修改了两个地方：第一、在叶子节点的属性集合里添加了file属性。第二、在BplusTree里添加了将叶子节点的链表保存到文件的代码

**为什么需要用B+树设计索引文件？**

因为单个索引文件太大了，当你进行查询的时候需要打开文件，然后在磁盘上寻道，读取文件内容，这会花费大量的时间。所以我们需要通过B+树来将原来的大的索引文件分成很多个小的索引文件。比如说，本来一个文件有一亿个int数。那当你查询某一个数时要花费很长世间【要读取所有的文件内容】。但是如果你通过将一亿个数分成一千个文件，每个文件平均一万个数，然后B+数查询的复杂度是log（一亿），约为27。然后打开叶子节点的文件，再读取这一万个数，再用二分查找，就可以减少大量的读文件内容时间。

不难看出，读一亿个数和读一万个数差距还是很大的。当然，索引文件会占用大量的存储空间。【这里就是用空间换时间】

**怎么实现呢？**

不懂B+树的同志需要自己去研究一下B+树了。这里就不多讲了。

我们知道B+树的叶子节点会有一个链表。那么我们将数据生成B+树之后完全可以将叶子节点的数据保存到每一个叶子结点自己的文件里。

**我是用的Java写的**，直接将链表通过对象序列化，写到文件里。   
然后将B+树通过对象序列化写到文件里。

**下次查询的时候，读取文件里的B+树，然后通过节点上的关键字最终决定打开哪一个叶子节点的文件。**

**Show me code！**

**这里的B+树代码是网上的**，没错，大家搜java实现B+树就能搜到。我只是稍作修改。

public interface B {

public Object get(Comparable key); //查询

public void remove(Comparable key); //移除

public void insertOrUpdate(Comparable key, Object obj); //插入或者更新，如果已经存在，就更新，否则插入

}

* 1
* 2
* 3
* 4
* 5
* 1
* 2
* 3
* 4
* 5

package IndexTree;

/\*\*

\* Created by hms on 2016/12/12.

\*/

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Random;

public class BplusTree implements B , Serializable{

/\*\* 根节点 \*/

protected Node root;

/\*\* 阶数，M值 \*/

protected int order;

/\*\* 叶子节点的链表头\*/

protected Node head;

public Node getHead() {

return head;

}

public void setHead(Node head) {

this.head = head;

}

public Node getRoot() {

return root;

}

public void setRoot(Node root) {

this.root = root;

}

public int getOrder() {

return order;

}

public void setOrder(int order) {

this.order = order;

}

@Override

public Object get(Comparable key) {

return root.get(key);

}

@Override

public void remove(Comparable key) {

root.remove(key, this);

}

@Override

public void insertOrUpdate(Comparable key, Object obj) {

root.insertOrUpdate(key, obj, this);

}

public BplusTree(int order){

if (order < 3) {

System.out.print("order must be greater than 2");

System.exit(0);

}

this.order = order;

root = new Node(true, true);

head = root;

}

//测试

public static void main(String[] args) throws IOException {

BplusTree tree = new BplusTree(6);

Random random = new Random();

long current = System.currentTimeMillis();

for (int j = 0; j < 100000; j++) {

for (int i = 0; i < 100; i++) {

int randomNumber = random.nextInt(1000);

tree.insertOrUpdate(randomNumber, randomNumber);

}

for (int i = 0; i < 100; i++) {

int randomNumber = random.nextInt(1000);

tree.remove(randomNumber);

}

}

long duration = System.currentTimeMillis() - current;

System.out.println("time elpsed for duration: " + duration);

int search = 80;

System.out.print(tree.get(search));

Node next = tree.getHead();

int count = 0;

while(true){

if(next == null) break;

++count;

List<Entry<Comparable, Object>> entries = next.getEntries();

File file = new File(String.valueOf(count) + ".txt");

next.setFile(file);

ObjectOutputStream objectOutputStream =

new ObjectOutputStream(new FileOutputStream(file));

objectOutputStream.writeObject(next);

objectOutputStream.close();

next = next.getNext();

}

File treeFile = new File("BplusTree.txt");

ObjectOutputStream objectOutputStream =

new ObjectOutputStream(new FileOutputStream(treeFile));

objectOutputStream.writeObject(tree);

objectOutputStream.close();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114

package IndexTree;

/\*\*

\* Created by hms on 2016/12/12.

\*/

import java.io.File;

import java.io.Serializable;

import java.util.AbstractMap.SimpleEntry;

import java.util.ArrayList;

import java.util.List;

import java.util.Map.Entry;

public class Node implements Serializable {

/\*\* 是否为叶子节点 \*/

protected boolean isLeaf;

/\*\* 是否为根节点\*/

protected boolean isRoot;

/\*\* 父节点 \*/

protected Node parent;

/\*\* 叶节点的前节点\*/

protected Node previous;

/\*\* 叶节点的后节点\*/

protected Node next;

/\*\* 节点的关键字 \*/

protected List<Entry<Comparable, Object>> entries;

/\*\* 子节点 \*/

protected List<Node> children;

/\*\*每个叶子节点对应的索引文件\*/

protected File file;

public Node(boolean isLeaf) {

this.isLeaf = isLeaf;

entries = new ArrayList<Entry<Comparable, Object>>();

if (!isLeaf) {

children = new ArrayList<Node>();

}

}

public Node(boolean isLeaf, boolean isRoot) {

this(isLeaf);

this.isRoot = isRoot;

}

public Object get(Comparable key) {

//如果是叶子节点

if (isLeaf) {

for (Entry<Comparable, Object> entry : entries) {

if (entry.getKey().compareTo(key) == 0) {

//返回找到的对象

return entry.getValue();

}

}

//未找到所要查询的对象

return null;

//如果不是叶子节点

}else {

//如果key小于等于节点最左边的key，沿第一个子节点继续搜索

if (key.compareTo(entries.get(0).getKey()) <= 0) {

return children.get(0).get(key);

//如果key大于节点最右边的key，沿最后一个子节点继续搜索

}else if (key.compareTo(entries.get(entries.size()-1).getKey()) >= 0) {

return children.get(children.size()-1).get(key);

//否则沿比key大的前一个子节点继续搜索

}else {

for (int i = 0; i < entries.size(); i++) {

if (entries.get(i).getKey().compareTo(key) <= 0 && entries.get(i+1).getKey().compareTo(key) > 0) {

return children.get(i).get(key);

}

}

}

}

return null;

}

public void insertOrUpdate(Comparable key, Object obj, BplusTree tree){

//如果是叶子节点

if (isLeaf){

//不需要分裂，直接插入或更新

if (contains(key) || entries.size() < tree.getOrder()){

insertOrUpdate(key, obj);

if (parent != null) {

//更新父节点

parent.updateInsert(tree);

}

//需要分裂

}else {

//分裂成左右两个节点

Node left = new Node(true);

Node right = new Node(true);

//设置链接

if (previous != null){

previous.setNext(left);

left.setPrevious(previous);

}

if (next != null) {

next.setPrevious(right);

right.setNext(next);

}

if (previous == null){

tree.setHead(left);

}

left.setNext(right);

right.setPrevious(left);

previous = null;

next = null;

//左右两个节点关键字长度

int leftSize = (tree.getOrder() + 1) / 2 + (tree.getOrder() + 1) % 2;

int rightSize = (tree.getOrder() + 1) / 2;

//复制原节点关键字到分裂出来的新节点

insertOrUpdate(key, obj);

for (int i = 0; i < leftSize; i++){

left.getEntries().add(entries.get(i));

}

for (int i = 0; i < rightSize; i++){

right.getEntries().add(entries.get(leftSize + i));

}

//如果不是根节点

if (parent != null) {

//调整父子节点关系

int index = parent.getChildren().indexOf(this);

parent.getChildren().remove(this);

left.setParent(parent);

right.setParent(parent);

parent.getChildren().add(index,left);

parent.getChildren().add(index + 1, right);

setEntries(null);

setChildren(null);

//父节点插入或更新关键字

parent.updateInsert(tree);

setParent(null);

//如果是根节点

}else {

isRoot = false;

Node parent = new Node(false, true);

tree.setRoot(parent);

left.setParent(parent);

right.setParent(parent);

parent.getChildren().add(left);

parent.getChildren().add(right);

setEntries(null);

setChildren(null);

//更新根节点

parent.updateInsert(tree);

}

}

//如果不是叶子节点

}else {

//如果key小于等于节点最左边的key，沿第一个子节点继续搜索

if (key.compareTo(entries.get(0).getKey()) <= 0) {

children.get(0).insertOrUpdate(key, obj, tree);

//如果key大于节点最右边的key，沿最后一个子节点继续搜索

}else if (key.compareTo(entries.get(entries.size()-1).getKey()) >= 0) {

children.get(children.size()-1).insertOrUpdate(key, obj, tree);

//否则沿比key大的前一个子节点继续搜索

}else {

for (int i = 0; i < entries.size(); i++) {

if (entries.get(i).getKey().compareTo(key) <= 0 && entries.get(i+1).getKey().compareTo(key) > 0) {

children.get(i).insertOrUpdate(key, obj, tree);

break;

}

}

}

}

}

/\*\* 插入节点后中间节点的更新 \*/

protected void updateInsert(BplusTree tree){

validate(this, tree);

//如果子节点数超出阶数，则需要分裂该节点

if (children.size() > tree.getOrder()) {

//分裂成左右两个节点

Node left = new Node(false);

Node right = new Node(false);

//左右两个节点关键字长度

int leftSize = (tree.getOrder() + 1) / 2 + (tree.getOrder() + 1) % 2;

int rightSize = (tree.getOrder() + 1) / 2;

//复制子节点到分裂出来的新节点，并更新关键字

for (int i = 0; i < leftSize; i++){

left.getChildren().add(children.get(i));

left.getEntries().add(new SimpleEntry(children.get(i).getEntries().get(0).getKey(), null));

children.get(i).setParent(left);

}

for (int i = 0; i < rightSize; i++){

right.getChildren().add(children.get(leftSize + i));

right.getEntries().add(new SimpleEntry(children.get(leftSize + i).getEntries().get(0).getKey(), null));

children.get(leftSize + i).setParent(right);

}

//如果不是根节点

if (parent != null) {

//调整父子节点关系

int index = parent.getChildren().indexOf(this);

parent.getChildren().remove(this);

left.setParent(parent);

right.setParent(parent);

parent.getChildren().add(index,left);

parent.getChildren().add(index + 1, right);

setEntries(null);

setChildren(null);

//父节点更新关键字

parent.updateInsert(tree);

setParent(null);

//如果是根节点

}else {

isRoot = false;

Node parent = new Node(false, true);

tree.setRoot(parent);

left.setParent(parent);

right.setParent(parent);

parent.getChildren().add(left);

parent.getChildren().add(right);

setEntries(null);

setChildren(null);

//更新根节点

parent.updateInsert(tree);

}

}

}

/\*\* 调整节点关键字\*/

protected static void validate(Node node, BplusTree tree) {

// 如果关键字个数与子节点个数相同

if (node.getEntries().size() == node.getChildren().size()) {

for (int i = 0; i < node.getEntries().size(); i++) {

Comparable key = node.getChildren().get(i).getEntries().get(0).getKey();

if (node.getEntries().get(i).getKey().compareTo(key) != 0) {

node.getEntries().remove(i);

node.getEntries().add(i, new SimpleEntry(key, null));

if(!node.isRoot()){

validate(node.getParent(), tree);

}

}

}

// 如果子节点数不等于关键字个数但仍大于M / 2并且小于M，并且大于2

} else if (node.isRoot() && node.getChildren().size() >= 2

||node.getChildren().size() >= tree.getOrder() / 2

&& node.getChildren().size() <= tree.getOrder()

&& node.getChildren().size() >= 2) {

node.getEntries().clear();

for (int i = 0; i < node.getChildren().size(); i++) {

Comparable key = node.getChildren().get(i).getEntries().get(0).getKey();

node.getEntries().add(new SimpleEntry(key, null));

if (!node.isRoot()) {

validate(node.getParent(), tree);

}

}

}

}

/\*\* 删除节点后中间节点的更新\*/

protected void updateRemove(BplusTree tree) {

validate(this, tree);

// 如果子节点数小于M / 2或者小于2，则需要合并节点

if (children.size() < tree.getOrder() / 2 || children.size() < 2) {

if (isRoot) {

// 如果是根节点并且子节点数大于等于2，OK

if (children.size() >= 2) {

return;

// 否则与子节点合并

} else {

Node root = children.get(0);

tree.setRoot(root);

root.setParent(null);

root.setRoot(true);

setEntries(null);

setChildren(null);

}

} else {

//计算前后节点

int currIdx = parent.getChildren().indexOf(this);

int prevIdx = currIdx - 1;

int nextIdx = currIdx + 1;

Node previous = null, next = null;

if (prevIdx >= 0) {

previous = parent.getChildren().get(prevIdx);

}

if (nextIdx < parent.getChildren().size()) {

next = parent.getChildren().get(nextIdx);

}

// 如果前节点子节点数大于M / 2并且大于2，则从其处借补

if (previous != null

&& previous.getChildren().size() > tree.getOrder() / 2

&& previous.getChildren().size() > 2) {

//前叶子节点末尾节点添加到首位

int idx = previous.getChildren().size() - 1;

Node borrow = previous.getChildren().get(idx);

previous.getChildren().remove(idx);

borrow.setParent(this);

children.add(0, borrow);

validate(previous, tree);

validate(this, tree);

parent.updateRemove(tree);

// 如果后节点子节点数大于M / 2并且大于2，则从其处借补

} else if (next != null

&& next.getChildren().size() > tree.getOrder() / 2

&& next.getChildren().size() > 2) {

//后叶子节点首位添加到末尾

Node borrow = next.getChildren().get(0);

next.getChildren().remove(0);

borrow.setParent(this);

children.add(borrow);

validate(next, tree);

validate(this, tree);

parent.updateRemove(tree);

// 否则需要合并节点

} else {

// 同前面节点合并

if (previous != null

&& (previous.getChildren().size() <= tree.getOrder() / 2 || previous.getChildren().size() <= 2)) {

for (int i = previous.getChildren().size() - 1; i >= 0; i--) {

Node child = previous.getChildren().get(i);

children.add(0, child);

child.setParent(this);

}

previous.setChildren(null);

previous.setEntries(null);

previous.setParent(null);

parent.getChildren().remove(previous);

validate(this, tree);

parent.updateRemove(tree);

// 同后面节点合并

} else if (next != null

&& (next.getChildren().size() <= tree.getOrder() / 2 || next.getChildren().size() <= 2)) {

for (int i = 0; i < next.getChildren().size(); i++) {

Node child = next.getChildren().get(i);

children.add(child);

child.setParent(this);

}

next.setChildren(null);

next.setEntries(null);

next.setParent(null);

parent.getChildren().remove(next);

validate(this, tree);

parent.updateRemove(tree);

}

}

}

}

}

public void remove(Comparable key, BplusTree tree){

//如果是叶子节点

if (isLeaf){

//如果不包含该关键字，则直接返回

if (!contains(key)){

return;

}

//如果既是叶子节点又是跟节点，直接删除

if (isRoot) {

remove(key);

}else {

//如果关键字数大于M / 2，直接删除

if (entries.size() > tree.getOrder() / 2 && entries.size() > 2) {

remove(key);

}else {

//如果自身关键字数小于M / 2，并且前节点关键字数大于M / 2，则从其处借补

if (previous != null

&& previous.getEntries().size() > tree.getOrder() / 2

&& previous.getEntries().size() > 2

&& previous.getParent() == parent) {

int size = previous.getEntries().size();

Entry<Comparable, Object> entry = previous.getEntries().get(size - 1);

previous.getEntries().remove(entry);

//添加到首位

entries.add(0, entry);

remove(key);

//如果自身关键字数小于M / 2，并且后节点关键字数大于M / 2，则从其处借补

}else if (next != null

&& next.getEntries().size() > tree.getOrder() / 2

&& next.getEntries().size() > 2

&& next.getParent() == parent) {

Entry<Comparable, Object> entry = next.getEntries().get(0);

next.getEntries().remove(entry);

//添加到末尾

entries.add(entry);

remove(key);

//否则需要合并叶子节点

}else {

//同前面节点合并

if (previous != null

&& (previous.getEntries().size() <= tree.getOrder() / 2 || previous.getEntries().size() <= 2)

&& previous.getParent() == parent) {

for (int i = previous.getEntries().size() - 1; i >=0; i--) {

//从末尾开始添加到首位

entries.add(0, previous.getEntries().get(i));

}

remove(key);

previous.setParent(null);

previous.setEntries(null);

parent.getChildren().remove(previous);

//更新链表

if (previous.getPrevious() != null) {

Node temp = previous;

temp.getPrevious().setNext(this);

previous = temp.getPrevious();

temp.setPrevious(null);

temp.setNext(null);

}else {

tree.setHead(this);

previous.setNext(null);

previous = null;

}

//同后面节点合并

}else if(next != null

&& (next.getEntries().size() <= tree.getOrder() / 2 || next.getEntries().size() <= 2)

&& next.getParent() == parent){

for (int i = 0; i < next.getEntries().size(); i++) {

//从首位开始添加到末尾

entries.add(next.getEntries().get(i));

}

remove(key);

next.setParent(null);

next.setEntries(null);

parent.getChildren().remove(next);

//更新链表

if (next.getNext() != null) {

Node temp = next;

temp.getNext().setPrevious(this);

next = temp.getNext();

temp.setPrevious(null);

temp.setNext(null);

}else {

next.setPrevious(null);

next = null;

}

}

}

}

parent.updateRemove(tree);

}

//如果不是叶子节点

}else {

//如果key小于等于节点最左边的key，沿第一个子节点继续搜索

if (key.compareTo(entries.get(0).getKey()) <= 0) {

children.get(0).remove(key, tree);

//如果key大于节点最右边的key，沿最后一个子节点继续搜索

}else if (key.compareTo(entries.get(entries.size()-1).getKey()) >= 0) {

children.get(children.size()-1).remove(key, tree);

//否则沿比key大的前一个子节点继续搜索

}else {

for (int i = 0; i < entries.size(); i++) {

if (entries.get(i).getKey().compareTo(key) <= 0 && entries.get(i+1).getKey().compareTo(key) > 0) {

children.get(i).remove(key, tree);

break;

}

}

}

}

}

/\*\* 判断当前节点是否包含该关键字\*/

protected boolean contains(Comparable key) {

for (Entry<Comparable, Object> entry : entries) {

if (entry.getKey().compareTo(key) == 0) {

return true;

}

}

return false;

}

/\*\* 插入到当前节点的关键字中\*/

protected void insertOrUpdate(Comparable key, Object obj){

Entry<Comparable, Object> entry = new SimpleEntry<Comparable, Object>(key, obj);

//如果关键字列表长度为0，则直接插入

if (entries.size() == 0) {

entries.add(entry);

return;

}

//否则遍历列表

for (int i = 0; i < entries.size(); i++) {

//如果该关键字键值已存在，则更新

if (entries.get(i).getKey().compareTo(key) == 0) {

entries.get(i).setValue(obj);

return;

//否则插入

}else if (entries.get(i).getKey().compareTo(key) > 0){

//插入到链首

if (i == 0) {

entries.add(0, entry);

return;

//插入到中间

}else {

entries.add(i, entry);

return;

}

}

}

//插入到末尾

entries.add(entries.size(), entry);

}

/\*\* 删除节点\*/

protected void remove(Comparable key){

int index = -1;

for (int i = 0; i < entries.size(); i++) {

if (entries.get(i).getKey().compareTo(key) == 0) {

index = i;

break;

}

}

if (index != -1) {

entries.remove(index);

}

}

public Node getPrevious() {

return previous;

}

public void setPrevious(Node previous) {

this.previous = previous;

}

public Node getNext() {

return next;

}

public void setNext(Node next) {

this.next = next;

}

public boolean isLeaf() {

return isLeaf;

}

public void setLeaf(boolean isLeaf) {

this.isLeaf = isLeaf;

}

public Node getParent() {

return parent;

}

public void setParent(Node parent) {

this.parent = parent;

}

public List<Entry<Comparable, Object>> getEntries() {

return entries;

}

public void setEntries(List<Entry<Comparable, Object>> entries) {

this.entries = entries;

}

public List<Node> getChildren() {

return children;

}

public void setChildren(List<Node> children) {

this.children = children;

}

public boolean isRoot() {

return isRoot;

}

public void setRoot(boolean isRoot) {

this.isRoot = isRoot;

}

public File getFile() {

return file;

}

public void setFile(File file) {

this.file = this.file;

}

public String toString(){

StringBuilder sb = new StringBuilder();

sb.append("isRoot: ");

sb.append(isRoot);

sb.append(", ");

sb.append("isLeaf: ");

sb.append(isLeaf);

sb.append(", ");

sb.append("keys: ");

for (Entry entry : entries){

sb.append(entry.getKey());

sb.append(", ");

}

sb.append(", ");

return sb.toString();

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 153
* 154
* 155
* 156
* 157
* 158
* 159
* 160
* 161
* 162
* 163
* 164
* 165
* 166
* 167
* 168
* 169
* 170
* 171
* 172
* 173
* 174
* 175
* 176
* 177
* 178
* 179
* 180
* 181
* 182
* 183
* 184
* 185
* 186
* 187
* 188
* 189
* 190
* 191
* 192
* 193
* 194
* 195
* 196
* 197
* 198
* 199
* 200
* 201
* 202
* 203
* 204
* 205
* 206
* 207
* 208
* 209
* 210
* 211
* 212
* 213
* 214
* 215
* 216
* 217
* 218
* 219
* 220
* 221
* 222
* 223
* 224
* 225
* 226
* 227
* 228
* 229
* 230
* 231
* 232
* 233
* 234
* 235
* 236
* 237
* 238
* 239
* 240
* 241
* 242
* 243
* 244
* 245
* 246
* 247
* 248
* 249
* 250
* 251
* 252
* 253
* 254
* 255
* 256
* 257
* 258
* 259
* 260
* 261
* 262
* 263
* 264
* 265
* 266
* 267
* 268
* 269
* 270
* 271
* 272
* 273
* 274
* 275
* 276
* 277
* 278
* 279
* 280
* 281
* 282
* 283
* 284
* 285
* 286
* 287
* 288
* 289
* 290
* 291
* 292
* 293
* 294
* 295
* 296
* 297
* 298
* 299
* 300
* 301
* 302
* 303
* 304
* 305
* 306
* 307
* 308
* 309
* 310
* 311
* 312
* 313
* 314
* 315
* 316
* 317
* 318
* 319
* 320
* 321
* 322
* 323
* 324
* 325
* 326
* 327
* 328
* 329
* 330
* 331
* 332
* 333
* 334
* 335
* 336
* 337
* 338
* 339
* 340
* 341
* 342
* 343
* 344
* 345
* 346
* 347
* 348
* 349
* 350
* 351
* 352
* 353
* 354
* 355
* 356
* 357
* 358
* 359
* 360
* 361
* 362
* 363
* 364
* 365
* 366
* 367
* 368
* 369
* 370
* 371
* 372
* 373
* 374
* 375
* 376
* 377
* 378
* 379
* 380
* 381
* 382
* 383
* 384
* 385
* 386
* 387
* 388
* 389
* 390
* 391
* 392
* 393
* 394
* 395
* 396
* 397
* 398
* 399
* 400
* 401
* 402
* 403
* 404
* 405
* 406
* 407
* 408
* 409
* 410
* 411
* 412
* 413
* 414
* 415
* 416
* 417
* 418
* 419
* 420
* 421
* 422
* 423
* 424
* 425
* 426
* 427
* 428
* 429
* 430
* 431
* 432
* 433
* 434
* 435
* 436
* 437
* 438
* 439
* 440
* 441
* 442
* 443
* 444
* 445
* 446
* 447
* 448
* 449
* 450
* 451
* 452
* 453
* 454
* 455
* 456
* 457
* 458
* 459
* 460
* 461
* 462
* 463
* 464
* 465
* 466
* 467
* 468
* 469
* 470
* 471
* 472
* 473
* 474
* 475
* 476
* 477
* 478
* 479
* 480
* 481
* 482
* 483
* 484
* 485
* 486
* 487
* 488
* 489
* 490
* 491
* 492
* 493
* 494
* 495
* 496
* 497
* 498
* 499
* 500
* 501
* 502
* 503
* 504
* 505
* 506
* 507
* 508
* 509
* 510
* 511
* 512
* 513
* 514
* 515
* 516
* 517
* 518
* 519
* 520
* 521
* 522
* 523
* 524
* 525
* 526
* 527
* 528
* 529
* 530
* 531
* 532
* 533
* 534
* 535
* 536
* 537
* 538
* 539
* 540
* 541
* 542
* 543
* 544
* 545
* 546
* 547
* 548
* 549
* 550
* 551
* 552
* 553
* 554
* 555
* 556
* 557
* 558
* 559
* 560
* 561
* 562
* 563
* 564
* 565
* 566
* 567
* 568
* 569
* 570
* 571
* 572
* 573
* 574
* 575
* 576
* 577
* 578
* 579
* 580
* 581
* 582
* 583
* 584
* 585
* 586
* 587
* 588
* 589
* 590
* 591
* 592
* 593
* 594
* 595
* 596
* 597
* 598
* 599
* 600
* 601
* 602
* 603
* 604
* 605
* 606
* 607
* 608
* 609
* 610
* 611
* 612
* 613
* 614
* 615
* 616
* 617
* 618
* 619
* 620
* 621
* 622
* 623
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110
* 111
* 112
* 113
* 114
* 115
* 116
* 117
* 118
* 119
* 120
* 121
* 122
* 123
* 124
* 125
* 126
* 127
* 128
* 129
* 130
* 131
* 132
* 133
* 134
* 135
* 136
* 137
* 138
* 139
* 140
* 141
* 142
* 143
* 144
* 145
* 146
* 147
* 148
* 149
* 150
* 151
* 152
* 153
* 154
* 155
* 156
* 157
* 158
* 159
* 160
* 161
* 162
* 163
* 164
* 165
* 166
* 167
* 168
* 169
* 170
* 171
* 172
* 173
* 174
* 175
* 176
* 177
* 178
* 179
* 180
* 181
* 182
* 183
* 184
* 185
* 186
* 187
* 188
* 189
* 190
* 191
* 192
* 193
* 194
* 195
* 196
* 197
* 198
* 199
* 200
* 201
* 202
* 203
* 204
* 205
* 206
* 207
* 208
* 209
* 210
* 211
* 212
* 213
* 214
* 215
* 216
* 217
* 218
* 219
* 220
* 221
* 222
* 223
* 224
* 225
* 226
* 227
* 228
* 229
* 230
* 231
* 232
* 233
* 234
* 235
* 236
* 237
* 238
* 239
* 240
* 241
* 242
* 243
* 244
* 245
* 246
* 247
* 248
* 249
* 250
* 251
* 252
* 253
* 254
* 255
* 256
* 257
* 258
* 259
* 260
* 261
* 262
* 263
* 264
* 265
* 266
* 267
* 268
* 269
* 270
* 271
* 272
* 273
* 274
* 275
* 276
* 277
* 278
* 279
* 280
* 281
* 282
* 283
* 284
* 285
* 286
* 287
* 288
* 289
* 290
* 291
* 292
* 293
* 294
* 295
* 296
* 297
* 298
* 299
* 300
* 301
* 302
* 303
* 304
* 305
* 306
* 307
* 308
* 309
* 310
* 311
* 312
* 313
* 314
* 315
* 316
* 317
* 318
* 319
* 320
* 321
* 322
* 323
* 324
* 325
* 326
* 327
* 328
* 329
* 330
* 331
* 332
* 333
* 334
* 335
* 336
* 337
* 338
* 339
* 340
* 341
* 342
* 343
* 344
* 345
* 346
* 347
* 348
* 349
* 350
* 351
* 352
* 353
* 354
* 355
* 356
* 357
* 358
* 359
* 360
* 361
* 362
* 363
* 364
* 365
* 366
* 367
* 368
* 369
* 370
* 371
* 372
* 373
* 374
* 375
* 376
* 377
* 378
* 379
* 380
* 381
* 382
* 383
* 384
* 385
* 386
* 387
* 388
* 389
* 390
* 391
* 392
* 393
* 394
* 395
* 396
* 397
* 398
* 399
* 400
* 401
* 402
* 403
* 404
* 405
* 406
* 407
* 408
* 409
* 410
* 411
* 412
* 413
* 414
* 415
* 416
* 417
* 418
* 419
* 420
* 421
* 422
* 423
* 424
* 425
* 426
* 427
* 428
* 429
* 430
* 431
* 432
* 433
* 434
* 435
* 436
* 437
* 438
* 439
* 440
* 441
* 442
* 443
* 444
* 445
* 446
* 447
* 448
* 449
* 450
* 451
* 452
* 453
* 454
* 455
* 456
* 457
* 458
* 459
* 460
* 461
* 462
* 463
* 464
* 465
* 466
* 467
* 468
* 469
* 470
* 471
* 472
* 473
* 474
* 475
* 476
* 477
* 478
* 479
* 480
* 481
* 482
* 483
* 484
* 485
* 486
* 487
* 488
* 489
* 490
* 491
* 492
* 493
* 494
* 495
* 496
* 497
* 498
* 499
* 500
* 501
* 502
* 503
* 504
* 505
* 506
* 507
* 508
* 509
* 510
* 511
* 512
* 513
* 514
* 515
* 516
* 517
* 518
* 519
* 520
* 521
* 522
* 523
* 524
* 525
* 526
* 527
* 528
* 529
* 530
* 531
* 532
* 533
* 534
* 535
* 536
* 537
* 538
* 539
* 540
* 541
* 542
* 543
* 544
* 545
* 546
* 547
* 548
* 549
* 550
* 551
* 552
* 553
* 554
* 555
* 556
* 557
* 558
* 559
* 560
* 561
* 562
* 563
* 564
* 565
* 566
* 567
* 568
* 569
* 570
* 571
* 572
* 573
* 574
* 575
* 576
* 577
* 578
* 579
* 580
* 581
* 582
* 583
* 584
* 585
* 586
* 587
* 588
* 589
* 590
* 591
* 592
* 593
* 594
* 595
* 596
* 597
* 598
* 599
* 600
* 601
* 602
* 603
* 604
* 605
* 606
* 607
* 608
* 609
* 610
* 611
* 612
* 613
* 614
* 615
* 616
* 617
* 618
* 619
* 620
* 621
* 622
* 623

生成的文件，截个图：

