**实验二 数据库索引及查询算法实现**

1. **实验目的：**掌握B树索引查找算法，多路归并排序算法，并用高级语言实现
2. **实验环境：**

**操作系统：windows**

**编程语言：c++**

**3、实验内容及要求：**

选择熟悉的高级语言设计实现归并排序和B树索引。

具体要求如下：

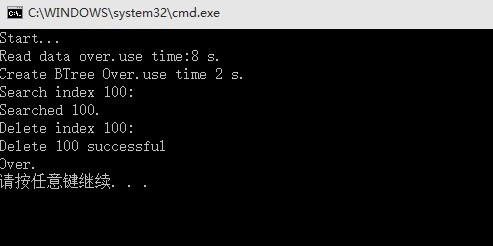
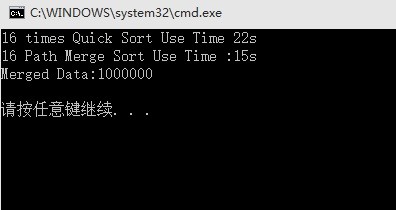
* + - 1. 随机生成具有1,000,000条记录的文本文件，每条记录的长度为16字节。

|  |  |
| --- | --- |
| 属性A(4字节整数) | 属性B（12字节字符串） |

* + - 1. 其中包含两个属性A和B。A为4字节整型， B为12字节字符串，属性值A随机生成，属性值B自己定义并填充。
      2. 以属性A为键值，实现B树索引。完成索引的插入，删除和查找操作。
      3. 针对属性A，用高级语言实现多路归并排序算法。
      4. 用于外部归并排序的内存空间不大于1MB。

1. **实验内容**

截屏给出实验结果



算法流程：

由于共有1000000个元组，每个元组128B，因此总大小为128M，又内存限制为8M，则至少要分成128M/8M=16组。为了尽可能的减少I/O次数, 因此采用16路归并的方法；为了减少归并的时间，采用败者树，在

O(lgn)的时间复杂度内选出最小的Ａ属性元组，这种方法从划分到归并，读入两次，写出两次，最少的I/O次数是4\*1000000.

程序代码：

1、生成1000000个128字节的元组代码：

#include <iostream> #include <fstream> #include <ctime> #include <cstdlib> using namespace std;

const int N = 1000000;

string RandomChineseCharacters()

{

//srand( (unsigned)time(NULL));

int high = 0xd7 - 0xc1;//16-55区汉字int low= 0xfe - 0xa1;

int high\_zn ; int low\_zn; char name[3]; name[2] = '\0';

string s;

for(int i = 0; i < 60; i++)

{

high\_zn = rand()%high + 0xc1; low\_zn = rand()%low + 0xa1; name[0]=high\_zn; name[1]=low\_zn; s.append(name);

}

return s;

}

int main()

{

ofstream ofp ("example2.txt"); if (!ofp.is\_open())

{

cout<<"can't open file!"<<endl; return 0;

}

srand( (unsigned)time(NULL)); int t1 = 0,t2 = 0;//4+

string name;

for(int i = 1;i <= N;i++)

{

t1 = rand(); t2 = i;

name = RandomChineseCharacters();

//cout<<name <<sizeof(t1)<<sizeof(t2)<<name.length()<<endl;

ofp<<t1<<" "<<t2<<" "<<name<<endl;

}

return 0;

}

2、16组快速排序和16路归并排序的代码

// Extenal\_Merge\_Sort.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h" #include <iostream> #include <fstream> #include <ctime> #include <cstdlib> #include <string.h> #include <assert.h> using namespace std;

typedef struct Data

{

int data; int num;

char name[121];

} Data;

//利用败者树

const int N = 1000000;//数据总量const int FILE\_NUM = 16;//文件个数

const int MAX\_PART = 62500;//每一个文件大小FILE \*fpreads[FILE\_NUM];

const int MIN = -1; //最小值,必须比要排序数字的最小值要小，否则出错

const int MAX = 9999999; //最大值,必须比要排序数字的最大值要大，否则出错

int result = 0;

int cmp(const void\* a, const void \*b)

{

if ((\*(Data\*)a).data > (\*(Data \*)b).data) return 1;

else if ((\*(Data\*)a).data < (\*(Data \*)b).data) return -1;

else

return 0;

}

//从unsort\_data.txt中读取数据

int read\_data(FILE \*fp, Data \*array, int N)

{

int length = 0;

for (int i = 0; i < MAX\_PART && (EOF !=

fscanf\_s(fp, "%d %d %s\n", &array[i].data, &array[i].num,

array[i].name,\_countof(array[i].name))); i++)

{

length++;

}

return length;

}

//打开data0.txt - data9.txt这10个文件FILE\* open\_file(int count, char \*mode)

{

FILE \*fpwrite = NULL; char filename[20]; memset(filename, 0, 20);

sprintf\_s(filename,20, "data%d.txt", count); fopen\_s(&fpwrite,filename, mode); assert(fpwrite != NULL);

return fpwrite;

}

//向data0.txt - data9.txt这10个文件写入排好序的数据void write\_data(Data \*array, int N, int count)

{

FILE \*fpwrite = open\_file(count, "w"); int i = 0;

for (i = 0; i < N; i++)

{

fprintf(fpwrite, "%d %d %s\n", array[i].data, array[i].num, array[i].name);

}

fprintf(fpwrite, "%d %d %s\n", MAX, array[i - 1].num, array[i-1].name);

//在每个文件最后写入一个最大值，表示文件结束fclose(fpwrite);

}

//内部排序，调用16次快速排序，产生data0.txt - data16.txt这10个有序文件

void interior\_sort(void)

{

clock\_t begin = clock(); FILE \*fpread = NULL;

fopen\_s(&fpread,"unsort\_data.txt", "r"); assert(fpread != NULL);

int count = 0;

Data \*array = new Data[MAX\_PART]; assert(array != NULL);

while (1)

{

memset(array, 0, sizeof(Data)\* MAX\_PART);

int length = read\_data(fpread, array, MAX\_PART); if (length == 0)

{

break;

}

qsort(array, length, sizeof(Data), cmp); write\_data(array, length, count); count++;

}

delete[] array; fclose(fpread); clock\_t end = clock();

cout << "16 times Quick Sort Use Time " << (end - begin) / CLK\_TCK

<< "s" << endl;

}

//调整

void adjust(int ls[], Data data[], int s)

{

int t = (s + FILE\_NUM) / 2; while (t)

{

if (data[s].data > data[ls[t]].data)

{

int temp = s; s = ls[t]; ls[t] = temp;

}

t /= 2;

}

ls[0] = s;

}

void create\_loser\_tree(int ls[], Data data[])

{

data[FILE\_NUM].data = MIN;

for (int i = 0; i < FILE\_NUM; i++)

{

ls[i] = FILE\_NUM;

}

for (int i = FILE\_NUM - 1; i >= 0; i--)

{

adjust(ls, data, i);

}

}

void merge\_sort\_by\_losertree()

{

clock\_t begin = clock();

FILE \*fpreads[FILE\_NUM]; //10个文件的描述符

Data data[FILE\_NUM + 1]; //10个文件的10个当前最小数据int ls[FILE\_NUM]; //存放败者索引的节点

int index;

FILE \*fpwrite = NULL; fopen\_s(&fpwrite,"sort\_data\_by\_losertree.txt", "w"); assert(fpwrite != NULL);

for (int i = 0; i < FILE\_NUM; i++)

{

fpreads[i] = open\_file(i, "r");

}

for (int i = 0; i < FILE\_NUM; i++)

{

fscanf\_s(fpreads[i], "%d %d %s\n", &data[i].data, &data[i].num, data[i].name,\_countof(data[i].name));

}

create\_loser\_tree(ls, data); //创建败者树while (data[ls[0]].data != MAX)

{

index = ls[0];

fprintf(fpwrite, "%d %d %s\n", data[index].data, data[index].num, data[index].name);

result++;//测试数据是否全部读完。

fscanf\_s(fpreads[index], "%d %d %s\n", &data[index].data, &data[index].num, data[index].name, \_countof(data[index].name));

adjust(ls, data, index);

}

for (int i = 0; i < FILE\_NUM; i++)

{

fclose(fpreads[i]);

}

fclose(fpwrite); clock\_t end = clock();

cout << "16 Path Merge Sort Use Time :" << (end - begin) / CLK\_TCK

<< "s" << endl;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

interior\_sort(); merge\_sort\_by\_losertree();

cout << "Merged Data:" << result << endl; getchar();

return 0;

}

3、B树索引的建立、查找、删除的代码

#include "stdafx.h" #include <stdio.h> #include <stdlib.h> #include <assert.h> #include <ctime>

/\*\*

@brief the degree of btree

key per node: [M-1, 2M-1]

child per node: [M, 2M]

\*/

#define M 2

#define MaxSize 1000001 typedef struct btree\_node {

int k[2 \* M - 1];

struct btree\_node \*p[2 \* M]; int num;

bool is\_leaf;

} btree\_node;

/\*\*

@brief allocate a new btree node

default: is\_leaf == true

\*

@return pointer of new node

\*/

btree\_node \*btree\_node\_new();

/\*\*

@brief create a btree root

\*

@return pointer of btree root

\*/

btree\_node \*btree\_create();

/\*\*

@brief split child if num of key in child exceed 2M-1

\*

@param parent: parent of child

@param pos: p[pos] points to child

@param child: the node to be splited

\*

@return

\*/

int btree\_split\_child(btree\_node \*parent, int pos, btree\_node \*child);

/\*\*

@brief insert a value into btree

the num of key in node less than 2M-1

\*

@param node: tree root

@param target: target to insert

\*/

void btree\_insert\_nonfull(btree\_node \*node, int target);

/\*\*

@brief insert a value into btree

\*

@param root: tree root

@param target: target to insert

\*

@return: new root of tree

\*/

btree\_node\* btree\_insert(btree\_node \*root, int target);

/\*\*

@brief merge y, z and root->k[pos] to left

this appens while y and z both have M-1 keys

\*

@param root: parent node

@param pos: postion of y

@param y: left node to merge

@param z: right node to merge

\*/

void btree\_merge\_child(btree\_node \*root, int pos, btree\_node \*y, btree\_node

\*z);

/\*\*

@brief delete a vlue from btree

\*

@param root: btree root

@param target: target to delete

\*

@return: new root of tree

\*/

btree\_node \*btree\_delete(btree\_node \*root, int target);

/\*\*

@brief delete a vlue from btree

root has at least M keys

\*

@param root: btree root

@param target: target to delete

\*

@return

\*/

void btree\_delete\_nonone(btree\_node \*root, int target);

/\*\*

@brief find the rightmost value

\*

@param root: root of tree

\*

@return: the rightmost value

\*/

int btree\_search\_predecessor(btree\_node \*root);

/\*\*

@brief find the leftmost value

\*

@param root: root of tree

\*

@return: the leftmost value

\*/

int btree\_search\_successor(btree\_node \*root);

/\*\*

@brief shift a value from z to y

\*

@param root: btree root

@param pos: position of y

@param y: left node

@param z: right node

\*/

void btree\_shift\_to\_left\_child(btree\_node \*root, int pos, btree\_node \*y, btree\_node \*z);

/\*\*

@brief shift a value from z to y

\*

@param root: btree root

@param pos: position of y

@param y: left node

@param z: right node

\*/

void btree\_shift\_to\_right\_child(btree\_node \*root, int pos, btree\_node \*y, btree\_node \*z);

/\*\*

@brief inorder traverse the btree

\*

@param root: root of treee

\*/

void btree\_inorder\_print(btree\_node \*root);

/\*\*

@brief level print the btree

\*

@param root: root of tree

\*/

void btree\_level\_display(btree\_node \*root);

btree\_node \*btree\_node\_new()

{

btree\_node \*node = (btree\_node \*)malloc(sizeof(btree\_node)); if (NULL == node) {

return NULL;

}

for (int i = 0; i < 2 \* M - 1; i++) { node->k[i] = 0;

}

for (int i = 0; i < 2 \* M; i++) { node->p[i] = NULL;

}

node->num = 0; node->is\_leaf = true;

}

btree\_node \*btree\_create()

{

btree\_node \*node = btree\_node\_new(); if (NULL == node) {

return NULL;

}

return node;

}

int btree\_split\_child(btree\_node \*parent, int pos, btree\_node \*child)

{

btree\_node \*new\_child = btree\_node\_new(); if (NULL == new\_child) {

return -1;

}

new\_child->is\_leaf = child->is\_leaf; new\_child->num = M - 1;

for (int i = 0; i < M - 1; i++) { new\_child->k[i] = child->k[i + M];

}

if (false == new\_child->is\_leaf) { for (int i = 0; i < M; i++) {

new\_child->p[i] = child->p[i + M];

}

}

child->num = M - 1;

for (int i = parent->num; i > pos; i--) { parent->p[i + 1] = parent->p[i];

}

parent->p[pos + 1] = new\_child;

for (int i = parent->num - 1; i >= pos; i--) { parent->k[i + 1] = parent->k[i];

}

parent->k[pos] = child->k[M - 1];

parent->num += 1;

}

void btree\_insert\_nonfull(btree\_node \*node, int target)

{

if (1 == node->is\_leaf) { int pos = node->num;

while (pos >= 1 && target < node->k[pos - 1]) { node->k[pos] = node->k[pos - 1];

pos--;

}

node->k[pos] = target; node->num += 1;

}

else {

int pos = node->num;

while (pos > 0 && target < node->k[pos - 1]) { pos--;

}

if (2 \* M - 1 == node->p[pos]->num) { btree\_split\_child(node, pos, node->p[pos]); if (target > node->k[pos]) {

pos++;

}

}

btree\_insert\_nonfull(node->p[pos], target);

}

}

btree\_node\* btree\_insert(btree\_node \*root, int target)

{

if (NULL == root) { return NULL;

}

if (2 \* M - 1 == root->num) {

btree\_node \*node = btree\_node\_new(); if (NULL == node) {

return root;

}

node->is\_leaf = 0; node->p[0] = root;

btree\_split\_child(node, 0, root); btree\_insert\_nonfull(node, target); return node;

}

else {

btree\_insert\_nonfull(root, target); return root;

}

}

void btree\_merge\_child(btree\_node \*root, int pos, btree\_node \*y, btree\_node

\*z)

{

y->num = 2 \* M - 1;

for (int i = M; i < 2 \* M - 1; i++) { y->k[i] = z->k[i - M];

}

y->k[M - 1] = root->k[pos];

if (false == z->is\_leaf) {

for (int i = M; i < 2 \* M; i++) { y->p[i] = z->p[i - M];

}

}

for (int j = pos + 1; j < root->num; j++) { root->k[j - 1] = root->k[j];

root->p[j] = root->p[j + 1];

}

root->num -= 1; free(z);

}

btree\_node \*btree\_delete(btree\_node \*root, int target)

{

if (1 == root->num) { btree\_node \*y = root->p[0]; btree\_node \*z = root->p[1];

if (NULL != y && NULL != z &&

M - 1 == y->num && M - 1 == z->num) { btree\_merge\_child(root, 0, y, z); free(root);

btree\_delete\_nonone(y, target); return y;

}

else {

btree\_delete\_nonone(root, target); return root;

}

}

else {

btree\_delete\_nonone(root, target); return root;

}

}

void btree\_delete\_nonone(btree\_node \*root, int target)

{

if (true == root->is\_leaf) { int i = 0;

while (i < root->num && target > root->k[i]) i++; if (target == root->k[i]) {

for (int j = i + 1; j < 2 \* M - 1; j++) { root->k[j - 1] = root->k[j];

}

root->num -= 1;

}

else {

printf("target not found\n");

}

}

else {

int i = 0;

btree\_node \*y = NULL, \*z = NULL;

while (i < root->num && target > root->k[i]) i++; if (i < root->num && target == root->k[i]) {

y = root->p[i];

z = root->p[i + 1];

if (y->num > M - 1) {

int pre = btree\_search\_predecessor(y); root->k[i] = pre; btree\_delete\_nonone(y, pre);

}

else if (z->num > M - 1) {

int next = btree\_search\_successor(z); root->k[i] = next; btree\_delete\_nonone(z, next);

}

else {

btree\_merge\_child(root, i, y, z); btree\_delete(y, target);

}

}

else {

y = root->p[i];

if (i < root->num) { z = root->p[i + 1];

}

btree\_node \*p = NULL; if (i > 0) {

p = root->p[i - 1];

}

if (y->num == M - 1) {

if (i > 0 && p->num > M - 1) { btree\_shift\_to\_right\_child(root, i - 1, p, y);

}

else if (i < root->num && z->num > M - 1) {

btree\_shift\_to\_left\_child(root, i, y, z);

}

else if (i > 0) {

btree\_merge\_child(root, i - 1, p, y); // note y = p;

}

else {

btree\_merge\_child(root, i, y, z);

}

btree\_delete\_nonone(y, target);

}

else {

btree\_delete\_nonone(y, target);

}

}

}

}

int btree\_search\_predecessor(btree\_node \*root)

{

btree\_node \*y = root;

while (false == y->is\_leaf) { y = y->p[y->num];

}

return y->k[y->num - 1];

}

int btree\_search\_successor(btree\_node \*root)

{

btree\_node \*z = root;

while (false == z->is\_leaf) { z = z->p[0];

}

return z->k[0];

}

void btree\_shift\_to\_right\_child(btree\_node \*root, int pos, btree\_node \*y, btree\_node \*z)

{

z->num += 1;

for (int i = z->num - 1; i > 0; i--) {

z->k[i] = z->k[i - 1];

}

z->k[0] = root->k[pos];

root->k[pos] = y->k[y->num - 1];

if (false == z->is\_leaf) {

for (int i = z->num; i > 0; i--) {

z->p[i] = z->p[i - 1];

}

z->p[0] = y->p[y->num];

}

y->num -= 1;

}

void btree\_shift\_to\_left\_child(btree\_node \*root, int pos,

btree\_node \*y, btree\_node \*z)

{

y->num += 1;

>k[y->num - 1] = root->k[pos];

root->k[pos] = z->k[0];

for (int j = 1; j < z->num; j++) { z->k[j - 1] = z->k[j];

}

if (false == z->is\_leaf) {

y->p[y->num] = z->p[0];

for (int j = 1; j <= z->num; j++) { z->p[j - 1] = z->p[j];

}

}

>num -= 1;

}

void btree\_inorder\_print(FILE\* fp, btree\_node \*root)

{

if (NULL != root) { btree\_inorder\_print(fp,root->p[0]); for (int i = 0; i < root->num; i++) {

fprintf\_s(fp,"%d ", root->k[i]); btree\_inorder\_print(fp,root->p[i + 1]);

}

}

}

void btree\_level\_display(btree\_node \*root)

{

// just for simplicty, can't exceed 200 nodes in the tree btree\_node \*queue[MaxSize\*2] = { NULL };

int front = 0; int rear = 0;

queue[rear++] = root;

while (front < rear) {

btree\_node \*node = queue[front++];

printf("[");

for (int i = 0; i < node->num; i++) { printf("%d ", node->k[i]);

}

printf("]");

for (int i = 0; i <= node->num; i++) { if (NULL != node->p[i]) {

queue[rear++] = node->p[i];

}

}

}

printf("\n");

}

int ReadData(int \* arr)//return length

{

FILE\* fp = NULL;

fopen\_s(&fp,"unsort\_data.txt", "r"); assert(fp != NULL);

int a = 0, length = 0; char temp[121];

for (int i = 0; (EOF != fscanf\_s(fp, "%d %d %s\n", &arr[i], &a, temp,\_countof(temp))); i++)

{

length++;

}

fclose(fp); return length;

}

int main()

{

printf("Start...\n");

int \*arr = new int[MaxSize]; clock\_t t = clock();

int length = ReadData(arr);

printf("Read data over.use time:%d s.\n", (clock() - t) / CLK\_TCK); t = clock();

FILE \* fp = NULL;

fopen\_s(&fp, "BTree.txt", "w"); assert(fp != NULL);

btree\_node \*root = btree\_create(); for (int i = 0; i < length; i++) {

root = btree\_insert(root, arr[i]);

}

printf("Create BTree Over.use time %d s.\n", (clock() - t) / CLK\_TCK); btree\_inorder\_print(fp,root);

return 0;

}

对实验结果的分析：

实验结果表明，分成16个组后，快速排序用时22秒，而对于归并排序， 每个文件用时15秒，在128M的文件大小下表现优异。且未见遗漏或异常。