**数据结构实验报告**

姓名：郭宇瑶 学号：U201717031 班级：软工1704班

**实验四 桶式排序、基数排序和堆栈、中缀表达式转化等问题的求解**

一、实验描述

1．Radix Sort。实现桶式排序和基于桶式排序的基数排序。在基数B，数组长度n和最大元素值m中，对排序时间影响最大的是哪一个？元素在未排序数组中的顺序是否对时间复杂度有影响？设计试验证明你的想法

（1）桶式排序：桶排序(Bucket Sort)假设输入数据服从均匀分布，然后将输入数据均匀地分配到有限数量的桶中，然后对每个桶再分别排序，比如对每个桶再使用插入排序算法，最后将每个桶中的数据有序的组合起来。假设输入是由一个随机过程生成，该过程将元素均匀的分布在一个区间[a,b]上，在[a,b]之间放置一定数量的桶，由于桶排序和计数排序一样均对输入的数据进行了某些假设限制，因此比一般的基于比较的排序算法复杂度低。

（2）基数排序：基数排序是一种借助多关键字排序的思想对单逻辑关键字进行排序的方法。它是一种稳定的排序算法。多关键字排序中有两种方法：最高位优先法(MSD)和最低位优先法（LSD）。通常用于对数的排序选择的是最低位优先法，即先对最次位关键字进行排序，再对高一位的关键字进行排序，以此类推。

2．Stack。用C语言设计堆栈，并实现中缀表达式到后缀表达式的转换。

（1）C语言实现堆栈：由栈顶、栈底和栈的大小三个元素决定，实现pop和push:栈顶指针移动，栈的大小改变。

（2）中缀到后缀的转换：

此方法需要遵循几个规则（Rule）：

R1）如果读入操作数，则直接放入输出字符串；

R2）如果读入一般运算符如+-\*/，则放入堆栈，但是放入堆栈之前必须要检查栈顶，并确定栈顶运算符的优先级比放入的运算符的优先级低；如果放入的优先级较低或两者相等的话，则需要将栈顶的运算符先放入输出字符串， 然后再将刚读入的运算符压栈；

R3）如果读入(，因为左括号优先级最高，因此放入栈中，但是注意，当左括号放入栈中后，则优先级最低；

R4）如果读入），则将栈中运算符取出放入输出字符串，直到取出（为止，注意：（）不输出到输出字符串；

R5）顺序读完表达式，如果栈中还有操作符，则弹出，并放入输出字符串；

二、实验设计

1．Radix Sort。

（1）桶式排序：

1.初始化装入连续区间元素的n个桶，每个桶用来装一段区间中的元素。

2.遍历待排序的数据，将其映射到对应的桶中，保证每个桶中的元素都在同一个区间范围中。

3.对每个桶进行排序，最终将所有桶中排好序的元素连起来。

用例：ElementType data[] = {9, 6, 3, 2, 7, 7, 1, 4, 1, 0, 3, 9, 1, 1};

（2）基数排序：

根据关键字中各位的值，通过对排序的N个元素进行若干趟“分配”与“收集”来实现排序的。

1. 确定最大位数；
2. 根据序列的个位数的数字来进行分类，将其分到指定的桶中。例如：R[0] = 50，个位数上是0，将这个数存入编号为0的桶中。
3. 分类后，从各个桶中，将这些数按照从编号0到编号9的顺序依次将所有数取出来。这时，得到的序列就是个位数上呈递增趋势的序列。
4. 接下来对十位数、百位数也按照这种方法进行排序，最后就能得到排序完成的序列。

用例：ElementType data[] = {11, 245, 895, 658, 321, 852, 147, 458, 469, 159, 347, 28};

2.栈和表达式转化

1.链表实现栈，两个指针表示栈顶和栈底，一个int型数据记录栈的大小，实现pop、push等功能：记录数据，改变栈顶指针和栈的大小；

2.读取表达式，对字符进行判断，按照规则进行压栈、出栈、输出等操作：数字直接压栈，符号对栈顶元素进行判断，右括号在栈中匹配左括号等。

用例：1+2\*3-（16/4+5）+4

期望输出：1 2 3 \* + 16 4 / 5 + - 4 +

三、实验实现过程

1．Radix Sort。

（1）桶式排序：

#include <stdio.h>

#include <malloc.h>

#define MAX 10

#define ElementType int

#define Error(str) printf("\n\t error: %s \n",str)

struct Node;

typedef struct Node \*Node;

Node \*initBuckets();

void bucketSort(Node\* buckets, ElementType data, int index);

void radixSort(Node\* buckets, ElementType\* data, int size);

void bucketsToData(Node\* buckets, ElementType \*data);

void printBuckets(Node\* data);

void printArray(ElementType data[], int size);

Node makeEmpty();

struct Node

{

int value;

Node next;

};

Node \*initBuckets()

{

Node\* buckets;

int i;

buckets = (Node\*)malloc(MAX \* sizeof(Node));

if(!buckets)

{

Error("out of space, from func bucketSort!");

return NULL;

}

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

buckets[i] = makeEmpty();

return buckets;

}

//allocate the memory for the node and make it empty with evaluation of next

Node makeEmpty()

{

Node temp;

temp = (Node)malloc(sizeof(struct Node));

if(!temp)

{

Error("out of space, from func makeEmpty!");

return NULL;

}

temp->next = NULL;

return temp;

}

void bucketsToData(Node\* buckets, ElementType \*data)

{

int i;

int j;

Node temp;

i = 0;

j = 0;

while(i< MAX) // and now, we update the data array from buckets

{

temp = buckets[i]->next;

while(temp)

{

data[j++] = temp->value;

temp = temp->next;

}

i++;

}

// updating over

}

// details of bucketSort for the input array data with size

void bucketSort(Node\* buckets, ElementType data, int index)

{

Node temp;

temp = buckets[index];

while(temp->next)

temp = temp->next;

temp->next = makeEmpty();

temp->next->value = data;

}

int main()

{

int size;

int i;

Node\* buckets;

ElementType data[] = {9, 6, 3, 2, 7, 7, 1, 4, 1, 0, 3, 9, 1, 1};

printf("\n\t====== test for bucket sorting towards the data array =====\n");

printf("\n\t=== the initial array is as follows ===\n");

size = 14;

printArray(data, size);

buckets = initBuckets();

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

bucketSort(buckets, data[i], data[i]);

bucketsToData(buckets, data);

printf("\n\t=== the buckets array is as follows ===\n");

printArray(data, size);

return 0;

}

void printBuckets(Node\* buckets)

{

int i;

Node node;

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

{

if(!buckets[i]->next)

continue;

for(node = buckets[i]->next; node != NULL; node = node->next)

printf("\n\t buckets[%d] = %d", i, node->value);

}

printf("\n");

}

void printArray(ElementType data[], int size)

{

int i;

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

printf("\n\t data[%d] = %d", i, data[i]);

printf("\n\n");

}

（2）基数排序：

#define Round 3

ElementType singleBit(ElementType value, int bit)

{ int i;

i = 1;

while(i++ < bit)

value = value / 10;

return value % 10;

}

//free the memory the buckets own

void clearBuckets(Node\* buckets)

{

int i;

Node temp;

Node tempTwo;

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

{

temp = buckets[i]->next;

buckets[i]->next = NULL;

while(temp)

{

tempTwo = temp->next;

free(temp);

temp = NULL;

temp = tempTwo;

}

}

}

// proceeding the radix sorting for the array

void radixSort(Node\* buckets, ElementType\* data, int size)

{

int i;

int j;

for(i = 1; i <= Round; i++)

{

for(j=0; j<size; j++)

bucketSort(buckets, data[j], singleBit(data[j],i));

// coducting bucket sorting for data array over

bucketsToData(buckets, data);// and now, we update the data array from buckets

clearBuckets(buckets);

}

}

int main()

{

int size;

Node\* buckets;

ElementType data[] = {11, 245, 895, 658, 321, 852, 147, 458, 469, 159, 347, 28};

printf("\n\t====== test for radix sorting towards the data array ======\n");

printf("\n\t=== the initial array is as follows ===\n");

size = 12;

printArray(data, size);

buckets = initBuckets();

printf("\n\t=== the buckets array is as follows ===\n");

radixSort(buckets, data, size);

printArray(data, size);

return 0;

}

2.链表实现栈并转化中缀表达式为后缀表达式

#include <stdio.h>

#include <stdlib.h>

#define STACK\_INIT\_SIZE 20

#define STACKINCREMENT 10

typedef char ElemType;

typedef struct

{

ElemType \*base;

ElemType \*top;

int stackSize;

}sqStack;

InitStack(sqStack \*s)

{

s->base = (ElemType \*)malloc(STACK\_INIT\_SIZE \* sizeof(ElemType));

if( !s->base )

exit(0);

s->top = s->base;

s->stackSize = STACK\_INIT\_SIZE;

}

Push(sqStack \*s, ElemType e)

{

// If the stack is full, add space.

if( s->top - s->base >= s->stackSize )

{

s->base = (ElemType \*)realloc(s->base, (s->stackSize + STACKINCREMENT) \* sizeof(ElemType));

if( !s->base )

exit(0);

s->top = s->base + s->stackSize;

s->stackSize = s->stackSize + STACKINCREMENT;

}

\*(s->top) = e; // save the data

s->top++;

}

Pop(sqStack \*s, ElemType \*e)

{

if( s->top == s->base )

return;

\*e = \*--(s->top); // pop the top element of the stack and modify the stack top pointer

}

int StackLen(sqStack s)

{

return (s.top - s.base);

}

int main()

{

sqStack s;

char c, e;

InitStack( &s );

printf("请输入中缀表达式，以#作为结束标志：");

scanf("%c", &c);

while( c != '#' )

{

while( c>='0' && c<='9' )

{

printf("%c", c);

scanf("%c", &c);

if( c<'0' || c>'9' )

{

printf(" ");

}

}

if( ')' == c )

{

Pop(&s, &e);

while( '(' != e )

{

printf("%c ", e);

Pop(&s, &e);

}

}

else if( '+'==c || '-'==c )

{

if( !StackLen(s) )

{

Push(&s, c);

}

else

{

do

{

Pop(&s, &e);

if( '(' == e )

{

Push(&s, e);

}

else

{

printf("%c ", e);

}

}while( StackLen(s) && '('!=e );

Push(&s, c);

}

}

else if( '\*'==c || '/'==c || '('==c )

{

Push(&s, c);

}

else if( '#'== c )

{

break;

}

else

{

printf("\n出错：输入格式错误！\n");

return -1;

}

scanf("%c", &c);

}

while( StackLen(s) )

{

Pop(&s, &e);

printf("%c ", e);

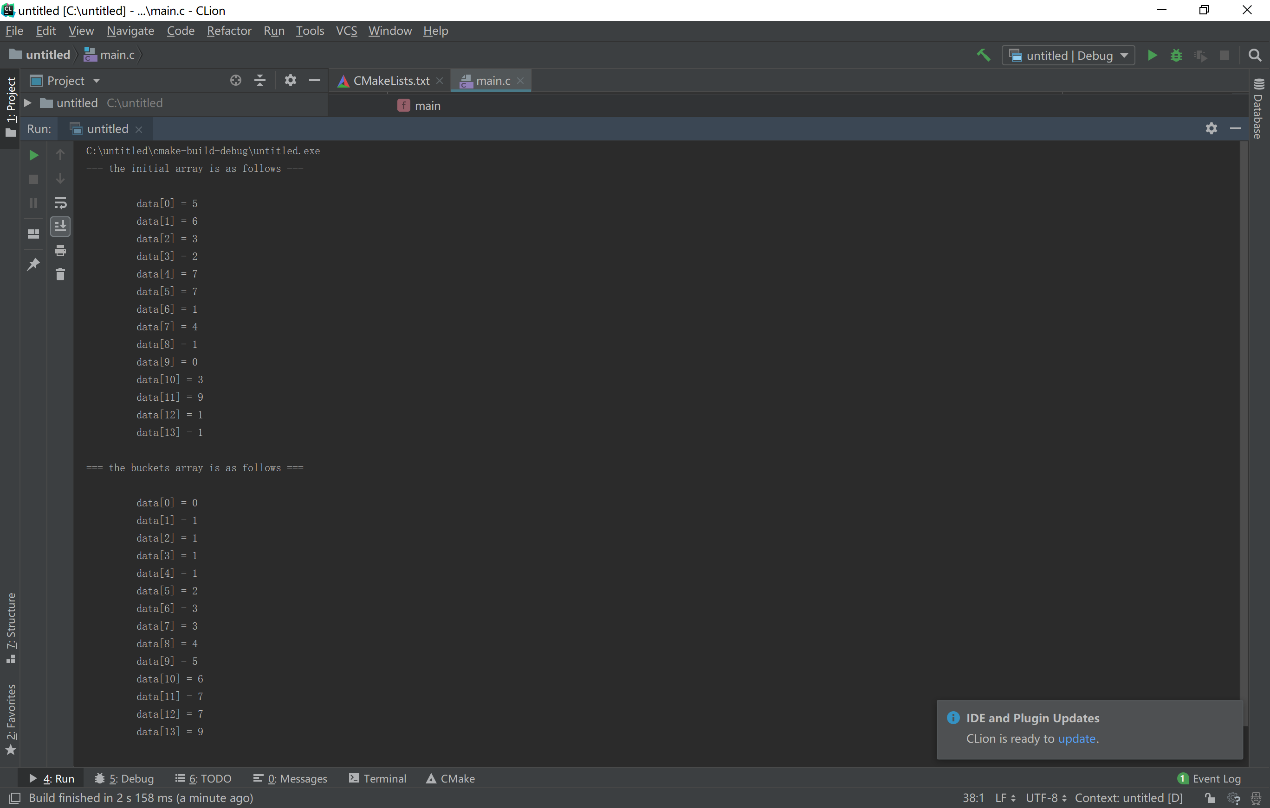
}

return 0;

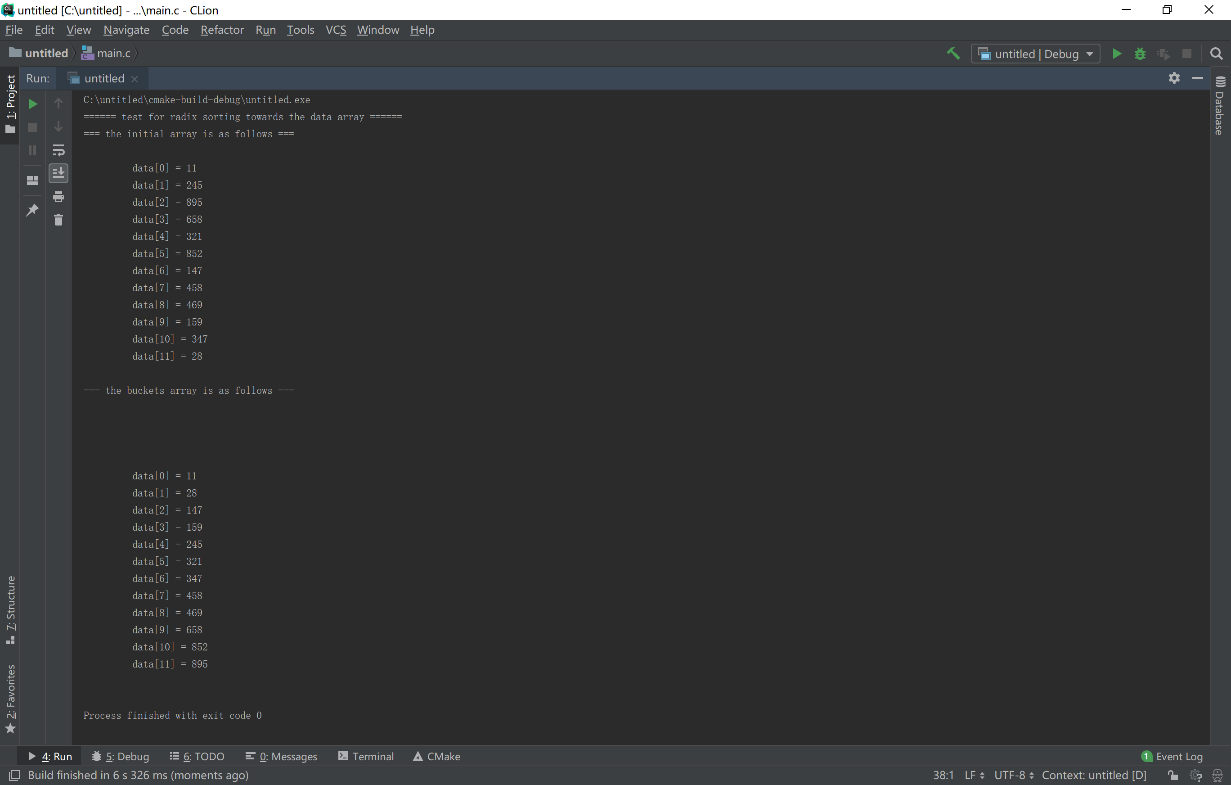
}

四、实验结果

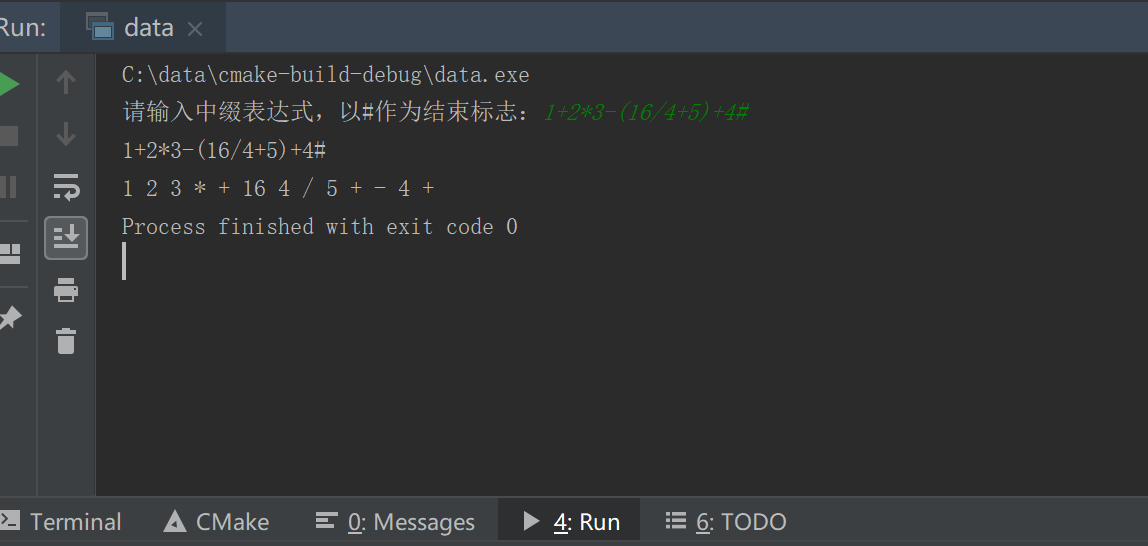
1.bucket sort



2.radix sort



3.栈和表达式转化



五、实验分析与结论

1.桶排序

桶排序的性能

时间消耗包括两部分一部分为初始化桶，连接排好序的桶，其时间复杂度为O(m) 一般有m<n (m个桶)

另一部分为对桶中的元素进行排序，这部分的复杂度，通过代码中的for和while循环直接看不太容易，这样考虑：每个桶里面有ni个元素，对ni个元素进行插入排序的耗时为O(ni^2)。

于是T(n)=O(n)+∑O(ni^2)，平均意义下认为ni=n/m，于是有T(n)=O(n)+n\*O((n/m)^2)=O(n^2/m)+O(n)

当n=m时，T(n)=O(n)+O(n)=O(n)

对于每个桶采用其他的排序算法：m个桶，每个桶中的元素平均假设有n/m个，在上面进行基于比较的排序，复杂度不会低于n\*O(n/m\*lg(n/m))，平均意义下每个桶中的元素有n/m个，O(m \* n/m \*lg(n/m) = O(n\*lg(n/m))，所以总的时间复杂度为T(n)=O(n+n\*lg(n/m))

当m=n时时间复杂度为O(n)，此时和计数排序一样，桶数量越多，时间效率越高，然而桶数量越多占用空间也就越大。

总结：桶排序的平均时间复杂度为线性的O(N+C)，其中C=N\*(logN-logM)。如果相对于同样的N，桶数量M越大，其效率越高，最好的时间复杂度达到O(N)。当然桶排序的空间复杂度为O(N+M)，如果输入数据非常庞大，而桶的数量也非常多，则空间代价无疑是昂贵的。此外，桶排序是稳定的。

2.基数排序

设待排序列为n个记录，d个关键码，关键码的取值范围为radix，则进行链式基数排序的时间复杂度为O(d(n+radix))，其中，一趟分配时间复杂度为O(n)，一趟收集时间复杂度为O(radix)，共进行d趟分配和收集。 空间效率：需要2\*radix个指向队列的辅助空间，以及用于静态链表的n个指针。

3. 在基数B，数组长度n和最大元素值m中，对排序时间影响最大的是哪一个？元素在未排序 数组中的顺序是否对时间复杂度有影响？

时间复杂度为O(d(n+m)),d由最大元素值决定，所以对排序时间影响最大的是最大元素值m。元素的顺序对时间复杂度没有影响。

4.中缀表达式转后缀表达式

易知时间复杂度为O(n)。