数据结构实验报告

姓名： 杨家玺 学号：U201717007 班级：软工1703班

2018. 12. 08

实验四 基于桶式排序的基数排序的影响因素、后缀表达式的转换

一、实验描述

1．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

2．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

二、实验环境

1．开发环境： OS X
2．IDE： VSCode
3．编译器： Clang 9.1.0 Apple LLVM
4．C 标准： C11

三、问题分析

1．排序与分析

桶式排序与基数排序的原理不再赘述，下文主要针对影响基数排序性能的三个变量进行讨论与分析，并编写程序验证分析的正确性。

从桶式排序的代码可以看出，代码的主循环进行d次，d是在B进制下最大值M的位数，例如$B=10,\ M=6655\rightarrow\ d=4$ or $B=16,\ M=255(FF)\rightarrow\ d=2$，在每次循环中先初始化$B$个桶的内容，然后将桶内元素全部倒出，这一过程花费时间$O(B+N)$，N为数据量。

所以桶式排序的时间复杂度为$T(B,N,M)=O((N+B)*log_BM)$，当$N\gg B$时，有$T(B,N,M)=O(B*log_BM)$

A．B,N,M的变化对$T(B,N,M)$的影响。

令$f=(N+B)*log_BM=\frac{lnM}{lnB}*(N+B)$

$I.\ \frac{\partial f}{\partial N}=\frac{lnM}{lnB}$

$II.\ \frac{\partial f}{\partial B}=-\frac{lnM}{B*(lnB)^2}*(N+B)+\frac{lnM}{lnB}=\frac{lnM}{B*(lnB)^2}*(B*lnB-B-N)$

$III.\ \frac{\partial f}{\partial M}=\frac{N+B}{M*lnB}$

首先，对于B，考察f对B偏导的正负性

令导数大于0，$(B*lnB-B-N)\geq 0$

变形：$B*lnB\geq B+N\sim N$

考虑到现实中数据量N远大于排序基数B，所以认为这个不等式在正常情况下不可能成立，故认为B的增长会使f下降。

对于M和N，比较两个偏导数的大小：

令$\frac{lnM}{lnB}>\frac{N+B}{M*lnB}, M*lnM>N+B\sim N, M*lnM>N$，

所以：1．当M与N差别较大时，由大项贡献主要的变化，

2．当M与N差别不大，N需要大于M较多才能提高自己的贡献率

综上所述：N与M对排序的贡献差别不算很大；B越大算法效率越高。

B．$T-\alpha$的图像，α为任意变量

a) 基数B：有$f=\frac{lnM}{lnB}*(N+B)\sim\frac{lnM}{lnB}*N$，所以$f$与$\frac{1}{lnB}$成正比，$T-B$类似反比例函数曲线

b) 数据规模N：有$f=\frac{lnM}{lnB}*(N+B)$，所以f与N成正比，$T-N$为一正斜率直线

c) 最大值M：有$f=log_BM*(N+B)=\mathrm{k}*log_BM$，所以$f$与$log_BM$成正比，$T-N$为多段连续平台，平台的分割点为$B^1,B^2,\dots,B^n$，例如$\mathrm{B}=10$，平台的分割点便是$M=10,100,1000$ …且对于相邻的平台，考虑$f_1=k*log_BB^t=k*t$，$f_2=\mathrm{k}*log_BB^{t+1}=k*(t+1)$，于是$f_2-f_1=k*(\mathrm{t}+1)-k*t=k$，故相邻两平台间的差值为定值。

d) 是否排序：该项不影响排序所消耗的时间。考虑算法原理，只在第一趟时有差异，但单趟排序的时间复杂度始终为$O(B+N)$，所以消耗的时间不会改变。令T为原数组的排序时间，T'为有序数组的排序时间，故$T'-T$曲线的斜率应为1。

2．中缀表达式转后缀表达式

e.g. $\{\frac{\mathrm{a+b}}{[\mathrm{c+d*(e-f)}]}\}-\mathrm{g}*(\mathrm{h}+\mathrm{i}*\mathrm{j})$

考虑到描述的准确性与统一性，先定义以下名词：

1) 操作数、操作符延用数学定义

2) 表达式：合法的运算表达式，例如上式与其子表达式

3) 子表达式：组成表达式的部分，例如$\{\frac{\mathrm{a+b}}{[\mathrm{c+d*(e-f)}]}\}-\mathrm{g}*(\mathrm{h}+\mathrm{i}*\mathrm{j})$，$\mathrm{g}*(\mathrm{h}+\mathrm{i}*\mathrm{j})$，$\mathrm{i}*\mathrm{j}$，e，j

4) 运算单元：子表达式中能够依照运算优先级正确完成的二元运算的操作符与操作数，例如$\mathrm{a}+\mathrm{b},\mathrm{e}-\mathrm{f},\mathrm{i}*\mathrm{j}$。值得注意的是，运算产生的中间值同样为操作数，按照相同的法则纳入运算。

根据上述定义，后缀表达式转换算法可以简单描述为：通过合并运算单元逐步使表达式缩小，最终化为单个运算单元。值得注意的是，括号具有最高的优先级，括号内的表达式当做独立的表达式处理。

所以一个不具有括号的表达式的转换过程为：

1) 遇到操作数，放入结果

2) 遇到操作符，与符号栈顶比较

   a) 若当前操作符优先级低，说明符号栈栈顶符号连接的两个子表达式可以组成一个运算单元，弹栈并写入结果

   b) 否则，当前符号入栈，说明当前正处理的表达式尚未成为运算单元

3) 重复 2 直至扫描到表达式尾

4) 将符号栈中残留的元素依次出栈放入结果

若将表达式的括号分割，例如$\left\{\frac{\mathrm{a+b}}{[\mathrm{c+d*(e-f)}]}\right\}-\mathrm{g}*(\mathrm{h}+\mathrm{i}*\mathrm{j})\rightarrow\frac{A}{B}-C$，其中$A=a+b,B=c+d*(e-f),C=g*(h+i*j)$，计算$\frac{A}{B}-C$即可得到结果．

所以该算法可以使用**递归**来实现，但本程序采用了**迭代**的方法。

四、算法实现

1．桶式排序与基数排序

```
int* RadixSort(int* A, int* B, int size, int base)
{
    int d = 1;
    int max = -1;
    for (int i = 0; i < size; i++)
    {
        if (max < A[i])
            max = A[i];
    }
    while (max /= base)
        ++d;
    for (int i = 1; i <= d; i++)
    {
        BucketSort(A, B, size, i, base);
        if (i < d)
        {
            SwapArr(A, B, size);
        }
    }
    return B;
}
// inline int gen(int num, int d, int base) 获得 base 进制下 num 的第 d 位
void BucketSort(int* A, int* B, int size, int digit, int base)
{
    int C[base];
    for (int i = 0; i <= base - 1; i++)
        C[i] = 0;
    for (int i = 0; i < size; i++)
        C[gen(A[i], digit, base)]++;
    for (int i = 1; i <= base - 1; i++)
        C[i] += C[i - 1];
    // 因为原始数组中低一位已经升序排好
    // 所以较大但高位相同的元素应当提前进入数组
    for (int i = size - 1; i >= 0; i--)
    {
        int tmp = gen(A[i], digit, base);
        B[C[tmp] - 1] = A[i];
        C[tmp]--;  // 处理相同情况
    }
}
```

2．表达式转换

```
Expression LNRToLRN(Expression LNR)
{
    Expression LRN = InitList(NULL);
    Unit Pori = FirstElement(LNR);       // 指向待转换表达式 Point-to-Original-data
    Unit Pres = Header(LRN);             // 指向结果Point-to-Result
    Stack stack = InitStack(NULL);       // 符号栈
    char data;                           // 当前数据
    int level;                           // 当前数据的优先级    0:Number    1:+-    2:*/    10:Bracket
    while (Pori)
    {
        // PrintStack(stack);
        data = Retrieve(Pori);
        level = UnitLevel(data);
        if (level == LEVEL_NUMBER)  // 是数字，直接送进结果
            Pres = InsertAfter(data, LRN, Pres);
        else  // 不是数字
        {
            if (level != LEVEL_BRACKET)  // 不是括号
            {
                // 符号栈为空 或 栈顶为括号(相当于压入左括号后开辟了一个子栈),
                // 直接入栈
                if (IsEmpty(stack) || UnitLevel(Top(stack)) == LEVEL_BRACKET)
                {
                    Push(data, stack);
                }
                // 当前元素优先级高于栈顶元素，直接入栈
                // 相当于当前表达式还未完成
                else if (level > UnitLevel(Top(stack)))
                {
                    Push(data, stack);
                }
                // 当前元素优先级小于或等于栈顶元素
                else
                {
                    while (1)
                    {
                        // 遇到左括号，说明之前的子表达式完成
                        if (IsEmpty(stack) || BracketLevel(Top(stack)))
                        {
                            Push(data, stack);
```

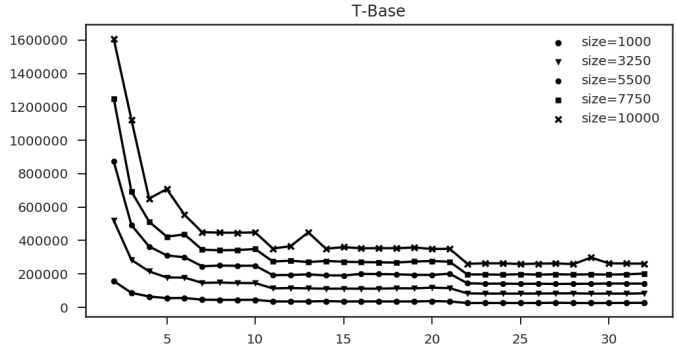
```
                    break;
                }
                // 优先级较大(相当于完成了一个子表达式)
                else if (level > UnitLevel(Top(stack)))
                {
                    Push(data, stack);
                    break;
                }
                // 优先级小于栈顶元素，栈顶元素出栈并进入结果
                // 说明栈顶符号与其连接的两个元素为子表达式
                else
                {
                    Pres = InsertAfter(Pop(stack), LRN, Pres);
                }
            }
        }
    }
    else  // 为括号
    {
        // 是左括号，直接入栈
        if (BracketLevel(data) < BRACKET_BORDER)
        {
            Push(data, stack);
        }
        // 是右括号，pop 直到栈顶为左括号
        else
        {
            while (1)
            {
                // 遇到左括号，停止 pop
                if (UnitLevel(Top(stack)) == LEVEL_BRACKET &&
                    BracketLevel(Top(stack)) < BRACKET_BORDER)
                {
                    Pop(stack);
                    break;
                }
                // pop 并直接进入结果
                else
                {
                    Pres = InsertAfter(Pop(stack), LRN, Pres);
                }
            }
        }
    }
    Pori = Advance(Pori);
}
// 将剩余操作符全部 pop 进结果
while (!IsEmpty(stack))
{
    Pres = InsertAfter(Pop(stack), LRN, Pres);
}
return LRN;
}
```

五、实验结果与分析

1．基数排序

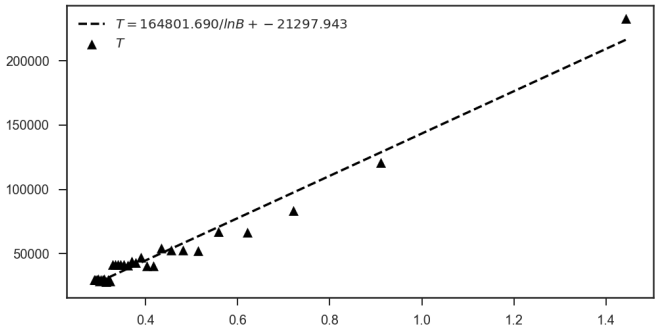
1) 基数B，由分析知基数B，由分析知$T-B$类似反比例函数曲线

a) 则当最大元素M固定时，有



可看出趋势线基本吻合理论分析。

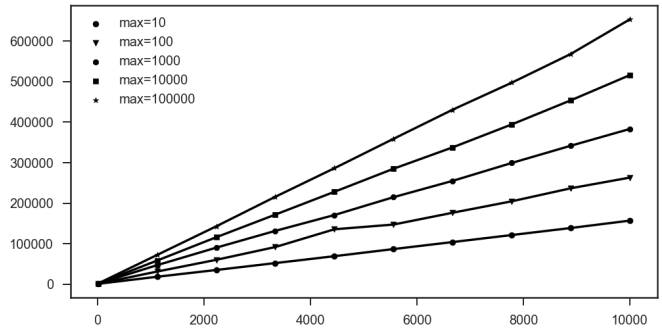
b) 又因为f与$\frac{1}{lnB}$成正比，所以在$N=1000$时，绘制$T-\frac{1}{lnB}$图线：



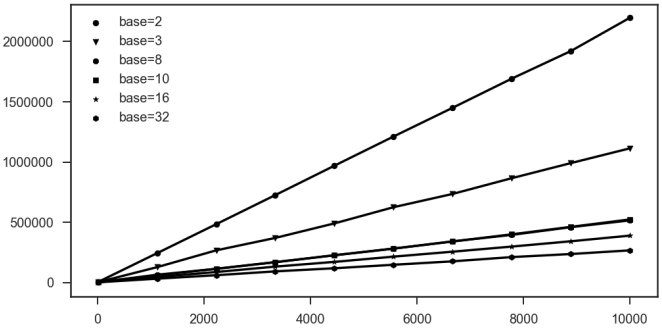
可见数据近似呈线性分布，可认为分析正确

2) 数据规模N

a) 基数B固定



b) 最大值M固定



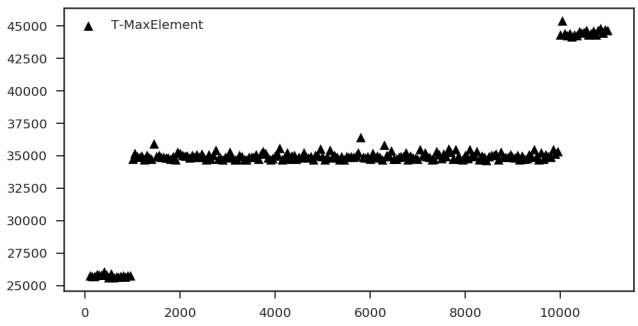
c) 由以上两图的线性关系可以认为实际结果与理论分析完全吻合

d) 还可以从图中读出：

i. 最大值呈指数级变化时，曲线斜率呈线性增长

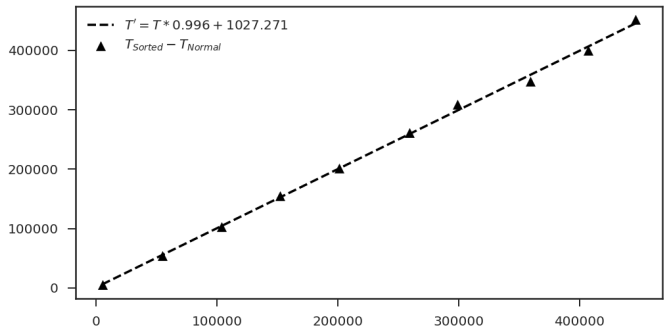
ii. 基数增大时曲线斜率减小，基数越大斜率减小得越慢

3) 最大值M，取$B = 10$



可看出散点图呈平台状分布，以1000与10000为间隔，相邻平台差值为10000， 与理论分析完全吻合。

4） 数组是否有序，取$B = 10, M = 10000$



可看出斜率几乎为1，与理论分析完全吻合。

2．后缀表达式转换

a） 括号不匹配

```
----------------[ Input String: ]---------------
a + (b + [c * d) + e]
---------------------[ END ]--------------------

-----[ Print: After Fix and Parse as List ]-----
a + ( b + [ c * d ) + e ]
---------------------[ END ]--------------------
[ERROR] input error, bracket not match!
```

b） 书中样例

```
----------------[ Input String: ]---------------
a + b * c + (d * e + f) * g
---------------------[ END ]--------------------

-----[ Print: After Fix and Parse as List ]-----
a + b * c + ( d * e + f ) * g
---------------------[ END ]--------------------

----[ Print: Convert to Postfix Expression ]----
a b c * + d e * f + g * +
---------------------[ END ]--------------------
```

c） 正常输入

```
----------------[ Input String: ]---------------
{(a + b) / [c + d * (e - f)]} - g * (h + i * j)
---------------------[ END ]--------------------

-----[ Print: After Fix and Parse as List ]-----
{ ( a + b ) / [ c + d * ( e - f ) ] } - g * ( h + i * j )
---------------------[ END ]--------------------

----[ Print: Convert to Postfix Expression ]----
a b + c d e f - * + / g h i j * + * -
---------------------[ END ]--------------------
```