周鹏宇+2023e8013282150

设计思路

- 考虑到派续后求间距最大值依然是最为直观的方式，同时又求设计一个$O(N)$复杂度的算法，因而自然而然地采用桶排序
- 考虑到如下规律：

$$L_{max} \geq \frac{V_{max} - V_{min}}{n-1}$$

其中L_{max}为最大间隔，V_{max}，V_{min}为读入数据的最大最小值，n为读入数据的数量，则若每个桶的大小控制在$\frac{V_{max}-V_{min}}{n-1}$，即可保证产生最大间距的两个点，一定在不同的桶中

- 因此对于每个桶，我们只需要知道桶内的最大最小值（第一个桶只需知道最大值，最后一个桶只需要知道最小值），然后比较相邻非空桶的最小值与最大值之差即可

复杂度分析

- 桶的数量为$\frac{数据最大差值}{桶大小}$，其中桶大小已经确定（见上），故数量为$n-1$
- 由于找一个数组中的最大最小值都是线性复杂度，故对每个桶内的最大最小值查询与桶中数据数有关，记桶编号为i，则其数据量记为t_i，有$\sum_{i=1}^{n-1} t_i = n$则有总计算复杂度为

$$\sum_{i=1}^{n-1} O(t_i) = O(n)$$

- 若在维护桶时只维护最大最小值而非链表，则遍历最大最小值的复杂度可以忽略，只需关注插入复杂度和后续对桶间间距比大小的复杂度，后两者显然为$O(n)$复杂度

代码设计（详见附件中代码文件）

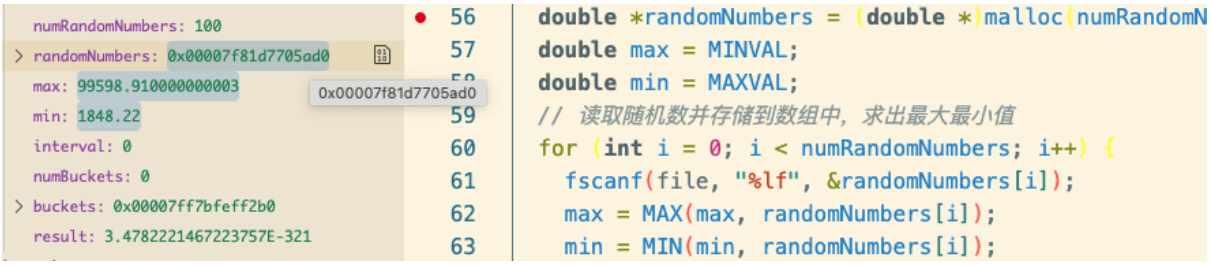
```
//
// Created by 周鹏宇 on 9/14/23.
//
#include <stdio.h>
#include <stdlib.h>
```

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))

#define MIN(a, b) ((a) < (b) ? (a) : (b))

#define MAXVAL 2144967295
#define MINVAL -2144967295

//As a golden model, not involved in my algorithm
void BubbleSort(double arr[], int n) {
  int swapped;
  do {
    swapped = 0;
    for (int i = 1; i < n; i++) {
      if (arr[i - 1] > arr[i]) {
        double temp = arr[i - 1];
        arr[i - 1] = arr[i];
        arr[i] = temp;
        swapped = 1;
      }
    }
  } while (swapped);
}


typedef struct bucket {
  double min;
  double max;
  int flag;
} bucket;

int main() {
  FILE *file;
  int numRandomNumbers;

  // 打开文件以读取模式
  file = fopen("input.txt", "r");

  // 检查文件是否成功打开
  if (file == NULL) {
    printf("Unable to open the file\n");
    return 1;
  }
  // 读取随机数的数量
  fscanf(file, "%d", &numRandomNumbers);

  // 声明一个数组来存储随机数
  double *randomNumbers = (double *)malloc(numRandomNumbers * sizeof(double));
  double max = MINVAL;
  double min = MAXVAL;
  // 读取随机数并存储到数组中，求出最大最小值
  for (int i = 0; i < numRandomNumbers; i++) {
    fscanf(file, "%lf", &randomNumbers[i]);
    max = MAX(max, randomNumbers[i]);
    min = MIN(min, randomNumbers[i]);
  }
```

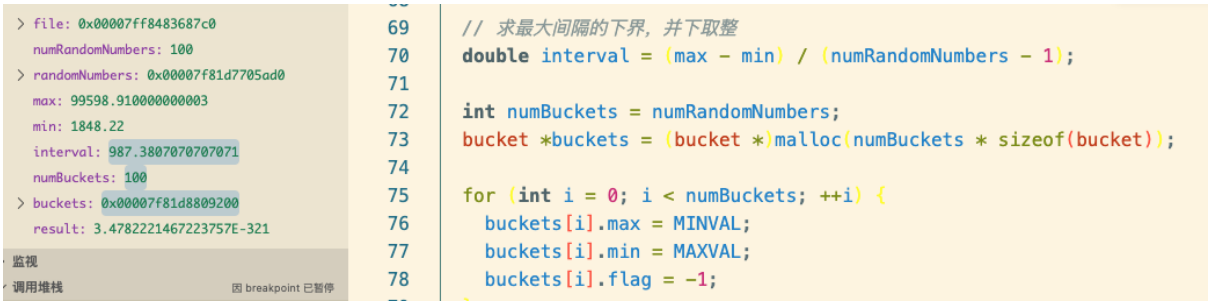
```
// 关闭文件
fclose(file);

// 求最大间隔的下界，并下取整
double interval = (max - min) / (numRandomNumbers - 1);

int numBuckets = numRandomNumbers;
bucket *buckets = (bucket *)malloc(numBuckets * sizeof(bucket));

for (int i = 0; i < numBuckets; ++i) {
  buckets[i].max = MINVAL;
  buckets[i].min = MAXVAL;
  buckets[i].flag = -1;
}

for (int i = 0; i < numRandomNumbers; ++i) {
  int BucketIdx = (int)((randomNumbers[i] - min) / interval);
  buckets[BucketIdx].min = MIN(buckets[BucketIdx].min, randomNumbers[i]);
  buckets[BucketIdx].max = MAX(buckets[BucketIdx].max, randomNumbers[i]);
  buckets[BucketIdx].flag = 1;
}

double result = 0.0;

for (int i = 0; i < numBuckets;) {
  double bmax = MINVAL;
  double bmin = MAXVAL;
  while (buckets[i].flag == -1) {
    ++i;
  }
  bmax = buckets[i].max;
  ++i;
  while (buckets[i].flag == -1) {
    ++i;
  }

  if (i > numBuckets - 1) {
    break;
  }
  bmin = buckets[i].min;
  result = MAX(result, bmin - bmax);
}

//    //进行快排后得到结果进行比较
//    BubbleSort(randomNumbers, numRandomNumbers);
//    printf("run to here\n");

//    double golden = 0;
//    for (int i = 0; i < numRandomNumbers - 1; ++i) {
//        golden = MAX(golden, randomNumbers[i + 1] - randomNumbers[i]);
//    }
//
//    // 打印数组中的随机数
//    for (int i = 0; i < numRandomNumbers; i++) {
//        printf("%.2lf\n", randomNumbers[i]);
```

```
//    }
//
//    printf("max is %2lf\n", result);
//
//    if (result == golden) {
//        printf("Right\n");
//    } else {
//        printf("False\n");
//    }

  // 释放动态分配的内存
  free(randomNumbers);

  file = fopen("output.txt", "w");

  if (file == NULL) {
    printf("Unable to open the file\n");
    return 1;
  }

  fprintf(file, "%lf\n", result);

  fclose(file);

  return 0;
}
```

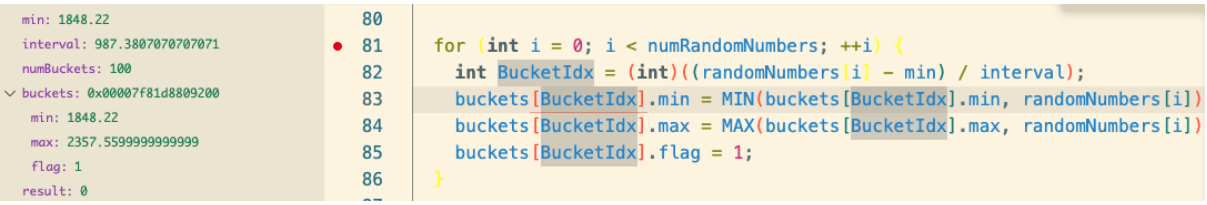
算法运行截图



读取随机数文件，并求出最大最小值（线性复杂度），本例中随机数数量为100，最大值为99598.910000000003，最小值为1848.22，



求桶的数量并初始化桶，操作量为线性，本例中桶大小为987.3807070707071，桶数量为100

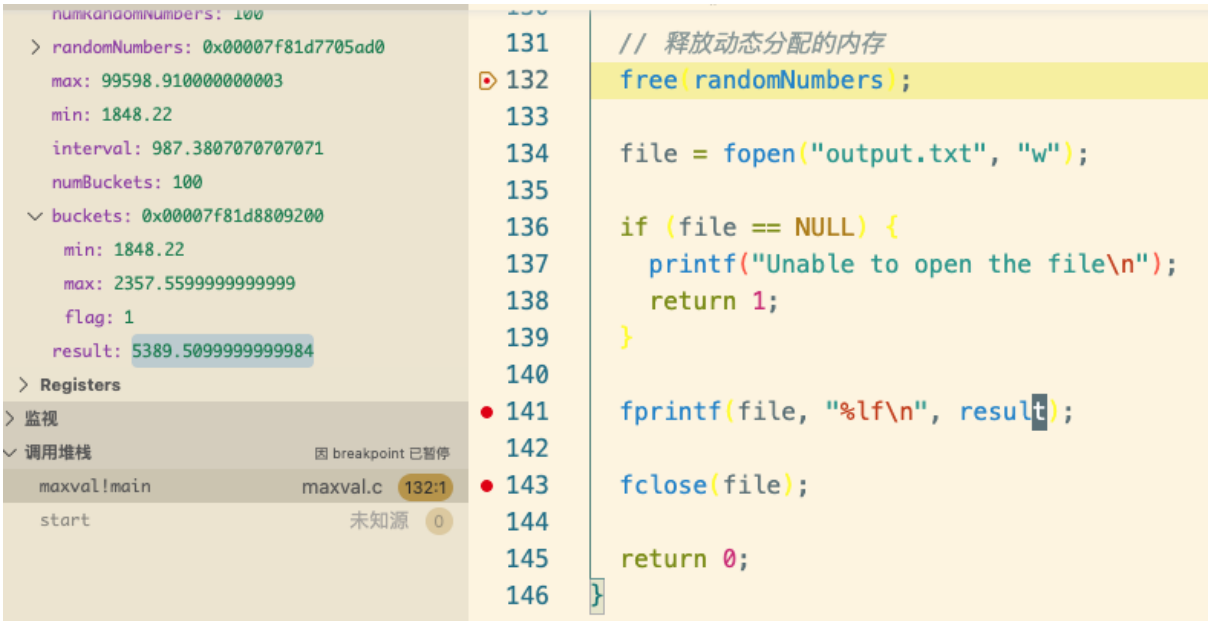


向桶内填充数据，本例中，第一个桶内最大数据是2357.5599999999999，最小数据为1848.22，flag为1代表此桶非空，同时我们检查另一个桶

```
buckets[3].min
2144967295
buckets[3].max
-2144967295
buckets[3].flag
-1
```

可以看到桶内最大最小值依然是初始值，flag为-1代表此桶为空桶

随后不断检查相邻非空桶的最大最小值差，具体设计思路参考前文算法描述，复杂度为为线性



最终得到结果5389.5099999999984并存入输出文件中

此外，通过先排序（冒泡排序）得到参考值后对结果进行比对，可以看到

```
printf("max is %2lf\n", result);

if (result == golden) {
    printf("Right\n");
} else {
    printf("False\n");
}
```

```
max is 5389.510000
Right
```

即结果无误。