数据库系统实验3报告

## 1.实验环境

操作系统: windows11

编程语言: Java

编译器: jdk1.8

开发环境: Intellij IDEA

依赖环境: maven

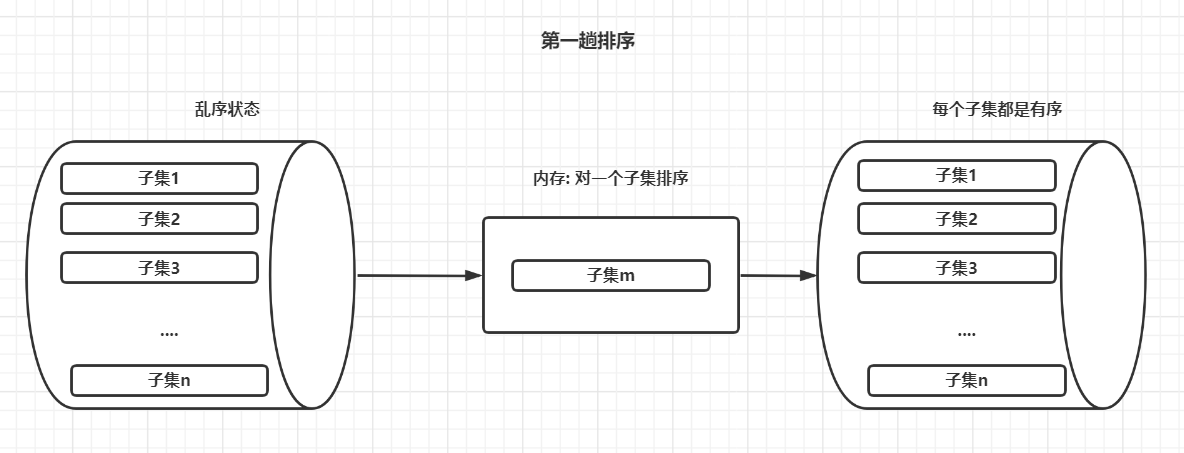
## 2.算法设计与实现

两阶段多路归并排序算法是用来解决当内存不足时, 如何将大量的元组一次放入内存中进行排序, 可以分两次装入内存进行排序。

先确定**元组的结构体**, 按照要求是4字节整数和12字节字符串, 整数和字符串都随机生成。

1. // 一个元组
2. **typedef** **struct** tuple{
3. **int** a;
4. **char**\* b;
5. } Tuple;

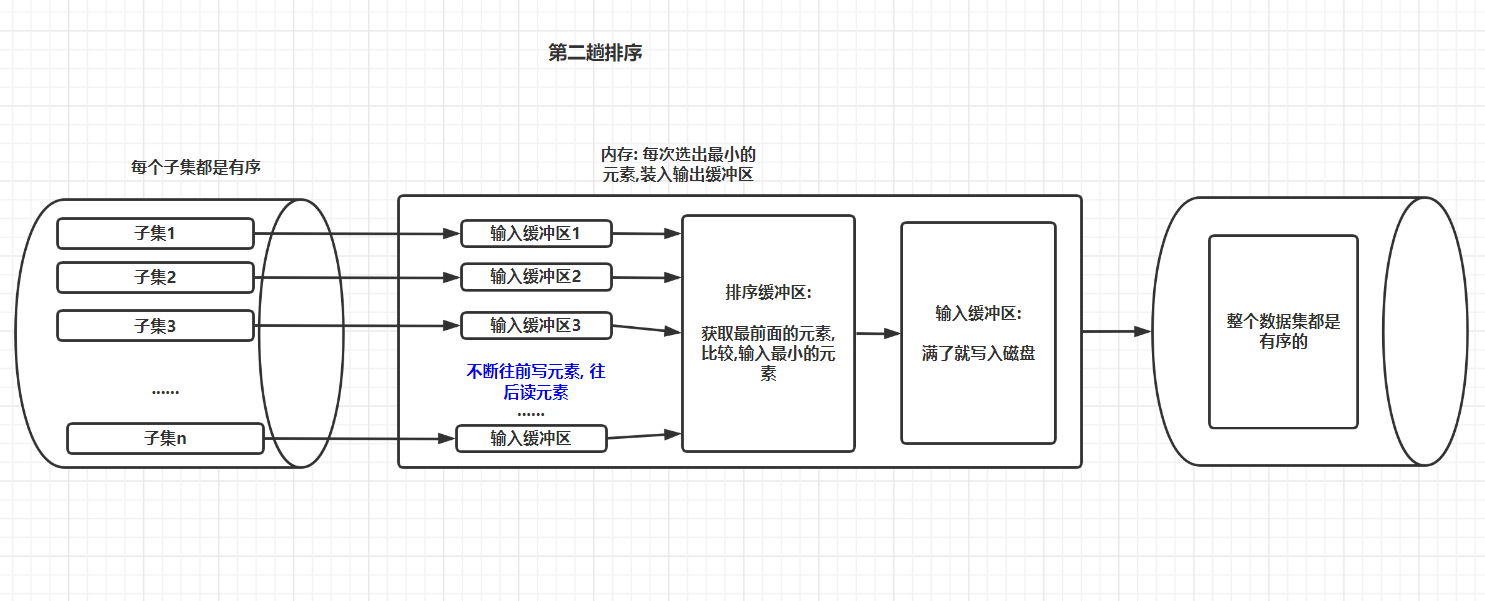
**第一趟归并排序**: 要先把所有元组分成若干个子集(每个子集可以完全装进内存), 再把各个子集排好序后写回去。



因为有1M的内存,每个元组的大小为16B, 所以每次可以最多可以放入62500个元组, 那么可以把所有的元组分成16路, 先读入文件, 再在内存中进行内部排序,使用的排序函数是stdlib.h中的qsort (快速排序)。然后对排好序的元组分别写入16个中间文件'A' - 'P'中。

1. void firstSort() {
2. int tupleNums[ways]; *// 每个子集中存储的元组数*
3. for (int i = 0; i < ways; i++) {
4. tupleNums[i] = i < allTuples % ways ? allTuples / ways + 1 : allTuples / ways;
5. printf("%d ", tupleNums[i]);
6. }
7. printf("\n");
8. *// 从输入中读取数据*
9. FILE \*fr;
10. if ((fr = fopen(inputPath, "r")) == NULL) {
11. printf("Fail to open file: input!\n");
12. exit(0);
13. }
14. for (int i = 0; i < ways; i++) {
15. Tuple\* pTuple = readTuples(fr, allTuples / ways); *// 读入长度为每个子集的长度*
16. *// 使用快速排序库函数*
17. qsort(pTuple, allTuples / ways, sizeof(Tuple), compare);
18. *// 向输出中写数据*
19. char\* sortedPath = getFilePath(i);
20. printf("%s\n", sortedPath);
21. FILE\* fw;
22. if ((fw = fopen(sortedPath, "w")) == NULL) {
23. printf("Fail to open file: %c!\n", i + 64);
24. exit(0);
25. }
26. writeTuples(fw, pTuple, allTuples / ways);
27. printf("the %d th has been sorted firstly\n", i + 1);
28. free(sortedPath);
29. freeTuples(pTuple, allTuples / ways);
30. fclose(fw);
31. }
32. fclose(fr);
33. }

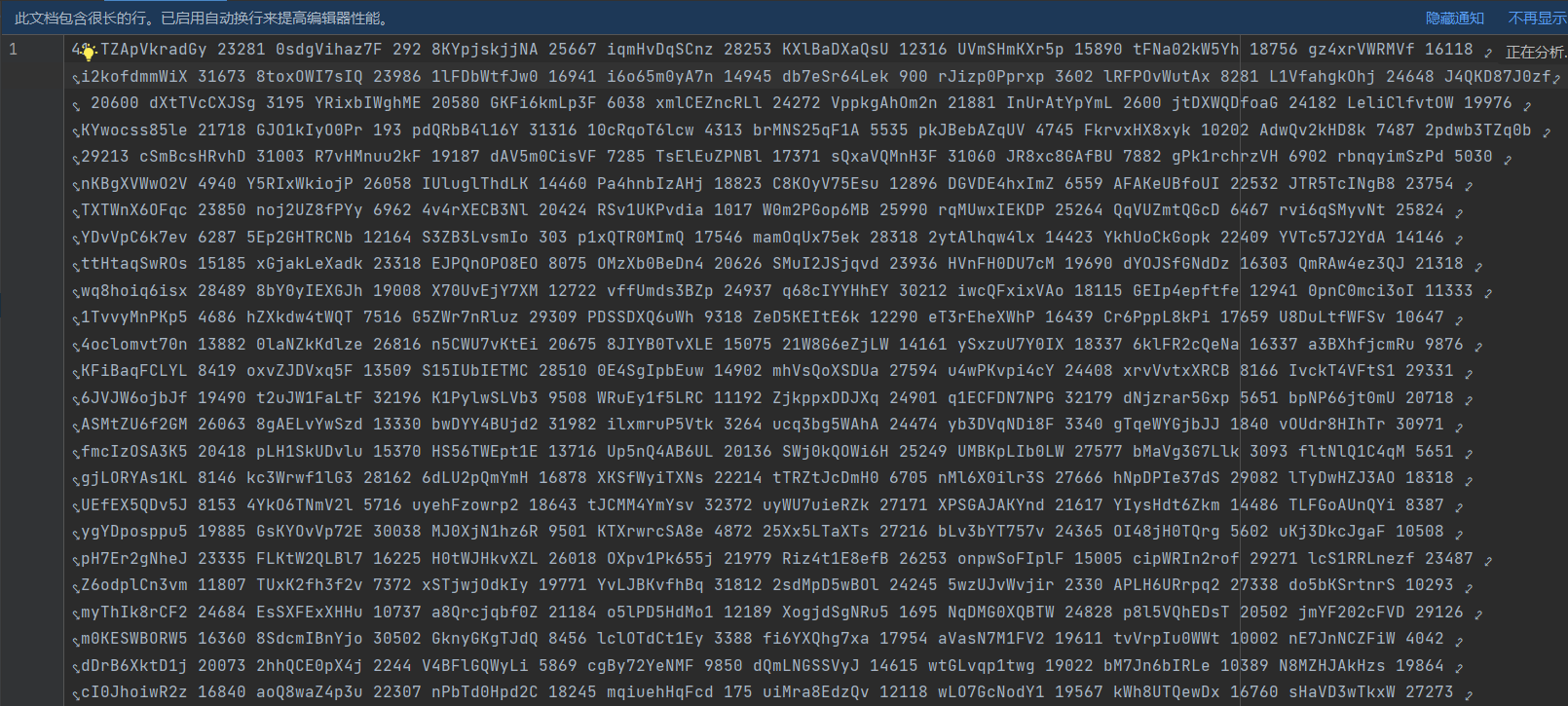
**第二趟归并排序**: 把排好序的若干子集, 分别取出他们的最前面若干元素放入内存(输入缓冲区), 然后对么每个子集最前面的元素进行比较(一个子集最前面的元素一定是这个子集最小的元素), 这些正在比较的元素称为排序缓冲区, 最小的元素放入输出缓冲区中, 然后一次比较后面的元素, 容易得知每次进入输出缓冲区的元素是整个数组中最小的元素。



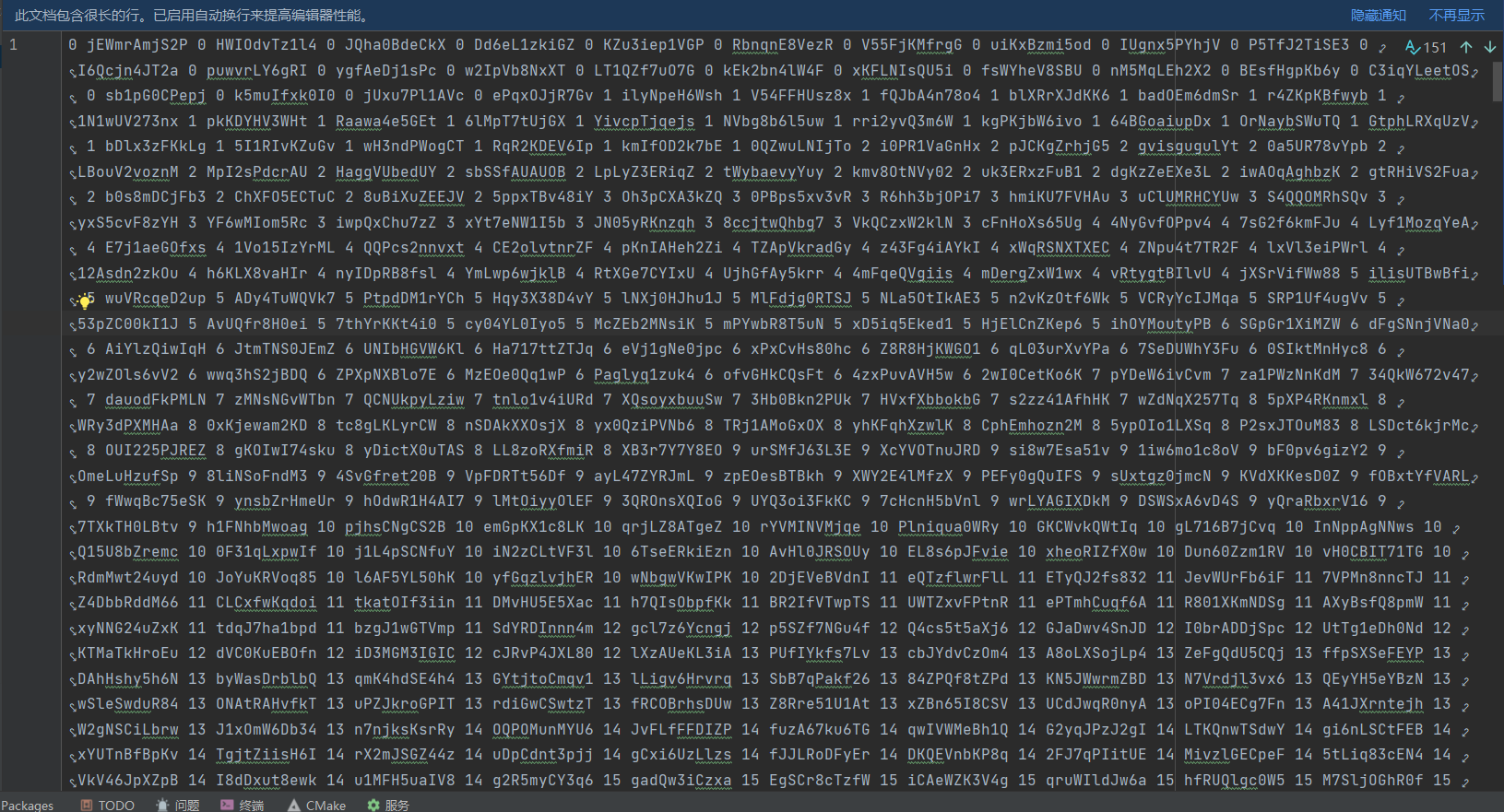
1. *// 二趟排序*
2. void secondSort() {
3. Tuple\*\* pTuple2 = createTuples2(ways); *// 输入缓冲区*
4. FILE\* fr[ways];
5. for(int i = 0; i < ways; i ++){ *// 向ways个文件读数据*
6. char inputPathX[50] = "";
7. strcpy(inputPathX, directoryPath);
8. char tap[2];
9. tap[0] = (char) (i + 65);
10. tap[1] = '\0';
11. strcat(inputPathX, tap);
12. if ((fr[i] = fopen(inputPathX, "r")) == NULL) {
13. printf("Fail to open file: input!\n");
14. exit(0);
15. }
16. pTuple2[i] = readTuples(fr[i], waitToSort);
17. }
18. *// 初始化缓冲区*
19. int waitNum[ways]; *// 每个子集下一个正在等的元素*
20. bool isEnd[ways]; *// 每个子集是否已经读完*
21. Tuple sorting[ways]; *// 正在排序的元素*
22. Tuple output[waitToOutput]; *// 等待被输出的元组(输出缓冲区)*
23. int outputNum = 0; *// 当前输出缓冲区的元素数量*
24. for(int i = 0; i < ways; i ++){
25. isEnd[i] = false;
26. sorting[i] = pTuple2[i][0]; *// 把第一个元组放入排序中*
27. waitNum[i] = 1;
28. }
29. int writeCount = 0; *// 输出缓冲区输出的次数*
30. FILE\* fw;
31. if ((fw = fopen(outputPath, "w")) == NULL) {
32. printf("Fail to open file: output!\n");
33. exit(0);
34. }
35. while(true){
36. globalMin = getMin(sorting, isEnd, ways);
37. if(globalMin == -1){ *// 所有排序结束*
38. *// 输出缓冲区里没有多余的元素, 只要计算好[输出缓冲区大小]整除[总元组数量]*
39. printf("all the tuples are sorted finally, End!\n");
40. for(int i = 0; i < ways; i ++){ *// 释放资源*
41. fclose(fr[i]);
42. if(pTuple2[i] != NULL)
43. freeTuples(pTuple2[i], globalOutputNum[globalMin]);
44. }
45. fclose(fw);
46. free(pTuple2);
47. break;
48. }
49. output[outputNum ++] = sorting[globalMin]; *// 最小值放入输出缓冲区*
50. sorting[globalMin] = pTuple2[globalMin][waitNum[globalMin] ++];
51. *// 第min个等待区++*
52. *// 如果输出缓冲区已满*
53. if(outputNum == globalOutputNum[globalMin]){
54. writeTuples( fw, output, globalOutputNum[globalMin]);
55. outputNum = 0; *// 置零*
56. writeCount ++;
57. printf("output data to disk for %d times\n", writeCount);
58. }
59. if(waitNum[globalMin] == waitToSort){ *// 如果某一行已经输出完*
60. free(pTuple2[globalMin]);
61. pTuple2[globalMin] = readTuples(fr[globalMin], waitToSort);
62. waitNum[globalMin] = 0; *// 正在等待的元素置零*
63. *// 输入缓冲区读完要重新赋值*
64. if(globalOutputNum[globalMin] < waitToOutput){
65. isEnd[globalMin] = true;
66. }
67. }
68. }
69. }

## 3.结果测试

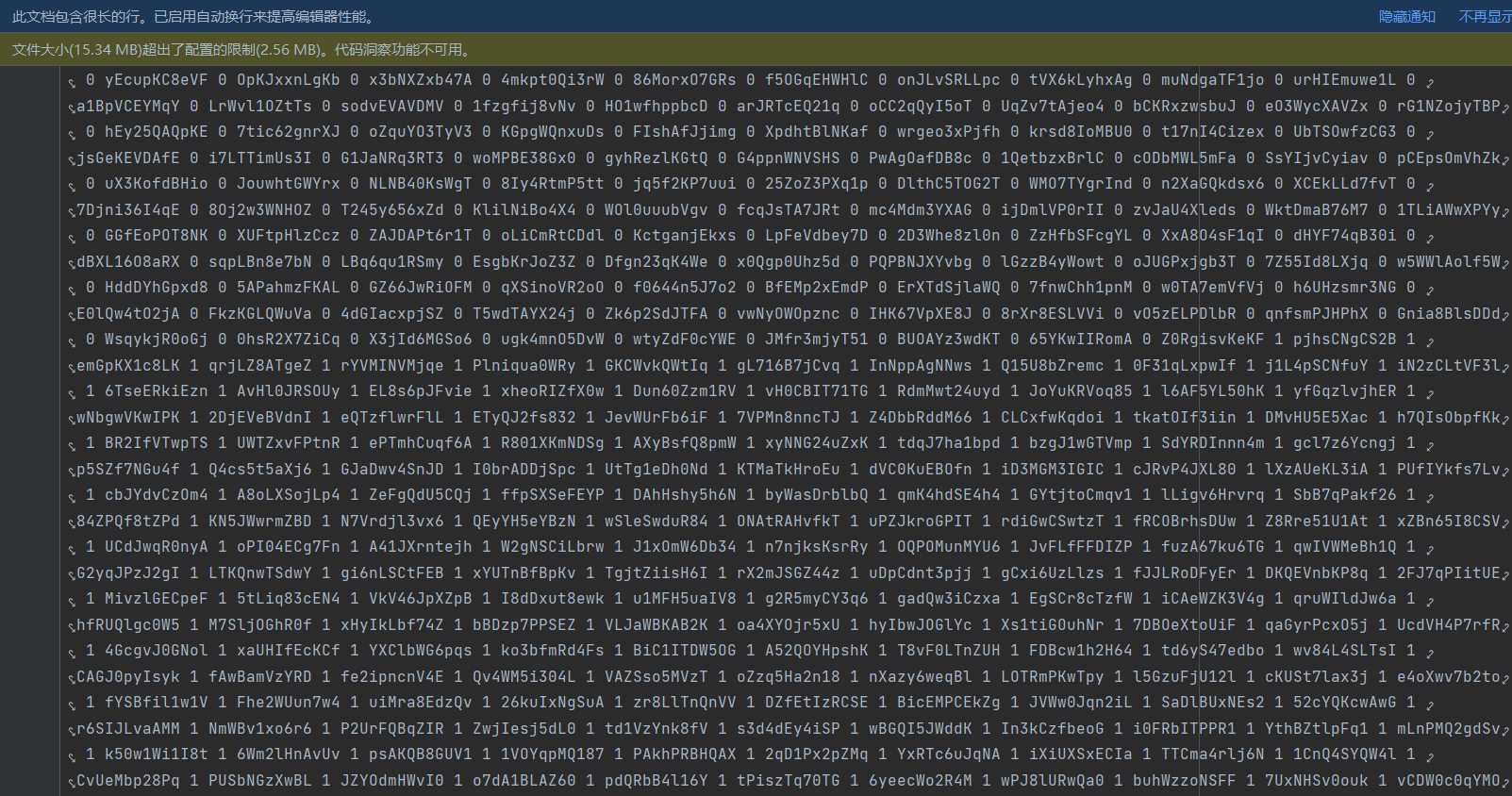
初始的数据:



一次排序之后的数据(其中一项):



二次排序之后的数据:



运行时间:

