**算法分析与设计实验报告**

**第 1 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邹林壮 | 学号 | 202208040412 | | 班级 | 计算机科学与技术（拔尖班）2201 |
| 时间 | 2024.10.19 | 地点 | 院楼432 | | | |
| 实验名称 | 用分治法实现合并排序 | | | | | |
| 实验目的 | 通过上机实验，要求掌握分治算法的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 利用用分治法实现合并排序，给定任意几组数据，将数据进行排序并将排好的数据进行输出，最终计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | ①判断是否需要进行分割：对数组 a[l...r]，判断当前子数组是否需要继续分割。若 l >= r，说明子数组只包含一个元素或为空，无需继续分割，递归返回；否则，继续进行分割操作。  ②计算中间位置： 计算数组的中间位置 mid = l + (r - l) / 2，将当前数组划分为两个部分：a[l...mid] 和 a[mid+1...r]。  ③递归对左右两部分分别进行排序：对左半部分数组 a[l...mid] 进行递归调用 mergesort。对右半部分数组 a[mid+1...r] 进行递归调用 mergesort。  ④归并两部分数组：使用归并函数 merge，将已经排好序的两个子数组 a[l...mid] 和 a[mid+1...r] 合并成一个有序数组 a[l...r]。在 merge 过程中，使用双指针分别指向两个子数组的起始位置，逐个比较并将较小的元素放入临时数组 tmp 中，最后将 tmp 复制回原数组对应的位置。  ⑤输出排序结果：当整个数组都通过归并排序完成后，最终得到有序数组，并输出排序结果。  其中①,②,③,④步也可以递推实现  ①先判断是否需要继续分割数组：当步长 step 小于数组大小 n 时，继续进行排序操作。  ②确定当前分组的起始、中间和结束位置：对于每一轮迭代，将数组分成两部分进行归并，需要确保min和r的位置正确性  ③调用归并操作：调用 merge(num, l, mid, r); 将两个已排序的子数组 num[l...mid] 和 num[mid+1...r] 合并成一个有序的数组。  ④更新步长：将步长 step 加倍，准备进行下一轮更大范围的归并操作。 | | | | | |
| 关键代码 | void merge(int\* a,int l,int mid,int r){  int i=l,j=mid+1,k=0;  vector<int> tmp(r-l+1);  while(i<=mid&&j<=r){  if(a[i]<=a[j])tmp[k++]=a[i++];  else tmp[k++]=a[j++];  }  while(i<=mid)tmp[k++]=a[i++];  while(j<=r)tmp[k++]=a[j++];  for(int k=0;k<tmp.size();k++)a[l+k]=tmp[k];  }  //递归实现mergesort：  void mergesort(int\* a,int l,int r){  if(l>=r)return;  int mid=l+(r-l)/2;  mergesort(a,l,mid);  mergesort(a,mid+1,r);  merge(a,l,mid,r);  }  //递推实现mergesort void mergesort(vector<int> &num，int l,int r){  int step=1;  int n=num.size();  while(step<n){  for(int i=0;i<n;i+=2\*step){  int l=i;  int mid=min(i+step-1,n-1);  int r=min(i+2\*step-1,n-1);  merge(num,l,mid,r);  }  step\*=2;  }  } | | | | | |
| 测试结果 | 1. 确保了正确性   IMG_256  2)时间复杂度在最好，最差和平均情况下均为O(nlogn)  公式推到为T(n)=2T(n/2)+O(n),经过推导可以得到这个结论  空间复杂度为O(n)（其中数组占用O(n),递归实现每次开辟两个空间，但是递归深度为logn，因此最终也是占用O(n))  3)构造了三个规模的数据集：10，10000，1000000  递归时间： | | | | | |
| 实验心得 | 1. 归并排序和堆排序一样是效率最高的排序算法之一，通过学习这个算法，我深入理解了分治法的思路，首先将问题分解为可以解决的独立子问题，也就是分解数组到每一个元素，然后逐步进行合并，也就是分治法先分解后合并的解决思路。 2. 子问题的效率主要参考其时间复杂度和空间复杂度，这两个与数据量及其规模有关。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**问题代码：**

**#include <iostream>**

**#include <chrono> // 使用高精度时间测量的头文件**

**#include<vector>**

**using namespace std;**

**using namespace std::chrono; // 使用 chrono 命名空间**

**const int N=5e7;**

**vector<int> a(N+1);**

**void merge(vector<int>& a,int l,int mid,int r) {**

**int i=l,j=mid+1,k=0;**

**vector<int> tmp(r-l+1);**

**while(i<=mid&&j<=r) {**

**if(a[i]<=a[j])tmp[k++]=a[i++];**

**else tmp[k++]=a[j++];**

**}**

**while(i<=mid)tmp[k++]=a[i++];**

**while(j<=r)tmp[k++]=a[j++];**

**for(int k=0; k<tmp.size(); k++)a[l+k]=tmp[k];**

**}**

**void mergesort(vector<int>& a,int l,int r) {**

**if(l>=r)return;**

**int mid=l+(r-l)/2;**

**mergesort(a,l,mid);**

**mergesort(a,mid+1,r);**

**merge(a,l,mid,r);**

**}**

**//void mergesort(vector<int>& a,int l,int r){**

**// int step=1;**

**// int n=a.size();**

**// while(step<n){**

**// for(int i=0;i<n;i+=2\*step){**

**// int l=i;**

**// int mid=min(i+step-1,n-1);**

**// int r=min(i+2\*step-1,n-1);**

**// merge(a,l,mid,r);**

**// }**

**// step\*=2;**

**//}**

**//}**

**int main() {**

**int n=0;**

**cin>>n;**

**//int a[10]= {4,5,2,8,6,3,4,9,88,45};**

**for(int i=0; i<n; i++)cin>>a[i];**

**mergesort(a, 0, n - 1);**

**for (int i = 0; i < n; i++)cout << a[i] << " ";**

**return 0;**

**}**

**计算时间代码：**

**#include <iostream>**

**#include <chrono>**

**#include <vector>**

**#include <fstream>**

**#include <cstdlib>**

**#include <ctime>**

**using namespace std;**

**using namespace std::chrono;**

**const int N = 5e7; // 设置数据的最大值**

**vector<int> a(N+1);**

**// 合并两个有序区间**

**void merge(vector<int>& a, int l, int mid, int r) {**

**int i = l, j = mid + 1, k = 0;**

**vector<int> tmp(r - l + 1);**

**while (i <= mid && j <= r) {**

**if (a[i] <= a[j]) tmp[k++] = a[i++];**

**else tmp[k++] = a[j++];**

**}**

**while (i <= mid) tmp[k++] = a[i++];**

**while (j <= r) tmp[k++] = a[j++];**

**for (int k = 0; k < tmp.size(); k++) a[l + k] = tmp[k];**

**}**

**//归并排序**

**void mergesort(vector<int>& a, int l, int r) {**

**if (l >= r) return;**

**int mid = l + (r - l) / 2;**

**mergesort(a, l, mid);**

**mergesort(a, mid + 1, r);**

**merge(a, l, mid, r);**

**}**

**//void mergesort(vector<int>& a,int l,int r){**

**// int step=1;**

**// int n=a.size();**

**// while(step<n){**

**// for(int i=0;i<n;i+=2\*step){**

**// int l=i;**

**// int mid=min(i+step-1,n-1);**

**// int r=min(i+2\*step-1,n-1);**

**// merge(a,l,mid,r);**

**// }**

**// step\*=2;**

**//}**

**//}**

**// 随机数据生成，增加 minRange 和 maxRange 参数控制数据范围**

**void generateRandomData(int n, const string& filename, int minRange, int maxRange) {**

**ofstream fout(filename);**

**srand(time(0)); // 初始化随机数种子**

**for (int i = 0; i < n; i++) {**

**fout << minRange + rand() % (maxRange - minRange + 1) << " "; // 生成指定范围内的随机数**

**}**

**fout.close();**

**}**

**// 从文件中读取数据**

**void readDataFromFile(vector<int>& a, const string& filename, int n) {**

**ifstream fin(filename);**

**for (int i = 0; i < n; i++) {**

**fin >> a[i];**

**}**

**fin.close();**

**}**

**int main() {**

**vector<int> sizes = {10, 10000, 1000000}; // 三种不同规模的数据**

**int minRange = 0; // 最小值**

**int maxRange = 1000000; // 最大值 (可以控制数据的范围大小)**

**for (int size : sizes) {**

**string filename = "mergesort" + to\_string(size) + ".txt";**

**// 1. 生成随机数据并写入文件，控制数据范围为 [minRange, maxRange]**

**generateRandomData(size, filename, minRange, maxRange);**

**// 2. 从文件中读取数据**

**readDataFromFile(a, filename, size);**

**// 3. 输出相关信息：数据规模**

**cout << "Data size: " << size << " numbers"<<endl;**

**cout<<"Random Range: [" << minRange << ", " << maxRange << "]." << endl;**

**// 4. 测量归并排序的执行时间**

**auto start = high\_resolution\_clock::now();**

**mergesort(a, 0, size - 1);**

**auto end = high\_resolution\_clock::now();**

**auto duration = duration\_cast<milliseconds>(end - start);**

**// 5. 输出排序时间**

**cout << "Sorting " << size << " numbers took: " << duration.count() << " milliseconds." << endl;**

**cout << "-------------------------------------------" << endl;**

**}**

**return 0;**

**}**