**算法分析与设计实验报告**

**第 2 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 | 题 2-6 排列的字典序问题 | | | | | |
| 实验目的 | 通过上机实验，要求掌握排列的字典序问题的问题描述、算法设计思想、程序设计。 | | | | | |
| 实验原理 | 根据排列的计算方式，采用递归策略，将问题分解成可解决的子问题，然后通过递推实现排列和字典序的相互转换，并计算出程序运行所需要的时间。 | | | | | |
| 实验步骤 | 1.首先确定问题的输入输出，输入是字典序长度n，以及字典序列，输出为字典序值，约束条件是字典序列是连续的，从1-n  2.  3.由递归式可以设计从排列到字典序值，使用rp[],使用all计算总和，all每次加上(rp[i]-1)\*f[n-i],然后将i之后大于rp[i]减一，将其重新凑成长度为n-i的排列  4.由递归式可以设计从字典序到排列，由于最后一位无法直接确定，将最后一位置置为1，可以通过后续计算得到，然后由字典序倒数第二位向前计算，每次用字典序值对f[i+1]取余后除以f[i]得到d，最后d+1得到该位当前的值，对于该值后面的数，如果大于d,则该数+1  5.也可以直观地求出下一个排列，首先找到递减序列前的最后一个数i，然后在后续递减序列从后往前进行依次比较，直到找到第一个比i大的数，然后交换这两个数，最后将i+1和最后进行逆序，得到下一个序列。其中i和j交换的原因是j是后续单调递减序列中第一个大于i的数，交换这两个数，实现了前面的数变大，那么后面的递减序列就应该转为单调递增序列，确保比之前的序列只大1 | | | | | |
| 关键代码 | int permRank(int n,int \*pi){  int r=0;  int rp[n+1];  for(int i=1;i<=n;i++)rp[i]=pi[i];  for(int i=1;i<=n;i++){  r+=(rp[i]-1)\*f[n-i];  for(int j=i+1;j<=n;j++)if(rp[j]>rp[i])rp[j]--;  }  return r;  }  void permUnrank(int n,int r,int \*pi){  int d=0;  pi[n]=1;  for(int i=1;i<n;i++){  d=(r%f[i+1])/f[i];  r-=d\*f[i];  pi[n-i]=d+1;  for(int j=n-i+1;j<=n;j++)if(pi[j]>d)pi[j]++;  }  }  int permSucc(int n,int \*pi) {      int flag=0;      pi[0]=0;      int i=n-1;      while(pi[i+1]<pi[i])i--;      if(i==0)flag=0;      else {          flag=1;          int j=n;          while(pi[j]<pi[i])j--;          swap(pi[i],pi[j]);          int l=i+1,r=n;          int temp;          while(l<=r) {              temp=pi[l];              pi[l]=pi[r];              pi[r]=temp;              l++;              r--;          }      }      return flag;  } | | | | | |
| 测试结果 | 1. 确定正确性：测试了多个案例，确定其正确性   IMG_256   1. 时间复杂度：permRank的时间复杂度是O(n^2),permUnrank的时间复杂度也是O(n^2),最终整个过程的时间复杂度为O(n^2)，而permSucc实现了O(n)的时间复杂度求解   空间复杂度：O(n),由输入的字典排序决定  3）构造了三个规模的数据集：6，10，12  IMG_256  已知20! = 2432902008176640000，由于在求解问题时会使用到20！，但是显然是超过计算机储存的，所以限制了问题的规模，最大为20 | | | | | |
| 实验心得 | 1. 要经常性使用分治法思路，看是否可以将复杂问题，转化为相同的子问题，将子问题不断地分解，最终得到可以解决的子问题，再将所有子问题进行合并，得到最终问题的解 2. 限制程序运行的条件不止时间复杂度和空间复杂度，还有数据的范围大小，如果某些数据过大，也会限制程序的正常运行。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**问题代码：**

**#include<bits/stdc++.h>**

**using namespace std;**

**#define N 10001**

**typedef long long ll;**

**ll f[N];**

**int permRank(int n,int \*pi){**

**int r=0;**

**int rp[n+1];**

**for(int i=1;i<=n;i++)rp[i]=pi[i];**

**for(int i=1;i<=n;i++){**

**r+=(rp[i]-1)\*f[n-i];**

**for(int j=i+1;j<=n;j++)if(rp[j]>rp[i])rp[j]--;**

**}**

**return r;**

**}**

**void permUnrank(int n,int r,int \*pi){**

**int d=0;**

**pi[n]=1;**

**for(int i=1;i<n;i++){**

**d=(r%f[i+1])/f[i];**

**r-=d\*f[i];**

**pi[n-i]=d+1;**

**for(int j=n-i+1;j<=n;j++)if(pi[j]>d)pi[j]++;**

**}**

**}**

**int permSucc(int n,int \*pi) {**

**int flag=0;**

**pi[0]=0;**

**int i=n-1;**

**while(pi[i+1]<pi[i])i--;**

**if(i==0)flag=0;**

**else {**

**flag=1;**

**int j=n;**

**while(pi[j]<pi[i])j--;**

**swap(pi[i],pi[j]);**

**int l=i+1,r=n;**

**int temp;**

**while(l<=r) {**

**temp=pi[l];**

**pi[l]=pi[r];**

**pi[r]=temp;**

**l++;**

**r--;**

**}**

**}**

**return flag;**

**}**

**int main(){**

**f[0]=1;**

**for(int i=1;i<=N;i++)f[i]=f[i-1]\*i;**

**int n;**

**int pi1[n+1],pi2[n+1];**

**int r1,r2;**

**cin>>n;**

**for(int i=1;i<=n;i++)cin>>pi1[i];**

**r1=permRank(n,pi1);**

**cout<<r1<<endl;**

**r2=r1+1;**

**permUnrank(n,r2,pi2);**

**for(int i=1;i<=n;i++)cout<<pi2[i]<<" ";**

**}**

**计算时间代码：**

**#include <bits/stdc++.h>**

**using namespace std;**

**#define N 500**

**typedef long long ll;**

**ll f[N];**

**// 计算排列的秩**

**int permRank(int n, int \*pi) {**

**int r = 0;**

**int rp[n + 1];**

**for (int i = 1; i <= n; i++) rp[i] = pi[i];**

**for (int i = 1; i <= n; i++) {**

**r += (rp[i] - 1) \* f[n - i];**

**for (int j = i + 1; j <= n; j++)**

**if (rp[j] > rp[i]) rp[j]--;**

**}**

**return r;**

**}**

**// 根据秩计算排列**

**void permUnrank(int n, int r, int \*pi) {**

**pi[n] = 1;**

**for (int i = 1; i < n; i++) {**

**int d = (r % f[i + 1]) / f[i];**

**r -= d \* f[i];**

**pi[n - i] = d + 1;**

**for (int j = n - i + 1; j <= n; j++)**

**if (pi[j] > d) pi[j]++;**

**}**

**}**

**// 生成随机排列并写入文件**

**void generateRandomPermutation(int n, const string& filename) {**

**vector<int> perm(n);**

**iota(perm.begin(), perm.end(), 1); // 生成 [1, 2, ..., n]**

**random\_shuffle(perm.begin(), perm.end()); // 随机打乱**

**ofstream fout(filename);**

**fout << n << endl;**

**for (int num : perm) {**

**fout << num << " ";**

**}**

**fout << endl;**

**fout.close();**

**}**

**// 从文件读取排列**

**void readPermutationFromFile(const string& filename, int \*pi, int &n) {**

**ifstream fin(filename);**

**fin >> n;**

**for (int i = 1; i <= n; i++) {**

**fin >> pi[i];**

**}**

**fin.close();**

**}**

**int main() {**

**// 预计算阶乘**

**f[0] = 1;**

**for (int i = 1; i <= N; i++) f[i] = f[i - 1] \* i;**

**vector<int> sizes = {6, 10, 12}; // 三种不同规模的排列**

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

**// 生成随机排列并写入文件**

**string filename = "random\_permutation\_" + to\_string(n) + ".txt";**

**generateRandomPermutation(n, filename);**

**// 从文件中读取排列**

**int pi[N], pi2[N];**

**readPermutationFromFile(filename, pi, n);**

**// 测量 permRank 和 permUnrank 运行时间**

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

**int r1 = permRank(n, pi); // 计算排列的秩**

**permUnrank(n, r1 + 1, pi2); // 计算下一个排列**

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

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

**// 输出结果**

**cout << "排列大小: " << n << endl;**

**cout << "运行时间: " << duration.count() << " 毫秒。" << endl;**

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

**}**

**return 0;**

**}**