**算法分析与设计实验报告**

**第 3 次实验**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 邹林壮 | 学号 | 202208040412 | | 班级 | 计科拔尖2201班 |
| 时间 | 2024.11.23 | 地点 | 院楼432 | | | |
| 实验名称 | 分析题 3-2 单调递增最长子序列问题 | | | | | |
| 实验目的 | 通过本实验，深入理解和掌握动态规划算法和贪心算法的设计思想，特别是如何通过耐心排序解决单调递增最长子序列问题.以及通过优化动态规划元素内容来优化算法，并且通过特别思路技巧和处理，将时间复杂度减小到。同时，通过实际编程练习，提高对C++语言的运用能力，以及对算法性能的分析能力。 | | | | | |
| 实验原理 | 单调递增最长子序列问题是求一个序列中最长的严格递增子序列的长度。耐心排序是一种基于贪心策略的算法，通过维护一个辅助数组（伪堆）来记录当前找到的最小尾元素，从而实现的时间复杂度。也可以通过对dp数组存储含义进行优化，并通过二分搜索进行查询优化，从而实现的时间复杂度。 | | | | | |
| 实验步骤 | 1. **确定输入输出**   输入为子序列长度n，子序列数组  输出为单调递增最长子序列  约束为时间复杂度为   1. **使用动态规划解决问题**   用数组记录以为结尾元素的最长递增子序列的长度。序列a的最长递增子序列的长度为。容易证明满足最优子结构性质，可以递归定义为   1. **使用耐心排序，并证明最小堆数=LIS长度**   耐心排序可以找到耐心游戏的最小堆数，证明**耐心排序的最小堆数 = LIS 的长度**  **引理 1：最小堆数 >= LIS 长度** 证明：假设我们有一个最长递增子序列：。 如果我们知道牌 c(i) 的位置，那么牌 c(i+1) 在哪里？ 首先，c(i+1) 不可能和 c(i) 放在同一堆中，因为 c(i) 下面所有的牌都必须小于 c(i) 本身（游戏规则的设定）。 其次，c(i+1) 不能放在 c(i) 左边的一堆上，否则 c(i) 会放在那堆的上面（耐心排序原理）。 因此，我们知道牌 c(i+1) 一定位于 c(i) 右侧的某个堆中，这意味着LIS的长度至多就是耐心排序的最小堆数。  **引理 2：最小堆数 <= LIS 的长度** 证明：再次假设我们有一个 LIS： 。 首先我们考虑一张牌，c(i)。这张牌必须大于 c(i) 左侧牌堆上的顶牌 c(i-1)，否则 c(i) 将被放入那堆的上面。然后我们来考虑 c(i-1)，这张牌必须大于 c(i-1) 左侧的牌堆顶牌。因此我们可以看到，每堆中必定有一张牌可以连起来并形成递增顺序（但不一定是最长的，这就是为什么最小堆数最多是 LIS 的长度）。  **因此：最小堆数 = LIS 的长度** 通过以上两个引理，我们知道最小桩数必须等于 LIS 的长度才能同时满足两个引理。因此，最小堆数 = LIS 的长度。   1. **通过记录最小结尾元素值来优化**   对该算法进行优化，容易看出i-1到i的循环中，a[i]的值起关键作用。如果a[i]能够扩展到序列 的最长递增子序列的长度，则k=k+1,否则k不变。设 中长度为**k**的最长可以扩展递增子序列的结尾元素是，则当时可以扩展，否则不可以扩展。  如果存在多个长度为k的递增子序列,只需要递增子序列中结尾元素的最小值，因此将作为序列中所有长度为k的递增子序列中的最小结尾元素值。  增强假设后，在i-1到i的循环中，当时，,否则k值不变；当时，如果,则应该将b[1]的值更新为a[i],如果,则二分搜索查找下标j,使得,此时和的值不变，b[j]的值更改为a[i]   1. **输出结果**   将计算结果和执行时间写入文件 result.txt，并在控制台输出相关信息 | | | | | |
| 关键代码 | **1.使用耐心排序优化**   1. **int** lengthOfLIS(vector<ll> a){ 2. **int** size=0; 3. vector<ll> q(a.size()); 4. **int** i,j; 5. **for**(**int** x:a){ 6. i=0,j=size; 7. **while**(i<j){ 8. **int** m=i+(j-i)/2; 9. **if**(q[m]<x)i=m+1; 10. **else** j=m; 11. } 12. q[i]=x; 13. size=max(i+1,size); 14. } 15. **return** size; 16. }   **2.通过记录最小结尾元素值来优化**   1. ll lengthOfLIS(**int** a[]){ 2. b[1]=a[0]; 3. ll k=1; 4. **for**(ll i=1;i<n;i++){ 5. **if**(a[i]>b[k]){ 6. b[++k]=a[i]; 7. }**else**{ 8. b[std::lower\_bound(b,b+k,a[i])-b]=a[i]; 9. /\*          //二分查找算法，用于在已排序的范围内查找第一个不小于给定值的元素 10. ForwardIterator lower\_bound(ForwardIterator first, 11. ForwardIterator last, 12. const T& value); 13. 返回的是一个地址，-b得到该元素所在位置 14. \*/ 15. } 16. } 17. **return** k; 18. } | | | | | |
| 测试结果 | **(1)正确性：**  在洛谷上进行测试，在时间和空间上均满足题目要求，且数据量达到要求，保证了正确性。  **(2)复杂度：**  时间复杂度：  **1.二分查找**：对于每个元素x在数组a中，代码使用二分查找在q数组中找到合适的位置插入x。二分查找的时间复杂度为，其中k是q数组的当前大小，即已找到的递增子序列的长度。  **2.总体时间复杂度**：由于每个元素都需要进行一次二分查找，总体时间复杂度为，其中n是数组a的长度。由于k最多为n，因此时间复杂度最差为。  空间复杂度：  **辅助数组**：代码中使用辅助数组q或者b，其大小与输入数组a相同，因此空间复杂度为 O(n)  (3)构造了不同规模数据集并进行图像化演示：  生成不同规模和大小的数据      **分析：**  **运行时间与数据规模关系**  1.小规模和中规模的数据量较小,运行时间都为0秒，说明针对小规模数据算法可以在极短时间内完成，性能非常出色  2.对于大规模数据，运行时间为0.081秒，展现了算法的高效性，符合其时间复杂度为的特点。  3. lengthOfLIS 算法的时间复杂度是，运行时间的增长与数据规模的对数成正比。  **LIS长度与数据规模关系**  1.随着数据规模的增大，最长递增子序列的长度也显著增长。  2.中规模数据的 LIS 长度为 61，而大规模数据达到了1958，说明数据规模增大后可能出现更长递增子序列的趋势。 | | | | | |
| 实验心得 | 1. 动态规划的有效性与状态选择有关，选择不同的状态其效率不同，可以通过改变dp所存储的内容，从而改变状态空间，在一定程度上可以优化算法效率。 2. 通过这道题，我深入理解了动态规划的应用，通过记录前面状态的选择情况，新状态通过常数级查询完成对新状态下的选择情况，并且有最优子结构的证明，动态规划法很严谨，在很多问题上都可以应用。 | | | | | |
| 实验得分 |  | 助教签名 | |  | | |

**附录：完整代码**

**1.动态规划解决**

|  |
| --- |
| **#include<bits/stdc++.h>**  **using namespace std;**  **#define ll long long**  **const int N=1e6+10;**  **ll n;**  **ll lengthOfLIS(vector<ll> a){**  **int n=a.size();**  **vector<ll> b(n);**  **ll max=0;**  **b[0]=1;**  **for(ll i=0;i<n;i++){**  **ll k=0;**  **for(ll j=0;j<i;j++){**  **if(b[j]>k&&a[i]>a[j])k=b[j];**  **}**  **b[i]=k+1;**  **if(b[i]>max)max=b[i];**  **}**    **return max;**  **}**  **int main(){**  **cin>>n;**  **vector<ll> a(n);**  **for(int i=0;i<n;i++)cin>>a[i];**  **ll ans=lengthOfLIS(a);**  **cout<<ans<<endl;**  **}** |

**2.耐心排序解决**

|  |
| --- |
| **//https://zhuanlan.zhihu.com/p/670544975**  **#include<bits/stdc++.h>**  **using namespace std;**  **#define ll long long**    **int lengthOfLIS(vector<ll> a){**  **int size=0;**  **vector<ll> q(a.size());**  **int i,j;**  **for(int x:a){**  **i=0,j=size;**  **while(i<j){**  **int m=i+(j-i)/2;**  **if(q[m]<x)i=m+1;**  **else j=m;**  **}**  **q[i]=x;**  **size=max(i+1,size);**  **}**  **return size;**  **}**  **int main(){**  **ll n;**  **cin>>n;**  **vector<ll> a(n);**  **for(int i=0;i<n;i++)cin>>a[i];**  **ll ans=lengthOfLIS(a);**  **cout<<ans<<endl;**  **}** |

1. **优化dp实现**

|  |
| --- |
| **//https://zhuanlan.zhihu.com/p/670544975**  **#include<iostream>**  **#include<vector>**  **using namespace std;**  **#define ll long long**  **const int N=5e5+10;**  **int a[N],b[N];**  **int n;**  **ll lengthOfLIS(int a[]){**  **b[1]=a[0];**  **ll k=1;**  **for(ll i=1;i<n;i++){**  **if(a[i]>b[k]){**  **b[++k]=a[i];**  **}else{**  **b[std::lower\_bound(b,b+k,a[i])-b]=a[i];**  **//二分查找算法，用于在已排序的范围内查找第一个不小于给定值的元素**  **/\* ForwardIterator lower\_bound(ForwardIterator first,**  **ForwardIterator last,**  **const T& value);**  **返回的是一个地址，-b得到该元素所在位置**  **\*/**  **}**  **}**  **return k;**  **}**  **int main(){**  **ios::sync\_with\_stdio(false);**  **cin.tie(0);**  **cout.tie(0);**  **cin>>n;**  **for(ll i=0;i<n;i++)cin>>a[i];**  **ll ans=lengthOfLIS(a);**  **cout<<ans<<endl;**  **}** |

**4.数据测试**

|  |
| --- |
| **#include <iostream>**  **#include <vector>**  **#include <fstream>**  **#include <algorithm>**  **#include <ctime>**  **using namespace std;**  **#define ll long long**  **// 求解最长递增子序列的长度**  **int lengthOfLIS(vector<ll> &a) {**  **int size = 0;**  **vector<ll> q(a.size() + 1);**  **int i, j;**  **for (int x : a) {**  **i = 0, j = size;**  **while (i < j) {**  **int m = i + (j - i) / 2;**  **if (q[m] < x)**  **i = m + 1;**  **else**  **j = m;**  **}**  **q[i] = x;**  **size = max(i + 1, size);**  **}**  **return size;**  **}**  **// 生成测试数据**  **void generate\_test\_data(const string &filename, ll size, ll max\_value) {**  **ofstream out(filename);**  **if (!out) {**  **cerr << "Error: Unable to open file " << filename << endl;**  **return;**  **}**  **out << size << "\n"; // 输出数组大小**  **for (ll i = 0; i < size; i++) {**  **out << rand() % max\_value + 1 << " "; // 随机生成 [1, max\_value] 范围内的数字**  **}**  **out.close();**  **cout << "Generated data for " << filename << " with size " << size << ".\n";**  **}**  **// 测试数据**  **void test\_data(const string &filename) {**  **ifstream in(filename);**  **if (!in) {**  **cerr << "Error: Unable to open file " << filename << endl;**  **return;**  **}**  **ll n;**  **in >> n;**  **vector<ll> a(n);**  **for (ll i = 0; i < n; i++) {**  **in >> a[i];**  **}**  **in.close();**  **// 测试算法并计时**  **clock\_t start\_time = clock();**  **ll ans = lengthOfLIS(a);**  **clock\_t end\_time = clock();**  **// 输出结果**  **cout << "Test for " << filename << ":\n";**  **cout << "Longest Increasing Subsequence Length: " << ans << "\n";**  **cout << "Time taken: " << double(end\_time - start\_time) / CLOCKS\_PER\_SEC << " seconds\n\n";**  **}**  **int main() {**  **srand(time(0)); // 设置随机种子**  **// 生成测试数据**  **generate\_test\_data("lis\_small.txt", 10, 100); // 小规模**  **generate\_test\_data("lis\_medium.txt", 1000, 10000); // 中规模**  **generate\_test\_data("lis\_large.txt", 1000000, 1000000); // 大规模**  **cout << "\nData generation complete.\n\n";**  **// 测试小规模数据**  **test\_data("lis\_small.txt");**  **// 测试中规模数据**  **test\_data("lis\_medium.txt");**  **// 测试大规模数据**  **test\_data("lis\_large.txt");**  **return 0;**  **}** |