《算法与程序设计竞赛提升》实验报告

信息 学院 智能科学与技术 专业 2020 级

实验时间 2023 年 1 月 5 日

姓名 ***Steven*** 学号

实验名称 2018年9月CSP真题

实验成绩

一、实验目的

1、练习综合运用编程技法、算法和数据结构等的综合能力。

2、能够灵活运用所学习的程序语言设计、面向对象编程、算法和数据结构综合知识。

3、从实验过程中的问题和收获中进行小结。

二、实验仪器设备及软件

笔记本电脑

Clion 2022.2.3

PyCharm 2022.2.3

三、实验方案

根据五道题目的具体要求，思考解决问题的具体步骤，选择合适的算法和数据结构，并根据不同语言的特点灵活地选择编程语言以解决问题。

四、实验步骤

**1. 201809-1卖菜**

这道题是基础题，不需要什么算法思想，只需要遍历每个元素，依次求出结果即可。

|  |
| --- |
| input() content = list(map(int, input().split())) price = [] for i in range(len(content)):  if i == 0:  price.append(int((content[i] + content[i + 1]) / 2))  elif i == len(content) - 1:  price.append(int((content[i] + content[i - 1]) / 2))  else:  price.append(int((content[i + 1] + content[i] + content[i - 1]) / 3)) print(" ".join([str(i) for i in price])) |

**2. 201809-2 买菜**

这道题也是基础题，分析问题后发现，只需要统计两者用于装车的时间段，然后取交集即可得出结果。这种方法属于蛮力法，虽然本题还可以在此基础上进行剪枝来提高效率，但本题中使用蛮力法即可AC，便没再进行优化。由于上述思路中有集合运算，而Python中自带集合数据类型，故使用Python。

|  |
| --- |
| n = int(input()) H\_loadTime = set() W\_loadTime = set() for \_ in range(n):  a, b = map(int, input().split())  H\_loadTime.update(set(range(a + 1, b + 1))) # range(a+1, b+1)用于生成[a+1, b]的区间 for \_ in range(n):  c, d = map(int, input().split())  W\_loadTime.update(set(range(c + 1, d + 1))) print(len(H\_loadTime & W\_loadTime)) # 先求二者交集，再求出长度，即所求结果 |

**3. 201809-3 元素选择器**

本题的需求就是给输入中每行元素都建立索引，以加快元素选择器的查找速度。由于各个元素有非常明显的父子关系，所以为其建立了树形索引，并使用bfs查找。此外，使用线性或哈希算法为每个label和id建立索引原理上是可行的，但这两种思路在处理“后代选择器”即多个具有父子关系的元素时会非常麻烦，因此没有使用。

此外，本题中使用类来描述树的各节点的属性，易于编写和管理。

|  |
| --- |
| # 标签类 class Node:  def \_\_init\_\_(self, label, id, father\_node, level, line\_num):  self.label = label # 标签名  self.id = id # id字段  self.father\_node = father\_node # 父节点  self.son\_node = [] # 子节点  self.level = level # 记录层级  self.line\_num = line\_num # 所在行   def bfs(selector):  queue = [root]  last = selector[-1] # 最低层级的被查找元素  indexs = []  while queue: # bfs  cur\_node = queue.pop(0)  if (cur\_node.label == last or cur\_node.id == last) and match(selector[:], cur\_node):  indexs.append(cur\_node.line\_num)  for son in cur\_node.son\_node:  queue.append(son)  return sorted(indexs)   # 是否匹配到选择器 def match(selector, cur\_node):  while cur\_node.father\_node and selector:  if cur\_node.label == selector[-1] or cur\_node.id == selector[-1]:  selector.pop()  cur\_node = cur\_node.father\_node  return False if selector else True  # 起始位置  line\_num, query\_num = map(int, input().split())  # line\_num: 行数 query\_num: 查询数 root = Node('', '', None, -1, 0) pre\_node = root # 节点指针j # 接受数据，整理结构化文档 for i in range(line\_num):  readline = input()  level = readline.count('.')  readline = readline[level:].split() # 不同字段恰好用空格分隔，readline现在是[label, id]的列表  level /= 2  label = readline[0].lower() # 标签转小写  id = readline[1] if len(readline) == 2 else '' # 是否有id字段  father = pre\_node  while father.level != level - 1: # 考虑到上一行不一定就是父节点，应迭代向前找  father = father.father\_node  node = Node(label, id, father, level, i + 1)  father.son\_node.append(node) # 构建树  pre\_node = node # 查找结果 for i in range(query\_num):  content = input().split()  selector = [elem if elem[0] == '#' else elem.lower() for elem in content] # 标签转小写  indexs = bfs(selector) # 符合条件的元素所在的行号  print(len(indexs), end=' ')  [print(elem, end=' ') for elem in indexs]  print() |

**4. 201809-4 再卖菜**

首先定义v1，v2两个vector分别代表这排店第一天和第二天的菜价，先读入数据到v2，然后对v1进行DFS

DFS的时候分四种情况：

①DFS(0)，填第一家店的菜价，即v1[0]，所有可能的取值都要考虑到，即1~2 \* v2[0]，然后递归调用DFS

②DFS(1)，这时候第一家店的菜价已经填好了，考虑到两家店求平均值可能有两种情况（整除、余1），故需第二家店的菜价也有两种情况，遍历这两种情况，然后递归调用DFS

③DFS(2) ~ DFS(n - 2)，每次看前两家店第一天的菜价，考虑到三家店求平均值有三种情况（整除、余1，余2），故需填的这家店的菜价有三种情况，遍历这三种情况，然后递归调用DFS

④DFS(n - 1)，即填最后一家店。跟③一样的思路，遍历三种情况。填完最后一家店后，如果最后两家店的平均值刚好等于v2的最后一个值，就说明找到答案了，输出，把find置为true，停止DFS。

|  |
| --- |
| #include<cstdio> #include<vector>  using namespace std;  int n, temp; vector<int> v1, v2, ans;*// v1，v2分别存放第一天和第二天的菜价* bool find = false;  void dfs(int pos) {  if (find) return;*// 如果找到了，直接终止* if (pos == 0) {*// 填第一家店，即v1[0]* for (int i = 1; i <= 2 \* v2[0]; i++) {  v1.push\_back(i);  dfs(1);  v1.pop\_back();  }  } else if (pos == 1) {*// 填第二家店，即v1[1]* for (int i = 0; i <= 1; i++) {  v1.push\_back(2 \* v2[0] + i - v1[0]);  dfs(2);  v1.pop\_back();  }  } else if (pos >= 2 && pos <= n - 2) {  for (int i = 0; i <= 2; i++) {  int temp = 3 \* v2[pos - 1] + i - v1[pos - 2] - v1[pos - 1];  if (temp < 1) continue;  v1.push\_back(temp);  dfs(pos + 1);  v1.pop\_back();  }  } else {*// 填最后一家店，即v1[n - 1]* for (int i = 0; i <= 2; i++) {  if (find) return;*// 最后两家店第一天菜价的平均值等于最后一家店第二天的菜价，满足* if ((v2[pos - 1] \* 3 + i - v1[pos - 2]) / 2 == v2[n - 1]) {  v1.push\_back(v2[pos - 1] \* 3 + i - v1[pos - 2] - v1[pos - 1]);  find = true;  for (int i = 0; i < v1.size(); i++) {*// 输出答案* if (i != 0) printf(" ");  printf("%d", v1[i]);  }  }  }  } }  int main() {  scanf("%d", &n);  for (int i = 0; i < n; i++) {*// 读入第二天的菜价* scanf("%d", &temp);  v2.push\_back(temp);  }  dfs(0);  return 0; } |

**5. 201809-5 线性递推式**

本题给出了递推公式，只需根据递推公式完成求值并输出即可。实现并不难，但本题对于时间要求非常高，很难拿到满分。

|  |
| --- |
| m, l, r = [int(i) for i in input().split()] # m:递推表达式中的阈值 l:起始阶数 r:终止阶数 k = [int(i) for i in input().split()] # k:递推表达式中的系数k1~km k = [0] + k # 为了方便计算，将k1~km的系数放在k[1]~k[m]中 a = [1] # a[0]=1 for n in range(1, r + 1):  i = 1  temp = 0  while i <= min(n, m):  temp += k[i] \* a[n - i]  i += 1  temp = temp % 998244353  a.append(temp) for i in a[l:]:  print(i) |

五、实验结果及分析

**1. 201809-1卖菜**

（此处放代码运行结果和结果分析）

**2. 201809-2 买菜**

（此处放代码运行结果和结果分析）

**3. 201809-3 元素选择器**

（此处放代码运行结果和结果分析）

**4. 201809-4 再卖菜**

（此处放代码运行结果和结果分析）

**5. 201809-5 线性递推式**

（此处放代码运行结果和结果分析）

六、实验总结及体会

本次实验练习了2018年的CSP真题，复习了常用的算法与数据结构，同时了解了真题的题型和特点，此外还通过博客等方式自主学习，编程能力有了进一步的提高。

七、教师评语