華中科技大學

课 程 实 验 报 告

课程名称：串行与并行数据结构及算法

专业班级：计算机本硕博 **2101**

学　　号：U**202115666**

姓　　名：刘文博

指导教师：陆枫

报告日期：**2021.6.15**

计算机科学与技术学院

目录

1 实验一：无重复排序

1.1 题目描述

给出一个具有N个互不相同元素的数组，请对它进行升序排序。第一行为一个整数 N，表示元素的个数。第二行为N个整数，表示这N个元素，保证每个元素均在 int 范围内

1.2 算法流程

本题使用快速排序,做法是用sml的$List.partition$函数将序列分为小于x和大于x的两部分,进行递归求解,然后将两个子序列连接得到原问题的解。

由于$partition$本身就做了比较操作,故递归边界是子问题规模为0时返回一个空序列。

```
val N = getInt();
val a = getIntTable(N);
fun quickSort [] = []
| quickSort(x::xs) =
let val (left, right) = List.partition(fn y => y < x) xs
in
quickSort(left) @ [x] @ quickSort(right) end
val res = quickSort(a);
printIntTable(res);
```

1.3 复杂度分析

$partition$的$Work$为$O(n)$,$Span$为$O(lgn)$,我们每次选择序列的第一个元素为pivot。

最好情况下每次都能将序列划分为相等的两部分

$$W(n) = 2W(\frac{n}{2}) + O(n)$$

$$S(n) = 2S(\frac{n}{2}) + S(lgn)$$

由主定理$W(n) = O(nlgn), S(n) = O(lg^2n)$

最坏情况下序列基本有序,总是将序列划分为一个大小为n-1的子序列和一个空序列

$$W(n) = W(n-1) + O(n)$$

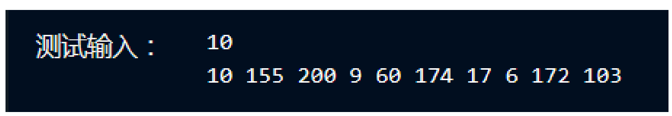
$$S(n) = S(n-1) + S(lgn)$$

由主定理$W(n) = O(n^2), S(n) = O(nlgn)$

在平均情况下,假设输入数组的元素是随机分布的,选择第一个元素作为枢轴的快速排序的平均复杂度分析为$W(n) = O(nlgn), S(n) = O(lg^2n)$

1.4 样例分析

测试样例如下图所示,它是一个长度为10的序列



首先选择 10 作为基准元素，进行划分得到[[9 6] 10 [155 200 60 174 17 172 103]]
然后对于左边和右边的再次进行同样的步骤，左边以 9作为基准元素，右边以 155 作为基准元素，进行划分得到[[6] 9 []] [[60 17 103] 155 [200 174 172]]
继续这样不断递归，直到子序列长度为 0 时终止递归，返回结果，按照左边 + 基准元素 + 右边的方式将其连接并返回
最终返回结果[6,9,10,17,60,103,155,172,174,200]

2 实验二：最短路

2.1 题目描述

给定一个带权无向图，一个源点，权值在边上。计算从源点到其他各点的最短路径。输入格式为：

- 第一行:3个由空格隔开的整数: N, M, T_s 。其中 N 表示结点的数量(从1到 N), M 表示边的数量, T_s 表示源点
- 第 2 到第 $M+1$ 行：描述每条边，每行包含 3 个由空格隔开的整数： R_s, R_e, C_i ,其中 R_s 和 R_e 是两个结点的编号， C_i 是它们之间的边的权值

输出格式为：

- N个整数，表示从源点T_s到各顶点的最短路径长度。如果到某个顶点不连通，对应最短路径长度输出 -1。

2.2 算法流程

本题需要计算从源点到其他各点的最短路径，因为不存在路径长度为负的情况，可以使用 Dijkstra 算法来解决
算法描述如下：

1. 创建一个初始值足够大的长度为N的数组$distance[]$来记录记录源点到所有点的最短距离
2. 创建一个初始值为false的长度为N的布尔数组$visited[]$来记录该点是否已被访问
3. 在未访问的顶点中，找到距离源点最近的顶点，将其标记为已访问
4. 更新与该顶点相邻的未访问顶点的最短路径长度，如果经过当前顶点到达相邻顶点的路径长度比原先的路径长度短，则更新最短路径长度
5. 如果还有节点没有访问,回到步骤3
6. 此时$distance[]$中的值即为源点到各顶点的最短路径长度,输出该值

算法伪代码描述如下：

```
function Dijkstra(graph, source):
distance[source] = 0
for i = 1 to N :
    if i ≠ source:
        distance [i] = ∞
visited[source] = true
for k = 1 to N − 1 :
    minDist = ∞
    u = −1
    for v = 1 to N :
        if visited [v] = false and distance[v] < minDist:
            minDist = distance[v]
            u = v
    if u = −1 :
        break
    visited [u] = true
    for v = 1 to N :
        if visited [v] = false and graph[u][v] > 0 :
            if distance[u] + graph[u][v] < distance[v] :
                distance[v] = distance[u] + graph[u][v]
return distance
```

2.3 复杂度分析

步骤三在未访问的顶点中，找到距离源点最近的顶点需要$O(n)$的$Work$和$O(lgn)$的$Span$
步骤四更新与该顶点相邻的未访问顶点的最短路径长度需要$O(n)$的$Work$和$O(1)$的$Span$
$Dijkstra$是一个典型的串行算法,它的每次计算都依赖前面的结果
那么总的$Work$为$O(n^2)$,$Span$为$O(nlgn)$
用邻接矩阵保存图，用两个长度为N的数组来记录距离和标记访问，则总的空间复杂度为$O(n^2)$

2.4 样例分析

测试样例如下图所示,它是一个包含7个节点，15条边的图，其中源点为5号节点算法过程如下：

```
测试输入：  7 11 5
            2 4 2
            1 4 3
            7 2 2
            3 4 3
            5 7 5
            7 3 3
            6 1 1
            6 3 4
            2 4 3
            5 6 3
            7 2 1
```

1. $distance$为$[\infty, \infty, \infty, \infty, 0, \infty, \infty]$

2. $visited$为$[false, false, false, false, true, false, false]$

3. minDist为未访问的6号节点到源点的距离3，更新$distance$为$[\infty, \infty, \infty, \infty, 0, 3, \infty]$，$visited$为$[false, false, false, false, true, true, false]$

4. 更新与6号节点相邻的未访问顶点的最短路径长度，更新$distance$为$[4, \infty, 7, \infty, 0, 3, \infty]$

5. 还有节点未访问，minDist为未访问的1号节点到源点的距离4，更新$visited$为$[true, false, false, false, tr$

6. 更新与1号节点相邻的未访问顶点的最短路径长度，更新$distance$为$[4, \infty, 7, 7, 0, 3, \infty]$

7. 还有节点未访问，minDist为未访问的7号节点到源点的距离5，更新$distance$为$[4, \infty, 7, 7, 0, 3, 5]$,$visited$为

8. 更新与7号节点相邻的未访问顶点的最短路径长度，更新$distance$为$[4, 6, 7, 7, 0, 3, 5]$

9. 还有节点未访问，minDist为未访问的2号节点到源点的距离6，更新$visited$为$[true, true, false, false, tru$

10. 更新与4号节点相邻的未访问顶点的最短路径长度，$distance$不变，此时我们其实已经得到了最终答案

11. 重复上面的过程，直到所有节点都被访问
12. 此时$distance[]$中的值即为源点到各顶点的最短路径长度,输出$[4, 6, 7, 7, 0, 3, 5]$

3 实验三：最大括号距离

3.1 题目描述

现在给你一个串，你需要找出所有这个串中匹配的子串（一个闭合的串，并且外侧由括号包裹）中最长的那个，输出它的长度。第一行输入一个数N，表示序列的长度，满足N⩽30000。接下来一行输入N个数，表示这个括号序列，0代表左括号，1代表右括号。

3.2 算法流程

我们可以用栈来解决该问题，遍历括号序列，当遇到左括号时，将其位置入栈；当遇到右括号时，从栈顶去除相对应的左括号的位置，计算括号的距离(右括号减左括号加1)，并更新最大距离。最后返回最大距离即可

实现代码如下

```
 (*****Begin*****)
val N = getInt();
val s = ListPair.zip(List.tabulate(N, fn x => x),getIntTable(N));
fun parenDist((pos, x), (stack, max)) =
    if x = 0 then (pos::stack, max)
    else if stack = [] then (stack, max)
    else
        let val top = hd stack
        val tmp = Int.max(max, pos - top + 1)
        in
        (tl stack, tmp) end;
val res = #2(foldl parenDist ([], 0) s);
printInt(res);
(*****End*****)
```

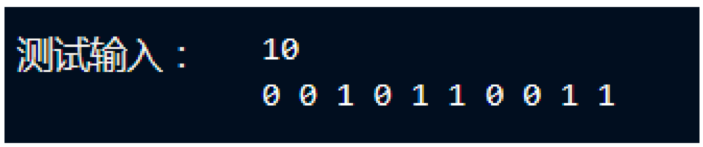
其中pos记录括号的位置，x是当前读到的括号序列的值，stack是一个序列用来实现栈

3.3 复杂度分析

该算法遍历输入序列，故其$Work = Span = O(n)$，使用栈来模式匹配,空间复杂度为$O(n)$

3.4 样例分析

测试样例如下图所示,它是一个长度为10的序列
算法过程如下：



1. 栈为空，max=0
2. 第零个元素是0，pos=0入栈
3. 第一个元素是0，pos=1入栈
4. 第二个元素是1，出栈并计算当前括号的距离为2-1+1=2，更新max=2
5. 第三个元素是0，pos=3入栈
6. 第四个元素是1，出栈并计算当前括号的距离为4-3+1=2
7. 第五个元素是1，出栈并计算当前括号的距离为5-0+1=6，更新max=2
8. 第六个元素是0，pos=6入栈
9. 第七个元素是0，pos=7入栈
10. 第八个元素是1，出栈并计算当前括号的距离为8-7+1=2
11. 第九个元素是1，出栈并计算当前括号的距离为9-6+1=4
12. 得到max=6

4 实验四：天际线

4.1 题目描述

4.2 算法流程

4.3 复杂度分析

4.4 样例分析

5 实验五：括号匹配

5.1 题目描述

5.2 算法流程

5.3 复杂度分析

5.4 样例分析

6 实验六：高精度整数

6.1 题目描述

6.2 算法流程

6.3 复杂度分析

6.4 样例分析

7 实验七：割点和割边

7.1 题目描述

7.2 算法流程

7.3 复杂度分析

7.4 样例分析

8 实验八：静态区间查询

8.1 题目描述

8.2 算法流程

8.3 复杂度分析

8.4 样例分析

9 实验九：素性测试

9.1 题目描述

9.2 算法流程

9.3 复杂度分析

9.4 样例分析