华中科技大学

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：ACM1901

学　　号：U201915035

姓　　名：邹雅

指导教师：陆枫

报告日期：2020.12.23

计算机科学与技术学院

目录

1. 无重复排序

1.1 题意描述

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

1.2 算法分析

采用快速排序算法，如下所示：

```
fun quick [] = []
|quick [x] = [x]
|quick (a::bs) =
let fun partition(left,right,[]) : int list =
       (quick(left)) @ (a::quick(right))
    | partition(left,right,x::xs) =
      if x<a then partition(x::left,right,xs)
      else partition(left,x::right,xs)
    in partition([],[],bs) end;
```

选取枢纽元，把比枢纽元小的数放到 left 中再递归调用进行排序，把比枢纽元大的数放到 right 中再递归调用进行排序，枢纽元放在两个排好序的 List 中间。由此可以实现整个排序。

1.3 复杂度分析

时间复杂度：

先对 work 进行分析如下。

由算法可以得出递推关系为$T(N) = T(i) + T(N - i - 1) + cN$,其中 i 的大小由枢纽元的选取好坏决定。我们对其平均情况进行分析，可得以下(1)式

$$T(N) = \frac{2}{N}\left[\sum_{j=0}^{N-1} T(j)\right] + cN$$

如果左右同乘以 N，可得以下(2)式

$$NT(N) = 2\left[\sum_{j=0}^{N-1} T(j)\right] + cN^2$$

再用 N-1 代换 N，可得(3)式

$$(N-1)T(N-1) = 2\left[\sum_{j=0}^{N-2} T(j)\right] + c(N-1)^2$$

用(2)式减去(3)式并化简整理可得以下(4)式

$$\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2c}{N+1}$$

此式子通过叠缩，最后能得到$T(N) = O(NlogN)$

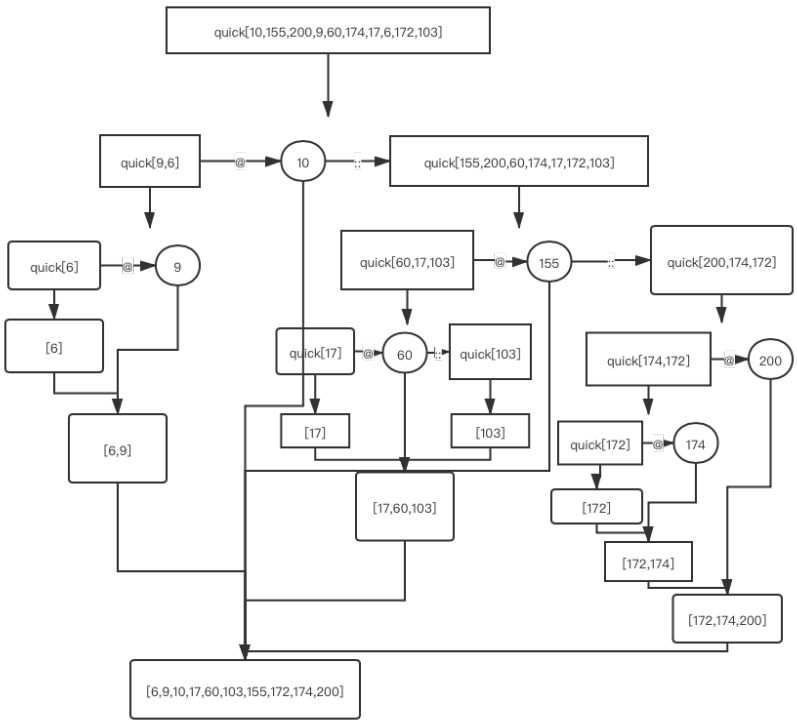
由算法可见，没有并行的部分在进行，所以$work = span = O(nlogn)$.

空间复杂度：

快速排序递归调用的过程中需要排序 left 和 right 部分而产生的空间平均为$O(logn)$,故空间复杂度为$O(logn)$.

1.4 测试样例

输入为[10,155,200,9,60,174,17,6,172,103]十个数。



最后输出为[6,9,10,17,60,103,155,172,174,200].

2. 最短路

2.1 题意描述

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。

2.2 算法分析

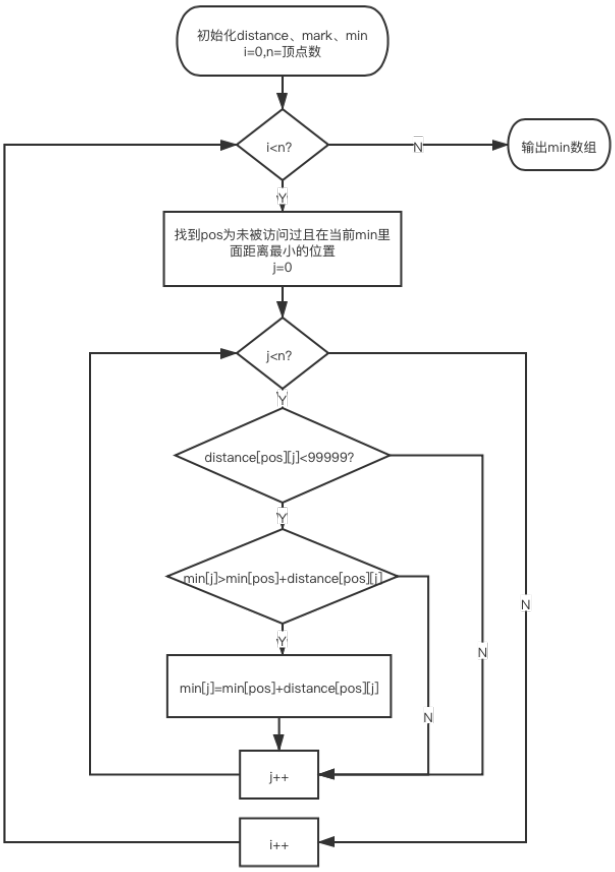
采用 Dijkstra 算法。

2.2.1 自然语言描述

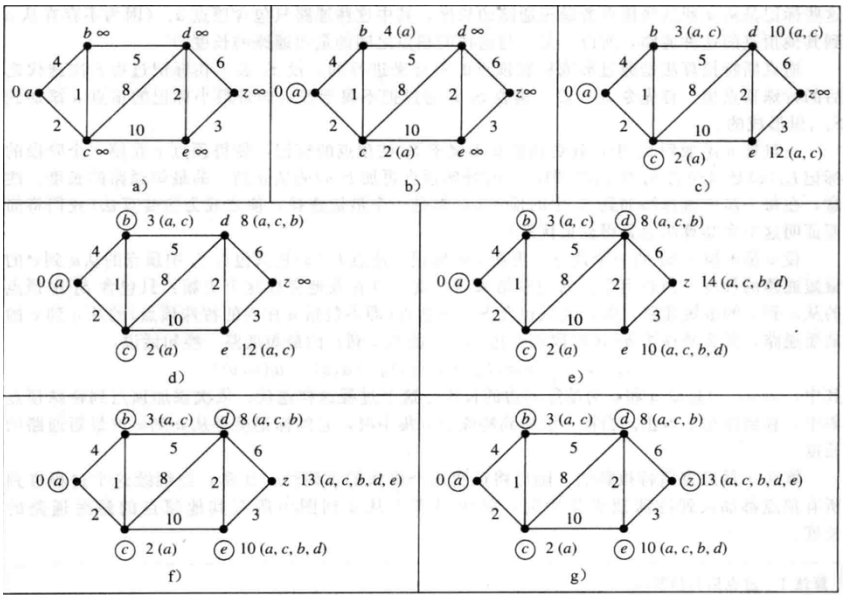
用一个二维数组 distance 来保存图的邻接矩阵：当前节点到自己为 0，不连通为 99999(∞)，连通的两个顶点之间为相应的权值；若有重复的边输入，则取更短的保存。用一个数组 mark 来标记节点是否被访问过，初始化只有起始节点被访问过；另一个数组 min 来记录起始节点到相应节点的当前最短值，初始化为起始节点到相应节点的直接距离。

初始化完成后开始对一个个遍历节点，每次找到一个未被访问过且当前距离最短的节点去访问，记录为中间值 pos。在当前访问的这个 pos 节点下，判断其余节点是否能通过 pos 找到更短的距离，如果能则更新 min 数组。当所有节点都被访问过后，算法结束。若此时 min 数组中的值仍保持在 99999，则说明两点不连通，输出-1；否则输出权值。

2.2.2 流程图



2.2.3 图解 Dijkstra



2.3 复杂度分析

时间复杂度：

由算法流程图可见其中的二重循环占算法主要地位，故复杂度为$O(n^2)$.

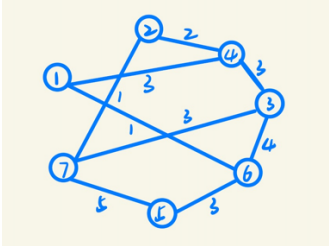
由于过程中没有并行成分，可见$work = span = O(n^2)$.

空间复杂度：

算法需要使用邻接矩阵存图，其空间复杂度为$O(n^2)$.而还需要附加的数组 mark 和 min，其复杂度为$O(n)$.综合来看其空间复杂度为$O(n^2)$.

2.4 测试样例

输入的图如下，选取的源点为 5：



邻接矩阵 distance 如下：

0	99999	99999	3	99999	1	99999
99999	0	99999	2	99999	99999	1
99999	99999	0	3	99999	4	3
3	2	3	0	99999	99999	99999
99999	99999	99999	99999	0	3	5
1	99999	4	99999	3	0	99999
99999	1	3	99999	5	99999	0

迭代过程如下：

mark([0,0,0,0,1,0,0])	min([99999, 99999, 99999, 99999,0,3,5])
[0,0,0,0,1,1,0]	[4,99999,7,99999,0,3,5]
[1,0,0,0,1,1,0]	[4,99999,7,7,0,3,5]
[1,0,0,0,1,1,1]	[4,6,7,7,0,3,5]
[1,1,0,0,1,1,1]	[4,6,7,7,0,3,5]
[1,1,1,0,1,1,1]	[4,6,7,7,0,3,5]
[1,1,1,1,1,1,1]	[4,6,7,7,0,3,5]

最终输出为 4 6 7 7 0 3 5.

3. 最大括号距离

3.1 题意描述

我们定义一个串 s 是闭合的，当且仅当它由'('和')'构成，而且满足以下条件之一：

空串：这个串为空

连接：这个串由两个闭合的串连接构成

匹配：这个串是一个闭合的串外面包裹一个括号构成的

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。

3.2 算法分析

找出当前所给串从头开始遍历时的最长匹配子串的算法如下所示：

```
fun matchAndCount(tag,count,max,[]) = max
|matchAndCount(tag,count,max,x::xs) =
 if tag = ~1 then Int.max(max,count)
 else if x = 1 then matchAndCount(tag-1,count+1,max,xs)
 else (if tag = 0 then
       (if count+1>max then matchAndCount(tag+1,0,count+1,xs)
       else matchAndCount(tag+1,0,max,xs))
       else matchAndCount(tag+1,count+1,max,xs)
 );
```

算法递归求解，通过 tag 参数标记当前是否匹配，count 参数记录当前正在遍历的匹配长度，max 记录已经遍历过的最大匹配长度。

但某些最大匹配串并不一定从第一个括号开始匹配，所以要对一个串进行递归，每次从第一个、第二个……直至最后一个开始应用 matchAndCount 函数，每次应用时都比较传入一个更大的匹配长度，函数如下所示：

```
fun iterateMatch(max,[]) = max
 |iterateMatch(max,x::xs) =
  let val curr = matchAndCount(0,0,0,xs)
  in (if curr > max then iterateMatch(curr,xs)
    else iterateMatch(max,xs)) end;
```

3.3 复杂度分析

假设括号串的长度为 n.

时间复杂度：

每次应用 matchAndCount 函数的时间复杂度都是线性的，故总共的时间复杂度为

$$\sum_{i=1}^{n} i = O(n^2)$$

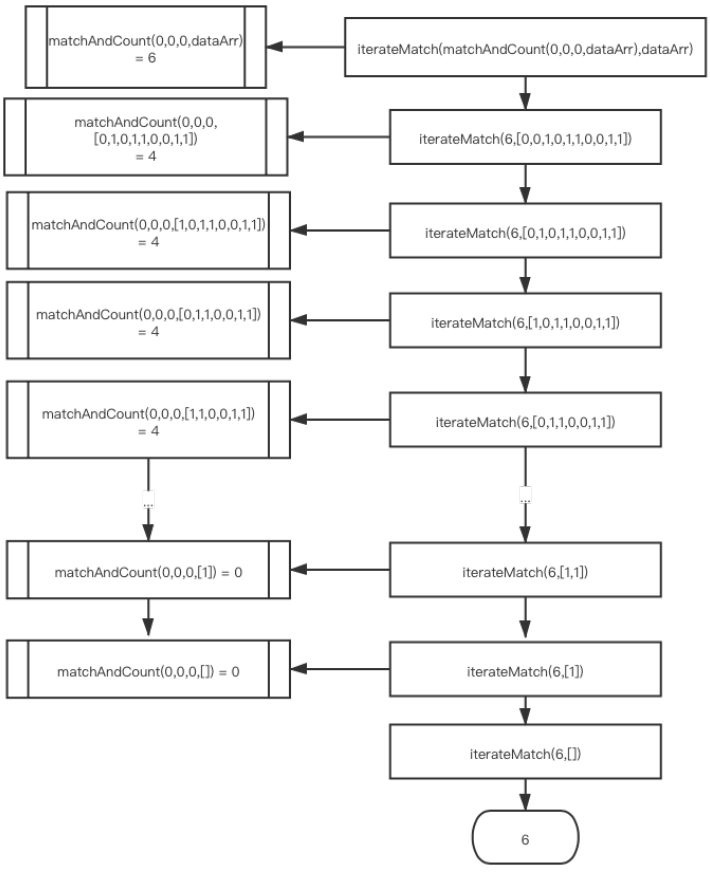
在当前所给算法中，$work = span = O(n^2)$.而假如每次调用 matchAndCount 函数是并行的，再对每个 max 进行比较，这会附加$O(n)$的空间复杂度，但是却能把 span 缩减为$O(n)$。可以用 List 的 drop 和 tabulate 操作实现，在此没有去实现。

空间复杂度：

如上的递归调用并不会造成空间的累积，故而其空间复杂度为$O(1)$.

3.4 测试样例

假设输入的括号序列为(( ) ( )) (( ))，即[0,0,1,0,1,1,0,0,1,1].



输出为 6.

4. 天际线

4.1 题意描述

城市的天际线是从远处观看该城市中所有建筑物形成的外部轮廓。现在，假设您获得了城市风光照片，照片上显示的所有建筑物的位置和高度，请编写一个程序以输出由这些建筑物形成的天际线。

第一行输入一个整数 N，表示建筑的数量。 接下来 N 行，每行输入 3 个整数 li,hi,ri，表示每个建筑物的几何信息，其中 li 和 ri 分别是第 i 座建筑物左右边缘的 x 坐标，hi 是其高度，您可以假设所有建筑物都是在绝对平坦且高度为 0 的表面上的完美矩形。输出各个关键点的 x 和 y 坐标。关键点是水平线段的左端点；请注意，最右侧建筑物的最后一个关键点仅用于标记天际线的终点，并始终为零高度。此外，任何两个相邻建筑物之间的地面都应被视作天际线轮廓的一部分。

4.2 算法分析

采用分治法，进行如下分析：

1.定义基本情况：第一种基本情况是建筑物列表为空，此时天际线也是一个空列表；第二种就是列表中只有一栋楼，天际线是该建筑的左上顶点和右下顶点。

2.拆分问题：与归并排序类似，将列表拆分成两部分（用 List.take 和 List.drop 进行），对每一部分分别求解天际线。

3.合并问题：将左右两个天际线中的 x 坐标取出从左到右进行比较，每次取较小的 x 坐标，取当前对应高度与另一边的当前高度的最大值为对应高度，若天际线列表为空或者这个高度与前一个天际线高度不同，则加入天际线列表，否则舍弃。直至最后若还有一方有剩余，则直接加入天际线列表。

4.递归求解。

4.3 复杂度分析

设建筑物数目为 N。

时间复杂度：

其递推关系式为 $T(N) = 2T\left(\frac{N}{2}\right) + 2N$，其求解结果为 O(NlogN)。故而，$work = span = O(nlogn)$.

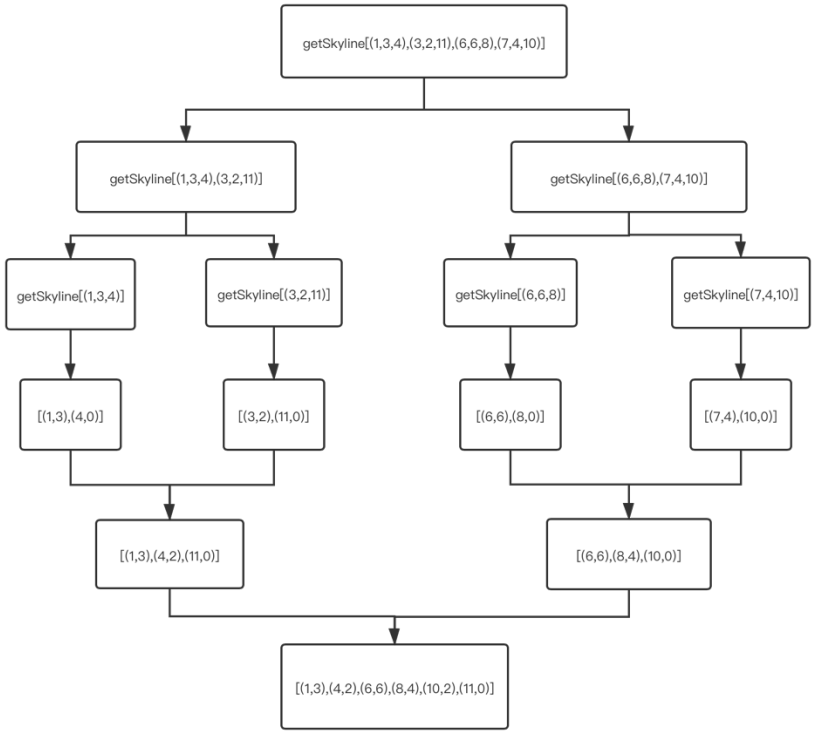
空间复杂度：

算法需要额外 O(N)的空间来保存结果。

4.4 测试样例

样例输入:(1,3,4),(3,2,11),(6,6,8),(7,4,10).

算法主体函数为 getSkyline(x:List),合并函数为 merge(x:List,y:List,lh,rh,h),图中为简便省略了 merge 函数的调用.



样例输出为:(1,3),(4,2),(6,6),(8,4),(10,2),(11,0).

5. 括号匹配

5.1 题意描述

给定一个括号序列，判断它是否是匹配的。注意( ) ( )在本题也当做匹配处理。

5.2 算法分析

用 dataArr 数组保存括号序列（0 表示左括号，1 表示右括号），算法如下所示（其中如果匹配，输出 1，否则输出 0）：

```
fun matchParens(a,[]) = a
|matchParens(a,s::xs) =
if a = ~1 then ~1
else if s = 1 then matchParens(a-1,xs)
else matchParens(a+1,xs);
fun judge(a) =
if a = 0 then 1
else 0;
printInt(judge(matchParens(0,dataArr)));
```

初始化 a 输入为 0，若遇到右括号则减 1，若遇到左括号则加 1，当 a 已经减为-1 时直接输出-1。最后进行判断，如果 a 为 0，即完全匹配，输出 1，否则不匹配。

5.3 复杂度分析

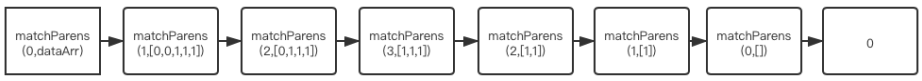
设括号序列的长度为 n.

时间复杂度：算法顺着括号序列一个个进行判断，故其$work = span = O(n)$.

空间复杂度：算法递归调用并没有产生额外的空间，故为$O(1)$.

5.4 测试样例

输入序列为((()))即为 0 0 0 1 1 1.



故该序列为完全匹配，输出 1.

6. 高精度整数

6.1 题意描述

给定两个任意精度的整数 a 和 b，满足 a≥b，求出 a+b, a−b, a×b 的值。

6.2 算法分析

输入用 List 保存整数 a 和 b，并进行翻转操作，使之从低位到高位显示。

定义 clearZ 函数，清除数字串头部可能出现的 0；定义 aZero 函数，在数字串低位前加入相应个数的 0 串。

相加操作的函数对应如下：

```
fun add([],[],last) = [last]
|add(x::xs,[],last) = ((x+last) mod 10)::add(xs,[],(x+last)div 10)
|add([],y::ys,last) = ((y+last) mod 10)::add(ys,[],(y+last)div 10)
|add(x::xs,y::ys,last)  =  ((x+y+last)mod  10)::add(xs,ys,(x+y+last)div
10);
```

相减操作的函数对应如下：

```
fun minus([],[],last) = []
|minus(x::xs,[],last) = if x < last then (10 + x - last)::minus(xs,[],1)
else (x - last)::minus(xs,[],0)
|minus(x::xs,y::ys,last) = if (x - last) < y then (10 + x - last -
y)::minus(xs,ys,1) else (x - last - y)::minus(xs,ys,0);
```

相乘操作以下两个函数构成：

```
fun singleToM([],t,cur) = [cur]
|singleToM(x::xs,t,cur)=((x*t+cur)mod
10)::singleToM(xs,t,((x*t+cur)div 10));
fun multiply(x,[],value) = [0]
|multiply(x,y::ys,value) =
   let val tmp = aZero(value) @ singleToM(x,y,0)
   in add(tmp,multiply(x,ys,value+1),0) end;
```

三个操作定义的函数都采用在函数参数中传递上一次操作产生的进位/借位的思想，而产生的最终结果需要进行翻转和 clearZ 操作。

6.3 复杂度分析

设整数 a 有 n 位，整数 b 有 m 位。

时间复杂度：

对于加法和减法，其时间复杂度都为 O(n)。因为每递归调用一次函数，都会使两个整数各减少一位，而 a≥b，故而取更大的位数为 n。

对于乘法，每次取 b 的一位数先进行 singleToM，再进行相加操作。每一次调用 singleToM 的时间复杂度是 O(n)，调用了 m 次，故为 O(mn).再进行 m 个数的相加，每一次加法的时间复杂度为 O(n)，故为 O(mn).综上所述，乘法的时间复杂度为 O(mn).

$$work = span = O(mn).$$

而后分析思考，若调用 singleToM 可以并行，那 span 为 O(n)；其后加法可以两两进行，span 缩减为$O(nlogm)$,综合其 span 为$O(nlogm)$.

空间复杂度：

对于加法和减法，由于递归造成的空间累积其空间复杂度都为 O(n).

对于乘法，由于递归调用，无法直接进行加法计算，每一次都要保存一位相乘之后的中间结果，故其空间复杂度为 O(mn).

6.4 测试样例

输入 133 和 100.保存为 a=[3,3,1],b=[0,0,1].

加法的递归调用返回[3,3,2]，再进行翻转后得到 233；减法递归调用返回[3,3,0]，再进行翻转后并去除串前的 0，得到 33；乘法得到的中间结果分别为[0,0,0],[0,0,0,0],[0,0,3,3,1]，再进行相加得到[0,0,3,3,1]，即 13300.

7. 割点和割边

7.1 题意描述

给定一个无向无权连通图，请求出这个图的所有割点和割边的数目。

7.2 算法分析

采用 Tarjan 算法求解割点割边。

算法采用深度优先搜索遍历每个节点，遍历到某顶点的次序称之为该节点的时间戳。设$dfn[x]$数组保存 DFS 中 x 点的时间戳；设$low[x]$保存该节点不经过父节点能回溯到的最早的时间戳，初始化时为 x 点的时间戳。

x 是割点有两种情况：x 是 DFS 的起始点且在相对于 DFS 的生成树中有两个或两个以上的儿子；或者是 x 不是起始点且$low[x的儿子] >= dfn[x]$。x—y 是割边的情况为，假设 x 是 y 的 DFS 父节点，那么当$low[y] > dfn[x]$时，该边为割边。以上分析为算法的基础。

7.3 复杂度分析

设图有 n 个顶点，e 条边。

时间复杂度：

Tarjan 算法对图进行 DFS 遍历，对每个顶点都遍历一次，而在对每个顶点访问时还会进行一个 n 次循环，所以其 work 为$O(n^2)$.

由于在对 dfs 函数进行调用时，采用了如下的方式：

```
fun main(flag:int) = if(Array.sub(dfn,flag)=0) then dfs(flag) else ();
List.tabulate(n,main);
```

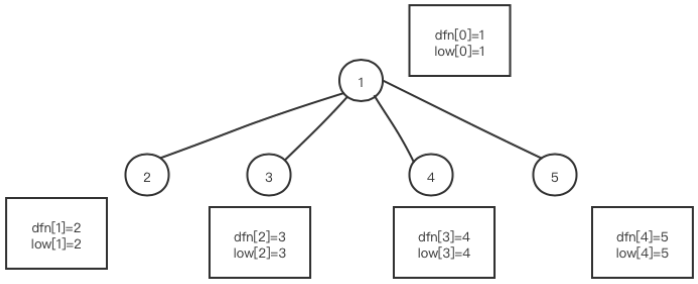
由于 dfs 函数内部仍有递归，故很难分析每次调用 dfs 的 span.可知最坏的情况为$O(n^2)$，而最好的情况为$O(n)$.在全图联通的情况下，span 为$O(n^2)$.

空间复杂度：

算法使用邻接矩阵进行存图，所以其空间复杂度为$O(n^2)$.

7.4 测试样例

样例输入 5 个顶点和 4 条边，分别为 1—2、1—3、1—4 和 1—5.


dfn[0]=1
low[0]=1
1
2
3
4
5
dfn[1]=2
low[1]=2
dfn[2]=3
low[2]=3
dfn[3]=4
low[3]=4
dfn[4]=5
low[4]=5

8. 静态区间查询

8.1 题意描述

给定一个长度为 N 的数列，和 M 次询问，求出每一次询问的区间内数字的最大值。

8.2 算法分析

输入一个 n 个数的数组 list，输入查询的次数 m.

先构建一个二维数组，在第 i 行第 j 列保存从第 i 个数到第 j 个数中的最大值；在每次查询时，直接找到相应的数输出即可。

构建二维数组时，先将该数组都初始化为 0，然后将主对角线上的数都初始化为 list 中的数，其后第 i 行第 j 列(i<j)中的数由第 i 行第 j-1 列和 list 中的第 j 个数的最大值得到。

输出的代码如下：

```
fun result _ =
    let val x = getInt() - 1 val y = getInt() - 1
    in printInt(Array2.sub(data,x,y)) end;
List.tabulate(m,result);
```

8.3 复杂度分析

时间复杂度：使用了 List.tabulate 进行输出，其 work 为$O(m)$,span 为$O(1)$.

空间复杂度：由于构建了一个二维数组保存结果，其空间复杂度为$O(n^2)$

8.4 测试样例

输入 n=8,m=8.数组为[9,3,1,7,5,6,0,8],构建的二维数组如下所示：

9	9	9	9	9	9	9	9
0	3	3	7	7	7	7	8
0	0	1	7	7	7	7	8
0	0	0	7	7	7	7	8
0	0	0	0	5	6	6	8
0	0	0	0	0	6	6	8

0	0	0	0	0	0	0	8
0	0	0	0	0	0	0	8

八次查询分别如下：

1 6	9
1 5	9
2 7	7
2 6	7
1 8	9
4 8	8
3 7	7
1 8	9

9. 素性测试

9.1 题意描述

给定一个数 N，判断它是否是素数。

9.2 算法分析

已知费马小定理：如果 p 是一个质数，而整数 a 不是 p 的倍数，则$a^{p-1} \equiv 1(mod\ p)$.

利用快速幂算法求出来计算出$a^{p-1}(mod\ p)$，如果这个数为 1，则 p 有可能为质数；当计算的个数增加，则 p 为质数的概率增加。此时选取a就显得很重要，我在此选用了 11、103 和 10009 三个数，如果三者同时能通过测试，则输出 True，但如果其中一个不满足则输出 False.这种算法存在偶然性。在算法之前，对 0、1 直接输出 False，对 2 直接输出 True.

9.3 复杂度分析

设输入的数为 n.

时间复杂度：

我们已知快速幂算法的时间复杂度为 O(log$n$),则本题的work = span = O(log$n$).

空间复杂度：$O(1)$

9.4 测试样例

输入 101.

快速幂计算$11^{100}(mod\ 101)$、$103^{100}(mod\ 101)$和$10009^{100}(mod\ 101)$，得出三个式子都等于 1，输出 True.