目录

1 无重复排序

1.1 本关任务

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 解法

我们可以采用归并排序算法解决。将数据分为即

Algorithm 1: MergeSort

```
1  MergeSort(⟨a_n⟩) =
2  if |a_n| = 1 then
3  |  a_1
4  else
5  |  let
6  |  |  L ⟵ MergeSort(⟨a_1, a_2, ..., a_{(n+1)/2}⟩);
7  |  |  R ⟵ MergeSort(⟨a_{(n+1)/2}, ..., a_n⟩);
8  |  in
9  |  |  Merge(L, R)
10 |  end
11 end
```

复杂度为

$$W = O(n \log n) \quad S = O(\log^2 n)$$

1.3 分析与证明

```
fun merge([], ys) = ys : int list
  | merge(xs, []) = xs
  | merge(x::xs, y::ys) =
    if x < y then x::merge(xs, y::ys)
             else y::merge(x::xs, ys);
```

首先 merge 函数是对两个已经排序好的数字串进行排序，这样可以分为两种情况。第一种是两个数字串都不为空，那么直接比较两串头元素，较小的放在返回数字串的开头，再对剩下的数字串进行迭代操作。如果其中已经有一个数字串为空，那么直接返回剩下的所有数字串。因为每次返回的都是两个串中最小的元素，因此 merge 函数得到的数字串必定是从小到大有序排列的。

```
1  fun mergesort xs =
2      let fun sort(0, xs) = ([], xs)
3            | sort(1, x::xs) = ([x], xs)
4            | sort(n, xs) =
5                  let val (l1, xs1) = sort((n+1) div 2, xs)
6                      val (l2, xs2) = sort(n div 2, xs1)
7                  in (merge (l1,l2), xs2)
8                  end
9          val (l, _) = sort (length xs, xs)
10     in l end;
```

mergesort 操作本质上是首先利用 sort 函数将数字串分为两半，再利用 mergesort 进行两数字串的排序。sort(l, x) 函数的两个参数分别是分出的左数字串的长度，和需要切分的数字串 x，最后返回的就是排序好的字符串。在第 6 句中返回的 x2 已经是空值，而 l2 是已经排序好的子串。

证明：显然，当 sort 函数第一个参数值等于需要排序数字串的长度时，返回值的第二个值就为空，即返回的第一个值包含了所有需要排序的数。并且已经在前面证明了 merge 函数的正确性，因此只需证明 l1，和 l2 是有序数列。

sort(n, x) 返回值 (l, y)，l 是 x 的前 n 项的有序值，y 是 x 的剩余项。当排序序列长度为 1 或 2 时显然成立。

假设，长度小于等于 k 的串经过 sort 函数的处理可以得到的是有序的串。

对于 $k+1$ 长度的串来说，将会先分成两个长度为 (k+2) div 2 和 (k+1) div 2 的串排序，然后再通过 merge 操作归并。因此得到的 $k+1$ 长度的串一定是有序的，证毕。

复杂度分析：在进行归并操作时需要的 $W(n) = n$，$S(n) = logn$，因此得到递推公式 $W(n) = 2W(\frac{n}{2}) + O(n)$,$S(n) = S(\frac{n}{2}) + O(logn)$，因此有 $W(n) = nlogn$, $S(n) = log^2 n$。

1.4 样例输入与输出

样例输入：

10

10 155 200 9 60 174 17 6 172 103

样例输出：

6 9 10 17 60 103 155 172 174 200

2 最短路

2.1 本关任务

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 解法

可以采用 Dijkstra 算法解决该问题。

利用二维数组保存两点之间的路径长度，即 $A[i][j]$ 即是编号为 $i-1$ 和 $j-1$ 点之间的路径长度。若无给定长度，则用 ∞ 来表示无法两点之间无法到达。创建一个一维数组 dis，用以储存源点到每个点的距离。并且集合 B 为中只有源点 a。

(1) 在 dis 数组中寻找一个长度最短的值 dis[k]，并且保证 k 点不在集合 B 中。

(2) 再对每个不属于 B 集合中的点进行长度收缩: 如果 dis[i] 的值 dis[k]+A[a][k] 的大小，就将 dis[i] 的值更新为 dis[k]+A[a][k]。将 k 也加入集合 B 中。

(3) 重复上述步骤，直至集合 B 中有所有点。得到的数组 dis 的值就是源点到所有对应点的最短路径值。

在最后输出最短路之前，再将所有长度为 99999（表示 inf）的数据转化成 1 说明没有此路径。

2.3 分析与证明

分析：Dijkstra 算法使用的迭代不超过 $n-1$ 次，在每次迭代的运算中，即比较与更新一共不超过 $2(n-1)$ 次，因此 $W = O(n^2)$。每一次的迭代都不能并行运算，

但是在每一次的迭代中的比较与收缩长度操作可以同时进行，因此 $S = O(n) + 1$。

证明：设源点 O 到每一个顶点的距离为 L(V)，P 是一个顶点集，在最初过程中，P 集合中只有源点。对于任意的 $V \in P$，则 $L(V)$ 是最短路径长度。对于任意的 V 不属于 P，最短路径 $L(V)$ 除 V 点经过的所有点都在 P 集合内。

可以用归纳法进行证明上面的结论：

当 $|P| = 1$ 时，$L(O) = 0$，显然满足。

假设 $|P| = k$ 时，满足上述条件。根据算法的 (3)，找出不在 P 中 L(U) 最小的 U 点。

再假设如果源点 O 到 U 的最短路径中有访问 P 之外（除 U）的顶点，设其中一点为 Q，则有 $L(Q) + PATH(Q,U) = L(U)$，即有 $L(Q) < L(U)$，与 L(U) 是不在 P 中到源点最短的距离这一前提矛盾，因此假设不成立。因此源点 O 到 U 的最短距离经过的点都在集合 P 中。

因此当 $|P| = k + 1$ 时，假设也成立。

因此这个算法是正确的。

2.4 样例输入与输出

样例输入：

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

样例输出：

4 6 7 7 0 3 5

3 最大括号距离

3.1 本关任务

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 解法

现在采用一个 stack 函数来储存括号的栈。

当栈为空时，判断下一个要入栈的括号的类型：如果是右括号，则必定不会有与之匹配的括号，直接丢弃该右括号；如果时左括号，就将左括号压栈。并在每次栈为空时更新当前最大的匹配括号长度。

当栈不为空时，如果栈顶元素是左括号，即将压栈的括号是右括号，就直接弹出栈顶元素，并增加当前括号距离 2。否则将新元素压栈。

由于这种解法只能够解决括号串整体匹配和右括号多于左括号的情况，因此需要将原来的括号串每个元素置换并颠倒，得到新的最大匹配长度不变但括号冗余情况完全相反的括号串。两个串同时经过 stack 函数的处理，得到的最小的值就是括号匹配的最大的距离。

关键函数 stack 的代码如下：

```
1 fun stack([], _, maxpair, tmppair) = Int.max(maxpair, tmppair)
2   | stack(x::xs, [], maxpair, tmppair) =
3         if(x=0)
4             then stack(xs, [x], Int.max(maxpair, tmppair), 0)
5             else stack(xs, [], Int.max(maxpair, tmppair), 0)
6   | stack(x::xs, y::ys, maxpair, tmppair) =
7         if((x=1) andalso (y=0))
8             then stack(xs, ys, maxpair, tmppair+2)
```

```
9 |             else stack(xs, x::y::ys, maxpair, tmppair);
```

3.3 分析与证明

字符串元素的置换和反转都需要 $O(n)$ 的 W 和 $O(1)$、$O(n)$ 的 S。

stack 函数只需要扫描整个 list 一遍，因此串行需要的 W 是 $O(n)$，S 也是 $O(n)$。

3.4 样例输入与输出

样例输入：

50 1 0 1 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 1 1 0 0 0

样例输出：

6

4 天际线

4.1 本关任务

城市的天际线是从远处观看该城市中所有建筑物形成的轮廓的外部轮廓。现在，假设您获得了城市风光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。

每个建筑物的几何信息用三元组 $[L_i\ H_i\ R_i]$ 表示，其中 L_i 和 R_i 分别是第 i 座建筑物左右边缘的 x 坐标，H_i 是其高度。可以保证 $0 \leq L_i,\ R_i \leq INT\_MAX$, $0 < Hi \leq INT\_MAX$ 和 $Ri - Li > 0$。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。

输出是以 $[[x_1, y_1], [x_2, y_2], [x_3, y_3], ...]$ 格式的“关键点”的列表，它们唯一地定义了天际线。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为天际线轮廓的一部分。

4.2 解法

4.3 分析与证明

4.4 样例输入与输出

5 括号匹配

5.1 本关任务

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 解法

5.3 分析与证明

5.4 样例输入与输出

6 高精度整数

6.1 本关任务

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 解法

6.2.1 加法的计算

将两个数列进行翻转，即将两个数的最后一位对齐，做相应的加法。
如果两个数相加大于 9，就将 c 置为 1，在下一位上加上 1。

6.2.2 减法的计算

减法的计算与加法同理，也是将最后一位对齐，从低位开始相减。并用 c 来表示在下一位是否要退位，如果 $a-b<0$。算法如下。

Algorithm 2: Addition

$a = (a_1 a_2 a_3 ... a_n)_{10}$
$b = (b_1 b_2 b_3 ... a_m)_{10}$
带入该函数时需要将 a 和 b 进行翻转。
Addition($a,\ b,\ result,\ c,\ i$) =
if ($a_i\ =\ 0\ andalso\ b_i\ =\ 0$) **then**
    $result_i\ =\ c$
else
    let
        $result_i = a_i + b_i + c$
        $c = (a_i + b_i + c) \bmod 10$
    in
        Addition($a,\ b,\ result,\ c,\ i + 1$)
    end
end

Algorithm 3: subtraction

```
1  a = (a_1 a_2 a_3 ... a_n)_10
2  b = (b_1 b_2 b_3 ... a_m)_10
3  带入该函数时需要将 a 和 b 进行翻转。
4  Subtraction(a, b, result, c, i) =
5  if (a_i = 0 andalso b_i = 0) then
6  |  result_i = c
7  else
8  |  let
9  |  |  if a_i − b_i + c < 0 then
10 |  |  |  c = (a_i - b_i + c)
11 |  |  |  result_i = 10 + a_i − b_i + c
12 |  |  else
13 |  |  |  result_i = a_i − b_i + c
14 |  |  |  c = 0
15 |  |  end
16 |  in
17 |  |  Addition(a, b, result, c, i + 1)
18 |  end
19 end
```

6.2.3 乘法的计算

先将多位数相乘转换为多个一位数与多位数相乘的乘法。

与上述加法和减法的操作类似，从最末位开始相乘，并用 c 来记录下一位的进位数。

对于第 k 位数完成乘法运算后还要在末尾加上 $k-1$ 个零。

再将所有得到的结果相加。

```
1 fun onemultiply(a, [], c) = [c]
2   | onemultiply(a, b::bs, c) =
3         if (a * b + c> 9) then ((a*b+c)mod 10)::onemultiply(a, bs, (a*b
              +c) div 10)
4                           else (a*b+c)::onemultiply(a, bs, 0);
5 fun multiply([], bs) = [0]
6   | multiply(a::ass, bs) =
7         Add(onemultiply(a, bs, 0), 0::multiply(ass, bs), 0);
```

6.3 分析与证明

对于加法来说:n 和 m 相加,需要线性扫描每一个数,因此 $W = O(max(|n|, |m|)+1)$。对于最优并行算法来说，可以用 map 操作对对应位数的两个数字相加，并记录每一位需要进位的个数。再将进位数相应相加。重复操作，直到所有每一位的进位数都为 0。所有的操作都是并行处理。

在第一次各位数相加时，

因此 $S = O(logn)$。

减法与加法同理，有同样的串行复杂度 W。

6.4 样例输入与输出

样例输入：

10 2 3 3 3 3 3 3 3 3 3 10 2 3 3 3 3 3 3 3 3 3
样例输出：
4 6 6 6 6 6 6 6 6 6 0 5 4 4 4 4 4 4 4 4 2 8 8 8 8 8 8 8 8 9

7 割点与割边

7.1 本关测试

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 解法

利用 Tarjan 算法进行实现。定义两个数组，dfn[ ], low[ ]。

dfn[ ] 表示在深度优先搜索时遍历的时间戳，每访问到一个新节点时间戳增加 1。

每个节点被第一次访问时，low 值与 dfn 值相同。当继续访问相邻节点，满足邻节点的 low 值小于节点 low 值，并且邻节点并不是父亲节点，则更新节点 low 值，并在回退中更新所有的 low 值。因此 low 表示了不通过父节点所能访问到的最早祖先节点。

深度优先搜索结束后，对图中每一个顶点 U，以及它的所有的孩子顶点 low 值进行比较，如果至少存在一个孩子顶点 V 满足 $low[v] \geq dfn[u]$，说明顶点 V 必须通过顶点 U 访问 U 的祖先节点，说明 U 时一个割点。

当满足 low[v]≥ dfn[u]，说明 V-U 是桥。

7.3 分析与证明

7.4 样例结果与输出

8 素性测试

8.1 本关任务

给定一个数 N，判断它是否是素数。

8.2 解法

利用 Miller-Rabin 算法来判断数字是否是素数。

费马小定理：当 p 为质数时，有 $a^{p-1} \equiv 1(mod\ p)$。

二次探测：如果 p 是一个素数，$0 < x < p$，则方程 $x^2 \equiv 1(mod\ p)$ 的解为 $x = 1$ 或 $x = p - 1$。

算法流程如下：

输入一个数 x。

(1) 当 x 是偶数或 2 时可以直接判断。

(2) 当 x 是奇数时，组一个比较小的质数 a, 计算 $2^s \cdot t = x - 1$。

(3) 先算出 a^t，然后进行 s 次平方进行二次探测。即如果在平方的过程中，当前数的平方模 x 与 1 同余并且该数不等于 1 或 $x-1$，那么 x 就是合数，直接退出循环。

(4) 根据费马定理判断 $a^{p-1} \equiv 1(mod\ p)$ 是否成立。如果成立，则从第 (2) 步重新挑选更大的质数 a 再次判断。

Algorithm 4: MergeSort

```
1  MergeSort(⟨a_n⟩) =
2  if |a_n| = 1 then
3  |  a_1
4  else
5  |  let
6  |  |  L ⟵ MergeSort(⟨a_1, a_2, ..., a_{(n+1)/2}⟩);
7  |  |  R ⟵ MergeSort(⟨a_{(n+1)/2}, ..., a_n⟩);
8  |  in
9  |  |  Merge(L, R)
10 |  end
11 end
```

复杂度为

$$W = O(n \log n) \quad S = O(\log^2 n)$$

8.3 分析与证明

8.4 样例结果与输出