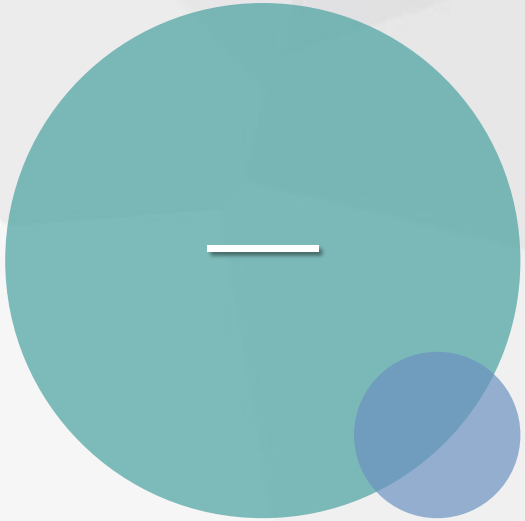
2018 暑假留校
算法练习讲解（1）

小组16级成员 娄泽豪



一

上周题目讲解

所用伪代码格式说明

keyword	关键字
Variable	关键变量
function	函数，函数调用
<-	(数据流)流向，插入，赋值
<--	追加
<-$	合理格式化后流向
a.*func*(..)	在a上调用func函数
[[INPUT]]	输入流
[[OUT]]	输出流

A. 字符排序

```
repeat Container <-- a, b, c <- [[INPUT]]
    sort(Container)
    [[OUT]] <-$ Container
```

B. 球体积

repeat R <- [[INPUT]]

[[OUT]] <-$ $\frac{4}{3}\pi R^3$

C. 回文串

```
N <- [[INPUT]]
repeat N times:
        STR_string <- [[INPUT]]
        REVSTR_string <- STR.foldr("", { _1 + _2 })
        [[OUT]] <-$ if (STR == REVSTR) "yes" else "no"
```

D. 最小公倍数

```
let lcm(a, b, x defaults b) =
      if (x % a == 0)  b
      else                lcm(a, b, x + b)

repeat a, b <- <INPUT>
      <OUT> <-$ lcm(min(a, b), max(a, b))
```

D. 最小公倍数——数论方法

```
let euclidean(a, b) =
      if (b > 0) euclidean(b, a % b)
      else       a

let lcm(a, b) = a * b / eulidean(a, b)

repeat a, b <- <INPUT>
      <OUT> <-$ lcm(a, b)
```

E. 计算器

```
let calculator(OP, a, b) = OP match {
      case '+' => a + b
      case '-' => a - b
      case '*' => a * b
      case '/' => a / b
}

N <- [[INPUT]]
repeat N times:
      OP_char, a_int, b_int <- [[INPUT]]
      [[OUT]] <-$ calculator(OP, a, b)
```

F. 数字字符统计

```
N <- [[INPUT]]
repeat N times:
        STR_string <- [[INPUT]]
        [[OUT]] <-$ STR.fold(0, { _1 + if (isNumber(_2)) 1 else 0 })
```

G. 数字序列插入

```
repeat N, M <- [[INPUT]]
    repeat N times
        Container <-- [[INPUT]]
    Container <-- M    // Container.insert(M)
    sort(Container)    // do nothing
    [[OUT]] <-$ Container
```

H. 合法标识符

```
let isfch(c) = (isAlpha(c) || c == '_')
let isvch(c) = (isvch(c) || isNumber(c))
let valid(STR) =
      if (isfch(STR[0])) false
      else STR.fold(0, { _1 + (if (isvch(_2)) 0 else 1 )}) == 0

N <- [[INPUT]]
repeat N times:
      STR_string <- [[INPUT]]
      [[OUT]] <-$ if (valid(OP)) "yes" else "no"
```

I. 三角形

```
let valid(a, b, c) = (a + b > c && a + c > b && b + c > a)

N <- [[INPUT]]
repeat N times:
    a, b, c <- [[INPUT]]
    [[OUT]] <-$ if (valid(a, b, c)) "YES" else "NO"
```

J. 亲和数

```
let amicable(n) =
      (1 to n / 2).fold(0, { _¹ + if (n % _² == 0) 1 else 0 })

N <- [[INPUT]]
repeat N times:
      a, b, c <- [[INPUT]]
      [[OUT]] <-$ if (a == amicable(b) && b == amicable(a))
                        "YES"
                  else
                        "NO"
```

K. 素数

```
let f(x) = x² + x + 41
let valid(x, y) =
      (x to y).fold(0, { _¹ + if (isPrime(f(_²)) 1 else 0 })

N <- [[INPUT]]
repeat { x, y <- [[INPUT]] } x != 0 || y != 0
      [[OUT]] <-$ if (valid(x, y)) "OK" else "Sorry"
```

K. 素数——朴素素数判断法

```
let isPrime(n, x defaults n / 2) =
      if       (x < 2)         true
      else if (n % x == 0)   false
      else                          isPrime(n, x - 1)
```

K. 素数——素数筛

```
const RANGE uint = 3000
var prime array<boolean, RANGE + 1>
let initPrime = {
      prime.foreach({ _ = true })
      prime[0] <- prime[1] <- false
      (2 to RANGE).foreach({
            (_ to RANGE step _).foreach({ prime[_] <- false })
      })
}

let isPrime(n) = prime[n]
```

L. 首字母大写

```
let getState(ch) = ch match {
      case ' '  => STATE_SPACE
      case '\0' => STATE_END
      case _    => STATE_WORDCTX
}
let fsm(str, state, pos) = state match {
      case STATE_END => ;;
      case _ => {
            [[OUT]] <-$ if (_ == STATE_WORDCTX) str[pos]
                        else toUpper(str[pos])
            fsm(str, getState(str[pos]), pos + 1)
      }
}
```

L. 首字母大写

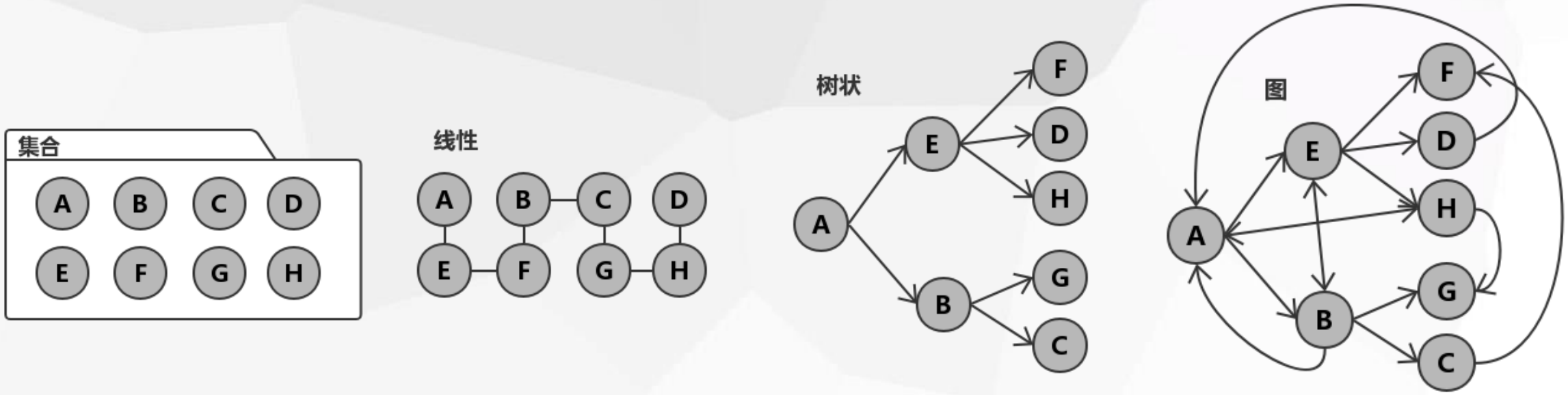
```
repeat STR_string <- [[INPUT]]
    fsm(STR, STATE_SPACE, 0)
```

二

基本搜索算法

搜索与图

搜索是一种基本的图算法



四种基本的数据结构

结点？

通俗地说，在搜索算法中提到的结点，即表示搜索范围内可能的一种情况。例如，对于在“迷宫中寻找路径”问题里，“在第t时间处于(x, y)坐标”即是一种可能的情况，即一个“结点”。

结点具有零个或多个“通向”的结点，即可能从这种情况通过某种操作到达的另一种情况。我们简单地称之为“通向”。

一般策略

1. 根据需要选择一个初始“结点”，我们将之称为当前结点
2. 判断当前结点是否是需要的（是否是最终答案）。
3. 如果当前结点虽然不是答案，但是当前结点还可以通向其他结点（即有可能通向答案或可以通向答案的结点），那么从当前结点通向的结点中选取一个作为下一个当前结点，并返回步骤2；如果没有，根据选取结点的策略不同，既可能返回上一个“当前结点”，也可能已经存在下一个“当前结点”，那么进行下一个“当前结点”，也可能直接结束。
4. 反复进行以上步骤后，应当已经找到答案，或者检查完了所有可能的情况，说明没有可能的答案。

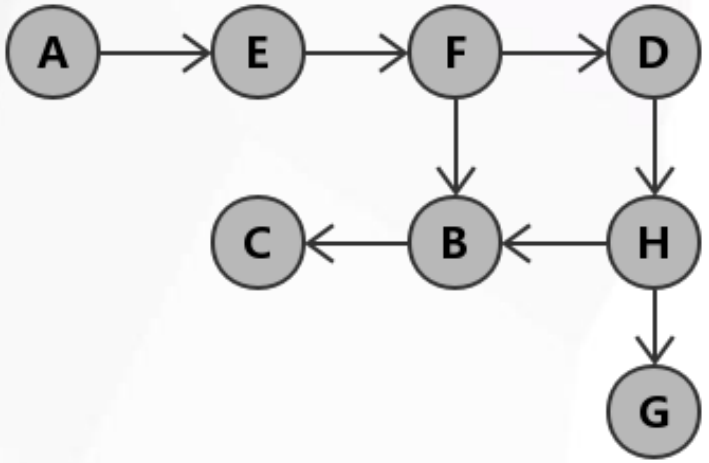
选取策略

1．深度优先

2．广度优先

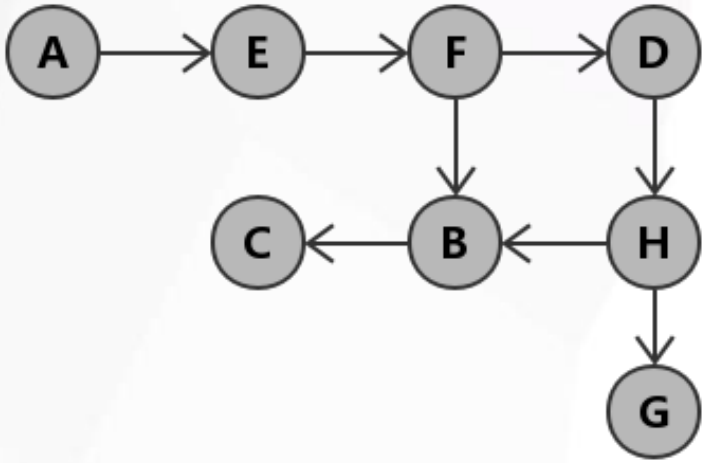
先试试看——深度优先搜索

深度优先在选取“下一个当前结点时”，按照一定的顺序逐个筛选选取，当成功地进行一次选取后，下一个当前结点将是这个被选取到的结点。若最终这个被选取的结点被证明即不是答案，也不能通向答案，那么再进行第二次选取。



普遍撒网——广度优先搜索

深度优先在选取“下一个当前结点时”，将所有的结点都按照顺序逐个放在队列中。下一个当前结点是从这个队列中拿出的队头结点。



优化策略

1．记忆化：访问过的结点可能不必再访问

2．排除明显不可能的结点：如果通向的结点显然不具备答案或已经证明其不存在答案，那么不再选取它。

3．一般来说，求“最”值的会倾向于使用广度优先，求“一个”值的会倾向于使用深度优先。

END

Thanks!