节选第六章

6.13. For the repair model presented in Section 6.7:

(b) Use your program to estimate the mean crash time in the case where $n = 4, s = 3, F(x) = 1 - e^{-x}$,and $G(x) = 1 - e^{-2x}$

解: 这里介绍两种方法: 一种基于过程考虑, 代码比较长, 但算法的效率比较高 (就是书上的方法); 另一种方法基于状态考虑, 代码只有二十来行, 但算法的效率比较低.

法一: 基于过程, 流程图如下 (书上 P108 可参考):

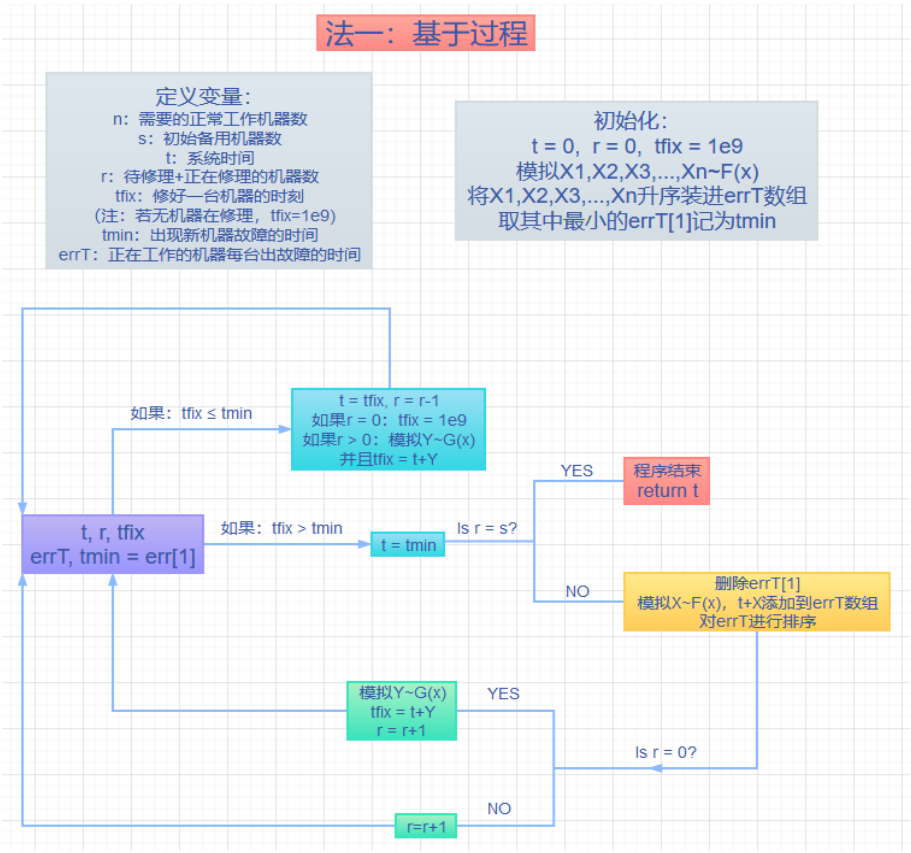

图 1: 13 题 (法一) 流程图

法一全过程解释:

系统正常工作所需要的机器数量为n, 初始备用机器数量为s. 系统初始化: 先模拟生成第一批工作的机器的工作时长, 并进行升序排序, 最先坏掉的机器, 其坏掉的时间为$errT[1]$, 记为$tmin$. 进入循环: 机器坏掉就要拿去修理, 由于同时在修理的机器最多只能有一台, 因此如果前一台机器没有修理好又有新机器坏掉了 ($tfix > tmin$), 那么新坏掉的机器只能加入等待队列 ($r = r+1$); 如果机器修好时, 还没有新机器坏掉 ($tfix \leq tmin$), 则 (待修理 + 修理中的机器数)-1, 即 ($r = r - 1$). 再看此时是否还有机器在等待队列中未修理, 如果有就生成修理机器所需的时间Y, 再计算出此机器修理好的时间 (即 $t + Y$). 机器坏掉需要立刻从备用池里取出一台替补上, 如果备用池里没有机器 (意味着坏掉的机器全在以下两种状态: 待修理或正在修, 即 $r = s$), 则系统崩溃退出. 如果备用池里还有机器, 这时需要模拟生成新加入机器的工作时间X, 计算其下次坏掉的时间 (即 $t + X$), 与所有正在工作机器下次坏掉的时间进行升序排序, 取最小的记为$tmin$. 上述工作完成后, 进入下一轮循环, 直到程序退出为止. 因此法一 (基于过程) 的代码如下:

```
f <- function(n,s) {
  m <- 66666
  result <- vector(length = m)
  g <- function(n, s) {
    t <- 0
    r <- 0
    errT <- sort(rexp(n, rate = 1))
    tfix <- 1e9
    for(k in 1:10000) {
      tmin <- errT[1]
      if(tfix <= tmin) {
        t <- tfix
        r <- r-1
        if(r==0) {
          tfix <- 1e9
        }
        if(r > 0) {
          Y <- rexp(1, rate = 2)
          tfix <- t+Y
```

```
        }
      } else {
        t <- tmin
        if(r==s) {
          return(t)
        } else {
          X <- rexp(1, rate = 1)
          errT <- sort(append(errT[2:length(errT)], t+X))
          if(r==0) {
            Y <- rexp(1, rate = 2)
            tfix <- t+Y
          }
          r <- r+1
        }
      }
    }
  }
  for(i in 1:m) {
    result[i] <- g(n,s)
  }
  hist(result, freq = F, breaks = 30,
       xlab = "持续时间", ylab = "频率",
       main = "持续时间分布图")
  print(mean(result))
  print(sd(result))
}
f(4,3)
```

```
> f(4,3)
[1] 1.529677
[1] 1.063099
```

图 2: 13 题 (法一) 运行截图

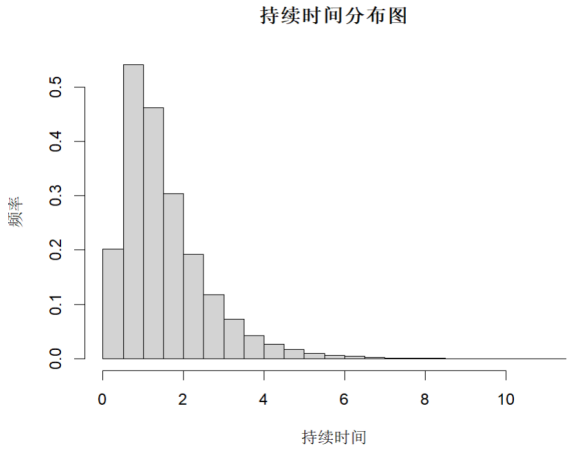

图 3: 13 题 (法一) 持续时间分布图

由输出结果: 经过 66666 次的模拟, 发现整个系统持续时间 (首次崩溃的发生时间) 的均值为 1.529677, 标准差为 1.063099

法二: 基于状态, 流程图如下:

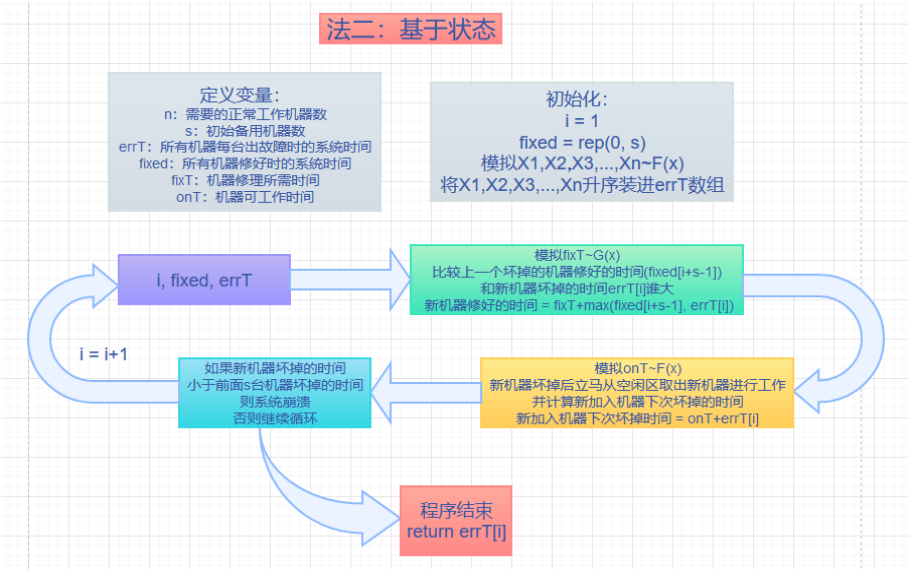

图 4: 13 题 (法二) 流程图

法二全过程解释:
这个思路其实很简单, 举个例子就能明白了, 就拿本题来说: 需要 4 台机器同时工作, 初始时备用池里有 3 台机器, 那么意味着: 第四台机器坏掉的时间不能小于第一台机器修好的时间, 第五台机器坏掉的时间不能小于第二台机器修好的时间 (第 i 台机器坏掉的时间不能小于第 $i-3$ 台机器修好的时间), 否则意味着没有一台机器处于备用状态. 因此: 整个算法中只需要两个核心的状态变量:$errT$ 和 $fixed$, 分别记录机器坏掉的时间和机器修好的时间就行了. 无需考虑更多的细节, 就无脑算每台机器的 $errT$ 和 $fixed$ 就行了, 循坏到直至整个工作系统崩溃.(但不要忘记一点: 前三台机器坏掉是无论如何都可以有机器替补的, 因此 $fixed$ 变量的前三个值都计为 0)
因此法二 (基于状态) 的代码如下:

```
f <- function(n, s) {
  m <- 66666
  result <- vector(length = m)
  g <- function(n, s) {
    fixed <- rep(0, s)
    errT <- sort(rexp(n, rate = 1))
    for(i in 1:6666) {
      fixT <- rexp(1, rate = 2)
      fixed <- append(fixed,
                fixT+max(errT[i], fixed[i+s-1]))
      onT <- rexp(1, rate = 1)
      errT <- sort(append(errT, onT+errT[i]))
      if(errT[i] < fixed[i]) {
        return(errT[i])
      }
    }
  }
  for(i in 1:m) {
    result[i] <- g(n,s)
  }
  hist(result, freq = F, breaks = 30,
       xlab = "持续时间", ylab = "频率",
```

6

```
        main = "持续时间分布图")
    print(mean(result))
    print(sd(result))
}
f(4,3)
```

```
> f(4,3)
[1] 1.525008
[1] 1.05804
```

图 5: 13 题 (法二) 运行截图

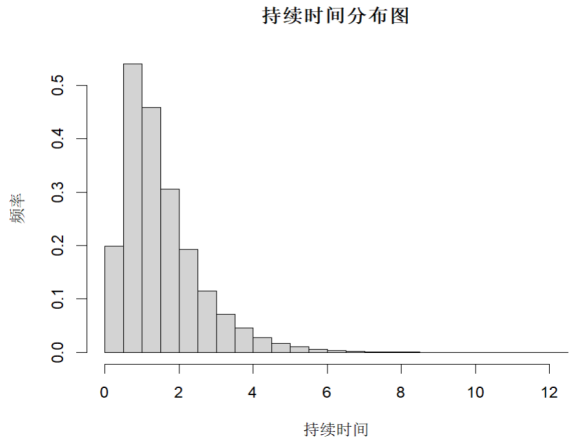

图 6: 13 题 (法二) 持续时间分布图

由输出结果: 经过 66666 次的模拟, 发现整个系统持续时间 (首次崩溃的发生时间) 的均值为 1.525008, 标准差为 1.058040. 两种方法的模拟结果非常相近.

6.14. In the model of Section 6.7, suppose that the repair facility consists of two servers, each of whom takes a random amount of time having distribution G to service a failed machine. Draw a flow diagram for this system.

翻译: 在第 6.7 节的模型中, 假设维修设施由两台服务器组成, 每台服务器都

需要随机分配 G 的时间来维修故障机器, 绘制此系统的流程图

解: 同样还是有 13 题的两种方法: 分别基于过程和状态, 前者代码量大但算法效率较高; 后者代码量小但算法效率较低。除此之外, 基于状态考虑的算法还有个其他的优势: 这个优势当只有一台维修设施时基本体现不出来, 就是不需要考虑每种情况的细节, 这一题很容易对情况的细节考虑不全, 因此如果使用基于状态的算法, 可以为我们节约很多思考时间.

法一: 基于过程, 流程图如下:

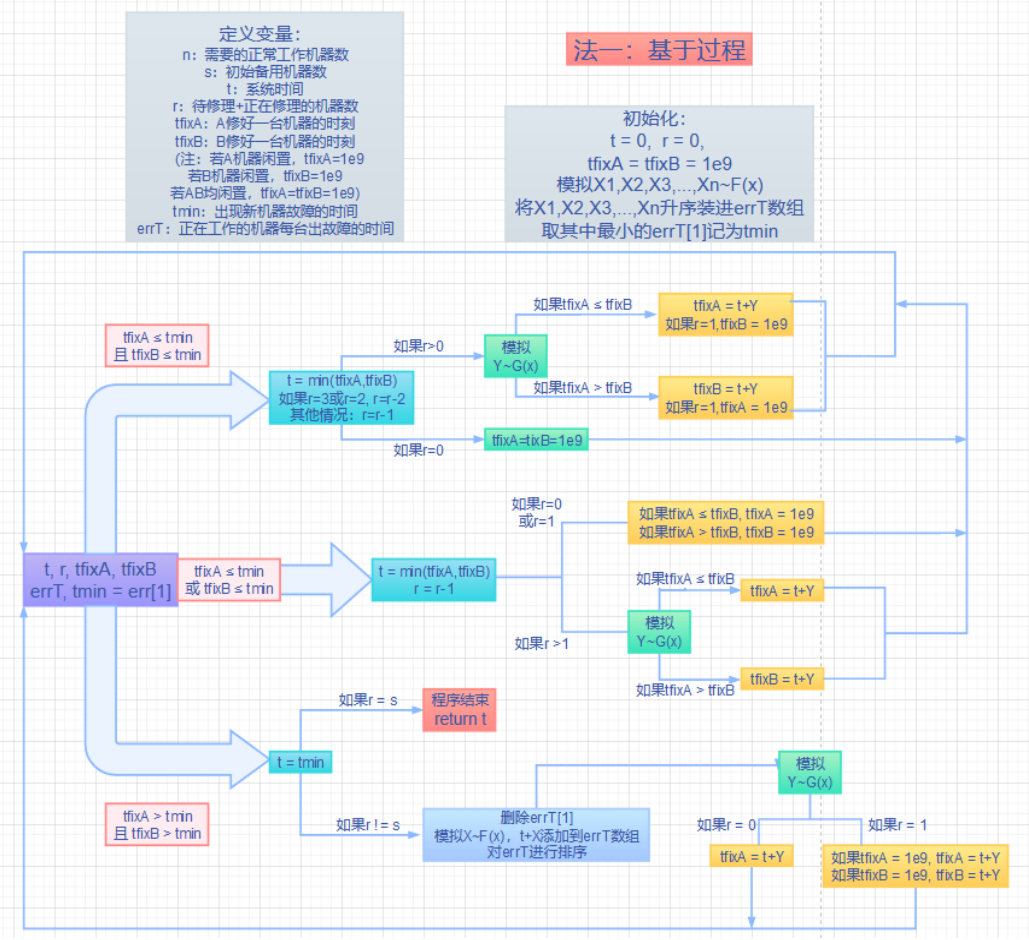

图 7: 14 题 (法一) 流程图

这里是改成了两台设施 (分别记为 A,B) 进行维修, 情况多了非常多. 一共有

以下 7 个细节需要去注意, 所以说用这种基于过程的算法是非常容易漏掉情况的

(1):A 和 B 都闲着没事在发呆 (意味着当前系统时间没有坏掉的机器, 坏掉的都修好了)

(2):A 在修, 且 A 修完后发现 B 在那闲着 (意味着在 A 修的期间内没有新的坏掉的机器), 那么下一台坏掉的机器默认还是给 A 修

(3):A 和 B 有一方在修, 且在其修的期间内, 又出现了一台坏的机器, 那么这台机器就得 A 和 B 中的另外一方去修

(4):A 在修 B 也在修, 且在他们修的期间内, 又出现了一台坏的机器, 那么这台机器只能暂时被放在等待队列

(5):A 在修 B 也在修, 如果有机器在等待队列, 那么 A 和 B 先修好手中的机器的需要把等待队列中的机器拿出来修

(6):A 在修 B 也在修, 如果没有机器在等待队列, 那么谁先修完谁先休息 (在此之前没有新的坏掉的机器)

(7): 如果备用池里没有机器, 意味着系统在此刻陷入崩溃, 循环结束

Tips: 这个方法太过于死板, 其实是可以通过推演获取到一些高级结论的, 这就是后面要说的基于状态的解决方案, 此处流程图真全部描述一遍起码得 1500+ 字篇幅, 所以就不描述咯.....

法一 (基于过程) 的代码如下:

```
f <- function(n,s) {
  m <- 16666
   result <- vector(length = m)
   g <- function(n, s) {
     t <- 0
     r <- 0
     errT <- sort(rexp(n, rate = 1))
     tfixA <- 1e9
     tfixB <- 1e9
     for(k in 1:10000) {
       tmin <- errT[1]
       if((tfixA <= tmin) && (tfixB <= tmin)) {
         t <- min(tfixA , tfixB)
         if((r==3)||(r==2)) {
```

```
    r <- r-2
  } else {
    r <- r-1
  }
  if(r==0) {
    tfixA <- 1e9
    tfixB <- 1e9
  }
  if(r>0) {
    Y <- rexp(1, rate = 2)
    if(tfixA <= tfixB) {
      tfixA <- t+Y
      if(r==1) {
        tfixB <- 1e9
      }
    } else {
      tfixB <- t+Y
      if(r==1) {
        tfixA <- 1e9
      }
    }
  }
} else if((tfixA <= tmin) || (tfixB <= tmin)){
  t <- min(tfixA, tfixB)
  r <- r-1
  if((r==1)||(r==0)) {
    if(tfixA <= tfixB) {
      tfixA <- 1e9
    } else {
      tfixB <- 1e9
    }
  }
  if(r > 1) {
```

```
          Y <- rexp(1, rate = 2)
          if(tfixA <= tfixB) {
            tfixA <- t+Y
          } else {
            tfixB <- t+Y
          }
        }
      } else {
        t <- tmin
        if(r==s) {
          return(t)
        } else {
          X <- rexp(1, rate = 1)
          errT <- sort(append(errT[2:length(errT)], t+X))
          Y <- rexp(1, rate = 2)
          if(r==0) {
            tfixA <- t+Y
          }
          if(r==1) {
            if(tfixA==1e9) {
              tfixA <- t+Y
            }
            if(tfixB==1e9) {
              tfixB <- t+Y
            }
          }
          r <- r+1
        }
      }
    }
  }
  for(i in 1:m) {
    result[i] <- g(n,s)
```

```
    }
    hist(result, freq = F, breaks = 30,
         xlab = "持续时间", ylab = "频率",
         main = "持续时间分布图")
    print(mean(result))
    print(sd(result))
}
f(4,3)
```

```
> f(4,3)
[1] 2.117459
[1] 1.69511
```

图 8: 14 题 (法一) 运行截图

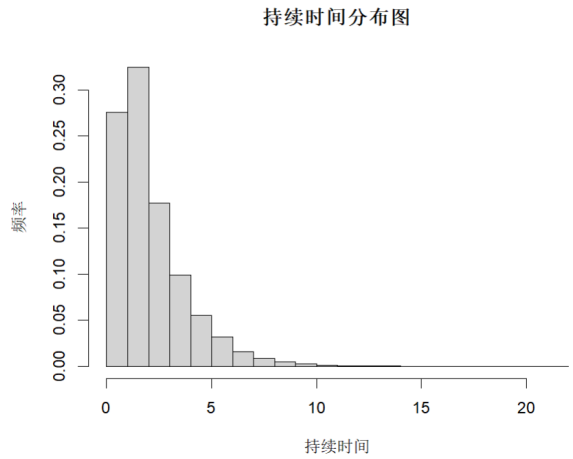

图 9: 14 题 (法一) 持续时间分布图

由输出结果: 经过 16666 次的模拟, 发现当系统有两台维修设施时, 整个系统持续时间 (首次崩溃的发生时间) 的均值提升到了 2.117459, 标准差为 1.695110

法二: 基于状态, 流程图如下:

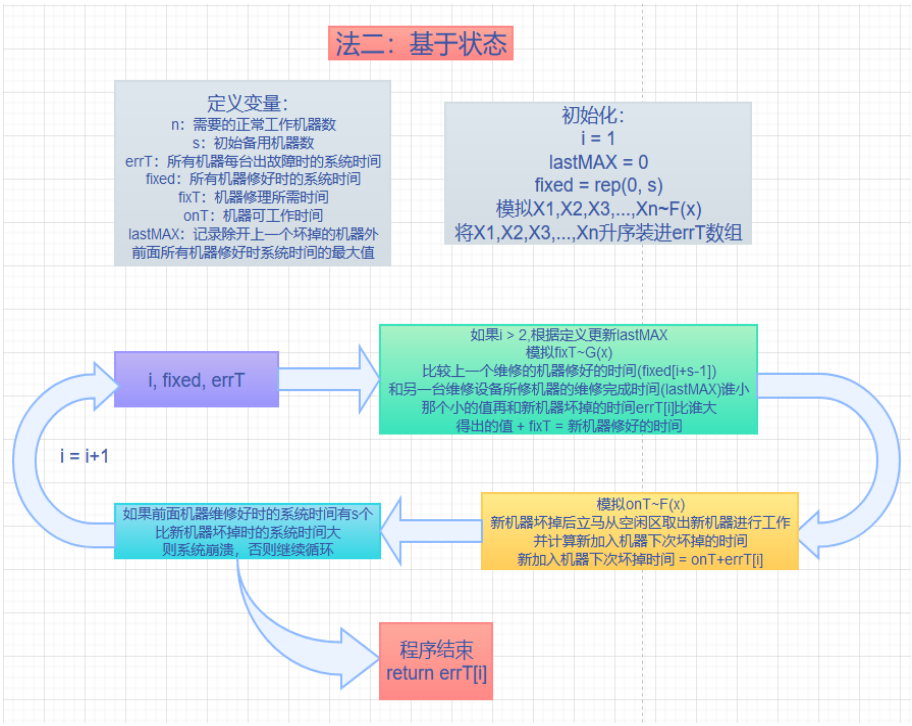

图 10: 14 题 (法二) 流程图

法二全过程解释:

这个 14 题相较于上面的 13 题, 其实就只改变了一层逻辑:" 重新计算机器开始维修的时间". 所以完全可以继承 13 题法二的解决方案, 在此之上改变这一层逻辑就行了. 具体该如何计算呢? 首先机器要维修的前提是它必须是坏掉, 因此需要判断机器啥时候坏, 坏掉时的系统时间记为 errT[i]. 其次, 一台机器想要被修, 必须确保 A 和 B 两方维修设施有一方率先完成了手里正在维修的机器,A 和 B 手里机器的情况有: 其中一方手里必定是上一台坏掉的机器; 另一方手里是除开上一台坏掉的机器外, 其他所有已坏掉的机器都有可能.

举个例子: 假如第 11 台坏掉的机器准备维修, 那么 A 和 B 其中一方的手里必定是第 10 台坏掉的机器 (这个其实不难理解: 假如两方手里正在修的都不是第 10 台机器, 比方说 A 和 B 分别维修的是第 8 台和第 9 台机器吧, 那么先修好的那一方必定要拿第 10 个坏掉的机器来修, 而不是拿第 11 台坏掉的机器过来修, 即先坏的必定比后坏的更早修, 因此两方手里正在维修的必定有第 10 台坏掉的机器); 而另一方是第 1 到 9 台坏掉的机器都有可能 (这里很容易

会误会为: 另一方只能是第 9 台坏掉的机器, 其实不是的, 你遗漏了以下这种极端情况: 第 1 台机器坏掉了 A 去维修, 需要七七四十九个小时才能修好, 而第 2 到第 10 台坏掉的机器可能分别都只需要 6 小时, 意味着 B 修了第 2 到第 10 台坏掉的机器, 而在 B 正在修第 10 台坏掉的机器时,A 终于把第 1 台坏掉的机器修好了. 你说第 11 台坏掉的机器应该谁去修? 明显是 A 去修啊), 因此第 i 台坏掉的机器还跟以下这两个变量扯上关系: 第 i-1 台机器修理完成的时间和第 1 到第 (i-2) 台坏掉的机器中最晚修理好的时间. 两者取个最小值, 再与 errT[i] 取个最大值, 就是第 i 台坏掉的机器开始维修的时间, 最后再加上这台机器的维修所需时间 Y, 即为最终修理好的时间.

因此不难发现, 还是只需要两个状态变量:$errT$ 和 $fixed$, 分别记录每台机器坏掉的时间和修好的时间就行了 (注意, $fixed$ 变量初始化时前 s 个值都为 0). 无需考虑更多的细节, 就无脑算每台机器的 $errT$ 和 $fixed$ 就行了, 无非就是多了个临时变量用来记录第 1 到 (i-2) 个坏掉的机器谁最晚修理完成而已

其实还有一个需要注意的点: 何时退出循环? 这跟 13 题只有一台维修设备时还不太一样, 不过举例例子肯定能懂了 (还是 13 题 (b) 的例子): 第 5 台机器坏掉了,$fixed$ 数组中前 7 个数据大于第 5 台机器坏掉的时间的不能大于等于 3 个 (第 i 台机器坏掉了,$fixed$ 数组中前 $i+s-1$ 个数据大于第 i 台机器坏掉的时间的不能大于等于 s 个)

```
f <- function(n, s) {
  m <- 16666
  result <- vector(length = m)
  g <- function(n, s) {
    fixed <- rep(0, s)
    errT <- sort(rexp(n, rate = 1))
    lastMAX <- 0
    for(i in 1:666) {
      if(i>2){
        lastMAX <- max(fixed[(s+1):(length(fixed)-1)])
      }
      fixT <- rexp(1, rate = 2)
      fixed <- append(fixed, fixT+max(errT[i],
                      min(fixed[i+s-1], lastMAX)))
      onT <- rexp(1, rate = 1)
```

14

```
      errT <- sort(append(errT, onT+errT[i]))
      if(sum(fixed[1:i+s-1]>errT[i]) >= s) {
        return(errT[i])
      }
    }
  }
  for(i in 1:m) {
    result[i] <- g(n,s)
  }
  hist(result, freq = F, breaks = 20,
       xlab = "持续时间", ylab = "频率",
       main = "持续时间分布图")
  print(mean(result))
  print(sd(result))
}
f(4,3)
```

```
> f(4,3)
[1] 2.111064
[1] 1.70365
```

图 11: 14 题 (法二) 运行截图

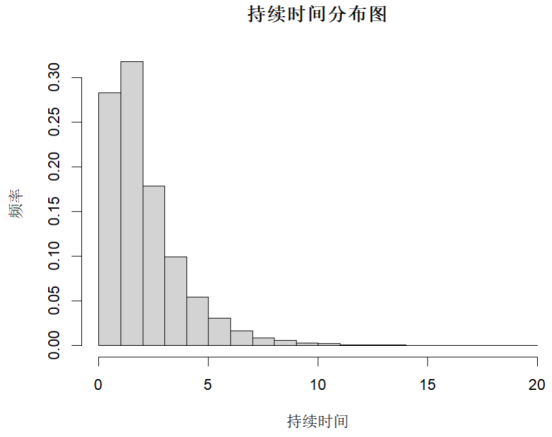

图 12: 14 题 (法二) 持续时间分布图

由输出结果: 经过 16666 次的模拟, 发现当系统有两台维修设施时, 整个系统持续时间 (首次崩溃的发生时间) 的均值提升到了 2.111064, 标准差为 1.703650. 两种方法的模拟结果近似.

附图:14 题 (法一) 单次运行过程变量输出值:
从上到下, 首先三个一行的, 从左到右依次是:A 维修完成手上坏掉的机器时的系统时间,B 维修完成手上坏掉的机器时的系统时间, 系统中正在维修和等待维修的机器数量之和; 有 14 个数据的那串是 errT 变量

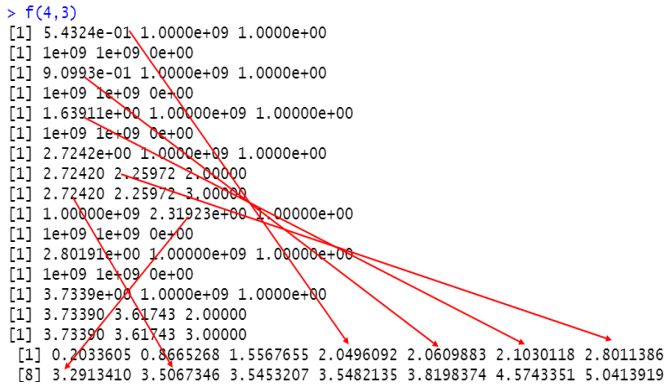

图 13: 14 题 (法一) 单次运行过程变量输出结果

附图:14 题 (法二) 单次运行过程变量输出值:
从上到下: 首先一行一个数据的, 是每个坏掉的机器需要修理的时间; 有 13 个数据的那串是 errT 变量; 有 12 个数据的那串是 fixed 变量

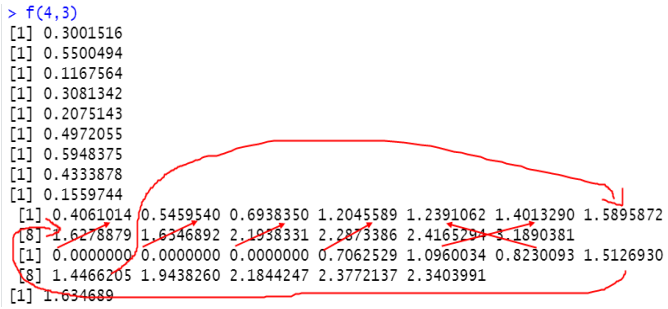

图 14: 14 题 (法二) 单次运行过程变量输出结果