```go
package main

import (
    "fmt"
    "sort"
)

type Job struct {
    JobNumber               int
    ArrivalTime             int
    EstimatedTime           int
    StartTime               int
    EndTime                 int
    TurnaroundTime          int
    WeightedTurnaroundTime float64
    ResponseRatio           float64
}

// 计算时间增加
func TimeAdd(initialTime int, addedTime int) int {
    hour := initialTime / 100
    minute := initialTime % 100
    new_minute := (addedTime + minute) % 60
    new_hour := (hour + (addedTime+minute)/60) % 24
    return 100*new_hour + new_minute
}

// 计算时间相减
func TimeSub(subbedTime int, initialTime int) int {
    ori_hour := initialTime / 100
    ori_minute := initialTime % 100
    subbed_hour := subbedTime / 100
    subbed_minute := subbedTime % 100
    return subbed_hour*60 + subbed_minute - ori_hour*60 -
ori_minute
}
```

```
// 计算响应比
func CalculateResponseRatio(startTime int, arrivalTime int,
estimatedTime int) float64 {
    return 1 + (float64(TimeSub(startTime,
arrivalTime)))/float64(estimatedTime)
}

func PrintTitle() {
    fmt.Println("作业\t\t进入时间\t估计运行时间\t开始时间\t结束时间\t周
转时间(分钟)\t带权周转时间\t")
}

func PrintJob(jobs []Job) {
    for _, m := range jobs {
        fmt.Printf("JOB%d\t", m.JobNumber)
        PrintTime(m.ArrivalTime)
        fmt.Printf("\t    %d \t ", m.EstimatedTime)
        PrintTime(m.StartTime)
        PrintTime(m.EndTime)
        fmt.Printf("\t\t%d\t", m.TurnaroundTime)
        fmt.Printf("\t%.2f\t", m.WeightedTurnaroundTime)
        fmt.Printf("\n")
    }
}

func PrintTime(originalTime int) {
    fmt.Printf("\t%02d:%02d\t", originalTime/100,
originalTime%100)
}

func main() {
    //var jobs []Job
    //var num int
    //fmt.Printf("输入作业个数:")
    //fmt.Scan(&num)
    //for i := 0; i < num; i++ {
```

```
    // var job Job
    // fmt.Scan(&job.JobNumber, &job.ArrivalTime,
&job.EstimatedTime)
    // jobs = append(jobs, job)
    //}
    jobs := []Job{
        {1, 800, 50, 0, 0, 0, 0, 0},
        {2, 815, 30, 0, 0, 0, 0, 0},
        {3, 830, 25, 0, 0, 0, 0, 0},
        {4, 835, 20, 0, 0, 0, 0, 0},
        {5, 845, 15, 0, 0, 0, 0, 0},
        {6, 900, 10, 0, 0, 0, 0, 0},
        {7, 920, 5, 0, 0, 0, 0, 0},
    }

    // 使用先来先服务（FIFO）算法进行作业调度
    fmt.Println("-----------------------------------------------
---先进先出(FIFO)调度算法-----------------------------------------
--------")
    PrintTitle()
    scheduleFIFO(jobs)

    // 使用短作业优先（SJF）算法进行作业调度
    fmt.Println("\n---------------------------------------------
----短作业优先(SJF)调度算法---------------------------------------
---------")
    PrintTitle()
    scheduleSJF(jobs)

    // 使用最高响应比优先（HRRN）算法进行作业调度
    fmt.Println("\n---------------------------------------------
--最高响应比优先(HRRN)调度算法-----------------------------------
---------")
    PrintTitle()
    scheduleHRRN(jobs)

}
```

```
func scheduleFIFO(jobs []Job) {
    copiedJobs := make([]Job, len(jobs))
    copy(copiedJobs, jobs)
    for i, _ := range copiedJobs {
       if i == 0 {
          copiedJobs[i].StartTime = copiedJobs[i].ArrivalTime
       } else {
          copiedJobs[i].StartTime = copiedJobs[i-1].EndTime
       }
       copiedJobs[i].EndTime = TimeAdd(copiedJobs[i].StartTime,
copiedJobs[i].EstimatedTime)
       copiedJobs[i].TurnaroundTime =
TimeSub(copiedJobs[i].EndTime, copiedJobs[i].ArrivalTime)
       copiedJobs[i].WeightedTurnaroundTime =
float64(copiedJobs[i].TurnaroundTime) /
float64(copiedJobs[i].EstimatedTime)
    }
    PrintJob(copiedJobs)
}

func scheduleSJF(jobs []Job) {
    // 第一个作业先执行
    jobs[0].StartTime = jobs[0].ArrivalTime
    jobs[0].EndTime = TimeAdd(jobs[0].ArrivalTime,
jobs[0].EstimatedTime)
    jobs[0].TurnaroundTime = jobs[0].EstimatedTime
    jobs[0].WeightedTurnaroundTime = 1
    copiedJobs := make([]Job, len(jobs))
    copy(copiedJobs, jobs)
    currentTime := jobs[0].EndTime
    scheduledJobs := []Job{}
    for len(copiedJobs) > 0 {
       availableJobs := []Job{}
       // 已经到达的
       for _, job := range copiedJobs {
          if job.ArrivalTime <= currentTime {
```

```
            availableJobs = append(availableJobs, job)
        }
    }
    if len(availableJobs) == 0 {
        // 如果没有可用作业，则将当前时间递增到下一个作业的到达时间
        currentTime = copiedJobs[0].ArrivalTime
        continue
    }
    sort.Slice(availableJobs, func(i, j int) bool {
        return availableJobs[i].EstimatedTime <
availableJobs[j].EstimatedTime
    })
    // 调度估计运行时间最短的作业
    shortestJob := availableJobs[0]
    scheduledJobs = append(scheduledJobs, shortestJob)
    // 更新当前时间并从列表中移除已调度的作业
    currentTime = TimeAdd(currentTime,
shortestJob.EstimatedTime)
    for i, job := range copiedJobs {
        if job.JobNumber == shortestJob.JobNumber {
            copiedJobs = append(copiedJobs[:i],
copiedJobs[i+1:]...)
            break
        }
    }
}

for i, _ := range scheduledJobs {
    if scheduledJobs[i].JobNumber != 1 {
        if i == 0 {
            scheduledJobs[i].StartTime = jobs[0].EndTime
        } else {
            if scheduledJobs[i-1].JobNumber == 1 {
                scheduledJobs[i].StartTime =
scheduledJobs[i-2].EndTime
            } else {
```

```
                scheduledJobs[i].StartTime =
scheduledJobs[i-1].EndTime
            }
        }
        scheduledJobs[i].EndTime =
TimeAdd(scheduledJobs[i].StartTime,
scheduledJobs[i].EstimatedTime)
        scheduledJobs[i].TurnaroundTime =
TimeSub(scheduledJobs[i].EndTime, scheduledJobs[i].ArrivalTime)
        scheduledJobs[i].WeightedTurnaroundTime =
float64(scheduledJobs[i].TurnaroundTime) /
float64(scheduledJobs[i].EstimatedTime)
      }
   }
   sort.Slice(scheduledJobs, func(i, j int) bool {
      return scheduledJobs[i].JobNumber <
scheduledJobs[j].JobNumber
   })
   PrintJob(scheduledJobs)
   return
}

func scheduleHRRN(jobs []Job) {
   jobs[0].StartTime = jobs[0].ArrivalTime
   jobs[0].EndTime = TimeAdd(jobs[0].ArrivalTime,
jobs[0].EstimatedTime)
   jobs[0].TurnaroundTime = jobs[0].EstimatedTime
   jobs[0].WeightedTurnaroundTime = 1
   jobs[0].ResponseRatio = 1
   copiedJobs := make([]Job, len(jobs))
   copy(copiedJobs, jobs)
   currentTime := jobs[0].ArrivalTime
   scheduledJobs := []Job{}

   for len(copiedJobs) > 0 {
      availableJobs := []Job{}
      // 已经到达的
```

```
        for _, job := range copiedJobs {
            if job.ArrivalTime <= currentTime {
                availableJobs = append(availableJobs, job)
            }
        }

        if len(availableJobs) == 0 {
            // 如果没有可用作业，则将当前时间递增到下一个作业的到达时间
            currentTime = copiedJobs[0].ArrivalTime
            continue
        } else {
            for i, _ := range availableJobs {
                if availableJobs[i].JobNumber == 1 {
                    availableJobs[i].ResponseRatio = 0
                } else {
                    lastestIndex := len(scheduledJobs) - 1
                    availableJobs[i].ResponseRatio =
CalculateResponseRatio(scheduledJobs[lastestIndex].EndTime,
availableJobs[i].ArrivalTime, availableJobs[i].EstimatedTime)
                }
            }
        }
        sort.Slice(availableJobs, func(i, j int) bool {
            return availableJobs[i].ResponseRatio >
availableJobs[j].ResponseRatio
        })

        // 调度响应比高的
        respJob := availableJobs[0]
        if respJob.JobNumber != 1 {
            lastIndex := len(scheduledJobs) - 1
            respJob.StartTime = scheduledJobs[lastIndex].EndTime
            respJob.EndTime = TimeAdd(respJob.StartTime,
respJob.EstimatedTime)
            respJob.TurnaroundTime = TimeSub(respJob.EndTime,
respJob.ArrivalTime)
```

```
            respJob.WeightedTurnaroundTime =
float64(respJob.TurnaroundTime) / float64(respJob.EstimatedTime)
        }
        scheduledJobs = append(scheduledJobs, respJob)
        // 更新当前时间并从列表中移除已调度的作业
        currentTime = TimeAdd(currentTime, respJob.EstimatedTime)

        for i, job := range copiedJobs {
            if job.JobNumber == respJob.JobNumber {
                copiedJobs = append(copiedJobs[:i],
copiedJobs[i+1:]...)
                break
            }
        }

    }
    sort.Slice(scheduledJobs, func(i, j int) bool {
        return scheduledJobs[i].JobNumber <
scheduledJobs[j].JobNumber
    })
    PrintJob(scheduledJobs)
    return
}
```