

```

1 // this code is achieved with golang 1.17
2 // package reflect is used to implement Calling Subprograms Indirectly
3
4 package main
5
6 import (
7     "flag"
8     "fmt"
9     "log"
10    "math"
11    "strconv"
12    "sync"
13    "time"
14 )
15 import "reflect"
16
17 var primeLTE101 = []int{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
18 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101}
19 // PrimeArray is used to generate prime number until sqrt(max)
20 var PrimeArray = []int{}
21 var ResultChan chan int
22 var ThreadLock sync.WaitGroup
23 var OutputLock sync.WaitGroup
24 var OutputCounter int
25 var JustCount bool
26
27 type IndirectMethodsLT100 struct {}
28
29 func (t *IndirectMethodsLT100) CheckPrime (primeCheckMin,primeCheckMax int)
30 int {
31     outputCounter :=0
32     for _,primeNum:= range primeLTE101{
33         if primeNum > primeCheckMax{
34             break
35         }
36         if primeNum >= primeCheckMin{
37             if !JustCount{
38                 fmt.Printf("%d is a prime number \n",primeNum)
39             }
40             outputCounter++
41         }
42     }
43     return outputCounter
44 }
45
46 type IndirectMethodsGTE100 struct {}
47
48 func (t *IndirectMethodsGTE100) CheckPrime (primeCheckMin,primeCheckMax
49 int)int{
50     outputCounter :=0
51     // just output the prime number in the list first
52     if primeCheckMin<primeLTE101[len(primeLTE101)-1]{
53         for _,primeNum:= range primeLTE101{

```

```

53         if primeNum >= primeCheckMin{
54             if !JustCount{
55                 fmt.Printf("%d is a prime number \n",primeNum)
56             }
57             outputCounter++
58         }
59     }
60     primeCheckMin= primeLTE101[len(primeLTE101)-1]+2
61 }
62 if primeCheckMin%2==0{
63     primeCheckMin++
64 }
65 // start to use the prime list to check for larger prime number
66 for currentNum :=
primeCheckMin;currentNum<=primeCheckMax;currentNum+=2{
67     checkMax := math.Sqrt(float64(currentNum))
68     //check with each prime number
69     for _,primeNum := range primeLTE101{
70         //fmt.Printf("%f",checkMax)
71         if float64(primeNum)> checkMax{
72             //this is a prime number
73             if !JustCount{
74                 fmt.Printf("%d is a prime number \n",currentNum)
75             }
76             outputCounter++
77             break
78         }
79         if currentNum%primeNum==0{
80             //not prime
81             break
82         }
83     }
84 }
85 return outputCounter
86 }
87
88 type IndirectMethodsGTE10000 struct {}
89
90 func (t *IndirectMethodsGTE10000) CheckPrime (primeCheckMin,primeCheckMax
int)int{
91     // populate the prime list that the largest prime should be gte
sqrt(primeCheckMax) first
92     start := time.Now()
93     OutputCounter =0
94     PrimeArray = primeLTE101
95     primeNumberAtLeast := int(math.Ceil(math.Sqrt(float64(primeCheckMax))))
96     for currentNum := 103;;currentNum+=2{
97         checkMax := int(math.Ceil(math.Sqrt(float64(currentNum))))
98         //check with each prime number
99         if PrimeArray[len(PrimeArray)-1] >=primeNumberAtLeast{
100             // populate completed
101             break
102         }
103         for _,primeNum := range PrimeArray {
104             if primeNum> checkMax{
105                 //this is a prime number
106                 PrimeArray = append(PrimeArray,currentNum)
107                 //fmt.Printf("add %d to array",currentNum)

```

```

108         break
109     }
110     if currentNum%primeNum==0{
111         //not prime
112         break
113     }
114 }
115 }
116 // PrimeArray populate completed
117 // output the prime number in the list first
118 if primeCheckMin< PrimeArray[len(PrimeArray)-1]{
119     for _,primeNum:= range PrimeArray {
120         if primeNum >= primeCheckMin{
121             //fmt.Printf("%d is a prime number \n",primeNum)
122             OutputCounter++
123         }
124     }
125     primeCheckMin= PrimeArray[len(PrimeArray)-1]+2
126 }
127 if primeCheckMin%2==0{
128     primeCheckMin++
129 }
130 // start to use the prime list to check for larger prime number
131 // use multithreading for larger dataset
132 threadNum:=64
133 // use single thread for little dataset
134 if (primeCheckMax-primeCheckMin)<=10{
135     threadNum = 1
136 }else if (primeCheckMax-primeCheckMin)/2<=threadNum{
137     threadNum = (primeCheckMax-primeCheckMin)/2
138 }
139
140 ResultChan = make(chan int,threadNum*100)
141 ThreadLock = sync.WaitGroup{}
142 // use PrimeArray to check larger prime number
143 for i:=0;i<threadNum;i++){
144     // add a lock for each thread to avoid early exit
145     ThreadLock.Add(1)
146     // provision specific range for each thread
147     subCheckStart := primeCheckMin+
148     int(math.Floor(float64(i)*float64(primeCheckMax-
149     primeCheckMin)/float64(threadNum)))
149     subCheckEnd := primeCheckMin+
150     int(math.Floor(float64(i+1)*float64(primeCheckMax-
151     primeCheckMin)/float64(threadNum)))-1
152     if subCheckStart%2==0{
153         subCheckStart+=1
154     }
155     if i == threadNum-1{
156         subCheckEnd++
157     }
158     //log.Printf("start %d,end %d",subCheckStart,subCheckEnd)
159     // start an async go routine for specific data range
160     go CheckPrimesSub(subCheckStart,subCheckEnd)
161 }
162 OutputLock.Add(1)
163 // start an async go routine to output
164 go outputResult()

```

```

162     ThreadLock.Wait()
163     fmt.Printf("finish calculation in %f
second\n",time.Since(start).Seconds())
164     close(ResultChan)
165     outputLock.Wait()
166     return OutputCounter
167 }
168
169 // outputResult is called by go routine to read from ResultChan channel and
output/count
170 func outputResult(){
171     for primeNum := range ResultChan{
172         OutputCounter++
173         if !JustCount{
174             fmt.Printf("%d is a prime number\n",primeNum)
175         }
176     }
177     outputLock.Done()
178 }
179
180 // CheckPrimeSub is called by go routine to calculate prime number in range
181 func CheckPrimeSub(primeCheckMin,primeCheckMax int){
182     for currentNum :=
primeCheckMin;currentNum<=primeCheckMax;currentNum+=2{
183         checkMax := int(math.Ceil(math.Sqrt(float64(currentNum))))
184         //check with each prime number
185         for _,primeNum := range PrimeArray {
186             if primeNum> checkMax{
187                 //this is a prime number
188                 ResultChan <- currentNum
189                 break
190             }
191             if currentNum%primeNum==0{
192                 //not prime
193                 break
194             }
195         }
196     }
197     ThreadLock.Done()
198 }
199
200 // main takes two positive int input for range and a -counting flag for
only count how many prime number is in range
201 // The indirect subprogram is defined at primeMethod
202 // primeMethod is called later on, by the input size
203 // primeMethod will choose the appropriate function from
IndirectMethodsGTE10000/IndirectMethodsGTE100/IndirectMethodsLT100
204 // the Theory used in defining a prime number is:
205 // if a number can't be fully divided by each prime number less than the
root(sqrt) of the number. Then this number is a prime number
206 func main() {
207     flag.BoolVar(&JustCount, "counting", false, "set this flag for massive
dataset")
208     flag.Parse()
209     primeCheckStartStr := flag.Args()[0]
210     primeCheckEndStr := flag.Args()[1]
211     primeCheckStart,err := strconv.Atoi(primeCheckStartStr);if err!=nil{
212         log.Fatalf("input1 is not valid int")

```

```

213     }
214     primeCheckEnd,err := strconv.Atoi(primeCheckEndStr);if err!=nil{
215         log.Fatalf("input2 is not valid int")
216     }
217     if primeCheckStart>primeCheckEnd{
218         //swap
219         primeCheckStart+=primeCheckEnd
220         primeCheckEnd=primeCheckStart-primeCheckEnd
221         primeCheckStart=primeCheckStart-primeCheckEnd
222     }
223     if primeCheckStart<0{
224         log.Fatalf("minus input detected")
225     }
226     // use of Call() method
227     // decide which function to use
228     var funcToUse interface{}
229     if primeCheckEnd>=10000{
230         funcToUse = new(IndirectMethodsGTE10000)
231     }else if primeCheckEnd >=100{
232         funcToUse = new(IndirectMethodsGTE100)
233     }else{
234         funcToUse = new(IndirectMethodsLT100)
235     }
236
237     fmt.Printf("outputting prime number between
%d,%d\n",primeCheckStart,primeCheckEnd)
238     // define the function that will be used later
239     primeMethod := reflect.ValueOf(funcToUse).MethodByName("CheckPrime")
240     reflectValue := make([]reflect.Value, primeMethod.Type().NumIn())
241     reflectValue[0]=reflect.ValueOf(primeCheckStart)
242     reflectValue[1]=reflect.ValueOf(primeCheckEnd)
243     // indirectly call the function upon runtime
244     val := primeMethod.Call(reflectValue)
245     if val[0].Int()==0{
246         fmt.Printf("no prime number between
%d,%d\n",primeCheckStart,primeCheckEnd)
247     }else{
248         fmt.Printf("%d prime number between
%d,%d\n",val[0].Int(),primeCheckStart,primeCheckEnd)
249     }
250 }

```