```
package main

import (
	"fmt"
	"math"
	"math/rand"
	"os"
	"strings"
	"time"
)

//generate a sequence of l,r,f
func RandomFold(length int) []string {

	var command string
	foldList := make([]string, 0)
	//produce a slice with the same length of HP sequence
	//n-length sequence, n-1 command
	for i := 0; i < length-1; i++ {
		//use time to produce a seed
		seed := time.Now().UTC().UnixNano()
		//use the seed to break the current state
		rand.Seed(seed)
		//produce three different case
		randNum := rand.Int() % 3
		//parse the random number to the command
		switch randNum {
		case 0:
			command = "l"
		case 1:
			command = "r"
		case 2:
			command = "f"
		}
		foldList = append(foldList, command)
	}
	return foldList
}

//"draw" the fold on to a 2D matrix
func DrawFold(foldList, seq []string) [][]string {
	//sufficiently large
	size := 100
	start := size / 2
	//initialize the 2D matrix, with sufficient large size
	seqMat := make([][]string, size)
	for i := 0; i < len(seqMat); i++ {
		seqMat[i] = make([]string, size)
	}
	//start from the center
	x, y := start, start
	seqMat[x][y] = seq[0]
	//it's a digraph, we need to specify the direction for the nodes
	//"f":front,"l":left,"r":right,"d":down
```

```
pointTo := "f"
//each command present a new node
for j := 0; j < len(foldList); j++ {
      //command varies with direction
      switch pointTo {
      case "f":
            switch string(foldList[j]) {
            //command will be "l","r","f"
            case "l":
                  //show conncection
                  seqMat[x-1][y] = "-"
                  if seqMat[x-2][y] != "" {
                        seqMat[x-2][y] = seq[j+1]
                        pointTo = "l"
                  }
            case "r":
                  seqMat[x+1][y] = "-"
                  if seqMat[x+2][y] != "" {
                        seqMat[x+2][y] = seq[j+1]
                        pointTo = "r"
                  }
            case "f": //front+front=front
                  seqMat[x][y+1] = "-"
                  if seqMat[x][y+2] != "" {
                        seqMat[x][y+2] = seq[j+1]
                  }
            }
      case "l":
            switch string(foldList[j]) {
            case "l": //left + left = down
                  seqMat[x][y-1] = "-"
                  if seqMat[x][y-2] != "" {
                        seqMat[x][y-2] = seq[j+1]
                        pointTo = "d"
                  }
            case "r": //left + right =front
                  seqMat[x][y+1] = "-"
                  if seqMat[x][y+2] != "" {
                        seqMat[x][y+2] = seq[j+1]
                        pointTo = "f"
                  }
            case "f": //left+front=left
                  seqMat[x-1][y] = "-"
                  if seqMat[x-2][y] != "" {
                        seqMat[x-2][y] = seq[j+1]
                  }
            }
      case "r":
            switch string(foldList[j]) {
            case "l": //right+left=front
                  seqMat[x][y+1] = "-"
                  if seqMat[x][y+2] != "" {
                        seqMat[x][y+2] = seq[j+1]
                        pointTo = "f"
```

```
                        }
                  case "r": //right+right=down
                        seqMat[x][y-1] = "-"
                        if seqMat[x][y-2] != "" {
                              seqMat[x][y-2] = seq[j+1]
                              pointTo = "d"
                        }
                  case "f": //right+front=right
                        seqMat[x+1][y] = "-"
                        if seqMat[x+2][y] != "" {
                              seqMat[x+2][y] = seq[j+1]
                        }
                  }
            case "d":
                  switch string(foldList[j]) {
                  case "l": //down+left=left
                        seqMat[x-1][y] = "-"
                        if seqMat[x-2][y] != "" {
                              seqMat[x-2][y] = seq[j+1]
                              pointTo = "l"
                        }
                  case "r":
                        seqMat[x+1][y] = "-"
                        if seqMat[x+2][y] != "" {
                              seqMat[x+2][y] = seq[j+1]
                              pointTo = "r"
                        }
                  case "f":
                        seqMat[x][y-1] = "-"
                        if seqMat[x][y-2] != "" {
                              seqMat[x][y-2] = seq[j+1]
                        }
                  }
            }
      }
      return seqMat
}

func Infield(r, c int, seqMat [][]string) bool {
      switch {
      case r < 0 || r > len(seqMat):
            return false
      case c < 0 || c > len(seqMat):
            return false
      default:
            return true
      }
}

//compute the energy of a fold S and sequence p
func Energy(foldList, seq []string) int {

      seqMat := DrawFold(foldList, seq)
      //loop over all squares,calculate the sum of p*s
```

```
	//we only consider "H" since pi=0 for "P"
	var sum int
	for i := 2; i < len(seqMat); i++ {
		for j := 2; j < len(seqMat); j++ {
			//only consider "H"
			if seqMat[i][j] == "H" {
				//adjacent square has amino acid and not connected
				if (seqMat[i-2][j] == "H" || seqMat[i-2][j] == "P") &&
seqMat[i-1][j] != "-" {
					sum += 1
				}
				if (seqMat[i+2][j] == "H" || seqMat[i+2][j] == "P") &&
seqMat[i+1][j] != "-" {
					sum += 1
				}
				if (seqMat[i][j-2] == "H" || seqMat[i][j-2] == "P") &&
seqMat[i][j-1] != "-" {
					sum += 1
				}
				if (seqMat[i][j+2] == "H" || seqMat[i][j+2] == "P") &&
seqMat[i][j+1] != "-" {
					sum += 1
				}
				if seqMat[i-2][j-2] == "H" || seqMat[i-2][j-2] == "P"
{ //diagonal elements won't be connected
					sum += 1
				}
				if seqMat[i+2][j+2] == "H" || seqMat[i+2][j+2] == "P" {
					sum += 1
				}
				if seqMat[i-2][j+2] == "H" || seqMat[i-2][j+2] == "P" {
					sum += 1
				}
				if seqMat[i+2][j-2] == "H" || seqMat[i+2][j-2] == "P" {
					sum += 1
				}
			}
		}
	}
	return sum
}

//randomly change commands
func RandomFoleChange(foldList []string) []string {
	//randomly change one of the commands
	seed := time.Now().UTC().UnixNano()
	//break the current state
	rand.Seed(seed)
	//choose one of the index
	idx := rand.Int() % len(foldList)
	idx_cmd := rand.Int() % 2
	//replace the target commands
	switch foldList[idx] {
	case "l":
```

```
            if idx_cmd == 0 {
                  foldList[idx] = "r"
            } else {
                  foldList[idx] = "f"
            }
      case "r":
            if idx_cmd == 0 {
                  foldList[idx] = "l"
            } else {
                  foldList[idx] = "f"
            }
      case "f":
            if idx_cmd == 0 {
                  foldList[idx] = "l"
            } else {
                  foldList[idx] = "r"
            }
      }
      return foldList
}

//return the lowest energy fold, use annealing method
func OptimizeFold(seq []string) ([]string, int) {

      maxTime := 100000           //loop time
      m := 10 * len(seq)          //after m iterations, stop,m=10n
      k := float64(6 * len(seq))  //parameters
      T := float64(10 * len(seq)) //parameters

      var iter int //iterations
      var foldList []string
      var foldList_cur []string
      var q float64 //probability
      var e_cur int //the energy of new structure

      //random structure
      foldList = RandomFold(len(seq))
      e_origin := Energy(foldList, seq)

      //set i=1
      for i := 1; i < maxTime && (iter < m); i++ {

            //produce a float number as probability between (0,1)
            seed := time.Now().UTC().UnixNano()
            rand.Seed(seed)
            prob := rand.Float64()
            //change a random letter
            foldList_cur = RandomFoleChange(foldList)
            e_cur = Energy(foldList_cur, seq)

            //if E'<E,set S[i+1]=S'
            if e_cur < e_origin {
                  foldList = foldList_cur
                  iter = 0 //after change, recount the iterations
```

```
            } else if e_cur == e_origin {
                  iter += 1 //not change
            } else {
                  //E'>E, q=exp(-(E'-E)/kT), q will be the probability
                  q = math.Exp(-(float64(e_cur) - float64(e_origin)) / (k * T))
                  if prob < q { //q probability that set new sequence
                        foldList = foldList_cur
                        e_origin = e_cur //new energy standard
                        iter = 0         //after change
                  } else {
                        iter += 1 //not change
                  }
            }
            if i%100 == 0 {
                  T *= 0.999
            }
      }
      return foldList, e_origin
}

//draw the real fold and save image
func PaintFold(foldList, seq []string) {

      //size is 100, so we draw 100*n squares
      w := 500
      h := 500
      pic := CreateNewCanvas(w, h)
      //set parameters
      pic.SetLineWidth(1)

      //start from the center
      x := float64(w / 2)
      y := float64(h / 2)
      pic.gc.ArcTo(x, y, 5, 5, 0, 2*math.Pi)
      if seq[0] == "H" {
            white := MakeColor(255, 255, 255)
            pic.SetFillColor(white)
            pic.Stroke() //draw a white circle for H
      } else {
            black := MakeColor(0, 0, 0)
            pic.SetFillColor(black)
            pic.Stroke() //black circle for P
      }
      //now start drawing
      pic.MoveTo(x, y)
      //first direction is front
      angle := math.Pi / 2
      increment := math.Pi / 2
      //loop over all commands
      for i := 0; i < len(foldList); i++ {
            if foldList[i] == "l" { //front+left=left=pi
                  angle += increment
            } else if foldList[i] == "r" { //front+right=right
                  angle -= increment
```

```
            }
            //produce next point based on angels
            x += 10 * math.Cos(angle)
            y -= 10 * math.Sin(angle)
            //conncet two points
            pic.LineTo(x, y)
            //then next point,the same as above
            if seq[i+1] == "H" {
                  pic.gc.ArcTo(x, y, 5, 5, 0, 2*math.Pi)
                  white := MakeColor(255, 255, 255)
                  pic.SetFillColor(white)
                  pic.Stroke() //draw a white circle for H
            } else {
                  pic.gc.ArcTo(x, y, 5, 5, 0, 2*math.Pi)
                  black := MakeColor(0, 0, 0)
                  pic.SetFillColor(black)
                  pic.Stroke() //black circle for P
            }
      }
      pic.Stroke()
      pic.SaveToPNG("fold.png")

}

func main() {

      //first parse the arguments from the command line
      //needs to run the sequence like HHPHPHPHPHPHPHPH, at least HP
      if len(os.Args) <= 1 {
            fmt.Println("Error: a sequence of HP is needed")
            os.Exit(1)
      }

      //store the HP sequence
      var seq []string
      //the arguments will be sequence of HP
      seqList := os.Args[1]
      //parse the command line sequence
      for _, i := range seqList {
            if (string(i) != "H") && (string(i) != "P") {
                  fmt.Println("Error: not valid sequence")
                  fmt.Println("Please enter a sequence of H and P")
                  os.Exit(1)
            }
            seq = append(seq, string(i))
      }
      foldList, erg := OptimizeFold(seq)

      fmt.Println("Energy:", erg)
      fmt.Println("Structure", strings.Join(foldList, "")) //the concatenation
of the command list

      PaintFold(foldList, seq)
```

```
}
```