KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)

Laboratorinių darbų ataskaita

Atliko:

IFF – 6/8 gr. studentas

Tadas Laurinaitis

2019 m. birželio 5 d.

Priėmė:

Lekt. Evaldas Guogis

KAUNAS 2019

TURINYS

1. Python arba Ruby (L1) 3

1.1. Darbo užduotis 3

1.2. Programos tekstas 3

1.3. Pradiniai duomenys ir rezultatai 4

2. Scala (L2) 4

3. Haskell (L3) 4

4. Prolog (L4) 4

#### 1.1 Darbo užduotis

Pav #1 Darbo užduotis

#### 1.2 Programos tekstas

#Number class used to store a number and a number which shows a cumulative sum of each numbers divisors from 1 to number

class Number:

def \_\_init\_\_(self, number, cumulativeSum):

self.number = number

self.cumulativeSum = cumulativeSum

def get\_number(self):

return self.number

#finds sum of all viable divisors of number n

def findSumOfDivisors(n):

sum = 0

for x in range(2, int(n)):

z = n / x #temporary result of division

if z == int(z):

sum = sum + z

return sum

#finds cumulative sum of divisors for numbers 1 to Number.number

def findCumulativeSumOfDivisors(Number):

for x in range(0, Number.number + 1):

Number.cumulativeSum = Number.cumulativeSum + findSumOfDivisors(x)

print("Cumulative sum of divisors of number n: " + str(Number.number) + " is: " + str(Number.cumulativeSum))

return Number

#reads data from file into integer array

def readIntoArray(fileName):

array = []

with open('data.txt') as f:

for line in f: # read all lines

array.append(int(line))

return array

#finds results for all integers in array

def findResults(array):

numberArray = []

for x in array:

temp = Number(x, 0)

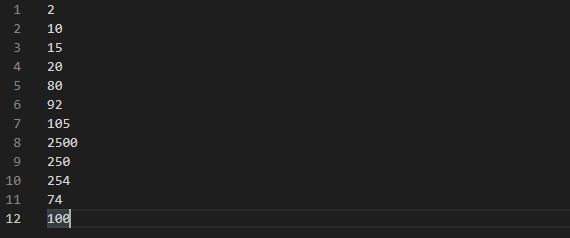
temp = findCumulativeSumOfDivisors(temp)

numberArray.append(temp)

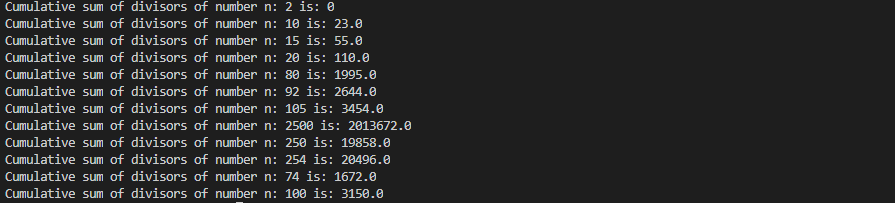
array = readIntoArray("data.txt")

findResults(array)

#### 1.3 Pradiniai duomenys ir rezultatai

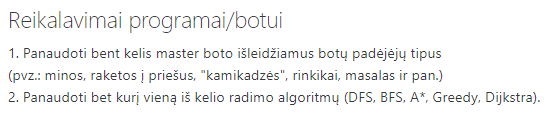


Pav #2 Duomenų failas



Pav #3 Gauti rezultatai

#### Darbo užduotis



Pav #1 Darbo užduotis

#### 1.2 Programos tekstas

// Example Bot #1: The Reference Bot

/\*\* This bot builds a 'direction value map' that assigns an attractiveness score to

\* each of the eight available 45-degree directions. Additional behaviors:

\* - aggressive missiles: approach an enemy master, then explode

\* - defensive missiles: approach an enemy slave and annihilate it

\*

\* The master bot uses the following state parameters:

\* - dontFireAggressiveMissileUntil

\* - dontFireDefensiveMissileUntil

\* - lastDirection

\* The mini-bots use the following state parameters:

\* - mood = Aggressive | Defensive | Lurking

\* - target = remaining offset to target location

\*/

import scala.collection.mutable.ArrayBuffer

object ControlFunction

{

def forMaster(bot: Bot) {

// demo: log the view of the master bot into the debug output (if running in the browser sandbox)

// bot.log(bot.view.cells.grouped(31).mkString("\n"))

val (directionValue, nearestEnemyMaster, nearestEnemySlave, nearestEnemyMasterAround) = analyzeViewAsMaster(bot.view)

val dontFireAggressiveMissileUntil = bot.inputAsIntOrElse("dontFireAggressiveMissileUntil", -1)

val dontFireDefensiveMissileUntil = bot.inputAsIntOrElse("dontFireDefensiveMissileUntil", -1)

//---Mano---

val dontSpawnGatherer = bot.inputAsIntOrElse("dontSpawnGatherer", -1)

val dontSpawnMine = bot.inputAsIntOrElse("dontSpawnMine", -1)

//----------

val lastDirection = bot.inputAsIntOrElse("lastDirection", 0)

// determine movement direction

directionValue(lastDirection) += 10 // try to break ties by favoring the last direction

val bestDirection45 = directionValue.zipWithIndex.maxBy(\_.\_1).\_2

val direction = XY.fromDirection45(bestDirection45)

val ma: String = bot.viewString

//bot.log(""+bot.viewString)

//if()

try{

val instrunctions = Dijkstra(ma)

if(instrunctions.length > 0){

if(instrunctions(0) == "Up"){

bot.move(XY.Up)

bot.log("UP")

}else if(instrunctions(0) == "Down"){

bot.move(XY.Down)

bot.log("DOWN")

}else if(instrunctions(0) == "Left"){

bot.move(XY.Left)

bot.log("LEFT")

}else if(instrunctions(0) == "Right"){

bot.move(XY.Right)

bot.log("RIGHT")

}else{

bot.move(direction)

bot.set("lastDirection" -> bestDirection45)

bot.log("ELSE MOVE")

}

}else{

bot.move(direction)

bot.set("lastDirection" -> bestDirection45)

bot.log("ELSE ELSE MOVE")

}

}catch{

case e: Exception => {

bot.move(direction)

bot.set("lastDirection" -> bestDirection45)

bot.log("Exception MOVE")

}

}

//var (a, b) = prepare(ma)

//Dijkstra(bot.view.cells)

//--------------- MANO ----------------

//bot.log("" + bot.view.cells)

//val tst = Test(bot, bot.viewString)

//tst.stringViewToMatrix()

//bot.log(""+tst.viewMatrix(0)(0))

//for(i <- 0 to 31){

// for(j <- 0 to 31){

// print(""+bot.view.cells(i))

// }

// print("\n")

//}

//var no = Node(-1, -1, 999, List(), "")

//bot.log("" + bot.view.cells)

//var map: Array[Array[String]] = Array.ofDim[String](31, 31)

//var map = Array[String](31,31)

//bot.view.cells(ss)

//bot.log("" + map(15)(15))

//bot.log("" + bot.view.cells(480))

if(dontSpawnGatherer < bot.time && bot.energy > 10000){

bot.spawn(bot.view.center, "mood" -> "Gatherer", "target" -> "", "limit" -> 500)

bot.set("dontSpawnGatherer" -> (bot.time + 10))

}

if(dontSpawnMine < bot.time && bot.energy > 100000 && nearestEnemyMasterAround > -1){

bot.spawn(bot.view.center, "mood" -> "Mine")

bot.set("dontSpawnMine" -> (bot.time + 20))

}

//--------------- /MANO ---------------

if(dontFireAggressiveMissileUntil < bot.time && bot.energy > 100) { // fire attack missile?

nearestEnemyMaster match {

case None => // no-on nearby

case Some(relPos) => // a master is nearby

val unitDelta = relPos.signum

val remainder = relPos - unitDelta // we place slave nearer target, so subtract that from overall delta

bot.spawn(unitDelta, "mood" -> "Aggressive", "target" -> remainder)

bot.set("dontFireAggressiveMissileUntil" -> (bot.time + relPos.stepCount + 1))

}

}

else

if(dontFireDefensiveMissileUntil < bot.time && bot.energy > 100) { // fire defensive missile?

nearestEnemySlave match {

case None => // no-on nearby

case Some(relPos) => // an enemy slave is nearby

if(relPos.stepCount < 8) {

// this one's getting too close!

val unitDelta = relPos.signum

val remainder = relPos - unitDelta // we place slave nearer target, so subtract that from overall delta

bot.spawn(unitDelta, "mood" -> "Defensive", "target" -> remainder)

bot.set("dontFireDefensiveMissileUntil" -> (bot.time + relPos.stepCount + 1))

}

}

}

}

//Buvo

def forSlave(bot: MiniBot) {

bot.inputOrElse("mood", "Lurking") match {

case "Aggressive" => reactAsAggressiveMissile(bot)

case "Defensive" => reactAsDefensiveMissile(bot)

case "Gatherer" => reactAsGatherer(bot)

case "Mine" => reactAsMine(bot)

case s: String => bot.log("unknown mood: " + s)

}

}

//Buvo

def reactAsAggressiveMissile(bot: MiniBot) {

bot.view.offsetToNearest('m') match {

case Some(delta: XY) =>

// another master is visible at the given relative position (i.e. position delta)

// close enough to blow it up?

if(delta.length <= 2) {

// yes -- blow it up!

bot.explode(4)

} else {

// no -- move closer!

bot.move(delta.signum)

bot.set("rx" -> delta.x, "ry" -> delta.y)

}

case None =>

// no target visible -- follow our targeting strategy

val target = bot.inputAsXYOrElse("target", XY.Zero)

// did we arrive at the target?

if(target.isNonZero) {

// no -- keep going

val unitDelta = target.signum // e.g. CellPos(-8,6) => CellPos(-1,1)

bot.move(unitDelta)

// compute the remaining delta and encode it into a new 'target' property

val remainder = target - unitDelta // e.g. = CellPos(-7,5)

bot.set("target" -> remainder)

} else {

// yes -- but we did not detonate yet, and are not pursuing anything?!? => switch purpose

bot.set("mood" -> "Lurking", "target" -> "")

bot.say("Lurking")

}

}

}

//Buvo

def reactAsDefensiveMissile(bot: MiniBot) {

bot.view.offsetToNearest('s') match {

case Some(delta: XY) =>

// another slave is visible at the given relative position (i.e. position delta)

// move closer!

bot.move(delta.signum)

bot.set("rx" -> delta.x, "ry" -> delta.y)

case None =>

// no target visible -- follow our targeting strategy

val target = bot.inputAsXYOrElse("target", XY.Zero)

// did we arrive at the target?

if(target.isNonZero) {

// no -- keep going

val unitDelta = target.signum // e.g. CellPos(-8,6) => CellPos(-1,1)

bot.move(unitDelta)

// compute the remaining delta and encode it into a new 'target' property

val remainder = target - unitDelta // e.g. = CellPos(-7,5)

bot.set("target" -> remainder)

} else {

// yes -- but we did not annihilate yet, and are not pursuing anything?!? => switch purpose

bot.set("mood" -> "Lurking", "target" -> "")

bot.say("Lurking")

}

}

}

//------------------------ MANO -----------------------

def getLow(nodes: ArrayBuffer[Node]) = {

var cost = 999

var index = -1

var tempindex = 0

for (i <- nodes) {

if (cost > i.cost) {

cost = i.cost

index = tempindex

}

tempindex = tempindex + 1

}

index

}

def prepare(bot: String) = {

var map = Array.ofDim[Char](31, 31)

var nodes = Array.ofDim[Node](31, 31)

var ss: Int = 0

for (i <- 0 to 30) {

for (j <- 0 to 30) {

map(i)(j) = bot(ss)

print(map(i)(j) + "")

nodes(i)(j) = new Node(i, j, 999, ArrayBuffer(), map(i)(j))

ss = ss + 1

}

print("\n")

}

var start = new Node(15, 15, 0, ArrayBuffer(), 'M')

for (i <- 0 to 30) {

for (j <- 0 to 30) {

// iskiriamos "sienos" (sonines, virsutines ir apatines reiksmes)

if (i > 0 && j > 0 && i < 30 && j < 30 && map(i)(j) != 'W') {

if (map(i)(j + 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j + 1)

if (map(i)(j - 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j - 1)

if (map(i + 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i + 1)(j)

if (map(i - 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i - 1)(j)

}

// kairaiusias virsutinis kampas

if (i == 0 && j == 0 && map(i)(j) != 'W') {

if (map(i)(j + 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j + 1)

if (map(i + 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i + 1)(j)

}

// desiniausias virsutinis kampas

if (i == 0 && j == 30 && map(i)(j) != 'W') {

if (map(i)(j - 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j - 1)

if (map(i + 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i + 1)(j)

}

// desiniausias apatinis kampas

if (i == 30 && j == 30 && map(i)(j) != 'W') {

if (map(i)(j - 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j - 1)

if (map(i - 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i - 1)(j)

}

// kairiausias apatinis kampas

if (i == 30 && j == 0 && map(i)(j) != 'W') {

if (map(i)(j + 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j + 1)

if (map(i - 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i - 1)(j)

}

// virsus be kampu

if (i == 0 && j > 0 && j < 30 && map(i)(j) != 'W') {

if (map(i)(j + 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j + 1)

if (map(i)(j - 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j - 1)

}

// desine be kampu

if (j == 30 && i > 0 && i < 30 && map(i)(j) != 'W' && map(i)(j) != 'p' && map(

i)(j) != 'b') {

if (map(i + 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i + 1)(j)

if (map(i - 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i - 1)(j)

}

// apacia be kampu

if (i == 30 && j > 0 && j < 30 && map(i)(j) != 'W') {

if (map(i)(j + 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j + 1)

if (map(i)(j - 1) != 'W')

nodes(i)(j).neighbors += nodes(i)(j - 1)

}

// kaire be kampu

if (j == 0 && i > 0 && i < 30 && map(i)(j) != 'W') {

if (map(i + 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i + 1)(j)

if (map(i - 1)(j) != 'W')

nodes(i)(j).neighbors += nodes(i - 1)(j)

}

if (map(i)(j) == 'M') {

start = nodes(i)(j)

nodes(i)(j).cost = 0

}

}

}

(start, nodes)

}

def Dijkstra(bot: String) = {

var (start, nodes) = prepare(bot)

val visited: ArrayBuffer[Node] = new ArrayBuffer[Node]()

val unvisited: ArrayBuffer[Node] = new ArrayBuffer[Node]()

var it: Int = 0

var itt: Int = 0

for (i <- 0 to 30) {

for (j <- 0 to 30) {

if (nodes(i)(j).symbol != 'W'){

unvisited += nodes(i)(j)

it = it + 1

}

if(i == 15 && j == 15){

itt = it

}

}

}

start.cost = 0

start.xi = 15

start.yi = 15

start.previous = null

start.symbol = 'M'

var current = start

while (unvisited.length - 1 > 0) {

for(n <- 0 to unvisited(itt).neighbors.length-1){

if(unvisited(itt).neighbors(n).cost > unvisited(itt).cost+1){

unvisited(itt).neighbors(n).cost = unvisited(itt).cost+1

unvisited(itt).neighbors(n).previous = unvisited(itt)

}

}

visited += unvisited(itt)

unvisited.remove(itt)

var index = getLow(unvisited)

itt = index

}

var target: Node = null

var lowCost: Int = 999

for(v <- visited)

{

if(v.symbol == 'P' || v.symbol == 'B'){

if(v.cost < lowCost){

lowCost = v.cost

target = v

}

}

}

val road: ArrayBuffer[Node] = new ArrayBuffer[Node]()

// veikia

val instrunctions: ArrayBuffer[String] = new ArrayBuffer[String]()

if(target != null)

{

while(target.previous != null)

{

road += target

target = target.previous

}

target = target.previous;

for(r <- road){

if(r.xi < r.previous.xi && nodes(14)(15).symbol != 'W')

instrunctions += "Up"

if(r.xi > r.previous.xi && nodes(16)(15).symbol != 'W')

instrunctions += "Down"

if(r.yi < r.previous.yi && nodes(15)(14).symbol != 'W')

instrunctions += "Left"

if(r.yi > r.previous.yi && nodes(15)(16).symbol != 'W')

instrunctions += "Rigth"

}

}

instrunctions

}

def reactAsMine(bot: MiniBot) {

bot.view.offsetToNearest('m') match {

case Some(delta: XY) =>

// another master is visible at the given relative position (i.e. position delta)

// close enough to blow it up?

if(delta.length <= 1) {

// yes -- blow it up!

bot.explode(2)

} else {

// no -- move closer!

}

case None =>

bot.set("mood" -> "Lurking", "target" -> "")

bot.say("Im a snakeee")

}

}

def reactAsGatherer(bot: MiniBot) {

val (directionValue, nearestEnemyMaster, nearestEnemySlave) = analyzeViewAsGatherer(bot.view)

val lastDirection = bot.inputAsIntOrElse("lastDirection", 0)

directionValue(lastDirection) += 10

val bestDirection45 = directionValue.zipWithIndex.maxBy(\_.\_1).\_2

val direction = XY.fromDirection45(bestDirection45)

bot.move(direction)

bot.set("lastDirection" -> bestDirection45)

}

def analyzeViewAsGatherer(view: View) = {

val directionValue = Array.ofDim[Double](8)

var nearestEnemyMaster: Option[XY] = None

var nearestEnemySlave: Option[XY] = None

val cells = view.cells

val cellCount = cells.length

for(i <- 0 until cellCount) {

val cellRelPos = view.relPosFromIndex(i)

if(cellRelPos.isNonZero) {

val stepDistance = cellRelPos.stepCount

val value: Double = cells(i) match {

case 'm' => // another master: not dangerous, but an obstacle

nearestEnemyMaster = Some(cellRelPos)

if(stepDistance < 2) -1000 else 0

case 's' => // another slave: potentially dangerous?

nearestEnemySlave = Some(cellRelPos)

-100 / stepDistance

case 'S' => // out own slave

0.0

case 'B' => // good beast: valuable, but runs away

if(stepDistance == 1) 600

else if(stepDistance == 2) 300

else (150 - stepDistance \* 15).max(10)

case 'P' => // good plant: less valuable, but does not run

if(stepDistance == 1) 500

else if(stepDistance == 2) 300

else (150 - stepDistance \* 10).max(10)

case 'b' => // bad beast: dangerous, but only if very close

if(stepDistance < 4) -400 / stepDistance else -50 / stepDistance

case 'p' => // bad plant: bad, but only if I step on it

if(stepDistance < 2) -1000 else 0

case 'W' => // wall: harmless, just don't walk into it

if(stepDistance < 2) -1000 else 0

case \_ => 0.0

}

val direction45 = cellRelPos.toDirection45

directionValue(direction45) += value

}

}

(directionValue, nearestEnemyMaster, nearestEnemySlave)

}

//------------------------ /MANO ----------------------

//Buvo

/\*\* Analyze the view, building a map of attractiveness for the 45-degree directions and

\* recording other relevant data, such as the nearest elements of various kinds.

\*/

def analyzeViewAsMaster(view: View) = {

val directionValue = Array.ofDim[Double](8)

var nearestEnemyMaster: Option[XY] = None

var nearestEnemySlave: Option[XY] = None

var nearestEnemyMasterAround: Int = -1

val cells = view.cells

val cellCount = cells.length

for(i <- 0 until cellCount) {

val cellRelPos = view.relPosFromIndex(i)

if(cellRelPos.isNonZero) {

val stepDistance = cellRelPos.stepCount

val value: Double = cells(i) match {

case 'm' => // another master: not dangerous, but an obstacle

nearestEnemyMaster = Some(cellRelPos)

nearestEnemyMasterAround = i

if(stepDistance < 2) -1000 else 0

case 's' => // another slave: potentially dangerous?

nearestEnemySlave = Some(cellRelPos)

-100 / stepDistance

case 'S' => // out own slave

0.0

case 'B' => // good beast: valuable, but runs away

if(stepDistance == 1) 600

else if(stepDistance == 2) 300

else (150 - stepDistance \* 15).max(10)

case 'P' => // good plant: less valuable, but does not run

if(stepDistance == 1) 500

else if(stepDistance == 2) 300

else (150 - stepDistance \* 10).max(10)

case 'b' => // bad beast: dangerous, but only if very close

if(stepDistance < 4) -400 / stepDistance else -50 / stepDistance

case 'p' => // bad plant: bad, but only if I step on it

if(stepDistance < 2) -1000 else 0

case 'W' => // wall: harmless, just don't walk into it

if(stepDistance < 2) -1000 else 0

case \_ => 0.0

}

val direction45 = cellRelPos.toDirection45

directionValue(direction45) += value

}

}

(directionValue, nearestEnemyMaster, nearestEnemySlave, nearestEnemyMasterAround)

}

}

// -------------------------------------------------------------------------------------------------

// Framework

// -------------------------------------------------------------------------------------------------

//Buvo

class ControlFunctionFactory {

def create = (input: String) => {

val (opcode, params) = CommandParser(input)

opcode match {

case "React" =>

val bot = new BotImpl(params)

if( bot.generation == 0 ) {

ControlFunction.forMaster(bot)

} else {

ControlFunction.forSlave(bot)

}

bot.toString

case \_ => "" // OK

}

}

}

//Buvo

trait Bot {

// inputs

def inputOrElse(key: String, fallback: String): String

def inputAsIntOrElse(key: String, fallback: Int): Int

def inputAsXYOrElse(keyPrefix: String, fallback: XY): XY

def view: View

def energy: Int

def time: Int

def generation: Int

def viewString: String

// outputs

def move(delta: XY) : Bot

def say(text: String) : Bot

def status(text: String) : Bot

def spawn(offset: XY, params: (String,Any)\*) : Bot

def set(params: (String,Any)\*) : Bot

def log(text: String) : Bot

}

//Buvo

trait MiniBot extends Bot {

// inputs

def offsetToMaster: XY

// outputs

def explode(blastRadius: Int) : Bot

}

//Buvo

case class BotImpl(inputParams: Map[String, String]) extends MiniBot {

// input

def inputOrElse(key: String, fallback: String) = inputParams.getOrElse(key, fallback)

def inputAsIntOrElse(key: String, fallback: Int) = inputParams.get(key).map(\_.toInt).getOrElse(fallback)

def inputAsXYOrElse(key: String, fallback: XY) = inputParams.get(key).map(s => XY(s)).getOrElse(fallback)

val view = View(inputParams("view"))

val energy = inputParams("energy").toInt

val time = inputParams("time").toInt

val generation = inputParams("generation").toInt

def offsetToMaster = inputAsXYOrElse("master", XY.Zero)

val viewString = inputParams("view")

// output

private var stateParams = Map.empty[String,Any] // holds "Set()" commands

private var commands = "" // holds all other commands

private var debugOutput = "" // holds all "Log()" output

/\*\* Appends a new command to the command string; returns 'this' for fluent API. \*/

private def append(s: String) : Bot = { commands += (if(commands.isEmpty) s else "|" + s); this }

/\*\* Renders commands and stateParams into a control function return string. \*/

override def toString = {

var result = commands

if(!stateParams.isEmpty) {

if(!result.isEmpty) result += "|"

result += stateParams.map(e => e.\_1 + "=" + e.\_2).mkString("Set(",",",")")

}

if(!debugOutput.isEmpty) {

if(!result.isEmpty) result += "|"

result += "Log(text=" + debugOutput + ")"

}

result

}

def log(text: String) = { debugOutput += text + "\n"; this }

def move(direction: XY) = append("Move(direction=" + direction + ")")

def say(text: String) = append("Say(text=" + text + ")")

def status(text: String) = append("Status(text=" + text + ")")

def explode(blastRadius: Int) = append("Explode(size=" + blastRadius + ")")

def spawn(offset: XY, params: (String,Any)\*) =

append("Spawn(direction=" + offset +

(if(params.isEmpty) "" else "," + params.map(e => e.\_1 + "=" + e.\_2).mkString(",")) +

")")

def set(params: (String,Any)\*) = { stateParams ++= params; this }

def set(keyPrefix: String, xy: XY) = { stateParams ++= List(keyPrefix+"x" -> xy.x, keyPrefix+"y" -> xy.y); this }

}

//Buvo

/\*\* Utility methods for parsing strings containing a single command of the format

\* "Command(key=value,key=value,...)"

\*/

object CommandParser {

/\*\* "Command(..)" => ("Command", Map( ("key" -> "value"), ("key" -> "value"), ..}) \*/

def apply(command: String): (String, Map[String, String]) = {

/\*\* "key=value" => ("key","value") \*/

def splitParameterIntoKeyValue(param: String): (String, String) = {

val segments = param.split('=')

(segments(0), if(segments.length>=2) segments(1) else "")

}

val segments = command.split('(')

if( segments.length != 2 )

throw new IllegalStateException("invalid command: " + command)

val opcode = segments(0)

val params = segments(1).dropRight(1).split(',')

val keyValuePairs = params.map(splitParameterIntoKeyValue).toMap

(opcode, keyValuePairs)

}

}

//Buvo

/\*\* Utility class for managing 2D cell coordinates.

\* The coordinate (0,0) corresponds to the top-left corner of the arena on screen.

\* The direction (1,-1) points right and up.

\*/

case class XY(x: Int, y: Int) {

override def toString = x + ":" + y

def isNonZero = x != 0 || y != 0

def isZero = x == 0 && y == 0

def isNonNegative = x >= 0 && y >= 0

def updateX(newX: Int) = XY(newX, y)

def updateY(newY: Int) = XY(x, newY)

def addToX(dx: Int) = XY(x + dx, y)

def addToY(dy: Int) = XY(x, y + dy)

def +(pos: XY) = XY(x + pos.x, y + pos.y)

def -(pos: XY) = XY(x - pos.x, y - pos.y)

def \*(factor: Double) = XY((x \* factor).intValue, (y \* factor).intValue)

def distanceTo(pos: XY): Double = (this - pos).length // Phythagorean

def length: Double = math.sqrt(x \* x + y \* y) // Phythagorean

def stepsTo(pos: XY): Int = (this - pos).stepCount // steps to reach pos: max delta X or Y

def stepCount: Int = x.abs.max(y.abs) // steps from (0,0) to get here: max X or Y

def signum = XY(x.signum, y.signum)

def negate = XY(-x, -y)

def negateX = XY(-x, y)

def negateY = XY(x, -y)

/\*\* Returns the direction index with 'Right' being index 0, then clockwise in 45 degree steps. \*/

def toDirection45: Int = {

val unit = signum

unit.x match {

case -1 =>

unit.y match {

case -1 =>

if(x < y \* 3) Direction45.Left

else if(y < x \* 3) Direction45.Up

else Direction45.UpLeft

case 0 =>

Direction45.Left

case 1 =>

if(-x > y \* 3) Direction45.Left

else if(y > -x \* 3) Direction45.Down

else Direction45.LeftDown

}

case 0 =>

unit.y match {

case 1 => Direction45.Down

case 0 => throw new IllegalArgumentException("cannot compute direction index for (0,0)")

case -1 => Direction45.Up

}

case 1 =>

unit.y match {

case -1 =>

if(x > -y \* 3) Direction45.Right

else if(-y > x \* 3) Direction45.Up

else Direction45.RightUp

case 0 =>

Direction45.Right

case 1 =>

if(x > y \* 3) Direction45.Right

else if(y > x \* 3) Direction45.Down

else Direction45.DownRight

}

}

}

def rotateCounterClockwise45 = XY.fromDirection45((signum.toDirection45 + 1) % 8)

def rotateCounterClockwise90 = XY.fromDirection45((signum.toDirection45 + 2) % 8)

def rotateClockwise45 = XY.fromDirection45((signum.toDirection45 + 7) % 8)

def rotateClockwise90 = XY.fromDirection45((signum.toDirection45 + 6) % 8)

def wrap(boardSize: XY) = {

val fixedX = if(x < 0) boardSize.x + x else if(x >= boardSize.x) x - boardSize.x else x

val fixedY = if(y < 0) boardSize.y + y else if(y >= boardSize.y) y - boardSize.y else y

if(fixedX != x || fixedY != y) XY(fixedX, fixedY) else this

}

}

//Buvo

object XY {

/\*\* Parse an XY value from XY.toString format, e.g. "2:3". \*/

def apply(s: String) : XY = { val a = s.split(':'); XY(a(0).toInt,a(1).toInt) }

val Zero = XY(0, 0)

val One = XY(1, 1)

val Right = XY( 1, 0)

val RightUp = XY( 1, -1)

val Up = XY( 0, -1)

val UpLeft = XY(-1, -1)

val Left = XY(-1, 0)

val LeftDown = XY(-1, 1)

val Down = XY( 0, 1)

val DownRight = XY( 1, 1)

def fromDirection45(index: Int): XY = index match {

case Direction45.Right => Right

case Direction45.RightUp => RightUp

case Direction45.Up => Up

case Direction45.UpLeft => UpLeft

case Direction45.Left => Left

case Direction45.LeftDown => LeftDown

case Direction45.Down => Down

case Direction45.DownRight => DownRight

}

def fromDirection90(index: Int): XY = index match {

case Direction90.Right => Right

case Direction90.Up => Up

case Direction90.Left => Left

case Direction90.Down => Down

}

def apply(array: Array[Int]): XY = XY(array(0), array(1))

}

//Buvo

object Direction45 {

val Right = 0

val RightUp = 1

val Up = 2

val UpLeft = 3

val Left = 4

val LeftDown = 5

val Down = 6

val DownRight = 7

}

//Buvo

object Direction90 {

val Right = 0

val Up = 1

val Left = 2

val Down = 3

}

// Buvo

case class View(cells: String) {

val size = math.sqrt(cells.length).toInt

val center = XY(size / 2, size / 2)

def apply(relPos: XY) = cellAtRelPos(relPos)

def indexFromAbsPos(absPos: XY) = absPos.x + absPos.y \* size

def absPosFromIndex(index: Int) = XY(index % size, index / size)

def absPosFromRelPos(relPos: XY) = relPos + center

def cellAtAbsPos(absPos: XY) = cells.charAt(indexFromAbsPos(absPos))

def indexFromRelPos(relPos: XY) = indexFromAbsPos(absPosFromRelPos(relPos))

def relPosFromAbsPos(absPos: XY) = absPos - center

def relPosFromIndex(index: Int) = relPosFromAbsPos(absPosFromIndex(index))

def cellAtRelPos(relPos: XY) = cells.charAt(indexFromRelPos(relPos))

def offsetToNearest(c: Char) = {

val matchingXY = cells.view.zipWithIndex.filter(\_.\_1 == c)

if( matchingXY.isEmpty )

None

else {

val nearest = matchingXY.map(p => relPosFromIndex(p.\_2)).minBy(\_.length)

Some(nearest)

}

}

}

//----------------------MOVEMENT-----------------------

case class Test(bot: Bot, view: String){

val viewMatrix = Array.ofDim[Char](31,31)

val center = 480

val up = XY(480, 480)

def stringViewToMatrix() = for(i <- 0 to 30) viewMatrix(i) = bot.viewString.substring(i\*31, (i+1)\*31).toArray

bot.log("N " + bot.view.cells(center-31))

bot.log("E " + bot.view.cells(center+1))

bot.log("S " + bot.view.cells(center+31))

bot.log("W " + bot.view.cells(center-1))

}

class Node(var xi: Int, var yi: Int, var cost: Int, var neighbors: ArrayBuffer[Node], var symbol: Char) {

var previous: Node = null

}

//---------------------/MOVEMENT-----------------------

#### 1.3 Pradiniai duomenys ir rezultatai

Rezultatas: Botas vaikščiojantis po žemėlapį, dėliojantis vergus kurie renka taškus ir dėliojantis minas kurios sprogsta priešui priėjus arti.

#### Darbo užduotis

Pav #1 Darbo užduotis

#### 1.2 Programos tekstas

// Tadas Laurinaitis, IFF - 6/8, uzduoties nr. - 294, Divisors https://uva.onlinejudge.org/index.php?option=com\_onlinejudge&Itemid=8&category=4&page=show\_problem&problem=230

open System

open System.IO

let readDataFromFile file =

File.ReadAllLines(file)

let writeResultToFile file L U maxNum maxCount =

let file = File.AppendText(file)

Console.WriteLine("Between {0} and {1}, {2} has a maximum of {3} divisors", L, U, maxNum, maxCount)

file.WriteLine("Between {0} and {1}, {2} has a maximum of {3} divisors", L, U, maxNum, maxCount)

file.Close()

//let rec findDivisionsOfNumber(number : int, divisionCount : int, current : int) =

let rec findDivisionsOfNumber number divisionCount current =

if (number % current = 0 && current <= number) then

let temp1 = divisionCount + 1

let temp2 = current + 1

findDivisionsOfNumber number temp1 temp2

else if (number % current <> 0 && current <= number) then

let temp2 = current + 1

findDivisionsOfNumber number divisionCount temp2

else

let temp = divisionCount

temp

let rec findNumber L U current maxNum maxCount =

let divisionCount = findDivisionsOfNumber current 0 1

//Console.WriteLine(divisionCount)

if(divisionCount > maxCount && current <= U) then

let nextStep = current + 1

let currentMaxNum = current

let currentMaxCount = divisionCount

findNumber L U nextStep currentMaxNum divisionCount

else if (divisionCount <= maxCount && current <= U) then

let nextStep = current + 1

findNumber L U nextStep maxNum maxCount

else

let temp = maxNum

writeResultToFile "Results.txt" L U maxNum maxCount

temp

let rec doStuff current =

let numbers = readDataFromFile("D:\Tadas\KALBUTEORIJA\Fsharp\Lab3\Lab3\Data.txt")

let firstLine = numbers.[0].Split(' ')

let NN = Int32.Parse(firstLine.[0])

if(current > NN) then

printf "Job is done "

else

let strings = numbers.[current].Split(' ')

let L = Int32.Parse(strings.[0])

let U = Int32.Parse(strings.[1])

Console.WriteLine(NN)

Console.WriteLine(current)

Console.WriteLine(L)

Console.WriteLine(U)

if (U - L >= 0 && U - L <= 10000 && current <= NN) then

printf "Data looks fine "

let num = findNumber L U L 0 0

let nextStep = current + 1

doStuff nextStep

else if(U - L < 0 || U - L > 10000 && current <= NN) then

printf "The Data is incorrect "

let nextStep = current + 1

doStuff nextStep

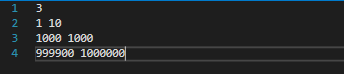
[<EntryPoint>]

let main argv =

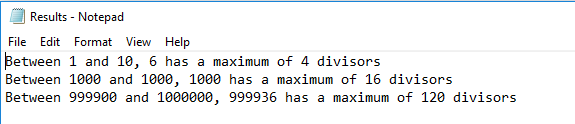
let k = doStuff 1

0

#### 1.3 Pradiniai duomenys ir rezultatai



Pav #2 Duomenų failas



Pav #3 Gauti rezultatai

#### Darbo užduotis

Darbo užduotis:

1. Surasti ar du skaičiai yra kopiriminiai
2. Surasti ar skaičius yra pirminis

#### Programos tekstas

1. % Tadas Laurinaitis IFF - 6/8 suma 39, 12 ir 13
2. % Surasti ar du skaiciai yra kopirminiai - 12 uzduotis
3. coprime(X, Y, Div, Output) :-
4. (X == Div ; Y == Div ->
5. write(" Finished ")
6. ;(X > Div, Y > Div ->
7. write("Still going "),
8. R1 is mod(X, Y),
9. R2 is mod(Y, X),
10. (R1 \== 0, R2 \== 0 ->
11. write("3333\_"),
12. write(Div),
13. R3 is mod(X, Div),
14. R4 is mod(Y, Div),
15. write(R3),
16. write(R4),
17. (( R3 \== 0 ; R4 \== 0) ->
18. write("4444"),
19. Output = 'true',
20. Z is Div + 1,
21. coprime(X, Y, Z, Output)
22. ;(R3 == 0, R4 == 0 ->
23. write("5555"),
24. Output2 = 'false',
25. coprime(100, 100, 100, Output2)
26. )
27. )
28. ;(R1 == 0 ; R2 == 0 ->
29. write("6666"),
30. Output = 'false'
31. )
32. )
33. )
34. ).
35. %Surasti ar skaicius yra pirminis - 13 uzduotis
36. isprime(1) :-
37. write("The number 1 is prime ").
38. isprime(2) :-
39. write("The number 2 is prime ").
40. isprime(X) :-
41. X > 2,
42. isprimee(X, 2, Output).
43. isprimee(X, Y, Output) :-
44. (Y < X ->
45. R1 is mod(X, Y),
46. (R1 \== 0 ->
47. Output = "true",
48. isprimee(X, Y+1, Output)
49. ; (R1 == 0 ->
50. write("The number is not a prime ")
51. )
52. )
53. ; ( Y >= X ->
54. Output = "true",
55. write("The number "),
56. write(X),
57. write(" is prime. ")
58. )
59. ).
60. /\*
61. something(X, Y) :-
62. ( X > Y ; X < Y ->
63. write("11111111"),
64. write("111111111111111111"),
65. something(X - 1,Y)
66. ; ( X = Y ->
67. write("aaa222222222"),
68. write("bbb2222222222222222222"),
69. something(X,Y+1)
70. )
71. ).
72. coprime(X, Y, Divider, Output) :-
73. R1 is mod(X,Y),
74. R2 is mod(Y,X),
75. (R1 \== 0, R2 \== 0 ->
76. R3 is mod(X, Divider),
77. R4 is mod(Y, Divider),
78. (R3 \== 0, R4 \== 0 ->
79. Output = "true",
80. coprime(X, Y, Divider, Output),
81. ; (R3 == 0 ; R4 == 0 ->
82. Output = "false"
83. )
84. )
85. ; (R1 == 0 ; R2 == 0 ->
86. Output = "false";
87. )
88. ).
89. coprime(X, Y, Div, Output) :-
90. write("iejau"),
91. ( X == Div ; Y == Div ->
92. write("as lygus esu"),
93. Output = "true",
94. coprime(X, Y, Div + 1, Output)
95. )
96. R1 is mod(X, Y),
97. R2 is mod(Y, X),
98. (R1 \== 0, R2 \== 0 ->
99. write("praslydau 1"),
100. R3 is mod(X, Div),
101. R4 is mod(Y, Div),
102. (R3 \== 0, R4 \== 0 ->
103. write("praslydau 2"),
104. Output = "true",
105. coprime(X, Y, Div+1, Output)
106. ; (R3 == 0 ; R4 == 0 ->
107. write("nepraslydau 2"),
108. Output = "false"
109. )
110. )
111. ; ( R1 == 0 ; R2 == 0 ->
112. write("nepraslydau 1"),
113. Output = "false"
114. )
115. ).
116. isprimaryy(X, Y, Output) :-
117. (Y < X ->
118. R1 is mod(X, Y),
119. (R1 \== 0 ->
120. Output = "true",
121. isprimaryy(X, Y+1, Output)
122. ; (R1 == 0 ->
123. Output = "false"
124. )
125. )
126. ; ( Y >= X ->
127. Output = "true",
128. write("The number "),
129. write(X),
130. write(" is prime. ")
131. )
132. ).
133. coprimary(X, Y) :-
134. isprimary(X, Output1),
135. isprimary(Y, Output2),
136. Output1 == "true",
137. Output2 == "true",
138. R1 is mod(X,Y),
139. R2 is mod(Y,X),
140. R1 \== 0,
141. R2 \== 0,
142. write("Numbers are coprime").
143. isprimary(1, Output) :-
144. Output = "true",
145. write("The number 1 is prime ").
146. isprimary(2, Output) :-
147. Output = "true",
148. write("The number 2 is prime ").
149. isprimary(X, Output) :-
150. X > 2,
151. isprimaryy(X, 2, Output).
152. isprimaryy(X, Y, Output) :-
153. (Y < X ->
154. R1 is mod(X, Y),
155. (R1 \== 0 ->
156. Output = "true",
157. isprimaryy(X, Y+1, Output)
158. ; (R1 == 0 ->
159. Output = "false"
160. )
161. )
162. ; ( Y >= X ->
163. Output = "true",
164. write("The number "),
165. write(X),
166. write(" is prime. ")
167. )
168. ).
169. jealous(X, Y) :-
170. loves(X, Z),
171. loves(Y, Z).
172. \*/
173. /\*\* <examples>
174. ?- loves(X, mia).
175. ?- jealous(X, Y).
176. \*/

#### 1.3 Pradiniai duomenys ir rezultatai

1užduotis:

Įvedus užklausą *coprime*(15, 18, 2, Out), gaunamas rezultatas Out -> True, o įvedus užklausą

*coprime*(15, 5, 2, Out), Out reikšmė yra false.

2užduotis:

Įvedus užklausą isprimee(3, 2, Output), gaunamas rezultatas Output -> True, o įvedus užklausą

isprimee(4, 2, Output), Output reikšmė yra false.