Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления
Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №1
«Простейший протокол прикладного уровня»
по курсу: «Компьютерные сети»

Выполнил:
Студент группы ИУ9-31Б
Старовойтов А. И.

Проверил:
Посевин Д. П.

Москва, 2022

Цели

Целью данной работы является знакомство с принципами разработки протоколов прикладного уровня и их реализацией на языке Go.

Задачи

Выполнение лабораторной работы состоит из двух частей.

1. Разработать вариант протокола из таблиц 1–3. Протокол должен базироваться на текстовых сообщениях в формате JSON. Результатом разработки протокола должен быть набор типов языка Go, представляющих сообщения, и документация к ним в виде комментариев в исходном тексте.
2. Написать на языке Go клиент и сервер, взаимодействующие по разработанному протоколу.

Основные требования к клиенту и серверу:

1. полная проверка данных, получаемых из сети (необходимо учитывать, что сообщения могут приходить в неправильном формате и в неправильном порядке, а также могут содержать неправильные данные);
2. устойчивость к обрыву соединения;
3. возможность одновременного подключения нескольких клиентов к одному серверу;
4. сервер должен вести подробный лог всех ошибок, а также других важных событий (установка и завершение соединения с клиентом, приём и передача сообщений, и т.п.).

Задание:

37 Протокол редактирования многоугольника на плоскости с возможностью проверки его выпуклости.

Решение

В проекте реализован простейший протокол прикладного уровня для редактирования многоугольника с возможностью проверки его выпуклости.

Многоугольник представлен как список вершин в порядке обхода. Вершины, в свою очередь, описываются как целочисленные координаты точки на плоскости. Поддерживаются следующие операции редактирования многоугольника: - вставка вершины; - удаление вершины; - изменение вершины.

При разработке серверной части, было сделано несколько реалзаций класса многоугольник, проведено сравнение их производительности, а также последовательная оптимизация одного из них. Т.к. при разработке использовалась методология TDD, это позволило свободно изменять реализацию класса, не боясь, что код сломается.

Описание алгоритмов

В реализованном алгоритме выпуклость простого многоугольника определяется по знаку векторного произведения смежных сторон.

Для хранения многоугольника реализованы два класса.

Первый вариант - массив. Реализация простая, но операции удаления и вставки занимают линейное время. Остальные операции O(1) с маленькой константой.

Второй вариант - декартово дерево по неявному ключу. Эта структура данных реализует интерфейс динамического массива. Несет в себе дополнительные расходы по памяти, т.к. является деревом на указателях. С другой стороны, все операции на таком массиве работают за O(logn).

Сравнение производительности

Реализация на массиве и итоговая реализация на декартовом дереве:

```
$ benchstat slice.txt treap_preallocate_small.txt
name      old time/op    new time/op        delta
Init-8       219µs ± 0%        4128µs ± 2%    +1784.93%  (p=0.000 n=20+20)
Insert-8     1.50s ± 2%         0.15s ± 1%      -89.95%  (p=0.000 n=20+19)
Set-8       3.20ms ± 4%       91.60ms ± 2%    +2758.46%  (p=0.000 n=20+20)
Delete-8     1.44s ± 4%         0.10s ± 2%      -93.09%  (p=0.000 n=20+20)
Get-8       1.64ms ±15%       13.39ms ± 1%     +716.84%  (p=0.000 n=20+20)

name      old alloc/op   new alloc/op       delta
Init-8       0.00B        5767216.00B ± 0%        +Inf%  (p=0.000 n=20+19)
Insert-8    8.93MB ± 0%       16.00MB ± 0%      +79.15%  (p=0.000 n=20+20)
Set-8        0.00B       16000606.30B ± 0%        +Inf%  (p=0.000 n=20+20)
Delete-8     0.00B       21353682.60B ± 0%        +Inf%  (p=0.000 n=20+20)
Get-8        0.00B              0.00B              ~     (all equal)

name      old allocs/op  new allocs/op      delta
Init-8        0.00               2.00 ± 0%        +Inf%  (p=0.000 n=20+20)
Insert-8      33.4 ±11%      200047.1 ± 0%  +598843.41%  (p=0.000 n=20+20)
Set-8         0.00          200007.37 ± 0%        +Inf%  (p=0.000 n=20+19)
Delete-8      0.00          266920.55 ± 0%        +Inf%  (p=0.000 n=20+20)
Get-8         0.00               0.00              ~     (all equal)
```

В процессе реализации класса на декартовом дереве, было сделано несколько оптимизаций.

1. Линейное построение дерева по списку вершин.

```
$ benchstat treap.txt treap_fast_build.txt
name      old time/op    new time/op    delta
Init-8       115ms ± 5%       7ms ±14%  -93.73%  (p=0.000 n=20+20)
Insert-8     198ms ± 2%     203ms ± 2%   +2.62%  (p=0.000 n=20+20)
Set-8        170ms ± 3%     130ms ± 2%  -23.20%  (p=0.000 n=20+20)
Delete-8     143ms ± 2%     141ms ± 2%   -1.64%  (p=0.001 n=20+18)
Get-8       20.4ms ± 3%    13.7ms ± 4%  -33.09%  (p=0.000 n=20+20)

name      old alloc/op   new alloc/op   delta
Init-8      49.6MB ± 0%     4.8MB ± 0%  -90.32%  (p=0.000 n=20+19)
Insert-8    40.0MB ± 0%    40.0MB ± 0%     ~     (p=0.659 n=18+20)
Set-8       48.0MB ± 0%    48.0MB ± 0%   -0.00%  (p=0.002 n=20+16)
Delete-8    38.9MB ± 0%    39.2MB ± 0%   +0.57%  (p=0.000 n=20+20)
Get-8        0.00B          0.00B          ~     (all equal)

name      old allocs/op  new allocs/op  delta
Init-8       1.20M ± 0%     0.10M ± 0%  -91.67%  (p=0.000 n=20+20)
Insert-8      800k ± 0%      800k ± 0%   -0.00%  (p=0.028 n=17+20)
Set-8         800k ± 0%      800k ± 0%     ~     (p=0.898 n=20+20)
Delete-8      833k ± 0%      834k ± 0%   +0.06%  (p=0.000 n=20+19)
Get-8         0.00           0.00          ~     (all equal)
```

2. Преаллокация вершин в последовательный участок памяти.

```
$ benchstat treap_fast_build.txt treap_custom_allocator.txt
name      old time/op    new time/op    delta
Init-8      7.22ms ±14%    4.11ms ± 0%   -43.05%  (p=0.000 n=20+18)
Insert-8     203ms ± 2%     172ms ± 2%   -15.61%  (p=0.000 n=20+19)
Set-8        130ms ± 2%     114ms ± 1%   -12.33%  (p=0.000 n=20+20)
Delete-8     141ms ± 2%     119ms ± 3%   -15.23%  (p=0.000 n=18+19)
Get-8       13.7ms ± 4%    13.5ms ± 3%    -1.51%  (p=0.026 n=20+20)

name      old alloc/op   new alloc/op   delta
Init-8      4.80MB ± 0%    5.77MB ± 0%   +20.15%  (p=0.000 n=19+20)
```

```
Insert-8    40.0MB ± 0%    35.2MB ± 0%   -12.00%  (p=0.000 n=20+20)
Set-8       48.0MB ± 0%    48.0MB ± 0%    -0.00%  (p=0.000 n=16+20)
Delete-8    39.2MB ± 0%    39.2MB ± 0%      ~     (p=0.068 n=20+20)
Get-8        0.00B          0.00B           ~     (all equal)

name      old allocs/op  new allocs/op  delta
Init-8        100k ± 0%        0k ± 0%  -100.00%  (p=0.000 n=20+20)
Insert-8      800k ± 0%      700k ± 0%   -12.50%  (p=0.000 n=20+20)
Set-8         800k ± 0%      800k ± 0%    -0.00%  (p=0.003 n=20+20)
Delete-8      834k ± 0%      834k ± 0%    -0.01%  (p=0.040 n=19+20)
Get-8         0.00           0.00           ~     (all equal)
```

3. Преаллокация массивов для подсчета углов при изменении вершины

```
name      old time/op    new time/op    delta
Init-8      4.11ms ± 0%    4.13ms ± 2%   +0.47%  (p=0.048 n=18+20)
Insert-8     172ms ± 2%     150ms ± 1%  -12.50%  (p=0.000 n=19+19)
Set-8        114ms ± 1%      92ms ± 2%  -19.80%  (p=0.000 n=20+20)
Delete-8     119ms ± 3%     100ms ± 2%  -16.48%  (p=0.000 n=19+20)
Get-8       13.5ms ± 3%    13.4ms ± 1%     ~     (p=0.512 n=20+20)

name      old alloc/op   new alloc/op   delta
Init-8      5.77MB ± 0%    5.77MB ± 0%     ~     (all equal)
Insert-8    35.2MB ± 0%    16.0MB ± 0%  -54.54%  (p=0.000 n=20+20)
Set-8       48.0MB ± 0%    16.0MB ± 0%  -66.67%  (p=0.000 n=20+20)
Delete-8    39.2MB ± 0%    21.4MB ± 0%  -45.46%  (p=0.000 n=20+20)
Get-8        0.00B          0.00B          ~     (all equal)

name      old allocs/op  new allocs/op  delta
Init-8        2.00 ± 0%      2.00 ± 0%     ~     (all equal)
Insert-8      700k ± 0%      200k ± 0%  -71.43%  (p=0.000 n=20+20)
Set-8         800k ± 0%      200k ± 0%  -75.00%  (p=0.000 n=20+19)
Delete-8      834k ± 0%      267k ± 0%  -67.99%  (p=0.000 n=20+20)
Get-8         0.00           0.00           ~     (all equal)
```

Листинги и примеры работы

```
$ ./server
2022/10/09 21:36:14 Server started at 0.0.0.0:8080
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:36:38 New client
  ↪  connected: 127.0.0.1:50752
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:36:50 Recieved
  ↪  command: insert
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:36:53 Recieved
  ↪  command: convexity
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:36:59 Recieved
  ↪  command: delete
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:37:02 Recieved
  ↪  command: convexity
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:37:04 Recieved
  ↪  command: quit
 edcf4f0e-0ae1-4cc3-b9c5-92b5a96c31fb 2022/10/09 21:37:04 Ended
  ↪  connection
 bc5f2274-aeb4-4e10-91d0-91231690da94 2022/10/09 21:37:09 New client
  ↪  connected: 127.0.0.1:49126
 bc5f2274-aeb4-4e10-91d0-91231690da94 2022/10/09 21:37:12 Recieved
  ↪  command: quit
 bc5f2274-aeb4-4e10-91d0-91231690da94 2022/10/09 21:37:12 Ended
  ↪  connection
^C
```

```
$ ./client
Usage of ./client:
```

```
$ ./client 127.0.0.1:8080
available commands:
quit - end connection
new - use new polygon
insert - insert vertex
set - set vertex
delete - delete vertex
convexity - check current polygon convexity
help - write this message
command> insert
0-based index> 0
vertex in format "X Y"> 1 2
success
command> convexity
not convex
command> delete
0-based index> 0
success
command> convexity
convex
command> quit
$ ./client 127.0.0.1:8080
available commands:
quit - end connection
new - use new polygon
insert - insert vertex
set - set vertex
delete - delete vertex
convexity - check current polygon convexity
help - write this message
command> quit
```

cmd/server/server.go

```go
package main

import (
    "encoding/json"
    "errors"
    "fmt"
    "io"
    "log"
    "net"
    "os"

    "github.com/google/uuid"
    "github.com/stewkk/iu9-networks/lab1/internal/polygon"
    "github.com/stewkk/iu9-networks/lab1/internal/proto"
)

func main() {
    logger := log.New(os.Stdout, "", log.Ldate|log.Ltime)

    ln, err := net.Listen("tcp", ":8080")
    if err != nil {
        logger.Fatalln(err)
    }
    logger.Println("Server started at 0.0.0.0:8080")
    for {
        conn, err := ln.Accept()
        if err != nil {
            logger.Println(err)
            continue
```

```
        }
        go handle(conn, logger)
    }
}

func handle(conn net.Conn, logger *log.Logger) {
    client := newClient(conn, logger)
    client.log("New client connected: ", client.conn.RemoteAddr())
    var req proto.Request
    for {
        err := json.NewDecoder(conn).Decode(&req)
        if err == io.EOF {
            client.log("Connection interrupted")
            conn.Close()
            return
        }
        if err != nil {
            client.handleError(fmt.Errorf("%w: %v", ErrBadRequest, err))
            continue
        }
        client.log("Recieved command: ", req.Command)

        switch req.Command {
        case "quit":
            conn.Close()
            client.log("Ended connection")
            return
        case "new":
            err = client.handleNew(req)
        case "convexity":
            err = client.handleConvexity(req)
        case "insert":
            err = client.handleInsert(req)
        case "set":
            err = client.handleSet(req)
        case "delete":
            err = client.handleDelete(req)
        }
        if err != nil {
            client.handleError(err)
        }
    }
}

func newClient(conn net.Conn, logger *log.Logger) client {
    polygon, _ := polygon.NewTreapPolygon([]polygon.Vertex{{X: 0, Y: 0},
 ↪  {X: 1, Y: 1}, {X: 1, Y: 0}})
    return client{
        conn:    conn,
        polygon: polygon,
        logger:  logger,
        id:      uuid.New(),
    }
}

var (
    ErrBadRequest = errors.New("bad request")
)

func (c *client) handleError(err error) {
    if err := proto.MakeResponse(c.conn, "error", proto.ErrorResponse{
        Description: err.Error(),
```

```
    }); err != nil {
        c.log(err)
    }
    c.log(err)
}

type client struct {
    id      uuid.UUID
    conn    net.Conn
    polygon polygon.Polygon
    logger  *log.Logger
}

func (c *client) handleNew(req proto.Request) error {
    var payload proto.CreatePolygonRequest
    err := json.Unmarshal(*req.Data, &payload)
    if err != nil {
        return fmt.Errorf("%w: %v", ErrBadRequest, err)
    }

    c.polygon, err = polygon.NewTreapPolygon(payload.Vertices)
    if err != nil {
        return err
    }
    return proto.MakeResponse(c.conn, "ok", nil)
}

func (c *client) handleConvexity(_ proto.Request) error {
    return proto.MakeResponse(c.conn, "result", proto.ConvexityResponse{
        IsConvex: c.polygon.IsConvex(),
    })
}

func (c *client) handleInsert(req proto.Request) error {
    var payload proto.InsertVertexRequest
    err := json.Unmarshal(*req.Data, &payload)
    if err != nil {
        return fmt.Errorf("%w: %v", ErrBadRequest, err)
    }

    err = c.polygon.Insert(payload.Index, payload.Vertex)
    if err != nil {
        return err
    }

    return proto.MakeResponse(c.conn, "ok", nil)
}

func (c *client) handleSet(req proto.Request) error {
    var payload proto.SetVertexRequest
    err := json.Unmarshal(*req.Data, &payload)
    if err != nil {
        return fmt.Errorf("%w: %v", ErrBadRequest, err)
    }

    err = c.polygon.Set(payload.Index, payload.Vertex)
    if err != nil {
        return err
    }

    return proto.MakeResponse(c.conn, "ok", nil)
}
```

```
func (c *client) handleDelete(req proto.Request) error {
    var payload proto.DeleteVertexRequest
    err := json.Unmarshal(*req.Data, &payload)
    if err != nil {
        return fmt.Errorf("%w: %v", ErrBadRequest, err)
    }

    err = c.polygon.Delete(payload.Index)
    if err != nil {
        return err
    }

    return proto.MakeResponse(c.conn, "ok", nil)
}

func (c *client) log(args ...any) {
    prefix := c.logger.Prefix()
    c.logger.SetPrefix(prefix + " " + c.id.String() + " ")
    c.logger.Print(args...)
    c.logger.SetPrefix(prefix)
}
```

cmd/server/server_test.go

```
package main

import (
    "encoding/json"
    "io"
    "log"
    "net"
    "os"
    "testing"

    "github.com/stewkk/iu9-networks/lab1/internal/polygon"
    "github.com/stewkk/iu9-networks/lab1/internal/proto"
    . "gopkg.in/check.v1"
)

// Hook up gocheck into the "go test" runner.
func Test(t *testing.T) { TestingT(t) }

var (
    _ = Suite(&HandleConnectionSuite{})
)

type HandleConnectionSuite struct {
    client net.Conn
    server net.Conn
}

func (s *HandleConnectionSuite) SetUpTest(c *C) {
    s.client, s.server = net.Pipe()
    logger := log.New(os.Stdout, c.TestName(), log.Ldate|log.Ltime)
    go handle(s.server, logger)
}

func (s *HandleConnectionSuite) TestEndsConnectionOnQuitCommand(c *C) {
    proto.MakeRequest(s.client, "quit", nil)

    tmp := []byte{}
    _, err := s.client.Read(tmp)
```

```
    c.Assert(err, Equals, io.EOF)
}

func (s *HandleConnectionSuite) TestNewTriangleReturnsOk(c *C) {
    proto.MakeRequest(s.client, "new", proto.CreatePolygonRequest{
        Vertices: []polygon.Vertex{{X: 1, Y: 2}, {X: 3, Y: 4}, {X: 5, Y:
 ↪  6}},
    })

    res := proto.Response{}
    json.NewDecoder(s.client).Decode(&res)
    c.Assert(res.Status, Equals, "ok")
}

func (s *HandleConnectionSuite) TestNewEmptyPolygonReturnsError(c *C) {
    proto.MakeRequest(s.client, "new", struct{}{})

    res := proto.Response{}
    json.NewDecoder(s.client).Decode(&res)
    c.Assert(res.Status, Equals, "error")

    var errorResponse proto.ErrorResponse
    json.Unmarshal(*res.Data, &errorResponse)
    c.Assert(errorResponse.Description, Matches, InvalidOperationRegex)
}

func (s *HandleConnectionSuite) TestCanCheckConvexity(c *C) {
    proto.MakeRequest(s.client, "convexity", nil)

    var res proto.Response
    json.NewDecoder(s.client).Decode(&res)
    var convexityResponse proto.ConvexityResponse
    json.Unmarshal(*res.Data, &convexityResponse)

    c.Assert(res.Status, Equals, "result")
    c.Assert(convexityResponse, DeepEquals, proto.ConvexityResponse{
        IsConvex: true,
    })
}

func (s *HandleConnectionSuite) TestNonConvexPolygon(c *C) {
    proto.MakeRequest(s.client, "new", proto.CreatePolygonRequest{
        Vertices: []polygon.Vertex{{X: 1, Y: 1}, {X: 2, Y: 2}, {X: 3, Y:
 ↪  3}},
    })
    decoder := json.NewDecoder(s.client)
    var res proto.Response
    decoder.Decode(&res)

    proto.MakeRequest(s.client, "convexity", nil)
    decoder.Decode(&res)
    var payload proto.ConvexityResponse
    json.Unmarshal(*res.Data, &payload)

    c.Assert(payload.IsConvex, Equals, false)
}

func (s *HandleConnectionSuite) TestInsertsVertex(c *C) {
    var res proto.Response
    decoder := json.NewDecoder(s.client)

    proto.MakeRequest(s.client, "insert", proto.InsertVertexRequest{
```

```
        Index:  0,
        Vertex: polygon.Vertex{X: 1, Y: 2},
    })
    decoder.Decode(&res)
    c.Assert(res.Status, Equals, "ok")

    proto.MakeRequest(s.client, "convexity", nil)
    decoder.Decode(&res)
    var payload proto.ConvexityResponse
    json.Unmarshal(*res.Data, &payload)
    c.Assert(payload.IsConvex, Equals, false)
}

func (s *HandleConnectionSuite) TestEndsOnConnectionInterruption(c *C) {
    s.client.Close()
}

var InvalidOperationRegex = `.*invalid operation on polygon.*`
```

internal/proto/proto.go

```
package proto

import (
    "encoding/json"

    "github.com/stewkk/iu9-networks/lab1/internal/polygon"
)

// Request represents client request.
type Request struct {
    // Command can take values:
    // - "quit" ends connection;
    // - "new" applyes following commands to new polygon;
    // - "insert" inserts vertex in polygon;
    // - "delete" removes vertex from polygon;
    // - "set" sets new coordinates to vertex in polygon;
    // - "convexity" checks if current polygon is convex.
    Command string `json:"command"`

    // Data stores payload.
    // For "quit" there is no payload.
    // For "new" there is CreatePolygonRequest.
    // For "insert" there is InsertVertexRequest.
    // For "delete" there is DeleteVertexRequest.
    // For "set" there is SetVertexRequest.
    Data *json.RawMessage `json:"data"`
}

// Response represents server response.
type Response struct {
    // Status represents type of response:
    // - "ok" means success but no data;
    // - "error" means that command failed and data field has
     ↪  ErrorResponse;
    // - "result" means success and data field has corresponding payload.
    Status string `json:"status"`

    // Data represents response payload.
    // For "convexity" command there is ConvexityResponse.
    Data *json.RawMessage `json:"data"`
}
```

```go
// CreatePolygonRequest represents payload for command "new".
type CreatePolygonRequest struct {
    Vertices []polygon.Vertex `json:"vertices"`
}

// GenericVertexRequest represents payload for commands related to
 ↪  vertices.
type GenericVertexRequest struct {
    Index  int            `json:"index"`
    Vertex polygon.Vertex `json:"vertex"`
}

type (
    // InsertVertexRequest represents payload for command "insert".
    InsertVertexRequest GenericVertexRequest
    // SetVertexRequest represents payload for command "set".
    SetVertexRequest GenericVertexRequest
    // DeleteVertexRequest represents payload for command "delete".
    DeleteVertexRequest struct {
        Index int `json:"index"`
    }
)

// ConvexityResponse represents payload for response to command
 ↪  "convexity".
type ConvexityResponse struct {
    IsConvex bool `json:"isConvex"`
}

// ErrorResponse represents payload for error response.
type ErrorResponse struct {
    Description string `json:"description"`
}
```

internal/proto/helpers.go

```go
package proto

import (
    "encoding/json"
    "io"
)

func MakeRequest(w io.Writer, command string, payload any) error {
    var data json.RawMessage
    data, _ = json.Marshal(payload)
    return json.NewEncoder(w).Encode(&Request{
        Command: command,
        Data:    &data,
    })
}

func MakeResponse(w io.Writer, status string, payload any) error {
    var data json.RawMessage
    data, _ = json.Marshal(payload)
    return json.NewEncoder(w).Encode(&Response{
        Status: status,
        Data:   &data,
    })
}
```

internal/polygon/polygon.go

```go
package polygon
```

```
import "errors"

type Polygon interface {
    Vertex(idx int) Vertex
    Vertices() []Vertex
    Insert(idx int, v Vertex) error
    Size() int
    Delete(idx int) error
    Set(idx int, v Vertex) error
    IsConvex() bool
}

var ErrOutOfBounds = errors.New("out of bounds")
var ErrInvalidOperation = errors.New("invalid operation on polygon")

func countPolygonAngleSignSum(vertices []Vertex) (sum int) {
    len := len(vertices)
    sum += angleSign(vertices[len-1], vertices[0], vertices[1])
    sum += angleSign(vertices[len-2], vertices[len-1], vertices[0])
    sum += polylineAngleSignSum(vertices)
    return
}

func polylineAngleSignSum(vertices []Vertex) (sum int) {
    for i := 0; i < len(vertices)-2; i++ {
        sum += angleSign(vertices[i], vertices[i+1], vertices[i+2])
    }
    return
}
```

internal/polygon/slice_polygon.go

```
package polygon

func NewSlicePolygon(vertices []Vertex) (Polygon, error) {
    if len(vertices) < 3 {
        return &slicePolygon{}, ErrInvalidOperation
    }
    return &slicePolygon{vertices: vertices, angleSignSum:
     ↪  countPolygonAngleSignSum(vertices)}, nil
}

type slicePolygon struct {
    vertices     []Vertex
    angleSignSum int
}

func (p slicePolygon) Vertex(idx int) Vertex {
    return p.vertices[idx]
}

func (p *slicePolygon) Insert(idx int, v Vertex) error {
    if idx < 0 || idx > p.Size() {
        return ErrOutOfBounds
    }
    p.angleSignSum -= polylineAngleSignSum(p.verticesOfAngles(idx-1, 2))
    p.vertices = append(p.vertices, Vertex{})
    copy(p.vertices[idx+1:], p.vertices[idx:])
    p.vertices[idx] = v
    p.angleSignSum += polylineAngleSignSum(p.verticesOfAngles(idx-1, 3))
    return nil
}
```

```go
func (p slicePolygon) Size() int {
    return len(p.vertices)
}

func (p slicePolygon) Vertices() []Vertex {
    return p.vertices
}

func (p slicePolygon) verticesOfAngles(from int, count int) []Vertex {
    if from <= 0 {
        return append(p.vertices[p.Size()+from-1:],
         ↪  p.vertices[:count+from+1]...)
    }
    if from+count+1 > p.Size() {
        return append(p.vertices[from-1:],
         ↪  p.vertices[:count+from-p.Size()+1]...)
    }
    return p.vertices[from-1 : from+count+1]
}

func (p *slicePolygon) Delete(idx int) error {
    if p.Size() == 3 {
        return ErrInvalidOperation
    }
    if idx < 0 || idx >= p.Size() {
        return ErrOutOfBounds
    }
    p.angleSignSum -= polylineAngleSignSum(p.verticesOfAngles(idx-1, 3))
    copy(p.vertices[idx:], p.vertices[idx+1:])
    p.vertices = p.vertices[:p.Size()-1]
    p.angleSignSum += polylineAngleSignSum(p.verticesOfAngles(idx-1, 2))
    return nil
}

func (p *slicePolygon) Set(idx int, v Vertex) error {
    if idx < 0 || idx >= p.Size() {
        return ErrOutOfBounds
    }
    p.angleSignSum -= polylineAngleSignSum(p.verticesOfAngles(idx-1, 3))
    p.vertices[idx] = v
    p.angleSignSum += polylineAngleSignSum(p.verticesOfAngles(idx-1, 3))
    return nil
}

func (p *slicePolygon) IsConvex() bool {
    return p.Size() == abs(p.angleSignSum)
}
```

internal/polygon/slice_polygon_test.go

```go
package polygon

import (
    "testing"

    . "gopkg.in/check.v1"
)

// Hook up gocheck into the "go test" runner.
func Test(t *testing.T) { TestingT(t) }

var (
```

```
    _ = Suite(&SliceGetVertexSuite{})
    _ = Suite(&SliceInsertVertexSuite{})
    _ = Suite(&SliceDeleteVertexSuite{})
    _ = Suite(&SliceSetVertexSuite{})
    _ = Suite(&SlicePolygonSizeSuite{})
    _ = Suite(&SliceConvexitySuite{})
    _ = Suite(&SlicePolygonConstructorSuite{})
)

type SliceGetVertexSuite struct {
    p Polygon
}

func (s *SliceGetVertexSuite) SetUpTest(c *C) {
    s.p, _ = NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}})
}

func (s *SliceGetVertexSuite) TestReturnsVertex(c *C) {
    c.Assert(s.p.Vertex(0), Equals, Vertex{1, 2})
    c.Assert(s.p.Vertex(1), Equals, Vertex{3, 4})
    c.Assert(s.p.Vertex(2), Equals, Vertex{5, 6})
}

func (s *SliceGetVertexSuite) TestVerticesReturnsListOfVertices(c *C) {
    c.Assert(s.p.Vertices(), DeepEquals, []Vertex{{1, 2}, {3, 4}, {5,
 ↪  6}})
}

type SliceInsertVertexSuite struct {
    p Polygon
}

func (s *SliceInsertVertexSuite) SetUpTest(c *C) {
    s.p, _ = NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}})
}

func (s *SliceInsertVertexSuite) TestInsertsOnIndex(c *C) {
    s.p.Insert(0, Vertex{9, 10})
    c.Assert(s.p.Vertex(0), Equals, Vertex{9, 10})
}

func (s *SliceInsertVertexSuite) TestAppend(c *C) {
    s.p.Insert(3, Vertex{9, 10})
    c.Assert(s.p.Vertex(3), Equals, Vertex{9, 10})
}

func (s *SliceInsertVertexSuite) TestInsertOutOfBoundsReturnsError(c *C)
 ↪  {
    c.Assert(s.p.Insert(4, Vertex{3, 4}), ErrorMatches, OutOfBoundsRegex)
    c.Assert(s.p.Insert(-1, Vertex{3, 4}), ErrorMatches,
 ↪  OutOfBoundsRegex)
}

type SliceDeleteVertexSuite struct {
    p Polygon
}

func (s *SliceDeleteVertexSuite) SetUpTest(c *C) {
    s.p, _ = NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}, {7, 8}})
}

func (s *SliceDeleteVertexSuite) TestDeletesVertex(c *C) {
```

```
    s.p.Delete(0)
    c.Assert(s.p.Vertices(), DeepEquals, []Vertex{{3, 4}, {5, 6}, {7,
 ↪  8}})
}

func (s *SliceDeleteVertexSuite) TestDeletesLastVertex(c *C) {
    s.p.Delete(3)
    c.Assert(s.p.Vertices(), DeepEquals, []Vertex{{1, 2}, {3, 4}, {5,
 ↪  6}})
}

func (s *SliceDeleteVertexSuite) TestDeleteOutOfBoundsReturnsError(c *C)
 ↪  {
    c.Assert(s.p.Delete(4), ErrorMatches, OutOfBoundsRegex)
    c.Assert(s.p.Delete(-1), ErrorMatches, OutOfBoundsRegex)
}

func (s *SliceDeleteVertexSuite) TestCanNotDeleteFromThreeVertexPolygon(c
 ↪  *C) {
    p, _ := NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}})
    c.Assert(p.Delete(1), ErrorMatches, InvalidOperationRegex)
}

type SliceSetVertexSuite struct {
    p Polygon
}

func (s *SliceSetVertexSuite) SetUpTest(c *C) {
    s.p, _ = NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}})
}

func (s *SliceSetVertexSuite) TestSetVertex(c *C) {
    s.p.Set(0, Vertex{8, 9})
    c.Assert(s.p.Vertex(0), Equals, Vertex{8, 9})
}

func (s *SliceSetVertexSuite) TestSetOutOfBoundsReturnsError(c *C) {
    c.Assert(s.p.Set(3, Vertex{}), ErrorMatches, OutOfBoundsRegex)
    c.Assert(s.p.Set(-1, Vertex{}), ErrorMatches, OutOfBoundsRegex)
}

type SlicePolygonSizeSuite struct {
    p Polygon
}

func (s *SlicePolygonSizeSuite) SetUpTest(c *C) {
    s.p, _ = NewSlicePolygon([]Vertex{{1, 2}, {3, 4}, {5, 6}, {7, 8}})
}

func (s *SlicePolygonSizeSuite) TestEquals4(c *C) {
    c.Assert(s.p.Size(), Equals, 4)
}

func (s *SlicePolygonSizeSuite) TestChangesOnInsert(c *C) {
    s.p.Insert(0, Vertex{})
    c.Assert(s.p.Size(), Equals, 5)
}

func (s *SlicePolygonSizeSuite) TestChangesOnDelete(c *C) {
    s.p.Delete(0)
    c.Assert(s.p.Size(), Equals, 3)
}
```

```
func (s *SlicePolygonSizeSuite) TestNotChangesOnSet(c *C) {
    s.p.Set(0, Vertex{})
    c.Assert(s.p.Size(), Equals, 4)
}

type SliceConvexitySuite struct{}

func (s *SliceConvexitySuite) TestRectangleIsConvex(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 2}, {0, 2}})
    c.Assert(p.IsConvex(), Equals, true)
}

func (s *SliceConvexitySuite) TestTriangleIsConvex(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {10, 0}, {5, 4}})
    c.Assert(p.IsConvex(), Equals, true)
}

func (s *SliceConvexitySuite) TestNotConvex(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {0, 2}, {3, 2}, {1, 1}})
    c.Assert(p.IsConvex(), Equals, false)
}

func (s *SliceConvexitySuite) TestStraightAngleIsNotConvex(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 1}, {1, 2}, {0,
 ↪  2}})
    c.Assert(p.IsConvex(), Equals, false)
}

func (s *SliceConvexitySuite)
 ↪  TestClockwiseAndCounterclockwiseVertexOrdersAreEqual(c *C) {
    clockwise, _ := NewSlicePolygon([]Vertex{{0, 2}, {1, 2}, {1, 0}, {0,
 ↪  0}})
    couterclockwise, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1,
 ↪  2}, {0, 2}})
    c.Assert(clockwise.IsConvex(), Equals, couterclockwise.IsConvex())
}

func (s *SliceConvexitySuite) TestAfterInsert(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 2}, {0, 2}})
    p.Insert(2, Vertex{3, 1})
    c.Assert(p.IsConvex(), Equals, true)
    p.Insert(1, Vertex{-1, 2})
    c.Assert(p.IsConvex(), Equals, false)
}

func (s *SliceConvexitySuite) TestAfterSet(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 2}, {0, 2}})
    p.Set(1, Vertex{2, -1})
    c.Assert(p.IsConvex(), Equals, true)
    p.Set(1, Vertex{1, 2})
    c.Assert(p.IsConvex(), Equals, false)
}

func (s *SliceConvexitySuite) TestAfterDelete(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {0, 2}, {1, 2}})
    c.Assert(p.IsConvex(), Equals, false)
    p.Delete(2)
    c.Assert(p.IsConvex(), Equals, true)
}

func (s *SliceConvexitySuite) TestMirrorRectangle(c *C) {
```

```
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 2}, {0, 2}})
    p.Set(1, Vertex{-1, 0})
    c.Assert(p.IsConvex(), Equals, false)
    p.Set(2, Vertex{-1, 2})
    c.Assert(p.IsConvex(), Equals, true)
}

func (s *SliceConvexitySuite) TestDeleteAndInsert(c *C) {
    p, _ := NewSlicePolygon([]Vertex{{0, 0}, {1, 0}, {1, 2}, {0, 2}})
    p.Delete(0)
    p.Insert(0, Vertex{0, 0})
    c.Assert(p.IsConvex(), Equals, true)
}

type SlicePolygonConstructorSuite struct{}

func (s *SlicePolygonConstructorSuite)
 ↪  TestConstructPolygonFromLessThanThreeVerticesReturnsError(c *C) {
    _, err := NewSlicePolygon([]Vertex{{1, 2}, {3, 4}})
    c.Assert(err, ErrorMatches, InvalidOperationRegex)
}

var OutOfBoundsRegex = `.*out of bounds.*`
var InvalidOperationRegex = `.*invalid operation on polygon.*`
```

internal/polygon/treap_polygon.go

```
package polygon

import (
    "fmt"
    "math/rand"
)

func NewTreapPolygon(vertices []Vertex) (Polygon, error) {
    res := treapPolygon{}
    if len(vertices) < 3 {
        return &res, fmt.Errorf("%w: can't construct polygon from less
         ↪  than 3 vertices", ErrInvalidOperation)
    }
    res.nodes = make([]node, 120000)
    res.root = res.build(vertices)
    res.angleSignSum = countPolygonAngleSignSum(vertices)
    return &res, nil
}

type treapPolygon struct {
    root         *node
    angleSignSum int
    nodes        []node
    last         int
}

func (tree *treapPolygon) Delete(idx int) error {
    if tree.Size() == 3 {
        return fmt.Errorf("%w: can't delete vertex from 3-vertex
         ↪  polygon", ErrInvalidOperation)
    }
    if idx < 0 || idx >= tree.Size() {
        return ErrOutOfBounds
    }
    tree.angleSignSum -=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 3))
```

```
    root := &tree.root
    for {
        leftSize := size((*root).left)
        if idx == leftSize {
            *root = merge((*root).left, (*root).right)
            tree.angleSignSum +=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 2))
            return nil
        }
        (*root).subtreeSize--
        if idx < leftSize {
            root = &(*root).left
        } else {
            root = &(*root).right
            idx -= leftSize + 1
        }
    }
}

func (tree *treapPolygon) Insert(idx int, v Vertex) error {
    if idx < 0 || idx > tree.Size() {
        return ErrOutOfBounds
    }

    tree.angleSignSum -=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 2))

    newNode := tree.newNode(v)
    l, r := split(tree.root, idx)
    tree.root = merge(l, newNode)
    tree.root = merge(tree.root, r)

    tree.angleSignSum +=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 3))

    return nil
}

func (tree *treapPolygon) Set(idx int, v Vertex) error {
    if idx < 0 || idx >= tree.Size() {
        return ErrOutOfBounds
    }
    tree.angleSignSum -=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 3))
    tree.node(idx).v = v
    tree.angleSignSum +=
 ↪  polylineAngleSignSum(tree.verticesOfAngles(idx-1, 3))
    return nil
}

func (tree *treapPolygon) Size() int {
    return tree.root.subtreeSize
}

func (tree *treapPolygon) Vertex(idx int) Vertex {
    return tree.node(idx).v
}

func (tree *treapPolygon) Vertices() (vertices []Vertex) {
    vertices = make([]Vertex, 0, 5)
    var recVertices func(cur *node)
    recVertices = func(cur *node) {
```

```
        if cur == nil {
            return
        }
        recVertices(cur.left)
        vertices = append(vertices, cur.v)
        recVertices(cur.right)
    }
    recVertices(tree.root)
    return
}

func (p *treapPolygon) IsConvex() bool {
    return p.Size() == abs(p.angleSignSum)
}

type node struct {
    left        *node
    right       *node
    v           Vertex
    subtreeSize int
    priority    int
}

func (tree *treapPolygon) newNode(v Vertex) *node {
    node := &tree.nodes[tree.last]
    node.priority = rand.Int()
    node.subtreeSize = 1
    node.v = v
    tree.last++
    return node
}

func (tree *treapPolygon) node(idx int) *node {
    root := tree.root
    for {
        leftSize := size(root.left)
        if idx == leftSize {
            return root
        }
        if idx < leftSize {
            root = root.left
        } else {
            root = root.right
            idx -= leftSize + 1
        }
    }
}

func split(root *node, key int) (l, r *node) {
    if root == nil {
        return nil, nil
    }
    if key <= size(root.left) {
        l, root.left = split(root.left, key)
        root.recalcSize()
        return l, root
    } else {
        root.right, r = split(root.right, key-size(root.left)-1)
        root.recalcSize()
        return root, r
    }
}
```

```
func size(root *node) int {
    if root == nil {
        return 0
    }
    return root.subtreeSize
}

func (root *node) recalcSize() {
    root.subtreeSize = size(root.left) + 1 + size(root.right)
}

func merge(l, r *node) *node {
    if l == nil {
        return r
    }
    if r == nil {
        return l
    }
    if l.priority > r.priority {
        l.right = merge(l.right, r)
        l.recalcSize()
        return l
    } else {
        r.left = merge(l, r.left)
        r.recalcSize()
        return r
    }
}

func (tree *treapPolygon) verticesOfAngles(from, count int) (vertices
 ↪  []Vertex) {
    if from <= 0 {
        vertices = tree.subset(tree.Size()+from-1, tree.Size())
        vertices = append(vertices, tree.subset(0, count+from+1)...)
        return
    }
    if from+count+1 > tree.Size() {
        vertices = tree.subset(from-1, tree.Size())
        vertices = append(vertices, tree.subset(0,
 ↪  count+from-tree.Size()+1)...)
        return
    }
    vertices = tree.subset(from-1, from+count+1)
    return
}

func (tree *treapPolygon) subset(start, end int) (vertices []Vertex) {
    l, m := split(tree.root, start)
    m, r := split(m, end-start)
    tmp := treapPolygon{root: m}
    vertices = tmp.Vertices()
    l = merge(l, m)
    tree.root = merge(l, r)
    return
}

func (tree *treapPolygon) build(vertices []Vertex) *node {
    if len(vertices) == 0 {
        return nil
    }
    m := len(vertices) / 2
```

```
    root := tree.newNode(vertices[m])
    root.left = tree.build(vertices[:m])
    root.right = tree.build(vertices[m+1:])
    heapify(root)
    root.recalcSize()
    return root
}

func heapify(root *node) {
    max := root
    if root.left != nil && root.left.priority > max.priority {
        max = root.left
    }
    if root.right != nil && root.right.priority > max.priority {
        max = root.right
    }
    if max != root {
        max.priority, root.priority = root.priority, max.priority
        heapify(max)
    }
}
```

internal/polygon/utils.go

```
package polygon

func abs(x int) int {
    if x < 0 {
        return -x
    }
    return x
}

func sgn(x int) int {
    if x > 0 {
        return 1
    }
    if x < 0 {
        return -1
    }
    return 0
}
```

internal/polygon/vertex.go

```
package polygon

type Vertex struct {
    X int `json:"x"`
    Y int `json:"y"`
}

type Vector Vertex

func NewVector(s, e Vertex) Vector {
    return Vector{
        X: e.X - s.X,
        Y: e.Y - s.Y,
    }
}

func SinSign(v1, v2 Vector) int {
    return sgn(v1.X*v2.Y - v2.X*v1.Y)
}
```

```go
func angleSign(prev, center, next Vertex) int {
    return SinSign(NewVector(prev, center), NewVector(center, next))
}
```

cmd/client/client.go

```go
package main

import (
    "bufio"
    "encoding/json"
    "errors"
    "flag"
    "fmt"
    "io"
    "net"
    "os"
    "strconv"
    "strings"

    "github.com/stewkk/iu9-networks/lab1/internal/polygon"
    "github.com/stewkk/iu9-networks/lab1/internal/proto"
)

func main() {
    var s server
    s.parseCli()
    err := s.connect()
    if err != nil {
        fmt.Println(err)
        os.Exit(1)
    }
    err = s.interact()
    if err != nil {
        s.conn.Close()
        fmt.Println(err)
        os.Exit(1)
    }
    s.conn.Close()
}

func (s *server) parseCli() {
    flag.CommandLine.Usage = func() {
        fmt.Printf("Usage:  %s ADDRESS\n", os.Args[0])
    }
    flag.Parse()
    if flag.NArg() != 1 {
        flag.Usage()
        os.Exit(1)
    }
    s.addr = flag.Arg(0)
}

func (s *server) connect() error {
    addr, err := net.ResolveTCPAddr("tcp", s.addr)
    if err != nil {
        return err
    }
    s.conn, err = net.DialTCP("tcp", nil, addr)
    if err != nil {
        return err
    }
```

```
    return nil
}

type server struct {
    addr string
    conn net.Conn
}

func (s *server) interact() error {
    r := bufio.NewReader(os.Stdin)
    helpMessage := `available commands:
quit - end connection
new - use new polygon
insert - insert vertex
set - set vertex
delete - delete vertex
convexity - check current polygon convexity
help - write this message`
    fmt.Println(helpMessage)
    for {
        fmt.Print("command> ")
        command, err := r.ReadString('\n')
        command = strings.TrimSpace(command)
        if err == io.EOF {
            return nil
        }
        if err != nil {
            return err
        }
        var payload any = nil
        switch command {
        case "quit":
        case "new":
            vertices, err := readVertices(r)
            if err != nil {
                return err
            }
            payload = proto.CreatePolygonRequest{
                Vertices: vertices,
            }
        case "insert":
            fallthrough
        case "set":
            index, err := readIndex(r)
            if err == io.EOF {
                return nil
            }
            if err != nil {
                return err
            }
            vertex, err := readVertex(r)
            if err == io.EOF {
                return nil
            }
            if err != nil {
                return err
            }
            payload = proto.GenericVertexRequest{
                Index:  index,
                Vertex: vertex,
            }
        case "delete":
```

```
            index, err := readIndex(r)
            if err == io.EOF {
                return nil
            }
            if err != nil {
                return err
            }
            payload = proto.DeleteVertexRequest{
                Index: index,
            }
        case "convexity":
        default:
            fmt.Println(helpMessage)
            continue
        }
        err = proto.MakeRequest(s.conn, command, payload)
        if err != nil {
            return err
        }
        if command == "quit" {
            return nil
        }
        var res proto.Response
        err = json.NewDecoder(s.conn).Decode(&res)
        if err != nil {
            return err
        }
        switch res.Status {
        case "ok":
            fmt.Println("success")
        case "result":
            var payload proto.ConvexityResponse
            err := json.Unmarshal(*res.Data, &payload)
            if err != nil {
                return err
            }
            if payload.IsConvex {
                fmt.Println("convex")
            } else {
                fmt.Println("not convex")
            }
        case "error":
            var payload proto.ErrorResponse
            err := json.Unmarshal(*res.Data, &payload)
            if err != nil {
                return err
            }
            fmt.Println(payload.Description)
        }
    }
}

func readVertices(r *bufio.Reader) ([]polygon.Vertex, error) {
    vertices := []polygon.Vertex{}
    for {
        fmt.Print(`vertex in format "X Y" or "end"> `)
        str, err := r.ReadString('\n')
        if err != nil {
            return nil, err
        }
        if strings.TrimSpace(str) == "end" {
            if len(vertices) < 3 {
```

```
                fmt.Println("enter at least 3 vertices")
                continue
            }
            return vertices, nil
        }
        vertex, err := parseVertex(str)
        if err != nil {
            fmt.Println("invalid input format")
            continue
        }
        vertices = append(vertices, vertex)
    }
}

func readVertex(r *bufio.Reader) (polygon.Vertex, error) {
    for {
        fmt.Print(`vertex in format "X Y"> `)
        str, err := r.ReadString('\n')
        if err != nil {
            return polygon.Vertex{}, err
        }
        vertex, err := parseVertex(str)
        if err != nil {
            fmt.Println("invalid input format")
            continue
        }
        return vertex, nil
    }
}

func readIndex(r *bufio.Reader) (int, error) {
    for {
        fmt.Print(`0-based index> `)
        str, err := r.ReadString('\n')
        if err != nil {
            return 0, err
        }
        index, err := strconv.Atoi(strings.TrimSpace(str))
        if err != nil {
            fmt.Println("invalid input format")
            continue
        }
        return index, nil
    }
}

func parseVertex(str string) (polygon.Vertex, error) {
    fields := strings.Fields(str)
    if len(fields) != 2 {
        return polygon.Vertex{}, ErrParse
    }
    x, err := strconv.Atoi(fields[0])
    if err != nil {
        return polygon.Vertex{}, ErrParse
    }
    y, err := strconv.Atoi(fields[1])
    if err != nil {
        return polygon.Vertex{}, ErrParse
    }
    return polygon.Vertex{
        X: x,
        Y: y,
```

```
    }, nil
}

var ErrParse = errors.New("parse error")
```