Ministry of Education and Science of the Republic of Kazakhstan
Suleyman Demirel University

Gennady Golovkin

Knockoutization of Arrogant Opponents by Nearest Neighbour Punching Algorithm

A thesis submitted for the degree of
Bachelor in Information Systems
(degree code: 5B070300)

Kaskelen, 2018

Ministry of Education and Science of the Republic of Kazakhstan
Suleyman Demirel University
Faculty of Engineering and Natural Sciences

Knockoutization of Arrogant Opponents by Nearest Neighbour Punching Algorithm

A thesis submitted for the degree of
Bachelor in Information Systems
(degree code: 5B070300)

Author: **Gennady Golovkin**

Supervisor: **Abel Sanchez**

Dean of the faculty:
Assist. Prof. Meirambek Zhaparov

Kaskelen, 2018

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Аңдатпа

Головкиннің әуесқой мансабы ұзаққа созылды әрі қанық, оқиғаға толы болды. Генадий бокспен 8 жасынан бастап айналыса бастады. 1993 жылы оны облыстық бокстан өткен жарысқа оның бапкері жіберген болатын. Соңында осы сайыстан 3 жеңіс әкелді. Бұдан кейін ол облыстық, мемлекеттік және халықаралық бокстан жарыстарға үміткер болып қатыса бастады. Осы уақытқа дейін Генадий Головкин өзінің қатысқан 350 жекпе-жегінде тек 5 рет қана жеңіліске ұшыраған болатын.19 жасында шығыс боксында бірінші орын алады да, 2002 жылы бұл жеңісін тағы бір мәрте қайталайды. 2003 жылы Таиландта өткен боксшылар жекпе-жегінде өзінің 4 қарсыласының екеуін нокаутқа жібергені үшін бірінші орынды алады.

Аннотация

Повседневная практика показывает, что начало повседневной работы по формированию позиции играет важную роль в формировании модели развития. Значимость этих проблем настолько очевидна, что консультация с широким активом играет важную роль в формировании дальнейших направлений развития. Задача организации, в особенности же рамки и место обучения кадров в значительной степени обуславливает создание систем массового участия. Равным образом постоянный количественный рост и сфера нашей активности обеспечивает широкому кругу (специалистов) участие в формировании соответствующий условий активизации. Повседневная практика показывает, что новая модель организационной деятельности обеспечивает широкому кругу (специалистов) участие в формировании модели развития.

Contents

Chapter 1

Introduction

1.1 Description

Mr.Maze is an online maze game. It creates mazes using procedural generation, so it will be challenging enough for players. Also users will be able to invite friends and play against them on generated mazes.

1.2 Motivation

We were inspired by Wordle game. It is an easy game with simple rules, so everybody can start playing it. Our case is similar. Everybody is familiar with mazes. Most likely You have played with mazes in childhood, it means You already know the rules. Our mission is only to give You a platform and content.

1.3 Thesis Outline

The first chapter is Introduction chapter. It is this one that you are currently reading. It gives insight into the work done. In Chapter 2 we review related work and formulate the problem to solve. Chapter 3 is describing the solution to the problem. And in Conclusion chapter we conclude our conclusion.

Chapter 2

Backend

2.1 Microservices

Our backend designed in microservice architecture. In this section we will describe advantages and disadvantages of that approach

2.1.1 Pros

Load handling

Each microservice has own resources. It means we can control which of them is going to have most computational power and distribute resources depending on the service's needs

Developing

Developing a new microservice is always easy, because all other services, except the one which is being developed, can be developed independently. Team's work is parallel. So the development stage is shorter

Database

Microservices has own database, so access is not limited for whole system. In monolith access to database is limited to number of database replicas and connection pool and whole system is limited by that. In microservices we have the same limitations, but only for microservice.

2.1.2 Cons

Integration

As we said, mircoservices are good with databases, but in case we need to move data across services, it becomes a problem. Microservices' internal communication

becomes a problem for system's network. Also, requests' idempotency becomes a problem, so we have to use message brockers to guarantee a recieve by another service. If we some data on different microservices are related, we will need to guarantee data's consistency, for us it means, we will load network with another bunch of requests between services

Bug tracking

If some service uses a lot of other services, it becomes hard to track bugs and handle them through all services. In such cases it is good to have some logger, which will collect logs from services and show us.

Refactoring

When we refactor microservices and change data representation or communication protocols, we should be ready for other services to be unable to communicate with current one

2.2 Authorization

2.2.1 JWT

In project we are using JWT (JSON Web Token). Token is divided on three parts by dot.

- Header
- Payload
- Signature

Structure: **xxx.yyy.zzz**

Header

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA (For example see Figure 2.1). Then, this JSON is Base64Url encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data (For example see Figure 2.2). There are three types of claims: registered, public, and private

claims. The payload is then Base64Url encoded to form the second part of the JSON Web Token.

Registered claims These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience), and others.

Public claims These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.

Private claims These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

Signature

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

Figure 2.1: Header

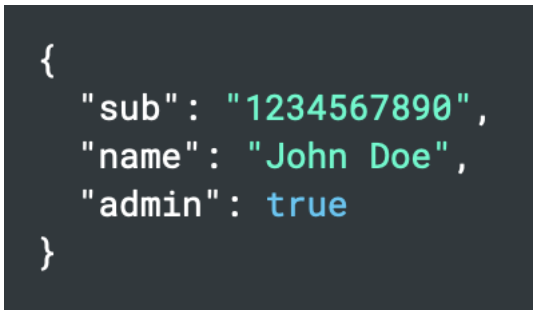

Figure 2.2: Payload

Summary

The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments, so we can easily transfer information like user id, and permissions through this token, and be sure data verified, because it is signed.

2.2.2 Implementation

In our project we developed an authorization service which will sign tokens and refresh them. Also we have verification middleware on each service, so we can be sure token is valid and not expired.

When user credentials passed to server it hashes it and compares to hash we have in database for this user. We keep no plain text passwords and hash functions can't be reversed, so they can not be leaked. In case everything okay it returns token string.

We are using jwt-go package to sign token.

2.3 Database

For our project we chose Postgres database because we are familiar with it, also it is open source, that's why we can use it for free.

SQL databases provide fast data access and data modification. We are not going to delete often or change table structure, so there is no need to use collection based databases like Mongo.

2.4 Maze generation

In this chapter we will explain some structures and approaches to make it clear how we produce valid maze with almost linear time complexity (We will talk about this "almost" a bit later)

2.4.1 Disjoint Set Union

First what we need is disjoint set union (DSU). It is a structure that stores indexes and the set it belongs to. Initially all indexes belong to set with only that index, so it is a representative of this set. DSU can say which set does exact index belong to and can unite two sets.

Data structure

In this code we see DSU structure with three fields.

```go
type DSU struct {
   Size   int
   parent []int
   rank   []int
}

func NewDSU(size int) *DSU {
   ret := &DSU{
      Size:   size,
      parent: make([]int, size),
      rank:   make([]int, size),
   }

   for i := 0; i < size; i++ {
      ret.MakeSet(i)
   }

   return ret
}
func (d *DSU) MakeSet(v int) {
   d.parent[v] = v
   d.rank[v] = 0
}
```

- Size - number of indexes
- parent - an index which is also is in that index's set
- rank - approximated height of tree, in case it is a perfect binary tree, used for optimization. (It is a thing that makes time complexity almost constant, because we will compress tree by reassigning a parent node every time we get a query to find set)

NewDSU function is a constructor, here we are declaring data according to size and setting them an initial value, which is 0 rank and set is same as number of node.

MakeSet method sets node's values to default

Methods

Here we will go through data structure's two methods

```go
func (d *DSU) FindSet(v int) int {
   if v == d.parent[v] {
      return v
   }

   d.parent[v] = d.FindSet(d.parent[v])
```

```
    return d.parent[v]
}

func (d *DSU) UnionSets(a, b int) {
    a = d.FindSet(a)
    b = d.FindSet(b)

    if a != b {
        if d.rank[a] < d.rank[b] {
            a, b = b, a
        }
        d.parent[b] = a
        if d.rank[a] == d.rank[b] {
            d.rank[a]++
        }
    }
}
```

- FindSet - returns a set of node. It goes up through parent array at exiting it reassigns parent value, so each time we make a request we make it method faster
- UnionSet - unites two set. First it find representatives of two set. Then it checks their ranks and assigns a set with bigger rank to the one with the smaller.

Time complexity

Time complexity analysis shows that rank heuristic optimization and path compressing gives us $O(\alpha(n))$ per query. $\alpha(n)$ is an Inverse Ackermann function and it is not exceeding 4 for $n <= 10^{600}$, that why we can consider it as constant.

2.4.2 Eller's algorithm

This algorithm for maze generating works row by row and it is crazy, because on output we will have perfect maze. Each point can be reached and there is exactly one way between them. Algorithm:

1. Initialize the cells of the first row to each exist in their own set.
2. Now, randomly join adjacent cells, but only if they are not in the same set. When joining adjacent cells, merge the cells of both sets into a single set, indicating that all cells in both sets are now connected (there is a path that connects any two cells in the set).

3. For each set, randomly create vertical connections downward to the next row. Each remaining set must have at least one vertical connection. The cells in the next row thus connected must share the set of the cell above them.

4. Flesh out the next row by putting any remaining cells into their own sets.

5. Repeat until the last row is reached.

6. For the last row, join all adjacent cells that do not share a set, and omit the vertical connections, and you're done!

Step 1

```
row = make([]string, width*2+1)
row[0] = "|"
row[width*2] = "|"
row[width*2-1] = "_"

dsu := domain.NewDSU(width)
```

Here we initialize a row and DSU. (width is twice bigger just for usability. We will go through odd indexes, so it makes no sense to algorithm itself)

Step 2

```
rand.Seed(time.Now().UnixNano())
for i := 0; i < width-1; i++ {
   row[2*i+1] = " "
   if dsu.FindSet(i) == dsu.FindSet(i+1) {
      row[2*i+2] = "|"
   } else {
      res := rand.Intn(10)
      if res > 3 {
         row[2*i+2] = " "
         dsu.UnionSets(i, i+1)
      } else {
         row[2*i+2] = "|"
      }
   }
}
```

Here we go through cells and decide if we are adding a wall between them or not.

Step 3

```
sz := map[int]int{}
for i := 0; i < width; i++ {
   sz[dsu.FindSet(i)]++
}
masks := map[int]int64{}
for k, v := range sz {
   maxBitmask := (int64(1) << v) - int64(1)
```

```
            sz[k] = 0
            masks[k] = rand.Int63n(maxBitmask) + 1
        }
        for i := 0; i < width; i++ {
            cur := dsu.FindSet(i)

            if (masks[cur] & (1 << sz[cur])) != 0 {
                row[2*i+1] = " "
            } else {
                row[2*i+1] = "_"
            }
            sz[cur]++
        }

        maze.Rows = append(maze.Rows, strings.Join(row, ""))
```

Here we count number of cells in each set and generating a number between 1 and 2^{cells} as a bitmask, after that we go through cells and adding a wall if bit in bitmask for this set is 0 for this position. As our number varies in $[1, 2^{cells}]$ we will always have at least 1 way down from each set. At the end we are adding this row to the maze

Step 4

```
        newDsu := domain.NewDSU(width)

        for i := 0; i < width-1; i++ {
            if dsu.FindSet(i) == dsu.FindSet(i+1) {
                if row[2*i+1] != "_" && row[2*i+3] != "_" {
                    newDsu.UnionSets(i, i+1)
                }
            }
        }

        dsu = newDsu
```

Reseting dsu to initals and keeping nodes together if they are in the same set

Step 5

```
        for k := 0; k < length-1; k++
```

Looping it

Step 6

```
    for i := 0; i < width-1; i++ {
        row[2*i+1] = "_"
        if dsu.FindSet(i) != dsu.FindSet(i+1) {
            dsu.UnionSets(i, i+1)
            row[2*i+2] = " "
        }
    }
```

```
    maze.Rows = append(maze.Rows, strings.Join(row, ""))
```

If two cells are in different sets, we are connecting them

2.4.3 Summing up

We go through each row multiple times. It is not affecting our time complexity in terms of big O notation, so we just skipping it and consider we are doing it once and have $O(length)$. At each row we are using DSU for each cell, so it gives us $O(width * \alpha(width))$ per row, as we already noticed inverse Ackermann's function can be written out, because it is constant for all reasonable numbers. So we have $O(length * width))$, which is pretty good at this point.

2.5 Multiplayer

TBD

Chapter 3

BBBBB

3.1 B one

Gennady Gennadyevich Golovkin is a legendary middle weight boxer (see Figure 3.1)

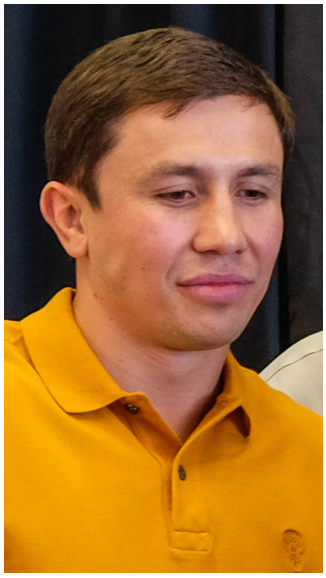

Figure 3.1: Triple G

3.2 B two

3.3 B three

Chapter 4

CCCCC

4.1 C one

Monkey is a beast that can jump. See Appendix B.

4.2 C two

4.3 C three

Chapter 5

Conclusion

Everything is great, but there is a space for future work.

Appendix A

AppendixATitle

A.1 Theorem

A.2 Proof

etc. etc.

Appendix B

AppendixBTitle

Five little monkeys are jumping on the bed.