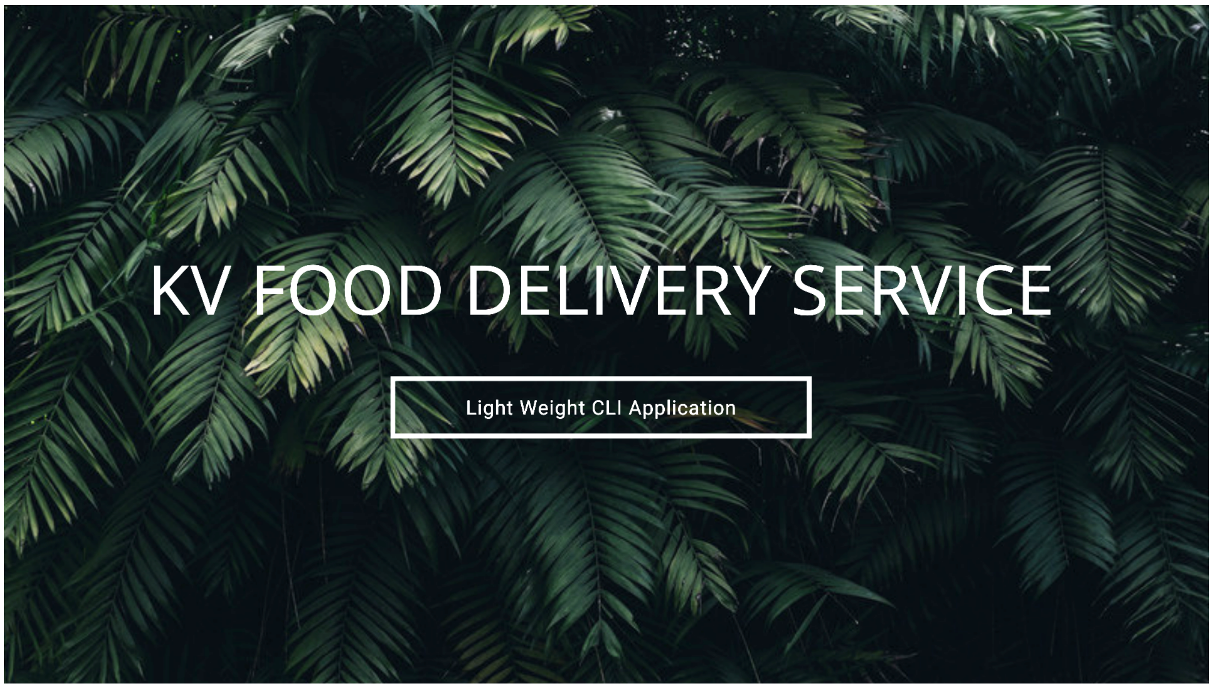
Design Document for KV Food Delivery Service:

Introduction:

The KV app was created in an attempt to solve teething customer service issues that have been affecting customer's average order values, customer satisfaction and ultimate customer loyalty.

While there were alternatives that were created before; there is no single application that can integrate the features that are now available.



Design Considerations:

The CLI application should be lightweight, having the options for offline and/or online data importation. UX of the CLI application must also be user friendly to encourage early adopters to start feedback / app revisions. The main functionality of the app must also allow fast searching of certain data especially merchant information, transaction IDs as well as system queue numbers.

Performance Considerations:

Without querying online databases, databases have to be imported locally, created and stored in memory. Utilizing GO routines are mandatory as part of performance control and flow. The CLI application should also ensure that the data can be populated accurately with the help of channels confirming routine completions before the system can be utilized.

As a query app loaded with features, the features themselves must also be able to present data fast and correctly.

Error and Panic Considerations:

Error handling and panic cases are to be meticulously tested and addressed for.

Handling data, writing and reading can cause panic issues especially with databases that are heavy on slices. The most common panic that can happen in the application is due to array or slice length mismatches that occur. The implementation of a prefix trie (discussed more in the search and auto complete section) can only contain certain characters.

Characters that are recognized and stored in the arrays will cause panics. In light of that; deferred recovery functions have to be utilised in the code to mitigate such issues and recover from potential panics.

Data Structure Implementation

Prefix Trie

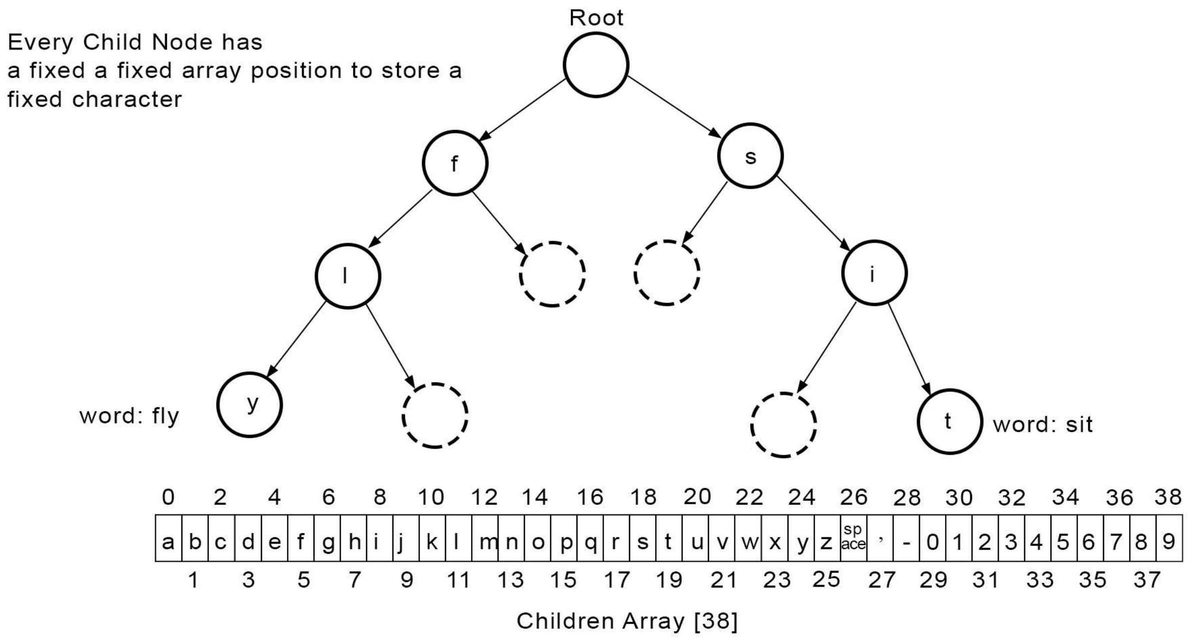

Root
Every Child Node has a fixed a fixed array position to store a fixed character
f
s
l
i
y
t
word: fly
word: sit
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
a b c d e f g h i j k l m n o p q r s t u v w x y z space ' - 0 1 2 3 4 5 6 7 8 9
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37
Children Array [38]

Diagram for a Prefix Trie

A prefix trie, also known as a trie is a tree-like data structure where all the nodes store letters of an alphabet, or a set of characters predetermined by codes.

The array size for each child is **39** where tries that only hold alphabets in either lower or upper case alphabets excluding symbols and numerical values can have an array size of 26.

While tries often take up more space in the memory, they are also very efficient in handling search data. In the application, we don't just store single-word keywords in the food list database. In fact, the trie data structure is also used to store the usernames in the application.

This is an example of what is being stored in the customized Trie structure.

roti prata - ah sik's curry - tampines ave 7

In the above example, you will realise that if we chose a regular 26 array sized Trie, the program would not be able to accommodate the other characters. These characters would not be able to be inserted into the trie.

The *space* , the *apostrophe*, the *hyphen* and the number 7 will be the affected characters.

And because each child node is fixed in size, it is guaranteed that as long as the characters that we are trying to store are within the 39 characters, the trie data structure will not cause panics but be a reliable, searchable data structure.

Please refer to the diagram above to understand the 39 characters that can be stored

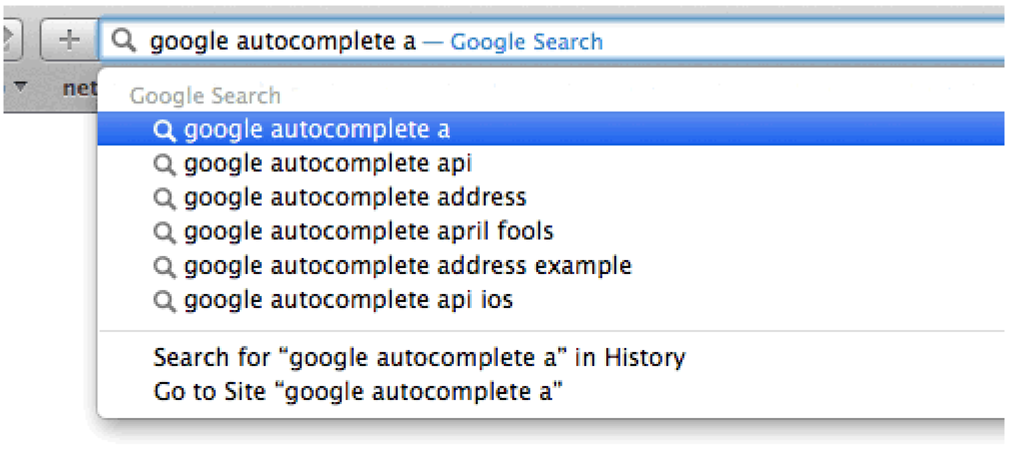
Time Complexity for Trie Data Struct

The **worst case** for looking up and inserting data is O(m) time where m is also the length of a searched string.

The **best case** for looking up data is O(m-n) time where m is the length of a searched string and n = (length of the search string - 1)

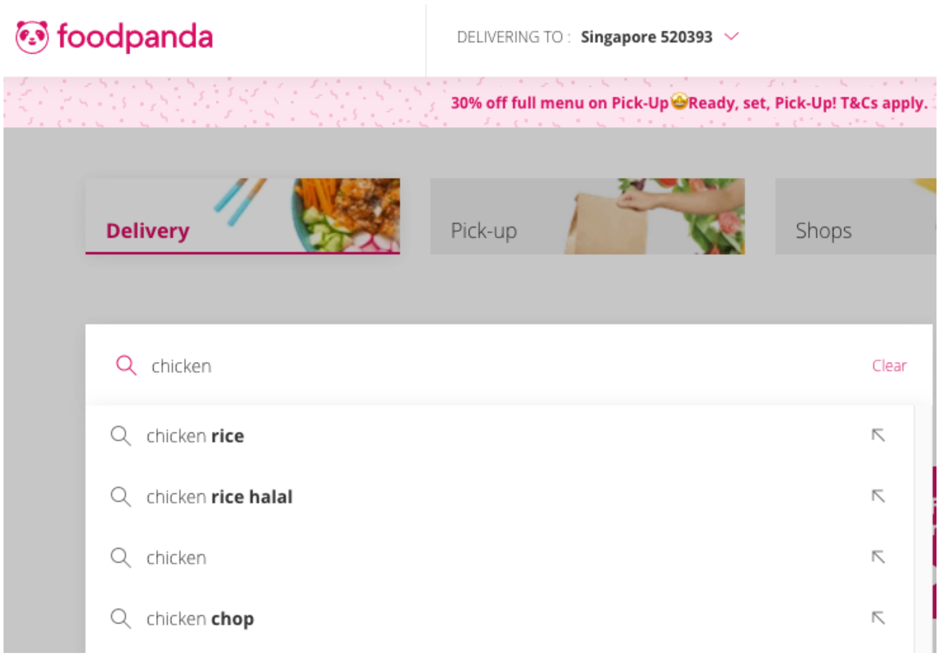
For example, searching for the word *cabbage* that **does not** exist in the trie, the best case would be O(m-n) because the first alphabet **c** would have thrown off a nil as a search result. Therefore, the other parts of the queried string should not be accounted for, for time complexity calculation.

Search is Good. Auto Complete is Better


google autocomplete a — Google Search
Google Search
google autocomplete a
google autocomplete api
google autocomplete address
google autocomplete april fools
google autocomplete address example
google autocomplete api ios
Search for "google autocomplete a" in History
Go to Site "google autocomplete a"

*https://miro.medium.com/max/500/1*x3uqttlp-NNAlHxqwRbXyg.gif*

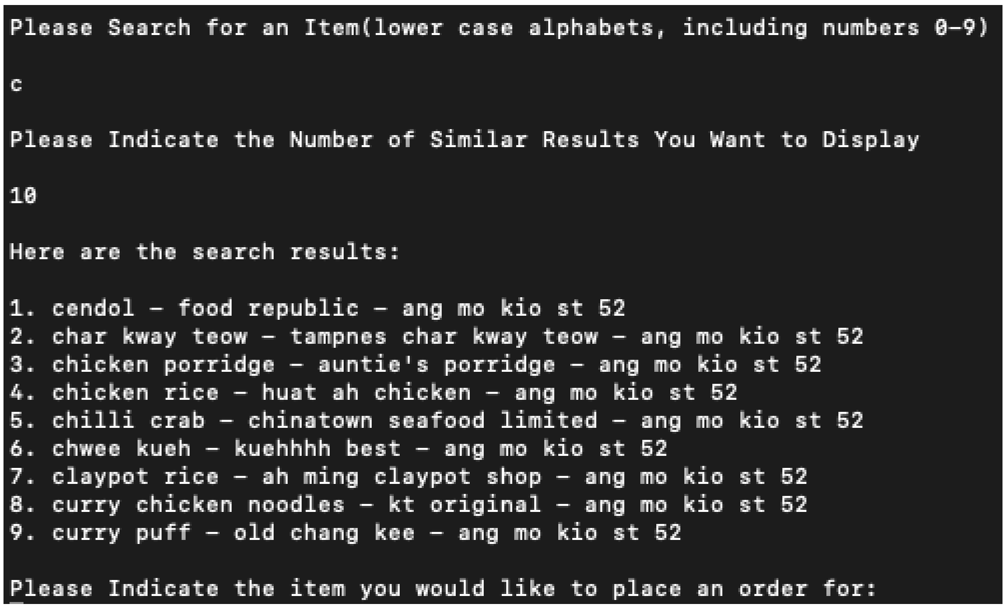
Prefix tries are also highly effective in generating suggestive search results and these type of functions can be commonly observed at sites like google.com


foodpanda
DELIVERING TO : Singapore 520393
30% off full menu on Pick-Up Ready, set, Pick-Up! T&Cs apply.
Delivery
Pick-up
Shops
chicken
Clear
chicken rice
chicken rice halal
chicken
chicken chop

Another example of auto suggestion and completion can be observed on Food Panda's website. Searching for the word *chicken* reveals keywords like *chicken rice, chicken rice halal, chicken chop* etc

In today's modern search solutions, people don't tend to type out entire keywords anymore especially when it comes to food delivery or general search on google, for example. The faster an auto complete functionality kicks in, and can **accurately** suggest matching results, the quicker a selection can be made.
Time, is really money. In this case, money for the company.

In the application itself, you can see a similar suggestion working through the search function.



```
Please Search for an Item(lower case alphabets, including numbers 0-9)

c

Please Indicate the Number of Similar Results You Want to Display

10

Here are the search results:

1. cendol - food republic - ang mo kio st 52
2. char kway teow - tampnes char kway teow - ang mo kio st 52
3. chicken porridge - auntie's porridge - ang mo kio st 52
4. chicken rice - huat ah chicken - ang mo kio st 52
5. chilli crab - chinatown seafood limited - ang mo kio st 52
6. chwee kueh - kuehhhh best - ang mo kio st 52
7. claypot rice - ah ming claypot shop - ang mo kio st 52
8. curry chicken noodles - kt original - ang mo kio st 52
9. curry puff - old chang kee - ang mo kio st 52

Please Indicate the item you would like to place an order for:
```

In the screenshot above, you can see that searching for the character *c* and suggesting a maximum of 10 search results, the application is able to accurately display the keyword(s) starting with *c*.

And with regards to design considerations, a customer service officer may not be able to remember the entire food dish name. But knowing that he or she can just type one or two characters to reveal other suggestions can be a very helpful and time-saving feature.

Binary search tree

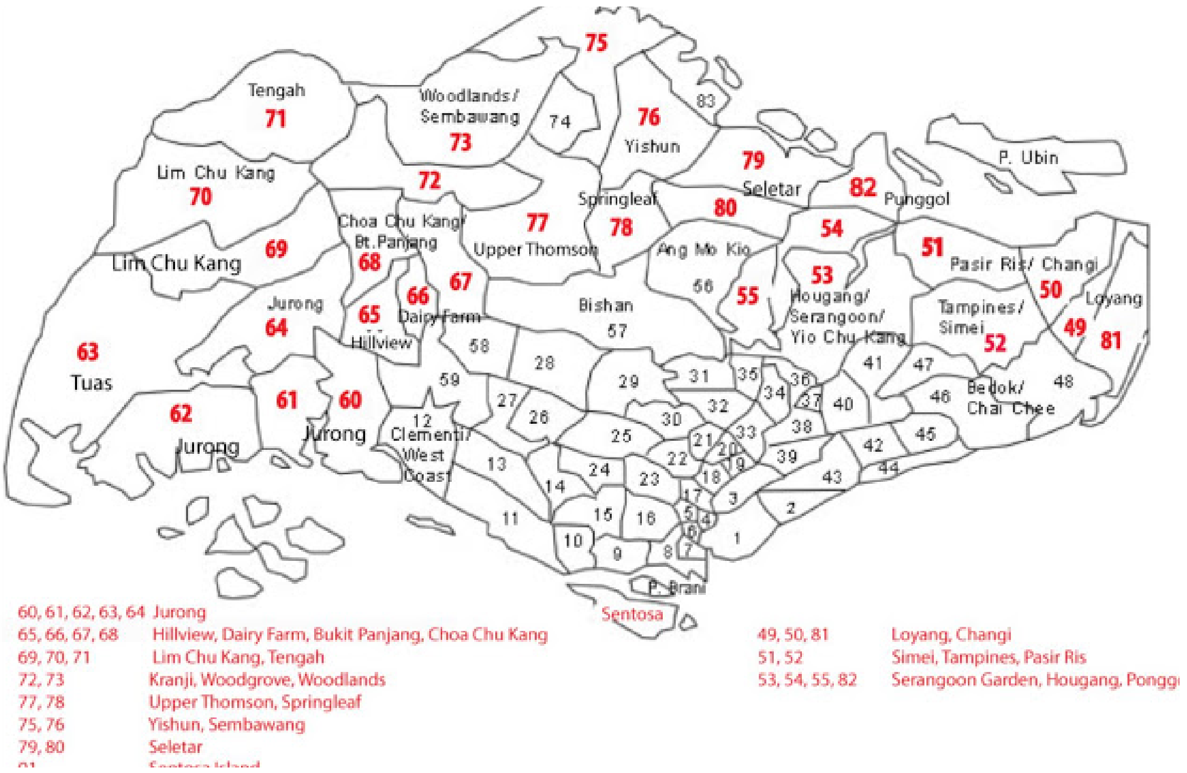

Tengah
Woodlands/ Sembawang
Yishun
Lim Chu Kang
Springleaf
Seletar
P. Ubin
Punggol
Choa Chu Kang/ Bt.Panjang
Upper Thomson
Ang Mo Kio
Pasir Ris/ Changi
Jurong
Dairy Farm
Hillview
Bishan
Hougang/ Serangoon/ Yio Chu Kang
Tampines/ Simei
Loyang
Tuas
Bedok/ Chai Chee
Clementi/ West Coast
P. Brani
Sentosa
60, 61, 62, 63, 64 Jurong
65, 66, 67, 68 Hillview, Dairy Farm, Bukit Panjang, Choa Chu Kang
69, 70, 71 Lim Chu Kang, Tengah
72, 73 Kranji, Woodgrove, Woodlands
77, 78 Upper Thomson, Springleaf
75, 76 Yishun, Sembawang
79, 80 Seletar
49, 50, 81 Loyang, Changi
51, 52 Simei, Tampines, Pasir Ris
53, 54, 55, 82 Serangoon Garden, Hougang, Pongg

Image Courtesy of : http://danielchoy.blogspot.com/2017/03/singapore-postal-codes-and-district.html

This is how Singapore's postal code system looks like; in a nutshell. Postal code numbers have prefixes (the first two digits) ranging from 1 through 80 are employed to ensure delivery companies are able to deliver things to their recipients accurately regardless of where they are in the country. In this diagram, we can also observe that a postal code can be unique among others but will also share a few similarities with other postal codes in the country.

A postal code data has the following attributes:

1. It is **6 digits.** No more; no less. Having a fixed number of digits; also means errors can be easily taken care of, error handling measures implemented. An example is to set the program to only accept 6 digits; not 5 not 7.

2. They adhere to the same prefix; over and over again. I.e. 40X XXX, 52X XXX etc. This means the data will share a form of similarity; making finding, sorting easily when the right data structures are employed.

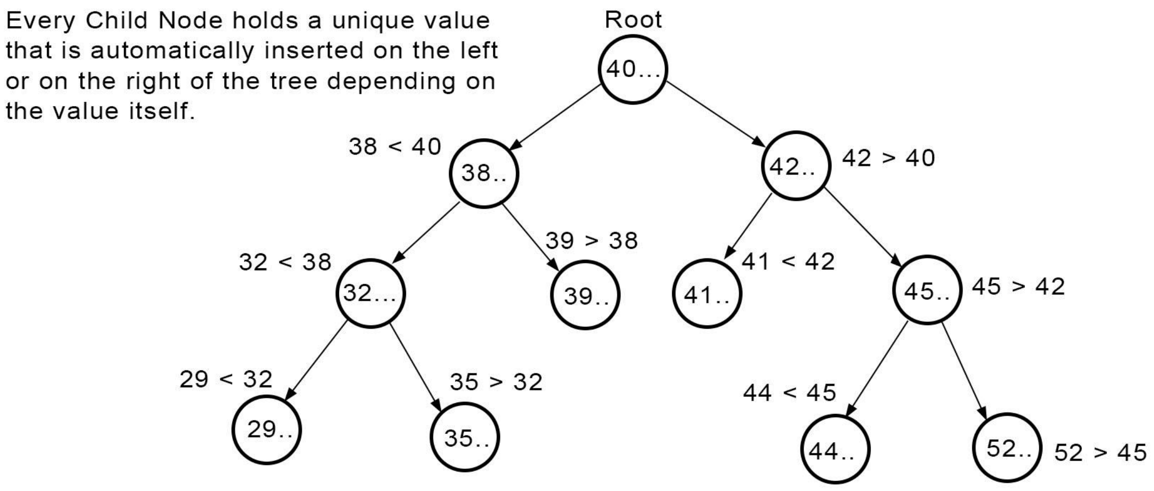
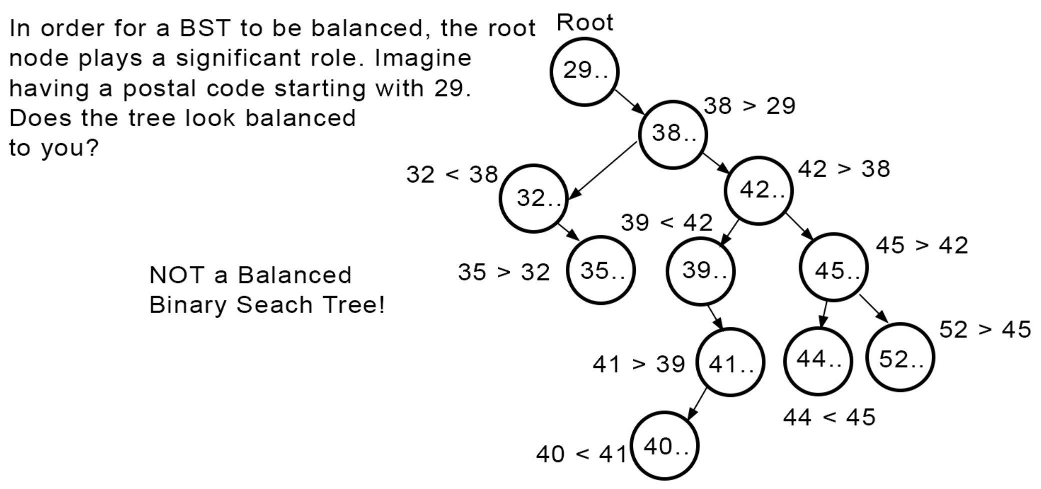

Every Child Node holds a unique value that is automatically inserted on the left or on the right of the tree depending on the value itself.
Root
40...
38 < 40
38..
42..
42 > 40
39 > 38
32 < 38
32...
39..
41..
41 < 42
45..
45 > 42
29 < 32
35 > 32
44 < 45
29..
35..
44..
52..
52 > 45

Diagram for a BST Tree

To store and to achieve a quick look up on postal codes, the Binary Search Tree data structure, or BST for short is employed. A BST is a tree-based ordered data structure that satisfies binary search property. Such a search property governs that for values that are less than its root node; it will be inserted on the left while values that are greater; will be inserted on the right.

Visual illustrations on where they are inserted is indicated in the diagram above. For example, a postal code starting from 38X XXX will be placed on the left side of the root note and a postal code of 52X XXX will be placed on the right side of 45X XXX.

A quick glance over the tree, you will realise that the value in the root node, plays a significant part in controlling how the tree will be skewed towards. Let's look at an example of how a tree will be skewed if the root node's value; the postal code starts with a small value such as 29.


In order for a BST to be balanced, the root node plays a significant role. Imagine having a postal code starting with 29. Does the tree look balanced to you?

NOT a Balanced Binary Seach Tree!

Diagram for a BST Tree

Can you observe that the left node (left side of 29) has been completely eliminated? If the data nodes are added in the same way as the previous tree, this is how the tree will end up looking, if the root value of the tree is 29.

The importance of a balanced BST can be observed when we look at the time complexity analysis of a BST.

One of the visual differences you can see from a balanced tree vs a non-balanced tree are the number of *levels* or *height* involved. In a balanced tree, we can observe 4 levels, including the root level. However, you can observe 6 levels instead of an unbalanced tree.

Time Complexity for Binary Search Tree

The time complexity for a balanced tree, is O(logn) where the height or level of the binary search tree is log(n)

This is also why a balanced tree is of significant importance. The more balanced a tree is; the fewer the nodes and the quicker operations on the BST can be executed.

Therefore; to ensure that a tree is balanced, the first postal code data to be added into the system must be within the 40X XXX range.

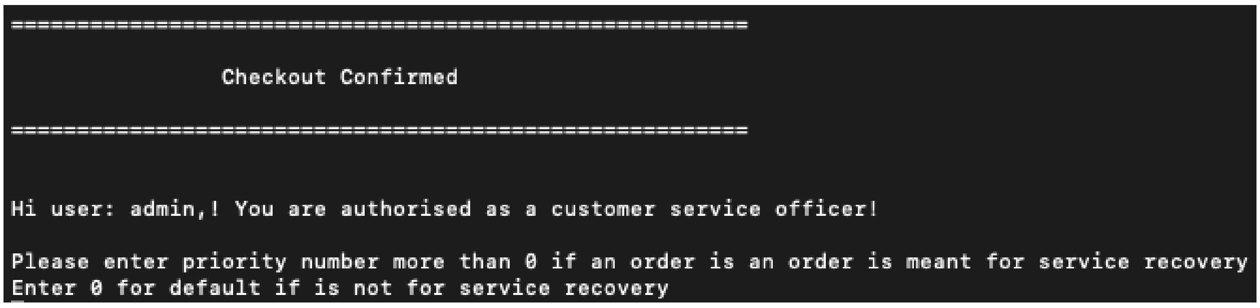
Priority Queues

The core business requirement of the KV business requires orders placed by customer service officers as well customers to be served in a first in first out approach.

In other words; if an order, named **order A** is made now, and another order, **order B** is made a second later, order A should be processed first.

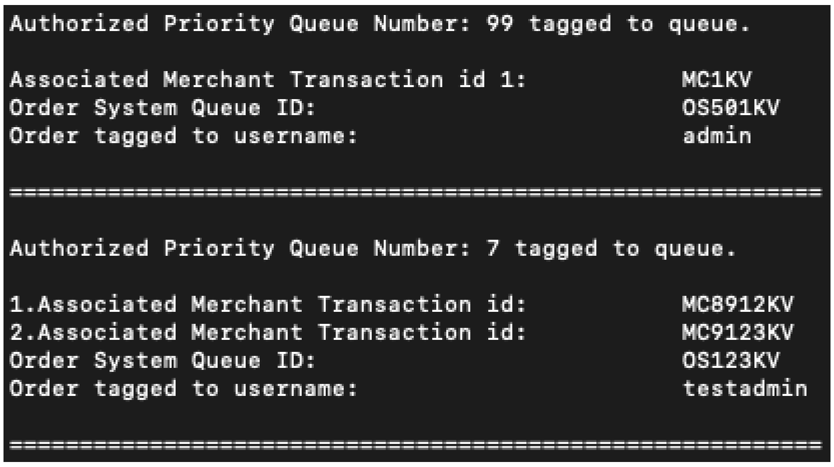
While that is correct, the aim of the KV app is to improve customer satisfaction and loyalty. In the event that there is a service recovery required, an admin that is logged in, is able to make changes to the queue by making an order of a higher priority than the others.

This can be achieved easily by settings numbers that are more than 0, when a user of admin status is logged in. And yes, setting to negative numbers would push the queue all the way to the back.



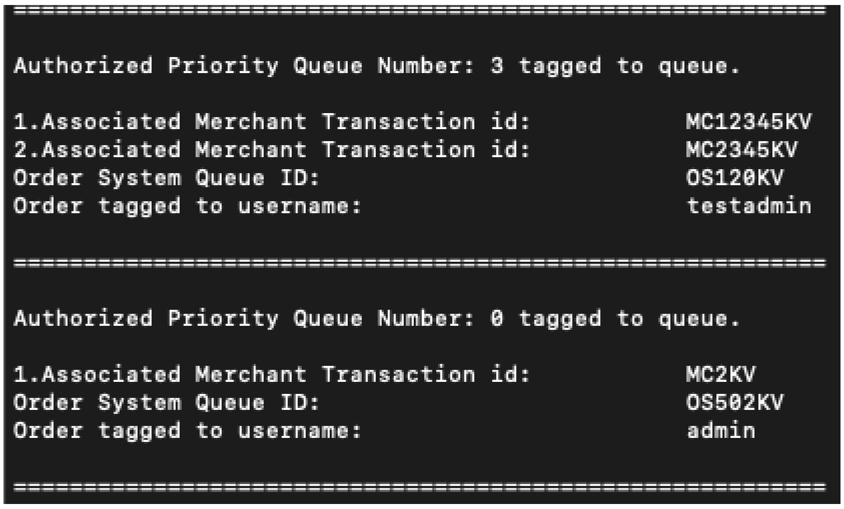
```
=======================================================

                Checkout Confirmed

=======================================================


Hi user: admin,! You are authorised as a customer service officer!

Please enter priority number more than 0 if an order is an order is meant for service recovery
Enter 0 for default if is not for service recovery
```

If a number such as 99 is added to portray an order of significant importance such as an order meant for service recovery, you can visually see that the queue is amended, placing this order on the first of the list. (right before an order of priority 7) This ensures that delivery partners will see the order first; thereby starting the service recovery process.



```
Authorized Priority Queue Number: 99 tagged to queue.

Associated Merchant Transaction id 1:           MC1KV
Order System Queue ID:                          OS501KV
Order tagged to username:                       admin

==========================================================

Authorized Priority Queue Number: 7 tagged to queue.

1.Associated Merchant Transaction id:           MC8912KV
2.Associated Merchant Transaction id:           MC9123KV
Order System Queue ID:                          OS123KV
Order tagged to username:                       testadmin

==========================================================
```

However, if another priority value such as 0 is set, which is the default value, the order gets added to the last of the queue, automatically.



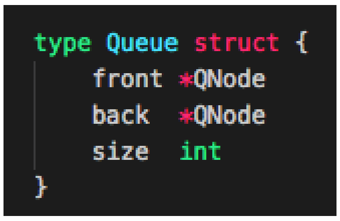
```
==========================================================

Authorized Priority Queue Number: 3 tagged to queue.

1.Associated Merchant Transaction id:          MC12345KV
2.Associated Merchant Transaction id:          MC2345KV
Order System Queue ID:                         OS120KV
Order tagged to username:                      testadmin

==========================================================

Authorized Priority Queue Number: 0 tagged to queue.

1.Associated Merchant Transaction id:          MC2KV
Order System Queue ID:                         OS502KV
Order tagged to username:                      admin

==========================================================
```

Queues can be also removed automatically when the dispatch menu is selected. You can imagine a dispatch supervisor with the correct privileges to be able to add an order or dispatch one if he/she thinks that there is a delivery partner/rider that is able to commit and fulfill the order.

Time Complexity for Queues

The Overall time complexity for general queue operations can be referred to as O(n) where n is the number of elements in the queue.

If there are 100 queues in line, adding a new queue at the end of the line is O(1) because there are no changes to the queue since a general queue struct has *back* as a field value as seen below and we can immediately add it to the end of the queue.

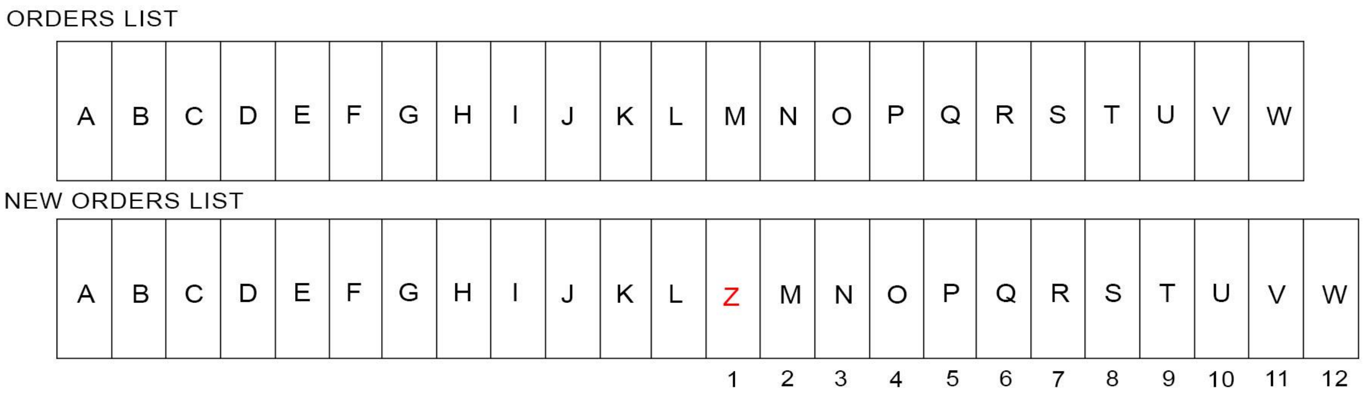


```go
type Queue struct {
    front *QNode
    back  *QNode
    size  int
}
```

The first in line, will still be the next to be dispatched; similar to the second in line which will be the next in line.

But as soon as the first in line gets dispatched, the time required for the elements in the queue will be shifted. *The second in line becomes first, third in line becomes second etc*. This is when the time complexity becomes O(n) where n is the number of elements in the queue.

However, priority queues like the one that has been implemented; the time complexity becomes O(n) + O(m) where *n* is the number of items it has to check for priority index, and *m* is the difference in the number of elements that have to be shifted.


ORDERS LIST
A B C D E F G H I J K L M N O P Q R S T U V W
NEW ORDERS LIST
A B C D E F G H I J K L Z M N O P Q R S T U V W
1 2 3 4 5 6 7 8 9 10 11 12

O(n) is illustrated in the diagram above. While checking for priority index, the program has to check from the first, in the list, to the last to see which *slot* it should take. In the diagram above, *n* refers to 13 as it has to check slot number 13, 'M' and evaluate the priority before deciding that M is of a "lower priority value".

O(m) is illustrated in the diagram above. In the event that a new order, order Z is of priority values in between orders L and M, then the older orders, orders from M to W have to be pushed m times. In this case, 12 times.

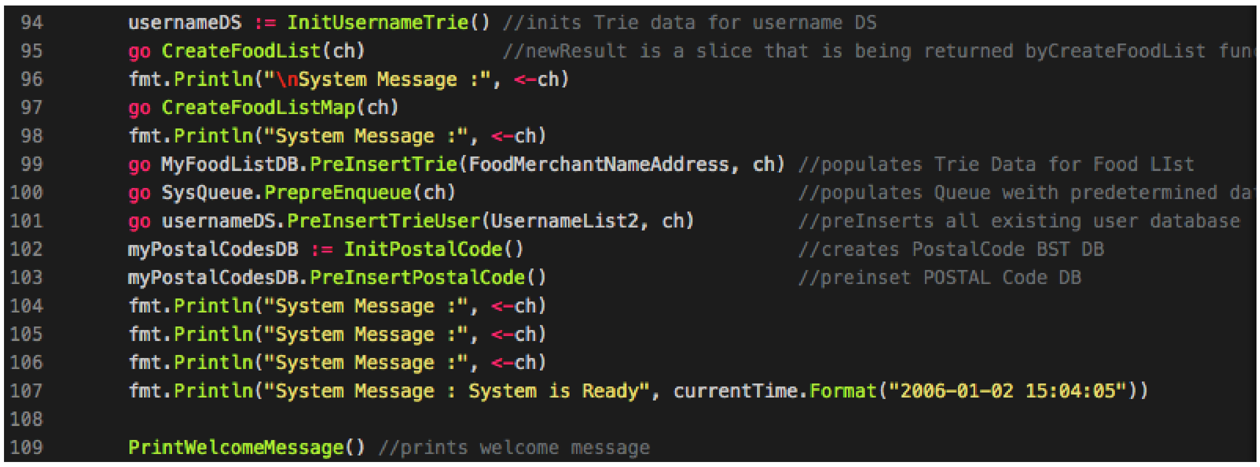
Therefore, the total time required for inserting a data into a priority queue is

O(n) + O(m) time

Channels for Data Management and Performance Enhancement

Implementation of different GO Routines, at the right place of the code allows for data to be initialized and populated accurately.

Let's look at how it is being implemented in the code:



```
 94        usernameDS := InitUsernameTrie() //inits Trie data for username DS
 95        go CreateFoodList(ch)            //newResult is a slice that is being returned byCreateFoodList fun
 96        fmt.Println("\nSystem Message :", <-ch)
 97        go CreateFoodListMap(ch)
 98        fmt.Println("System Message :", <-ch)
 99        go MyFoodListDB.PreInsertTrie(FoodMerchantNameAddress, ch) //populates Trie Data for Food LIst
100        go SysQueue.PrepreEnqueue(ch)                              //populates Queue weith predetermined da
101        go usernameDS.PreInsertTrieUser(UsernameList2, ch)         //preInserts all existing user database
102        myPostalCodesDB := InitPostalCode()                        //creates PostalCode BST DB
103        myPostalCodesDB.PreInsertPostalCode()                      //preinset POSTAL Code DB
104        fmt.Println("System Message :", <-ch)
105        fmt.Println("System Message :", <-ch)
106        fmt.Println("System Message :", <-ch)
107        fmt.Println("System Message : System is Ready", currentTime.Format("2006-01-02 15:04:05"))
108
109        PrintWelcomeMessage() //prints welcome message
```

At line 95, the first GO routine has been set to run.

```
   go CreateFoodList(ch)
```

This line of code, *CreateFoodList* is solely responsible for populating a slice data, from other forms of data that can be found in the rawData.go file and at the same time, passing a channel argument into the function.

As the slice data is of paramount importance, we need the data to be initialized, imported and created correctly. To ensure that the GO Routine **runs** the entire operation, we use a blocking code at line 96

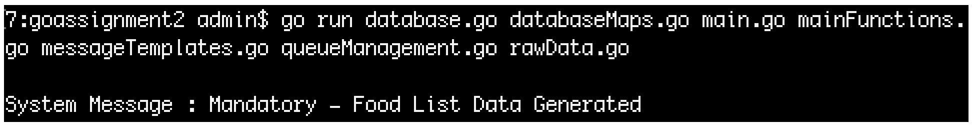
```
   fmt.Println("\nSystem Message :", <-ch)
```

<- *ch* in this instance is called a blocking of code because the channel is expecting to receive a message from the function that was called in line 95.

Once the Go routine has completed the function, at the end of the function *CreateFoodList(ch)* found in mainFunctions.go sends the following string data into the channel :

```
   ch <- "Mandatory - Food List Data Generated"
```

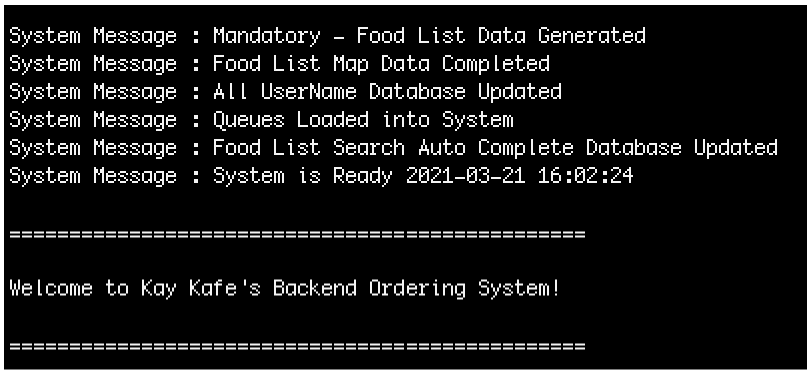
This is why line 96 code is always executed in the terminal before the other codes.



```
7:goassignment2 admin$ go run database.go databaseMaps.go main.go mainFunctions.
go messageTemplates.go queueManagement.go rawData.go

System Message : Mandatory - Food List Data Generated
```

And the GO Routines found in lines 99, 100, 101 are all set off together, to ensure that the GO Scheduler is able to manage these 3 processes concurrently.

The blocking codes in line 104, 105 and 106 ensure that these other 3 GO routines are run and completed, forcing a return message. This is how the application ensures that the data are all populated before allowing the system to be ready for usage.



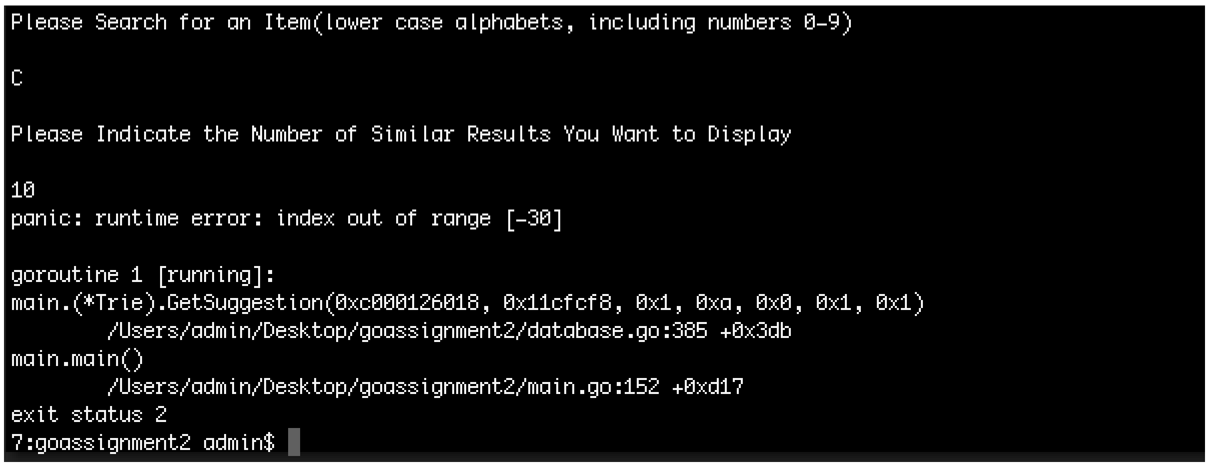
```
System Message : Mandatory - Food List Data Generated
System Message : Food List Map Data Completed
System Message : All UserName Database Updated
System Message : Queues Loaded into System
System Message : Food List Search Auto Complete Database Updated
System Message : System is Ready 2021-03-21 16:02:24

================================================

Welcome to Kay Kafe's Backend Ordering System!

================================================
```

As seen in the screengrab of an actual application usage, system messages received by the channels ensure the data is correctly implemented so features queried and displayed can be accurate at all times

Error, Panic and Recovery Handling

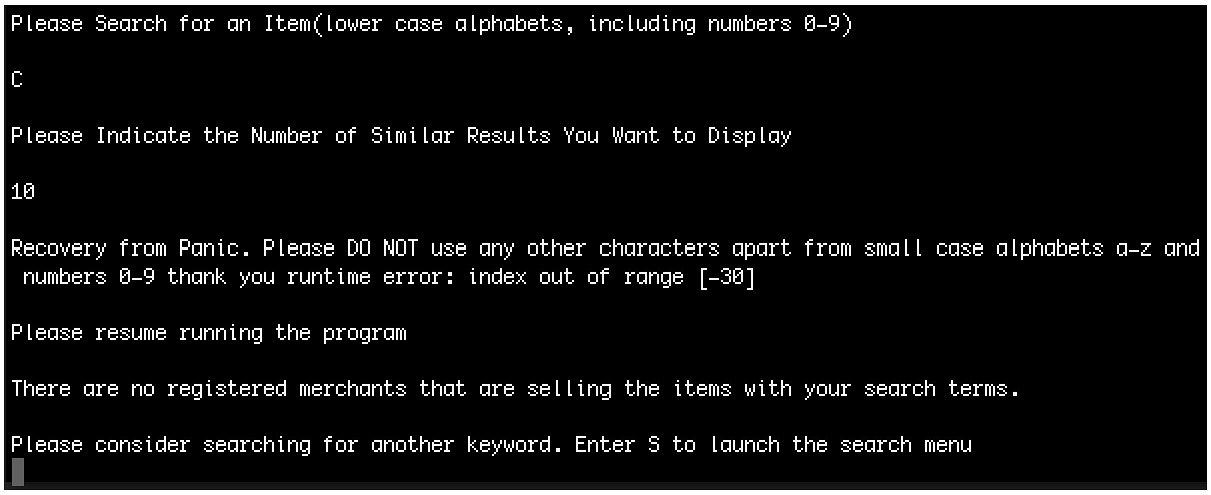
As required by the Error and Panic considerations, error handling in the program was designed by anticipating the possible errors that could happen in code. As briefly described before, the most vulnerable part of the code has to do with retrieving information on data structures that have fixed capacities.

Highlighted in the section of the prefix Trie, Panics can happen when characters that **do not** belong to the 39 characters get searched for. Here are two screenshots that showcases a screengrab without panic considerations and one with panic considerations.



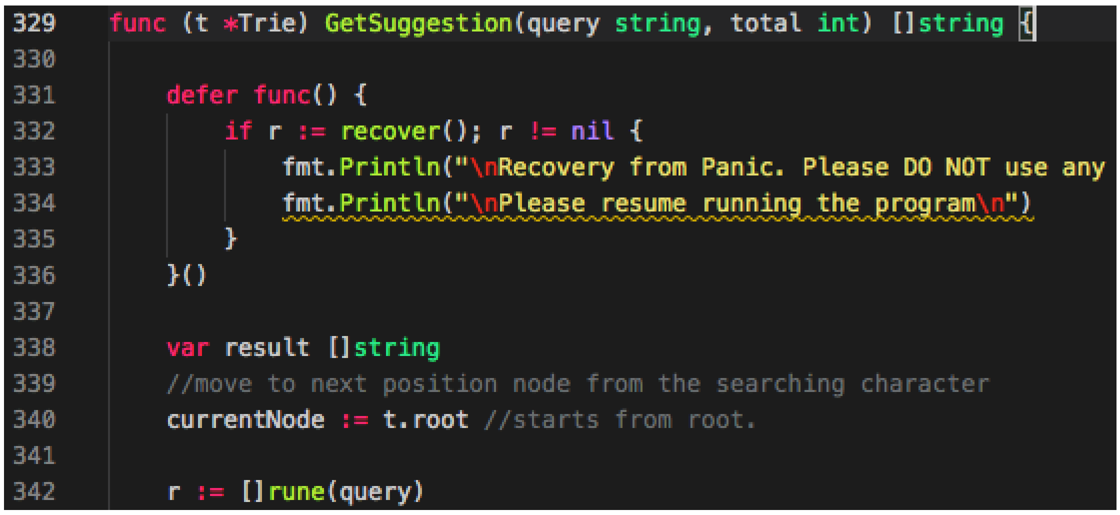
```
Please Search for an Item(lower case alphabets, including numbers 0-9)

C

Please Indicate the Number of Similar Results You Want to Display

10
panic: runtime error: index out of range [-30]

goroutine 1 [running]:
main.(*Trie).GetSuggestion(0xc000126018, 0x11cfcf8, 0x1, 0xa, 0x0, 0x1, 0x1)
        /Users/admin/Desktop/goassignment2/database.go:385 +0x3db
main.main()
        /Users/admin/Desktop/goassignment2/main.go:152 +0xd17
exit status 2
7:goassignment2 admin$
```

In the picture above, without proper panic considerations, the program **is forced to exit.**. In such the exited state, a user cannot continue the usage of the app. Reloading of the app would require time and effort; especially when importing big datas can be very time and memory consuming.



```
Please Search for an Item(lower case alphabets, including numbers 0-9)

C

Please Indicate the Number of Similar Results You Want to Display

10

Recovery from Panic. Please DO NOT use any other characters apart from small case alphabets a-z and
 numbers 0-9 thank you runtime error: index out of range [-30]

Please resume running the program

There are no registered merchants that are selling the items with your search terms.

Please consider searching for another keyword. Enter S to launch the search menu
```

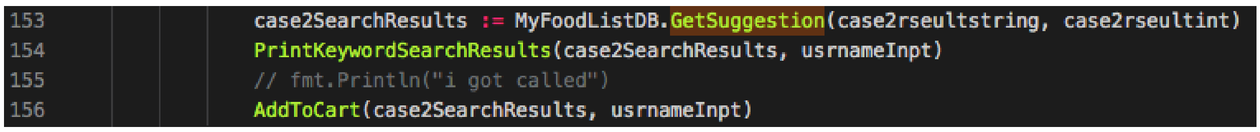
In the above diagram, panic and recovery codes were implemented right in the function that would cause the panic. Below, we'll see how the code is implemented.



```
329  func (t *Trie) GetSuggestion(query string, total int) []string {
330
331      defer func() {
332          if r := recover(); r != nil {
333              fmt.Println("\nRecovery from Panic. Please DO NOT use any
334              fmt.Println("\nPlease resume running the program\n")
335          }
336      }()
337
338      var result []string
339      //move to next position node from the searching character
340      currentNode := t.root //starts from root.
341
342      r := []rune(query)
```

In lines 331 of the function *GetSuggestion()* , we deferred a recovery function. In a nutshell, a deferred function can fire even during a panic, making this code a useful one for serving and recovering from a crash. And since this is the core function that searches for elements that may potentially go beyond the character limit, we know that this function is also the one that will crash.

With the firing of a deferred recovery code, the power is returned to the block of code that is responsible for calling *GetSuggestion().*

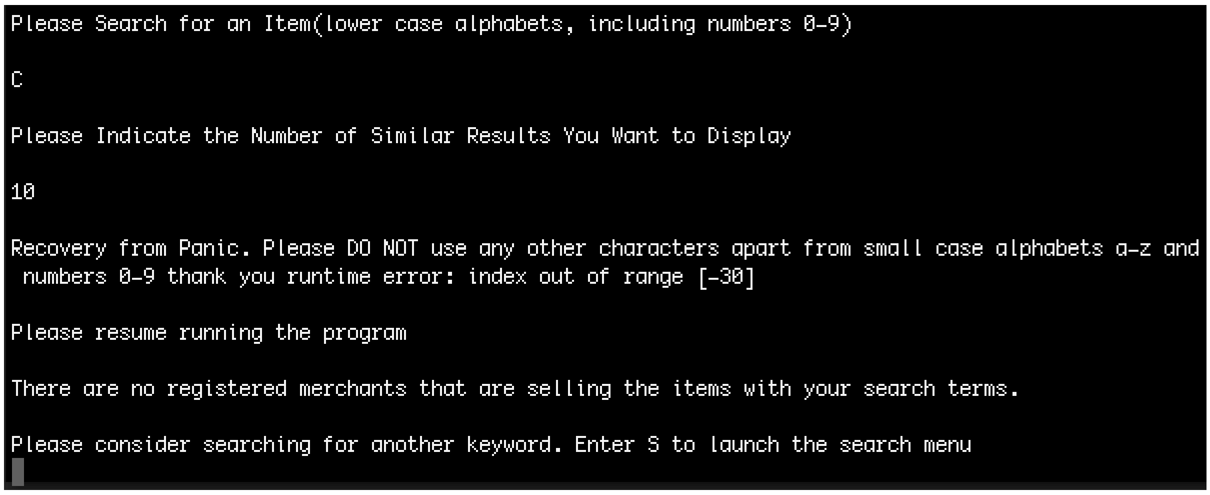


```
153              case2SearchResults := MyFoodListDB.GetSuggestion(case2rseultstring, case2rseultint)
154              PrintKeywordSearchResults(case2SearchResults, usrnameInpt)
155              // fmt.Println("i got called")
156              AddToCart(case2SearchResults, usrnameInpt)
```

Tracking back to the main routine, we see that it is code line 153 that calls *GetSuggestion().*

Since power returns to the main function thanks to the recovery code, the main function **continues running**. In this case, the next function in line 154, *PrintKeywordSearchResults*() gets executed.

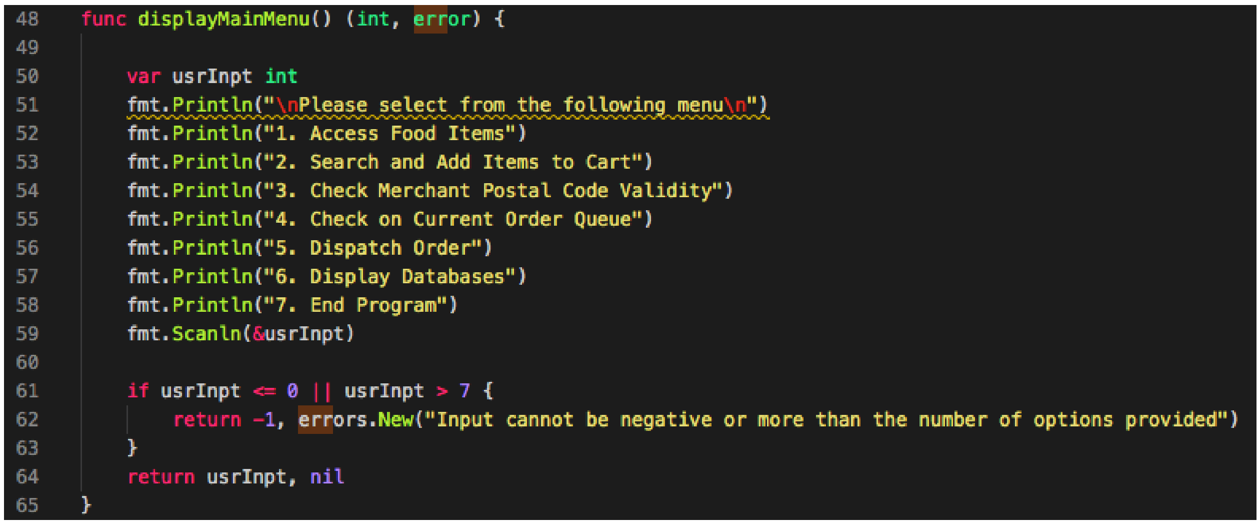
However, since the previous function panicked, there was no value assigned to *case2SearchResults* and since *PrintKeywordSearchResults* needs that argument to be of a non-nil data, the function intelligently tells you that the data is invalid, prompting you for another input.



```
Please Search for an Item(lower case alphabets, including numbers 0-9)

c

Please Indicate the Number of Similar Results You Want to Display

10

Recovery from Panic. Please DO NOT use any other characters apart from small case alphabets a-z and
 numbers 0-9 thank you runtime error: index out of range [-30]

Please resume running the program

There are no registered merchants that are selling the items with your search terms.

Please consider searching for another keyword. Enter S to launch the search menu
```

On top of that, the recovery function can only help prompt users in a helpful way regarding the error that occurred. You can see a prompt above that reads *Recovery from Panic. Please DO NOT use any other characters … " etc.*

Apart from Panic and Recovery functions, due care on error handling can also be found in the code. Let's visit an example below:



```go
func displayMainMenu() (int, error) {

	var usrInpt int
	fmt.Println("\nPlease select from the following menu\n")
	fmt.Println("1. Access Food Items")
	fmt.Println("2. Search and Add Items to Cart")
	fmt.Println("3. Check Merchant Postal Code Validity")
	fmt.Println("4. Check on Current Order Queue")
	fmt.Println("5. Dispatch Order")
	fmt.Println("6. Display Databases")
	fmt.Println("7. End Program")
	fmt.Scanln(&usrInpt)

	if usrInpt <= 0 || usrInpt > 7 {
		return -1, errors.New("Input cannot be negative or more than the number of options provided")
	}
	return usrInpt, nil
}
```

In the above diagram, you can find the function *disalayMainMenu()* which takes in no arguments but returns two types of variables.

The function checks if a user's input is valid. For example, a value of less than or equals to zero **or** a val that is above 7 (maximum choice of 7 inthe main menu) will return an error. This ensures that users are always made to choose the correct values.

File Structure:

All the .go files are of a single package - package main.

You can find files names and a brief description of what the file entails below:

main.go - You will find the main function here along with various function calls here.

rawData.go - This file contains all the raw database of the merchant information/ address/ postal code/ dish name that is stored in slice of custom struct

databaseMaps.go - maps are initialized and declared here. Declared as global variables to ensure they are accessible everywhere.

mainFunctions.go - This is the go file that contains functions that are called from main func. Many basic functions are stored in this file. Basic functions such as menu displays can be found here.

Database.go - declaration of trie, functions pertaining to trie and binary search trees are stored here. Functions like trie add/search/ suggestion etc can be found here.

queueManagement.go - Declaration of the priority queue, and related functions can be found here

messageTemplates.go - Function calls to generate repetitive messages like welcome messages.

Features Pending for Future Implementation:

1. Implementation of CSV imports or JSON imports to ensure standardization of data across programs.

2. Postal Code integration for customized search results. Algorithms should be written to serve search results based on google api / distances or postal code prefixes.

3. Better integration of data structure for storing username values to calculate spending. With better struct, members that hold high spending should be identified and given special privileges and data handed over to marketing teams

4. Date time of queue number should be able to be retrieved and logged in struct as well

5. Postal Codes should accurately reflect the number of merchants registered at that address. To do that, we'll need a bigger database and a more accurate one