Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2021.

PROYECTO 2 IPC2

202110568 – Damián Ignacio Peña Afre

Resumen

La empresa “Soluciones Guatemaltecas, S.A.” está construyendo un sistema de atención a clientes diseñado para cualquier organización que necesite brindar servicios presenciales a sus clientes. Dentro de este marco es importante garantizar la funcionalidad del sistema, por lo que se desea implementar un primer prototipo que daría una vista general de lo que el sistema final puede realizar. La empresa gestionará entonces múltiples empresas, cada una de estas con sus respectivos de atención ubicados en distintas direcciones y a su vez cada uno de estos puntos de atención con escritorios de atención. El sistema provee una forma de modificar y cargar datos de estas empresas, así como de seleccionar una en particular para su simulación. La simulación puede generar reportes del estado del punto de atención desplegando estadísticas útiles para la toma de decisiones. Es posible realizar cambios en tiempo de ejecución para personalizar aún mas el comportamiento de la simulación, estos pueden ser activar/desactivar escritorios o agregar nuevos clientes a la cola de espera.

Palabras clave

Simulación, estructura de datos, tipos de datos abstractos

Abstract

The company “Soluciones Guatemaltecas, S.A.” is building a customer service system designed for any organization that needs to provide face-to-face services to its customers. Within this framework, it is important to guarantee the functionality of the system, so it is desired to implement a first prototype that would give an overview of what the final system can do. The company will manage multiple companies, each of these with their respective attention points located at different addresses and in each of these service points would be cointained service desks. The system provides a way to modify and load data from these companies, as well as to select one in particular for simulation. The simulation can generate reports on the status of the attention point, displaying useful statistics for perfomance issues. It is possible to make changes at runtime to further customize the behavior of the simulation, these can be enabling/disabling desktops or adding new clients to the queue.

Keywords

Simulation, data structures, abstract data types

Introducción

El presente proyecto tiene la finalidad de implementar un programa de simulación, para ello se pretende hacer uso de estructuras que permitan manejar la memoria dinámica, como lo son las listas enlazadas. Se analizará asimismo el porque de utilizar ciertas estructuras.

El uso de programación orientada a objetos es inherente a la construcción de la aplicación por lo que también se examinará por qué el uso de ciertas clases y formas de estructurar el proyecto.

Por la naturaleza del proyecto, al ser solo un prototipo se optó por realizar una aplicación de consola que cuentan con una interacción sencilla y rápida.

Estructuras de datos

Llega un punto en el que las estructuras básicas o datos primitivos como *boolean*, *int* o *float* ya no nos son suficientes para realizar lo que requiere una aplicación y mucho menos cuando se requiere agrupar de alguna manera tipos de datos compuestos. Es ahí donde entran las estructuras de datos.

"Una estructura de datos es una colección de datos que pueden ser caracterizados por su organización y las operaciones que se definen en ella." (Ferrer, 2018)

Aprovechándose entonces de algun lenguaje orientado objetos como lo es *Python* podemos construir estas estructuras y manejar la memoria de la forma más conveniente para nuestra solución.

Dentro de las estructuras de datos de encuentran aquellas que son dinámicas y por tanto no tienen limitaciones en el tamaño de memoria que ocupan. Mediante *punteros* es posible construir estos tipos de estructuras.

Otra subclasificación de las estructuras de datos dinámicas es aquellas que se denominan como lineales y particularmente se analizará la listas enlazadas.

"Una lista es una colección de elementos organizados en secuencia que puede crecer y decrecer libremente y a cuyos elementos individuales se puede acceder, insertar y eliminar en cualquier posición." (Ferrer, 2018).

Entre algunas de las operaciones que se pueden realizar con las listas se encuentra: insertar, eliminar o localizar un elemento; determinar el tamaño de la lista; recorrer la lista; entre muchos otros.

Un *nodo* es un elemento de la lista que almacena la referencia algun otro u otros nodos y tiene un espacio reservado para almacenar algun dato.

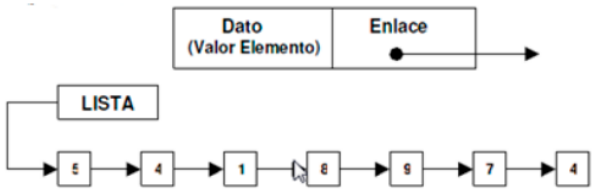

Figura 1. Lista enlazada

Fuente: Ferrer, 2018, pág.: 63.

Teniendo ya cubierta la parte para almacenar y agrupar datos con la lista simplemente enlazada otra estructura apropiada es la *pila*.

Una *pila* es un tipo especial de lista lineal en la que la inserción y eliminación siguen cierta lógica que es conocida como *LIFO* (last-in, first-out). Esto quiere decir que al intentar insertar un nuevo elemento estos se apilan de manera tal que al intentar remover alguno de estos elementos solo es posible eliminar el ultimo que haya sido insertado.

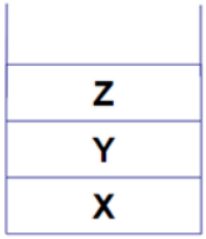

Figura 2. Pila

Fuente: Ferrer, 2018, pág.: 67.

Las operaciones que se suelen utilizar para la implementación de pilas son: *push*, para la el apilamiento de nuevos elementos; *pop*, para la eliminación del ultimo elemento insertado; *isEmpty*, para verificar si la pila está vacía; entre otros.

Conociendo de manera teórica la pila es posible implementar una utilizando una lista simple y restringiendo algunos métodos para que siga la lógica *LIFO*.

De manera similar surge la necesidad de una forma de almacenar los clientes, estos como comúnmente se suelen ver agrupados en una *cola*.

Una *cola* es otra estructura de datos que se rige bajo la lógica *FIFO* (first-in, first-out). Al ser también un tipo de estructura lineal es posible realizar una implementación a partir de una lista simple.

A diferencia de la pila la cola puede implementar los métodos *eliminarAlInicio* e *insertarAlFinal* de la lista simple, mientras que la pila utilizaría los métodos *eliminarAlFinal* e *insertarAlFinal*. Dentro de las implentaciones más comunes para colas se suele llamar a los métodos *enqueue* para agregar un nuevo elemento y *dequeue* para eliminar o “atender” a un elemento de la cola.

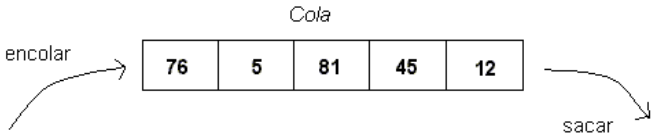

Figura 3. Pila

Fuente: Ferrer, 2018, pág.: 69.

Lógica de la aplicación

El punto de entrada para la información dentro de la aplicación es la lectura del archivo xml, para ello puede ser utilizada una de las librerías preinstaladas en Python que es ampliamente utilizada para manejar el DOM. El DOM es un representación del documento por medio de objetos de manera que es mucho más fácil manejar sus elementos. Y es por ello por lo que dicha librería tiene el nombre de minidom.

Posteriormente a realizar la lectura del archivo se extraen los datos requeridos para un nuevo registro de la empresa y sus respectivos puntos de atención y

escritorios. También existe la posibilidad cargar un archivo adicional para configurar los clientes y escritorios activos.

La lógica de activación es la siguiente: “Estos escritorios de servicio son habilitados siguiendo una lógica de primero en ser activado para atención a clientes último en ser desactivado”. Por lo que sigue la lógica *LIFO* y por lo tanto es necesario implementar una pila.

Al empezar la simulación se organizarán a los clientes de la cola en los escritorios activos disponibles para su atención, al entrar al escritorio el mismo cambia de estado para atender únicamente a ese cliente hasta que termina todas sus transacciones.

Luego de que se realizan todas las transacciones el cliente es retirado totalmente del sistema y el escritorio regresa a su estado para atender a un nuevo cliente.

Como se mencionó al inicio en cada momento de la simulación es posible generar un informe en PDF, para ello se utiliza la herramienta *Graphviz* que en conjunto con su propio lenguaje es posible representar todos los datos necesarios del punto de atención.

La implementación de *Graphviz* se realiza escribiendo un archivo *.dot* por lo que solo es necesario tener la capacidad de crear o sobrescribir archivos para luego volver a compilar utilizando el comando de la herramienta.

Dentro de los anexos se encuentra de forma detallada el diagrama de clases implementado para la solución.

Se puede destacar que para cada uno de los elementos principales es creada una clase y estos objetos dependen unos de otros para formar las relaciones de asociación y agregación.

También se puede mencionar la utilización de *genéricos* dentro de *Python*, que no son mas que una forma de especificar algun tipo que se deba pasar por parámetro cuando se instancia algun objeto. Resulta sumamente útil durante el desarrollo ya que al ser *Python* un lenguaje con tipado dinámico se pierde en ocasiones la información del objeto que se está requiriendo.

Conclusiones

- Es importante escoger la estructura de datos correcta para nuestra solución.
- Para reutilizar código es posible realizar una implantación de todos los métodos para una lista simple y restringir a algunos métodos únicamente para conseguir el comportamiento de una pila o cola según se requiera.

Referencias bibliográficas

Yarleque Ferrer, R. E. (2018). Estructura de datos Introducción, conceptos, tipos de datos, clasificación general, arrays, listas enlazadas, pilas, colas, inicialización y asignación de valores.

Anexos

- Diagrama de clases

Diagrama de clases Proyecto 2 - IPC2

Damian Ignacio Peña Afre