**Computación y estructuras discretas I**

**Método de ingeniería**

**Aplicación para la solución de la tarea Integradora I**

**Integrantes:**

* Johan Felipe Jojoa Bucheli
* Juan David Garzón Días
* Antonio José Hidalgo España

En base al contexto del problema planteado en el enunciado de la tarea integradora número uno, y siguiendo la orientación presentada en los materiales dispuestos con relación al desarrollo del método de ingeniería, este se desarrollará de la siguiente manera:

1. **Identificación del problema.**
2. **Recopilación de la información necesaria.**
3. **Búsqueda de soluciones creativas.**
4. **Transición de las ideas a diseños preliminares.**
5. **Evaluación y selección de la mejor solución.**
6. **Preparación de informes y especificaciones.**
7. **Implementación del diseño.**

**1. Identificación del problema.**

Necesidades:

* El usuario requiere simular el funcionamiento de los ascensores de sus futuras instalaciones.
* La solución debe implementar las estructuras de datos vistas a lo largo del curso.

Definición del problema:

Discreet Guys Inc. de la ciudad de Cali requiere un software que simule el funcionamiento de los ascensores que tendrán sus edificios.

**2. Recopilación de la información necesaria.**

En base al documento que fue suministrado para desarrollar la integradora, se obtuvieron los siguientes requerimientos:

* El programa debe recibir entradas de los movimientos de personal dentro de los edificios
* El programa debe permitir los movimientos dentro de cada edificio, mientras que el número de oficinas libres por pise concuerde con el número de personas
* El programa debe tener un ascensor que permita el movimiento entre pisos
* El programa imprimirá los resultados de los movimientos al final, este mostrará los movimientos efectuados y los movimientos fallidos

Requerimientos no funcionales

* El ingreso a los al ascensor se determina de acuerdo con el orden de llegada
* El ascensor tendrá prioridad para subir sobre bajar
* El ascensor siempre iniciara en el piso 1
* El programa contara con estructuras de datos para la resolución del problema (colas, pilas, colas de prioridad ,tablas hash)

Con respecto a las estructuras de datos, toda la información necesaria se obtuvo a través de las clases y las diapositivas de intu.

**3. Búsqueda de soluciones creativas.**

Por medio de una lluvia de ideas se buscaron varias formas de estructurar adecuadamente el mundo del problema, pasando por consideraciones como:

1. Tablas hash para cada piso, siendo cada hash una oficina.
2. Colas de prioridad para los pisos, siendo estas directamente ligadas al ascensor.
3. Una lista enlazada de pisos para cada edificio.
4. Una cola de prioridad para el ascensor, la cual priorizará los pisos superiores.

La solución en base a mostrar el movimiento de las personas dentro del edificio, al depender estrictamente de la estructura deseada, se definió cuando se eligió cómo estructurar el software.

**4. Transición de las ideas a diseños preliminares.**

Mediante observaciones analíticas se descartaron las dos primeras ideas, la número uno se descartó debido a que si el edificio tenía más de dos oficinas por piso no se podrían almacenar como una llave y un valor, la segunda idea resultó no ser eficiente debido a que el hacer que el ascensor tuviera su propia cola de prioridad hacía más sencillo identificar a qué piso debería dirigirse que el responsabilizar a los pisos para que gestionen ese aspecto.

Se usaron entonces las ideas 3 y 4, las cuales se desarrollarán en la siguiente sección.

**5. Evaluación y selección de la mejor solución.**

Al existir claridad con respecto a cómo se administrarían los diferentes tipos de estructuras de datos, se llegó al siguiente diagrama:



Diagrama, Esquemático

Descripción generada automáticamente

**6. Preparación de informes y especificaciones.**

Especificación del problema

Entradas:

* Cantidad de edificios.
* Identificador de cada edificio.
* Número de personas que se encuentran en el edificio.
* Cantidad de pisos.
* Cantidad de oficinas por piso.
* Nombre de cada persona.
* Piso donde se encuentra cada persona.
* Número de oficina a la que se dirige cada persona.

Salidas:

* Historial de movimientos dentro de cada edificio.
* Estado final de las oficinas en cada edificio.

Se diseñó un diagrama de clases que contiene las clases y estructuras de datos implementadas en el proyecto, además de sus relaciones y posibles métodos. Se encuentra en la documentación del proyecto.

**7. Implementación del diseño.**

Lo primero que se debe hacer es recibir las entradas necesarias para construir las clases y estructuras que se encuentran dentro del software, las entradas ingresan por consola de la siguiente manera:

* Número de edificios.
* Identificador del edificio A – Personas dentro del edificio – Cantidad de pisos – Cantidad de oficinas por piso.
* Nombre de la persona x – Piso en donde se encuentra la persona – Oficina a la que se dirige la persona.

Un ejemplo de entrada válida es el siguiente:

2

A 2 2 2

Carlos 2 2

Camilo 2 3

B 1 3 4

Juan 1 1

Luego de leer e introducir las entradas, se transmite la información a los constructores de las clases, después, mediante colas de prioridad se traslada a las personas a sus respectivos pisos, para que, dependiendo del orden en el que se bajen del ascensor, ingresen a su oficina.

Requerimientos funcionales

-El programa debe recibir entradas de los movimientos de personal dentro de los edificios

-El programa debe permitir los movimientos dentro de cada edificio, mientras que el numero de oficinas libres por pise concuerde con el número de personas

-El programa debe tener un ascensor que permita el movimiento entre pisos

-El programa imprimirá los resultados de los movimientos al final, este mostrará los movimientos efectuados y los movimientos fallidos

Requerimientos no funcionales

-El ingreso a los al ascensor se determina de acuerdo con el orden de llegada

-El ascensor tendrá prioridad para subir sobre bajar

-El ascensor siempre iniciara en el piso 1

-El programa contara con estructuras de datos para la resolución del problema (colas, pilas, colas de prioridad ,tablas hash)

Diseño de casos de prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar si los métodos básicos de la clase Table funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Table | contains | scenario1 | “key”  “key2” | Verdadero, en la Table hay una key “key”, y falso ya que no existe un elemento “key2” |
| Table | get | scenario2 | i=0 hasta 1000000 | Verdadero, Comprobar que se consigan los elementos indicados |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar si los métodos básicos de la clase Stack funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Stack | pop y push | scenario1 | “1” | Que agregue un elemente mediante push y lo saque mediante pop |
| Stack | size | scenario3 | 10000 | Que el valor del size sea igual a la entrada |
| Stack | isEmpty | scenario1  scenario3 |  | Que el primero esta vacío y el siguiente no |
| Stack | peek | scanrio3 | “10000” | Que el peek demuestre el ultimo valor ingresado |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar si los métodos básicos de la clase Queue funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Queue | isEmpty | scenario1  scenario2 |  | Que el primero esta vacío y el siguiente no |
| Queue | size | scenario3 | 10000 | Que el tamaño de la queue sea igual a 10000 |
| Queue | poll | scenario2 | “1” | Que el valor eliminado sea el primer elemento ingresado |
| Queue | peek | scenario3 | “1” | Que el valor retornado por el peek sea el primer elemento ingresado |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Objetivo de la prueba: Comprobar si los métodos básicos de la clase PriorityQueue funcionan correctamente | | | | |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| PriorityQueue | pop | scenario2 | 15 10 5 1 | Que elimine los valores de mayor a menor |
| PriorityQueue | peek | scenario2 | 15 | Que el valor retornado por el peek sea igual al más alto |
| PriorityQueue | size | scenario3 | 10000 | Que el valor de size sea igual a 10000 |
| PriorityQueue | isEmpty | scenario1  scenario2 |  | Que el primero esta vacío y el siguiente no |

Análisis de Complejidad temporal y espacial

sacarDeAscensor

|  |  |
| --- | --- |
| Instrucción | Veces que se repite |
| Stack<Persona> personas=ascensor.getPersonas(piso); | 1 |
| if(personas!=null) { | 1 |
| int size=personas.size(); | 1 |
| for(int i=0;i<size;i++) { | n+1 |
| pasillo.push(personas.pop()); | n |
| ascensor.setNumpersonas(); | n |

1+1+1+1+n+n+n

4+3n

Θ(n)= 4+3n

outP

|  |  |
| --- | --- |
| Intruccion | Veces que se repite |
| Persona out=new Persona(persona.getName(),persona.getObjetivo()); | 1 |
| persona=null; | 1 |
| return out; | 1 |

1+1+1

3

Θ(1)=3