# Método de la ingeniería

## Identificación del problema

La empresa Discreet Guys Inc. requiere el desarrollo de un programa que simule el funcionamiento de los ascensores y la asignación de oficinas de sus diferentes tipos de edificios inteligentes, teniendo en cuenta que los edificios tendrán varios pisos y cantidades de oficinas por piso. En los pisos del edificio se encontrarán diferentes cantidades de personas, las cuales dispondrán de los ascensores para moverse a otros pisos y llegar a sus oficinas. No deberá ser posible contener a mas de una persona por oficina, por lo que en caso de que todas las oficinas queden llenas se debe indicar quien se quedó sin oficina. La solución debe ser fácil de usar para el usuario y utilizar la cantidad mínima de recursos posible.

Adicionalmente se requiere que el programa cuente con un apartado de consultas, en donde se pueda acceder rápidamente a las personas que están en determinada oficina.

## Recopilación de información.

* Hay diversos tipos de edificio, con diferentes cantidades personas, pisos y oficinas por piso (Todos los pisos de un edificio tienen una misma cantidad de oficinas por piso).
* Las oficinas se identifican mediante un orden numérico ascendente a medida que se desciende por el edificio (La primera oficina (Con identificador 1) se encuentra en el piso superior y el numero aumenta a medida que se desciende de pisos).
* Al inicio del programa, a cada persona se le asigna una oficina donde ir.
* El programa esta simulado por tiempos, donde un tiempo equivale al movimiento del ascensor, salida o entrada de personas y cambios de edificio para una persona. Esto implica que por ejemplo salir del ascensor demorará el tiempo en entrar, moverse entre pisos y más el tiempo en salir. Lo anterior puede generar una acumulación de solicitudes para el ascensor.
* Los ascensores empiezan en el primer piso.
* La forma en la que se indicaran los datos de entrada del programa es: La primera línea indica la cantidad de edificios, la segunda línea contiene un identificador para cada edificio, luego el número de personas que se encuentra de momento en el edificio, el siguiente número es la cantidad de pisos y por último la cantidad de oficinas por piso. Las líneas posteriores indican el nombre de la persona que está en el edificio y el número del piso donde se encuentra. Por último, señalado en negrita, se indica el número de la oficina a la que se dirige la persona. A partir de la segunda línea se repiten estas entradas por cada edificio.
* Se pide que el ascensor se dirija a cada piso con base en el orden en el que se llamo al ascensor, pero el orden se ve afectado por la dirección en la que vaya el ascensor. De modo que si el ascensor esta subiendo se atienda primero a las solicitudes de pisos superiores.

Después haber detallado la información del problema, es necesario detallar la información de ciertas estructuras de datos que nos serán útiles para resolver el problema:

### Lista enlazada:

Una lista está formada por una serie de elementos llamados nodos los cuales son objetos que contiene como variable miembro un puntero asignado y variables de cualquier tipo para manejar datos. El puntero sirve para enlazar cada nodo con el resto de nodos que conforman la lista. De esto podemos deducir que una lista (lista) es una secuencia de nodos en el que cada nodo esta enlazado o conectado con el siguiente (por medio del puntero mencionado anteriormente).

El primer nodo de la lista se denomina cabeza de la lista y el último nodo cola de la lista. Este último nodo suele tener su puntero igualado a NULL Para indicar que es el fin de la lista.

Tomado de: <https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2019/i/guia-3.pdf>

### Cola:

Una cola en informática es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir. En este tipo de estructura el orden importa, ya que el primero de la cola deberá ser también el primero en atenderse mientras que el resto espera su turno en la cola de forma ordenada.

Basado en: <https://www.ecured.cu/Cola_(Estructura_de_datos)>

### Pila:

Una pila en informática es una lista ordinal o estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en informática debido a su simplicidad y la forma en la que se ordenan los elementos en la estructura.

Para el manejo de los datos se cuenta con dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, pop), que retira el último elemento apilado. En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado TOS, Top of Stack en inglés). La operación retirar permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo TOS.

Tomado de: <https://www.ecured.cu/Cola_(Estructura_de_datos)>

### Cola de prioridad:

Las colas de prioridad también son un tipo de datos abstracto. Cada elemento en la cola de prioridad tiene una prioridad, y aquellos con prioridad alta (o baja) serán eliminados primero, mientras que aquellos con la misma prioridad serán eliminados de acuerdo con su orden en la cola de prioridad.

Tomado de: <https://programmerclick.com/article/16147410/>

### Tabla hash:

Una tabla hash o mapa hash es una estructura de datos que asocia llaves o claves con valores. La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada usando el nombre, número de cuenta o id. Funciona transformando la clave con una función hash en un hash, un número que la tabla hash utiliza para localizar el valor deseado.

Tomado de: <https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2020/i/guia-8.pdf>

## Búsqueda de soluciones creativas:

En orden de buscar soluciones creativas se decidió optar por el método de “Lista de revisión”. Se decidió hacer una lista de revisión por cada área de del programa debido a que tiene varios elementos que pueden ser implementados de diferente manera.

La áreas u objetos en la que se puede dividir el programa son:

* Edificios
* Pisos
* Oficinas
* Ascensor
* Personas

Las ideas de implementación para cada área son:

### Edificios:

* Usar una tabla hash con direccionamiento abierto para guardar los edificios. Tienen un identificador que las diferencia; por lo que podemos colocar los edificios según su identificador en la tabla. No es necesario que estén ordenados los edificios. Nos definen la cantidad de edificios y en principio los edificios no deberían ser eliminados.
* Una tabla hash con direccionamiento directo para almacenar los edificios. Estos tienen un identificador que las diferencia; por lo que podemos colocar los edificios según su identificador en la tabla, evitando en principio las colisiones. Las colisiones, en caso de haber, se solucionarán mediante encadenamiento.
* Usar una lista para almacenar cada edificio gracias a que, nos definen la cantidad de edificios a almacenar y es más fácil de conocer la posición de cada elemento para obtenerlo.

### Pisos:

* Usar una lista para almacenar los pisos del edificio. Nos indican la cantidad de pisos de un edificio. Es fácil ordenarlos según un valor numérico que indique su posición (1 = piso 1, 2 = piso 2, etc. ) mediante algún algoritmo de ordenamiento.
* Usar una lista enlazada para almacenar los pisos del edificio, estando primero el primer piso.
* Usar una tabla hash con direccionamiento abierto para guardar los pisos. Nos definen la cantidad de pisos y en principio los pisos no deberían ser eliminados. Es posible asignar un identificador al piso según la posición de este (Piso 1, piso 2, piso 3, etc.)
* Una tabla hash con direccionamiento directo para almacenar los edificios. Es posible asignarle un identificador al piso según la posición de este (Piso 1, piso 2, piso 3, etc. ); por lo que en principio no debería haber colisiones. Las colisiones, en caso de haber, se solucionarán mediante encadenamiento.

### Oficinas:

* Usar una lista enlazada para almacenar las oficinas. Nos indican la cantidad de oficinas por piso. Tienen un identificador numérico. Es posible ordenarlo mediante este identificador numérico.
* Usar una tabla hash con direccionamiento abierto. Las oficinas tienen un identificador numérico diferente al resto de oficinas que permite una menor cantidad de colisiones al almacenarla. Se puede conjeturar la cantidad de oficinas mediante la multiplicación de los pisos por la cantidad de oficinas por piso, este dato servirá para definir el tamaño de la tabla.
  + De manera alternativa se puede crear una tabla hash de direccionamiento abierto por cada piso del edificio.
* Una tabla hash con direccionamiento directo para almacenar las oficinas. Se identificarán numéricamente por lo que en principio no debería haber colisiones. Las colisiones, en caso de haber, se solucionarán mediante encadenamiento.

### Ascensor:

* Implementar una cola de prioridad para el orden de recogida de las personas, en la que se priorizara la dirección en la que vaya el ascensor (Si el ascensor va subiendo recogerá primero a aquellos que estén en pisos superiores al que este está).
* Implementar una pila para almacenar las personas que entraran al ascensor.
* Implementar una cola para almacenar las personas que entrar al ascensor.

### Personas:

* Implementar una cola para asignar a las personas a una oficina según el orden en el que salieron del ascensor.
* Implementar una pila para asignar a las personas a una oficina.

## Transición de la formulación de ideas a los diseños preliminares.

Se rescata de cada área:

Edificio:

* La lista.
* La tabla hash.

Pisos:

* La lista.
* La lista enlazada se modifica a una lista doblemente enlazada para simular el desplazamiento del ascensor entre pisos.

Oficina:

* La lista enlazada.
* Tabla hash con direccionamiento abierto.

Ascensor:

* Cola de prioridad para el desplazamiento.
* Pila para el ingreso de personas.
* Cola para el ingreso de personas.

Personas:

* Implementar cola.
* Implementar pila.

El resto de las opciones se descarta.

## Evaluación y Selección de la Mejor Solución.

Se definen criterios para medir la efectividad de diferentes soluciones. Los criterios tienen asignados variables numéricas siendo un número menor aquella opción que cumple mejor con un criterio.

### Criterios:

Criterio A: Completitud de la solución.

1. Completo.
2. Poco completo

Criterio B: Acceso fácil a los datos.

1. Fácil acceso
2. Difícil acceso

Criterio C: Facilidad en la implementación.

1. Fácil de implementar.
2. Dificultad media.
3. Difícil de implementar.

### Evaluación:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opciones de Edificio | Criterio A | Criterio B | Criterio C | Total |
| Opción 1 | 1 | 1 | 1 | 3 |
| Opción 2 | 1 | 1 | 2 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opciones de Piso | Criterio A | Criterio B | Criterio C | Total |
| Opción 1 | 2 | 1 | 1 | 2 |
| Opción 2 | 1 | 1 | 2 | 2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opciones de Oficina | Criterio A | Criterio B | Criterio C | Total |
| Opción 1 | 1 | 1 | 1 | 3 |
| Opción 2 | 2 | 2 | 2 | 6 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opciones de Ascensor | Criterio A | Criterio B | Criterio C | Total |
| Opción 2 | 1 | 1 | 1 | 3 |
| Opción 3 | 2 | 1 | 1 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opciones de Persona | Criterio A | Criterio B | Criterio C | Total |
| Opción 1 | 1 | 1 | 1 | 3 |
| Opción 2 | 2 | 1 | 1 | 4 |

### Selección:

Según la evaluación anterior se seleccionan las soluciones:

* Edificio se almacenará en una lista.
* En piso hubo un empate, sin embargo, se decide priorizar la completitud de la solución por sobre su dificultad en la implementación. Por tanto, los pisos se almacenarán mediante listas doblemente enlazadas.
* Lista enlazada
* El desplazamiento del ascensor funcionara mediante una cola de prioridad. Este elemento no se evaluó con las otras opciones ya que ofrecen soluciones a aspectos diferentes del ascensor.
* El almacenamiento de las personas en el ascensor se realizará mediante una pila.
* Personas son asignadas a las oficinas mediante colas.

## Preparación de informes y especificaciones.

## Implementación del diseño.

# Especificación de requerimientos.

AUTH1: Cada persona va a una determinada oficina y su equipo debe poder operar hacia dónde se dirige cada persona (esto es piso y oficina).

AUTH2: El ingreso de las personas a los ascensores se determinará de acuerdo con el orden de llegada al ascensor. La salida será en el orden inverso.

AUTH3: Cada ascensor se dirige a cada piso con base al orden en el que los usuarios pulsan el botón. Dicho orden se va a ver afectado por la dirección en la que vaya el ascensor.

AUTH4: En caso de que todas las oficinas estén llenas se debe indicar cual persona se quedó sin oficina.

AUTH5: Se debe hacer un apartado de consultas, en donde se pueda acceder rápidamente a las personas que están en determinada oficina.

# Análisis de complejidad.

## Espacial.

## Temporal.

# Diseño de TAD.

|  |  |  |
| --- | --- | --- |
| **Pila** | | |
| Pila | | |
| Inv: Los elementos pertenecen a la pila, es la cantidad de elementos de la pila (Tamaño) y la cima será igual al último elemento (El del índice igual al tamaño de la pila) | | |
| Operaciones: | | |
| * Pila | - | pila de → |
| * Empujar | Pila x Nodo | pila de → |
| * Pop | Pila | →Pile |
| * Hacia arriba | Pila | Nodo → |
| * EsVacia | Pila | → Booleano |

|  |
| --- |
| **Pila– Operación constructora.** |
| Construye una pila vacía.  {pre: - }  {post: Pila s = } |

|  |
| --- |
| **Push – Operación modificadora.** |
| Añade un nuevo elemento a una pila .  {pre: Pila y un elemento , o una Pila vacía y un elemento }  {post: Pila o } |

|  |
| --- |
| **Pop – Operación modificadora.** |
| Extrae de la pila el último elemento insertado.  {pre: Pila o  {post: Pila y elemento } |

|  |
| --- |
| **Cima – Operación analizadora.** |
| Obtiene el último elemento insertado de la pila sin eliminarlo.  {pre: Pila o }  {post: Elemento } |

|  |
| --- |
| **EsVacia– Operación analizadora.** |
| Determina si la pila está vacía o no.  {pre: Pila }  {post: Valor verdadero (true) si , o falso (false) si } |

|  |  |  |
| --- | --- | --- |
| **Cola** | | |
| Cola = | | |
| inv: {} | | |
| Operaciones: | | |
| * Cola | - | Cola |
| * Enqueue | Cola Elemento | Cola |
| * Dequeue | Cola | Elemento |
| * Frente | Cola | Elemento |
| * EsVacia | Cola | Boolean |

|  |
| --- |
| **Cola– Operación constructora.** |
| Construye una cola vacía.  {pre: - }  {post: Cola } |

|  |
| --- |
| **Enqueue – Operación modificadora.** |
| Añade un nuevo elemento a una cola .  {pre: Cola y un elemento , o una Cola vacía Q y un elemento }  {post: Cola o Q } |

|  |
| --- |
| **Dequeue – Operación modificadora.** |
| Extrae de la Cola el primer elemento insertado.  {pre: Cola o }  {post: Cola y elemento = } |

|  |
| --- |
| **Frente – Operación analizadora.** |
| Obtiene el primer elemento insertado de la Cola sin eliminarlo.  {pre: Cola o }  {post: Elemento } |

|  |
| --- |
| **EsVacia– Operación analizadora.** |
| Determina si la Cola está vacía o no.  {pre: Cola }  {post: Valor verdadero (true) si , o falso (false) si } |

|  |  |  |
| --- | --- | --- |
| **Cola de prioridad** | | |
| Cola de prioridad =  Cola de prioridad | | |
| inv: {} | | |
| Operaciones: | | |
| * ColaDePrioridad | - | P |
| * Enqueue | P Elemento | P |
| * Dequeue | P | Elemento |
| * Maximo | P | Elemento |
| * EsVacia | P | Boolean |
| * IncrementarLlave |  | P |

|  |
| --- |
| **ColaDePrioridad– Operación constructora.** |
| Construye una cola vacía.  {pre: - }  {post: Cola de prioridad P } |

|  |
| --- |
| **Enqueue – Operación modificadora.** |
| Añade un nuevo elemento a una cola .  {pre: Cola de prioridad P y un elemento , o una Cola de prioridad vacía P y un elemento }  {post: Cola de prioridad o P } |

|  |
| --- |
| **Dequeue – Operación modificadora.** |
| Extrae de la Cola de prioridad el primer elemento insertado.  {pre: Cola de prioridad o }  {post: Cola de prioridad y elemento = } |

|  |
| --- |
| **Maximo – Operación analizadora.** |
| Obtiene el elemento en el frente de la Cola de prioridad sin eliminarlo; este elemento será el de máxima prioridad.  {pre: Cola de prioridad o }  {post: Elemento } |

|  |
| --- |
| **EsVacia– Operación analizadora.** |
| Determina si la Cola de prioridad está vacía o no.  {pre: Cola de prioridad }  {post: Valor verdadero (true) si P, o falso (false) si P } |

|  |
| --- |
| **IncrementarLlave– Operación modificadora.** |
| Decrementa el valor de la clave del elemento x de la cola de prioridad P a un nuevo valor de la clave K.  {pre: Cola de prioridad , Elemento y Clave }  {post: P con x en una nueva posición y con diferente clave, o P con x en la misma posición y con diferente clave} |

|  |  |  |
| --- | --- | --- |
| **Lista enlazada** | | |
| Lista enlazada =  Lista enlazada | | |
| inv: {} | | |
| Operaciones: | | |
| * ListaEnlazada | - | Lista enlazada |
| * Añadir | Lista enlazada Nodo | Lista enlazada |
| * Remover | Lista enlazada Integer | Lista enlazada |
| * EsVacio | Lista enlazada | Boolean |
| * Tamaño | Lista enlazada | Integer |
| * ObtenerNodo | Lista enlazada Integer | Nodo |

|  |
| --- |
| **ListaEnlazada – Operación constructora.** |
| Crea una lista enlazada L vacía.  {pre: -}  {post: Lista enlazada } |

|  |
| --- |
| **Añadir – Operación modificadora.** |
| Añade un nuevo elemento a la lista enlazada  {pre: Lista enlazada L o L, y elemento }  {post: Lista enlazada L o L} |

|  |
| --- |
| **Remover – Operación modificadora.** |
| Extrae de la Lista enlazada el elemento en la posición indicada  {pre: Lista enlazada o L, y un numero entero tal que }  {post: Lista enlazada } |

|  |
| --- |
| **EsVacio – Operación analizadora.** |
| Indica si la lista enlazada está vacía o no.  {pre: Lista enlazada }  {post: Valor verdadero (true) si , o falso (false) si } |

|  |
| --- |
| **Tamaño – Operación analizadora.** |
| Indica la cantidad de elementos que contine la lista enlazada.  {pre: Lista enlazada }  {post: = n; } |

|  |
| --- |
| **ObtenerNodo – Operación analizadora.** |
| Obtiene el elemento a partir de un valor entero indicado; Obtiene el elemento en la posicion indicada.  {pre: Lista enlazada y un numero entero tal que }  {post: Elemento } |

# Casos de prueba