Diseño de un teclado

Proyecto de PROP

Mariona Aguilera Folqué - mariona.aguilera@estudiantat.upc.edu
Eneko Sabaté Iturgaiz - eneko.sabate@estudiantat.upc.edu
Miguel Angel Montero Flores - miguel.angel.montero@estudiantat.upc.edu
Pol Ribera Moreno - pol.ribera@estudiantat.upc.edu

Documentación del proyecto presentada por
el subgrupo 41.2

Primera entrega
Version: 1.0
20/11/2023

Index

1

Casos de Uso

1.1. Diagrama de casos de uso

1.2. Descripción de casos de uso

1.2.1. Crear Teclado

Comportamiento:

- El usuario define el nombre del nuevo teclado a ser creado
- Comienza caso de uso “Seleccionar algoritmo”
- Comienza caso de uso “Seleccionar alfabeto”
- Comienza caso de uso “Seleccionar Textos/Listas”
- El sistema muestra el nuevo teclado
- El teclado se guarda

Errores posibles y cursos alternativos:

- Si ya existe un teclado con ese nombre, no se crea y salta una excepción
- Si alguno de los textos o listas no corresponden al alfabeto, no se crea el teclado y salta una excepción
- Si el algoritmo seleccionado es QAP y el alfabeto tiene más de 19 caracteres, no se crea el teclado y salta una excepcion

1.2.2. Modificar Teclado

Comportamiento:

- El usuario decide el teclado que quiere modificar
- Comienza caso de uso “Seleccionar Textos/Listas”
- El usuario decidirá si quiere introducir o no un nuevo alfabeto, en caso afirmativo comienza caso de uso “Seleccionar alfabeto”
- El sistema muestra el nuevo teclado formado por el algoritmo original y la nueva información
- El teclado se guarda

Errores posibles y cursos alternativos:

- Si el teclado no existe salta una excepción y no se crea el teclado
- Si los textos o listas no concuerdan con el alfabeto usado, salta una excepción y no se crea el teclado

1.2.3. Seleccionar algoritmo

Comportamiento:

- El usuario escoge cual de los dos algoritmos para crear un teclado quiere utilizar
- El sistema devuelve el algoritmo correspondiente

Errores posibles y cursos alternativos:

- Si no se escoge un algoritmo existente, salta una excepción y no se devuelve el algoritmo

1.2.4. Seleccionar alfabeto

Comportamiento:

- El usuario escoge el alfabeto a utilizar
- El sistema devuelve el alfabeto correspondiente

Errores posibles y cursos alternativos:

- Si el alfabeto no existe, salta una excepción y no se devuelve el algoritmo

1.2.5. Seleccionar Textos/Listas de palabras

Comportamiento:

- El usuario escoge los textos que quiere utilizar
- El usuario escoge las listas de palabras que quiere utilizar
- El sistema devuelve las listas y los textos en cuestión

Errores posibles y cursos alternativos:

- Si alguno de los textos no existe, salta una excepción y no se devuelve nada
- Idem para las listas

1.2.6. Guardar Teclado

Comportamiento:

- El usuario define el nombre del teclado
- El sistema guarda el teclado con toda la información necesaria

Errores posibles y cursos alternativos:

- Si un teclado con el mismo nombre existe, salta una excepción y no se guarda el teclado

1.2.7. Borrar Teclado

Comportamiento:

- El usuario escoge el teclado a borrar
- El sistema borra la información del teclado

Errores posibles y cursos alternativos:

- Si el teclado no existe, salta una excepción y no se borra ningún teclado

1.2.8. Importar Texto

Comportamiento:

- El usuario importa un texto y le asigna un nombre
- El sistema guarda la información del texto

Errores posibles y cursos alternativos:

- Si el nombre del texto ya está asignado a un alfabeto, texto o lista de palabras no se crea el texto y salta una excepción

1.2.9. Importar Lista de Palabras

Comportamiento:

- El usuario importa una lista de palabras y le asigna un nombre
- El sistema guarda la información introducida

Errores posibles y cursos alternativos:

- Si el nombre del texto ya está asignado a un alfabeto, texto o lista de palabras no se crea el texto y salta una excepción

1.2.10. Importar Alfabeto

Comportamiento:

- El usuario importa un alfabeto y le asigna un nombre
- El sistema guarda la información del alfabeto

Errores posibles y cursos alternativos:

- Si el nombre del texto ya está asignado a un alfabeto, texto o lista de palabras no se crea el texto y salta una excepción

1.2.11. Modificar Texto

Comportamiento:

- El usuario indica qué texto quiere modificar
- El usuario introduce el nuevo texto
- El sistema guarda la nueva información en el texto modificado

Errores posibles y cursos alternativos:

- Si el texto no existe salta una excepción y no se hace ninguna modificación

1.2.12. Modificar Alfabeto

Comportamiento:

- El usuario indica que alfabeto quiere modificar
- El usuario introduce un nuevo alfabeto
- El sistema guarda la nueva información en el alfabeto modificado

Errores posibles y cursos alternativos:

- Si el alfabeto no existe salta una excepción y no se modifica ningún alfabeto

1.2.13. Modificar Lista de Palabras

Comportamiento:

- El usuario indica la lista de palabras que quiere modificar
- El usuario introduce una nueva lista de palabras
- El sistema guarda la nueva información en la lista modificada

Errores posibles y cursos alternativos:

- Si la lista de palabras no existe no se modifica ninguna lista y salta una excepción

1.2.14. Borrar Texto

Comportamiento:

- El usuario elige qué texto quiere borrar
- El sistema borra el texto seleccionado

Errores posibles y cursos alternativos:

- Si el texto no existe, no se borra ningún texto y salta una excepción

1.2.15. Borrar Alfabeto

Comportamiento:

- El usuario elige qué alfabeto quiere borrar
- El sistema borra el alfabeto seleccionado
- El sistema guarda la nueva información en la lista modificada

Errores posibles y cursos alternativos:

- Si el alfabeto no existe, no se borra ningún alfabeto y salta una excepción
- Si el alfabeto ha sido usado para alguno de los teclados actualmente guardados, el alfabeto no se borra y salta una excepción

1.2.16. Borrar lista de Palabras

Comportamiento:

- El usuario elige qué lista de palabras quiere borrar
- El sistema borra la lista en cuestión

Errores posibles y cursos alternativos:

- Si la lista no existe, no se borra ninguna lista y salta una excepción

1.2.17. Salir

Comportamiento:

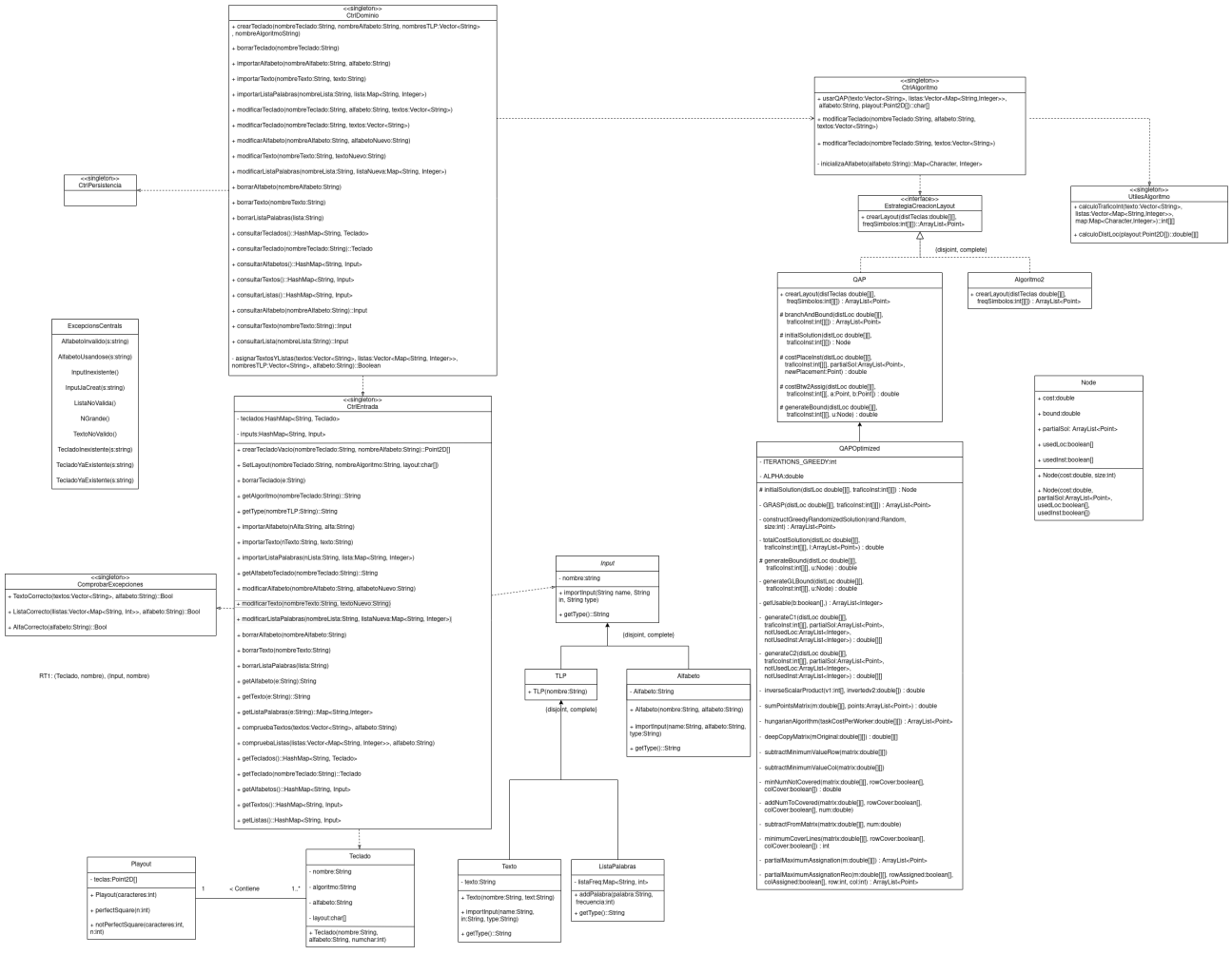
- Se detiene la ejecución del programa

2

Modelo Conceptual de Datos

2.1. Diagrama UML

Se puede encontrar el diagrama con una mejor resolución en el mismo directorio en el que se encuentra este documento.



2.2. Descripción diagrama de clases

2.2.1. CtrlDominio

El controlador del Dominio se encarga de proporcionar el flujo de información del resto de controladores, en este caso el controlador de algoritmo, entrada, persistencia y presentación. Como función adicional, hace saltar las excepciones que le lleguen.

2.2.2. CtrlAlgoritmo

El controlador del algoritmo se encarga de utilizar la información proporcionada por el controlador de dominio para hacer uso de los algoritmos que se quieran, y también se encarga de comunicar las funciones de UtilesAlgoritmo al EstrategiaCreacionLayout.

2.2.3. EstrategiaCreacionLayout

Interfaz que especifica los métodos necesarios a la hora de implementar algoritmos con el objetivo de resolver el problema de creación de layouts. El método definido en la intefaz çrearLayoutretorna un ArrayList¡Point¿que servirán para especificar el símbolo que corresponde con cada tecla del layout.

2.2.4. QAP

Clase que implementa un algoritmo para resolver el Quadratic Assignment Problem. Esta clase implementa la interfaz EstrategiaCreacionLayout, por lo que el objetivo de esta es resolver el problema de la creación de layouts reduciendolo al problema del QAP.

2.2.5. QAPOptimized

Subclase de la clase QAP. Haciendo uso del polimorfismo reimplementa clases de QAP con el objetivo de mejorar el rendimiento de este, valiéndose de técnicas de lower bounds.

2.2.6. Node

Estructura de datos usada por el algoritmo BranchAndBound para resolver el problema QAP. Este nodo representa una solución parcial en el árbol de soluciones creado por el algoritmo BranchAndBound. Da información sobre: el coste actual, una lower bound para la solución actual, la solución parcial actual, y las instalaciones y ubicaciones usadas hasta el momento

2.2.7. UtilesAlgoritmo

Se encarga de calcular los parámetros necesarios para utilizar correctamente los algoritmos de creación de layout. La función más complicada es la del cálculo del tráfico calculoTraficoInt": Devuelve la matriz del tráfico que corresponde a cuántas veces una letra es adyacente a otra. La fila y columna 1 representa a cuantas "a" es adyacente la letra correspondiente. En la primera fila se ve cómo la letra "a" es adyacente 0 veces a "a" y 1 a "b". En la segunda se ve cómo la b es adyacente a una "a" y 2 a una "b".

2.2.8. CtrlEntrada

El controlador de entrada tiene como función principal controlar el tráfico de datos de las entradas, que son Teclados e Inputs. Como función secundaria tiene también comprobar las excepciones que estos puedan tener. Un ejemplo de ello es, a la hora de crear un teclado, que alguno de los textos no coincida con el alfabeto, en cuyo caso se encargará de informar al controlador del dominio de la excepción.

2.2.9. Teclado

Los objetos de esta clase representan los teclados que los usuarios pueden crear, modificar y borrar. Un teclado se identifica por nombre y contiene el nombre del algoritmo, el nombre del alfabeto utilizado, el layout y el physical layout.

2.2.10. Input

Clase abstracta que representa todo input que el usuario puede introducir al programa. Son objetos utilizados únicamente para crear nuevos teclados o modificarlos. Se identifican por nombre.

2.2.11. Alfabeto

Subclase de Input. Representación de un alfabeto; se hace uso de un string para guardar la información del alfabeto (cada caracter del string equivale a un símbolo del alfabeto).

2.2.12. TLP

Subclase abstracta de Input. Abreviación de Textos/ListasPalabras. Mediante estos podemos definir las frecuencias entre símbolos a la hora de crear nuevos teclados o modificarlos.

2.2.13. Texto

Subclase de TLP. Representación de los textos disponibles en el programa. Se utiliza un string para definir el texto en sí.

2.2.14. ListaPalabras

Subclase de TLP. Representación de listas de palabras disponibles en el programa. Se utiliza un Map¡String, Integer¿para definirlas, tal que la key representa una palabra y el value la frecuencia de la palabra.

3

Estructuras de Datos y Algoritmos

3.1. Descripción algoritmos

En el siguiente apartado describiremos los algoritmos principales implementados para el funcionamiento del programa.

3.1.1. Branch And Bound

El algoritmo Branch And Bound es el algoritmo principal que usa nuestro programa a la hora de crear layouts. Este algoritmo construye un árbol de soluciones factibles y, cuando llega a las hojas del árbol, compara la solución final de la hoja con la mejor solución hasta el momento. Con el objetivo de mejorar la eficiencia, se introduce el concepto de Bounding: Para cada solución parcial se hace el cálculo de una cota inferior para todas las soluciones completas conseguibles a partir de esta y, si esta cota es mayor a la mejor solución encontrada hasta el momento, se descarta la solución. Aun así, el algoritmo es de coste peor exponencial, lo que lo hace inviable para casos grandes.

Los siguientes algoritmos son los que hemos implementado con el fin de calcular estas cotas inferiores.

3.1.2. GRASP

GRASP, o Greedy randomized adaptive search procedure, es el algoritmo utilizado para econtrar una primera cota inferior como preproceso del BranchAndBound. Este coniste en una serie de iteraciones, donde para cada una se construye una solución greedy randomizada para el problema QAP. Se va guardando la mejor solución encontrada hasta al momento y, cuando termina el bucle, devuelve la mejor solución encontrada de todas las construidas.

3.1.3. Gilmore-Lawler Bound

Calcula una cota inferior para una solución parcial del árbol de soluciones generado por el BranchAndBound. Dada una solución parcial, la cota de Gilmore-Lawler se puede calcular como la suma del coste de las instalaciones ya emplazadas, el coste de las instalaciones que qudean por colocar respecto a las ya colocadas y el coste entre instalaciones aún no emplazadas. Estos dos últimos términos no se puedes saber con exactitud, por eso este algoritmo los intenta aproximar mediante una cota inferior de esos costes. Este algoritmo proporcionará una cota inferior que se usa en el BranchAndBound, con el objetivo de podar las ramas que no pueden proporcionar soluciones óptimas.

3.1.4. Hungarian Algorithm

El Hungarian Algorithm es un algoritmo que resuelve problemas de asignación lineal. Para nuestro caso concreto, lo utilizamos para calcular de una manera más precisa la cota inferior generada por Gilmore-Lawler Bound. Esto se traduce en un mejor funcionamiento del algoritmo BranchAndBound en cuanto a eficiencia de computo.

3.2. Descripción estructuras de datos

En el siguiente apartado describiremos las estructuras de datos implementadas en las clases principales del programa.

3.2.1. Node

La clase Node implementada representa un nodo en el árbol de soluciones generado por el BranchAndBound. Así, un nodo tiene información sobre el coste de la solución parcial actual, una cota inferior para la solución parcial, la solución en sí, y las instalaciones y ubicaciones utilizadas hasta el momento. Esta estructura de datos es imprescindible para el correcto funcionamiento del BranchAndBound, y por lo tanto, para resolver el QAP.

Se ha implementado la solución parcial en Node como un ArrayList¡Point¿, donde cada punto representa una asignación. Se ha elegido utilizar la estructura de datos Point ya que nos era útil al ser análoga a un pair¡int, int¿, y esto era justo lo que necesitabamos para expresar asignaciones. En cuanto a la ArrayList, hemos decidido usarla en vez de un Array ya que los ArrayList proporcionan un uso dinámico de la memoria, cosa que con los Arrays es imposible ya que estos proporcionan memoria estática. Por otro lado, la implementación de las ubicaciones/instalaciones usadas la hemos hecho mediante Arrays, ya que en este caso sí que nos vale simplemente con memoria estática

3.2.2. QAP

Las dos estructuras de datos más importantes para implementar la clase QAP (y QAPOptimized) son la PriorityQueue y Node. La PriorityQueue es usada en el algoritmo BranchAndBound para almacenar y poder coger las nodos del árbol de soluciones parciales. Esta estructura nos es ideal ya que tratamos el problema de QAP mediante una estrategia .^{ea}ger", y por lo tanto necesitamos una estructura de datos mediante la que guardar las soluciones parciales y tratar las "mejores"soluciones cuanto antes (por eso es de prioridad). La segunda estructura es el Node, la cual utilizamos como elementos contenidores de la cola de prioridad. Esta estructura es totalmente imprescindible ya que es la forma que tenemos de guardar la información de la solución actual para un cierto nodo del árbol de soluciones.

3.2.3. CtrlEntrada

Con el objetivo de la persitencia de datos entre la ejecución de comandos del programa, se han creado estructuras de datos que mantengan el estado del programa cuando se realiza un cambio en el sistema. Para implementar el guardado de teclados hemos utilizado un HashMap¡String, Teclado¿, ya que nos permite operar con teclados usando su nombre como identificador de una manera eficiente. En cuanto a los Inputs hemos utilizado también un HashMap¡String, Input¿, por la misma razón que los teclados. Además nunca tendremos dos inputs con el mismo nombre, así que usar esta estructura no dará error.

3.2.4. Playout

Hemos utilizado un array de Point2D (Point2D[]) para guardar las teclas. La idea inicial era utilizar una matriz que contuviese los puntos que iban a ocupar las teclas en el layout fisco. Sin embargo, para layouts con un número impar de caracteres, la matriz quedaba con espacios en null. Para evitar ocupar más espacio del necesario, decidimos que el atributo de la clase Playout sería un array con el tamaño exacto que se requería.

3.2.5. Teclado

Para guardar el Layout que tiene el teclado, hemos utilizado un array de caracteres (char[]). Utilizamos un char[] frente a un String debido a que nos parece más sencillo la asignación del layout con el Playout, puesto que al iterar por un String se debe llamar a las funciones charAt() o toCharArray().

3.2.6. ListaPalabras

Para tener la lista de palabras con frecuencia, hemos usado un Map¡String, Integer¿, para poder enlazar fácilmente cada palabra con su frecuencia.

3.2.7. ComprobarExepciones

Hemos usado un HashSet¡Character¿en la función de comprobar si un alfabeto es válido, ya que en él ponemos todos los caracteres vistos hasta el momento y si alguno se repite sabremos que el alfabeto no es válido.

4

Clases implementadas por Miembro del equipo

En la siguiente tabla se especifica quien ha implementado cada clase. Respecto al la documentación, cada persona se ha ocupado de desarrollar la documentación pertinente a las clases implementadas, o se han hecho en conjunto, como es el caso de la descripción de los casos de uso.

Mariona Aguilera	**Eneko Sabaté**	**Miguel Angel Montero**	**Pol Ribera**
Teclado	EstrategiaCreacion Layout	CtrlDominio	CtrlEntrada
PLayout	QAP	CtrlAlgoritmo	Comprobar Excepciones
Input	QAPOptimized	UtilesAlgoritmo	
Alfabeto	Node	driverCtrlAlgoritmo	
TLP	QAPTest	driverCtrlDominio	
Texto	UtilesAlgoritmoTest		
ListaPalabras			
driverCtrlEntrada			
AlfabetoTest			
ListaPalabrasTest			
Playout			
TecladoTest			
TextoTest			