UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Estructura de Datos y Análisis de Algoritmos

Laboratorio 2 : Poto Sucio

Bastián Gonzalo Vera Palacios

Profesor: Jacqueline Köhler

Ayudantes: Nicole Henríquez

Sebastián Vallejos

Javiera Torres

Santiago - Chile
2-2017

Tabla de Contenidos

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

El poto sucio es un juego de cartas para más de dos jugadores, los cuales deben lograr deshacerse de los pares de su mano antes de que finalice el juego con el fin de evitar quedarse con el joker sin pareja. Durante el desarrollo del juego se permitirá al usuario enfrentarse con uno, dos o tres contrincantes según sea su elección, los cuales serán controlados por la computadora y cada uno posee una comportamiento diferente en la etapa final del juego, que será la única en la que participe el usuario ya que la etapa uno y dos serán generadas automáticamente.

Durante este laboratorio se solicita que implemente el juego del poto sucio. El objetivo por desarrollar es adquirir mayor conocimiento de las listas y cómo se implementan dentro del código, la facilidad que nos otorgan al permitir cambiar la información elemento por elemento y así acceder a la información de los jugadores y sus cartas durante el transcurso del juego. También comprender el funcionamiento de las estructuras de datos, que junto habilidades adquiridas durante el curso nos proporcionará las herramientas necesarias para enfrentar y desarrollar los diversos problemas que se presenten durante el desarrollo de este laboratorio.

En este informe se explicara detalladamente como fue analizado el problema, que sub-problemas fueron encontrados y que herramientas se utilizaron para poder resolverlos, los algoritmos y estructuras implementados en las diversas funciones creadas durante el desarrollo del código, los problemas que surgieron y como fueron planteadas las diversas soluciones para conseguir el desarrollo y funcionamiento completo del juego y finalmente las conclusiones que se obtuvieron a partir de los resultados.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1 MARCO TEÓRICO

Una vez analizada la temática de este juego, se plantea que existan jugadores los cuales vayan avanzando según turnos y a la vez cada uno de ellos debe tener su mano de cartas para poder retirar las parejas que haya formado e irlas desechando hasta llegar al punto en que solo quede una carta en la mano de un jugador, lo que implica la derrota del participante en el juego.

Antes de poder entregar a cada jugador su mano de cartas, se tiene que generar un mazo, el cual consiste en una baraja inglesa, que incluye en ella un par de jokers, de los cuales uno será extraído para poder seguir la dinámica del juego de eliminar pares, debido a que el joker se encuentra sin una pareja se permitirá encontrar al perdedor del juego. A la hora de generar el mazo, se utilizó un arreglo estático con las cartas ya definidas en el. Junto con esto se definen dos variables las cuales usaremos para revolver el mazo y generar un nuevo arreglo de cartas con la ayuda del módulo rand().

Para poder abordar la dinámica del juego, se plantea una solución, la cual consiste en la implementación de listas, ya que estas facilitan el trabajo al ser una estructura dinámica y que permite generar los turnos entre los jugadores y a la vez, la mano de cartas de cada uno de estos. El hecho de utilizar listas permite trabajar con cada jugador de manera independiente y con un fácil acceso para la lectura/escritura de información en este. Como se quieren definir distintos tipos de datos para cada lista, se utilizarán estructuras, las cuales nos ayudarán a mantener un formato en cada una de nuestros nodos y listas con la información del jugador y de las cartas.

Cuando ya se cuente con todas las funciones de listas creadas junto con sus respectivas estructuras, se puede trabajar en una función principal, la cual se encargará de llamar a todas las funciones y empezar a ensamblar todas las partes de nuestro juego.

Dentro de la función principal llamada *iniciar()* se deben realizar todas las consultas al usuario, respecto al nombre con el que jugará y la cantidad de rivales con los que quiera jugar.

2.2 HERRAMIENTAS Y TÉCNICAS

Entre de las herramientas utilizadas para el desarrollo de este laboratorio, se encuentran las librerías de:

- stdio.h: Para la entrada, salida de datos y manejo de archivos.
- stdlib.h: Utilizada para el manejo de la memoria dinámica.
- time.h: Utilizada para generar un número aleatorio con la función *srand( )*.
- ctype.h: Usada en la función *tolower( )* para ajustar el carácter recibido por el usuario.

Otro tipo de herramienta son los Punteros, los cuales se utilizaron para poder desplazarse entre los distintos tipos de nodos creados en cada lista, también para realizar modificaciones a las listas tales como agregar, borrar elementos o simplemente almacenar un dato en una variable asignada con memoria dinámica.

También se encuentran listas, las cuales fueron fundamentales para poder realizar los movimientos dentro del juego entre los jugadores y a la vez para poder desplazarnos entre las cartas en las manos.

Ya que para desarrollar este juego se necesitaba inicializar las listas, se utilizó para esto arreglos dinámicos con el fin de poder trabajarlos óptimamente. Para ello usamos la función *malloc( ),* la cual es la encargada de solicitar la memoria según las dimensiones solicitadas, y la función *free( ),* la cual es la que se utiliza al final del desarrollo del programa para poder liberar la memoria utilizada anteriormente.

Para sacar una carta dentro de una mano de tal manera que la posición de estas sea aleatoria, se usan las funciones *srand( )* y *rand( )*, las cuales se encargan de generar un número aleatorio, el cual es ubicado por la función *buscarCarta( )* y entregado como retorno de la función.

2.3 ALGORITMOS Y ESTRUCTURAS DE DATOS

Para ordenar de manera óptima la información se utilizan diferentes estructuras de datos, una estructura sirve para agrupar distintos tipos de datos con el fin de ordenar y trabajar de manera más clara la información.

Las estructuras utilizadas fueron las siguientes:

Turnos: Estructura utilizada para generar una lista circular con los Nodos Jugadores para darle dinámica al juego.

Jugador: Nodo el cual contiene la información de cada jugador, como su nombre, su mano, un verificador y un puntero al siguiente jugador.

Mano: Estructura utilizada para generar una lista enlazada con los Nodos carta para poder generar la mano de cada jugador.

Carta: Nodo el cual contenía el valor de cada carta y el puntero hacia la carta siguiente.

Para utilizar las distintas funcionalidades que posee una lista, se implementan modelos de operaciones definidas, llamados TDA(Tipo de Dato Abstracto), los cuales se aplican durante la implementación del código.

El TDA desarrollado es el siguiente:

TDA Mano de Cartas

Mano * crearMano() ~ T(n) = 4

Función que inicializa la estructura Mano y define en ella sus valores iniciales. Se pide memoria del tipo Mano y se crean los punteros head, tail y size inicializados en Null.

Entradas: Sin entrada

Salidas: lista inicializada

void agregarCarta(Mano * list,int valor) ~ T(n) = 7

Función encargada de insertar un Nodo Carta dentro de la lista Mano, es decir, las cartas de la mano de cada jugador. Si no existe ningún nodo en la lista Mano, la función inserta el primero y lo define como cabeza y cola de la lista.

Entradas: lista Mano, valor a agregar

Salidas: Sin salida

Mano * borrarDeMano(Mano * list,int valor) ~ T(n) = 2n+2

Función encargada de eliminar un nodo de la lista Mano según su valor. Se recorre la lista Mano hasta encontrar la carta con el valor a borrar y después se define el puntero del anterior al siguiente del elemento a eliminar omitiendo el valor para luego ser eliminado.

Entradas: lista Mano, valor a borrar

Salidas: lista Mano

void EliminarPares(Mano * list) ~ T(n) = 2n^2+5n+3

Función que recorre la lista Mano hasta encontrar dos elementos iguales y posterior a esto, eliminar ambos elementos.

Para esto, se pasaron todos los elementos de la lista dentro de un arreglo y a la vez, se eliminaron de esta, con el fin de facilitar la eliminación de elementos. Dentro del arreglo se realiza la doble búsqueda de elementos repetidos y una vez localizados, dentro de un arreglo auxiliar se definen como valores Nulos. Posterior a esto se revisan los elementos dentro de nuestro arreglo auxiliar, para así ir agregando a la lista los elementos que sean distintos al valor nulo. Finalmente nos encontramos con la lista Mano con los valores duplicados ya removidos.

Entradas: lista Mano

Salidas: Sin salida

TDA Turnos de cada Jugador

Turnos * crearTurnos( ) ~ T(n) = 3

Función encargada de inicializar la lista Turnos y definir sus valores iniciales. Se pide memoria del tipo Turnos y se crean los punteros head, tail y size inicializados en Null.

Entradas: Sin entrada

Salidas: lista Turnos

void agregarJugador(Turnos * list,char* nombre) ~ T(n) = 7

Función que inserta un Nodo Jugador dentro de la lista Turnos. Al igual que la función agregar cartas, en caso de no existir algún nodo en la lista, define el primero que agrega como cabeza y cola de la lista.

Entradas: lista Turnos, el nombre del jugador a guardar

Salidas: Sin salida

TDA Mazo

void revolverMazo( ) ~ T(n) $= n^2 + 3n$

Función que cambia las posiciones de los valores en el arreglo mazo de manera aleatoria. Utiliza el la función *rand()* y *srand()* para generar nuevas posiciones e ir insertándose en el arreglo que se retorna.

Entradas: Sin entradas, trabaja directo al arreglo Mazo

Salidas: Sin salidas

Función Principal

Iniciar( ) ~ T(n) $= 3n^3 + 10n^2 + 13n + 31$

Función principal en nuestro codigo la cual se encarga de juntar todo el TDA de Listas creado anteriormente y poder utilizarlo para el desarrollo del juego.

Dentro de las etapas que tenemos en la Función Iniciar( ) se encuentran:

- Solicitud de al usuario de la información que se utilizará.
- Inicialización de la lista Turnos con los jugadores correspondientes.
- Inicialización de la Mano de cada jugador junto con la repartición de cartas inicial.
- Etapa 1 : Eliminación de pares en la Mano.
- Etapa 2 : Robo de cartas al Mazo por turnos, con su respectiva eliminación de pares.
- Etapa 3 : Robo de cartas al jugador siguiente, eliminación de pares y Conclusión del juego.

Entradas: Sin Entrada

Salidas: Sin Salida

Anexo a estas funciones nos encontramos con otras funciones las cuales cumplen un rol puntual dentro del código, tales como *buscarCarta, mostrarMano, obtenerPrimerValor,* entre otras.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

Estos son los T(n) calculados del código:

Tabla 1.1: Tabla de tiempos de ejecución

Funciones	T(n)
crearMano ( )	4
agregarCarta ( )	7
mostrarMano ( )	$4n + 1$
buscarCarta ( )	$2n + 1$
borrarDeMano( )	$2n + 2$
obtenerPrimerValor( )	1
obtenerUltimoValor( )	$n + 1$
obtenerAleatorio( )	$2n + 2$
eliminarPares( )	$2n^2 + 5n + 3$
revolverMano( )	$n^2 + 5n + 5$
crearTurnos( )	3
agregarJugador ( )	7
mostrarJugador( )	$2n + 1$
revolverMazo( )	$n^2 + 3n$
mostrarMazo( )	n
Iniciar( )	$3n^3 + 10n^2 + 13n + 31$

El orden de complejidad de este codigo es O(n) = n^3

Como se puede ver en los tiempos de ejecución de las funciones en el código, el gasto no es alto, debido a que la mayoría de las funciones cumplen roles bastante simples pero a la vez efectivos. La función *iniciar()* es la cual está encargada de llamar a las otras funciones por lo que es la función que tiene un mayor tiempo de ejecución.

Dentro de lo trabajado se logró una correcta implementación del seguimiento por pantalla escribiendo el usuario JANICE y las listas, tanto las circular como la lista enlazada y a la vez un uso adecuado de los struct para poder ordenar la información. La mayor complejidad se torno por temas de tiempo en la verificación de quién perdió el juego y la eliminación de los jugadores de la partida una vez que se quedaron sin cartas en la mano.

Referente a las mejoras que pudo haber tenido el código, existen funciones las cuales se pudieron implementar de manera más genérica y que se hubiesen utilizado en ambas listas trabajadas.

CAPÍTULO 4. CONCLUSIONES

Durante el desarrollo de este laboratorio se pudo enfrentar a las diversas dificultades que tiene trabajar con estructuras un poco más complejas de lo anterior a desarrollar, tal como lo fue la implementación de las funciones de lista, las cuales estábamos acostumbrados a utilizar sin entender el trasfondo de cómo se generaba el algoritmo y el manejo de punteros a mayor escala. Pese a que no se desarrolló del todo el objetivo del laboratorio si se logró implementar todas las funciones respecto a las listas y la implementación de estas en la función principal, y a la vez se pudo obtener un aprendizaje y compresión a mayor profundidad respecto a todo el desarrollo de listas, las maneras de abordar estos problemas definiendo estructuras y el manejo de punteros en estructuras más complejas.

CAPÍTULO 5. REFERENCIAS

Culo Sucio. (2017, 16) de abril. Wikipedia, La enciclopedia libre. Fecha de consulta: 22:41, septiembre 28, 2017 from https://es.wikipedia.org/wiki/Culo_Sucio