Tideman’s Ranked Pairs

# Introducción

El objetivo de esta práctica es que utilices una representación de grafos mediante matrices de adyacencia para resolver el problema de elecciones mediante el método de ranked pairs desarrollado por Nicolaus Tideman (<https://en.wikipedia.org/wiki/Ranked_pairs>).

En este sistema electoral, los votantes pueden votar por varios candidatos. En lugar de solo votar por su candidato preferido, ellos pueden asignarles rangos de preferencia, es decir, elegir a quién prefieren de primero, luego una segunda opción en caso de que el primero fuese descalificado y así sucesivamente (imagina la forma en que se eligen carreras en la OPSU).

Supongamos que tenemos los siguientes candidatos: Harry, Ronald y Hermione. Las tarjetas de votación podrían quedar de la siguiente manera:



¿Quién ganaría estas elecciones? En un sistema en el que se tome en cuenta solo la primera opción de los candidatos, ganaría Harry, pues de los nueve votantes, solo dos pusieron a Hermione como primera opción, solo tres pusieron a Ronald como su primera opción, pero Harry fue preferido de primero por cuatro votantes. Sin embargo, se podría argumentar que Hermione debería ser la ganadora dado que, de los nueve votantes, cinco de ellos prefieren a Hermione sobre Harry.

En este sistema Electoral, Hermione sería la ganadora dado que si el enfrentamiento fuese solo Hermione vs Ronald y Hermione vs Harry, ella siempre es preferida sobre los otros dos. El método de ranked pairs, garantiza este tipo de enfrentamiento.

En líneas generales, este método trabaja mediante la construcción de un grafo dirigido de candidatos donde un arco que va desde el candidato A hacia el candidato B indica que el A gana contra B en un enfrentamiento 1 a 1.

El grafo resultante para las tarjetas de votación mostradas previamente, se vería como sigue:



La flecha entre Hermione y Ronald indica que más votantes prefirieron a Hermione que a Ronald. La que va de Hermione a Harry indica que más votantes prefieren a Hermione sobre Harry y la que va desde Harry a Ronald indica que más votantes prefieren a Harry sobre Ronald.

Hay ocaciones en las que podría no haber manera de elegir quien gane las elecciones. Considera las siguientes tarjetas de votación:



Cómo puedes notar, siete de los votantes prefieren a Hermione sobre Ronald, seis de los votantes prefieren a Harry sobre Hermione y 5 de los votantes prefieren a Ronald sobre Harry.

Estas tarjetas nos darían el siguiente grafo:



Para manejar esto, el algoritmo de Tideman debe ser cuidadoso de crear ciclos en el grafo de candidatos. ¿Cómo lo hace? El algoritmo agrega los arcos más fuertes primero dado que estos sean, probablemente, los más significativos. En particular, este algoritmo especifica que los arcos de enfrentamiento deberían ser agregados al grafo, uno a la vez, basado en la “fuerza” de la victoria (la victoria es más fuerte mientras más votantes prefieran a un candidato sobre su oponente). Si el arco puede ser agregado sin crear un ciclo, entonces lo agrega; de lo contrario, lo ignora.

Una vez que se tiene un grafo sin ciclos, quien gana las elecciones estará dado por aquel nodo que sea fuente, es decir, aquel cuyo grado de entrada sea cero.

# Distribución del proyecto

Se te provee un paquete de proyecto con una estructura básica, algunos códigos ya resueltos y otros que debes resolver.

El proyecto está estructurado de la siguiente manera:

**lib/include/IOUtils.hpp y lib/src/IOUtils.cpp**: Contienen la implementación de operadores para imprimir diversas estructuras de datos que manejaremos en el proyecto, también encontrarás una macro LOG que sirve para imprimir en pantalla cualquier mensaje con el que se quiera hacer seguimiento del comportamiento. Estos mensajes solo se verán si al compilar se hace con la bandera -DDEBUG. La forma general de usarla es similar a printf, es decir, el primer parámetro es una cadena formateada y luego vendrían las variables requeridas. También verás una macro OUT\_PARAM, esta macro no tiene ninguna funcionalidad específica, solo se usa para marcar parámetros de funciones que sean exclusivamente de salida, es algo solamente informativo a nivel de código fuente. Estudia estos archivos, puedes agregar cualquier cosa que requieras pero no modifiques lo existente.

**lib/include/TestUtils.hpp y lib/src/TestUtils.cpp**: Contienen algunas operaciones básicas para hacer pruebas unitarias de nuestras soluciones. Estudialos, puedes agregar cualquier cosa que requieras pero no modifiques lo existente.

**lib/include/NoWinnerException.hpp y lib/src/NoWinnerException.cpp:** Contienen la implementación de una excepción para ser arrojada cuando no sea posible determinar un ganador en las elecciones.

**lib/include/NoImplementedException.hpp y lib/src/NoImplementedException.cpp:** Contienen la implementación de una excepción para ser arrojada cuando una operación no ha sido implementada. Por defecto, los métodos que debes resolver, arrojan esta excepción. Elimina esa línea de código de tus métodos (y la inclusión de la cabecera) y retorna un resultado adecuado para que se pueda ejecutar la evaluación.

**lib/include/Tideman.hpp y lib/src/Tideman.cpp:** Contienen la implementación de la clase Tideman cuyos métodos debes resolver.

**resources**: Contiene un directorio llamado inputs con un ejemplo archivo de entrada de candidatos y votaciones. Este ejemplo es el primero que se muestra en este documento. Estudia este archivo, la primera línea contiene la cantidad de candidatos, luego en las siguientes líneas están los nombres de los candidatos, luego tiene la cantidad de votantes, luego siguen, por cada votante, tres líneas con los nombres de los candidatos en el orden de la preferencia del votante. En este ejemplo verás que son 9 votantes y luego siguen 27 líneas con nombres de los candidatos. También contiene el documento outputs con la salida esperada de esta votación.

**main/src:** Contiene un programa llamado Tideman.cpp que pide los datos para una elección y escribe el ganador. Esto lo puedes ejecutar y luego introducir los datos en el orden esperado, sin embargo, no muestra mensajes en pantalla diciendo los datos que requiere, así que es preferible que le inyectes un archivo con los datos de la siguiente manera: **./main/bin/Tideman < input\_file.**

**test/src:** Contiene un programa con pruebas unitarias para la clase Tideman.

**Makefile**: Contiene reglas de compilación para todo el proyecto: make main compilara todos los programas en main/src y generar los ejecutables en main/bin. make test hace algo similar con todo lo que está en test/src y make all compila todos los main y todos los test.

# Práctica

Debes concentrarte en el archivo **lib/src/Tideman.cpp.**

La clase Tideman tiene dos atributos, una lista de candidatos y una matriz de preferencias. Se te provee un constructor (ya resuelto) que inicializa la lista de candidatos con la que se pasa como parámetro e inicializa la matriz de preferencias para que sea de N x N (donde N es la cantidad de candidatos) llena de ceros, en esta matriz las cada fila y cada columna representan a un candidato, por ejemplo, la fila cero representa el primer candidato que se encuentra en la lista de candidatos. También se te proveen getters para ambos atributos.

Debes concentrarte en resolver los siguientes métodos:

**vote:** Este método recibe como parámetros el índice de preferencia, es un entero que va desde 0 hasta n - 1, donde n es la cantidad de candidatos. El valor 0 indica que es el primer candidato de preferencia, 1 el segundo candidato y así sucesivamente. Como segundo parámetro recibe el nombre del candidato por el que se desea votar, el tercer parámetro (de salida, nota el uso de OUT\_PARAM) es la lista que indica la preferencia de candidatos, tu trabajo en la posición rank (índice de preferencia) de la lista, la posición del candidato en la lista de candidatos que tiene la clase como atributo. Para esto, debes hacer una búsqueda del nombre pasado como parámetro. La función debe retornar true si el voto se pudo aplicar y false en caso de que el nombre del candidato no exista en la lista.

**record\_preferences:** Este método recibe como parámetros la lista completa de las preferencias de un votante y debe incrementar los valores de la matriz de cada celda (i, j) donde el candidato i es preferido sobre el candidato j. Supongamos que tenemos 3 candidatos y se nos da la lista de preferencias [1, 2, 0], esto quiere decir que la primera opción del votante fue el candidato 1, la segunda opción el candidato 2 y la tercera opción el candidato 0. Ese orden nos dice que el candidato 1 es preferido sobre el candidato 2, entonces se debe incrementar (en una unidad) el valor que se encuentra en preferences[1][2]. Luego, bajo el mismo razonamiento, debería incrementarse el valor que se encuentra en preferences[1][0] y, finalmente, el valor que se encuentra en preferences[2][0].

**create\_pairs:** Este método debe crear y retornar una lista de pares (first, second) que indicará que el candidato first es preferido sobre el candidato second. Si hay empate entre ambos candidatos, entonces no se agrega a la lista. Cuidado con agregar valores repetidos. Para esto podrías considerar mirar, por cada índice de candidato, todos los mayores a este y si el candidato i es preferido sobre j, entonces agregar el par (i, j) si el j es preferido sobre el i, entonces agregas el par (j, i). Para comparar, podrías evaluar y comparar los valores en la matriz en [i][j] y [j][i]. La lista que retornes deberá estar ordenada de forma decreciente por preferencia de los pares (first, second) dados en la matriz de preferencias.

**lock\_pairs:** este método es un método de clase, no de instancia (nota el modificador static en su declaración). Recibe la lista de pares construída en el método anterior y la cantidad total de candidatos. Debe retornar la matriz de adyacencia que representa el grafo de la votación. Este grafo no debe contener ciclos. La matriz resultantes es de booleanos, un valor true en la celda (i, j) indica que hay un arco que va desde i hasta j, es decir, el candidato i le gana al candidato j (nota que esta matriz debe ser de N x N donde N es la cantidad de candidatos). La forma sencilla de resolver esto es, inicializar la matriz llena de valores false, luego recorrer la lista de pares y, para cada par (i, j), si al agregar un arco allí no se genera ciclo, entonces marcar en true la posición (i, j) de la matriz resultante, de lo contrario, dejar como estaba. Hacer una función que retorne true si hay un ciclo a partir de (i, j) podrá ayudarte, esta función podrías agregarla en el mismo archivo Tideman.cpp sin problemas.

**get\_winner:** Este método recibe como parámetro la matriz de adyacencia construída en el método anterior y debe buscar un nodo fuente, es decir, uno cuyo grado de entrada sea 0 (recuerda que un valor true en la celda (i, j) indica que hay un arco que va desde i hasta j), una vez encontrado ese nodo, debe retornar el nombre del candidato ganador. En caso de que no haya ningún nodo fuente, entonces se deberá arrojar la excepción NoWinnerException.

# Evaluación

Esto lo puedes resolver en pareja (elige bien a quien te acompaña).

Tienes todo el semestre para entregar este proyecto y podrás hacer un envío una vez por semana. Si envías y fallas, entonces podrás enviar de nuevo la siguiente semana y así sucesivamente hasta terminar el semestre.

Para poder evaluarte, debes enviar el archivo Tideman.cpp, al correo [alejandro.j.mujic4@gmail.com](mailto:alejandro.j.mujic4@gmail.com) con el subject: [AYDA]-02-Tideman. El correo deberá tener los nombres y números de identificación de la pareja participante.