Contenido

[FUNCIONALIDAD DEL CÓDIGO ADJUNTO 2](#_Toc41993819)

[ENTRADA 3](#_Toc41993820)

[PROCESO 4](#_Toc41993821)

[SALIDA 6](#_Toc41993822)

[INCONSISTENCIAS Y RECOMENDACIONES 7](#_Toc41993823)

[SOBRE EL ENUNCIADO 7](#_Toc41993824)

[SOBRE EL CÓDIGO 8](#_Toc41993825)

[SOBRE EL REPORTE 9](#_Toc41993826)

[CASOS DE PRUEBA 10](#_Toc41993827)

# FUNCIONALIDAD DEL CÓDIGO ADJUNTO

Función principal: Asignar una puntuación a cada candidato dentro de una base de datos, cuya información fue extraída del perfil de Linkedin de cada candidato. Los candidatos son personas que ocupan cargos relevantes en una empresa que puede ser un cliente potencial. Para extraer el detalle del funcionamiento del código, se ejecutaron los siguientes pasos:

* Inicialmente se ejecutó el código para validar su funcionamiento
* Se inspeccionó el archivo Program.cs
* Se simularon errores para comprender casos de prueba que apliquen a esta aplicación y funcionamiento actual
* Se inspeccionaron los archivos Profile.cs y ScoringRun.cs para comprender su funcionamiento dentro de Program.cs
* Se inspeccionó el archivo ScoringFunction.cs, que posee las funciones encargadas de puntuar cada candidato
* Durante la inspección de cada programa, se realizaron algunas consultas de segmentos de código específicos para detectar inconsistencias, y se ejecutaron de forma aislada para comprender mejor el funcionamiento y su posible solución. Ejemplo relacionado a inconsistencia 4 sobre el código: <https://ideone.com/AYyXrs>
* Con algunas inconsistencias ya detectadas y un mejor entendimiento del programa, se analizan los resultados recibidos después de ejecutar y se detectan inconsistencias en la puntuación de los candidatos y en un par de apellidos. Se usó Excel como herramienta de apoyo para filtrar datos y comparar salidas con entradas.

## ENTRADA

Detalle extraído a partir de la base de datos y del archivo Program.cs:

* La base de datos debe ser un archivo de extensión .txt.
* La información de un candidato debe estar separada por pipes, ejemplo:

url| nombre | apellido | título

* La información entre candidatos está separada por saltos de línea (cada fila es un candidato)
* El primer dato (url) no debe tener un pipe al principio, y el último dato (país) no debe tener un pipe al final
* La estructura de la información debe cumplir con el siguiente orden:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **url** | **nombre** | **apellido** | **título** | **área geográfica** | **# de recomendaciones** | **# de conexiones** | **rol actual** | **industria** | **país** |

Usando pipes, se vería así:

url| nombre | apellido | título |área geográfica|número de recomendaciones|número de conexiones|rol actual|industria|país

* Existen candidatos de Latinoamérica, por lo tanto sus nombres y urls pueden contener tildes y deberían estar permitidas (se recomienda manejar encoding utf-8 en la base de datos)
* El número de recomendaciones y el número de conexiones son los únicos datos numéricos de la tabla. Adicionalmente, deberían ser enteros
* La base de datos no debe tener encabezados. Los mostrados anteriormente son meramente ejemplos para dar más claridad

## PROCESO

El proceso de ranking de Baires inicia con la lectura del archivo .txt con la información de los candidatos. Si ninguna ruta fue ingresada al programa, se imprime el mensaje "Please add the path for the LinkedIn data file".

A continuación se lee la información utilizando el encoding ASCII y se almacena todo dentro de un vector de cadenas (string[]) llamado data. En caso de que el archivo no exista, se atrapa una excepción (catch) de tipo FileNotFoundException y se imprime el mensaje “Couldn't find file:”, seguido de la ruta del archivo que no se pudo ubicar.

Con la información leída, inicia el proceso de cálculo del puntaje de los candidatos. Utilizando un ciclo for, cada candidato es analizado uno por uno y se hace el cálculo de su respectivo puntaje. El primer paso es dividir su información, que actualmente debe ser un texto con los datos separados por pipes, ejemplo: “arturo | perez | teleport engineering manager |”. Haciendo uso de los pipes, se separa cada dato y se almacena en un vector llamado elements. De este vector se extraen los datos, usando índices entre 0 y 9 y se hace el cálculo del puntaje de cada candidato.

Actualmente el puntaje que puede recibir un candidato es entre 0 y 100. Los puntos se reparten de la siguiente manera:

|  |  |
| --- | --- |
| **Info. Del Perfil** | **Puntaje** |
| Industria | 30 |
| Rol | 25 |
| Ubicación Geográfica | 15 |
| Conexiones | 15 |
| Título | 10 |
| Recomendaciones | 5 |

Si la industria aparece dentro del listado contra el que se compara, entonces dependiendo de la industria es que se genera el puntaje para este ítem. El puntaje obtenido puede ser 0, 7.5, 15, 22.5 o 30, siendo 30 el puntaje máximo cuando el candidato está en industrias como internet, computer games, computer software, y puede bajar hasta 7.5 (mínimo) para industrias como sports, retail, transportation/trucking/railroad. Si la industria del candidato no aparece en el listado, entonces el valor será 0.

Respecto al rol del candidato, también se compara contra un listado. En caso de tener varios roles, igual se compara con el listado en busca de cuales roles hacen match, incrementando así el puntaje del candidato. Al final la operación que determina el puntaje del rol consiste en multiplicar 25\*[suma de roles que hicieron match]. Si la suma de roles que hicieron match es mayor a 1, entonces se redondea a 1, para que 25 sea el puntaje más alto. Sin ningún rol del candidato hizo match, entonces será 0.

El puntaje por ubicación geográfica tiene dos entradas: la ubicación geográfica del candidato y su país. Si el país del candidato hace match con alguno de los países listados en el programa, esto incrementa el puntaje de este ítem. El valor de esta parte está entre 0.25 y 0.75 si se hace match, de lo contrario será 0. Esta misma dinámica se aplica para la ubicación geográfica. Si se hace match con alguna de las ubicaciones dentro del listado del programa, entonces el valor de esta parte será 0.25, de lo contrario será 0. Al final, se suma el valor obtenido al analizar el país y el valor obtenido al analizar la ubicación geográfica, donde el resultado es máximo 1 y mínimo 0.

El puntaje por conexiones se obtiene al dividir la cantidad de conexiones del candidato por 500. El valor obtenido se multiplica por 15. El valor obtenido estaría entre 1 (teniendo 500 conexiones) y 0 (teniendo 0 conexiones).

El puntaje por título se calcula exactamente igual al puntaje por rol. Incluso se usa el mismo listado.

Finalmente, el puntaje por recomendaciones se calcula de manera parecida al de conexiones. En este caso, la cantidad de recomendaciones del candidato se divide por 50. El valor obtenido se multiplica por 5. Antes de multiplicar por 5, si el valor obtenido es mayor a 1, entonces se redondea a 1, de lo contrario, se utiliza el valor obtenido (que estaría entre 1 y 0) en la multiplicación con el 5.

Por cada candidato, también se obtiene la longitud de su nombre y apellido, y se comparan con las longitudes del nombre y apellido del candidato anterior. Esto es con el propósito de indexar correctamente la tabla de resultados.

## SALIDA

La tabla de resultados presenta el score, nombre, apellido y url del perfil de linkedin de los candidatos puntuados. La tabla se presenta en orden descendente según el score. Los datos de salida se separan por pipes, por ejemplo:

Score | Name | LastName | ProfileUrl

Los espacios entre Name y LastName son generados por los nombres y apellidos más largos detectados en los candidatos evaluados.

# INCONSISTENCIAS Y RECOMENDACIONES

## SOBRE EL ENUNCIADO

1. **Inconsistencia**: La base de datos no es xlsx como dice el enunciado.

**Recomendación**: Especificar que se entregará un archivo txt

1. **Inconsistencia**: El rango de la puntuación de un candidato de acuerdo al enunciado es entre -1 y 40

**Recomendación**: Especificar que en realidad es entre 0 y 100

1. **Inconsistencia:** La muestra no fue de 10000 personas, fue de 300

**Recomendación:** Especificar que el tamaño es realmente de 300 personas

Si las inconsistencias 1 y 2 realmente son el diseño que se esperaba por parte del programa, entonces estas inconsistencias no aplican para el enunciado, realmente deberían estar en la siguiente sección (Sobre el código), ya que aunque el programa funciona, no cumple con algunos requisitos de diseño.

## SOBRE EL CÓDIGO

1. **Inconsistencia**: Archivo Program.cs, línea 24: Se utiliza encoding ASCII, el cual transforma las tildes y eñes en símbolos extraños. Algunas consecuencias de esto son:

* Afectar el scoring
* La tabla impresa con el scoring de los candidatos, aparece con la información dañada en alguno de ellos
* Si hay tildes en algún link, entonces el link se romperá y tocaría repararlo manualmente o buscar el candidato en la base de datos inicial para encontrar la url original

**Recomendación**: Utilizar encoding que soporte tildes y eñes como el encoding UTF-8, o que la base de datos no llegue con eñes y tildes (esto puede afectar a los links de los perfiles).

1. **Inconsistencia:** Archivo Program.cs, líneas 22-30: Se utiliza un bloque try-catch para un proceso que se podría reducir a un if-else.

**Recomendación**: Determinar si a nivel de performance, es más viable para este desarrollo usar un bloque if-else o continuar con el try-catch.

1. **Inconsistencia**: Archivo Program.cs, línea 38: Se crea un objeto de tipo int, cuyo único nombre es un guion bajo (int \_).

**Recomendación**: Revisar el estándar de nombramientos a nivel de desarrollo para determinar si es posible asignar un nombre más ideal.

1. **Inconsistencia**: Archivo Program.cs, líneas 46-47. Observando este segmento de la línea 46:

*int.TryParse(elements[5], out \_) ? int.Parse(elements[5]) : 0*

La función *TryParse* entrega un booleano, el cual es verdadero (true) cuando el valor dentro de elements[5] es un texto que contiene un número entero, de lo contrario es falso (false). Cuando TryParse entrega un verdadero, se ejecuta la función Parse y cuando entrega un falso, el resultado de la línea presentada es 0.

**Recomendación**: La ejecución del segmento *int.Parse(elements[5])* se puede reemplazar por el objeto \_, ya que este posee el valor numérico ya convertido por parte de la función *TryParse*, ahorrando así la doble conversión que actualmente sucede.

1. **Inconsistencia**: Archivo ScoringFunction.cs, líneas 25 y 39. En estas líneas es donde se calcula el puntaje por rol y por título. Sin embargo, considerando que se usa la función *Contains* en el segmento de código *normalized.Contains(key),* se puede generar incrementos de puntaje en ciertos cargos donde no se debería. Por ejemplo, si un candidato tiene dentro de su rol la palabra “director”, esto va a generar un incremento porque esta palabra (que se convertiría en el objeto *normalized*) posee dos llaves del listado de títulos: director y cto.

**Recomendación**: Refactorizar la lógica de estas operaciones para que el cálculo del puntaje sea confiable y **ejecutar pruebas unitarias**.

## SOBRE EL REPORTE

1. **Inconsistencia**: Como se mencionó en la inconsistencia 5 sobre el código, la lógica actual del programa afecta el puntaje de varios candidatos, específicamente en el puntaje recibido por título y por rol. La siguiente tabla muestra para que candidatos se debería revisar el puntaje:

|  |  |  |  |
| --- | --- | --- | --- |
| **Título o Rol del candidato** | **Título duplicado por el programa actualmente** | **Multiplicador de puntaje esperado** | **Multiplicador de puntaje recibido** |
| vice-president | president | 0.5 | 1 |
| co founder | founder | 0.75 | 1 |
| cofounder | founder | 0.75 | 1 |
| co-founder | founder | 0.75 | 1 |
| director | cto | 0.25 | 1 |

Un claro ejemplo es el candidato de nombre David y apellido rubal cissp. Su puntaje fue 99.1 y fue el candidato top de toda la muestra analizada.

Su título es “chief technologist, federal it visionary and technical director”. La única palabra que hace cruce con la lista del programa es “director”. El puntaje para el título de este candidato debería ser 0.25\*10=2.5, sin embargo, el multiplicador está siendo retornado como 1, recibiendo 7.5 puntos extra.

El análisis presentado anteriormente, puede aplicarse también para buscar y encontrar inconsistencias en el puntaje recibido por el rol del candidato.

**Recomendación:** Corregir la inconsistencia 5 sobre el código. Después de ejecutar pruebas unitarias, se recomienda agregar candidatos con títulos y roles iguales a los mencionados anteriormente en el caso de prueba 1 de la siguiente sección, así se puede validar con muestras reales la corrección de esta inconsistencia

1. **Inconsistencia**: Hay candidatos que tienen tildes en sus apellidos. Debido a la inconsistencia 1 sobre el código, estas tildes se pierden y los apellidos se transforman de la siguiente manera:

|  |  |
| --- | --- |
| **Apellido original** | **Apellido en el reporte** |
| gonzález pmp itil | gonz??lez pmp itil |
| castro feijóo | castro feij??o |

**Recomendación:** Corregir la inconsistencia 1 sobre el código e incluir en el caso de prueba 1, candidatos con tildes en sus nombres y apellidos para validar que se procese correctamente la data.

# CASOS DE PRUEBA

Los siguientes casos de prueba se diseñaron en base al conocimiento adquirido del enunciado y de la exploración del código:

|  |  |  |
| --- | --- | --- |
| **1. Calculo de score de muestra de candidatos (entre 5 y 10 registros)** | | |
| **Entradas** | **Proceso** | **Salidas** |
| 1. Base de datos con 5 a 10 registros válidos de candidatos 2. Uno de los candidatos debe tener información que le permita obtener un puntaje igual a 100 3. Uno de los candidatos debe tener información que le permita obtener un puntaje igual a 0 | 1. Se ejecutará el programa con la base de datos descrita 2. Se imprimirá en la consola el scoring, nombre, apellido y url de linkedin de cada candidato | 1. Tabla en consola correctamente indexada 2. Candidatos ordenados en el orden esperado (descendente según el scoring) 3. Scoring calculado correctamente para cada candidato. 4. Se deben imprimir primero los encabezados:   Score | Name | LastName | ProfileUrl |

|  |  |  |
| --- | --- | --- |
| **2. Ejecución del programa sin ingresar la ruta de un archivo** | | |
| **Entradas** | **Proceso** | **Salidas** |
| No aplica | Se ejecutará el programa sin ingresarle la ruta de un archivo | El programa debe imprimir el mensaje “Please add the path for the LinkedIn data file” y frenar la ejecución |

|  |  |  |
| --- | --- | --- |
| **3. Ejecución de programa ingresando una ruta hacia una carpeta** | | |
| **Entradas** | **Proceso** | **Salidas** |
| Ruta que apunte hacia una carpeta, ejemplo: C:/Users/Usuario/Documents/ | Se ejecutará el programa ingresando una ruta que apunte hacia una carpeta | El programa debería imprimir un mensaje que permita al usuario corregir el dato ingresado y frenar la ejecución |

|  |  |  |
| --- | --- | --- |
| **4. Ejecución de programa ingresando un archivo inexistente** | | |
| **Entradas** | **Proceso** | **Salidas** |
| Ruta que apunte hacia un archivo que no exista, o hacia una base de datos sin extensión, ejemplo:  C:/Users/Usuario/Documents/PeopleToRate | Se ejecutará el programa ingresando una ruta que cumpla con el criterio de entrada | El programa debe imprimir el mensaje “Couldn't find file:”, seguido de la ruta del archivo que intentó encontrar entre comillas simples y frenar la ejecución. Ejemplo:  Couldn't find file: 'C:\Users\Usuario\Documents\PeopleToRate' |

|  |  |  |
| --- | --- | --- |
| **5. Ejecución de programa ingresando una base de datos vacía** | | |
| **Entradas** | **Proceso** | **Salidas** |
| Ruta que apunte hacia una base de datos real, pero que esté vacía | Se ejecutará el programa con la base de datos descrita | Sólo se debería imprimir los encabezados de la tabla de resultados  *Revisar con el cliente si es más conveniente frenar la ejecución e imprimir un mensaje* |

Los anteriores serían los casos de prueba principales. En el caso de prueba 1 se pueden cubrir varios aspectos del programa, al igual que se puede validar la corrección de las inconsistencias, sin embargo, se **recomienda** la ejecución de **pruebas unitarias** para validar el funcionamiento de los métodos de la aplicación. Una técnica útil para generar una muestra ideal de data de prueba para el caso de prueba 1, y para las pruebas unitarias sería pairwise.