IC-2001 Estructuras - Lab 1

Estructuras de datos
II-Semestre 2025
Prof. Eddy Ramírez

3 de septiembre de 2025

Para la elaboración del ejercicio L11 del laboratorio #1 de estructuras de datos se realizaron varias implementaciones de una cola de prioridad. Una se realizó con una *Single Linked List* (SLL) y las demás se realizaron con una *Skipped List* pero con diferentes probabilidades de que un nodo ascienda a un nivel superior.

1. Entorno de las pruebas

Las pruebas se realizaron en una máquina virtual que tiene las siguientes características:

- CPU
  - Modelo: Intel Core i5-1035G1 (x64)
  - Núcleos: 3
  - Reloj: 1.19 GHz
- RAM:
  - Memoria: 8 GB
  - Tipo: SO-DIMM
- SO:
  - Edición: Ubuntu
  - Versión: 20.04.6 LTS

2. Generación de los casos de prueba

Los casos de prueba fueron generados utilizando el siguiente código.

```
int main() {
    srand(time(NULL));

    int numLineas = 10000;
    int probabilidadVacunas = 2;

    for (int i = 0; i++ < numLineas;) {
        if (!(rand() % probabilidadVacunas)) {
            cout << "V" << endl;
            continue;
        }

        cout << GenNombreAleatorio() << " " << GenPrioridadAleatoria() << endl;
    }
}
```

Donde:

- `GetNombreAleatorio()` retorna un `string` de 20 caracteres en letras minúsculas del abecedario inglés aleatorios.
- La función `GetPrioridadAleatoria()` retorna una numero entero aleatorio en el intervalo $[0, 10^4[$.
- Y la variable `probabilidadVacunas` puede ser modificada para determinar la frecuencia de vacunas con respecto a las personas.

El siguiente es un ejemplo de la salida del programa con 8 lineas y 50 % de vacunas:

```
ljnqtdtcbxygspsabost 1635
lthrhijxcuhpodtifwjg 3955
                     V
expwtclfvswddhdgekvu 4738
                     V
euopcutyuxymxtojbmey 7393
                     V
                     V
```

3. Implementaciones de cola de prioridad

Las colas de prioridad se implementaron de las siguientes maneras[1]:

1. SLL (Single Linked List)
   - Eliminación: $O(1)$
   - Inserción Esperada $O(n)$
2. Skipped List $P = \frac{1}{4}$
   - Eliminación: $O(1)$
   - Inserción Esperada $O(log(n))$
3. Skipped List $P = \frac{1}{2}$
   - Eliminación: $O(1)$
   - Inserción Esperada $O(log(n))$
4. Skipped List $P = \frac{3}{4}$
   - Eliminación: $O(1)$
   - Inserción Esperada $O(log(n))$

[1] Aunque las tres *skipped list* tienen la misma complejidad temporal a la hora de insertar elementos, también se desea analizar sus diferencias en el uso eficiente de memoria.

4. Metodología

Para obtener los datos deseados se generaron diferentes archivos de prueba que combinaban dos factores:

- **Cantidad de lineas:** archivos con 10,000, 5,000 y 2,500 lineas.
- **Frecuencia de vacunas:** se realizaron pruebas con 50 % y 25 % de posibilidad entre ingresar una persona o sacar una persona de la lista (que haya una vacuna disponible).

Para cada variante se generaron 30 archivos de casos de prueba diferentes. Además, cada archivo se ejecutó 100 veces[2] para obtener un mejor promedio de mediciones.

Para obtener un aproximado del espacio en memoria que ocupó cada implementación se llevó la cuenta de nodos insertados/eliminados y se obtuvo el máximo de esa cuenta. Finalmente, se calculó el promedio de tamaño ocupado en todos los casos.

5. Resultados de las pruebas

A continuación dos tablas con los tiempos de ejecución y espacio en memoria respectivamente. Donde L se refiere a "cantidad de lineas" y V se a "frecuencia de vacunas".

5.1. Tiempos de ejecucion promedio (ms)

Parametros	**SLL**	**SKP 1/2**	**SKP 1/4**	**SKP 3/4**
L: 10,000, V: 50 %	3	5	3	9
L: 5,000, V: 50 %	40	10	5	15
L: 2,500, V: 50 %	1	0	0	1
L: 10,000, V: 25 %	40	10	6	15
L: 5,000, V: 25 %	9	3	2	7
L: 2,500, V: 25 %	2	1	1	3

5.2. Espacio promedio ocupado en memoria (KB)

Parametros	**SLL**	**SKP 1/2**	**SKP 1/4**	**SKP 3/4**
L: 10,000, V: 50 %	6,35	15,82	9,8	307,25
L: 5,000, V: 50 %	237,8	555	370,6	1112,16
L: 2,500, V: 50 %	2,72	6,2	4,79	12,73
L: 10,000, V: 25 %	239	557,16	370,3	1127
L: 5,000, V: 25 %	122,2	282,59	190,97	567,4
L: 2,500, V: 25 %	59,74	136	93,18	277,77

[2]Excepto la *Skipped List* $\frac{3}{4}$ ya que la memoria que utilizó fue excesiva, se redució a 70 veces por archivo.

6. Análisis de resultados

De los resultados se pueden concluir ciertos aspectos importantes sobre las diferentes implementaciones de una cola de prioridad.

1. Una *Skipped List* es aparentemente más eficiente, no solo en tiempo de ejecución, sino también en el uso de memoria, si la probabilidad de que un nodo ascienda a un nivel superior es de $\frac{1}{4}$ en vez de $\frac{1}{2}$.

2. Una *Skipped List* con probabilidad $\frac{3}{4}$ es terriblemente ineficiente en términos de memoria. Tampoco es excelente en términos de tiempos de ejecución

3. Reducir la cantidad de operaciones `Pop()` que realiza la lista permite apreciar mejor los tiempos de ejecución y espacio utilizado.

4. Ninguna de las dos estructuras utilizadas en este laboratorio son las indicadas para utilizar una cola de prioridad. Es preferible utilizar un heap binario o árboles balanceados (como AVL o rojinegros).