1. Implementación hashing perfecto

a) Breve descripción de las estructuras de datos implementadas.

- Tabla Hash: almacena el primer nivel de tabla hash dentro de ella hay un arreglo de buckets además de contener a, b ,m del primer nivel
- Bucket : estructura que almacena los elementos contiene un arreglo para los elementos y además contiene la informacion de la funcion de hash a, b, m del segundo nivel

b) Pseudocódigo de todas las operaciones implementadas.

• Tabla Hash

Insertar:

```
while colisiones > 4*n do
    colisiones←0;
    a←rand() % (p);
    b←rand() % (p);
    for i=0 to lenght < vector > do
        temp[(((a*ins[i]+b) %p) %n)].push_back(ins[i]);
    end for
    for i=0 to n do
        if temp[i].size()>0) then
            colisiones ← colisiones + pow(temp[i].size(),2);
        end if
    end for
end while
for i=0 to n do
    cout<<i<< " − −− >> ";
    for j=0 ; j<temp[i].size() ; j++ do
        cout<< " " <<temp[i][j];
    end for
end for
for i=0 to n do
    tam ←temp[i].size();
    if tam> 0 then
        T[i]←new bucket_(tam);
        T[i]− >hash(temp[i]);
    end if
end for
```

Buscar:

```
it=(((a*x+b) %p) %n);
if T[it] then
    if T[it]− >busca(x) then
        return true;
    else return false;
    end if
elsereturn false;
```

end if

- # Bucket

Hash

```
while colisiones >= 1 do
    colisiones←0;
    a←rand() %(p);
    b←rand() %(p);
    for i = 0 to tamv do
        dir=(((a*v[i]+b) %p) %(tamv*tamv));
        if this− >B[dir] > -1  then
            this− >limpiar_bucket();
            colisiones++;
            break;
        else this− >B[dir]=v[i];
        end if
    end for
end while
dat[0]←a1;
dat[1]←b1;
```

Limpiar bucket

```
for  i=0; i< dat[2]; i++ do
    this− >B[ i ]= -2;
end for
```

Busca

```
if B[(((dat[0]*y+dat[1]) %p) %dat[2])]== y then
    return true;
else return false;
end if
```

c) Análisis de numero esperado de colisiones para hashing perfecto. Explique por que se necesita hacer $m_i = c^2{}_i$.

Se debe cumplir que $m_i = c_i^2$ dado que la probabilidad de que $x \neq y$ colisionen es $\leq \frac{1}{m}$, entonces al elegir una tabla de tamaño $m = c_i^2$, tenemos una probabilidad de $\frac{1}{2}$ de que una función $h \epsilon H$ elegida al azar sea perfecta, como se trata de un algoritmo tipo Las Vegas para encontrar una función perfecta, el numero esperado de intentos es 2 por lo que el tiempo esperado hasta encontrar una buena función h es O(n), por lo tanto, el tiempo esperado de este algoritmo es O(n) cuando se trata de insertar o cuando hay colisión.

d) Análisis de complejidad de tiempo esperado para todas las operaciones.

- Tabla Hash
  - Insertar: $O(n^2)$
  - Imprimir: O(n)

- Bucket
  - Hash: O(n^2)
  - Limpiar Bucket: O(n)
  - Elementos: O(n)

e) Evaluación experimental de las operaciones construir la tabla y búsqueda de elementos. Esta evaluación debe incluir gráficas donde en el eje X se representa el tamaño de las tablas y en el eje Y el tiempo que toma la operación de búsquedas aleatorias. Además, se debe mostrar un gráfico donde se muestre el número promedio de veces que se necesita ejecutar la búsqueda de los coeficientes a y b para la determinar la función hash h.