Archivos Directos
Y
Resolución de colisiones

75.06 Organización de Datos
Curso 1

Este Archivo en el grupo se llama
directos_clase1.pdf

Programa

Esta clase:

- Archivos Directos
- Resolución de Colisiones
- Hashing Perfecto

Próxima clase:

- Funciones de hashing
- Hashing extensible

Archivos Directos: Características

- Registros de longitud fija (*)
- Acceso Directo
- Cantidad máxima de registros fija (*)
- No hay ordenamiento (*)
- No hay acceso secuencial (*)
- Cada registro se identifica por una clave primaria

(*) El mundo esta lleno de excepciones

Implementación Base

Datos Administrativos	
	R0
	R1
	R2
	R3
	R4
	R5
	R6

Definiciones

- Espacio de claves (K):
  - Cantidad de registros a almacenar, a veces cantidad de claves posibles.
- Espacio de direcciones (N):
  - Capacidad, en registros, del archivo directo.
- Clave Primaria
  - Conjunto de datos (campos) que permite identificar univocamente a cada registro a almacenar.

Defniciones

- Datos estáticos:
  - Datos que no sufren modificaciones. Operaciones: Carga inicial, consulta.
- Datos dinámicos:
  - Datos que pueden sufrir modificaciones. Operaciones: Alta, Baja, Modificación, Consulta

Definiciones

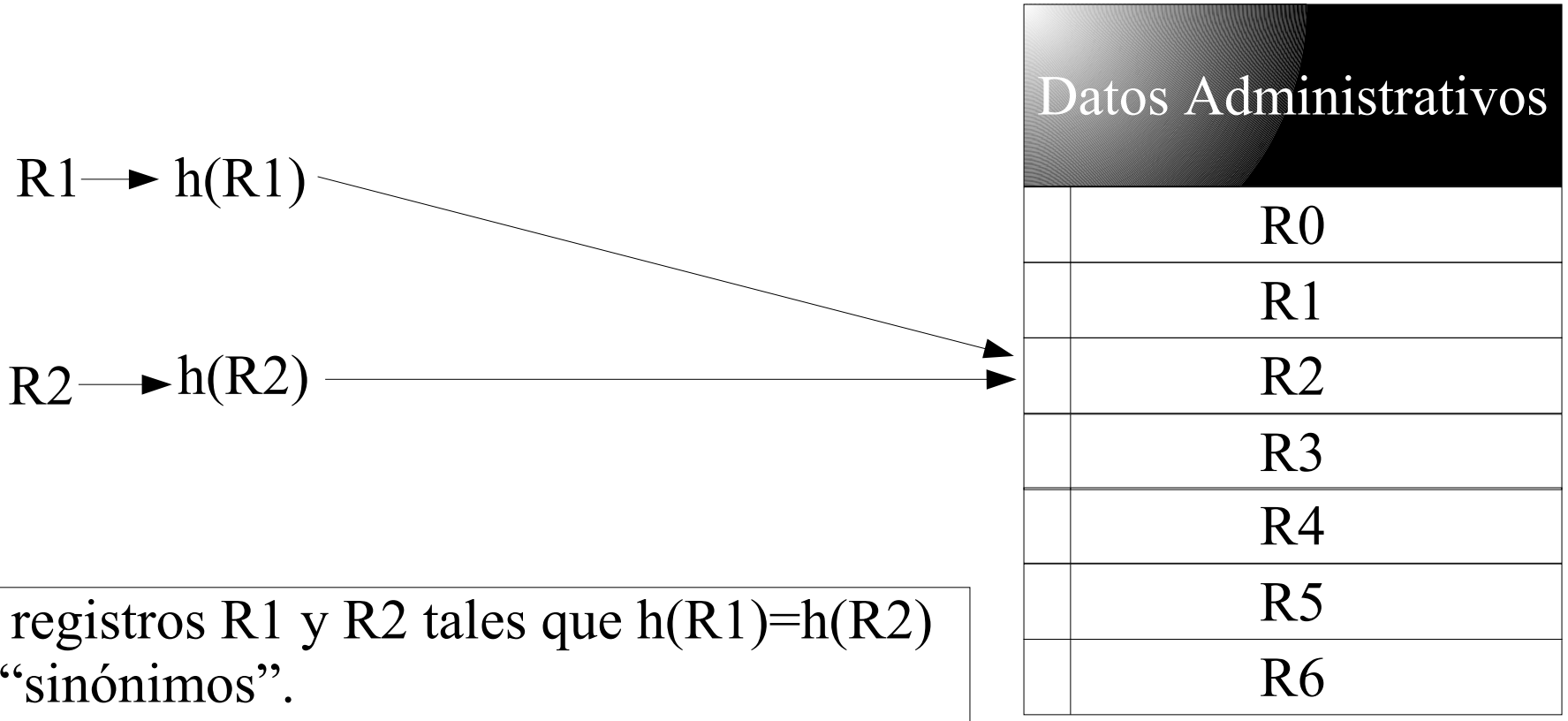
- Acceso directo:
  - Mecanismo de acceso que utiliza una función de hashing para determinar la ubicación de un registro en un archivo.
- Función de hashing:
  - Función que convierte la clave de un registro en su posición en el archivo directo.

Factor de Carga (Densidad de Carga)

- Simbolizado con el símbolo: λ

  λ = Registros Cargados / Registros libres

Sinónimos y Colisiones



Dos registros R1 y R2 tales que h(R1)=h(R2) son “sinónimos”.
Cuando se da este caso se produce una “colisión”.

Resolución de colisiones

- Existen varios métodos
  - Los métodos se pueden combinar entre si generando una amplia combinatoria de soluciones posibles.
  - Sin embargo algunas combinaciones no tienen sentido.

Direccionamiento Abierto

- En el archivo de datos no se hace distinción entre registros originales y sinónimos.
- Dicho de otra forma: No se encadenan los sinónimos.
- Cuando ocurre una colisión se prueban posiciones del archivo hasta encontrar una libre

h0(r), h1(r), ..., hn(r)

$$h_i(r) = (hash(r) + f(i)) \, mod \, N \; con \; f(0) = 0$$

Direccionamiento Abierto

$$h_i(r) = (hash(r) + f(i)) \; mod \; N \; con \; f(0) = 0$$

f(i) define la estrategia para la resolución de colisiones:

- Lineal
- Cuadrático
- Doble Hashing

Direccionamiento Abierto, Lineal

$f(i) = 1 + i$

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

En cada registro debe existir como dato administrativo una marca que indique Libre, Borrado u Ocupado.

Borrado y Libre no son la misma cosa (¿por que?)

El número de registro, posición obviamente no forma parte del archivo.

Direccionamiento Abierto, Lineal

- R1, h(R1) = 2

1	
2	**R1 (1)**
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

Direccionamiento Abierto, Lineal

- R1, h(R1) = 2
- R2, h(R2) = 3

1	
2	R1 (1)
3	**R2 (1)**
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	

Direccionamiento Abierto, Lineal

- R1, h(R1) = 2
- R2, h(R2) = 3
- R3, h(R3) = 5

1	
2	R1 (1)
3	R2 (1)
4	
5	**R3 (1)**
6	
7	
8	
9	
10	
11	
12	
13	

Direccionamiento Abierto, Lineal

- R1, h(R1) = 2
- R2, h(R2) = 3
- R3, h(R3) = 5
- R4, h(R4) = 2

1	
2	R1 (1)
3	R2 (1)
4	**R4 (3)**
5	R3 (1)
6	
7	
8	
9	
10	
11	
12	
13	

Direccionamiento Abierto, Lineal

- R1, h(R1) = 2
- R2, h(R2) = 3
- R3, h(R3) = 5
- R4, h(R4) = 2
- R5, h(R5) = 3

1	
2	R1 (1)
3	R2 (1)
4	R4 (3)
5	R3 (1)
6	**R5 (4)**
7	
8	
9	
10	
11	
12	
13	

Direccionamiento Abierto, Lineal: Operaciones

1	
2	R1 (1)
3	R2 (1)
4	R4 (3)
5	R3 (1)
6	R5 (4)
7	
8	
9	
10	
11	
12	
13	

- Búsquedas

  Buscar hasta encontrar el registro o bien uno libre (borrado=seguir)

- Altas

  Buscar el registro, al encontrar uno libre insertar.

- Bajas

  Buscar + Marcar

- Modificaciones

  Buscar + Modificar

Problemas: Clustering

- A medida que ocurren colisiones los sinónimos tienden a agruparse an áreas reducidas del archivo generando el fenómeno conocido como “clustering” o “aglomeramiento”

Accesos en promedio para una inserción (Búsqueda no exitosa)

$$Accesos = \frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)$$

Accesos en promedio para una búsqueda exitosa

$$Accesos = \frac{1}{2}\left(1 + \frac{1}{(1-\lambda)}\right)$$

Direccionamiento Abierto, Cuadrático

- Tiene como objetivo reducir el fenómeno de clustering.

$$f(i)=i^2$$

- i=1,4,9,16,25....

Direccionamiento Abierto, Cuadrático

Ejemplo H(R1) = 3, H(R2) = 3
N = 7
A partir de 3 los registros a probar son:
3 + 1 mod 7 = 4
3 + 4 mod 7 = 0
3 + 9 mod 7 = 5
3 + 16 mod 7 = 5
3 + 25 mod 7 = 0

Como puede verse no siempre se prueban todos los lugares libres del archivo, entonces, ¿podría darse que un registro no pueda insertarse aun con espacio disponible ?

Direccionamiento Abierto, Cuadrático
Teorema:

Si N es primo y $\lambda < 0,5$

Entonces si existe algún lugar libre en el archivo el método lo encuentra.Es decir que se garantiza el éxito de la inserción.

Interludio Demostrativo

Demostración

Sea N el espacio de direcciones un número primo.
Queremos demostrar que las primeros N/2 resultados son distintos.

Sean i,j tales que 0<=i,j<=N/2

Para que haya 2 resultados iguales se debe dar que:

$$h+i^2 = h+j^2 \text{ (con } i<>j)$$

$$h+i^2 = h+j^2 \pmod N$$

$$i^2 = j^2 \pmod N$$

$$i^2 - j^2 = 0 \pmod N$$

$$(i-j)(i+j) = 0 \pmod N$$

Entonces i-j o i+j son divisibles por N lo cual no puede darse pues N es un número primo.
Por lo tanto los primeros N/2 resultados son distintos y como hay a lo sumo N/2 registros en el archivo se garantiza el exito de la operación.

Direccionamiento Abierto, Cuadrático

- ¿Qué ocurre cuando $\lambda > 0.5$?
  - No se pueden garantizar las altas por lo tanto se debe "reorganizar" el archivo agregando espacio hasta que $\lambda < 0,5$.
  - Al extender el archivo se lo debe extender a un nuevo número primo.

Direccionamiento Abierto, Cuadrático: Cantidad de accesos promedio

- No se ha analizado hasta el momento (!)
- En general se soluciona el problema de clustering pero se observa que los registros que son sinonimos produciran la misma serie de lugares a testear, esto se conoce como clustering secundario y si bien no es tan grave tambien representa un problema.
- Para eliminar este fenómeno el método mas popular se conoce como “doble hashing”

Direccionamiento Abierto, Doble hashing

- El desplazamiento para la búsqueda lineal esta dado por una segunda función de hashing.

$$f(i) = h2(r)$$

- De esta forma los sinónimos no necesariamente generan secuencias iguales de posiciones a probar.

Direccionamiento Abierto, Doble Hashing

- Si el espacio de direcciones es N la función de hashing produce un número entre 0 y N-1
- La segunda funcion de hashing debe producir un número entre 0 y M-1 con M relativamente primo con respecto de N (sin divisores comunes)
- De esta forma se garantiza que se recorren todos los registros libres del archivo.

Doble Hashing, Ejemplo

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Con N = 11 cualquier M<11 sirve pues N es primo.

Direccionamiento Abierto, Doble Hashing.

- Ejemplo:
  - N = 13 (0..12)
  - h2(x) = 1 + X mod R (R primo menor que N)
  - R = 11
  - (Cuando N y N-2 son primos son “primos mellizos”)

Interludio: Primos mellizos

2	**3**	**5**	7	**11**	13	**17**	19	23	**29**
31	37	**41**	43	47	53	**59**	61	67	**71**
73	79	83	89	97	**101**	103	**107**	109	113
127	131	**137**	139	**149**	151	157	163	167	173
179	181	**191**	193	**197**	199	211	223	**227**	229
233	**239**	241	251	257	263	**269**	271	277	**281**
283	293	307	**311**	313	317	331	337	**347**	349
353	359	367	373	379	383	389	397	401	409
419	421	**431**	433	439	443	449	457	**461**	463
467	479	487	491	499	503	509	**521**	523	541
547	557	563	**569**	571	577	587	593	**599**	601
607	613	**617**	619	631	**641**	643	647	653	**659**
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	**809**
811	**821**	823	**827**	829	839	853	**857**	859	863
877	**881**	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997		

El último par de primos mellizos corresponde al:
33218925 · 2^169690-1
(5190 dígitos)
Descubierto en 2002

Conjetura: Hay infinitos primos mellizos.

Direccionamiento Abierto, Doble Hashing

- Cantidad promedio de accesos para una búsqueda:

$$\frac{-\ln(1-\lambda)}{\lambda}$$

Direccionamiento Abierto, Area de Overflow Independiente

- Puede estar dentro o fuera del mismo archivo.
- Para manejarse dentro del área de overflow se utiliza alguna de las estrategias que vimos hasta ahora (lineal, cuadrático, doble hashing).

Direccionamiento Abierto con área de overflow independiente.

Ventajas

- Ausencia de sinónimos en el area de datos se garantiza entonces que para registros sin sinónimos la cantidad de accesos es 1

Desventajas

- Limitado de acuerdo al tamaño del area de overflow
- Los sinónimos están físicamente distantes de los registros de datos (solución: distribuir el área de overflow)

Direccionamiento abierto, con área de overflow distribuida.

- Cada “n” registros de datos se utilizan “m” registros de overflow

Direccionamiento abierto, con área de overflow distribuida.

- Ejemplo n=3, m=2

0	D0
1	D1
2	D2
3	O1
4	O2
5	D3
6	D4
7	D5
8	O3
9	O4
10	D6
11	D7
12	D8
13	O5
14	O6

$$Datos(i) = (m+n)\ (i \operatorname{div} n) + (i \bmod n)$$

$$Overflow(i) = (m+n)\ (i \operatorname{div} M) + n + (i \bmod M)$$

Direccionamiento Cerrado

- Consiste en encadenar los sinónimos entre si.
- Requiere el manejo de una estructura de lista dentro del archivo directo para manipular los sinónimos.
- Reduce la cantidad de accesos promedio que deben hacerse para una búsqueda.

Direccionamiento Cerrado sin área de overflow independiente

R1=1

0	-1	
1	-1	**R1(1)**
2	-1	
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	

Direccionamiento Cerrado sin área de overflow independiente

R1=1
R2=2

0	-1	
1	-1	R1(1)
2	-1	**R2(1)**
3	-1	
4	-1	
5	-1	
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	

Direccionamiento Cerrado sin área de overflow independiente

R1=1
R2=2
R3=1

0	-1	
1	3	R1(1)
2	-1	R2(1)
3	-1	**R3(3)**
4	-1	
5	-1	
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	

Direccionamiento Cerrado sin área de overflow independiente

R1=1
R2=2
R3=1
R4=1

0	-1	
1	3	R1(1)
2	-1	R2(1)
3	4	R3(3)
4	-1	**R4(3)**
5	-1	
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	

Direccionamiento Cerrado sin área de overflow independiente

R1=1
R2=2
R3=1
R4=1
R5=3

0	-1	
1	3	R1(1)
2	-1	R2(1)
3	4	R3(3)
4	-1	R4(3)
5	-1	**R5(1)**
6	-1	
7	-1	
8	-1	
9	-1	
10	-1	

Observemos que antes de encadenar se debe verificar que los registros realmente sean sinónimos.
Las búsquedas iniciales degradan en un método lineal y solo se mejora el manejo de sinónimos.
Este método tiene mas sentido cuando los datos son estáticos aplicando “doble carga”

Operaciones

- Búsqueda:
  - Buscar a partir de la posición hasta encontrar un registro libre (no borrado) o bien un sinónimo.
- Alta:
  - Buscar a partir de la posición hasta encontrar un registro libre o un sinónimo. (Agregar o encadenar)
- Bajas y modificaciones:
  - Buscar + dar de baja o modificar

Direccionamiento cerrado con área de overflow independiente

- En general cuando se usa direccionamiento cerrado si los datos son dinámicos se usa area de overflow independiente.
- Cantidad promedio de accesos para las búsquedas: $1 + \lambda/2$
  - Mejor que los métodos abiertos
  - Pero las operaciones de alta, baja y modificación son mas costosas

Direccionamiento Cerrado con area de overflow independiente: operaciones

0	-1		R1=1
1	-1	**R1(1)**	
2	-1		
3	-1		
4	-1		
5	-1		
6	-1		
7	-1		
8	-1		
9	-1		
10	-1		
11			
12			
13			
14			
15			

Direccionamiento Cerrado con area de overflow independiente: operaciones

0	-1		R1=1
1	-1	R1(1)	R2=2
2	-1	**R2(1)**	
3	-1		
4	-1		
5	-1		
6	-1		
7	-1		
8	-1		
9	-1		
10	-1		
11			
12			
13			
14			
15			

Direccionamiento Cerrado con area de overflow independiente: operaciones

0	-1		R1=1
1	11	R1(1)	R2=2
2	-1	R2(1)	R3=1
3	-1		
4	-1		
5	-1		
6	-1		
7	-1		
8	-1		
9	-1		
10	-1		
11	-1	**R3(2)**	
12	-1		
13	-1		
14	-1		
15	-1		

Direccionamiento Cerrado con area de overflow independiente: operaciones

0	-1		R1=1
1	11	R1(1)	R2=2
2	-1	R2(1)	R3=1
3	-1		R4=1
4	-1		
5	-1		
6	-1		
7	-1		
8	-1		
9	-1		
10	-1		
11	12	R3(2)	
12	-1	**R4(3)**	
13	-1		
14	-1		
15	-1		

Direccionamiento Cerrado con area de overflow independiente: operaciones

0	-1		R1=1
1	11	R1(1)	R2=2
2	-1	R2(1)	R3=1
3	-1	**R5(1)**	R4=1
4	-1		R5=3
5	-1		
6	-1		
7	-1		
8	-1		
9	-1		
10	-1		
11	12	R3(2)	
12	-1	R4(3)	
13	-1		
14	-1		
15	-1		

Direccionamiento cerrado con área de overflow independiente (operaciones)

- Búsqueda
  - Buscar en la posición, si esta ocupada recorrer la cadena (deben ser sinónimos).
- Altas
  - Buscar en la posición, agregar o encadenar.
- Bajas y modificaciones
  - Buscar + dar de baja o modificar (al dar de baja actualizar la lista, eventualmente mover registros!)

Buckets

- En cada “registro” del archivo directo no se guarda un registro de datos sino hasta “n” registros. El archivo esta entonces dividido en “buckets” de tamaño “n”.
- Cada bucket puede contener unicamente registros que son sinónimos entre si.
- Se simplifica el manejo de sinónimos pero cuando el bucket se llena debe usarse alguno de los métodos que vimos hasta el momento.

Buckets

R0=1

0		
1	**R0**	
2		
3		
4		
5		
6		

Buckets

R0=1
R1=2

0		
1	R0	
2	**R1**	
3		
4		
5		
6		

Buckets

R0=1
R1=2
R2=1

0		
1	R0	**R2**
2	R1	
3		
4		
5		
6		

Buckets

R0=1	0		
R1=2	1	R0	R2
R2=1	2	R1	
R3=1	3	**R3**	
	4		
	5		
	6		

- Puede usarse direccionamiento abierto o cerrado para R3 con area de overflow independiente o no aplicando los métodos vistos hasta ahora.
- En cada bucket debe existir como dato administrativo la cantidad de registros ocupados en el Bucket.
- Al dar de baja es necesario actualizar este dato y mover los registros de forma tal que si en un bucket hay “k” registros estos sean los primeros

Reorganización

- Consiste en la reconstrucción del archivo directo
  - Por muestreo (para datos estáticos)
  - Para mejorar el promedio de accesos (reducir λ)
  - Porque no se puede dar de alta
  - Porque λ es demasiado chico

Hashing Perfecto

- Para toda clave ci

  $h(ci) = h(cj) <=> i=j$

- Es decir que no existen colisiones, por lo tanto no hace falta aplicar ningún método de resolución de colisiones.
- Encontrar una función de hashing perfecta es difícil, pero existen métodos para lograrlo.

Hashing Perfecto

- Objetivo:
  - Dado un conjunto de strings construir una función de hashinmg perfecta para los mismos
- Para N strings se debe elegir un número M>N
- Y se deben usar dos funciones de hashing que generen números entre 0 y M-1
- Se aplican ambas funciones a cada string y a partir de los resultados se construye un grafo

Hashing Perfecto

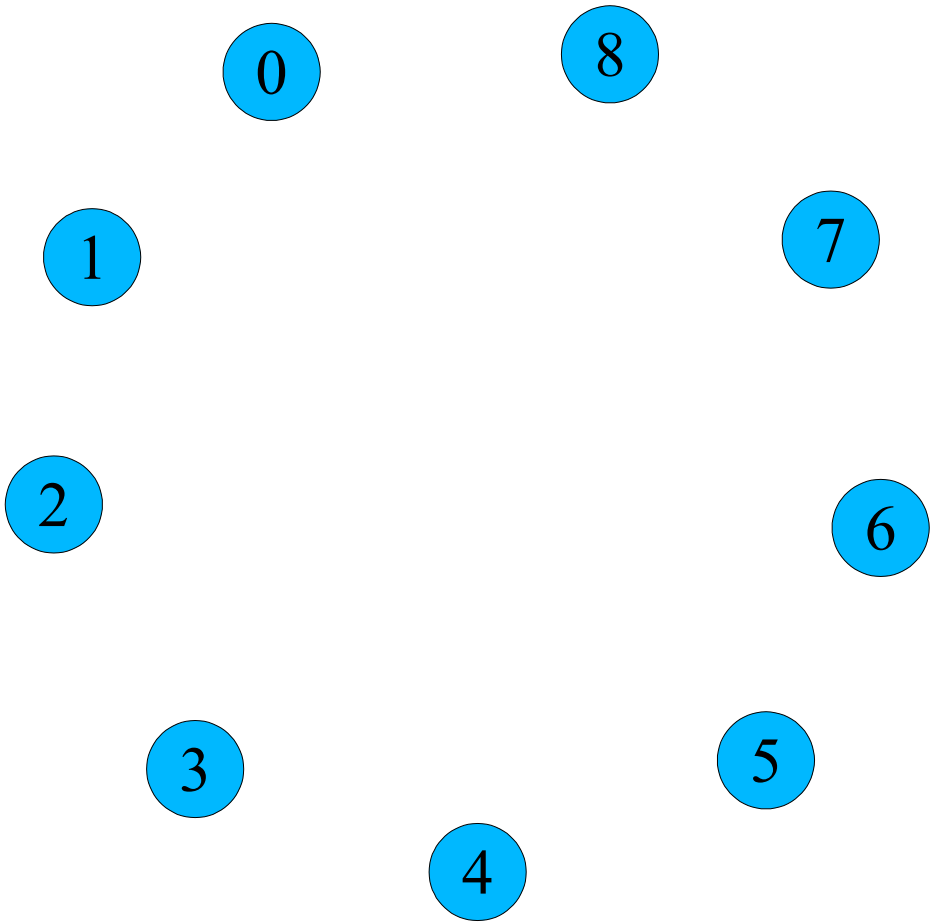
- Sea N=7, elegimos M=9 (M>7)

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4

- El grafo se arma con M vértices y N aristas, cada arista esta dada por los valores de h1 y h2 para cada string.

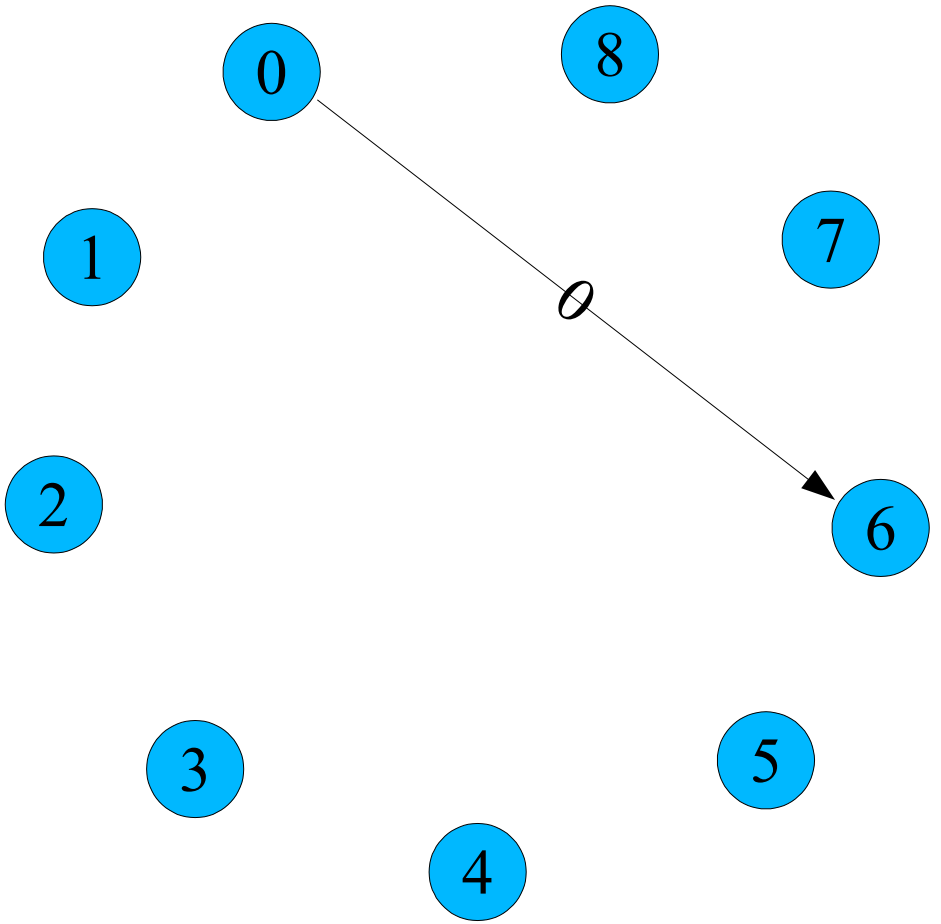
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



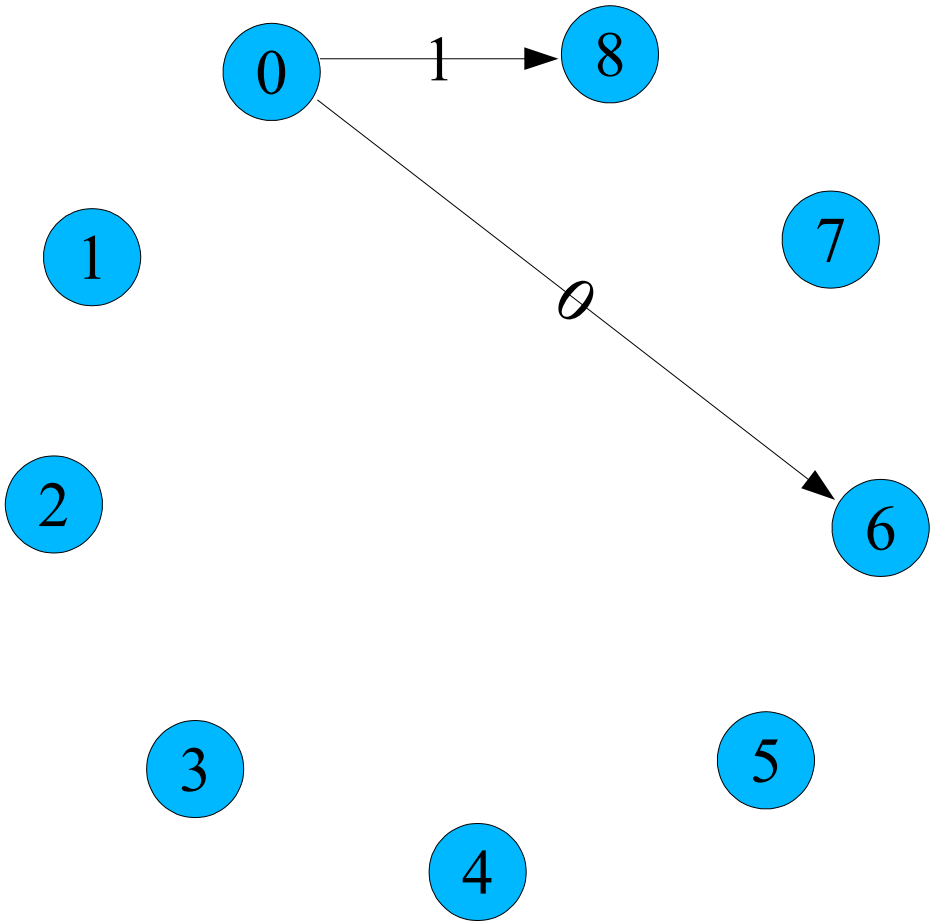
Hashing Perfecto

String	H1(s)	H2(s)
S0	**0**	**6**
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



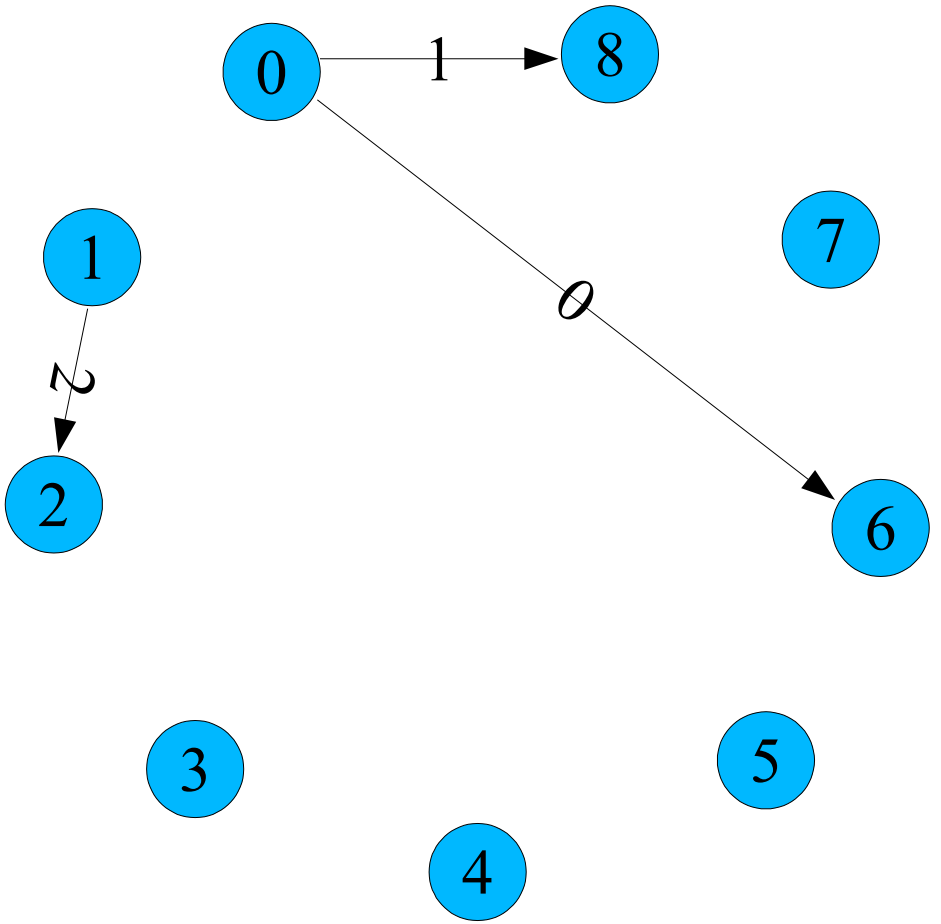
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	**0**	**8**
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



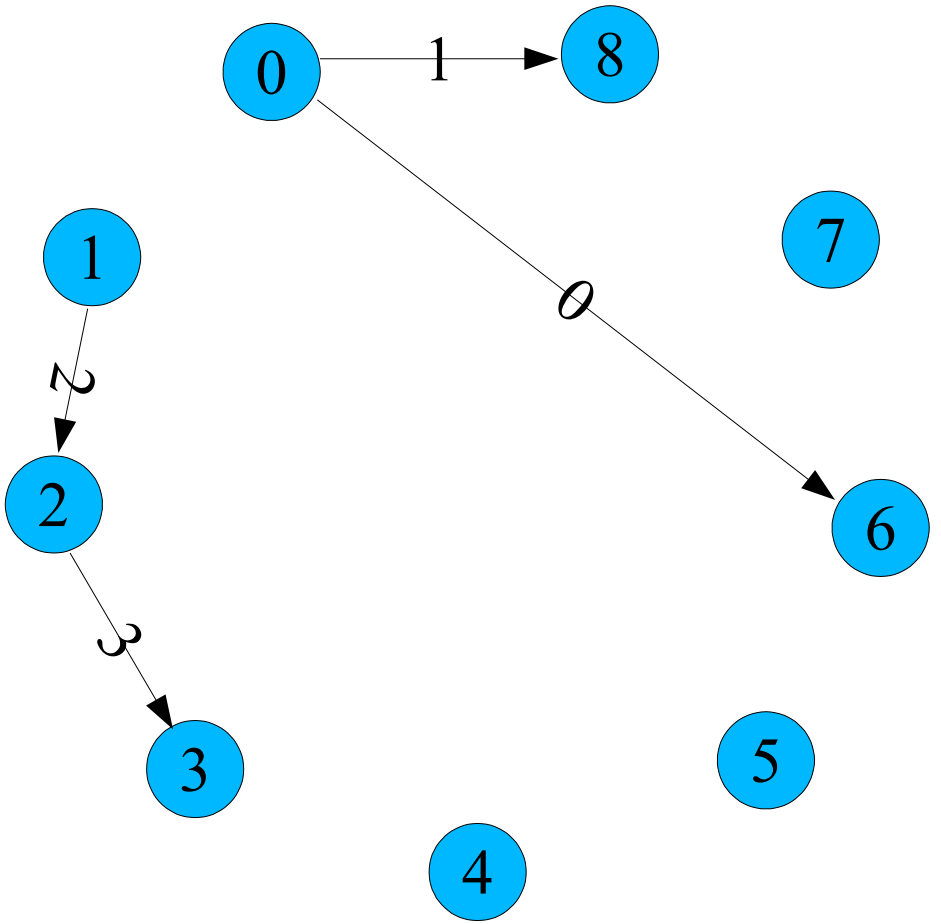
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	**1**	**2**
S3	2	3
S4	7	8
S5	4	5
S6	0	4



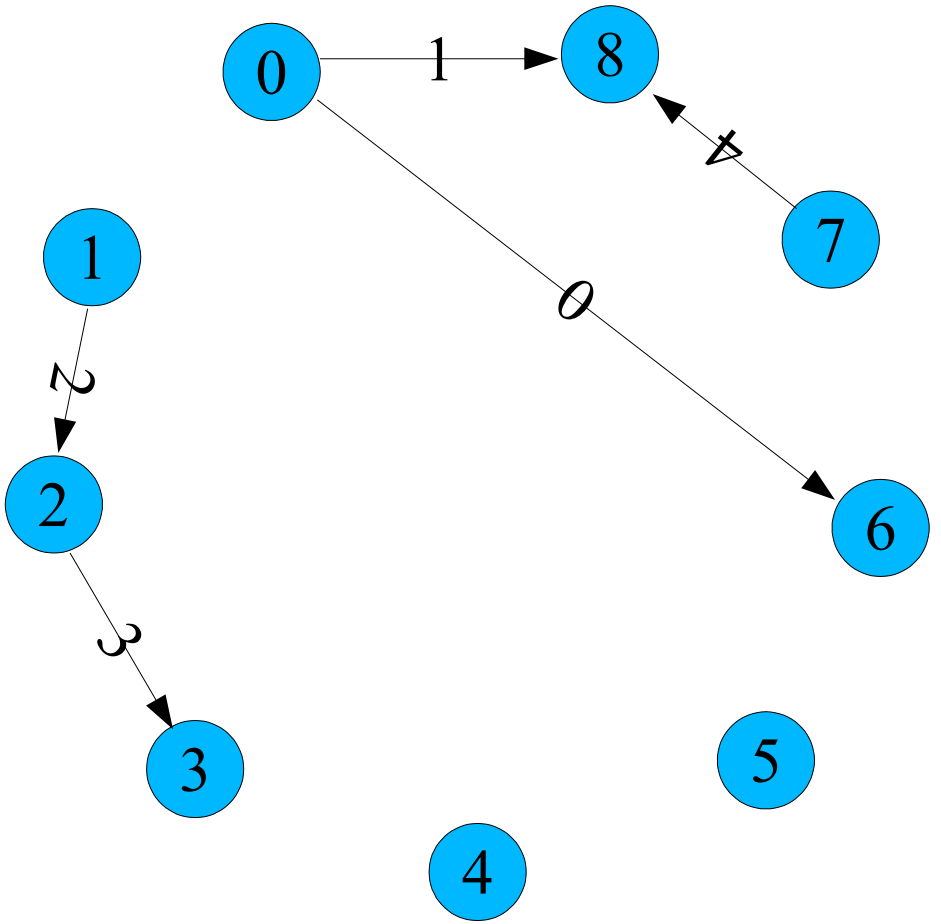
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	**2**	**3**
S4	7	8
S5	4	5
S6	0	4



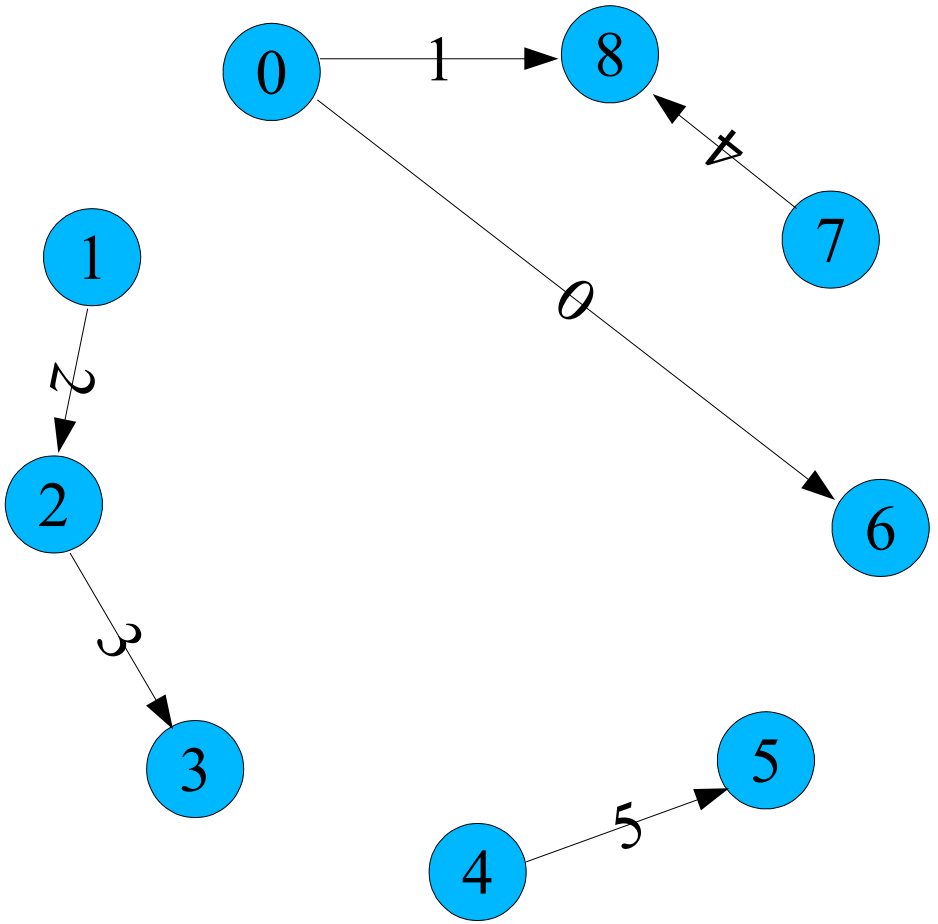
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	**7**	**8**
S5	4	5
S6	0	4



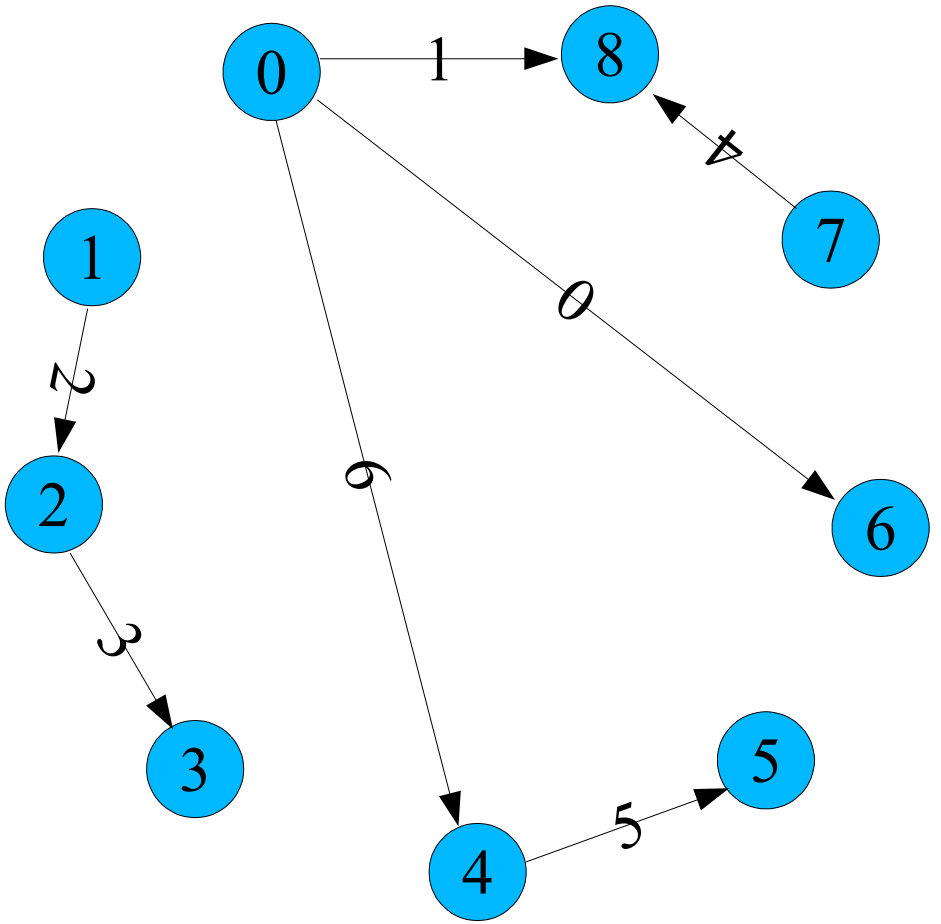
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	**4**	**5**
S6	0	4



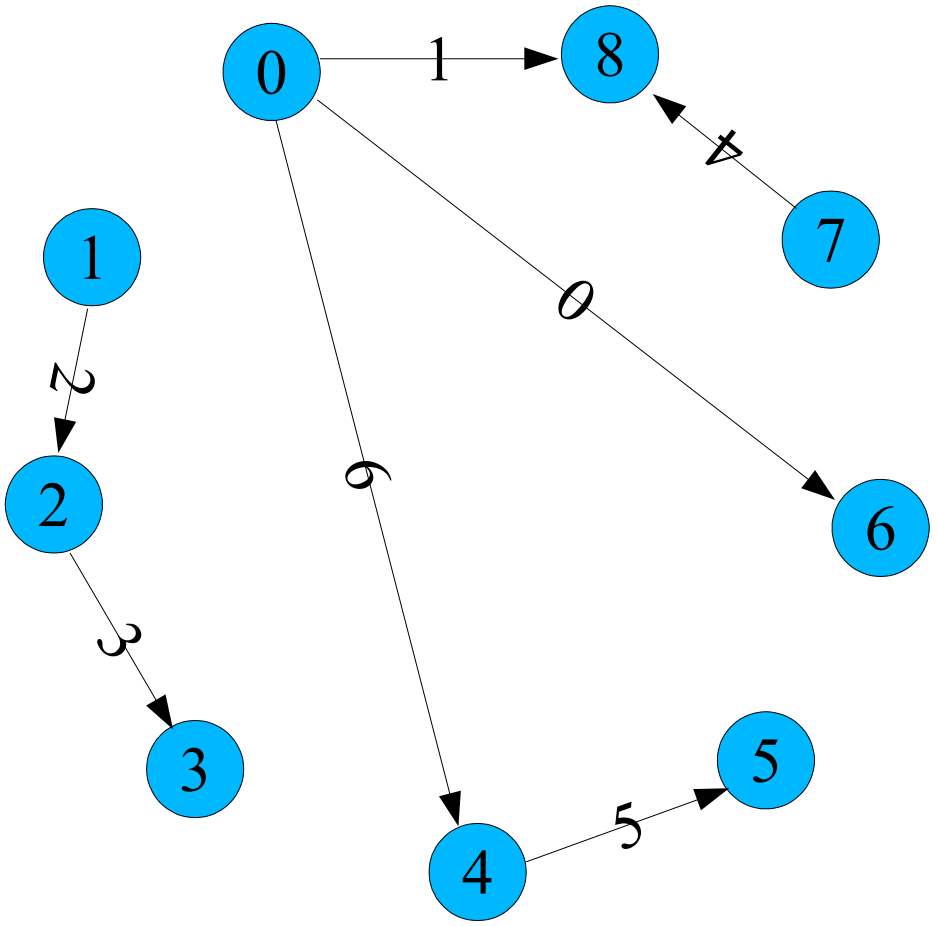
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	**0**	**4**



Hashing Perfecto
Verificar que el grafo no tenga ciclos

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	**0**	**4**

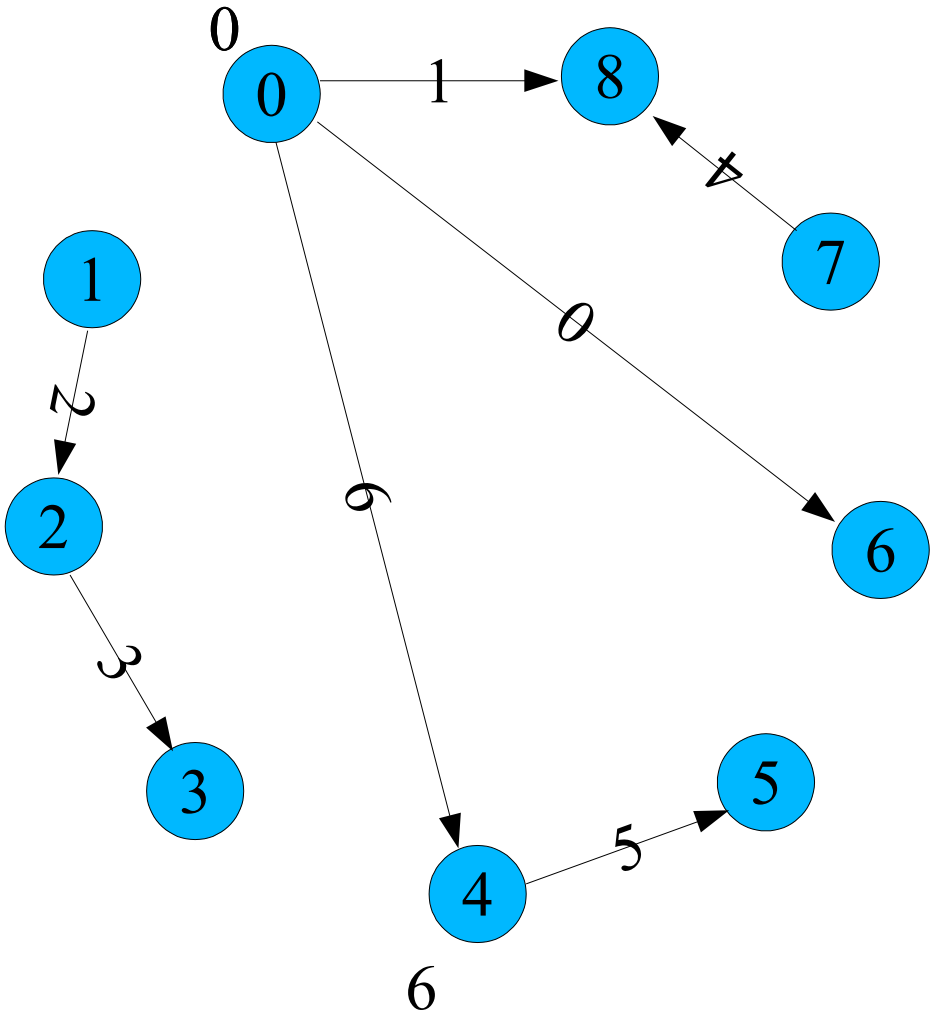


Hashing Perfecto

- Algoritmo para rotular los vértices
  - Mientras queden vértices sin rotular
    - Rotular un vértice cualquiera con 0 (podría ser cualquier otro número)
      - Rotular los vértices adyacentes con el valor de la arista menos el valor del vértice módulo N
      - Repetir recursivamente hasta que no haya mas vértices adyacentes a un vértice rotulado sin rotular

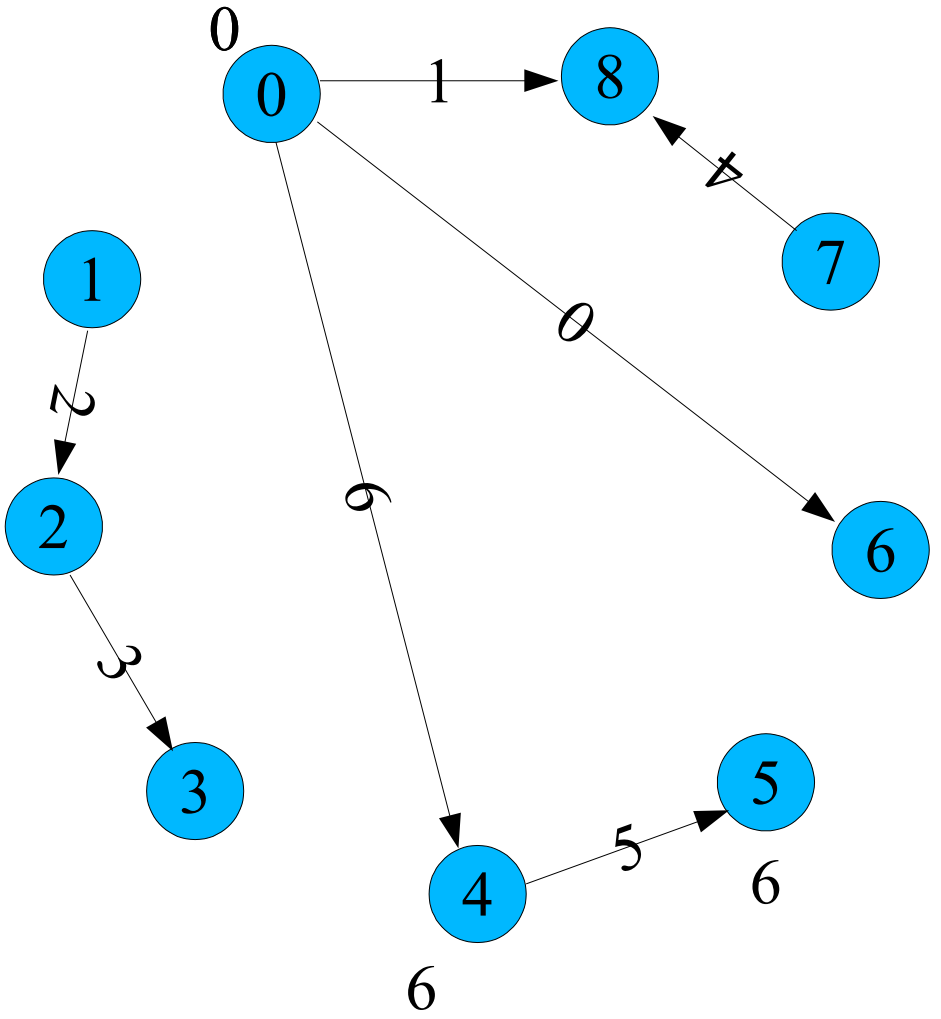
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



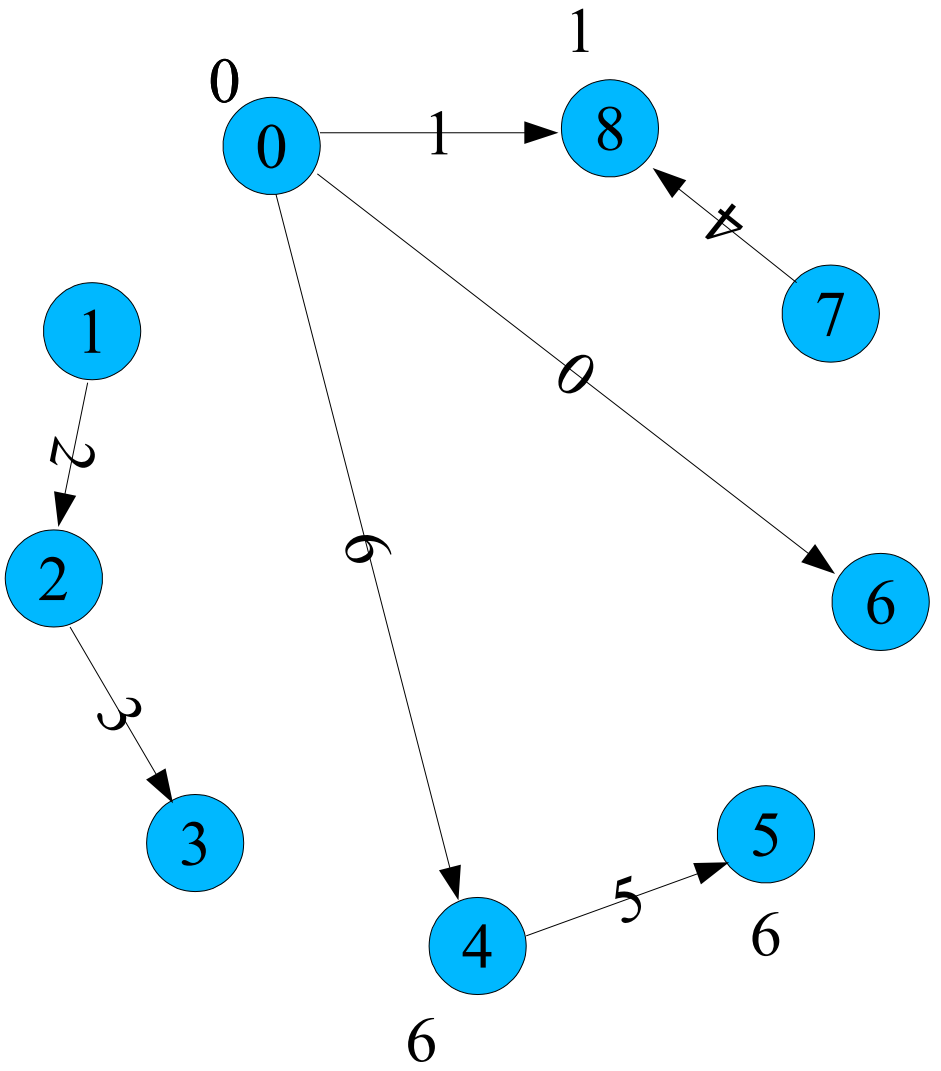
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



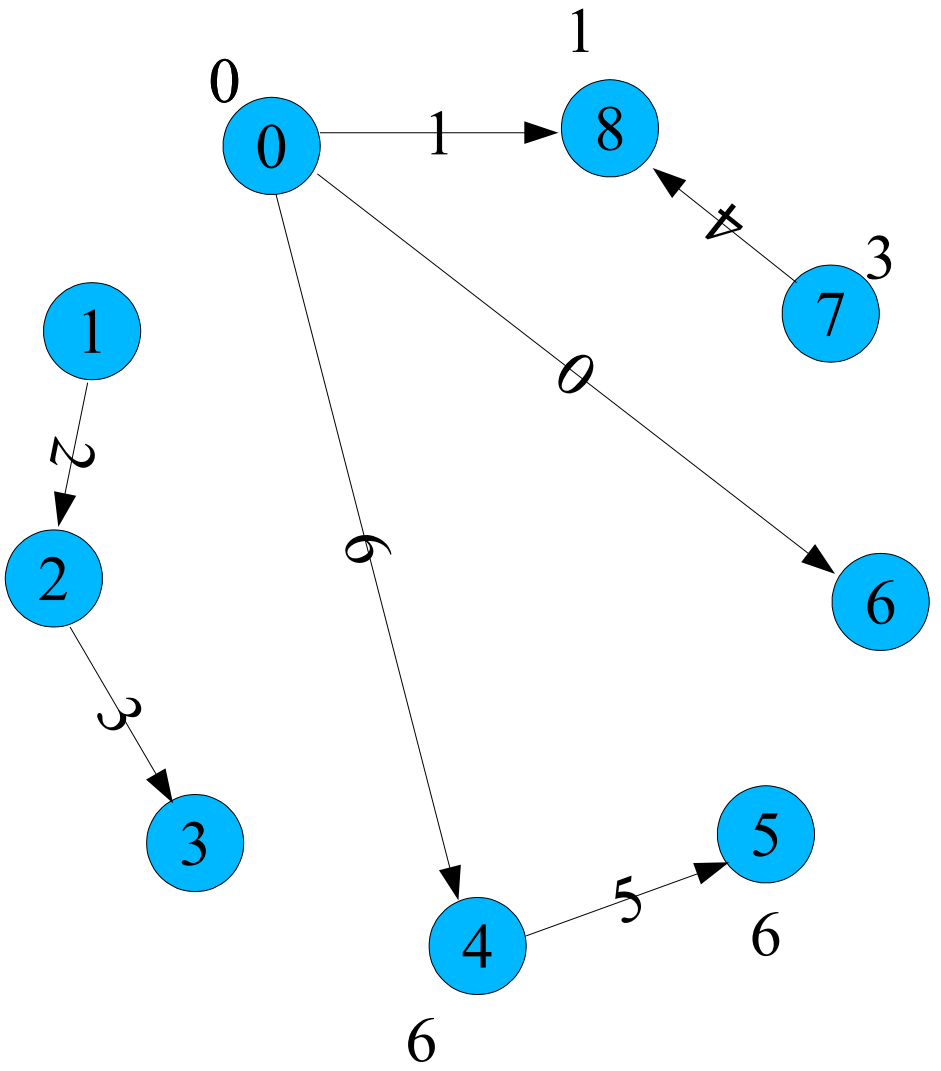
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



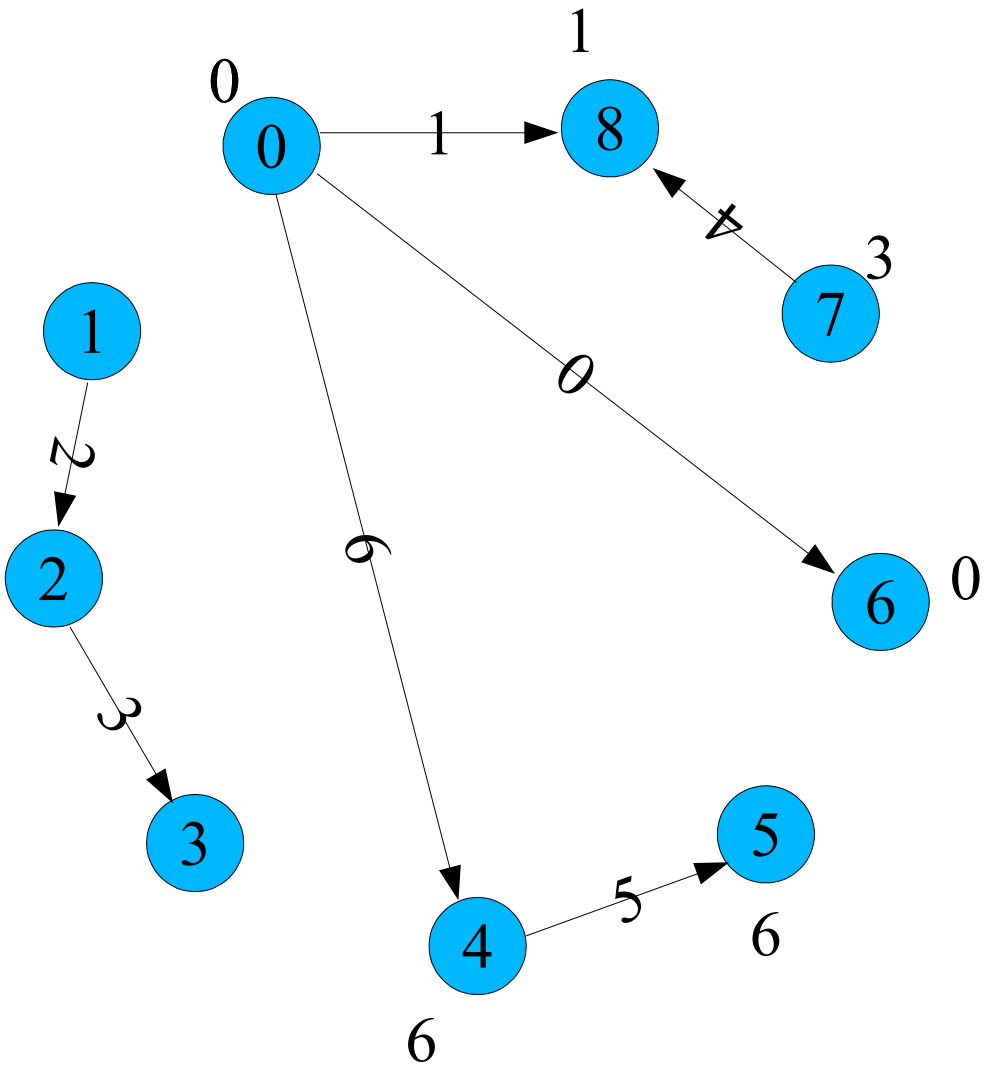
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



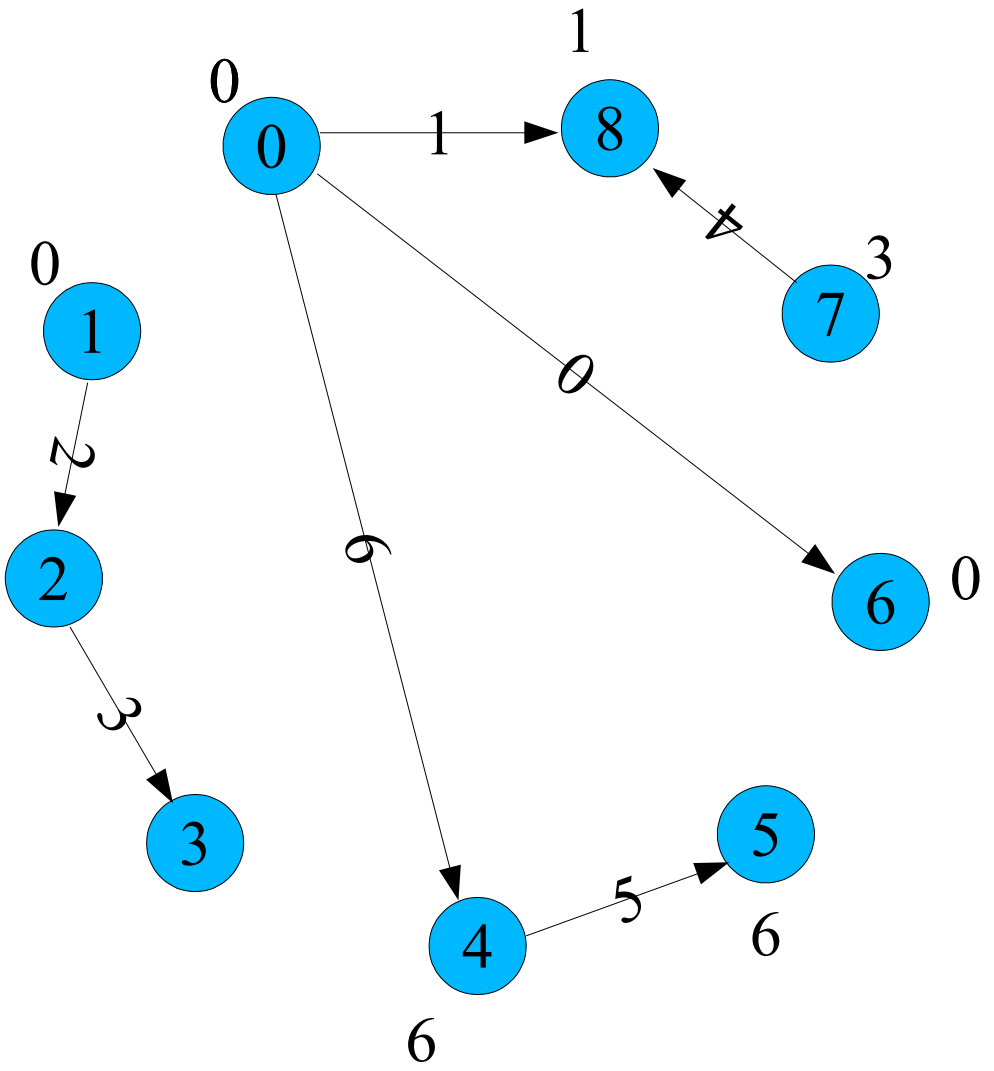
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



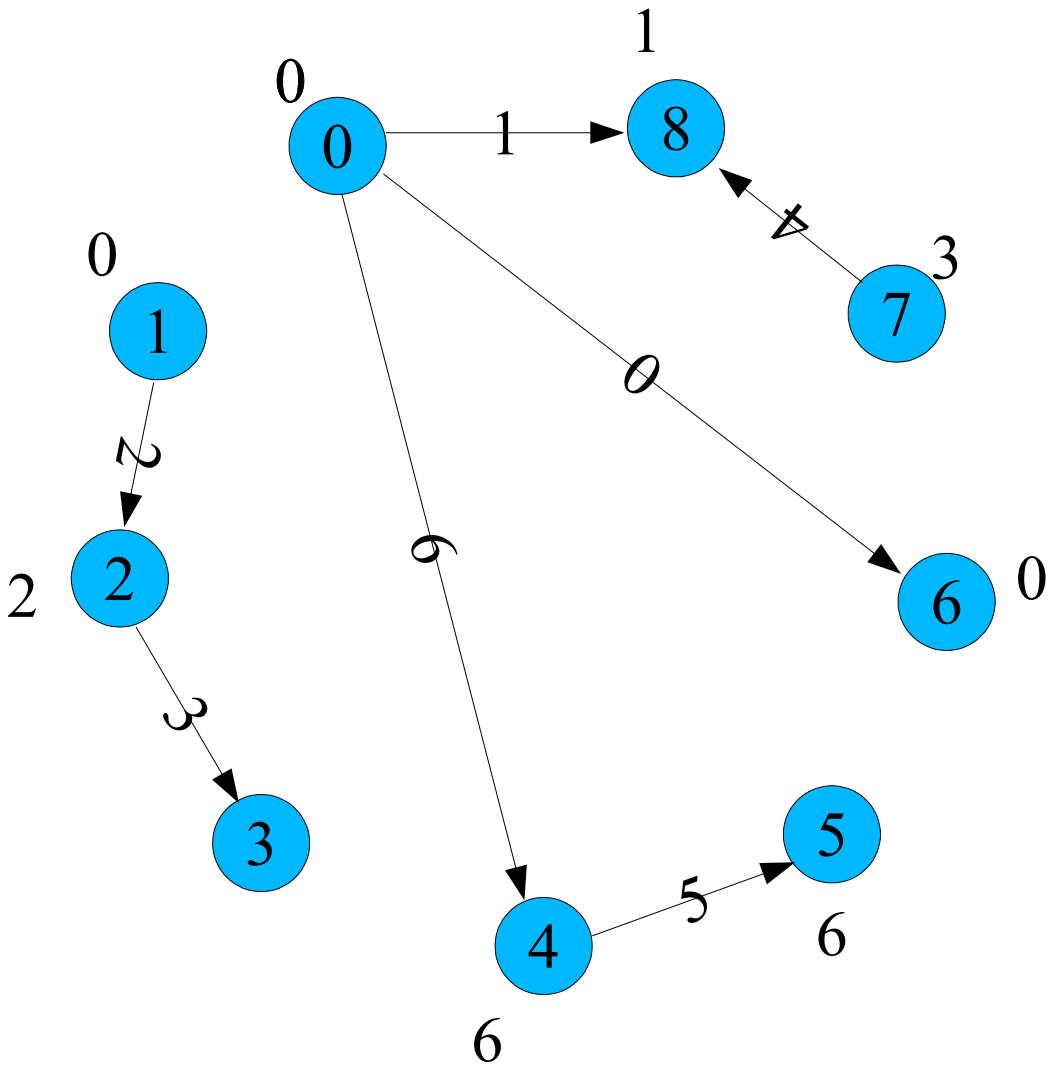
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



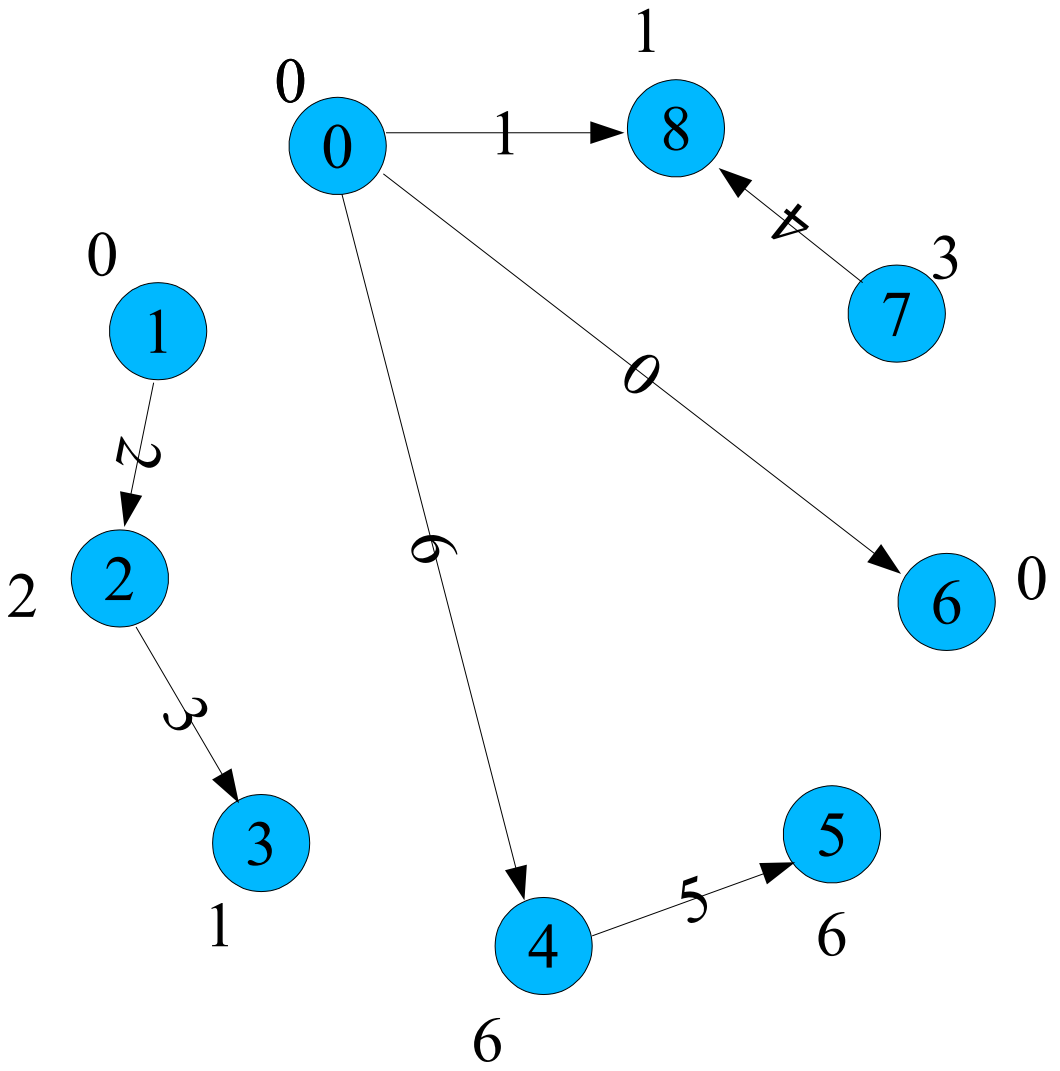
Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



Hashing Perfecto

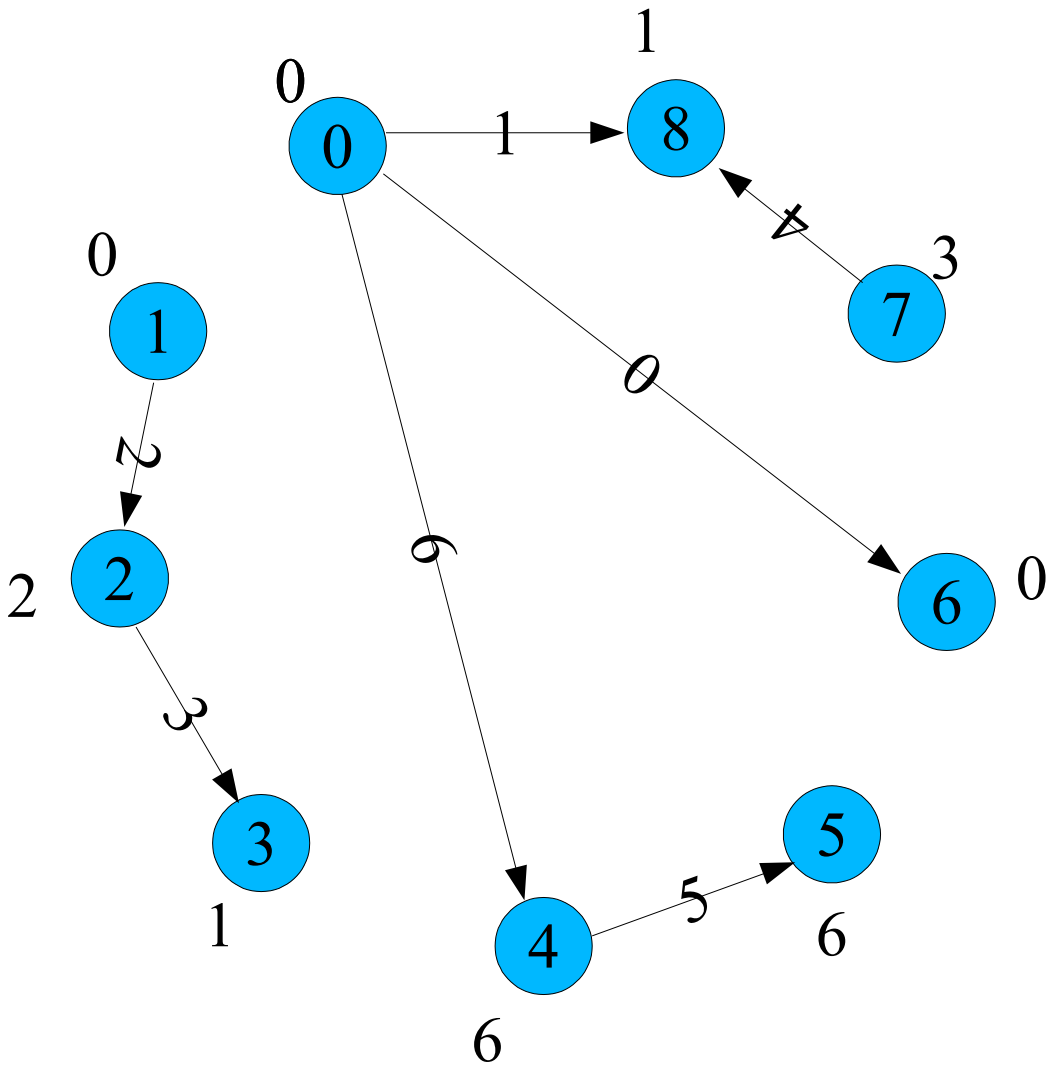
String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4



Hashing Perfecto

String	H1(s)	H2(s)
S0	0	6
S1	0	8
S2	1	2
S3	2	3
S4	7	8
S5	4	5
S6	0	4

X	G(x)
0	0
1	0
2	2
3	1
4	6
5	6
6	0
7	3
8	1



Hashing Perfecto

X	G(x)
0	0
1	0
2	2
3	1
4	6
5	6
6	0
7	3
8	1

String	H1(s)	H2(s)	G(H1(s))	G(H2(s))	G1+G2
S0	0	6	0	0	0
S1	0	8	0	1	1
S2	1	2	0	2	2
S3	2	3	2	1	3
S4	7	8	3	1	4
S5	4	5	6	6	5
S6	0	4	0	6	6

- La función “G” construida es almacenada junto con el archivo directo, en un header o en un archivo aparte.
- A partir del string, h1, h2 y G se calcula la posición de cada string de forma perfecta, mínima y preservando el orden.

Hashing Perfecto

- Si el grafo tuviera ciclos se deben cambiar las funciones de hashing h1 y h2 y repetir.
- ¿Cuántos grafos se prueban en promedio? Depende del valor de M

$$\sqrt{\frac{M}{M-2N}}$$

Hashing Perfecto

- Funciones de hashing configurables
- Sea un vector random v[0]...v[k-1] (k elementos)
- Se calcula h(s) como:

$$(\sum S_i * V_i) \, mod \, N$$

- De esta forma variando el vector es posible variar la función tantas veces como se quiera

Hashing Perfecto: Gperf

gperf

Introduction | Get the Software

Introduction to gperf

GNU gperf is a perfect hash function generator. For a given list of strings, it produces a hash function and hash table, in form of C or C++ code, for looking up a value depending on the input string. The hash function is *perfect*, which means that the hash table has no collisions, and the hash table lookup needs a single string comparison only.

GNU gperf is highly customizable. There are options for generating C or C++ code, for emitting `switch` statements or nested `if`s instead of a hash table, and for tuning the algorithm employed by gperf.

Online Manual is available at www.gnu.org/software/gperf/manual/

Downloading gperf

gperf can be found on in the subdirectory `/gnu/gperf/` on your favorite GNU mirror. For other ways to obtain gperf, please read How to get GNU Software

---