Grafos: detectar ciclos - union find

AED II

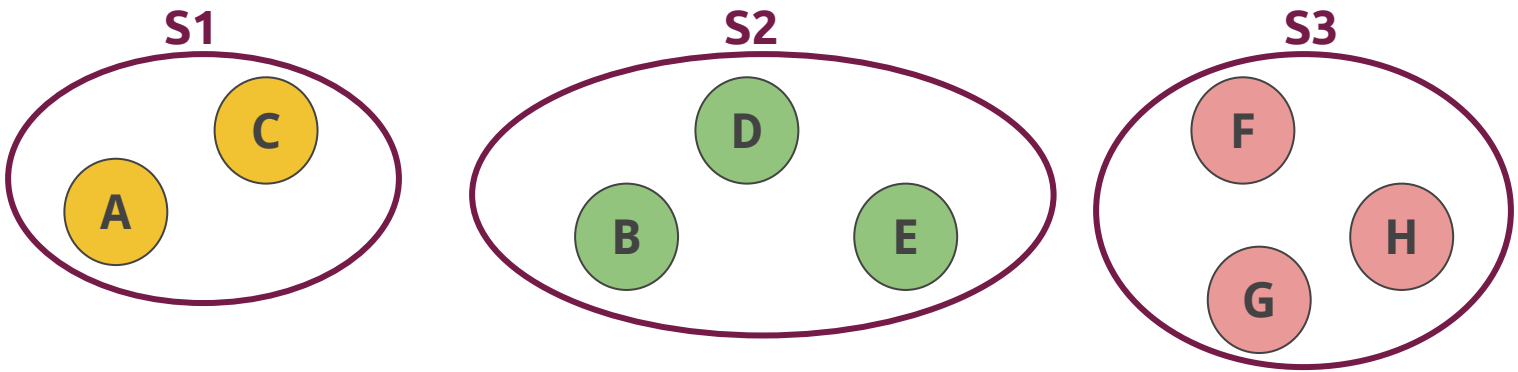
2020

UBP UNIVERSIDAD BLAS PASCAL

detectar ciclos en grafos: conjuntos disjuntos

Conjunto disjunto: es una estructura de datos que mantiene un conjunto de elementos particionados en un número de subconjuntos disjuntos

$S1 \cap S2 = \emptyset$, $S2 \cap S3 = \emptyset$, $S1 \cap S3 = \emptyset$,



conjuntos disjuntos: union - find

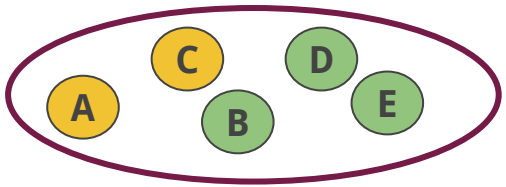
El conjunto disjunto soporta dos operaciones:

- Unión (union)
- Buscar (find)

conjuntos disjuntos: union - find

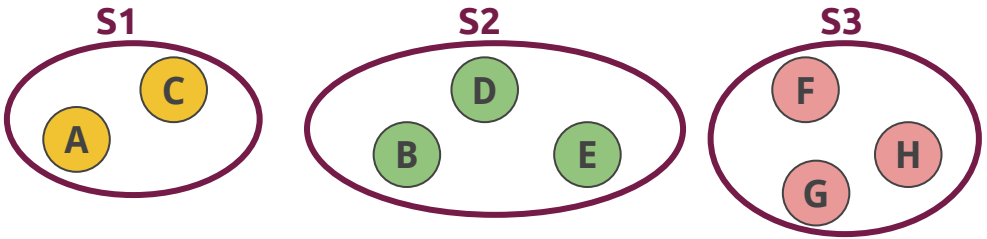
Union: permite combinar dos subconjuntos en uno nuevo que contiene los elementos de ambos.

UNION(S1, S2)

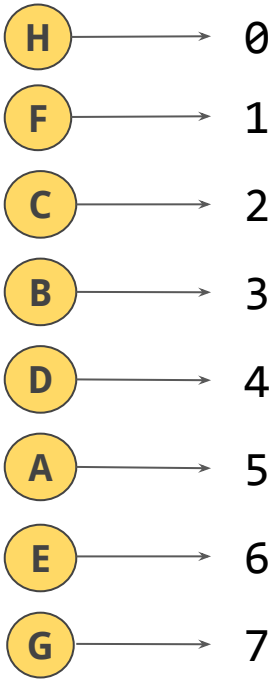


Find: permite determinar a qué subconjunto pertenece un elemento

FIND(H) = S3
FIND(D) = S2



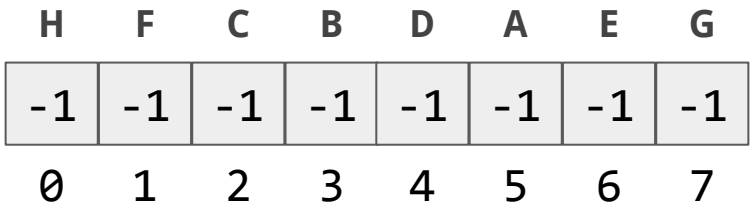
union - find: implementación



Para implementarla vamos a construir una biyección entre los elementos y los enteros del rango [0,n).

Este mapeo es arbitrario y no necesariamente se debe hacer así pero nos permite implementar la estructura en un array que es muy conveniente.

union - find: implementación



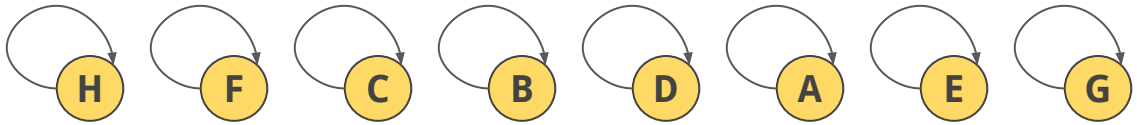
Almacenamos la información en un array en donde la posición indica el entero asociado al elemento y el valor contenido el padre de ese elemento.

Inicialmente todos los elementos se tienen a sí mismos como padres. Indicaremos esto con el valor -1

union - find: implementación

Union(C, A)

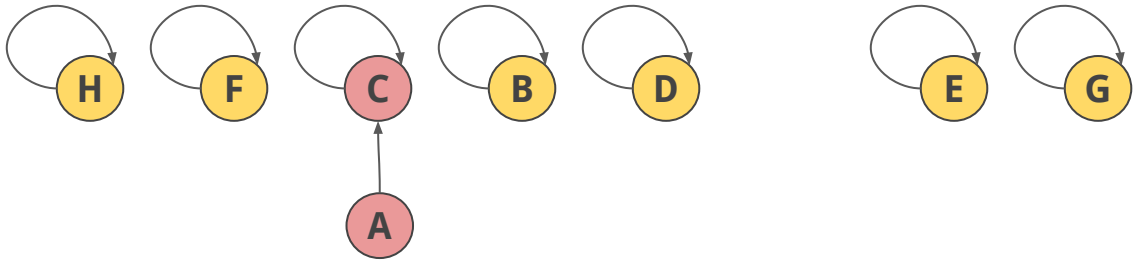
H	F	C	B	D	A	E	G
-1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7



union - find: implementación

Union(C, A)

H	F	C	B	D	A	E	G
-1	-1	-1	-1	-1	2	-1	-1
0	1	2	3	4	5	6	7

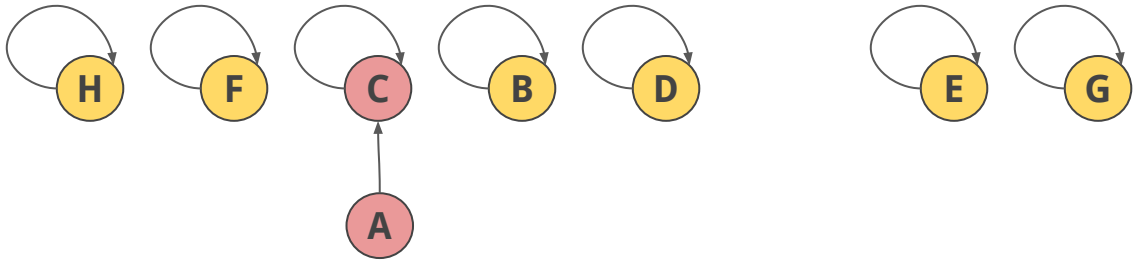


union - find: implementación

Union(C, A)

Union(F, H)

H	F	C	B	D	A	E	G
-1	-1	-1	-1	-1	2	-1	-1
0	1	2	3	4	5	6	7

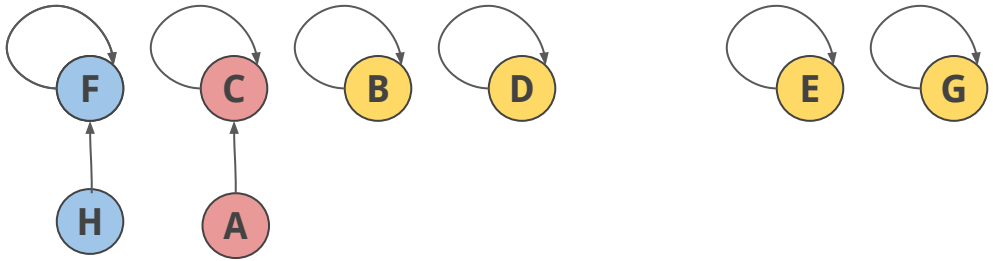


union - find: implementación

Union(C, A)

Union(F, H)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	-1	-1
0	1	2	3	4	5	6	7



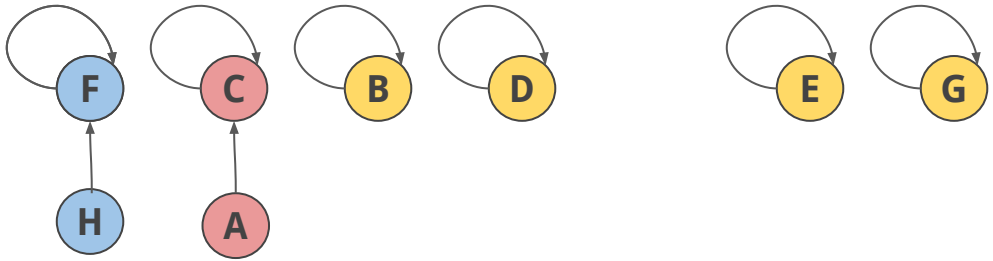
union - find: implementación

Union(C, A)

Union(F, H)

Union(D, E)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	-1	-1
0	1	2	3	4	5	6	7



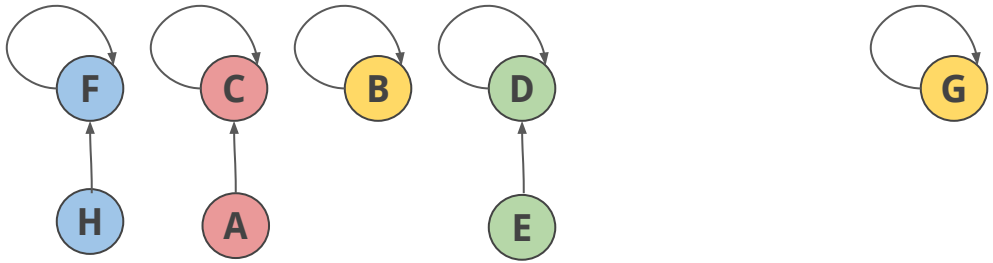
union - find: implementación

Union(C, A)

Union(F, H)

Union(D, E)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	4	-1
0	1	2	3	4	5	6	7



union - find: implementación

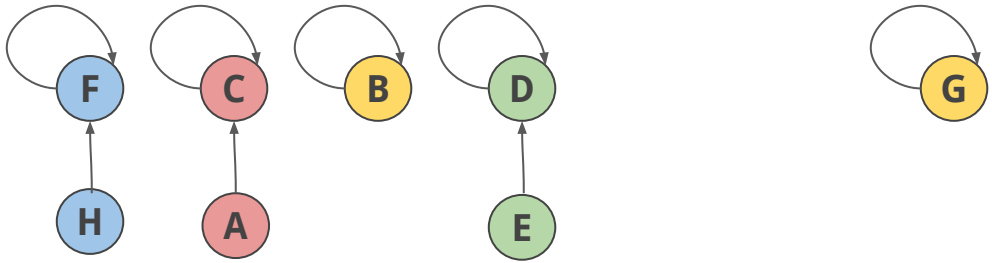
Union(C, A)

Union(F, H)

Union(D, E)

Union(G, A)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	4	-1
0	1	2	3	4	5	6	7



union - find: implementación

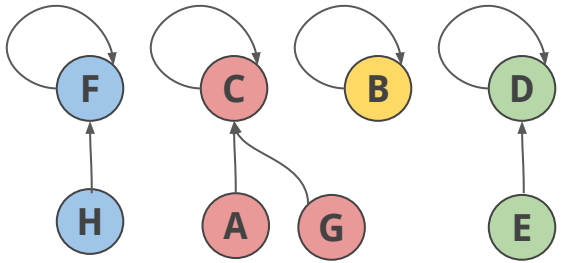
Union(C, A)

Union(F, H)

Union(D, E)

Union(G, A)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	4	2
0	1	2	3	4	5	6	7



union - find: implementación

Union(C, A)

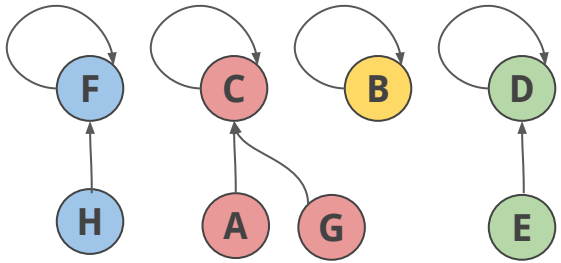
Union(F, H)

Union(D, E)

Union(G, A)

Union(B, F)

H	F	C	B	D	A	E	G
1	-1	-1	-1	-1	2	4	2
0	1	2	3	4	5	6	7



union - find: implementación

Union(C, A)

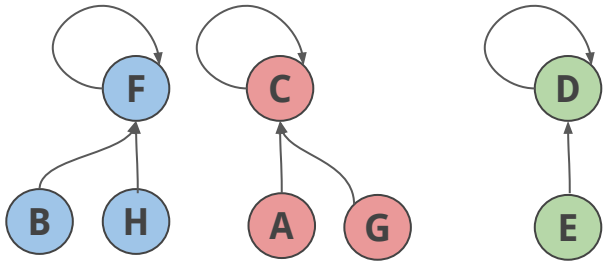
Union(F, H)

Union(D, E)

Union(G, A)

Union(B, F)

H	F	C	B	D	A	E	G
1	-1	-1	1	-1	2	4	2
0	1	2	3	4	5	6	7



union - find: implementación

Union(C, A)

Union(F, H)

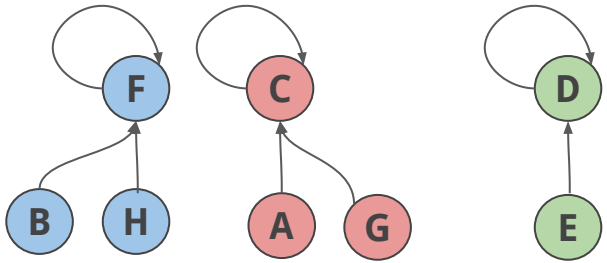
Union(D, E)

Union(G, A)

Union(B, F)

Union(E, C)

H	F	C	B	D	A	E	G
1	-1	-1	1	-1	2	4	2
0	1	2	3	4	5	6	7



union - find: implementación

Union(C, A)

Union(F, H)

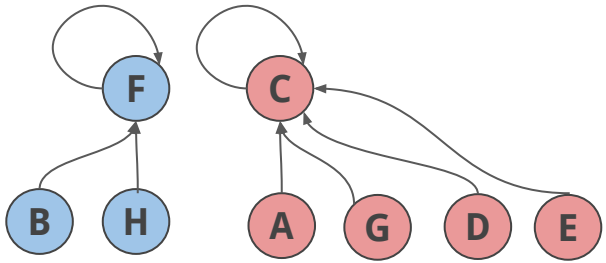
Union(D, E)

Union(G, A)

Union(B, F)

Union(E, C)

H	F	C	B	D	A	E	G
1	-1	-1	1	2	2	2	2
0	1	2	3	4	5	6	7



union - find: implementación

- Unión (union)
  Se busca cual es el padre de cada elemento y si son diferentes, se hace que el padre de un elemento sea el padre del otro

- Buscar (find)
  Para hallar a qué conjunto pertenece un elemento se siguen los padres hasta que se encuentra una autoreferencia

detectar ciclos con union - find

El algoritmo union find se puede usar para detectar ciclos en un grafo no dirigido

Para cada arista se crean subconjuntos con ambos vértices. Si para alguna arista los dos vértices están en el mismo subconjunto, entonces estamos en presencia de un ciclo

detectar ciclos con union - find

```
para cada arista (u,v){
    x = Find(u)
    y = Find(v)
    si (x == y)
        // El grafo tiene un ciclo
    si no
        Union (x, y)
}
```

Ejercicio 1.19.

Implementá el algoritmo union-find para detectar ciclos en un grafo no dirigido.

Ejercicio 1.19.

```
struct Arista{
    int origen;
    int destino;
    int peso;
};
class Grafo {
     private:
          int v;
          int a;
          int na;
          Arista* aristas;
          int Find(int* padre, int i);
          void Union(int* padre, int x, int y);
     public:
          Grafo(int, int);
          ~Grafo();
          void AgregarArista(int,int,int);
          void Mostrar();
          bool TieneCiclos();
};
```

Ejercicio 1.19.

```
int Grafo::Find(int* padre, int i){
      if (padre[i] == -1)
            return i;
      return Find (padre, padre[i]);
}

void Grafo::Union(int* padre, int x, int y){
      int conjuntox = Find(padre, x);
      int conjuntoy = Find(padre, y);
      padre[conjuntox] = conjuntoy;
}
```

Ejercicio 1.19.

```
bool Grafo::TieneCiclos(){
      int* padre = new int[v];
      for (int i = 0; i < v; i++){
            padre[i] = -1;
      }
      Arista* a = aristas;
      while (a){
            int x = Find(padre, a->origen);
            int y = Find(padre, a->destino);


            if (x == y)
                  return true;
            Union(padre, x, y);

            a = a->siguiente;
      }
      return false;
}
```