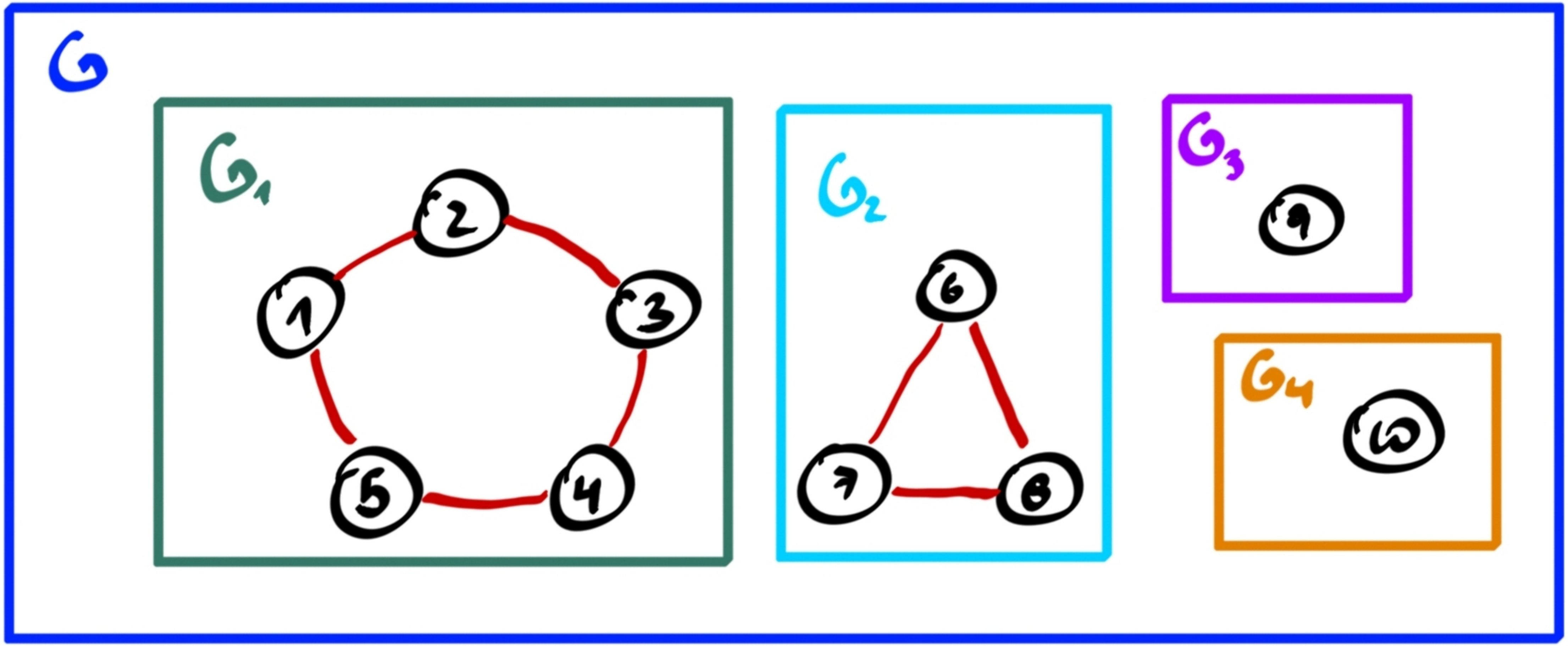
Grafos II: From $(0.3)\cdot$hero to $(0.5)\cdot$ hero

V. Benjamín Letelier Lazo
(benjamin.letelier@progcomp.cl)

Retomando en conexidad

Vimos que para obtener las componentes conexas, podemos hacer DFS/BFS iniciando en cada vértice del grafo que no haya sido visitado anteriormente.

Retomando en conexidad



Código para identificar la cantidad de componentes conexas

```cpp
//tengo una lista de adyacencia ya leída,
//con n nodos y m aristas.
//al inicio ningún nodo ha sido visitado.
vector<bool> visitados(n, false);
int n_comp_conexas = 0;
for(int i = 0; i < n; ++i) {
    if(visitados[i] == false) {
        dfs(i);
        //bfs(i);
        n_comp_conexas++;
    }
}
cout << "Existen " << n_comp_conexas << " en el grafo.\n";
```

Retomando en conexidad

¿Existirá una manera más sencilla de obtener información de las componentes conexas de un grafo?

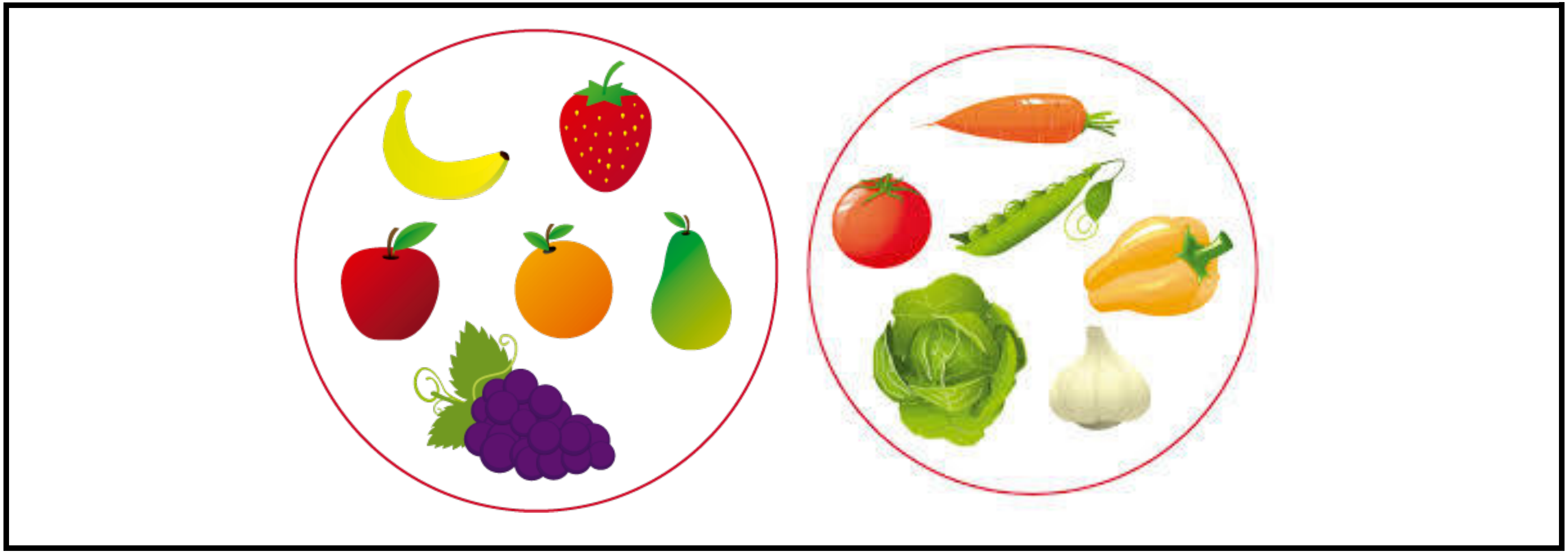
Retomando en conexidad

¡SÍ, SE PUEDE!

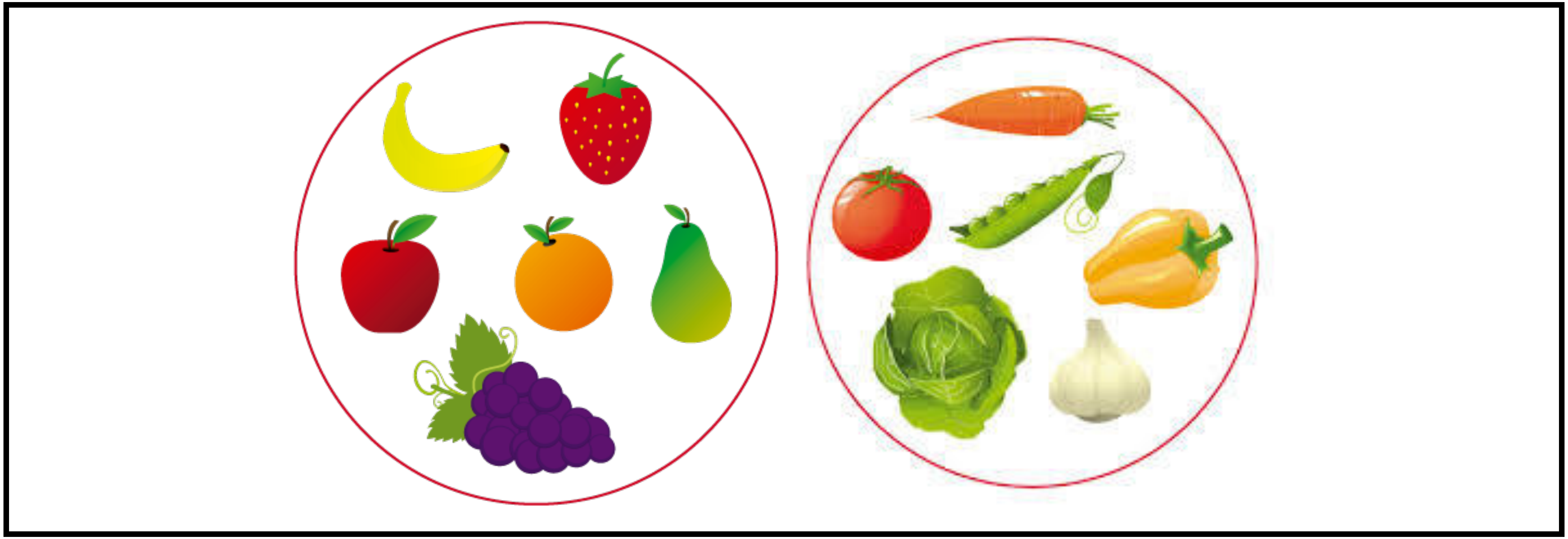
Estructura de datos Union-Find (DSU)

Union-Find es una estructura para *representar*, *consultar* y *modificar* **conjuntos**.

¿Qué?



¿Qué?



¿🍓 ∈ **Frutas** ∧ 🍐 ∈ **Frutas**? = **Verdadero**

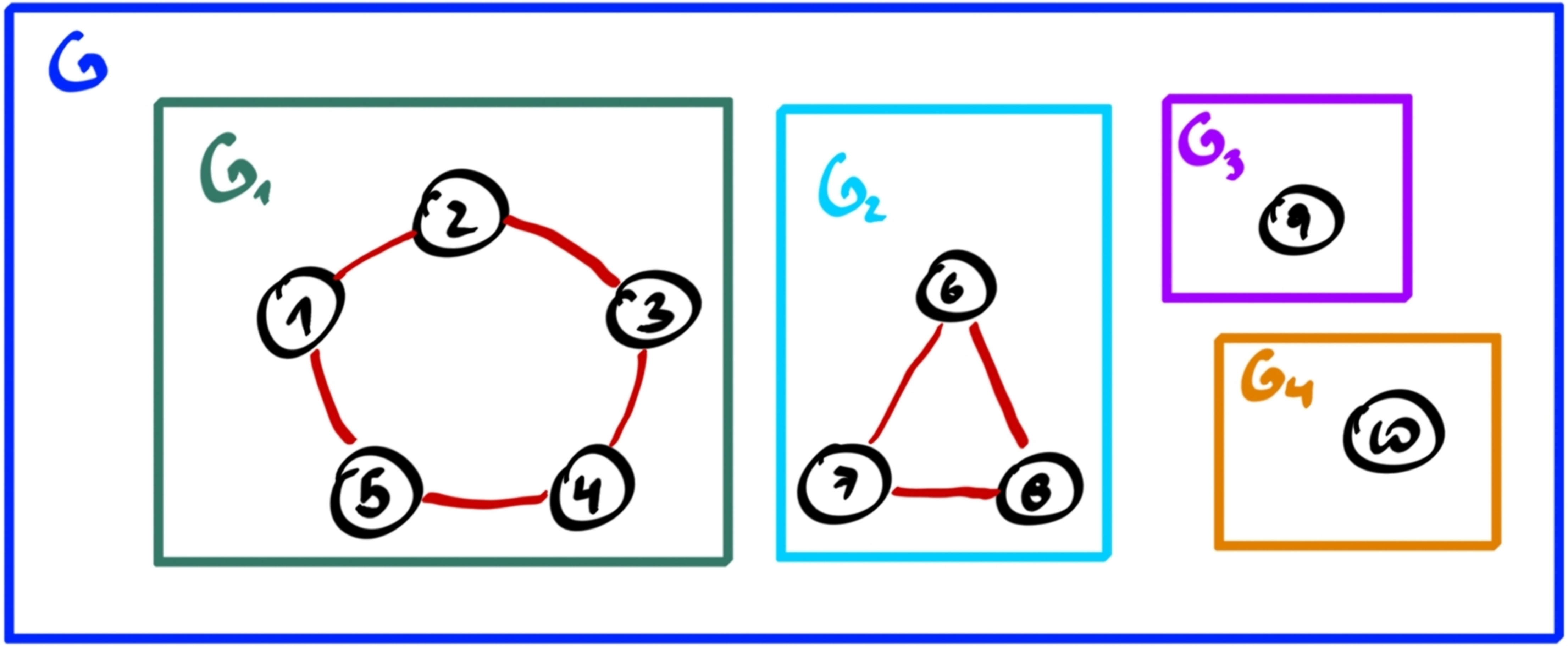
¿🍇 ∈ **Verduras**? = **Falso**

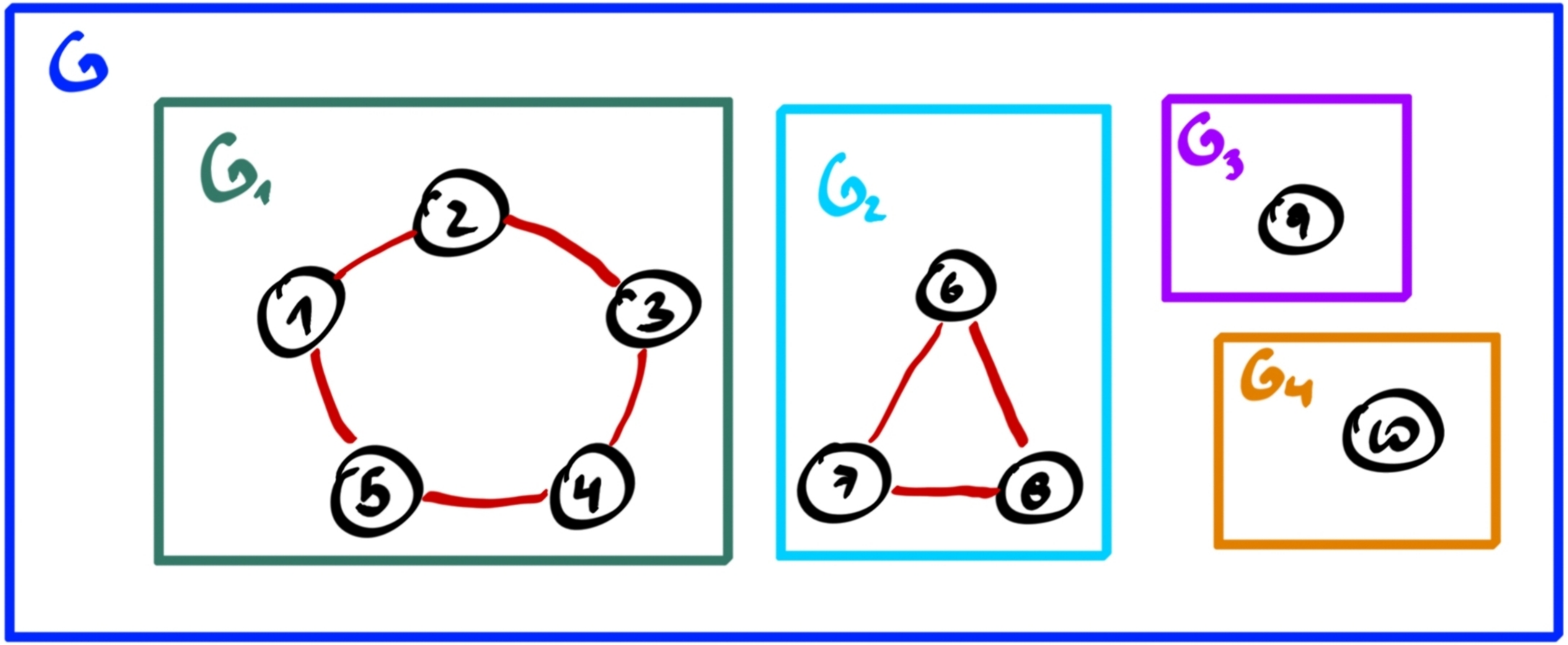
¿🍎 ∈ **Verduras** ∧ 🥕 ∈ **Verduras**? = **Falso**

Frutas ∪ {🥝}

¿|**Frutas**|? = **7** *(luego de añadir 🥝)*

¿Y qué tiene que ver con componentes conexas?





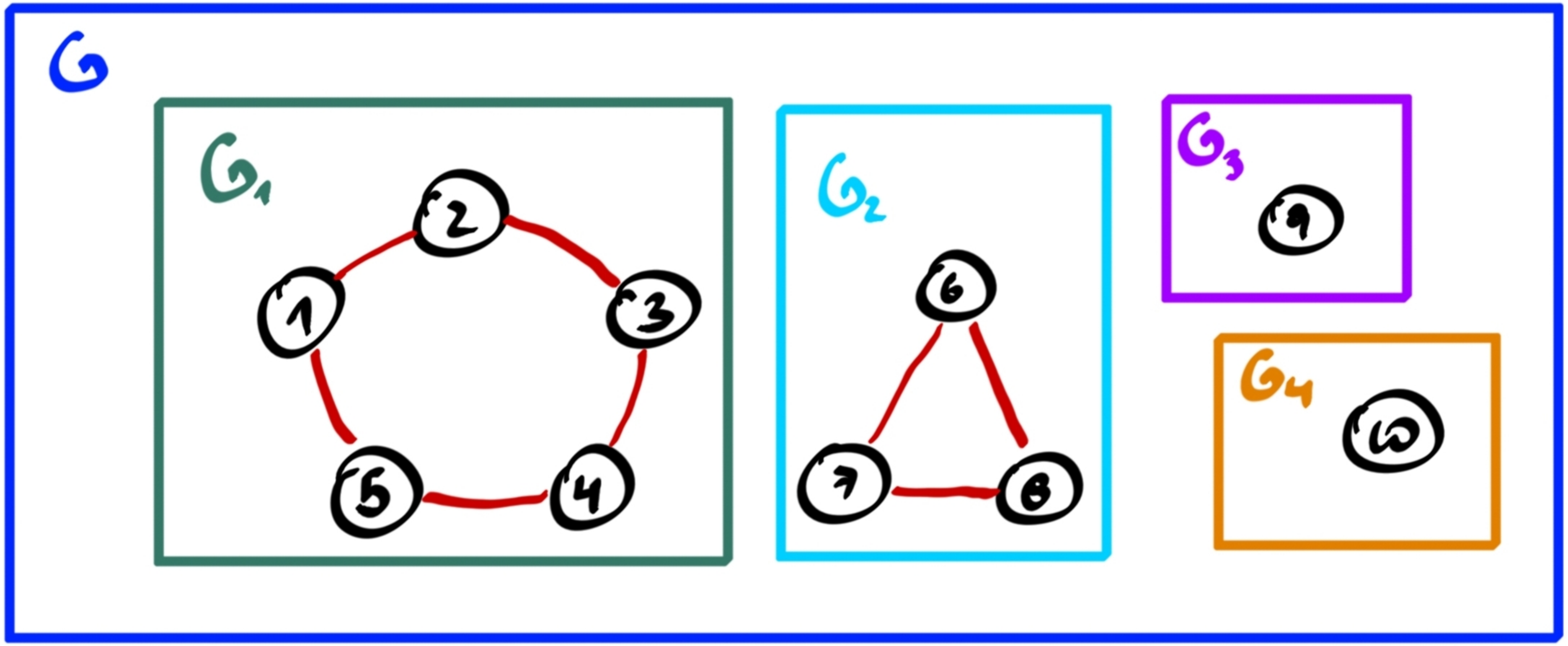
$$¿1 \in G_1 \wedge 4 \in G_1? = \textbf{Verdadero}$$

$$¿6 \in G_2? = \textbf{Verdadero}$$

$$¿9 \in G_3 \wedge 10 \in G_3? = \textbf{Falso}$$

$$¿|G_2|? = 3$$

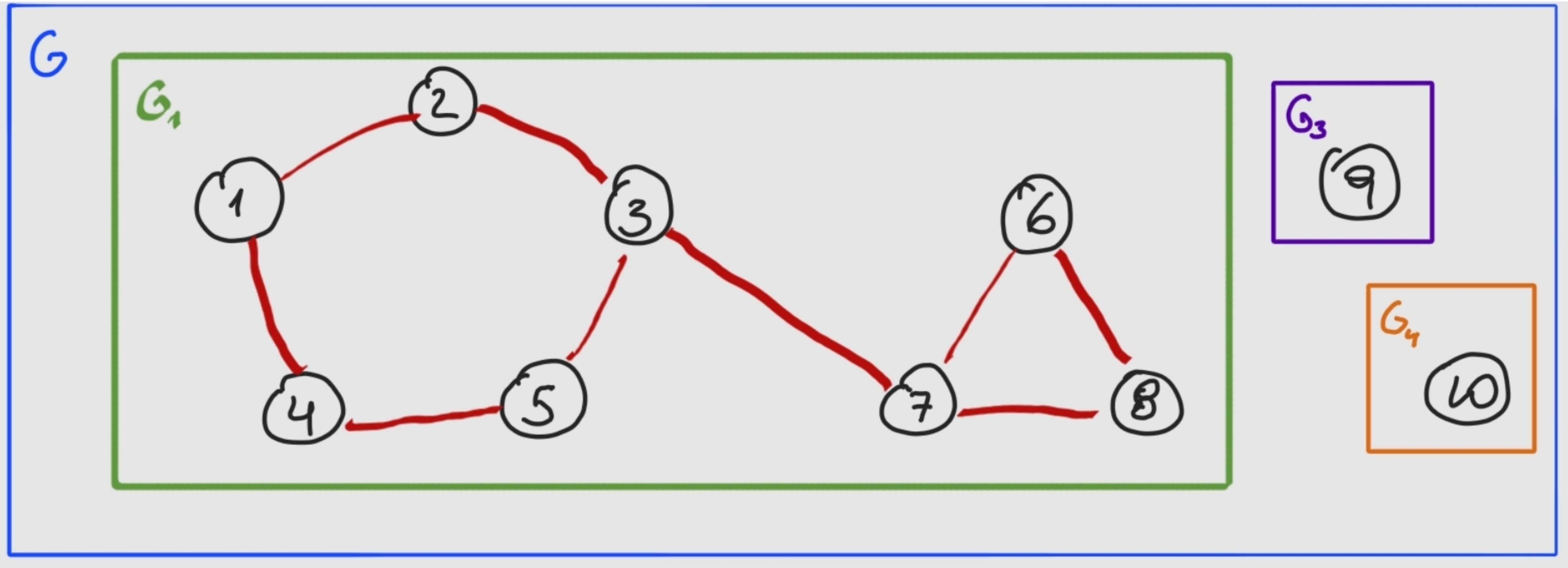
¿Qué pasa si justo llega una nueva arista?



Imaginen que de un momento a otro, **se crea la arista** $(7, 3)$.

Tenemos que unir los elementos de G_1 con los de G_2.

¿Qué pasa si justo llega una nueva arista?



$$G_1 \cup G_2$$

$$¿|G_1|? = 8$$

La estructura Union-Find es súper adaptable a lo que queramos.

Como mínimo debe soportar lo siguiente:

1. Unión de dos conjuntos que son diferentes (***union***).
2. Encontrar a que conjunto pertenece un elemento (***find***).

Mucho 🗣️ 🗣️ 🗣️ 🗣️ 🗣️ poco 🧑‍💻 🧑‍💻 (ejercicio 1, la venganza)

Resolvamos este problema mientras aprendemos sobre el Unión-Find.

Complejidad

	Union	**Find**
Union-Find	$O(log_2(n))$	$O(log_2(n))$

Distancia en un grafo (retomando la clase pasada 🐌 🐌 🐌) xd

Es común encontrarse ejercicios donde tenemos encontrar la distancia mínima para ir de un lugar a otro.

Primero comenzaremos cuando la distancia de moverse sobre una arista es igual 1. Luego, veremos el caso general.

Código para la distancia mínima sobre un grafo/digrafo con BFS

```cpp
//Usando lista de adyacencia
//¡Cuidado con usar int, puede causar overflow!
vector<int> distancia(n, 1'000'000'000);
void bfs(int vertice) {
    distancia[vertice] = 0;
    queue<int> cola; cola.push(vertice);
    while(!cola.empty()) {
        int enfrente = cola.front();
        cola.pop();
        for(auto vecino: lst_ady[enfrente]) {
            if(distancia[vecino] > distancia[enfrente] + 1) {
                distancia[vecino] = distancia[enfrente] + 1;
                cola.push(vecino);
            }
        }
    }
}
```

¿Qué hacemos si la distancia de la arista varía?

Se debe aplicar una técnica similar a las ya vistas, pero primero hay que saber como almacenar esto en una matriz y lista de adyacencia. A estos grafos y digrafos que poseen diferentes pesos/distancias en las aristas, los llamaremos *ponderados*.

Matriz de adyacencia para un grafo ponderado

```
//n = |V|, m = |E|.
int n, m;
cin >> n >> m;
//¡Cuidado con usar int, puede causar overflow!
//NO_EXISTE reemplaza al anterior 0 y debe ser mayor que el max valor de todos los w.
int NO_EXISTE = 1'000'000'007;
vector<vector<int>> mtz_ady(n, vector<int>(n, NO_EXISTE));
for(int e = 0; e < m; ++e) {
    //u, v son aristas y w es el peso/distancia.
    int u, v, w;
    cin >> u >> v >> w;
    //Comentar si u,v toman valores entre [0..n-1].
    u--; v--;
    mtz_ady[u][v] = w;
    //Comentar si es digrafo.
    mtz_ady[v][u] = w;
}
```

Lista de adyacencia para un grafo ponderado

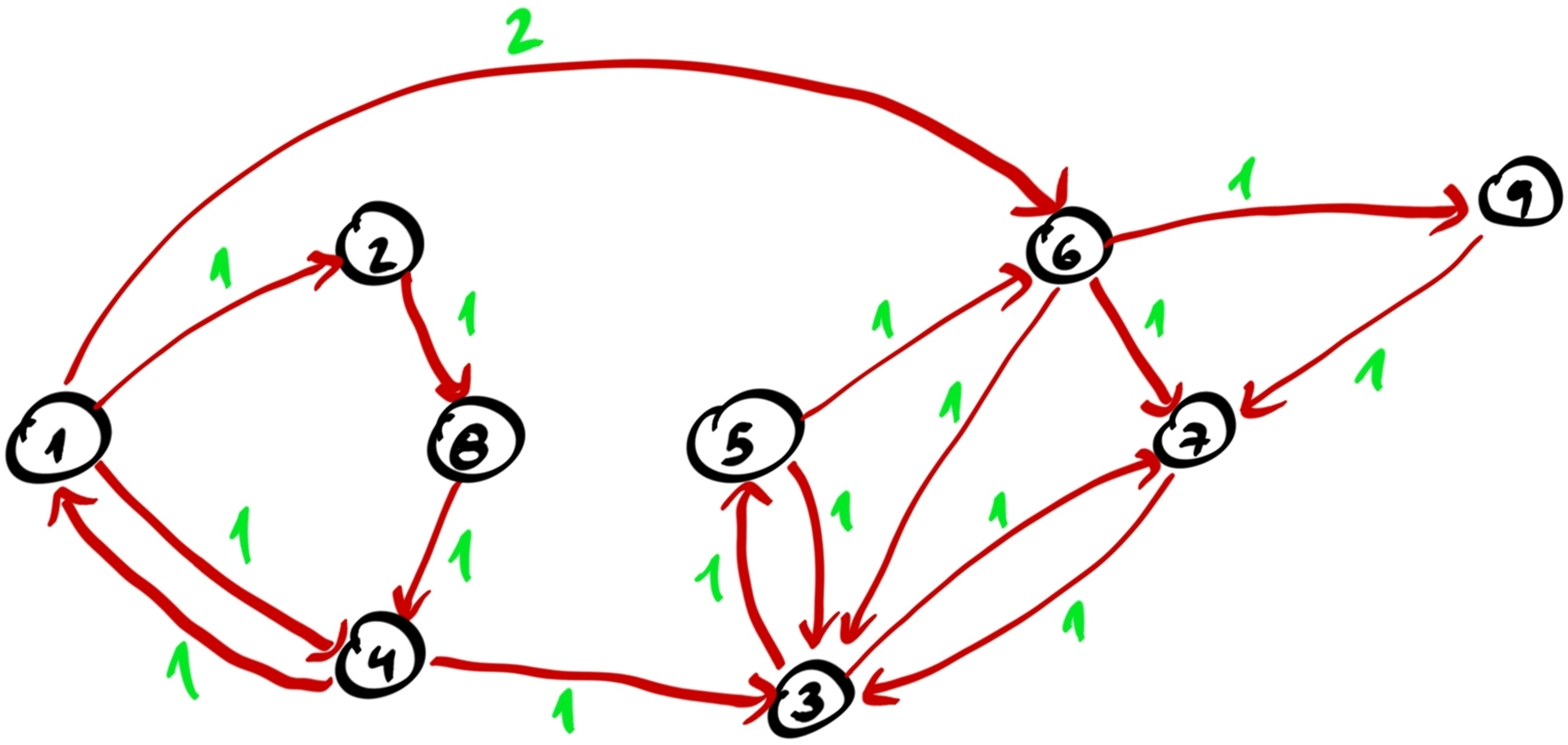
```
//n = |V|, m = |E|.
int n, m;
cin >> n >> m;
vector<vector<pair<int, int>>> lst_ady(n);
for(int e = 0; e < m; ++e) {
    //u, v son aristas y w es el peso/distancia.
    int u, v, w;
    cin >> u >> v >> w;
    //Comentar si u,v toman valores entre [0..n-1].
    u--; v--;
    lst_ady[u].push_back({v,w});
    //Comentar si es digrafo.
    lst_ady[v].push_back({u,w});
}
```

Algoritmo de Dijkstra

Sirve para la determinación del camino más corto **desde un vértice de origen al resto de vértices** de un grafo o digrafo ponderado. Se basa en una estrategia *greedy*.

Antes de programarlo, pensemos como podría funcionar basándonos en el caso particular cuando el peso/distancia en cada arista era 1.

Camino mínimo en este digrafo, iniciando en el vértice 1



Código Dijkstra sobre un grafo/digrafo ponderado

```
//Usando lista de adyacencia.
//¡Cuidado con usar int, puede causar overflow!
vector<int> distancia(n, 1'000'000'000);
void dijkstra(int vertice) {
    distancia[vertice] = 0
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> min_heap;
    min_heap.push({distancia[vertice], vertice});
    while(!min_heap.empty()) {
        auto [distancia_enfrente, enfrente] = min_heap.top();
        min_heap.pop();
        if(distancia_enfrente > distancia[enfrente]) continue;
        for(auto [vecino, peso_vecino]: lst_ady[enfrente]) {
            if(distancia[vecino] > distancia_enfrente + peso_vecino) {
                distancia[vecino] = distancia_enfrente + peso_vecino;
                min_heap.push({distancia[vecino], vecino});
            }
        }
    }
}
```

Complejidad

	Matriz	Lista
Dijkstra	$O(n^2 \cdot log_2(n))$	$O((n+m) \cdot log_2(n))$

Ejercicio en vivo 2 🧑‍💻 🧑‍💻

Resolviendo este ejercicio usando Dijsktra.

¿Siempre debo usar Dijkstra?

Dijkstra no puede usarse cuando existen pesos negativos.

¿Por qué?

- Se rompe la invariante con la que aseguramos la correctitud del procedimiento.

¿Qué hacer en ese caso?

- Usar Bellman-Ford o Floyd-Warshall (**si el grafo no posee ciclos negativos**).

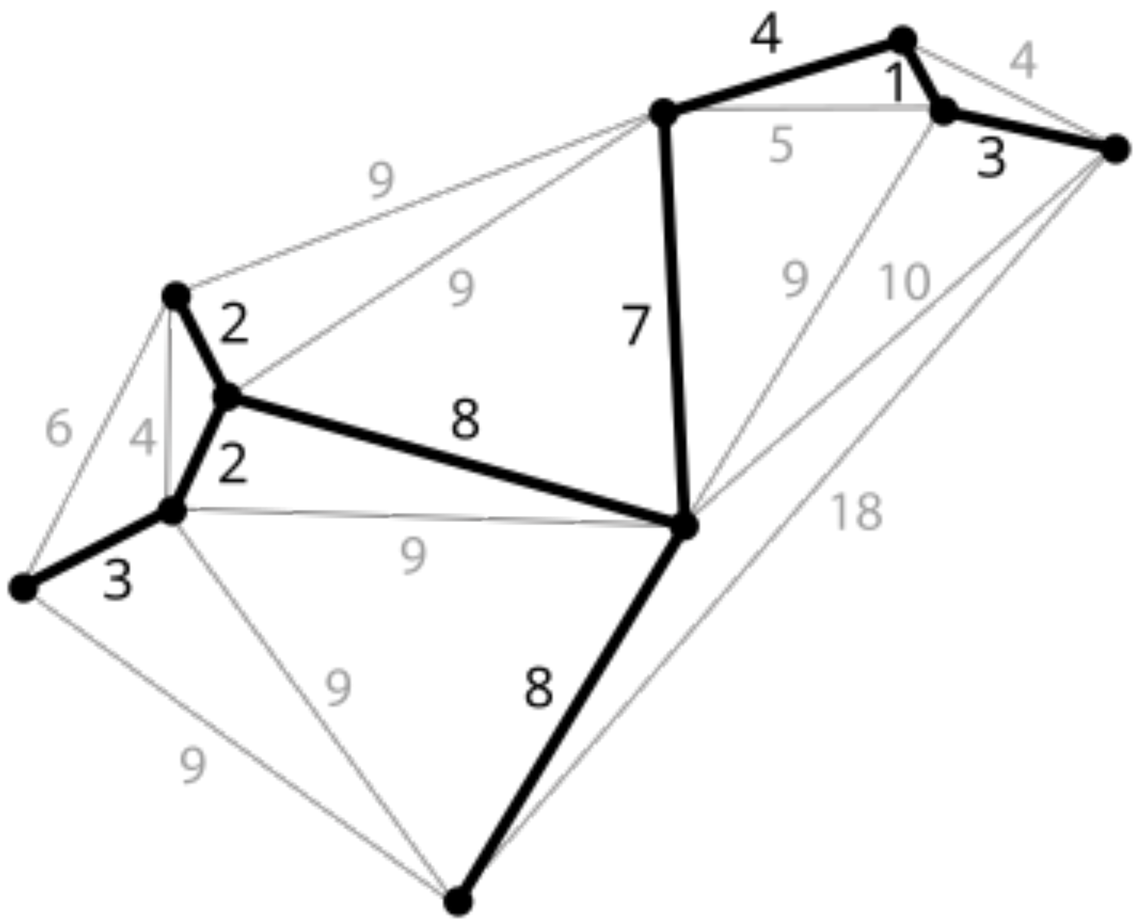
Una nueva forma de representar un grafo 🤓 👆

Podemos representar grafos ponderados utilizando sus aristas. Es útil para poder crear el árbol de cobertura mínimo del grafo.

Lista de aristas ordenadas para representar un grafo ponderado

```
//n = |V|, m = |E|.
int n, m;
cin >> n >> m;
vector<pair<int, pair<int, int>>> lst_aristas(m);
for(int e = 0; e < m; ++e) {
    //u, v son aristas y w es el peso/distancia.
    int u, v, w;
    cin >> u >> v >> w;
    //Comentar si u, v toman valores entre [0..n-1].
    u--; v--;
    lst_aristas[e] = {w, {u,v}};
}
//Ordenar las aristas en orden creciente, por el peso.
sort(lst_aristas.begin(), lst_aristas.end());
```

Árbol de cobertura mínimo (Minimum Spanning Tree)



1. Obtener la lista de aristas ordenadas del grafo.
2. Crear una estructura Union-Find vacía.
3. Crear un grafo ponderado vacío (matriz/lista de adyacencia).
4. Por cada arista en la lista de aristas:
   4.1. **Si los vértices u y v NO están conectados en el Union-Find**, creamos la arista en el grafo ponderado y seguimos a la siguiente.
   4.2 **Si no**, ignoramos la arista y vamos a la siguiente.

¿Cuál es la gracia del árbol de cobertura mínimo?

Obtenemos un árbol cuya suma de pesos de aristas es el mínimo posible.

¿Cuál es la gracia del árbol de cobertura mínimo?

¿Cuál es la gracia del árbol de cobertura mínimo?

1. El árbol de cobertura mínimo (MST) del grafo es único, si y sólo sí todos los pesos de las aristas son distintos. En otro caso, existen múltiples MST posibles.
2. Es un árbol, por lo tanto existe un único camino entre todo par de nodos (DFS/BFS for the win para recorrerlo sin miedo ya que tendrán n nodos y $n - 1$ aristas).
3. El peso de cada arista de este árbol es el mínimo entre todos los árboles de cobertura posibles a crear del grafo.

¿Cuál es la gracia del árbol de cobertura mínimo?

NO CONFUNDIR CON EL ÁRBOL QUE GENERA EL ALGORITMO DE DIJKSTRA.

Ejercicio en vivo 3 🧑‍💻 🧑‍💻

Resolviendo este ejercicio usando el algoritmo de Kruskal.

Complejidad

	Kruskal
MST	$O(m \cdot log_2(m \cdot n))$

Transformando problemas a grafos

Muchas veces podemos tener un problema de grafos y transformarlo en otro grafo más sencillo, como también tener un problema que no es de grafos y transformarlo a grafos.

Transformando problemas a grafos (vivo 📝)

¿Este problema era un grafo?

Ejercicio en vivo 4 🧑‍💻 🧑‍💻

Resolviendo este ejercicio transformando una grilla a un grafo ponderado.