# RECORRIDO EN ANCHURA

De un Grafo NO dirigido

Permite encontrar caminos mínimos con el menor número de aristas desde un vértice los demás alcanzables desde él.

UNIVERSIDAD COMPLUTENSE MADRID

ALBERTO VERDEJO

# Recorrido en anchura

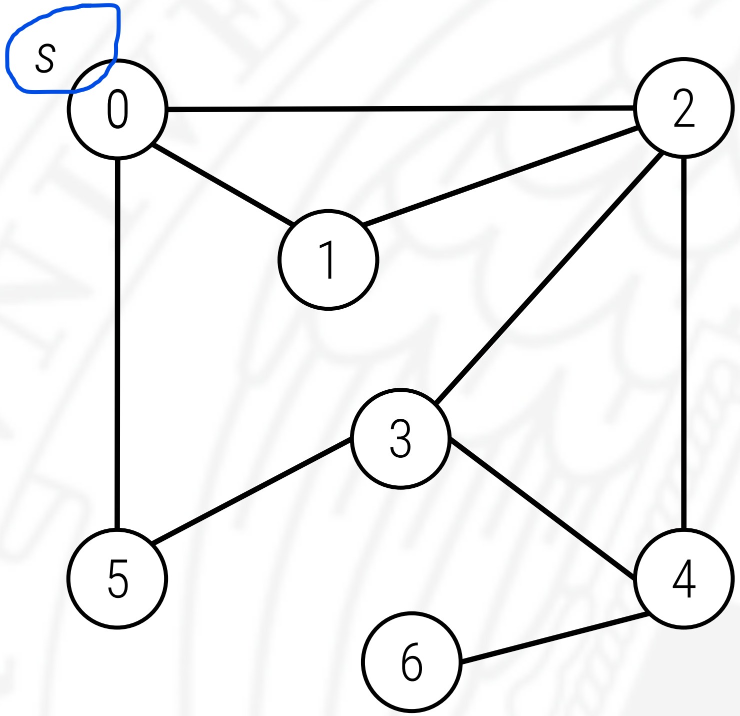
- En ocasiones estamos interesados en encontrar el camino más corto desde un origen *s* a otro vértice *v* (o a todos los vértices conectados a *s*).
- El recorrido en anchura (en inglés, *breadth-first search*) logra eso: primero visita todos los vértices alcanzables siguiendo una arista (a distancia 1); luego visita todos los vértices alcanzables utilizando dos aristas (a distancia 2); y así sucesivamente.

BFS

Los que son adyacentes supongo.

- Para lograrlo utiliza una cola donde guardar los vértices alcanzados pero que aún no se han explorado sus adyacentes.

# Recorrido en anchura

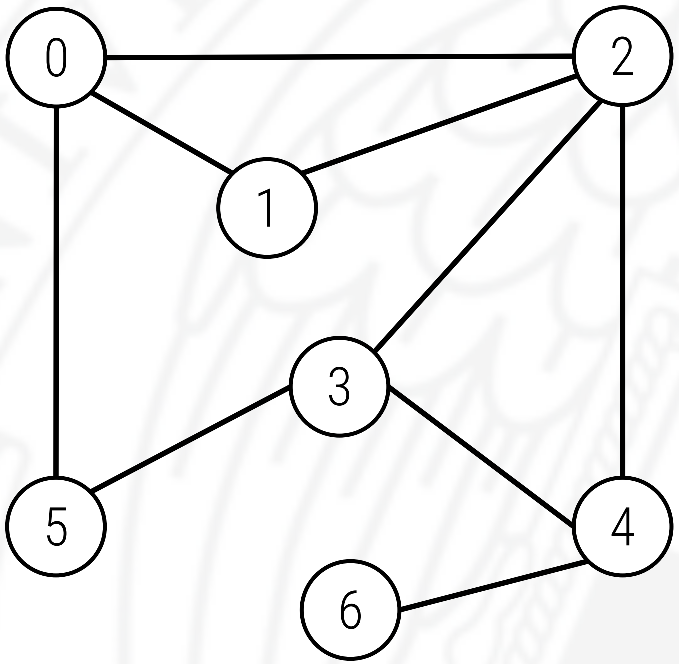



A qué distancia está cada vértice del origen.

| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | F | – | – |
| 1 | F | – | – |
| 2 | F | – | – |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Cola:

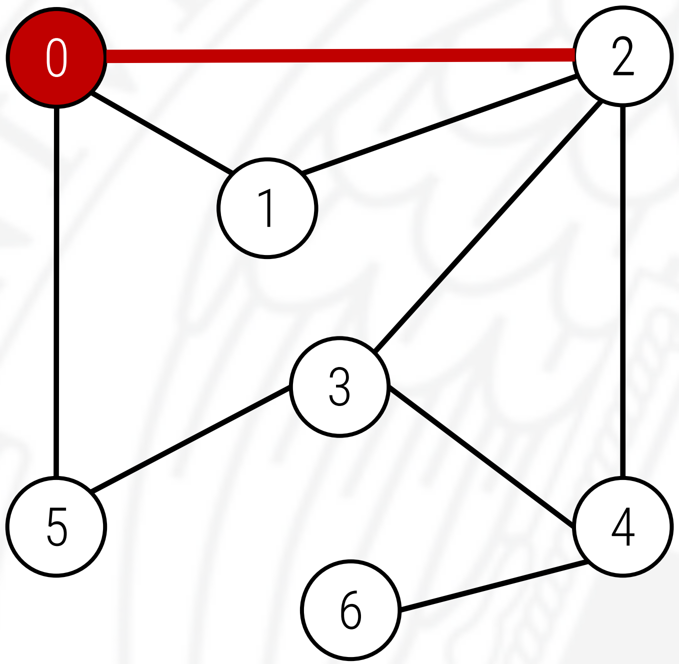
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | F | – | – |
| 2 | F | – | – |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Cola: 0

# Recorrido en anchura

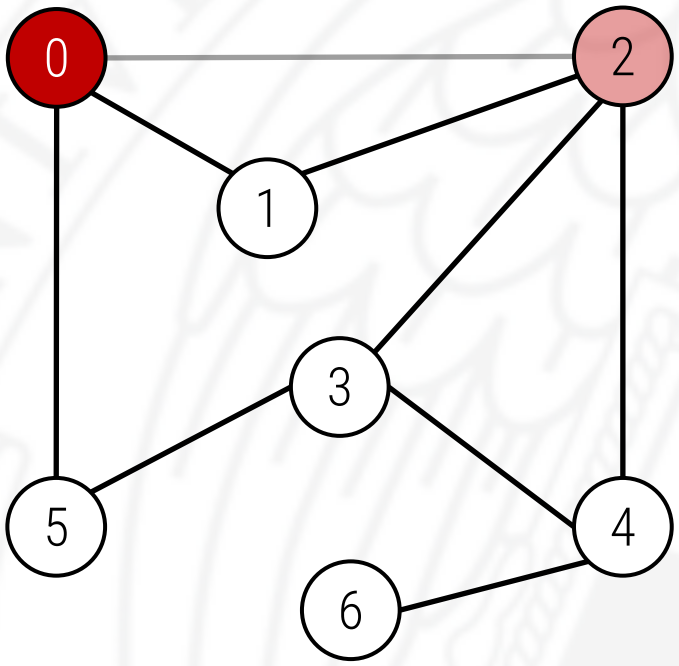



| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | F | – | – |
| 2 | F | – | – |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Cola: 0

Mientras que la cola NO sea vacía, se saca el primero y recorremos sus vértices adyacentes NO visitados.

# Recorrido en anchura

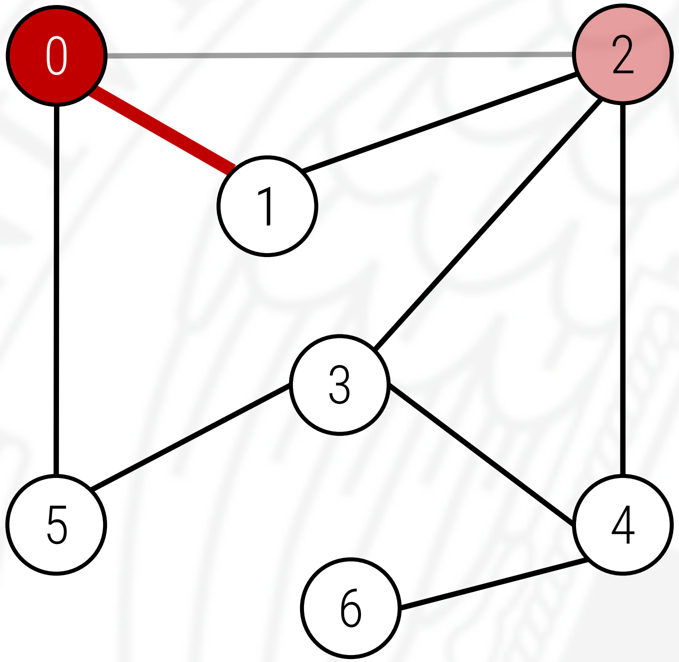



| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | F | – | – |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

No estaba visitado

Cola: 2

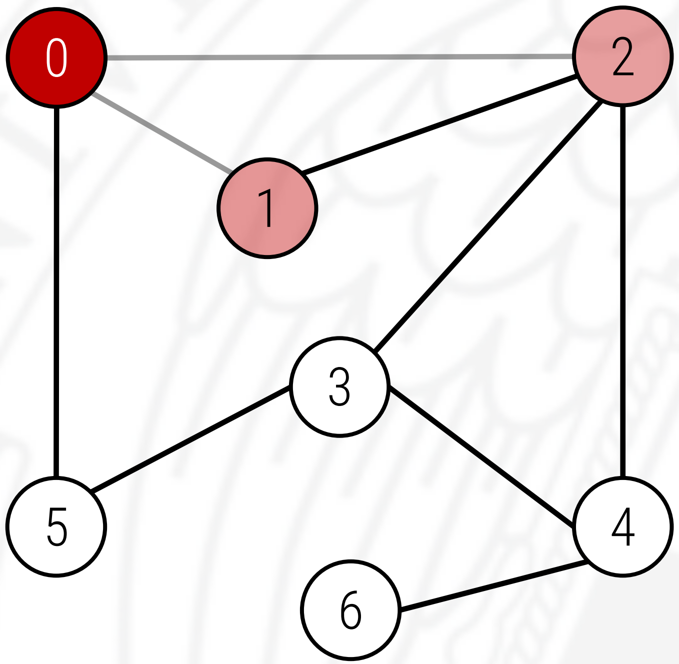
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | F | – | – |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Cola: 2

# Recorrido en anchura

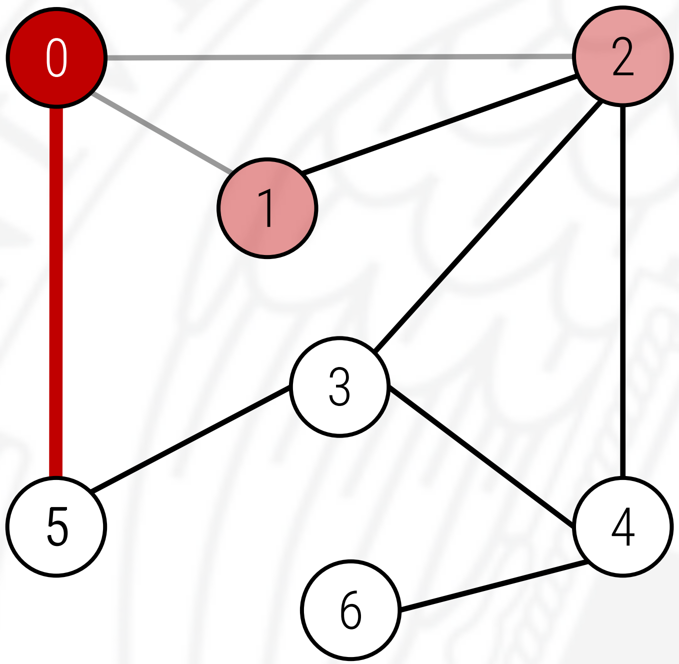


| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Lo metemos en la cola porque aun no está visitado.

Cola: 2 1

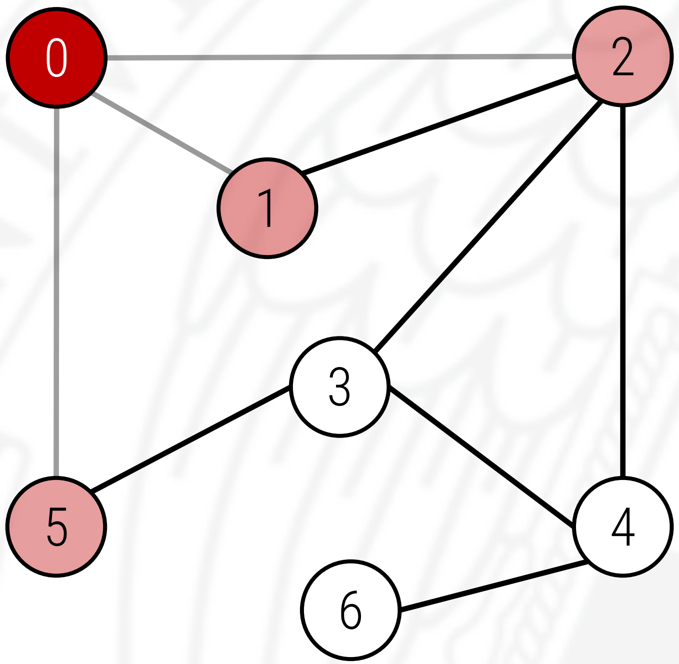
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | F | – | – |
| 6 | F | – | – |

Cola: 2 1

# Recorrido en anchura

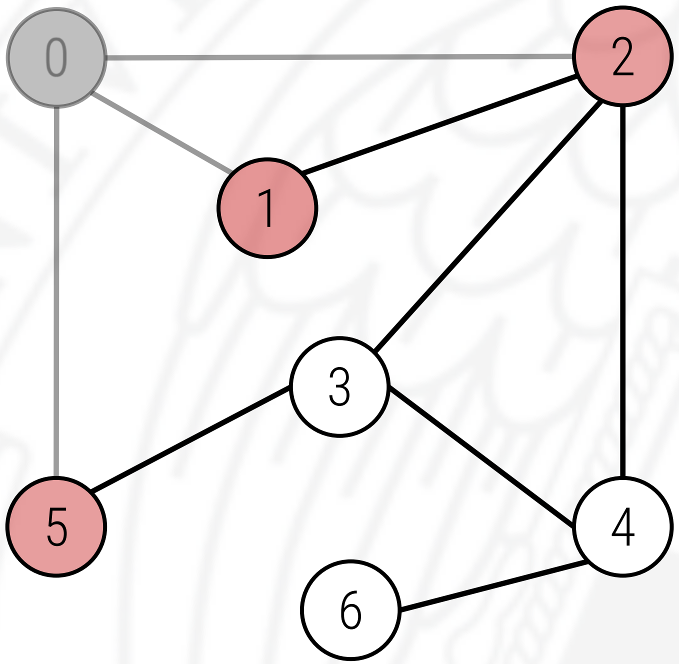



| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 2 1 5

# Recorrido en anchura

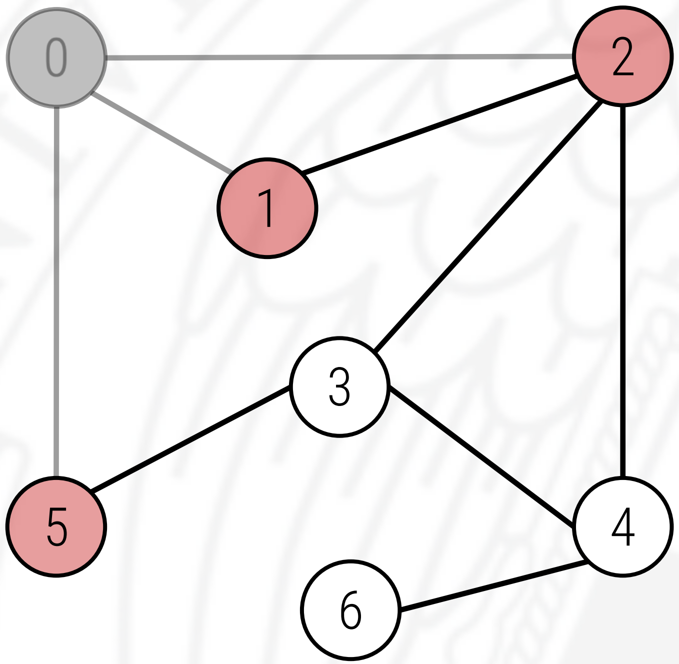
TURNO DE VISITAR LOS ADYACENTES DEL 2.




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 2 1 5

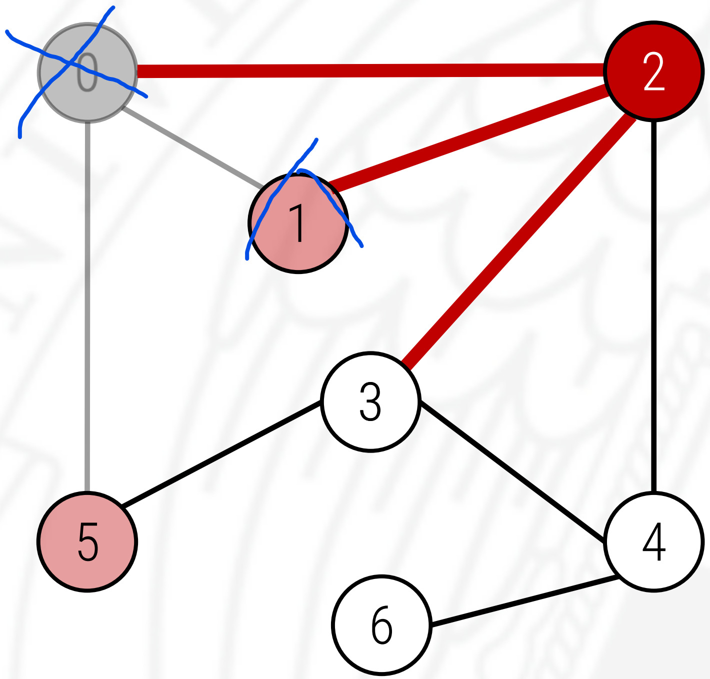
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 2 1 5

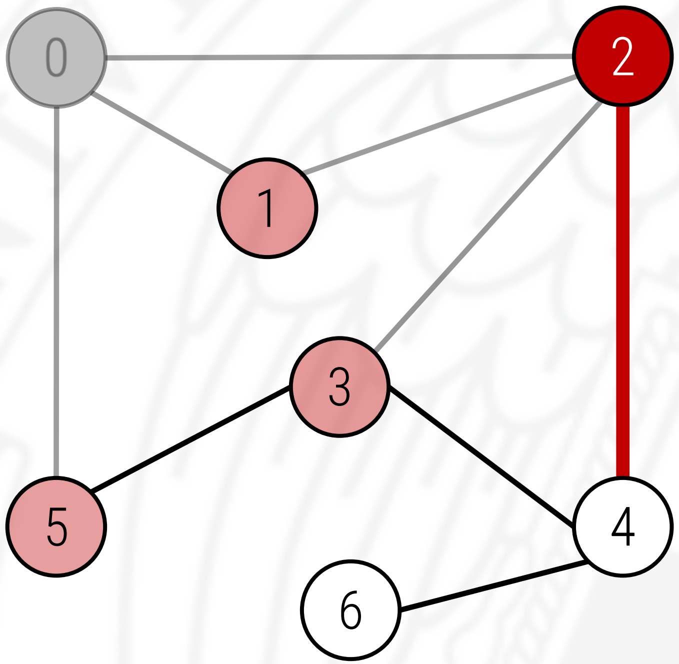
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | F | – | – |
| 4 | F | – | – |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 1 5

# Recorrido en anchura

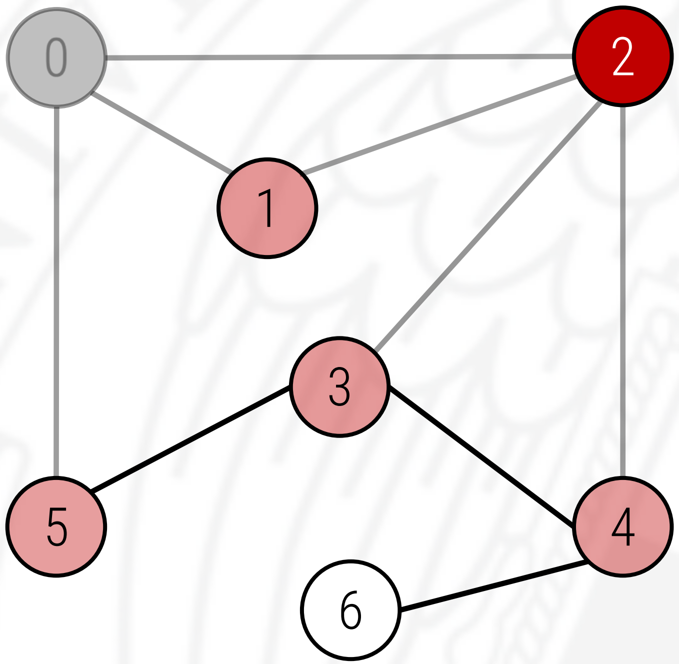



| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | F | – | – |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Dos aristas lo separan del origen.

Cola: 1 5 3

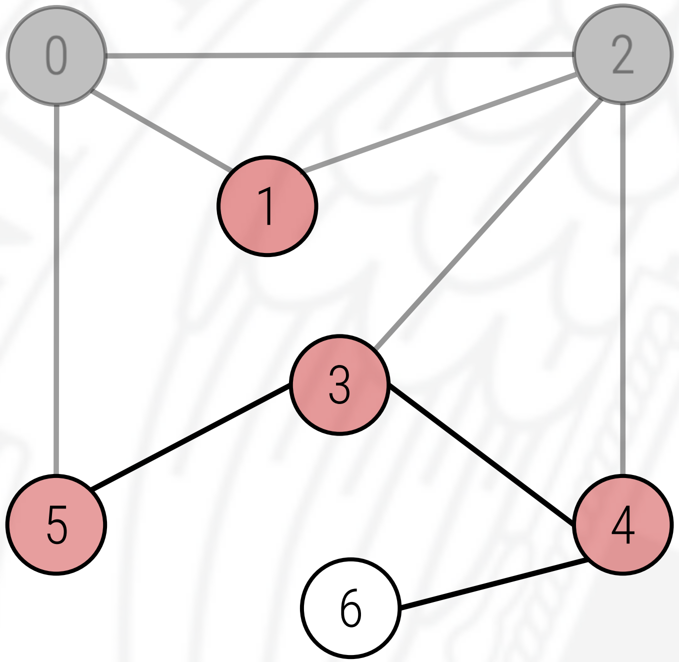
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 1 5 3 4

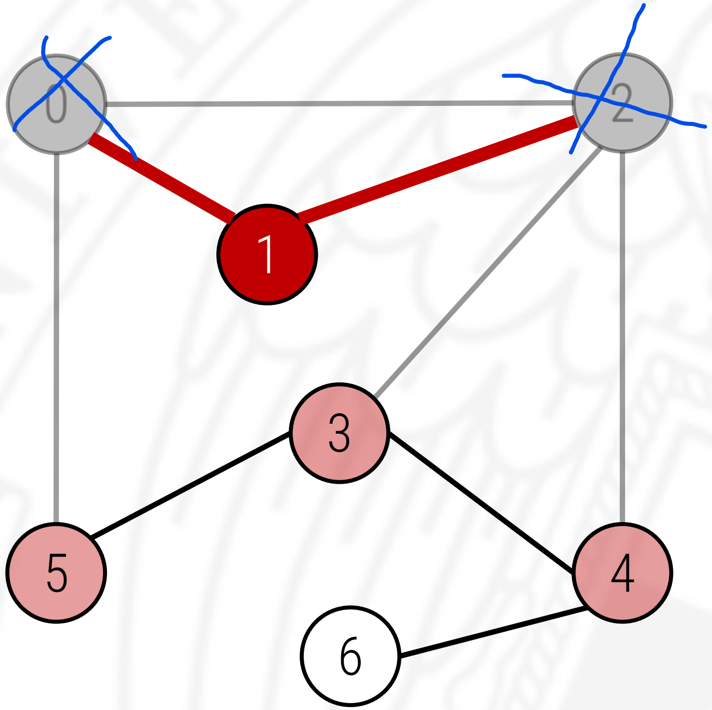
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 1 5 3 4

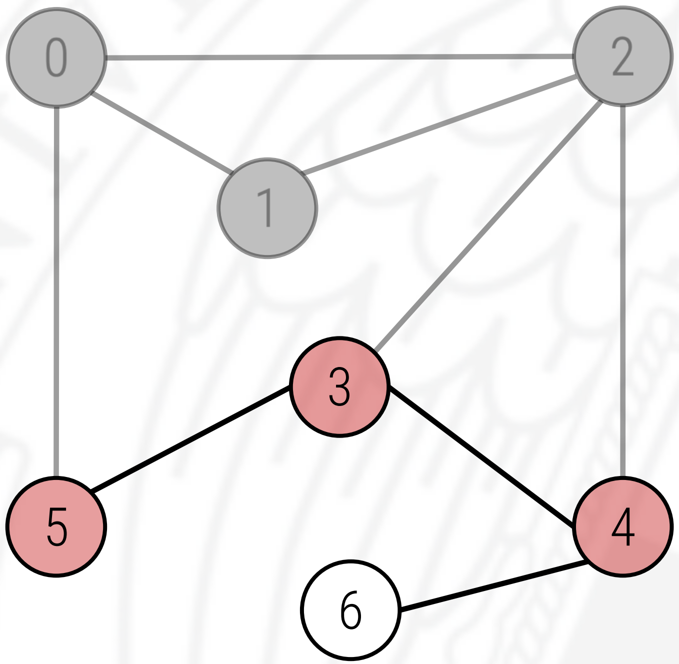
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 5 3 4

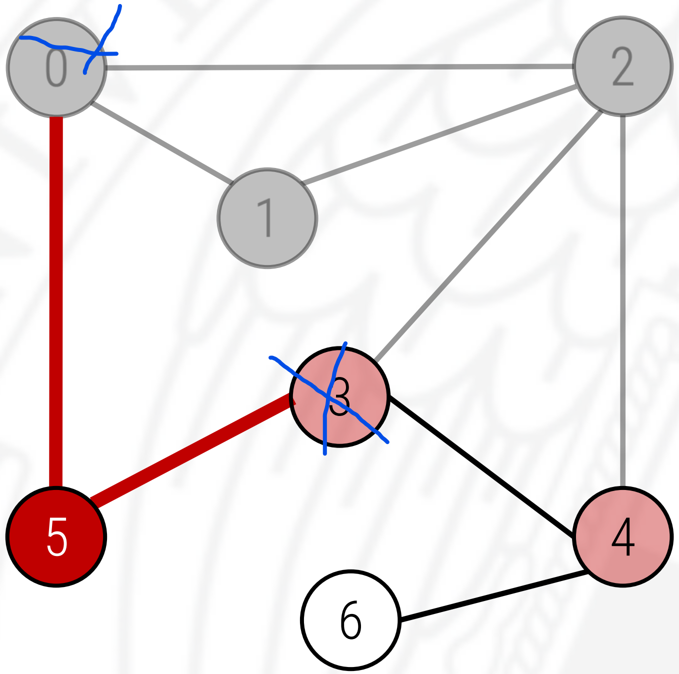
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 5 3 4

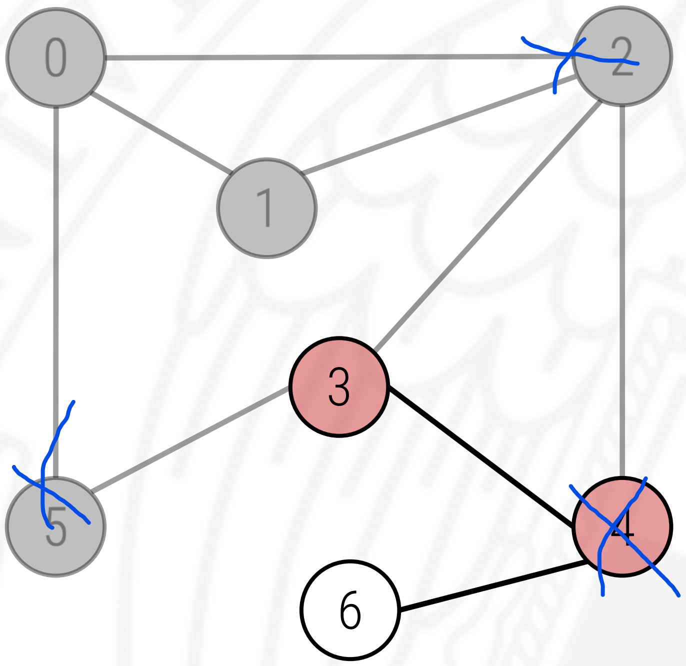
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 3 4

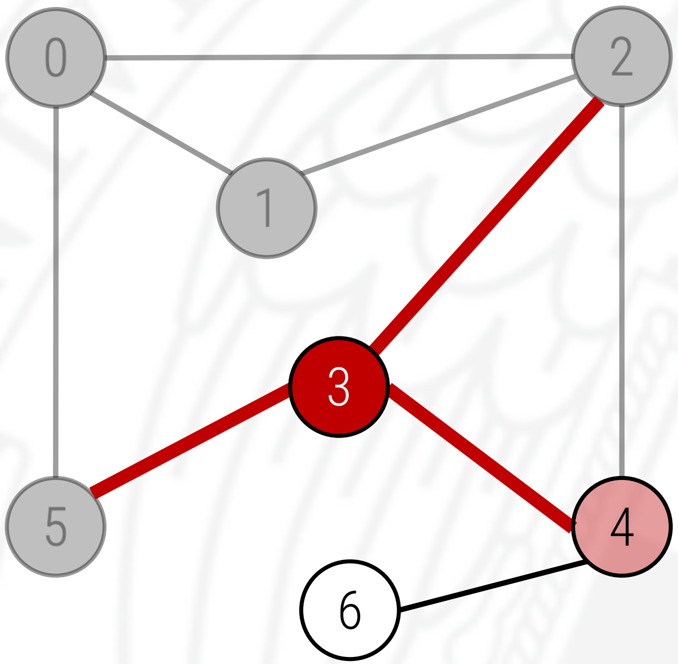
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 3 4

# Recorrido en anchura

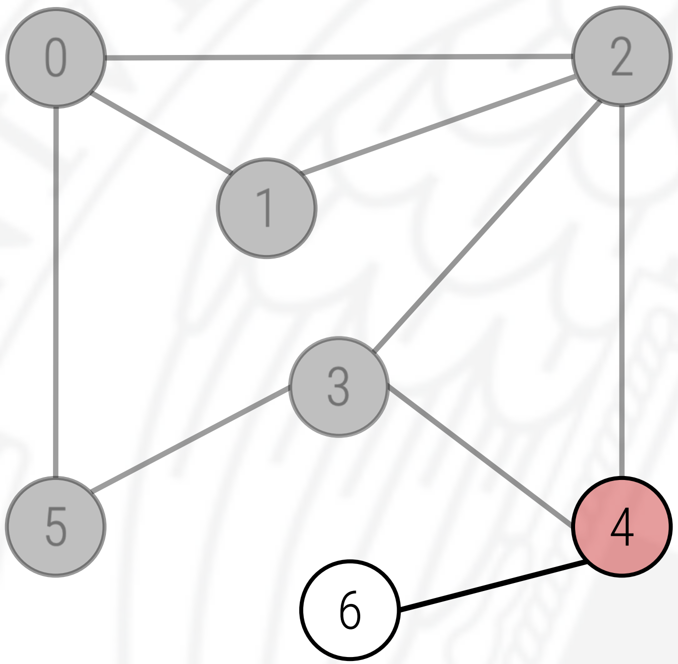



| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 4

# Recorrido en anchura

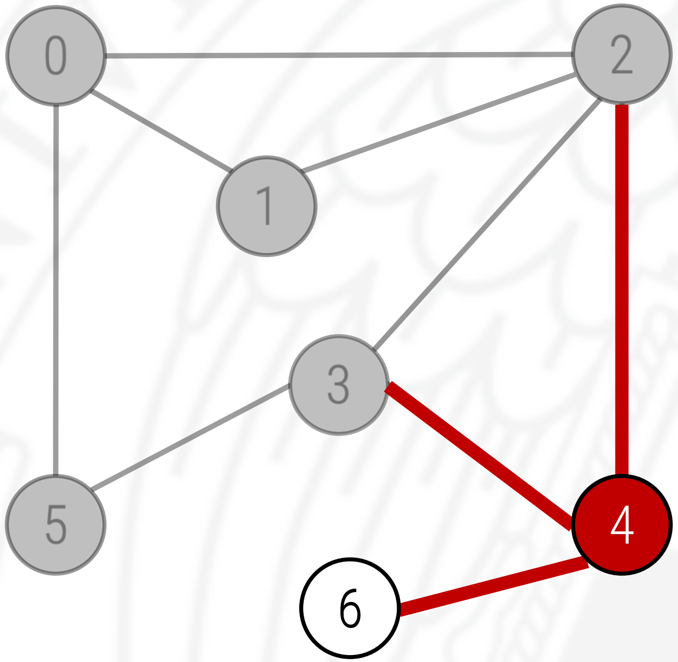
sacamos el 4 de la cola y visitamos su adyacente.




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola: 4

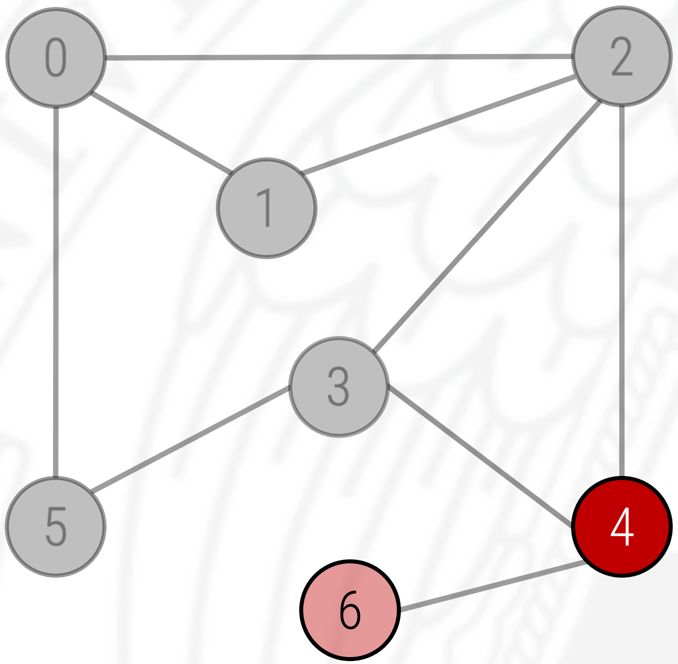
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | F | – | – |

Cola:

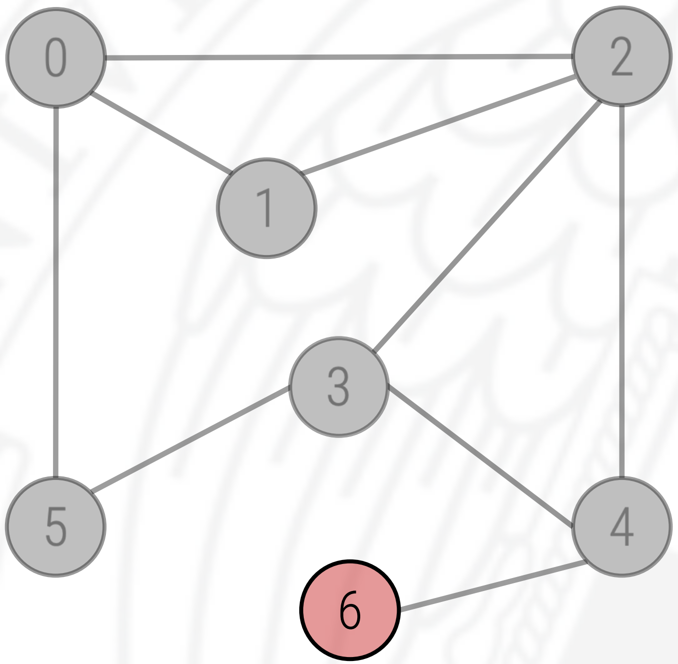
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | T | 4 | 3 |

Cola: 6

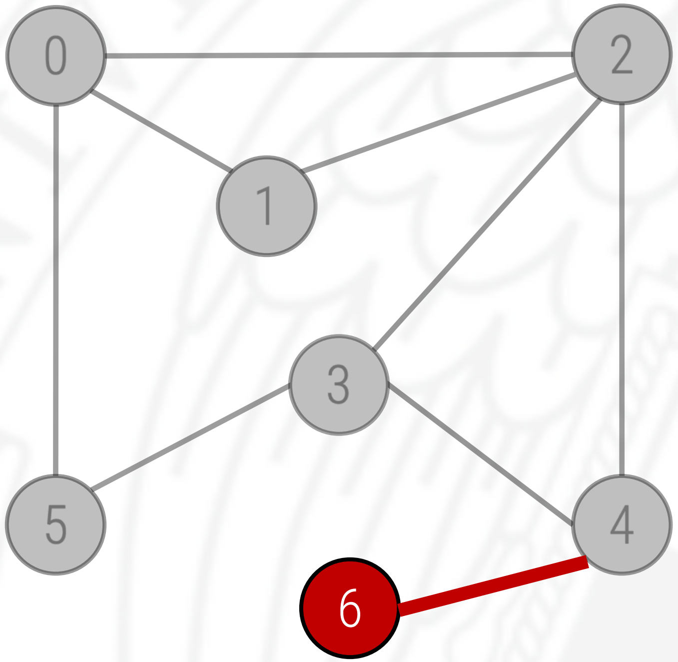
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | T | 4 | 3 |

Cola: 6

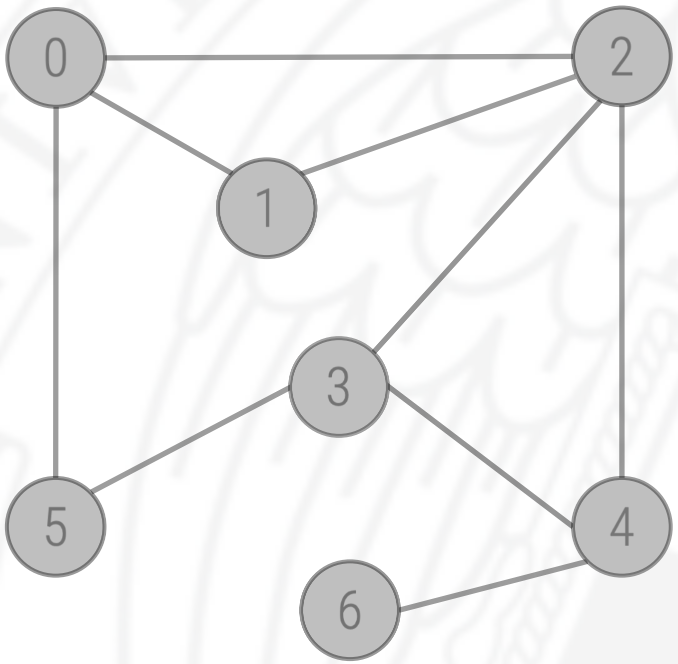
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | T | 4 | 3 |

Cola: 6

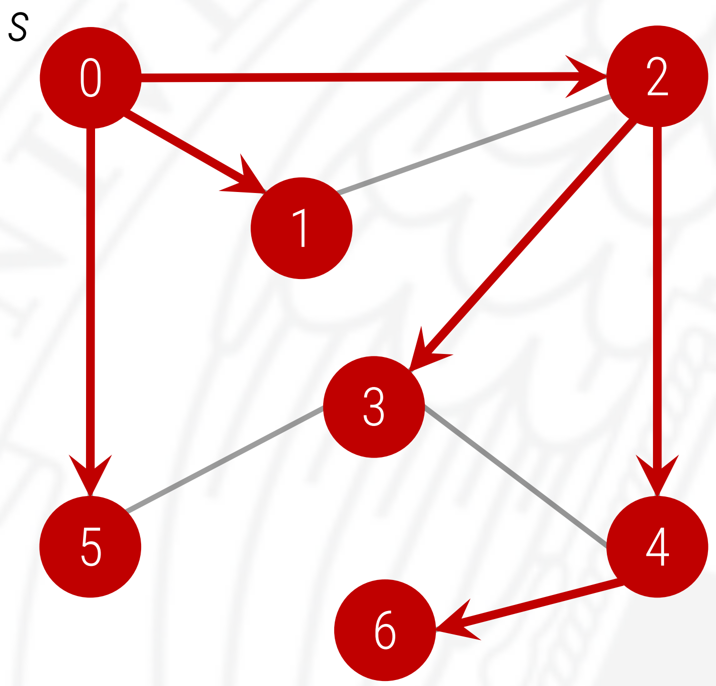
# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | T | 4 | 3 |

Cola:

# Recorrido en anchura




| v | visitado | anterior | distancia |
|---|---|---|---|
| 0 | T | – | 0 |
| 1 | T | 0 | 1 |
| 2 | T | 0 | 1 |
| 3 | T | 2 | 2 |
| 4 | T | 2 | 2 |
| 5 | T | 0 | 1 |
| 6 | T | 4 | 3 |

Cola:

Como el grafo es conexo hemos conseguido visitar todos los vértices.

# Caminos más cortos desde un origen

```
class CaminoMasCorto {
public:
   CaminoMasCorto(Grafo const& g, int s) : visit(g.V(), false),
                                            ant(g.V()), dist(g.V()), s(s) {
      bfs(g);
   }

   // ¿hay camino del origen a v?
   bool hayCamino(int v) const {
      return visit[v];
   }
```

orgien

vector de visitados.

apuntar los anteriores

vector de distancias

origen

Recorrido en ANCHURA.

Si hay camino es que esta marcado.

# Caminos más cortos desde un origen

```
// número de aristas entre s y v
int distancia(int v) const {
   return dist[v];
}
```

So esta marcado como visitado podemos saber a qué distancia esta-

```
// devuelve el camino más corto desde el origen a v (si existe)
Camino camino(int v) const {
    if (!hayCamino(v)) throw std::domain_error("No existe camino");
    Camino cam;
    for (int x = v; x != s; x = ant[x])
        cam.push_front(x);
    cam.push_front(s);
    return cam;
}
```

De v hasta el anterior hasta llegar a s.

# Caminos más cortos desde un origen

```
private:
    std::vector<bool> visit; // visit[v] = ¿hay camino de s a v?
    std::vector<int> ant;  // ant[v]  = último vértice antes de llegar a v
    std::vector<int> dist; // dist[v] = aristas en el camino s-v más corto
    int s;
```

# Caminos más cortos desde un origen

Cada vértice se añade solo 1 vez a la cola. Cuando sale de la cola se recorren los adyacentes. esto es lineal en el número de aristas. A esto se le suma el recorrer todos los vertices a través del for. Por tanto O(V+A)

```
void bfs(Grafo const& g) {
    std::queue<int> q;  Cola
    dist[s] = 0; visit[s] = true;  origen distancia 0 y visitado
    q.push(s);   Añade a la cola el origen
    while (!q.empty()) {  Mientras que la cola NO sea vacía...
        int v = q.front(); q.pop();  Lo eliminamos de la cola
 V      for (int w : g.ady(v)) {  Recorremos todos los adyacentes.
            if (!visit[w]) {  Si no están visitados
                ant[w] = v; dist[w] = dist[v] + 1; visit[w] = true;
                q.push(w);
            }
        }
    }
}
};
```

Están a una distancia 1 + que v y se marcan como visitados, y los metemos en la colita.

# Recorrido en anchura: corrección y coste

- El recorrido en anchura encuentra caminos más cortos a todos los vértices conectados con el origen $s$, en un tiempo en $O(V + A)$ (en el caso peor).