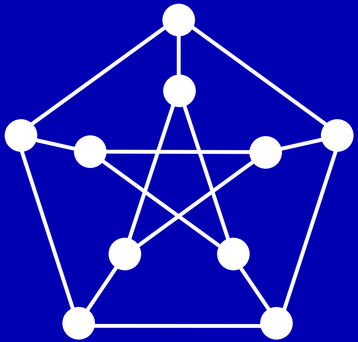
Gianluca Della Vedova

- Large-Scale Graph Algorithms
- Ufficio U14-2041
- https://www.unimib.it/gianluca-della-vedova
- gianluca.dellavedova@unimib.it
- https://github.com/gdv/large-scale-graph-algorithms
- Everything at https://elearning.unimib.it/

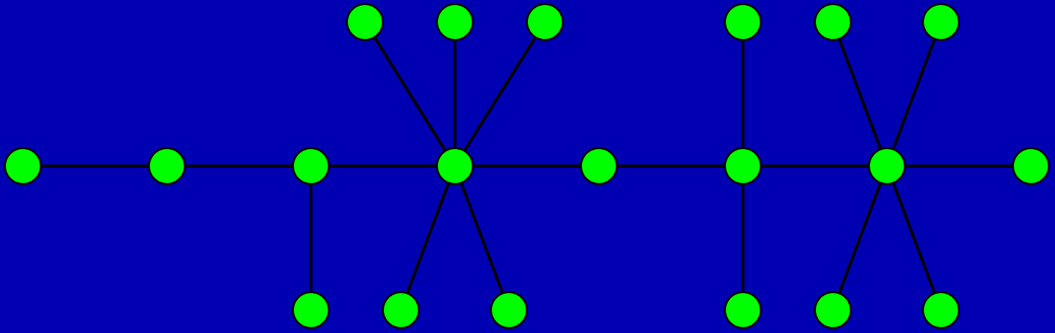
Example



Notation

- number of vertices: n
- number of edges/arcs: m

Better representation



Almost a path

- Compact representation

Breadth-first visit

Data: graph G, vertex root
Q $\leftarrow$ a queue;
label root as explored;
Q.enqueue(root);
while Q $\neq \emptyset$ **do**
  $v \leftarrow$ Q.dequeue();
  foreach *edge* (v, w) **do**
    if *w is not labeled as explored* **then**
      label w as explored;
      Q.enqueue(w)

Depth-first visit

```
Data: graph G, vertex root
S ← a stack;
S.push(root);
while S ≠ ∅ do
    v ← S.pop();
    if v is not labeled as explored then
        label v as explored;
        foreach edge (v, w) do
            S.push(w)
```

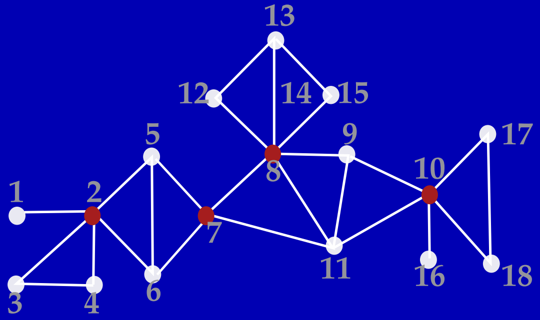
Dijkstra's algorithm

```
Data: graph G, vertex source
Q ← a queue;
foreach vertex v do
    dist[v] ← ∞;
    Q.enqueue(v)
dist[source] ← 0;
while Q ≠ ∅ do
    u ← vertex in Q minimizing dist[u];
    Q.deque(u);
    foreach neighbor v of u still in Q do
        alt ← dist[u] + Graph.Edges(u, v);
        if alt < dist[v] then
            dist[v] ← alt;
            prev[v] ← u;
return dist[], prev[];
```

Dijkstra's algorithm — priority queue

```
Data: graph G, vertex source
Q ← a priority queue;
foreach vertex v do
    dist[v] ← ∞;
    Q.add_with_priority(v, dist[v])
dist[source] ← 0;
while Q ≠ ∅ do
    u ← Q.extract_min;
    foreach neighbor v of u still in Q do
        alt ← dist[u] + Graph.Edges(u, v);
        if alt < dist[v] then
            dist[v] ← alt;
            prev[v] ← u;
            Q.decrease_priority(v, alt)
return dist[], prev[];
```

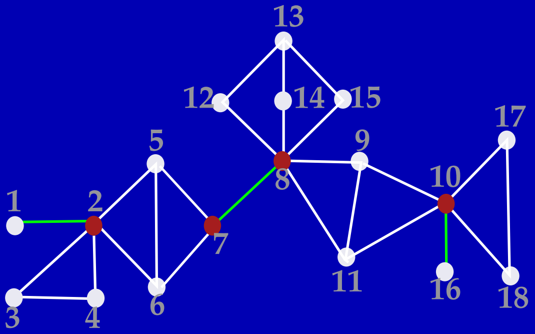
Biconnected components



Find articulation points

```
Data: connected graph G, vertex root
S ← a stack;
S.push((root, nil));
d ← −1;
while S ≠ ∅ do
    d ← d + 1;
    (v, p) ← S.peek();
    if v is not explored then
        label v as explored; parent(v) ← p; depth[v] ← d;
        lowpoint[v] =depth[v];
        foreach edge (v, w) do
            S.push((w, v));
    else
        lowpoint[v] =
         min {depth[v], min_{w∈N(v),w≠p} {depth[w]}, min_{w∈N(v),parent(w)=v} {lowpoint[w]}}
    d ← d − 1;
    v ← S.pop();
```

2-edge connected components



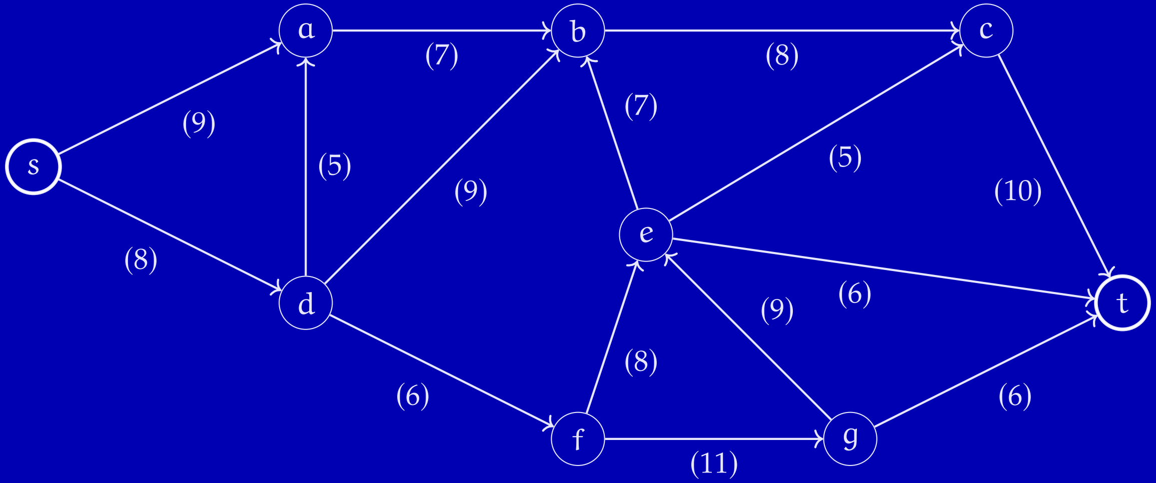
Find bridges

Data: connected graph G, vertex root

$S \leftarrow$ a stack;
S.push((root, nil));
$x \leftarrow -1$;
while $S \neq \emptyset$ **do**
    $(v, p) \leftarrow S$.peek();
    if v *is not explored* **then**
        $x \leftarrow x + 1$;
        label v as explored; $P(v) \leftarrow p$; $num[v] \leftarrow x$;
        foreach *edge* (v, w) **do**
            S.push((w, v));
    else
        $nd[v] = 1 + \sum_{w \in N(v), P(w)=v} nd[w]$;
        $l[v] =$
        $\min \left\{ num[v], \min_{w \in N(v), w \neq p, P(w) \neq v} \{num[w]\}, \min_{w \in children(v)} \{l[w]\} \right\}$;

        $h[v] =$
        $\max \left\{ num[v], \max_{w \in N(v), w \neq p, P(w) \neq v} \{num[w]\}, \max_{w \in children(v)} \{h[w]\} \right\}$;

Max flow



Max flow – Min cut theorem

Let f be a flow of a graph $G = (V, E)$. Then the following three conditions are equivalent:

1. f is a maximum flow
2. the residual graph has no augmenting path
3. there is a cut (S, T) of G such that $c(S, T) = |f|$

Figures

- David Eppstein, Public Domain,
  https://commons.wikimedia.org/w/index.php?curid=10261635

License