Advanced Graph Algorithms II

Strongly Connected Components, Bi-Connectivity, Max Flow

SoftUni Foundation

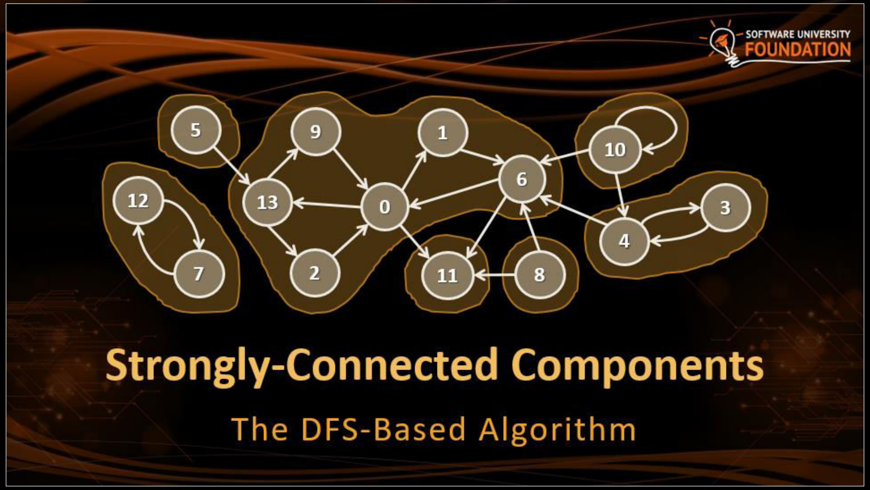
SoftUni Team

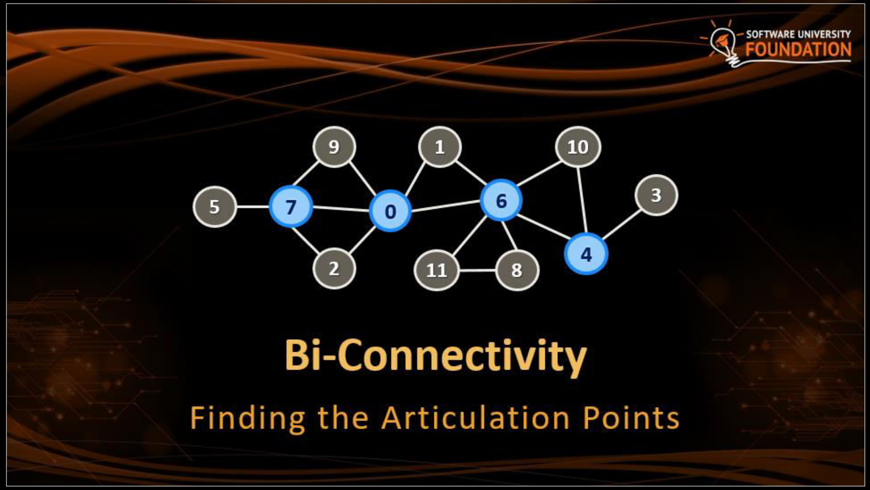
Technical Trainers

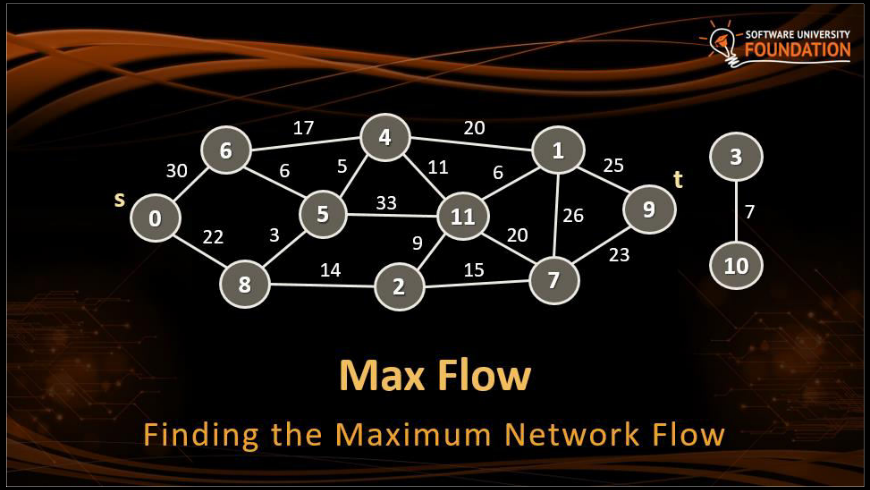
Software University

http://softuni.bg

Table of Contents





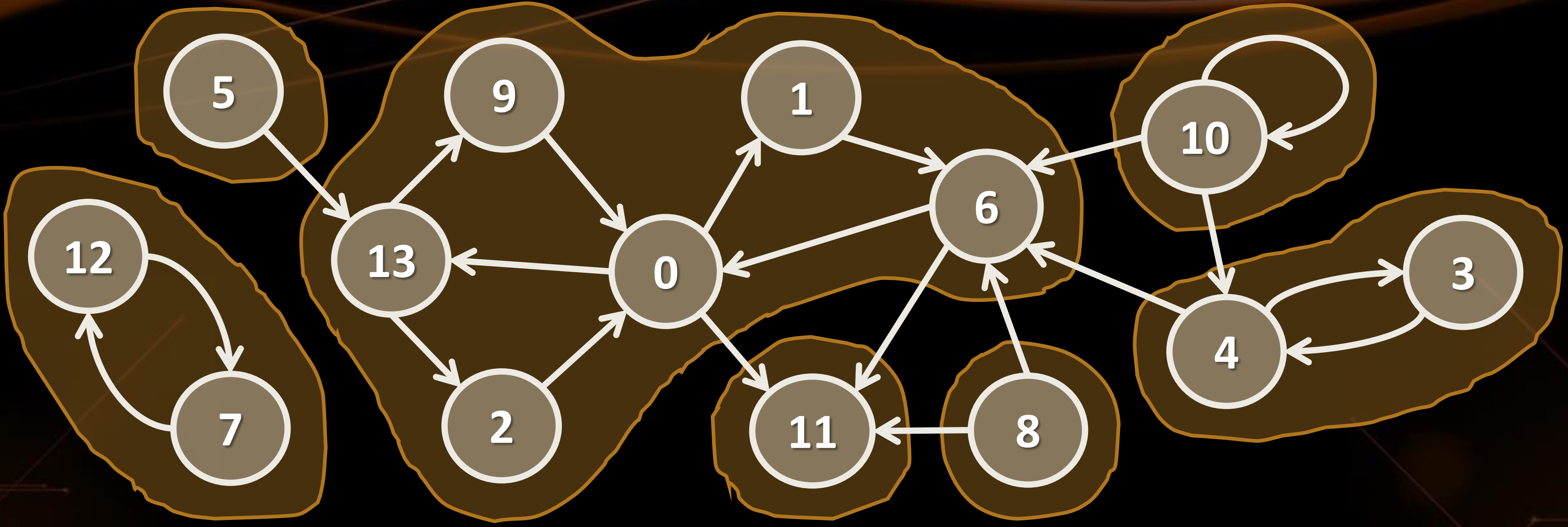


Have a Question?

sli.do

#DsAlgo

SoftUni Foundation

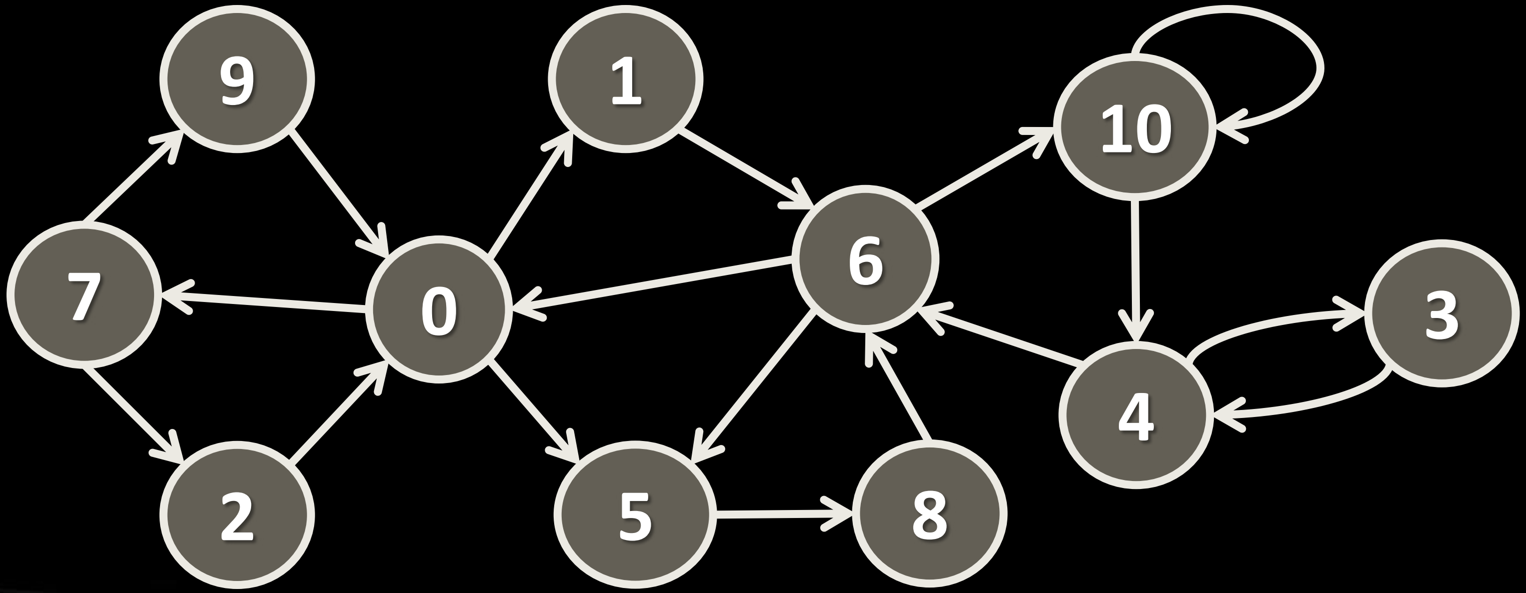


Strongly-Connected Components

The DFS-Based Algorithm

Strongly-Connected Components

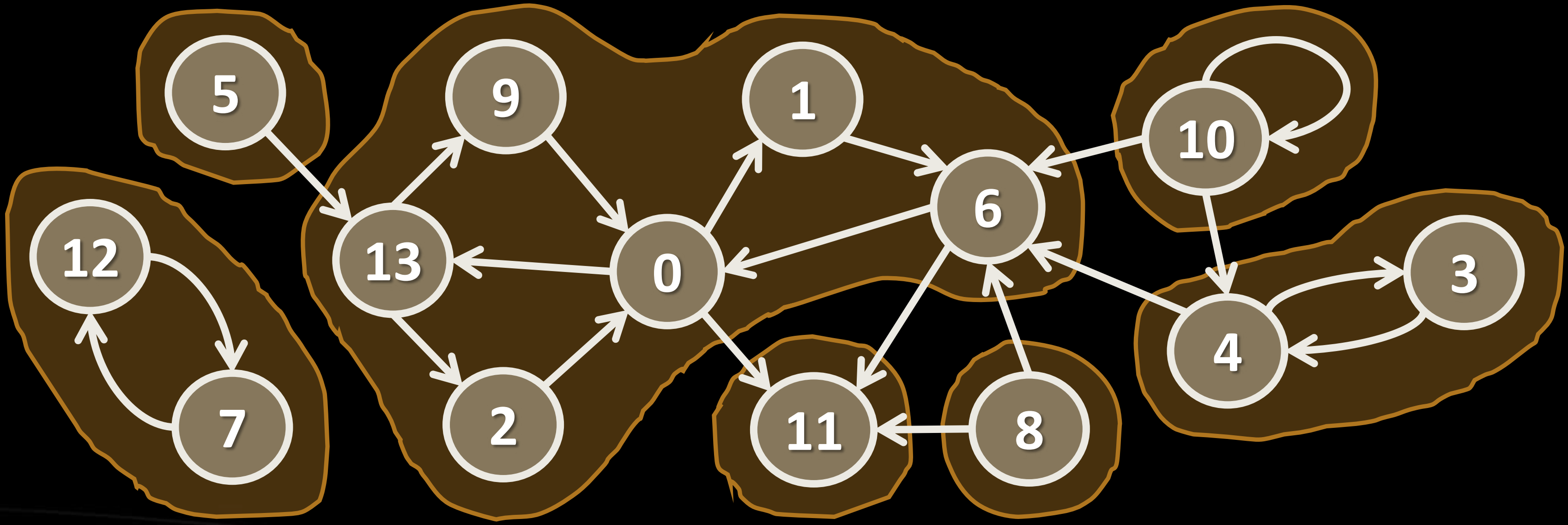
- A directed graph is strongly-connected
  - When every two vertices are connected by path
- Example of strongly-connected graph:



Strongly-Connected Components

SoftUni Foundation

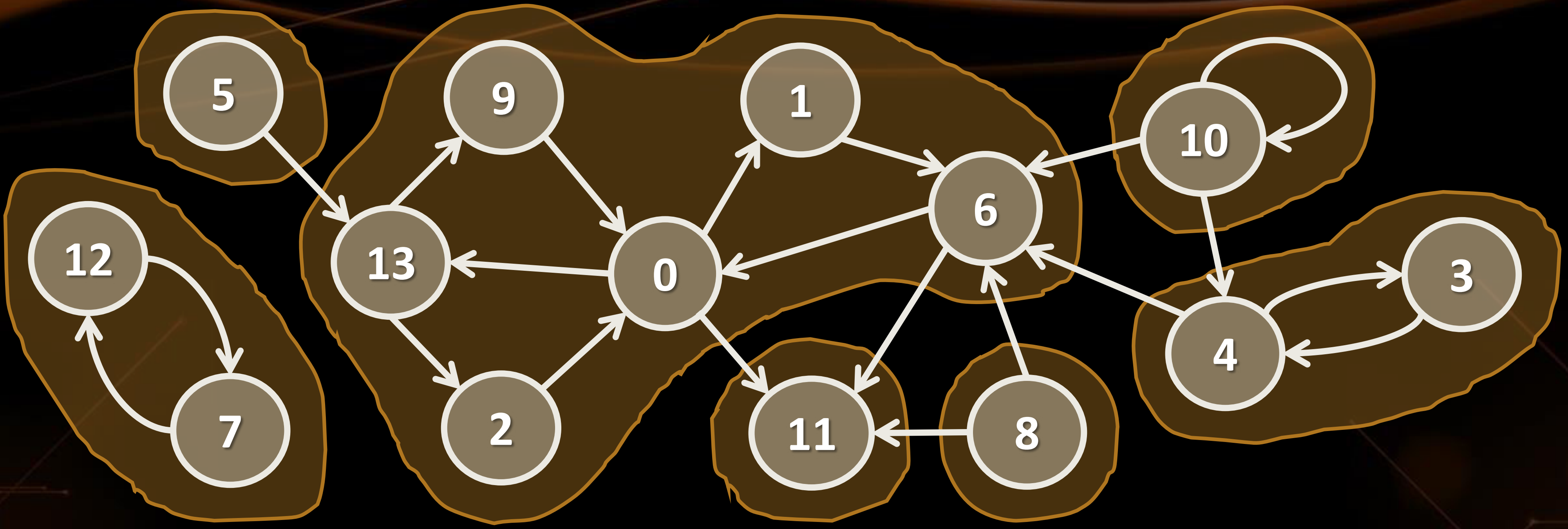
- Strongly-connected component is a maximal strongly-connected subgraph (component with paths between any two nodes)
- A directed graph can be decomposed into strongly-connected components, e.g.



Kosaraju–Sharir Algorithm

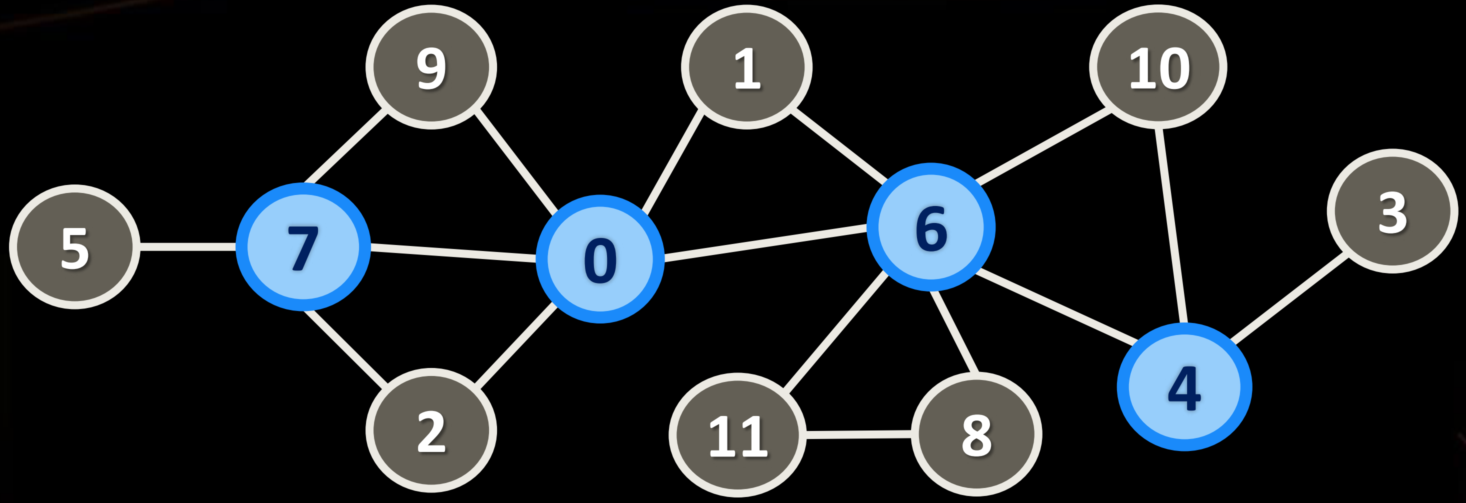
- Let **G** be a directed graph and **S** be an empty stack
- For each vertex *v* in **G**
  - Call DFS(*v*) to traverse the graph (visit each node once)
  - Each time DFS(*v*) finishes (before recursive return), push *v* onto **S**
- Build the reverse graph **G'** (reverse all edges from **G**)
- While **S** is non-empty:
  - Pop the top vertex *v* from **S**
  - if *v* is not visited, call ReverseDFS(*v*) to find the next strongly-connected component

SoftUni Foundation



Kosaraju–Sharir Algorithm

Lab

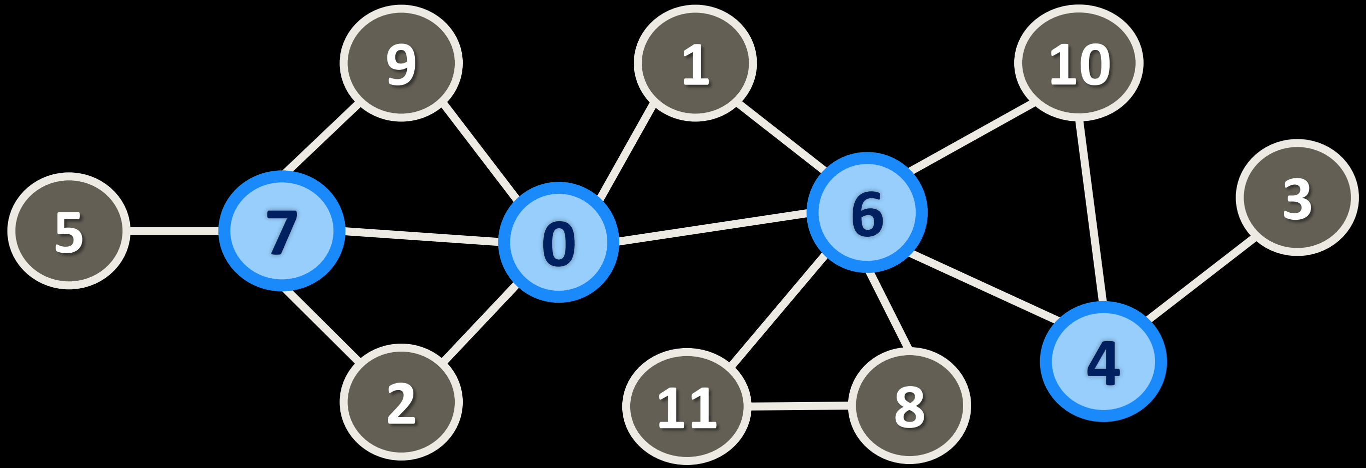


Bi-Connectivity

Finding the Articulation Points

Articulation Points

- In a connected undirected graph an articulation point is a node that when removed, splits the graph into several components
- The blue nodes below are the articulation points: 7, 0, 6, 4



- There are 6 bi-connected components:
  - {5, 7}, {0, 2, 7, 9}, {1, 0, 6}, {6, 8, 11}, {4, 6, 10}, {3, 4}

Articulation Points – The Slow Algorithm

SoftUni Foundation

- The straightforward algorithm – O(n * (m + n))
  - Remove each node and check whether the graph stays connected

```
foreach v ∈ graph nodes
  temporary remove v
  check for connectivity with DFS(u), where u ≠ v
  if the graph is not connected, print v
  restore v back in the graph
```

  - Check for graph connectivity by DFS traversal + counting the number of visited nodes

Articulation Points – The Fast Algorithm

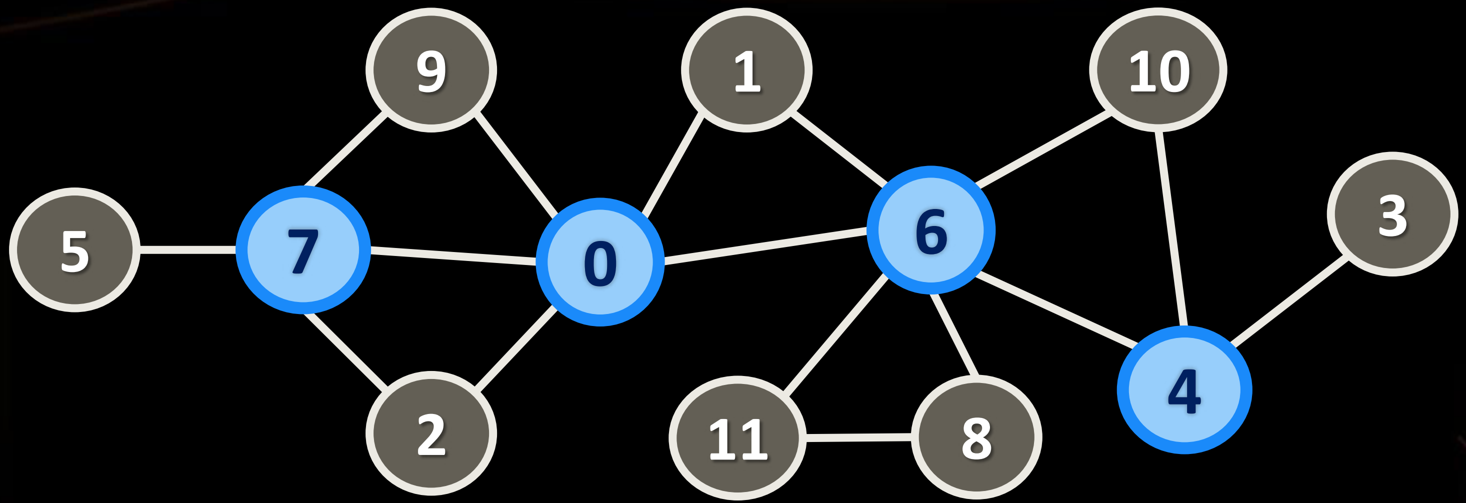
SoftUni Foundation

- The fast (linear) algorithm for finding articulation points is based on DFS traversal with some computations (Hopcroft, Tarjan, 1973)
- Run a DFS while maintaining the following information:
  - The depth(v) of each vertex in the DFS tree (once it gets visited)
  - For each vertex v, the lowest depth of neighbors of all descendants of v in the DFS tree, called the lowpoint(v)
- A non-root vertex v is an articulation point if there is a child y of v such that lowpoint(y) ≥ depth(v)
- The root vertex is an articulation point if it has at least two children

Articulation Points – Fast Algorithm

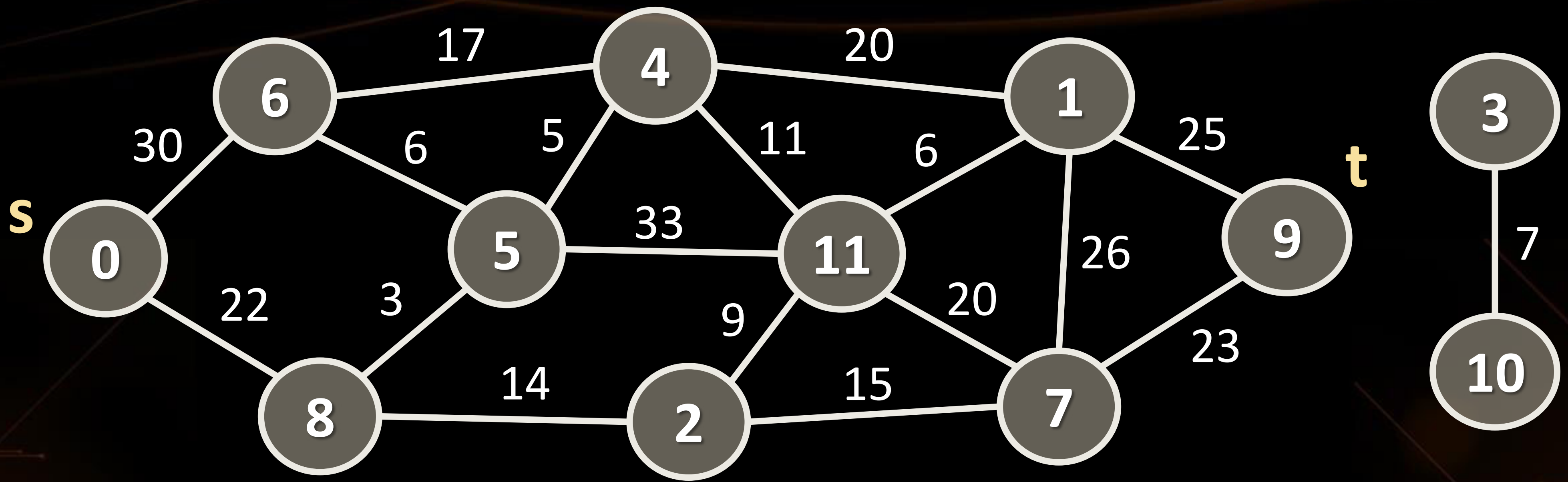
Time complexity: O(N + M)

```
FindArticulationPoints(node, d)
    visited[node] = true
    depth[node] = d
    lowpoint[node] = d
    childCount = 0
    isArticulation = false
    for each childNode in childNodes[node]
        if not visited[childNode]
            parent[childNode] = node
            FindArticulationPoints(childNode, d + 1)
            childCount = childCount + 1
            if lowpoint[childNode] >= depth[node]
                isArticulation = true
            lowpoint[node] = Min(lowpoint[node], lowpoint[childNode])
        else if childNode <> parent[node]
            lowpoint[node] = Min(lowpoint[node], depth[childNode])
    if (parent[node]<>null and isArticulation) or (parent[node]==null and childCount > 1)
          print node as articulation point
```



Finding the Articulation Points

Lab



Max Flow

Finding the Maximum Network Flow

Max Flow Problem

SoftUni Foundation

- Weighted directed / undirected graph
  - With capacities assigned to the edges c(u → v)
- Goal: compute the maximum flow from node s to node t
- Each edge {u → v} holds certain flow flow(u → v)
  - flow(u → v) ≤ c(u → v)
- For each node input flow == output flow
- Max flow == sum(flow(s → v)) == sum(flow(u → t))
  - Total output flow from s == total input from to t

Ford-Fulkerson Max-Flow Algorithm

SoftUni Foundation

- Ford-Fulkerson can be summarized in the following steps:
  1. Start from zero flow: flow(u → v) = 0 for each edge
  2. While possible:
     - Find augmenting path **p** such that
       - **p** is a valid path from **s** to **t** and for each edge {u → v} ∈ **p** → c(u → v) > 0
       - The flow(p) for the augmenting path **p** in **s → t** is the minimum capacity **c** of each edge {u → v} in the path **s → t**
       - Modify the capacities of the edges in the path **p:**
         - For each edge {u → v} ∈ **p** → c(u → v) = c(u → v) - flow(p)
         - For each edge {u → v} ∈ **p** → c(v → u) = c(v → u) + flow(p)
     - Add flow(p) to the maximum flow

Edmonds-Karp Max Flow Algorithm

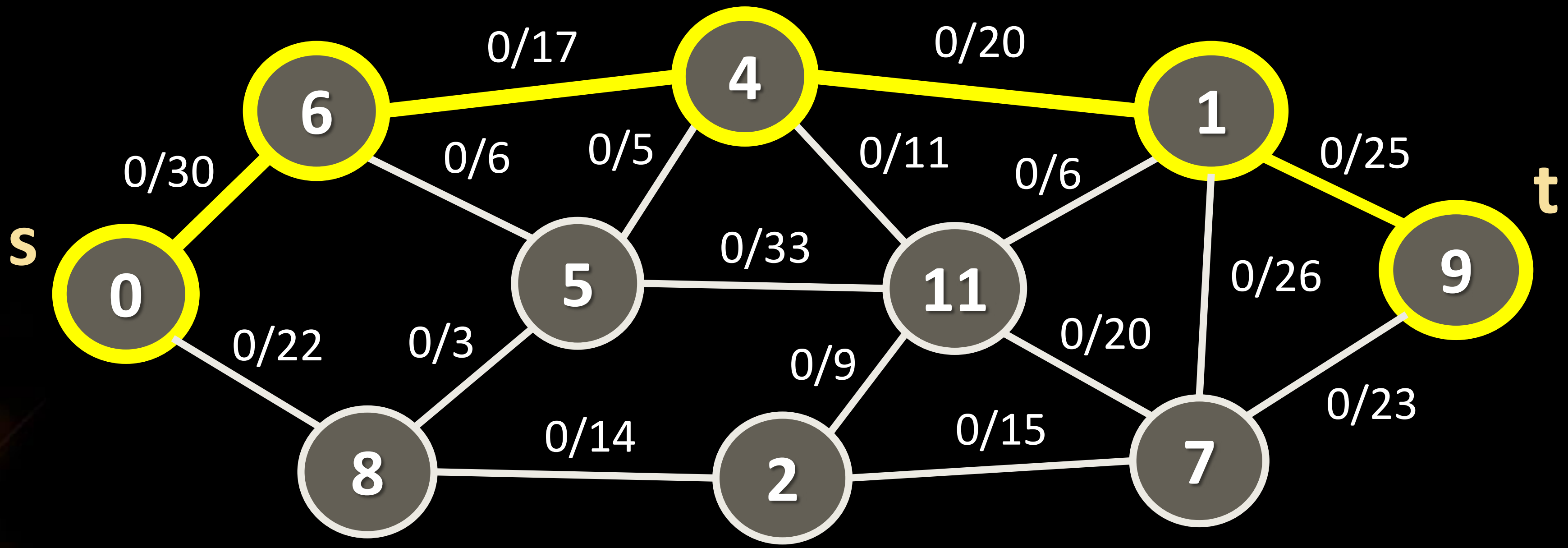
SoftUni Foundation

- If we use Breadth-first Search to Find the augmenting path we get the Edmonds-Karp algorithm
  1. Start from zero flow: `flow(u → v) = 0` for each edge
  2. While possible:
     - Find an augmenting path **p** from **s** to **t** using BFS such that:
       - For each {u → v} ∈ **p** → c(u → v) > 0
     - Keep track of the parent for each visited vertex
     - Reconstruct the path **p** using the parents
       - Set flow(**p**) as the smallest capacity **c** in the path **p**
       - Modify the capacities of the edges in the path **p** as in Ford-Fulkerson:
     - Add flow(**p**) to the maximum flow

Edmonds-Karp – Step #1

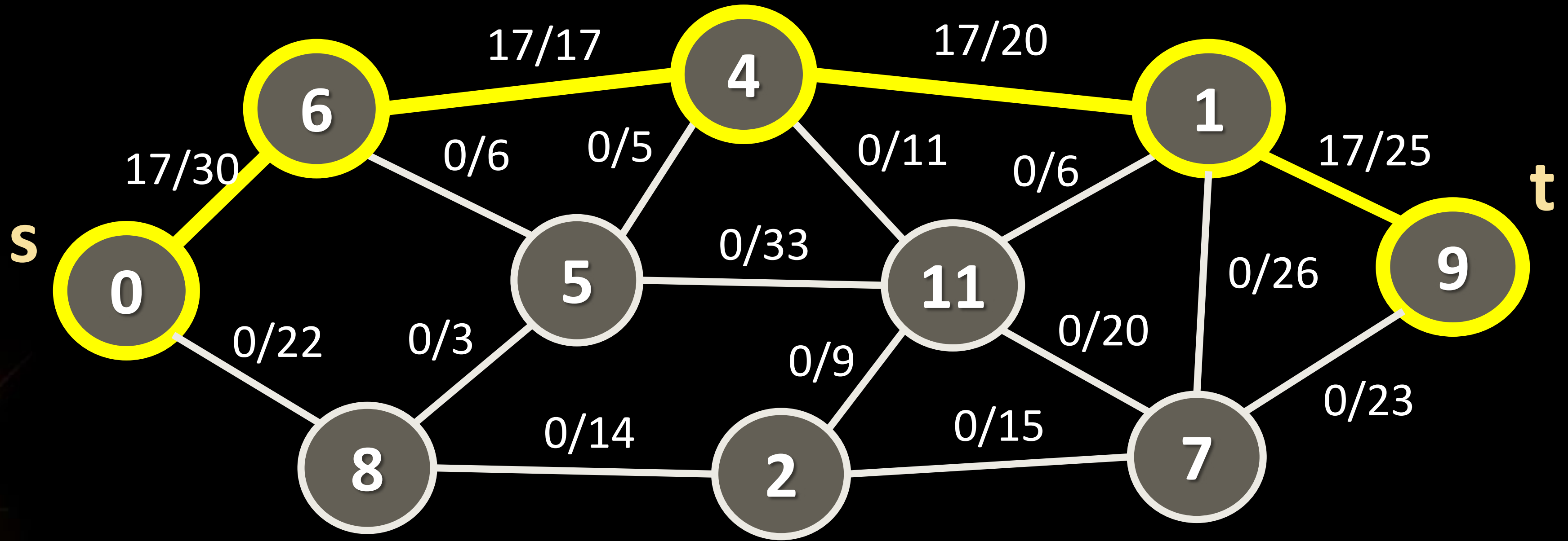
- Start from empty flows through all edges
- Find an augmenting path: 0 → 6 → 4 → 1 → 9 (increment = 17)

Time complexity: $O(VE^2)$



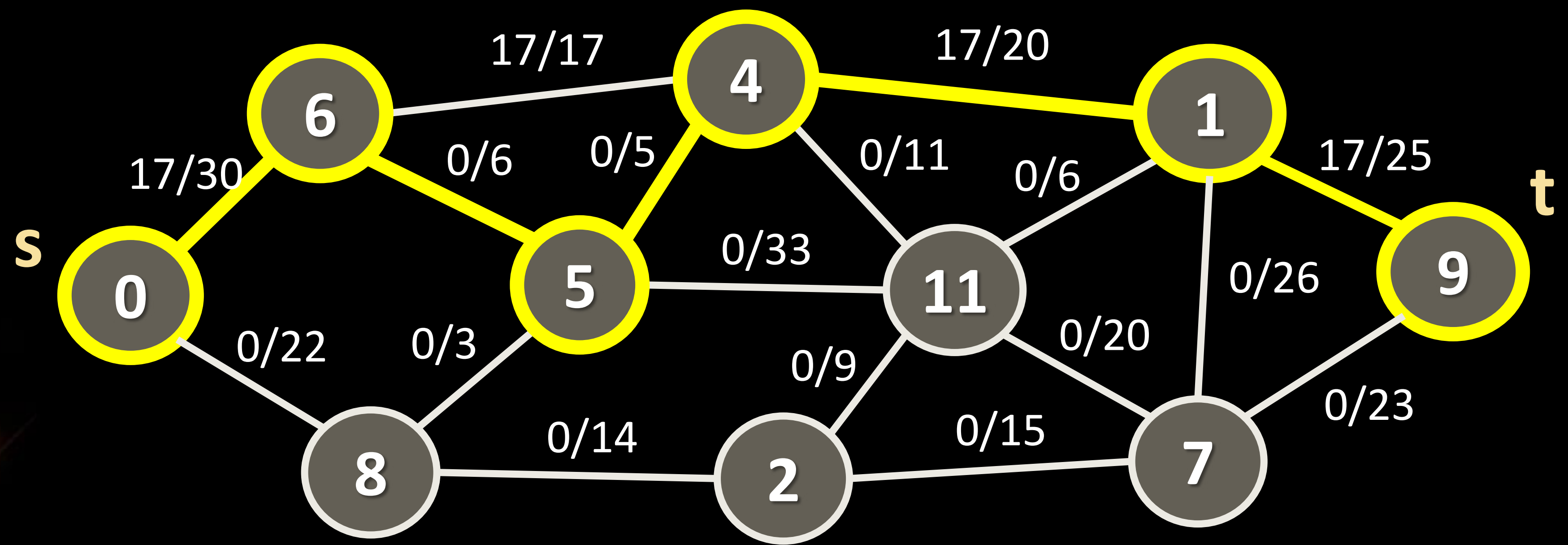
Edmonds-Karp – Step #2

- Augment the flow through the path: 0 → 6 → 4 → 1 → 9 (increment = 17)
- Current max flow = 17



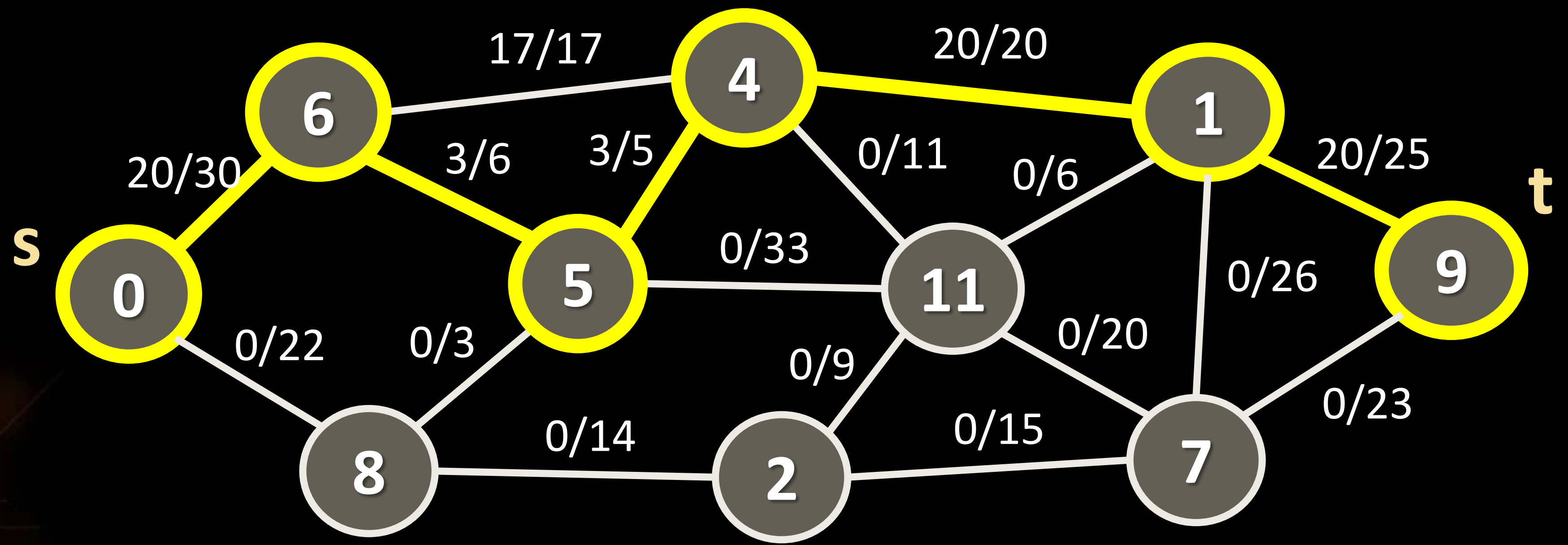
Edmonds-Karp – Step #3

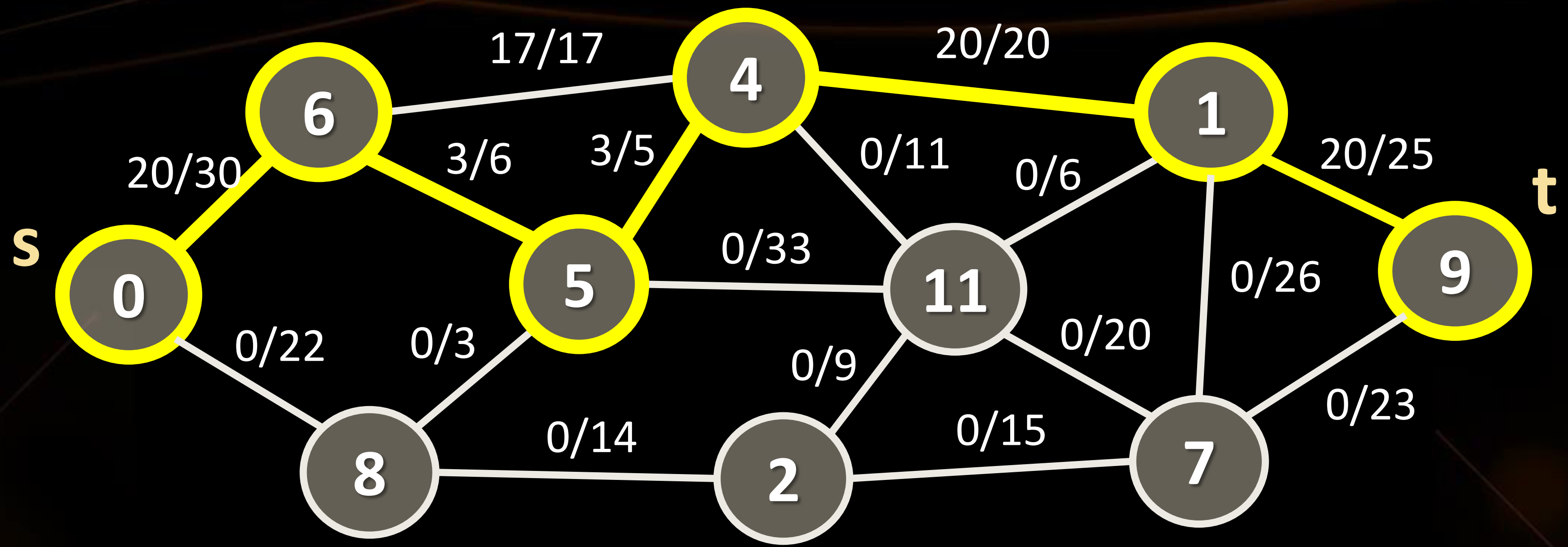
- Find an augmenting path: 0 → 6 → 5 → 4 → 1 → 9 (increment = 3)



Edmonds-Karp – Step #4

- Augment the flow through the path: 0 → 6 → 5 → 4 → 1 → 9 (increment = 3)
- Current max flow = 20





Edmonds-Karp Algorithm

Lab

Terminology

- **Level Graph** G_L :
  - A tree built by using BFS from the source **s**
  - Each node in the tree is assigned a value representing its distance from the start
  - The distance is the number of edges from the source **s**
- **Blocking flow**:
  - Combination of all augmenting paths that can be built on G_L
  - Since G_L is created using BFS, any path **p** from **s** → **t** is going to have the least amount of edges possible

Dinic/Dinitz Max-Flow Algorithm

SoftUni Foundation

- Dinitz can be summarized in the following steps:
  1. Start from zero max flow: **m** = `0`
  2. Construct the level Graph G_L using BFS
     - If **t** is **unreachable** in G_L → return the max flow **m**
  3. Find blocking flow **f** in G_L using DFS
  4. Add flow **f** to the max flow **m**

Dinitz Algorithm Pseudocode – 1/3

Time complexity: $O(V^2E)$

```
private static int[] childCounter;     // Track blocked edges
private static int[] bfsDist;          // Distances in the Level Graph
private static List<int>[] edges;      // adjacency list
private static int[][] capacities;
private static int endNode;

static int Dinic(int source, int destination)
{
    int result = 0;
    while (Bfs(source, destination))                 // While we can find a path from source to sink
    {
        for (int i = 0; i < childCounter.Length; i++)
        {
            childCounter[i] = 0;                     // Reset blocked edges on each Level Graph
        }

        int delta;
        do
        {
            delta = Dfs(source, int.MaxValue);       // Each delta is the flow from an augmenting path
            result += delta;
        }
        while (delta != 0);
    }
    return result;
}
```

Dinitz Algorithm Pseudocode – 2/3

SoftUni Foundation

```
static bool Bfs(int src, int dest)
{
    for (int i = 0; i < bfsDist.Length; i++)
    {
        bfsDist[i] = -1;         // Reset distances in Level Graph
    }
    bfsDist[src] = 0;
    Queue<int> queue = new Queue<int>();
    queue.Enqueue(src);
    while (queue.Count > 0)
    {
        int currentNode = queue.Dequeue();
        for (int i = 0; i < edges[currentNode].Count; i++)
        {
            int child = edges[currentNode][i];
            if (bfsDist[child] < 0 && capacities[currentNode][child] > 0)  // If node has not been visited
            {
                bfsDist[child] = bfsDist[currentNode] + 1;
                queue.Enqueue(child);
            }
        }
    }
    return bfsDist[dest] >= 0;   // If there is a path to the sink return true
}
```

Time complexity: $O(V^2E)$

Dinitz Algorithm Pseudocode – 3/3

Time complexity: $O(V^2E)$

```
static int Dfs(int source, int flow)
{
    if (source == endNode)    // If we reach the sink return the flow
    {
        return flow;
    }
    for (int i = childCounter[source]; i < edges[source].Count; i++, childCounter[source]++)
    {
        int child = edges[source][i];
        if (capacities[source][child] <= 0) continue;    // If the edge has no more room skip
        if (bfsDist[child] == bfsDist[source] + 1)       // Only check vertexes following the Level Graph
        {
            int augmentationPathFlow = Dfs(child, Math.Min(flow, capacities[source][child]));
            if (augmentationPathFlow > 0)
            {
                capacities[source][child] -= augmentationPathFlow;     // Fix capacities
                capacities[child][source] += augmentationPathFlow;
                return augmentationPathFlow;
            }
        }
    }
    return 0;          // If no path is found return 0 – path is blocked
}
```

Summary

1. Strongly-connected components
   - Use DFS + reverse DFS algorithm
2. Articulation points → use modified DFS
3. Maximum flow:
   - Ford-Fulkerson
   - Edmonds-Karp
   - Dinitz

Advanced Graph Algorithms II

SoftUni Foundation

Questions?

https://softuni.bg/opencourses/algorithms

License

SoftUni Foundation

- This course (slides, examples, labs, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

BY NC SA

- Attribution: this work may contain portions from
  - "Fundamentals of Computer Programming with C#" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "Data Structures and Algorithms" course by Telerik Academy under CC-BY-NC-SA license

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM