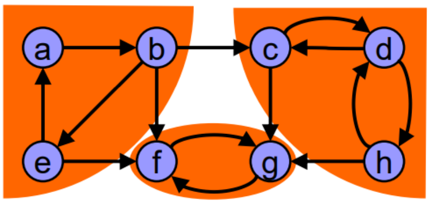
[3] Directed graphs, Strongly Connected Components, Euler trail

- **silně souvislé komponenty (strongly connected components)**
  - Orientovaný graf G=(V,E) se nazývá silně souvislý, existuje-li v každém směru mezi každými dvěma uzly v grafu cesta.
  - Silně souvislé komponenty grafu G jsou jeho maximální silně souvislé podgrafy.
  - **SCC(v)** = $\{u \in V \mid$ existuje cesta v G z u do v a cesta z v do $u\}$



Kosaraju-Sharirův algoritmus

input: graph $G = (V, E)$

output: set of strongly connected components (sets of vertices)

1. S = empty stack;
2. **while** S does not contain all vertices **do**
   Choose an arbitrary vertex v not in S;
   DFS-Walk'(v) and each time that DFS finishes expanding a vertex u, push u onto S;
3. Reverse the directions of all arcs to obtain the transpose graph;
4. **while** S is nonempty **do**
   v = pop(S);
   if v is UNVISITED **then** DFS-Walk(v);
   The set of visited vertices will give the strongly connected component containing v;

DFS-Walk

input: Graph G.

```
procedure DFS-Walk(Vertex u) {
    state[u] = OPEN; d[u] = ++time;
    for each Vertex v in succ(u)
        if (state[v] == UNVISITED) then {p[v] = u; DFS-Walk(v); }
    state[u] = CLOSED; f[u] = ++time;
}

procedure DFS-Walk'(Vertex u) {
    state[u] = OPEN; d[u] = ++time;
    for each Vertex v in succ(u)
        if (state[v] == UNVISITED) then {p[v] = u; DFS-Walk'(v); }
    state[u] = CLOSED; f[u] = ++time; push u to S;
}
```

output: array p pointing to predecessor vertex, array d with times of vertex opening and array f with time of vertex closing.

Optimalizovaný DFS-Walk

input: Graph G.

```
procedure DFS-Walk(Vertex u) {
    state[u] = OPEN;
    for each Vertex v in succ(u)
        if (state[v] == UNVISITED) then  DFS-Walk(v);
    state[u] = CLOSED;
}

procedure DFS-Walk'(Vertex u) {
    state[u] = OPEN;
    for each Vertex v in succ(u)
        if (state[v] == UNVISITED) then  DFS-Walk'(v);
    state[u] = CLOSED; push u to S;
}
```

- Kosaraju-Sharir uskuteční dva kompletní průchody grafem.
- Je-li graf reprezentován seznamem sousedů, pak má lineární složitost **$\Theta(|V|+|E|)$**.
- Je-li graf reprezentován maticí sousednosti, pak má složitost **$O(|V|^2)$**.

// Vizualizace algoritmů přikládám, ze šetrnostních důvodů, až na konec souboru.

Tarjanův algoritmus

input: graph $G = (V, E)$
output: set of strongly connected components

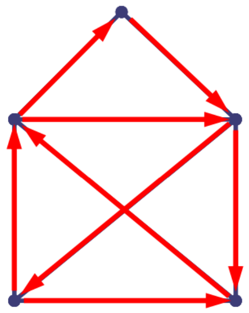
```
// every node has following fields:
// index: a unique number to ID node
// lowlink: ties node to others in SCC
// pred: pointer to stack predecessor
// instack: true if node is in stack

procedure push( v )
// stack may be null
  v.pred    = S;
  v.instack = true;
  S         = v;
end push;


function pop( v )
// val param v is stack copy
  S         = v.pred;
  v.pred    = null;
  v.instack = false;
  return v;
end pop;
```

```
procedure find_scc( v )
  v.index = v.lowlink = ++index;
  push( v );
  foreach node w in succ( v ) do
    if w.index = 0 then // not yet visited
      find_scc( w );
      v.lowlink = min( v.lowlink, w.lowlink );
    elsif w.instack then
      v.lowlink = min( v.lowlink, w.index );
    end if
  end foreach

  if v.lowlink = v.index then // v: head of SCC
    SCC++ // track how many SCCs found
    repeat
      x = pop( S );
      add x to current strongly connected component;
    until x = v;
    output the current strongly connected component;
  end if
end find_scc;

index = 0;    // unique node number > 0
S = null;     // pointer to node stack
SCC = 0;      // number of SCCs in G
foreach node v in V do
  if v.index = 0 then // yet unvisited
    find_scc( v );
  end if
end foreach;
```

- Tarjan uskuteční jediný kompletní průchod grafem.
- seznam sousedů → lineární složitost **Θ(|V|+|E|)**
- matice sousednosti → složitost **O(|V|²)**
- Tarjan poběží rychleji než Kosaraju-Sharir.

Eulerovský tah (Euler's trail)

- Existuje v (orientovaném či neorientovaném) grafu G tah, který navštíví každý uzel právě jednou?
- **tah** je podobný cestě, ale hrany se nemohou opakovat (uzly ano)



- [**Theorem**] Graf G obsahuje eulerovský tah, právě tehdy když je souvislý a má 0 či 2 uzly lichého stupně.

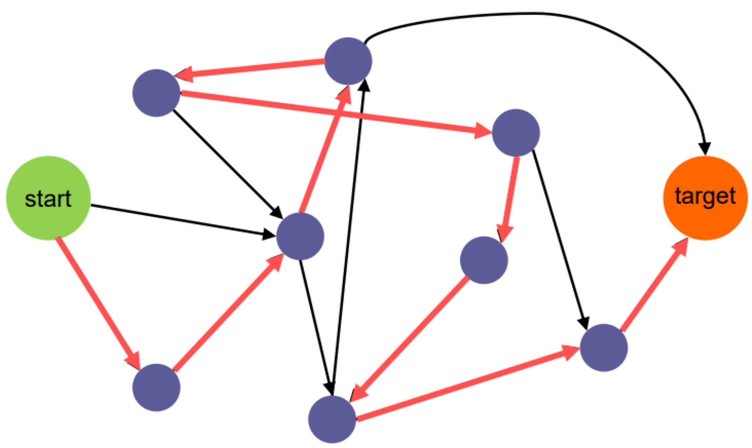
- lze rozlišit dva případy:
  - Eulerův tah začíná a **končí ve stejném uzlu** (Eulerian tour)
    - každý uzel musí mít sudý stupeň
  - začíná a končí **v různých uzlech**
    - oba uzly musí mít lichý stupeň a ostatní uzly musí mít sudý stupeň

- Euler uskuteční jen jeden kompletní průchod grafem.
- seznam sousedů → lineární složitost **Θ(|V|+|E|)**
- matice sousednosti → složitost **O(|V|²)**

```
input: graph G = (V, E)
output: trail (as a stack with edges)

procedure euler-trail(vertex v);
{
 foreach vertex u in succ(v) do {
   remove edge(v,u) from graph;
   euler-trail(u);
   push(edge(v,u));
  }
}
```

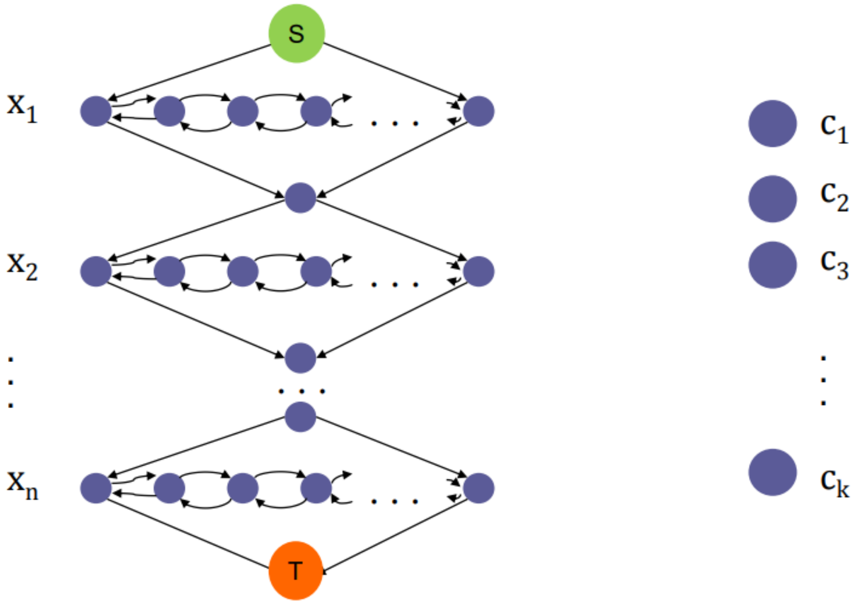
Hamiltonovská cesta (Hamiltonian Path)

- Existuje v (orientovaném či neorientovaném) grafu G cesta, která navštíví každý uzel právě jednou?

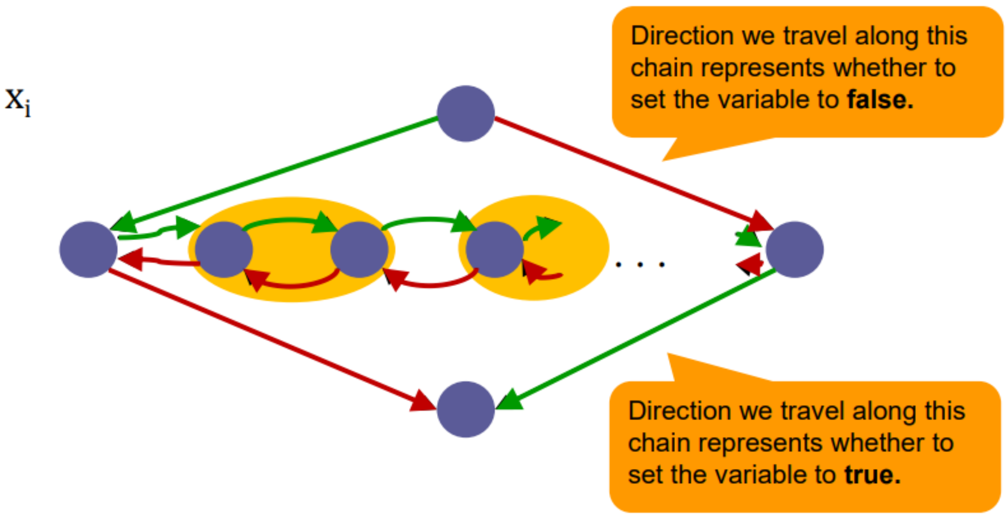


- velice složitý problém (**NP-úplný**)
- nápad na redukci:
  - Předpokládejme, že máme black box na řešení Hamiltonovské cesty.
  - Známe již NP-úplný problém splnitelnosti výrokových formulí (SAT – satisfiability)
  - Ideou je v polynomiálním čase převést vstup v podobě SAT instancí na instance vhodné pro náš black box, a to tak, že řešení z black boxu bude přímo reprezentovat SAT řešení daného vstupu.
  - Vyřešíme-li tedy black box v polynomiálním čase, tak můžeme v polynomiálním čase vyřešit i SAT.

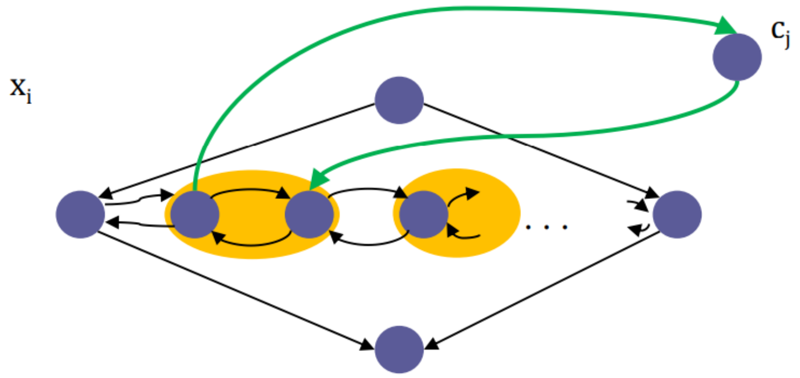
- **high-level struktura**:



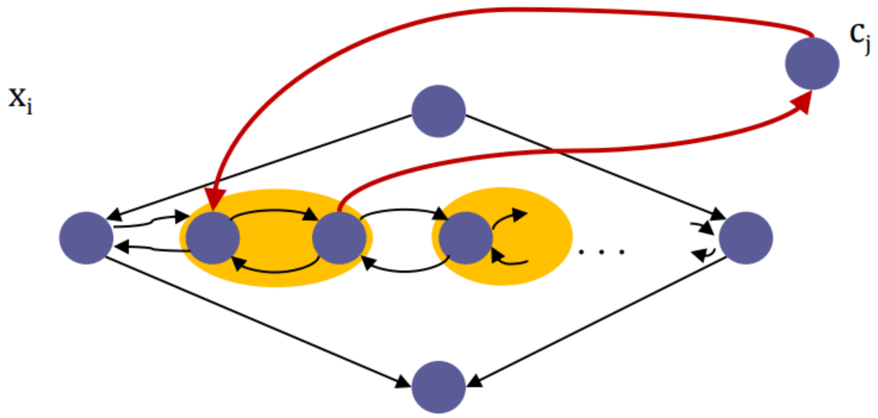
- **interní struktura proměnné x_i:**
  - počet výskytů proměnné x_i v celém SAT přesně koresponduje s počtem dvojic ve žlutých oválech



- pokud c_j obsahuje **pozitivní** literál: x_i



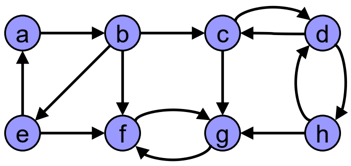
- pokud c_j obsahuje **negativní** literál: $\neg x_i$



Zdroj: https://cw.fel.cvut.cz/old/_media/courses/a4m33pal/2012pal03.pdf

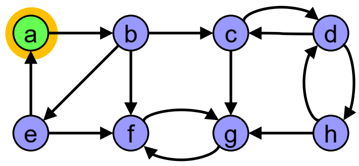
// Zde dokument nekončí, následují vizualizace algoritmů.

Kosaraju-Sharir Algorithm



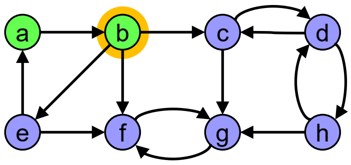
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



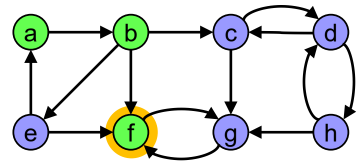
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



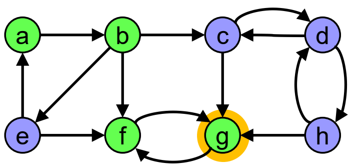
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



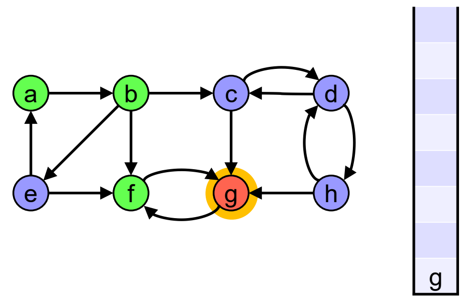
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



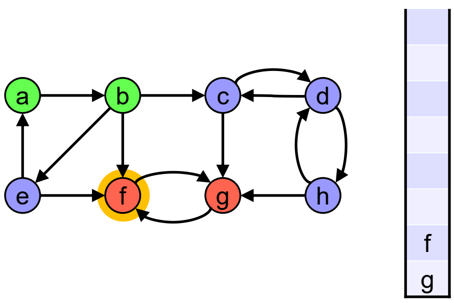
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



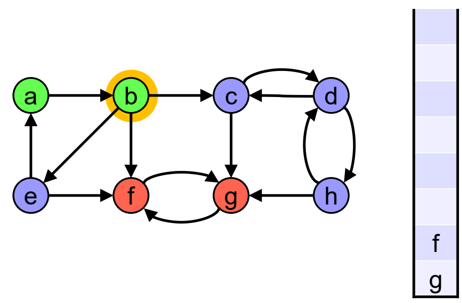
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



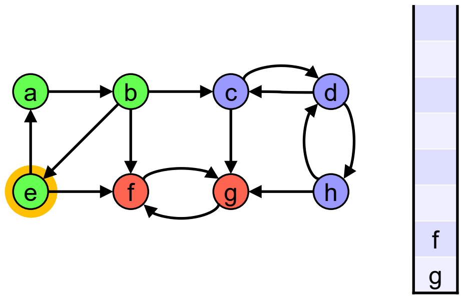
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



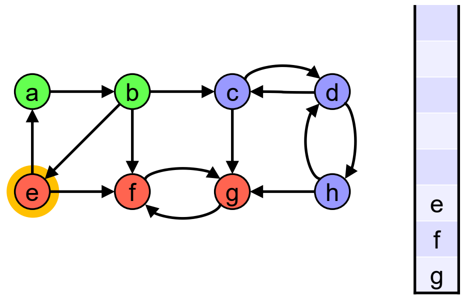
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



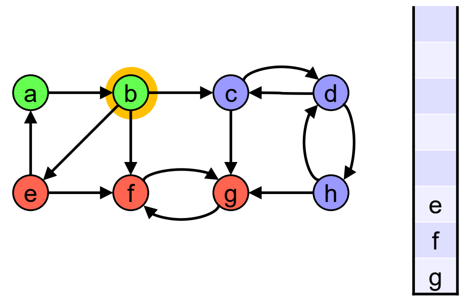
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



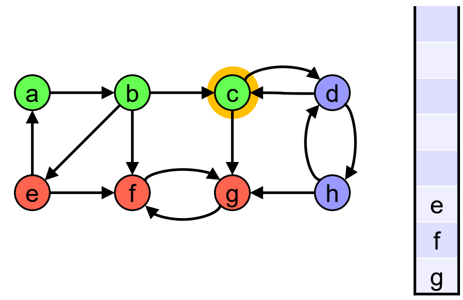
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



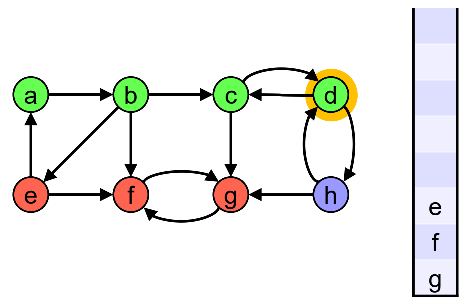
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



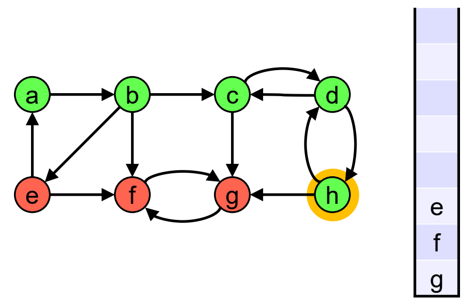
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



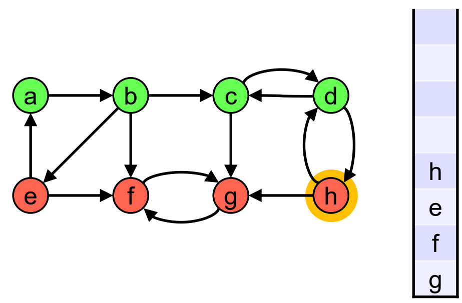
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



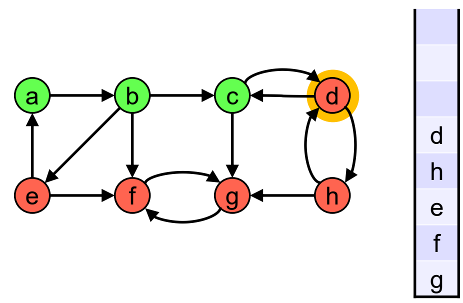
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



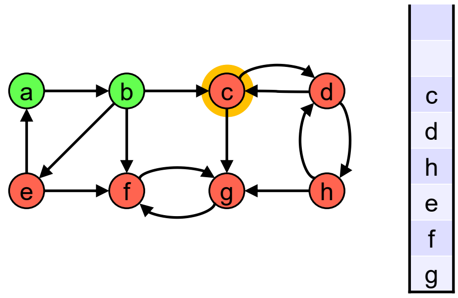
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



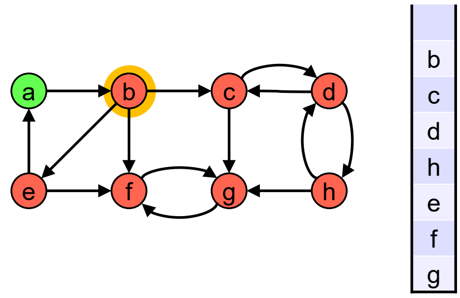
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



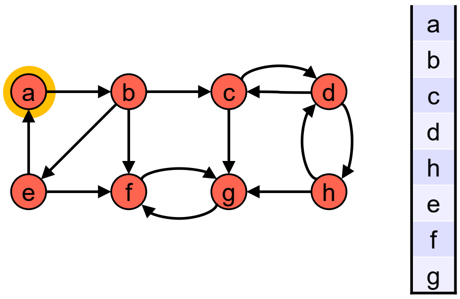
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



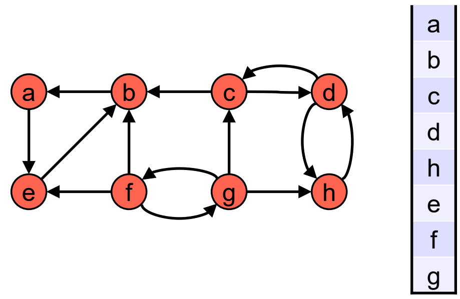
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



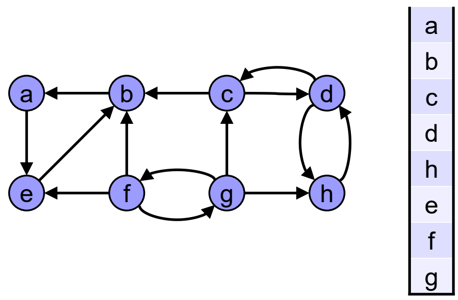
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



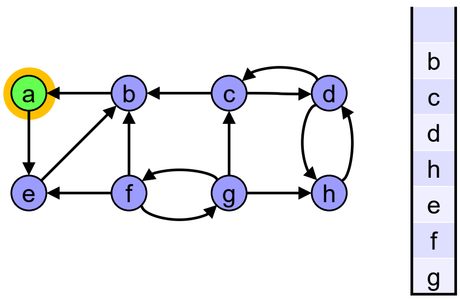
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



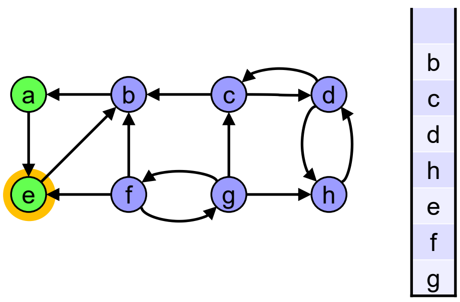
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



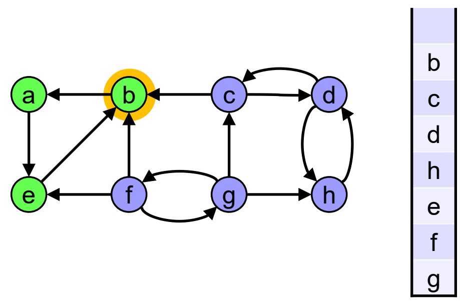
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



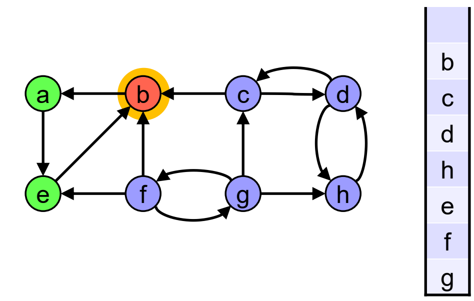
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



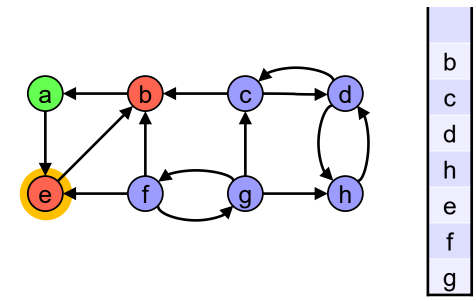
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



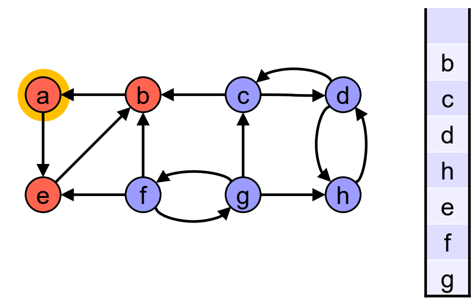
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



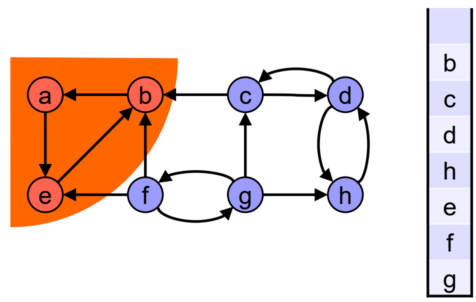
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



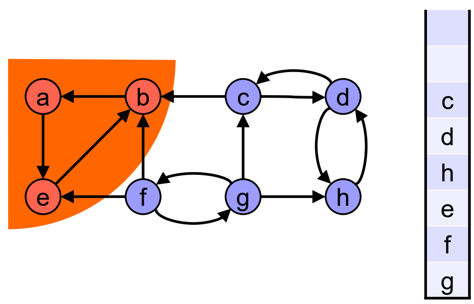
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



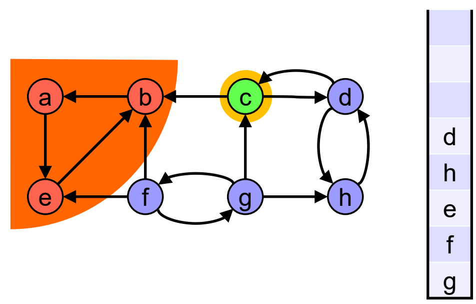
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



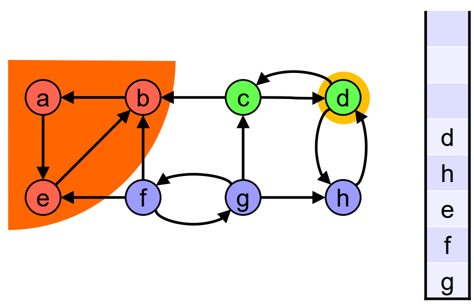
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



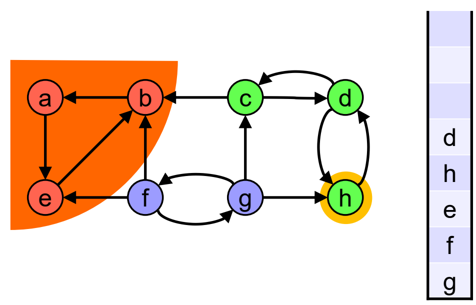
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm



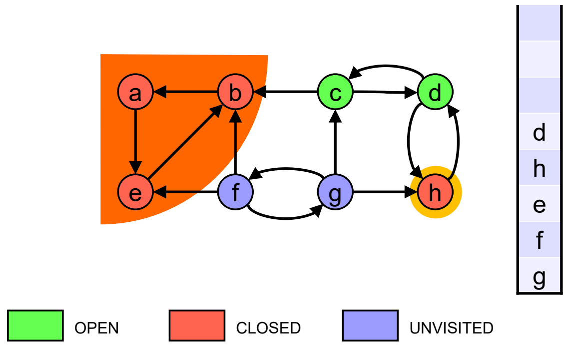
OPEN CLOSED UNVISITED

Kosaraju-Sharir Algorithm

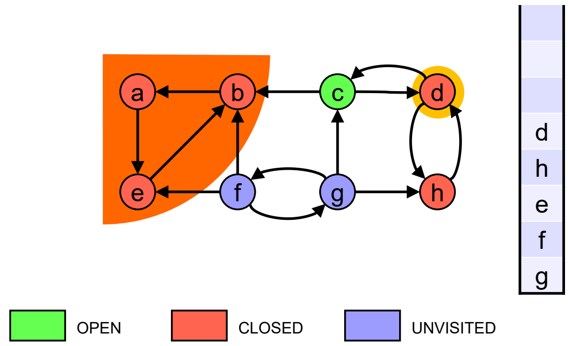


OPEN CLOSED UNVISITED

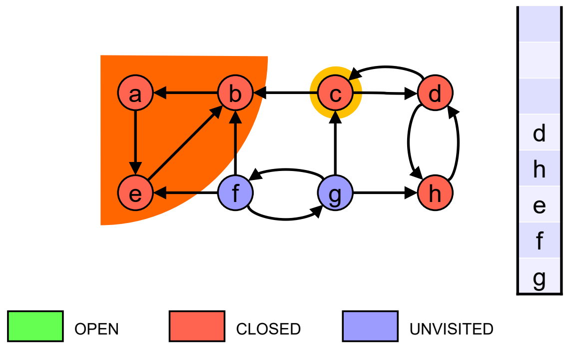
Kosaraju-Sharir Algorithm



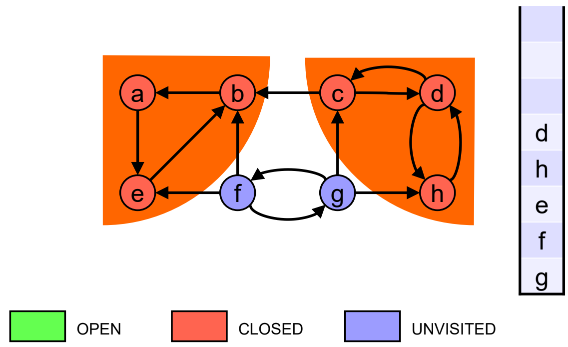
Kosaraju-Sharir Algorithm



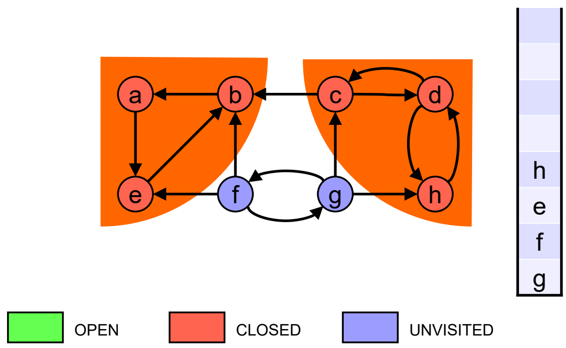
Kosaraju-Sharir Algorithm



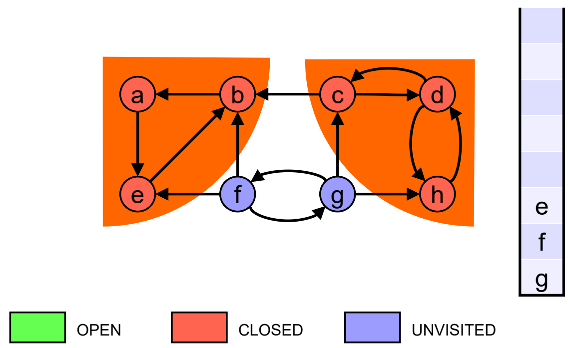
Kosaraju-Sharir Algorithm



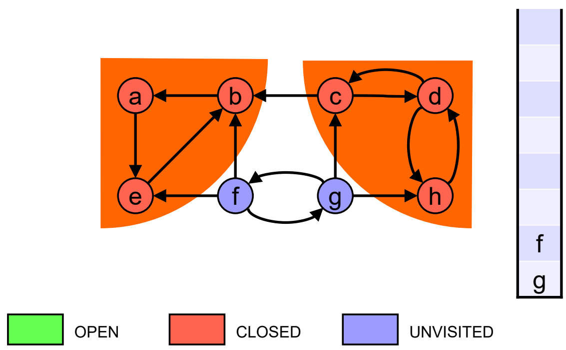
Kosaraju-Sharir Algorithm



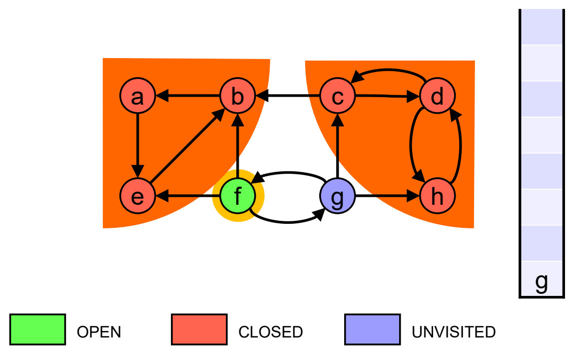
Kosaraju-Sharir Algorithm

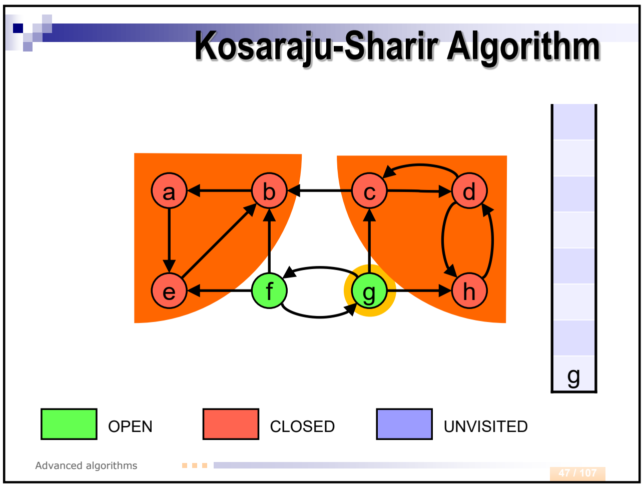


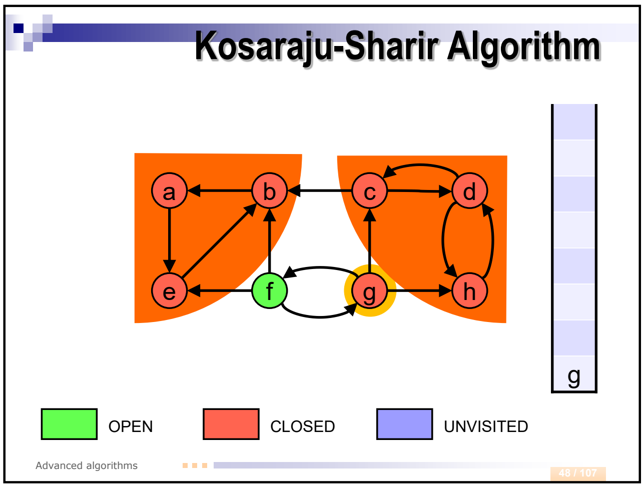
Kosaraju-Sharir Algorithm

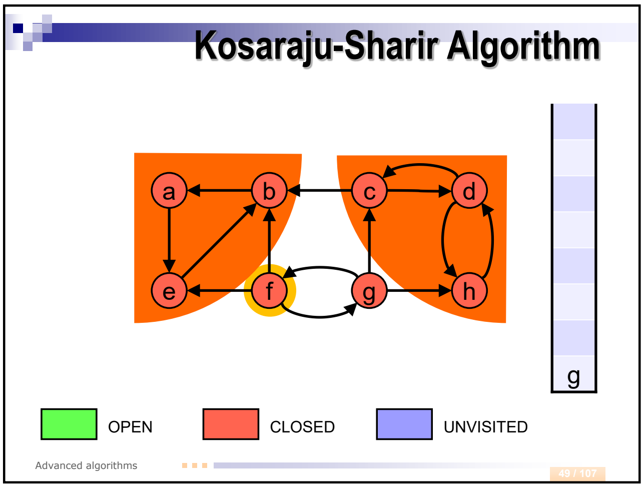


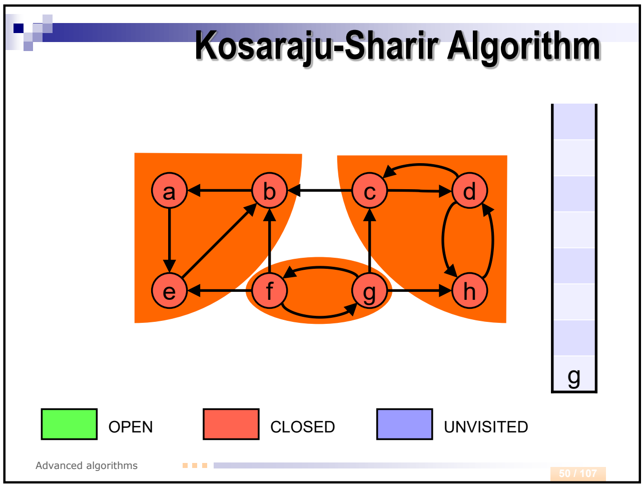
Kosaraju-Sharir Algorithm

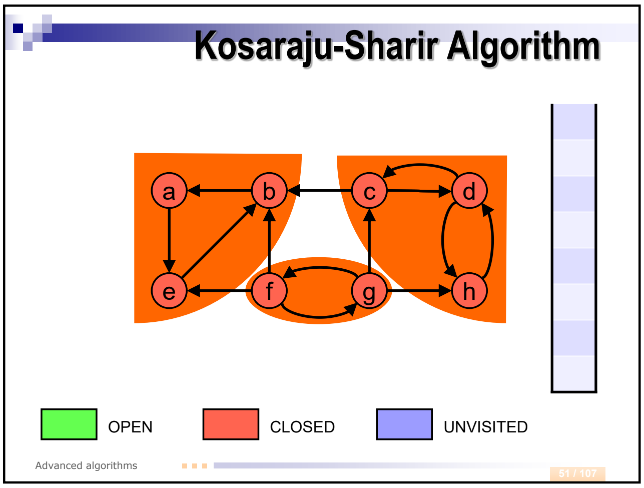


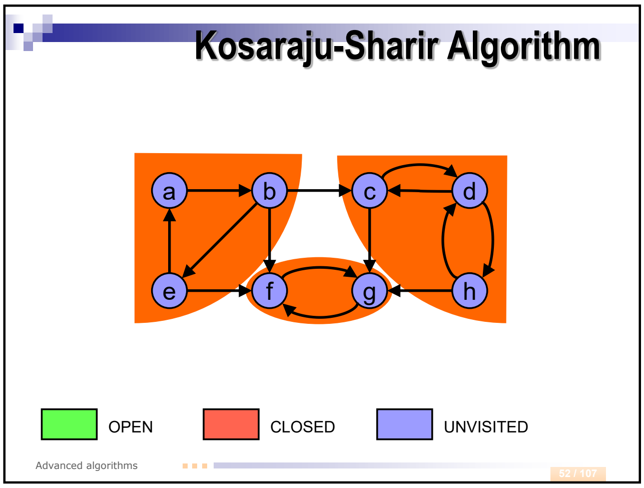

Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
g
OPEN
CLOSED
UNVISITED
Advanced algorithms
47 / 107


Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
g
OPEN
CLOSED
UNVISITED
Advanced algorithms
48 / 107

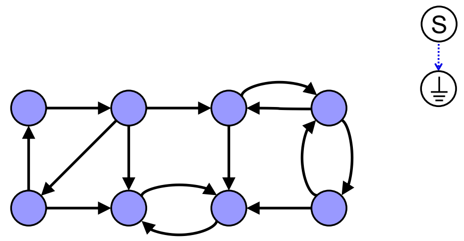

Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
g
OPEN
CLOSED
UNVISITED
Advanced algorithms
49 / 107


Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
g
OPEN
CLOSED
UNVISITED
Advanced algorithms
50 / 107


Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
OPEN
CLOSED
UNVISITED
Advanced algorithms
51 / 107

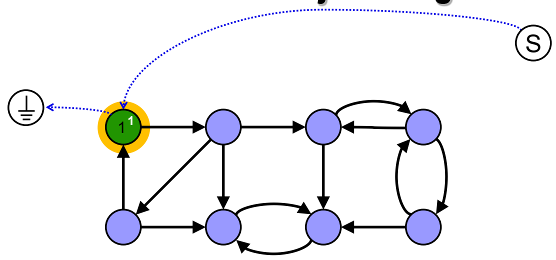

Kosaraju-Sharir Algorithm
a
b
c
d
e
f
g
h
OPEN
CLOSED
UNVISITED
Advanced algorithms
52 / 107

Tarjan's Algorithm



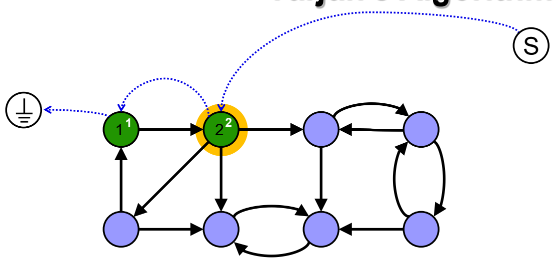
pred | instack = true | instack = false

Tarjan's Algorithm



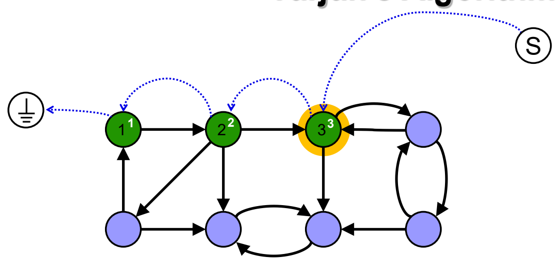
pred | instack = true | instack = false

Tarjan's Algorithm



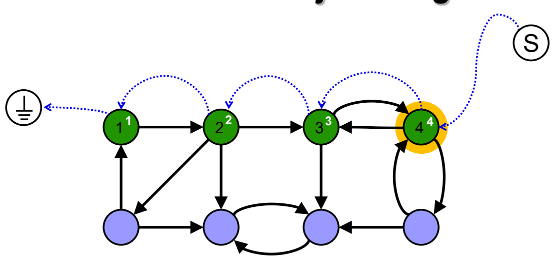
pred | instack = true | instack = false

Tarjan's Algorithm



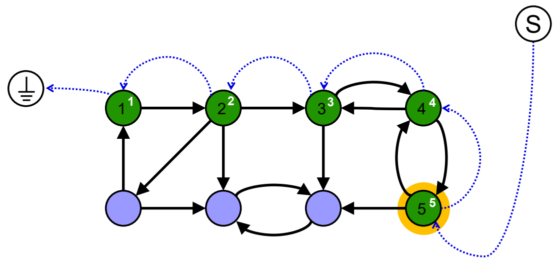
pred | instack = true | instack = false

Tarjan's Algorithm



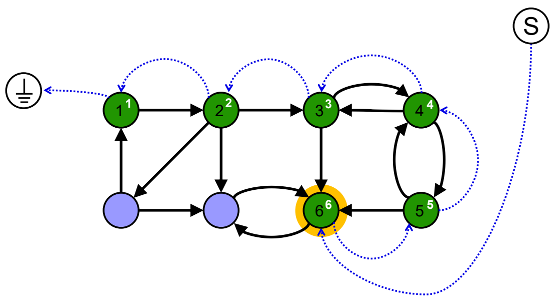
pred | instack = true | instack = false

Tarjan's Algorithm



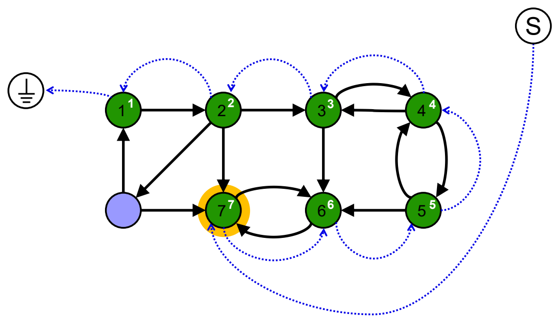
pred | instack = true | instack = false

Tarjan's Algorithm



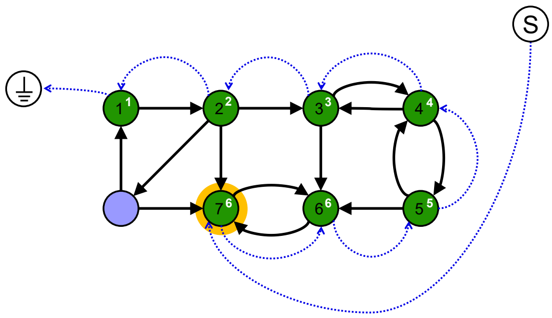
pred | instack = true | instack = false

Tarjan's Algorithm



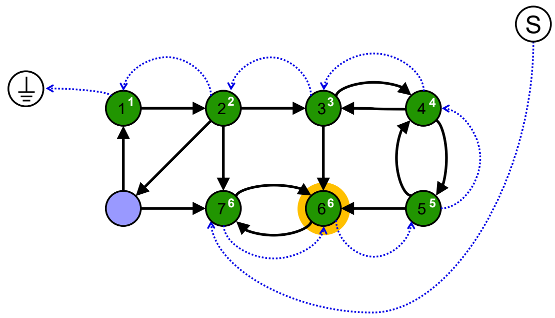
pred | instack = true | instack = false

Tarjan's Algorithm



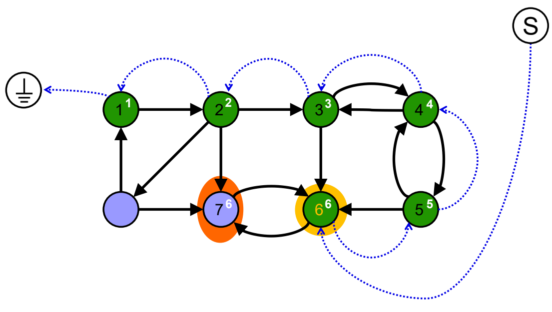
pred | instack = true | instack = false

Tarjan's Algorithm



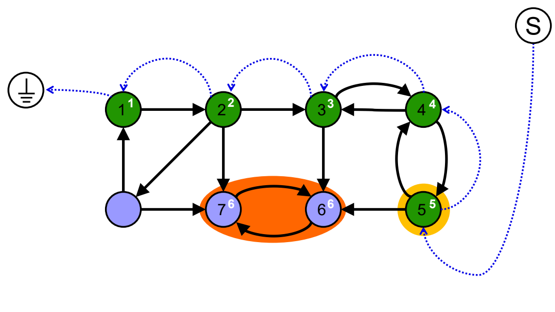
pred | instack = true | instack = false

Tarjan's Algorithm



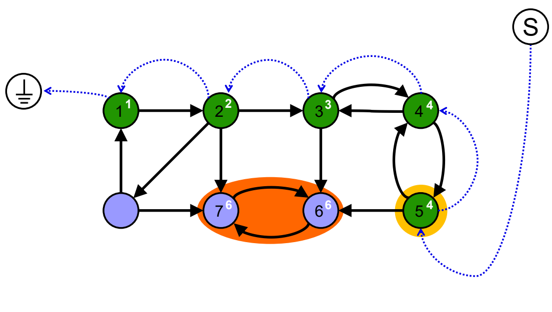
pred | instack = true | instack = false

Tarjan's Algorithm



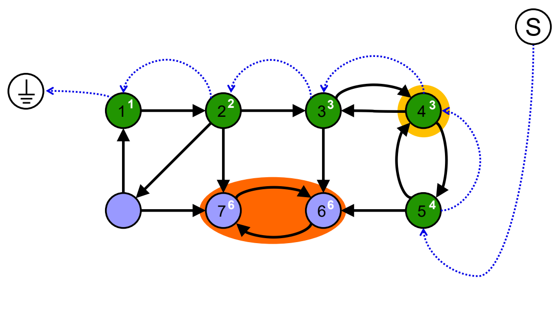
pred | instack = true | instack = false

Tarjan's Algorithm



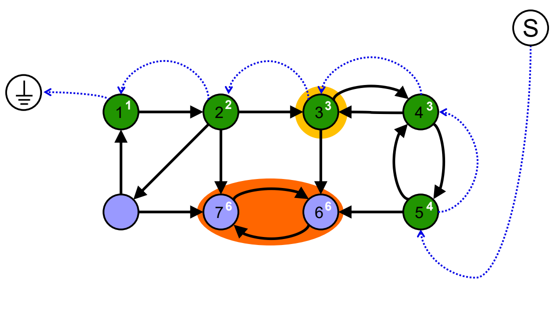
pred | instack = true | instack = false

Tarjan's Algorithm



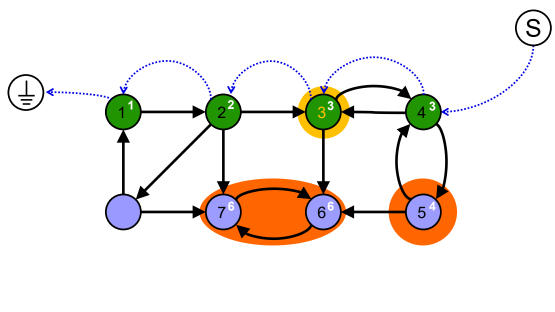
pred | instack = true | instack = false

Tarjan's Algorithm

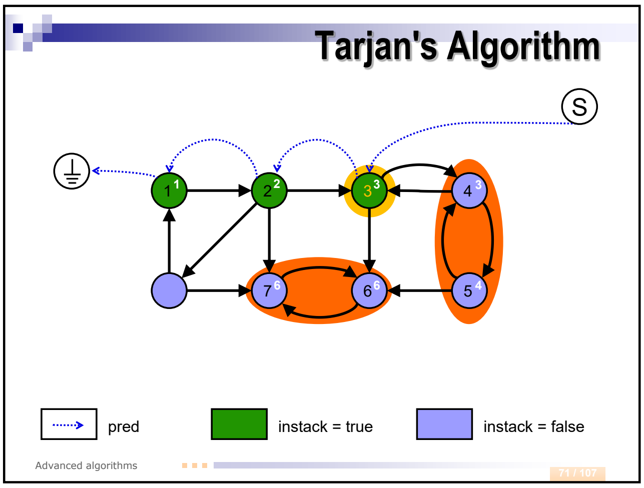


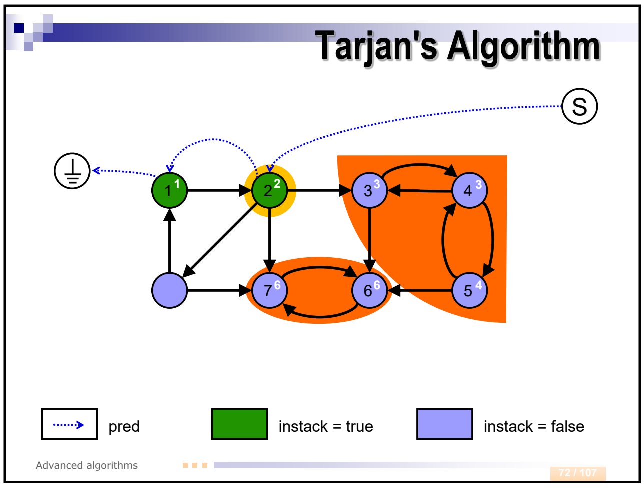
pred | instack = true | instack = false

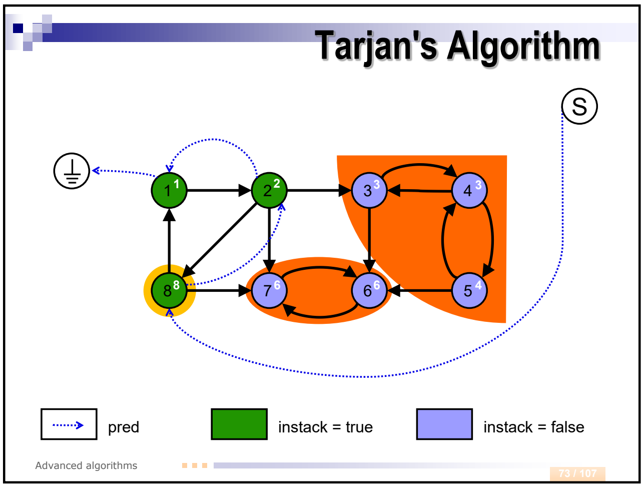
Tarjan's Algorithm

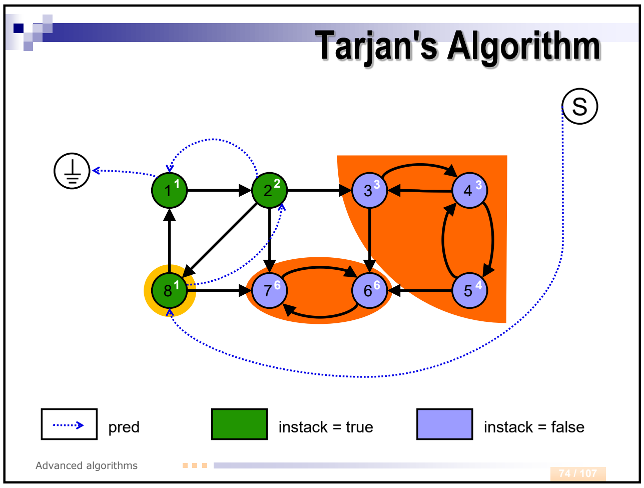


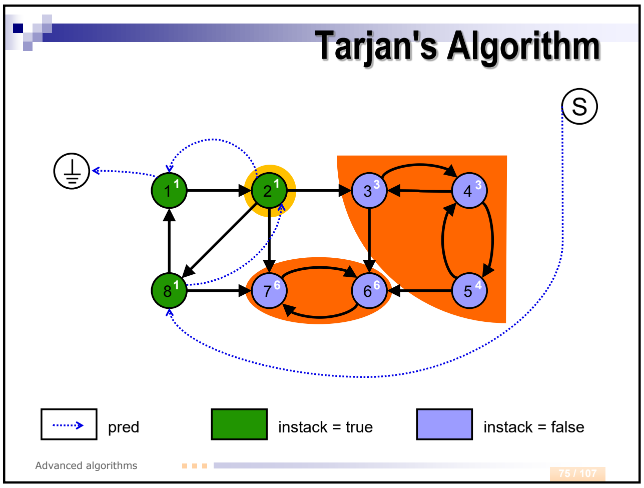
pred | instack = true | instack = false

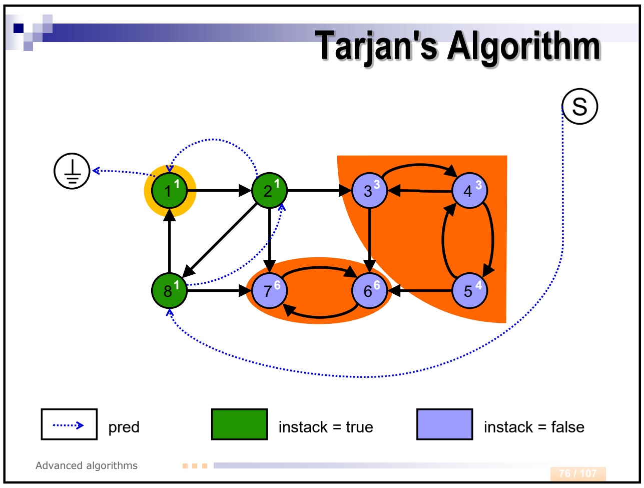

Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
71 / 107

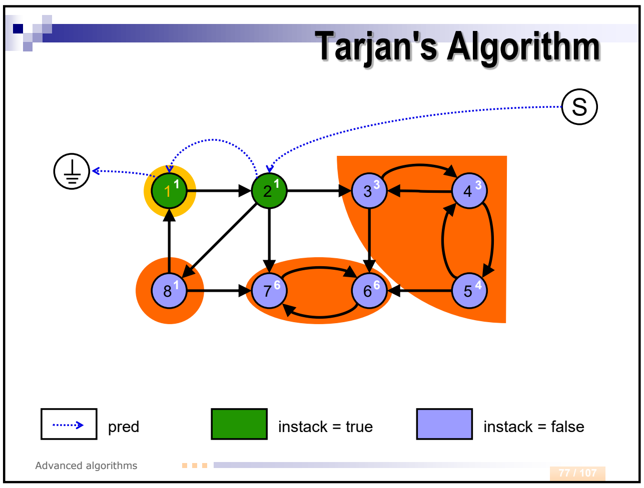

Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
72 / 107

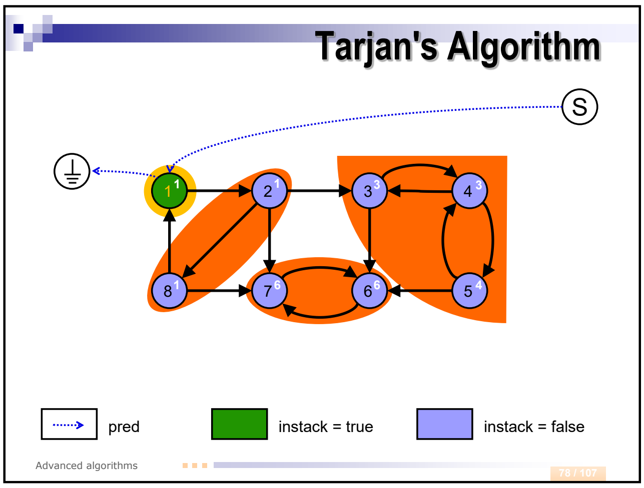

Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
73 / 107


Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
74 / 107

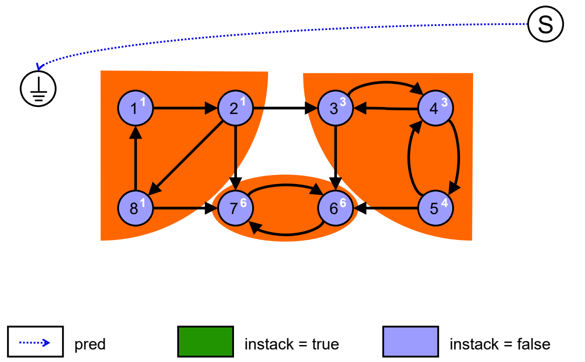

Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
75 / 107


Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
76 / 107

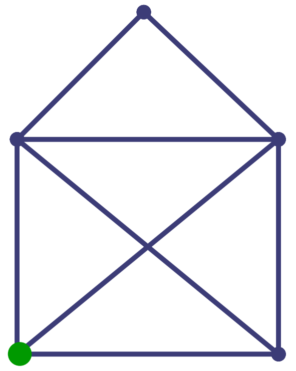

Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
77 / 107


Tarjan's Algorithm
S
pred
instack = true
instack = false
Advanced algorithms
78 / 107

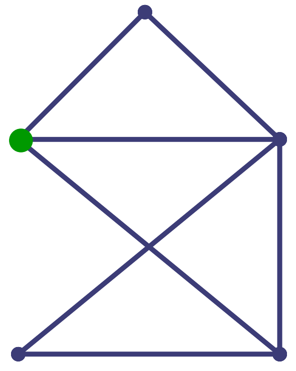
Tarjan's Algorithm



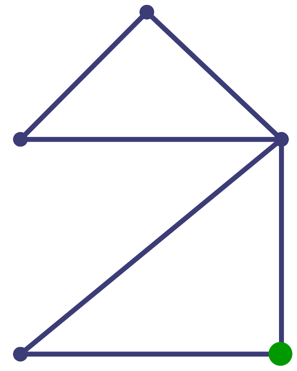
Euler Trail



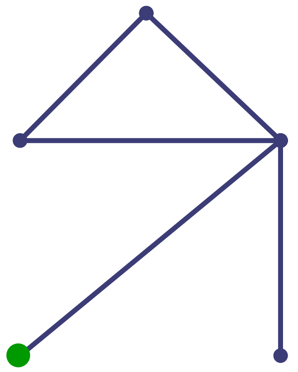
Euler Trail



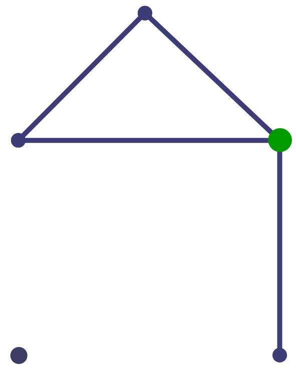
Euler Trail



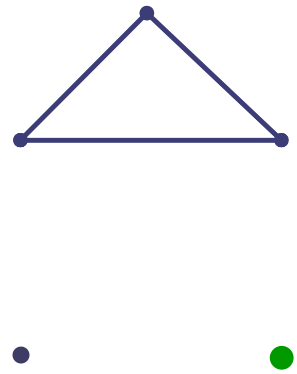
Euler Trail



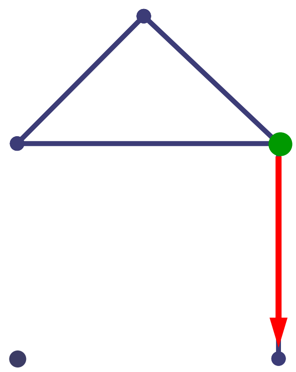
Euler Trail



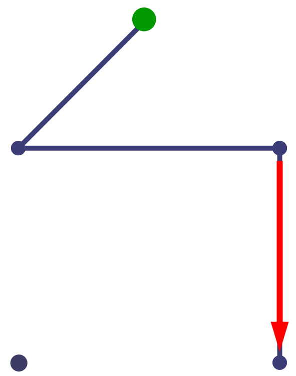
Euler Trail



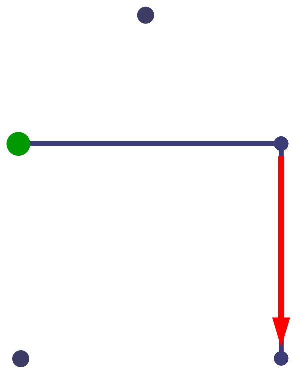
Euler Trail



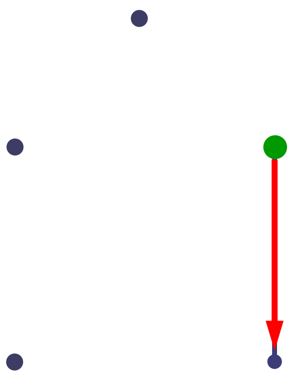
Euler Trail



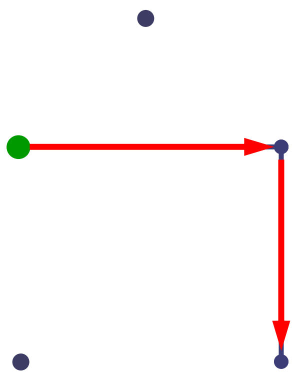
Euler Trail



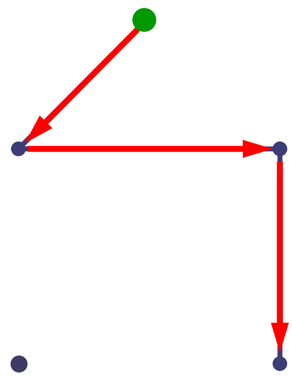
Euler Trail



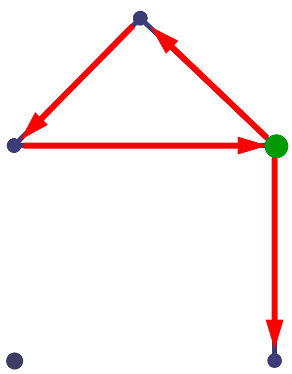
Euler Trail



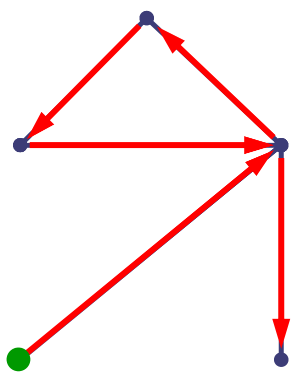
Euler Trail



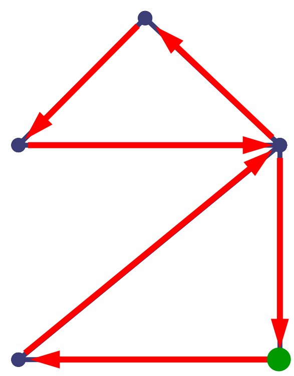
Euler Trail



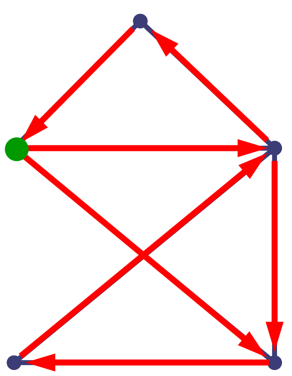
Euler Trail



Euler Trail



Euler Trail



Euler Trail