INF2220 - Algoritmer og datastrukturer

HØSTEN 2016

Institutt for informatikk, Universitetet i Oslo

Forelesning 7:

Grafer III

Dagens plan:

Bevis for Prim

Dybde-først søk

- Biconnectivity
- Strongly connected components

Fra siste forelesning:

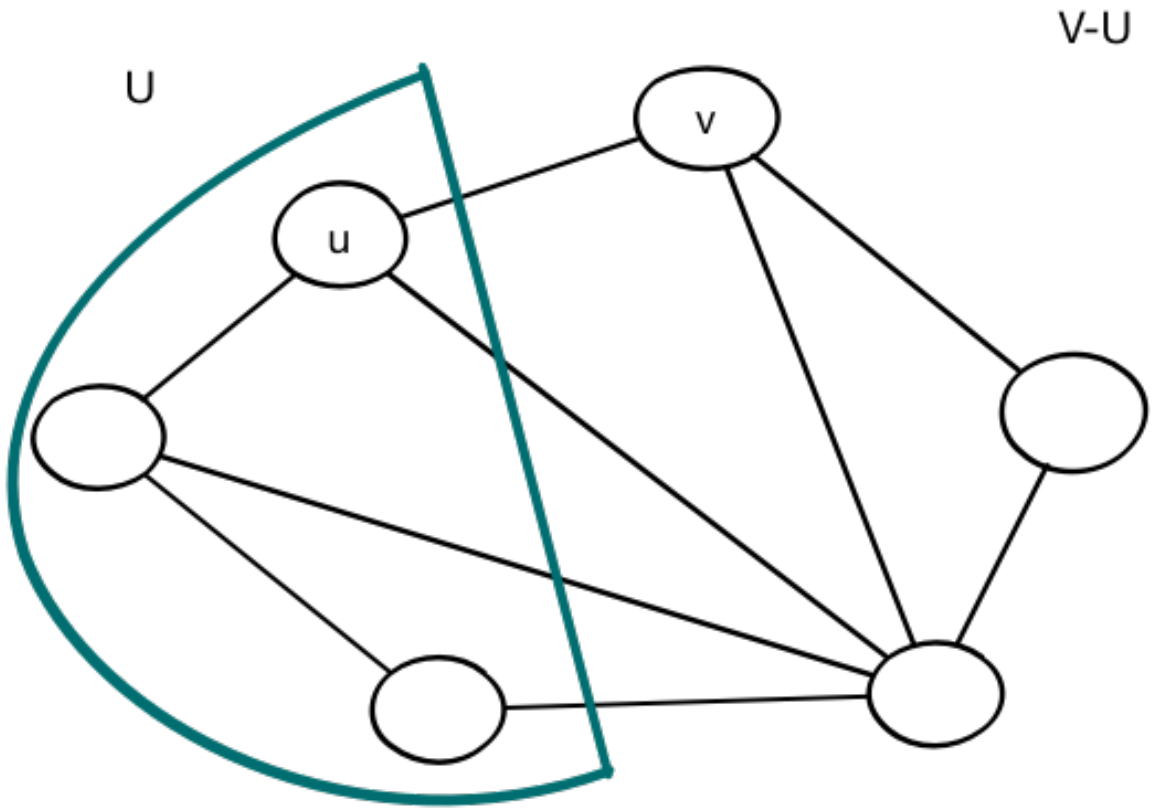
Lemma (Løkke-lemma for spenntrær)

Anta at **U** *er et spenntre for en graf, og at kanten* **e** *ikke er med i treet* **U**
Hvis vi legger kanten **e** *til treet* **U**, *dannes en entydig bestemt enkel løkke.*
Hvis vi fjerner en vilkårlig kant i denne løkken, vil vi igjen ha et spenntre for grafen

Invariant

Det treet **T** som dannes av de kantene (og deres endenoder) vi til nå har plukket ut, er slik at det finnes et minimalt spenntre **U** for grafen som inneholder (alle kantene i) **T**

Bevis med motsigelse

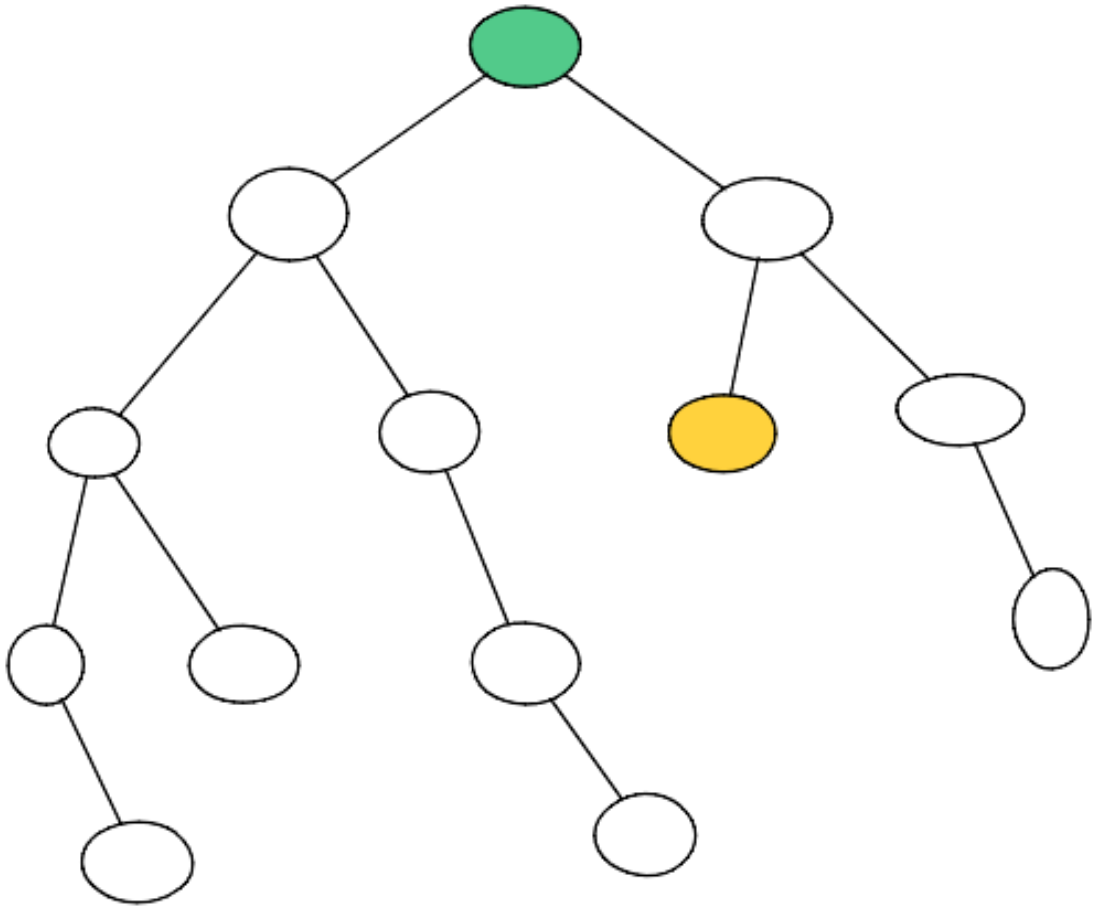


La (u, v) være den kanten med lavest vekt fra U til V-U.
Påstå: (u, v) er en del av MST for G

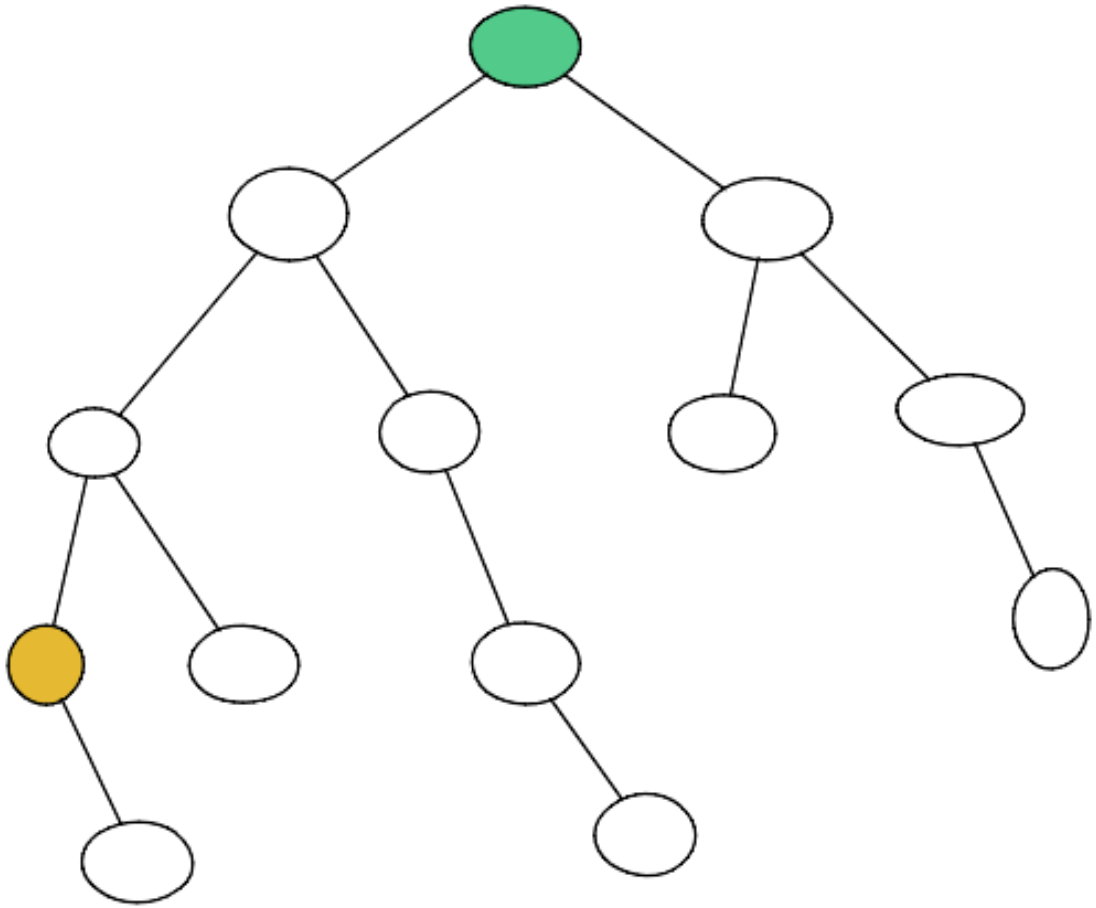
Dybde-først søk

```
void dybdeFørstSøk(Node v) {
    v.merke = true;
    for < hver nabo w til v > {
        if (!w.merke) {
            dybdeFørstSøk(w);
        }
    }
}
```

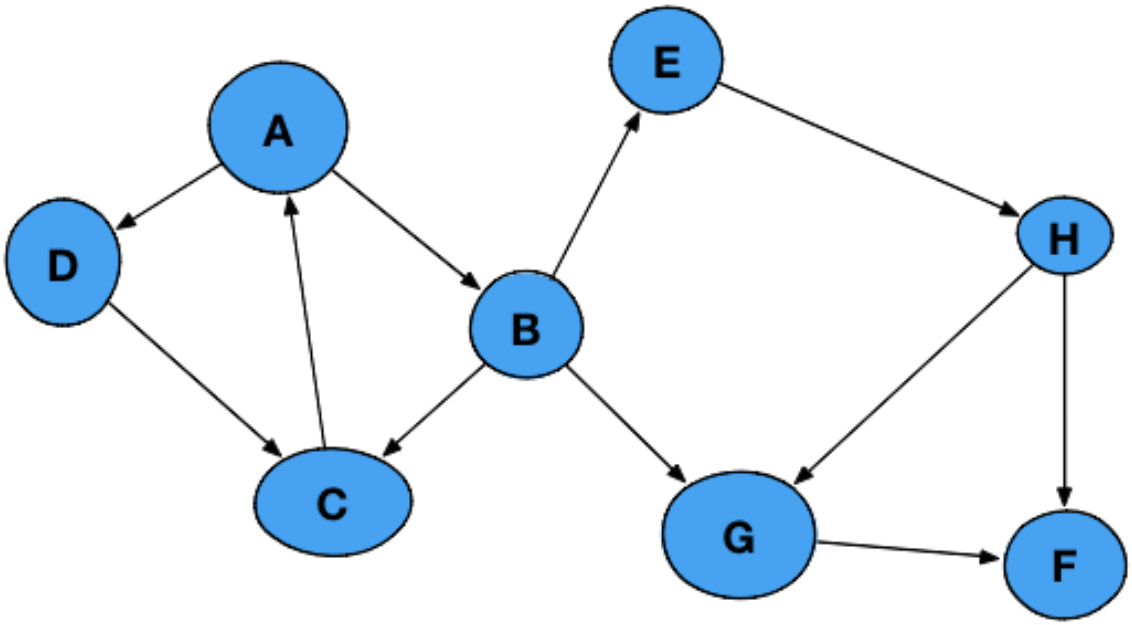
BFS vs DFS



BFS vs DFS



BFS vs DFS



Dybde-først-spenntre

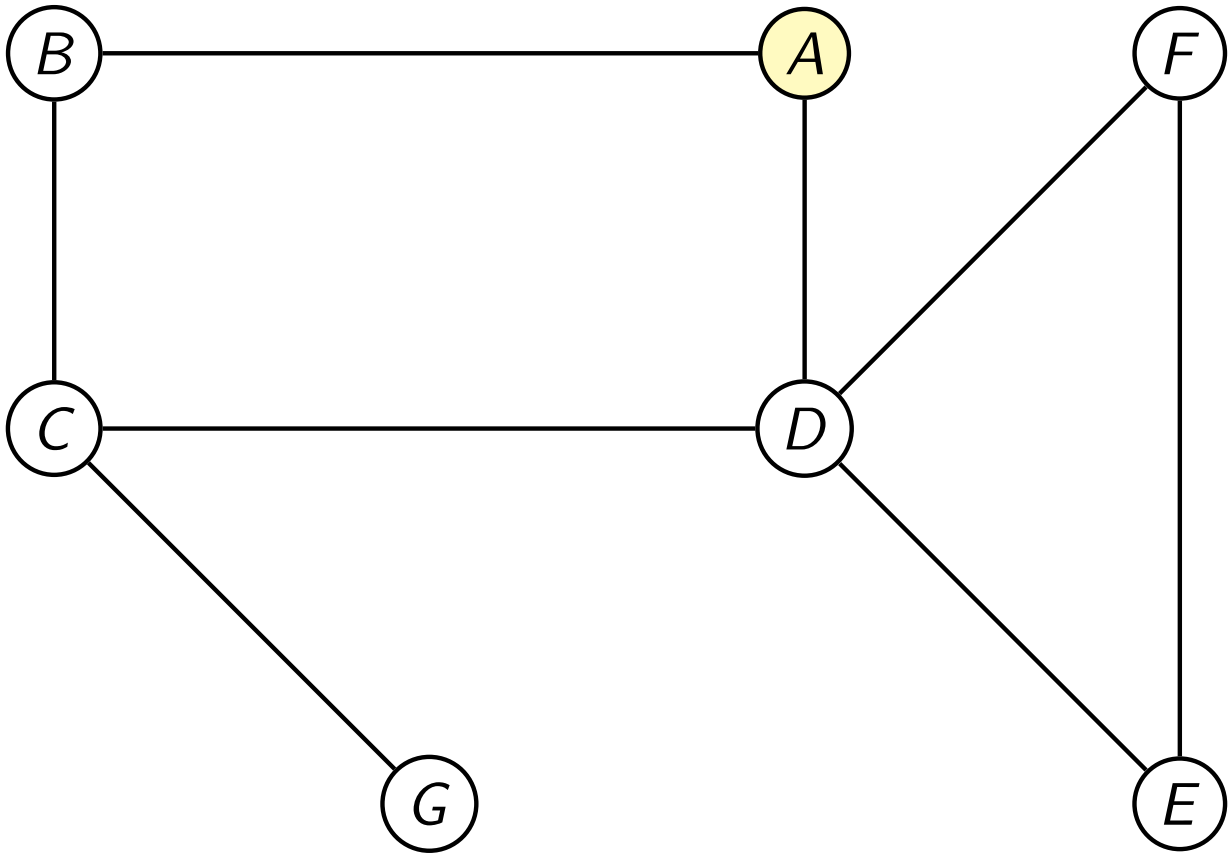
- for urettet sammenhengende grafer
- huske "back-pointers"

Ulike type kanter

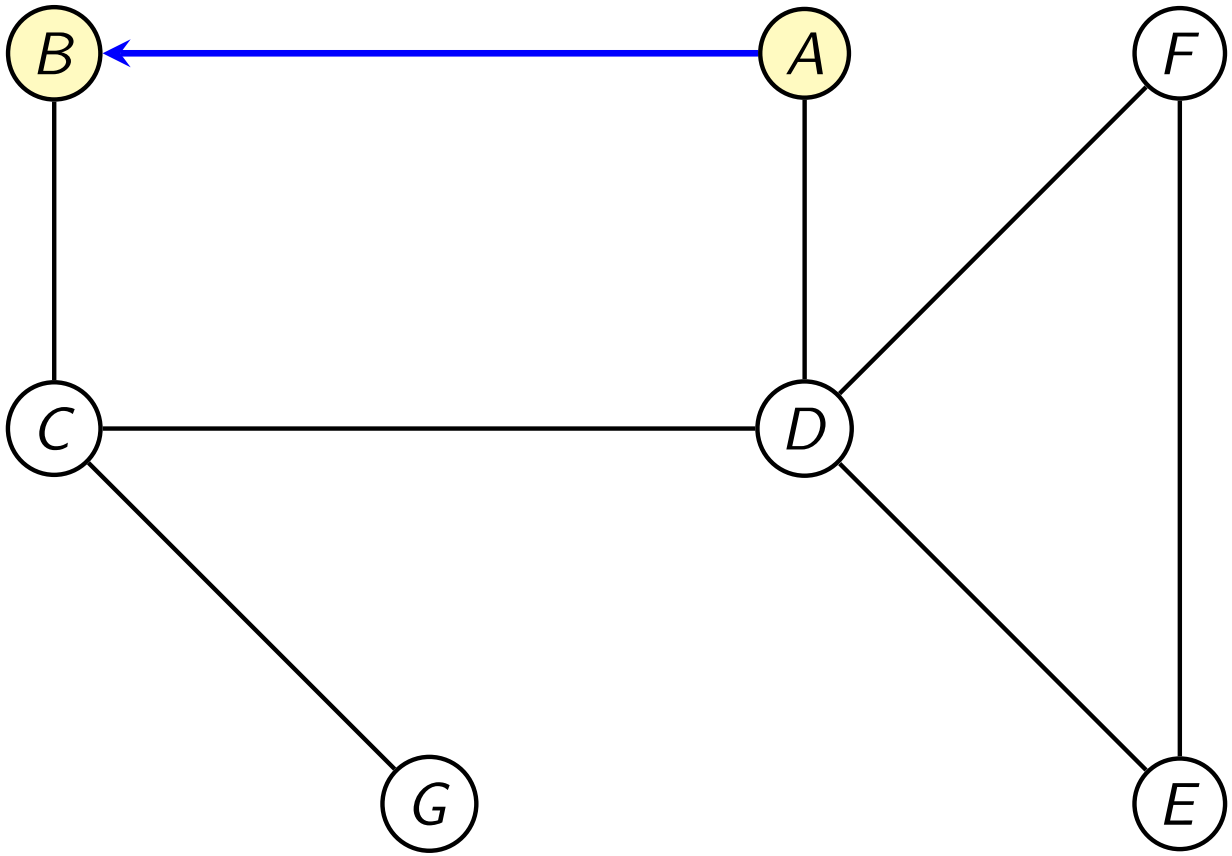
gitt en bestemt kjøring av dfs

1. tree edges
2. back edges

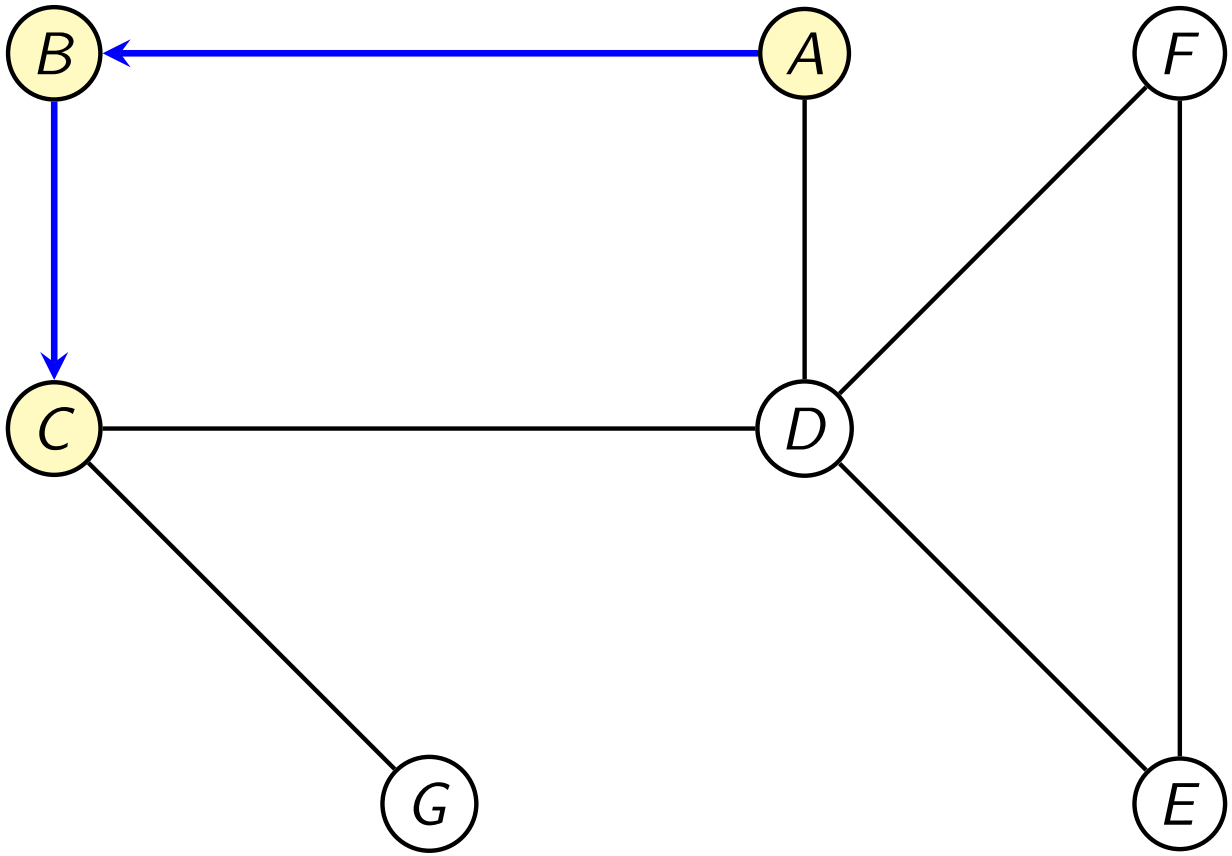
DFS, urettet graf (1)



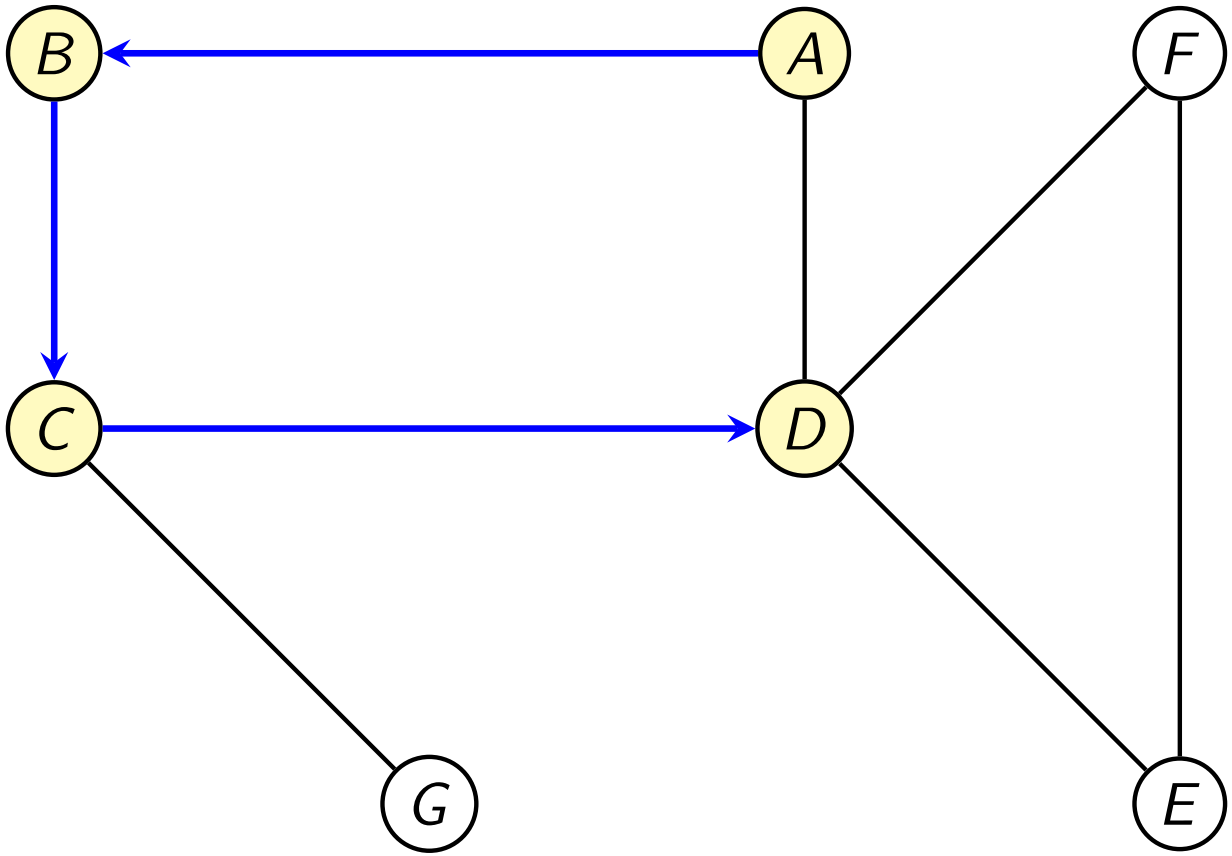
DFS, urettet graf (2)



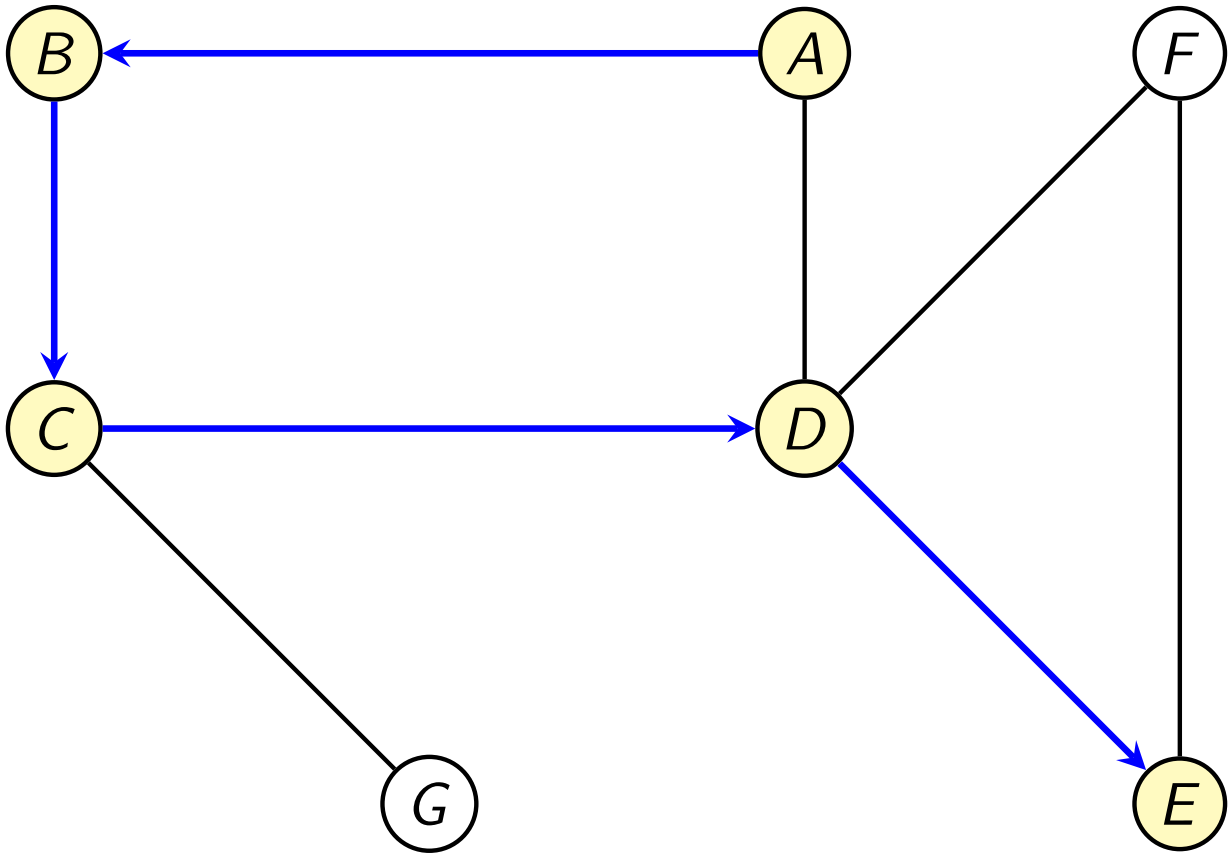
DFS, urettet graf (3)



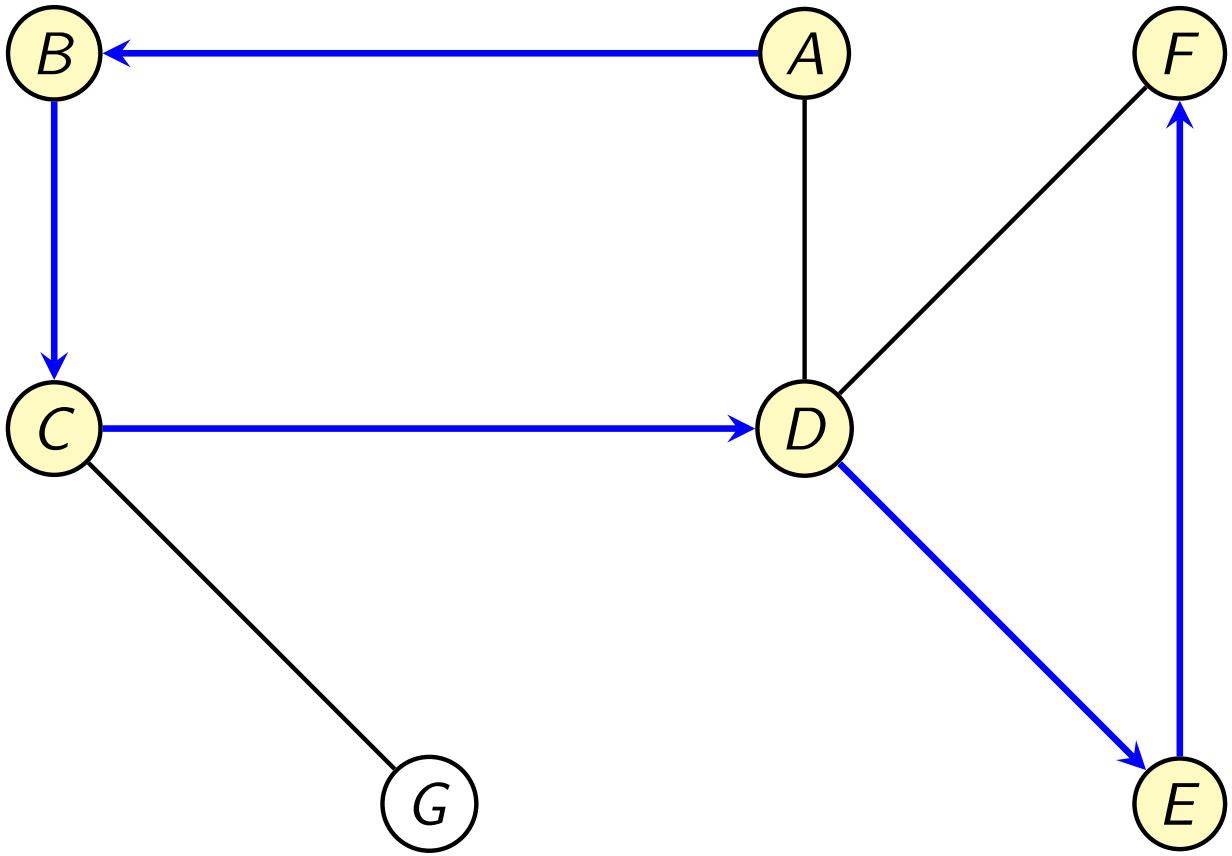
DFS, urettet graf (4)



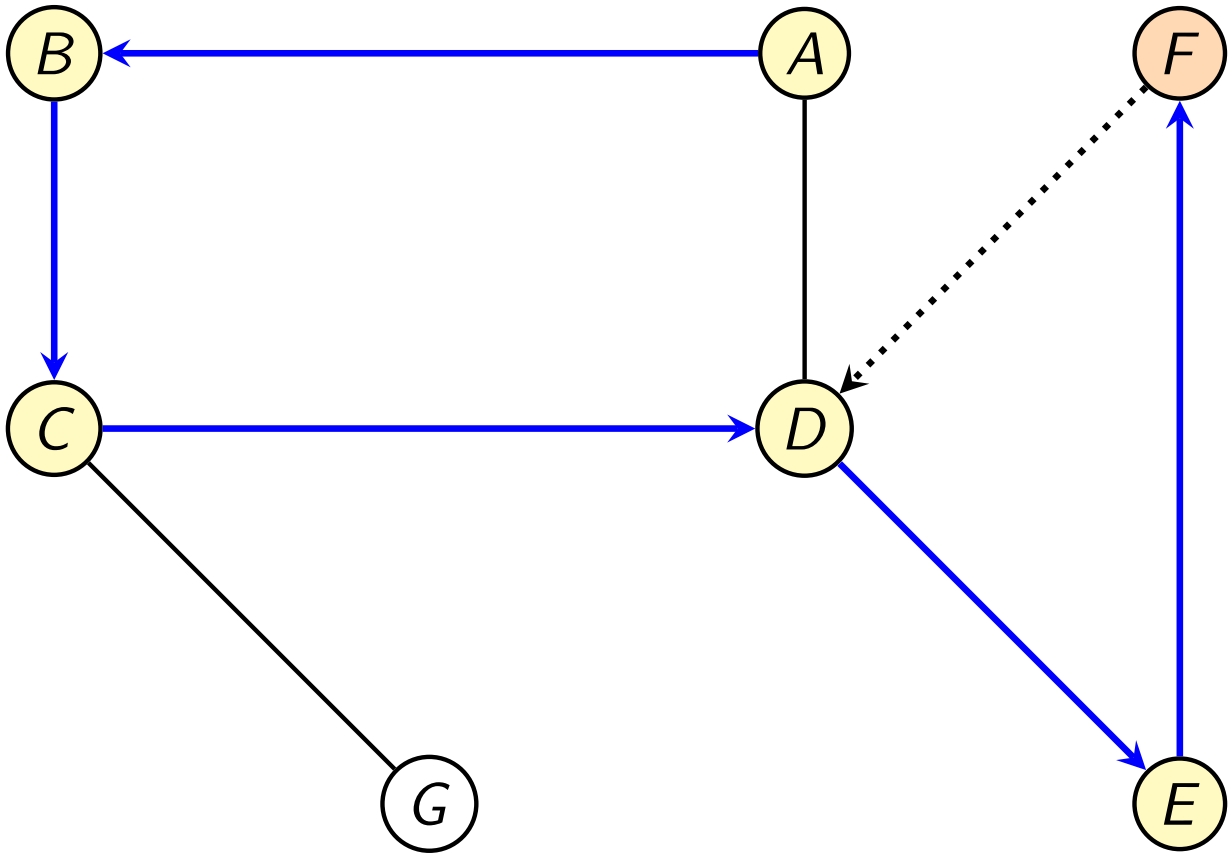
DFS, urettet graf (5)



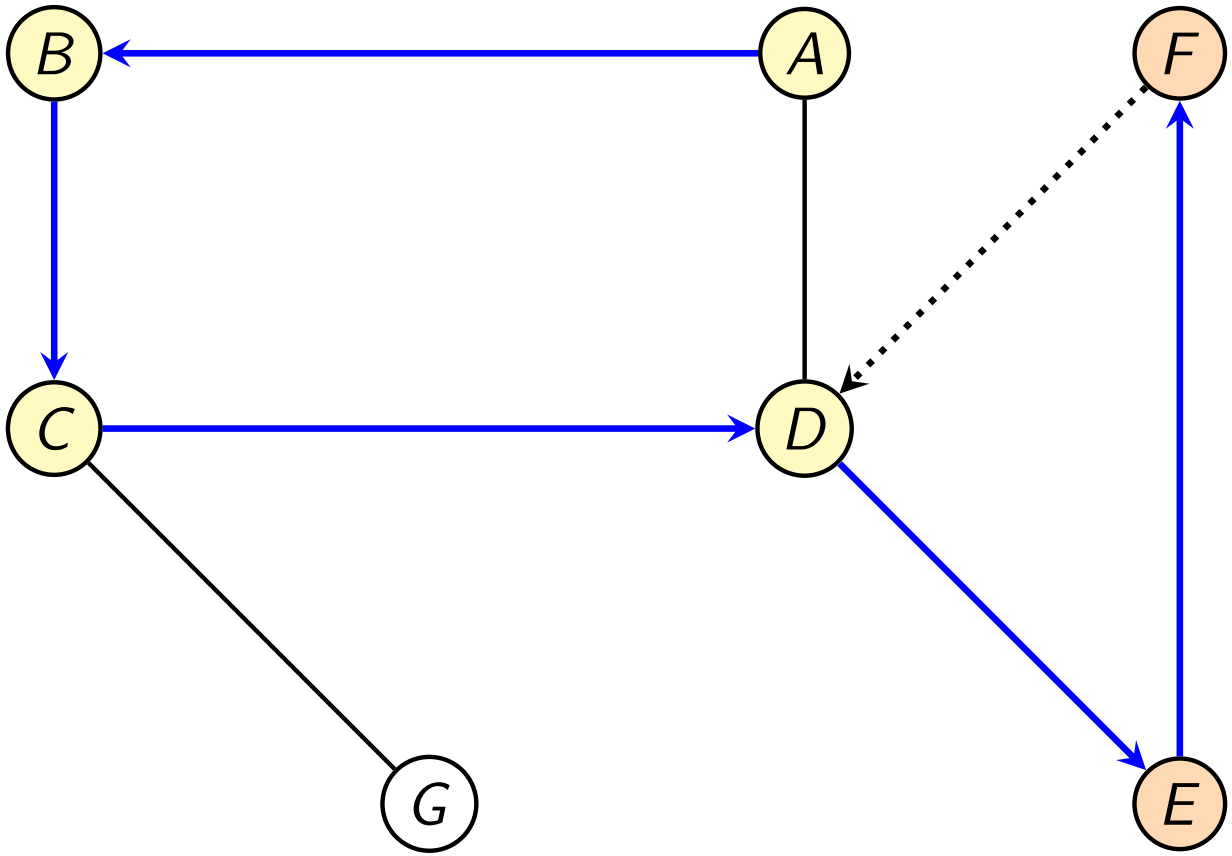
DFS, urettet graf (6)



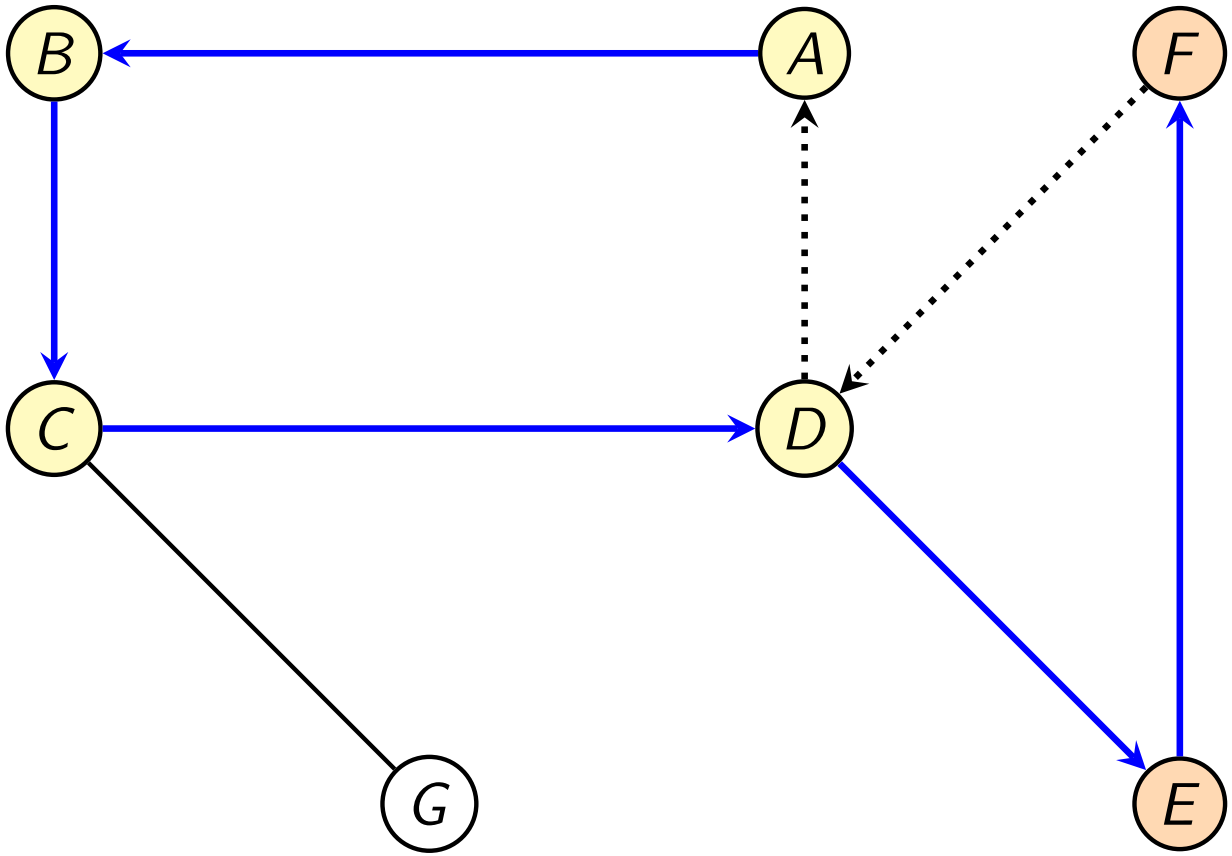
DFS, urettet graf (7)



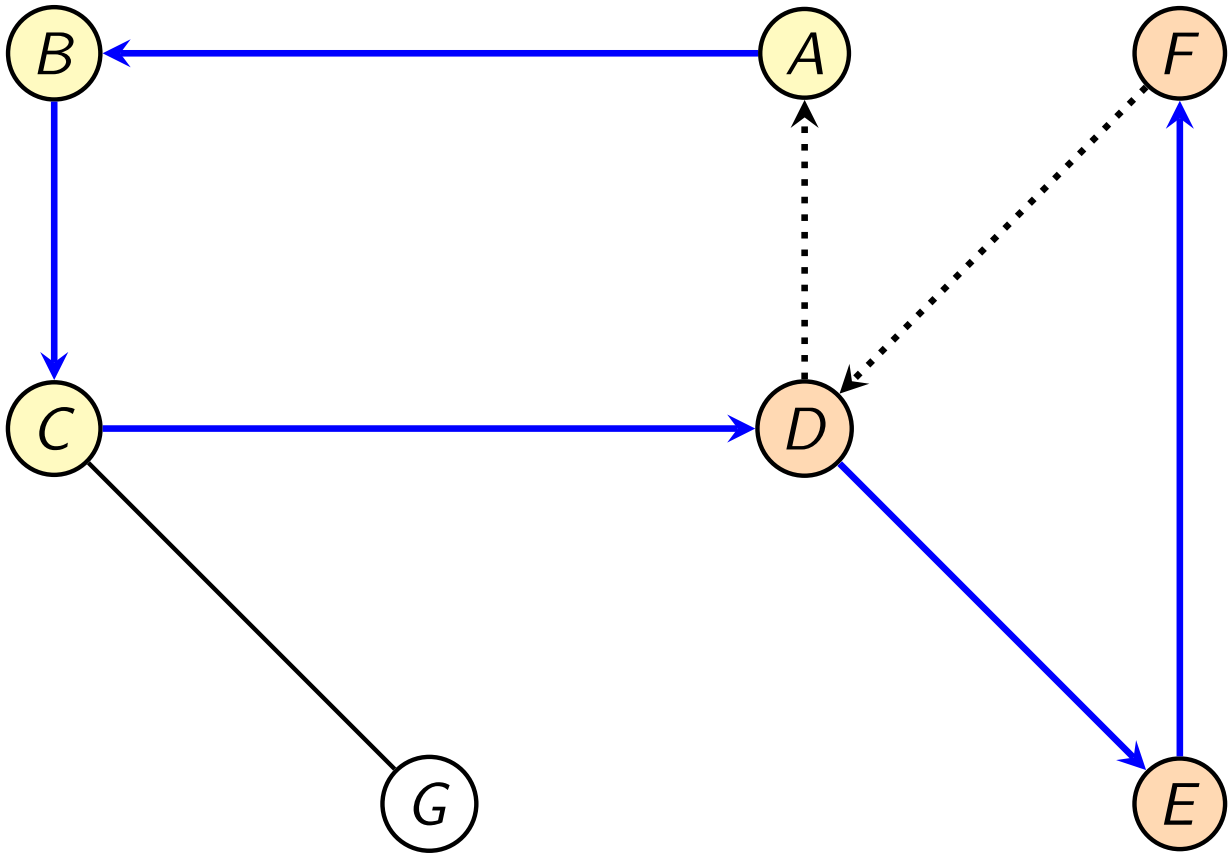
DFS, urettet graf (8)



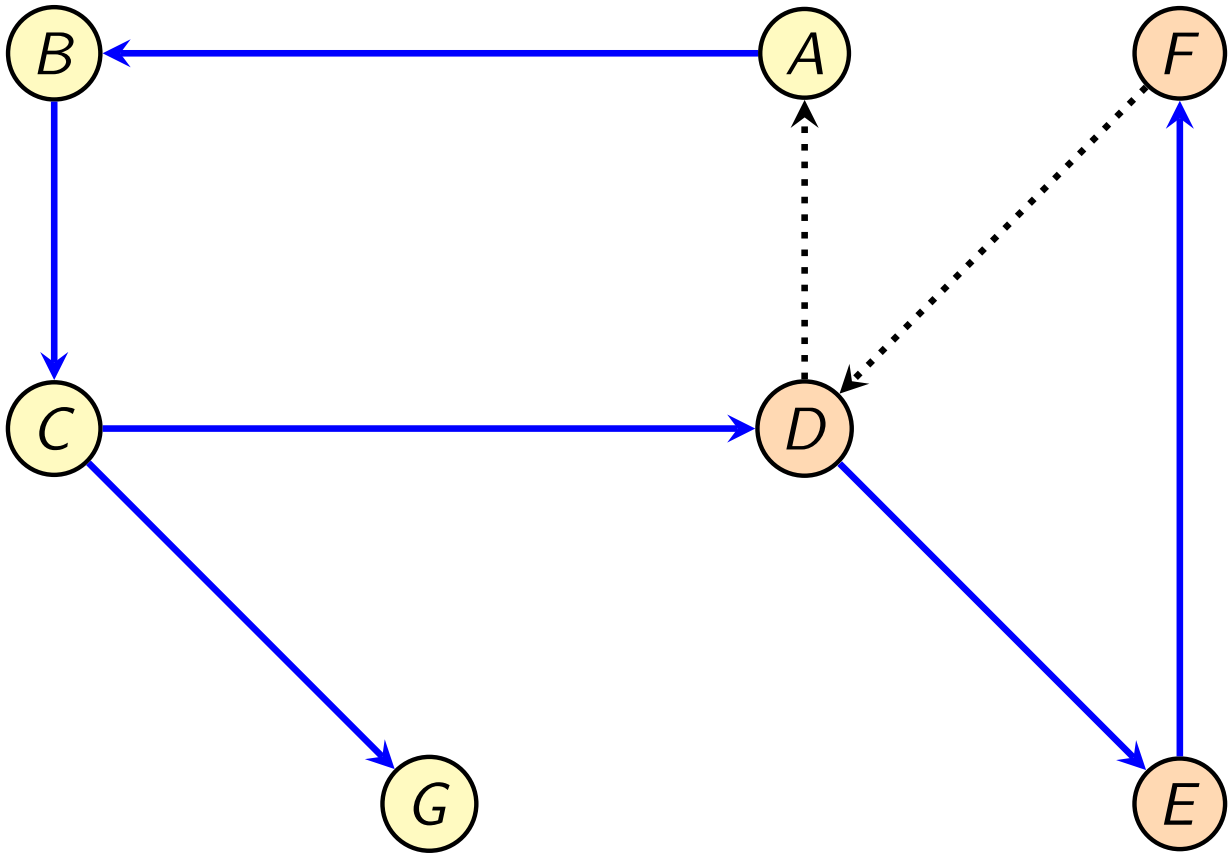
DFS, urettet graf (9)



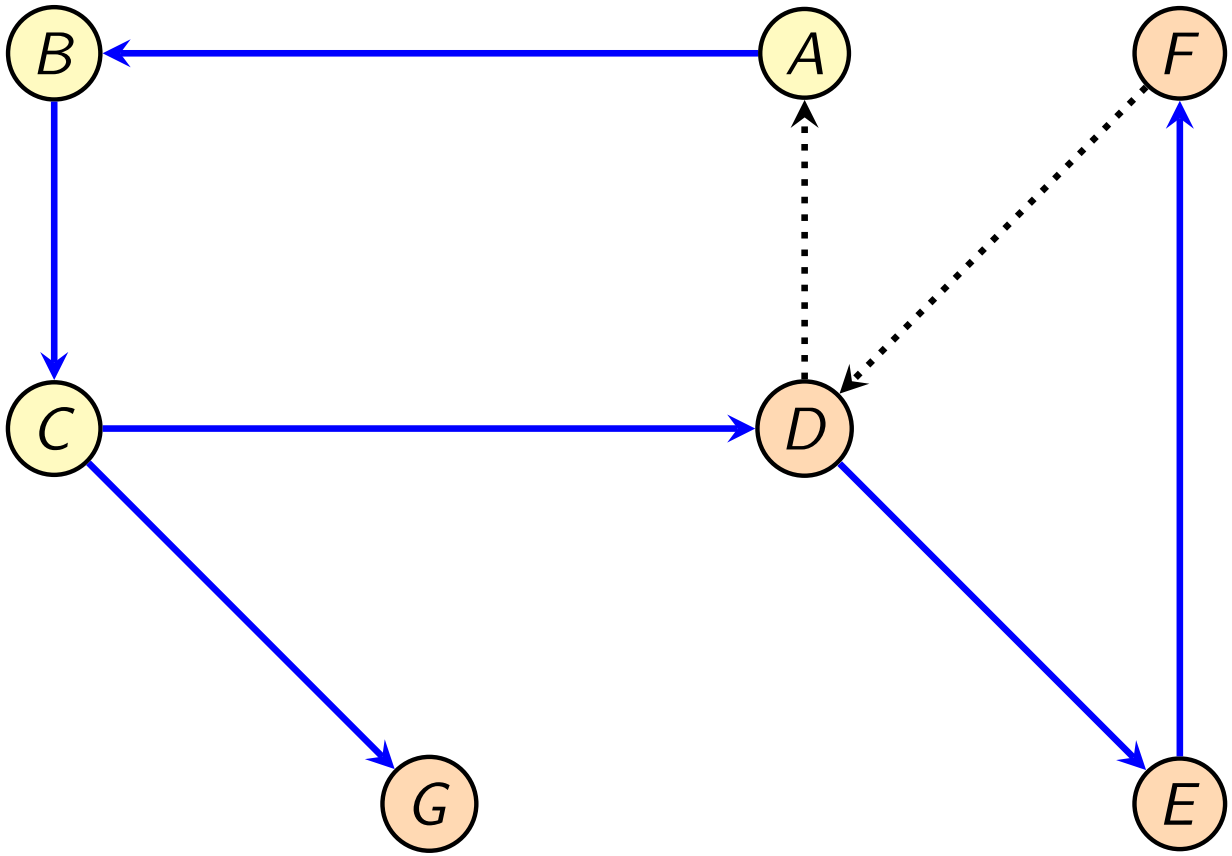
DFS, urettet graf (10)



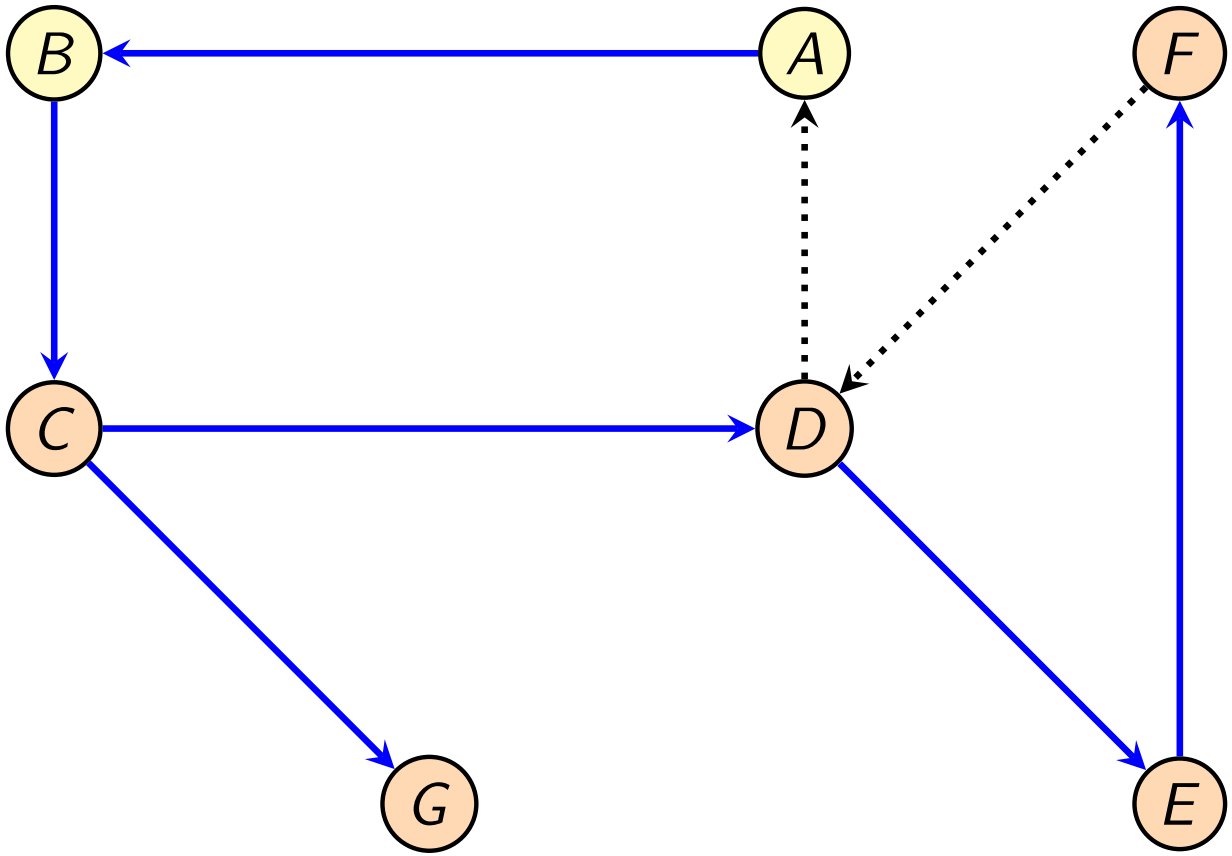
DFS, urettet graf (11)



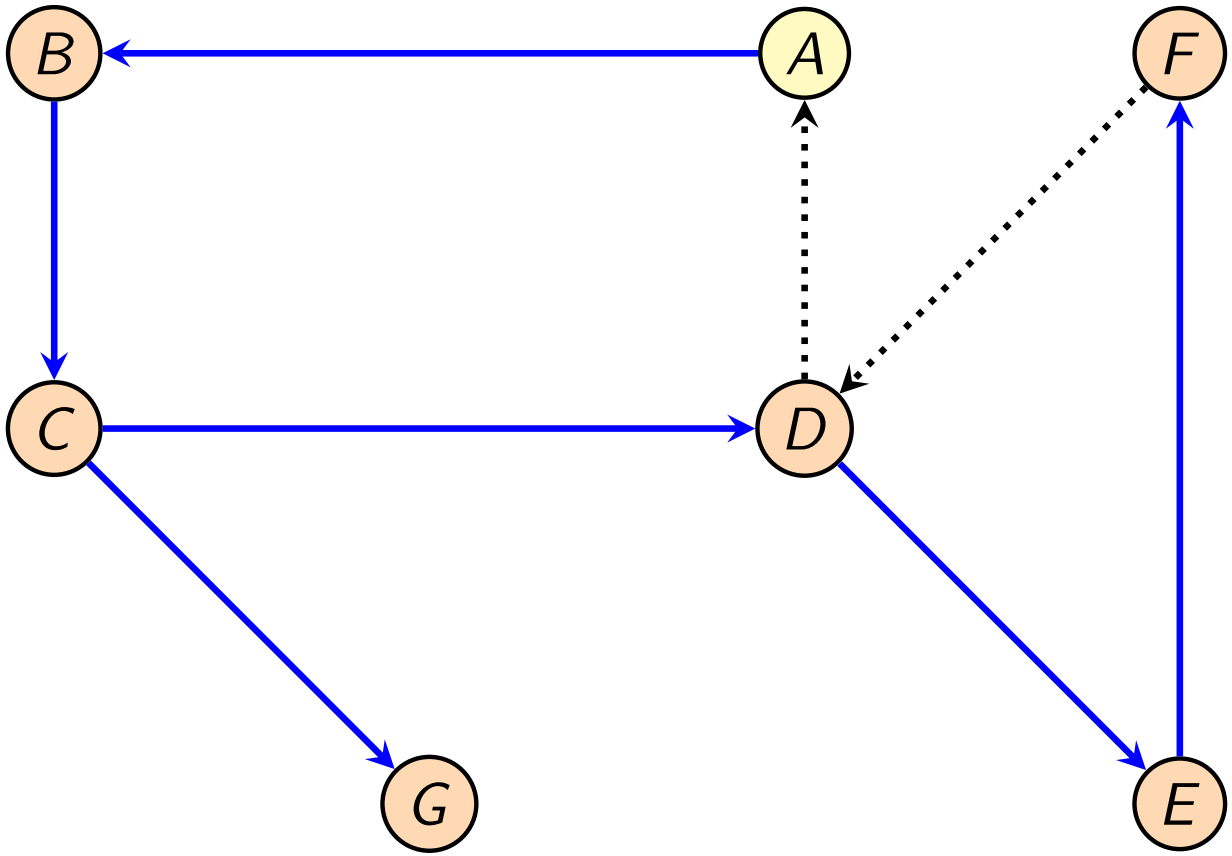
DFS, urettet graf (12)



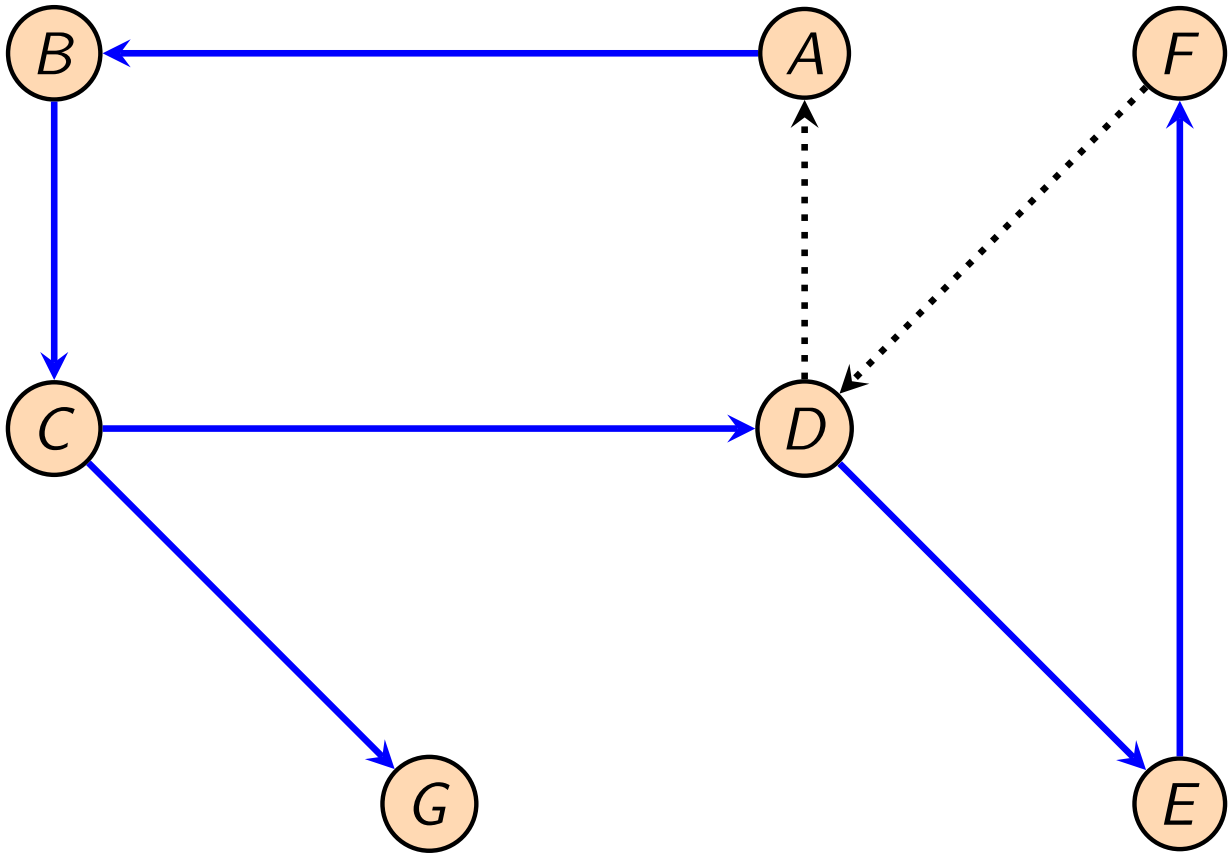
DFS, urettet graf (13)



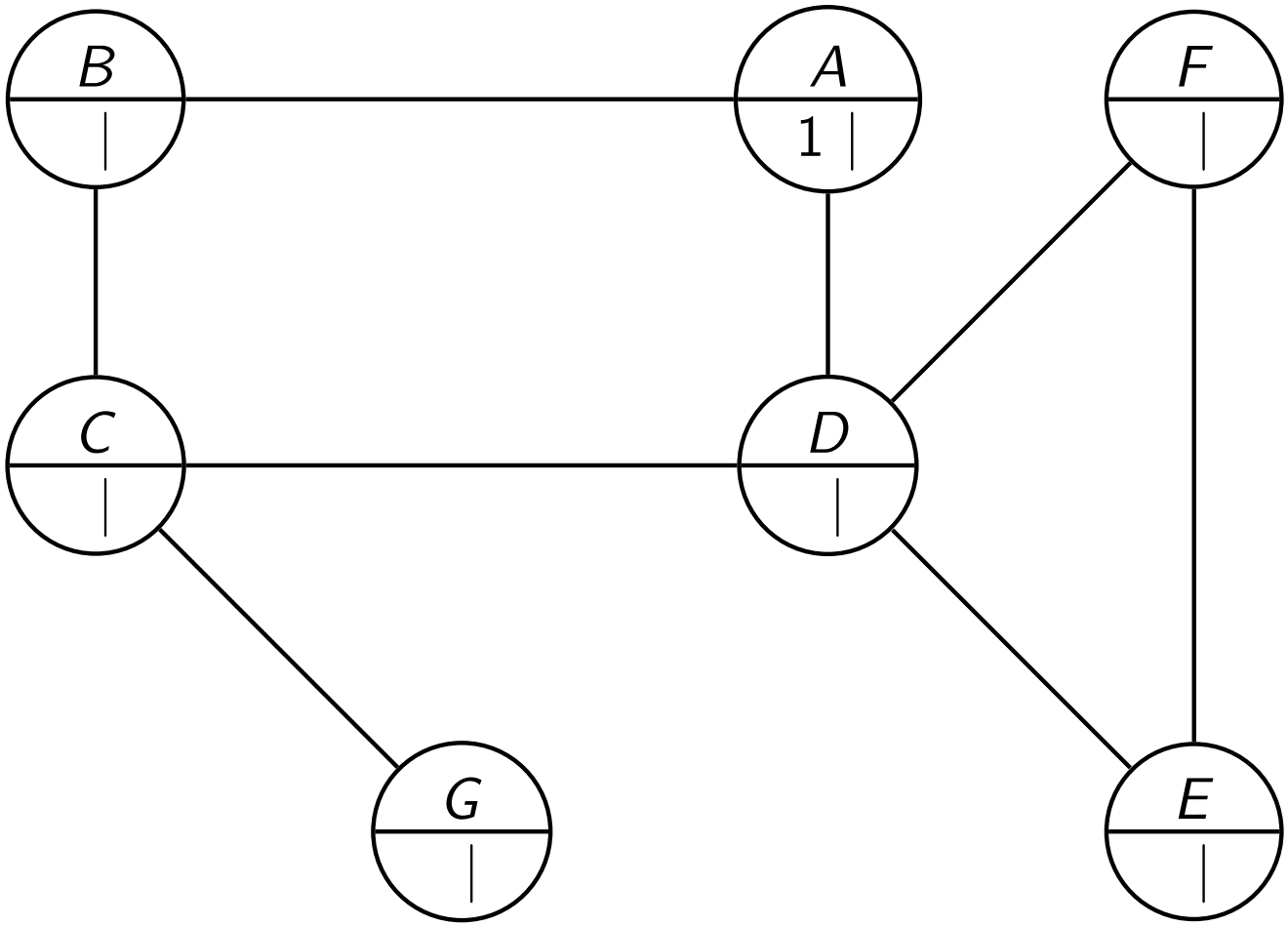
DFS, urettet graf (14)



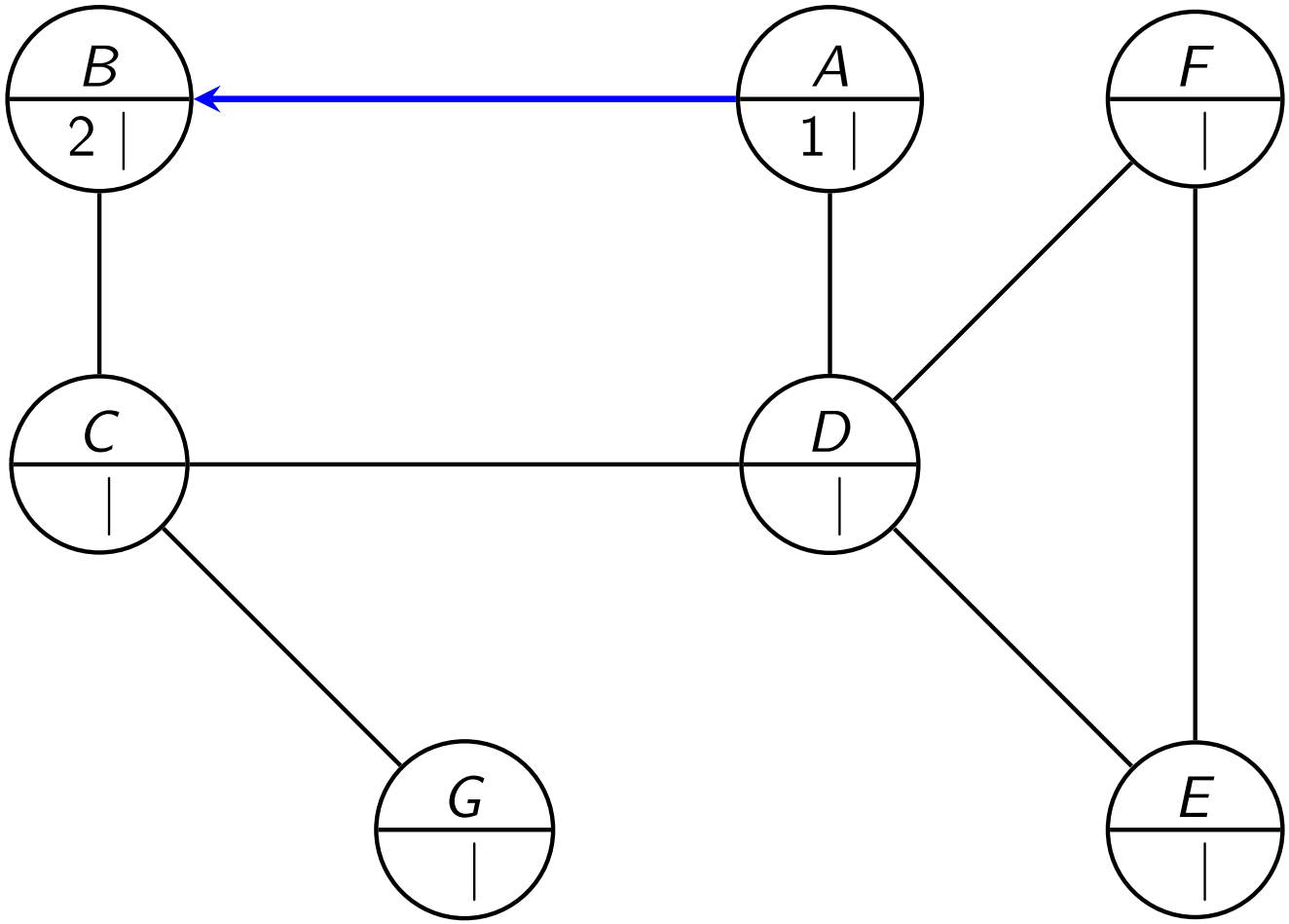
DFS, urettet graf (15)



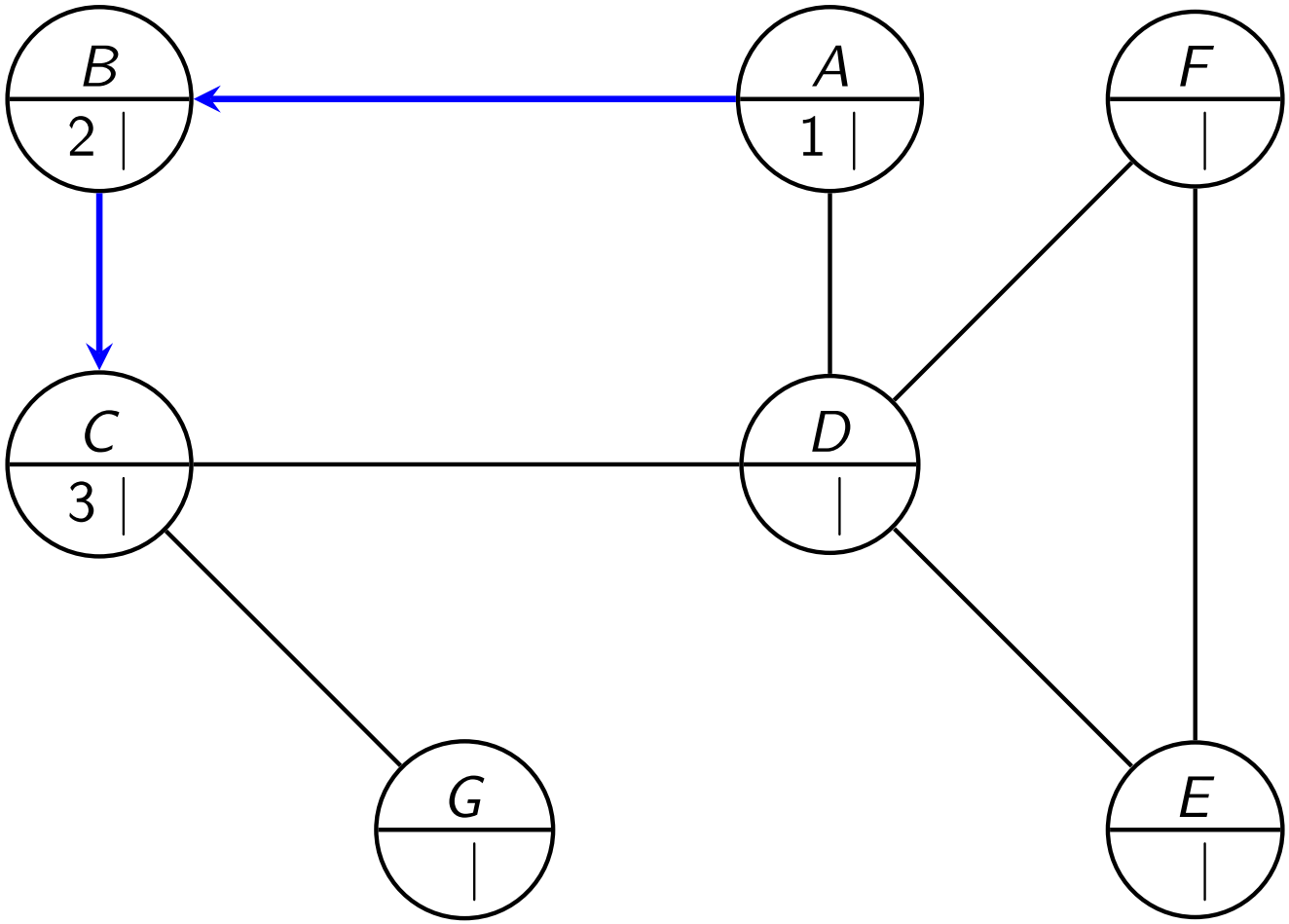
DFS: adding visiting time (1)



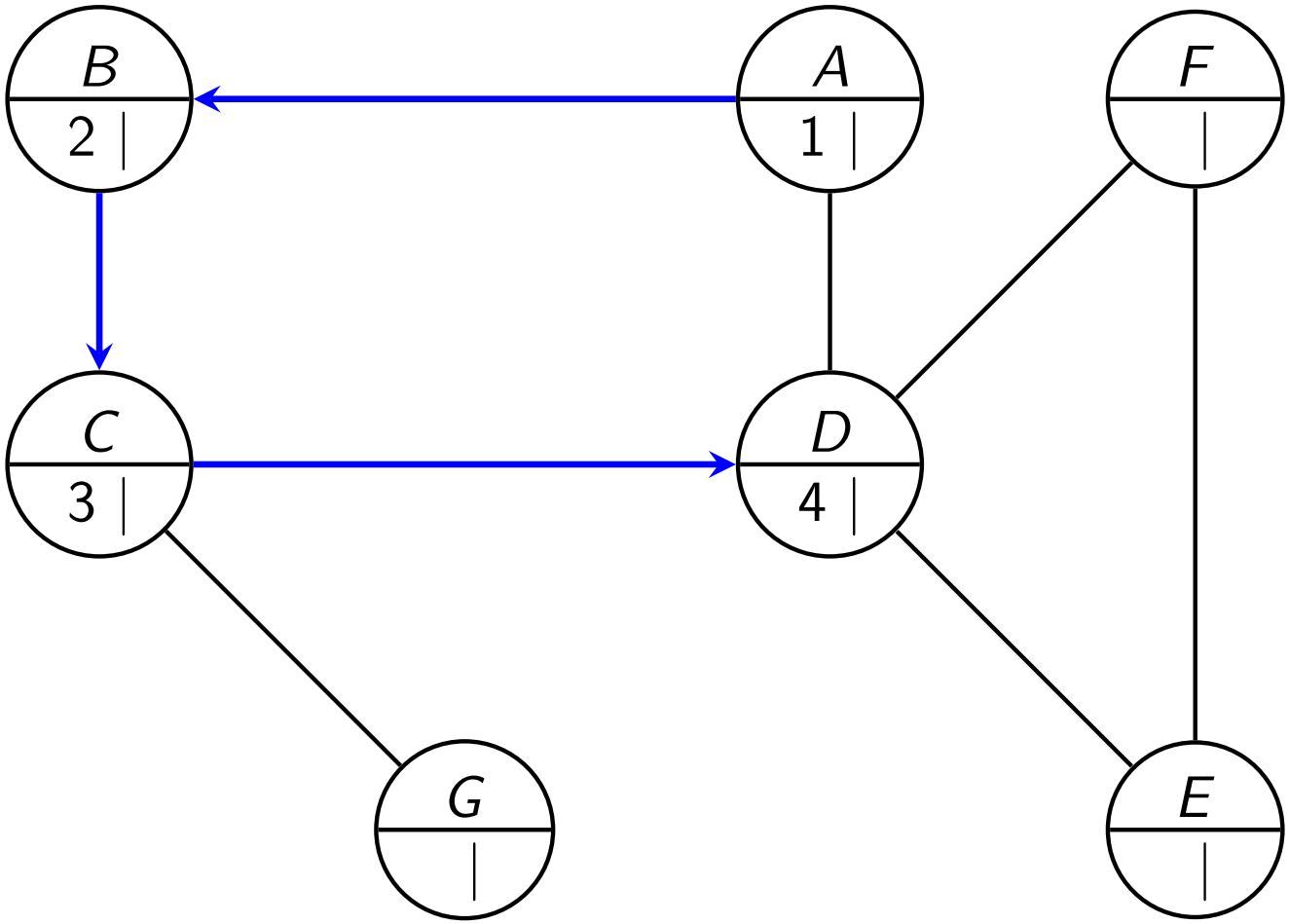
DFS: adding visiting time (2)



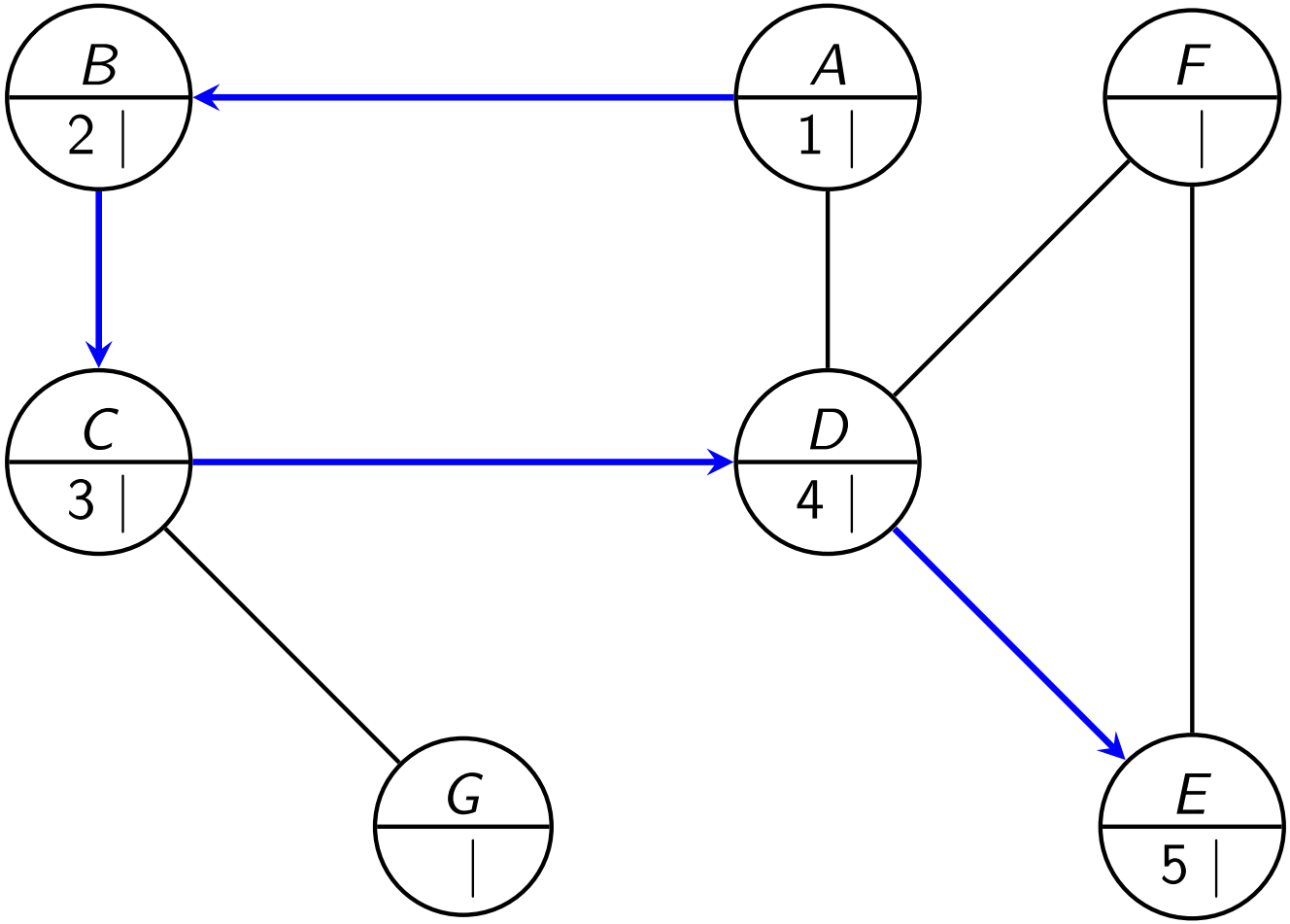
DFS: adding visiting time (3)



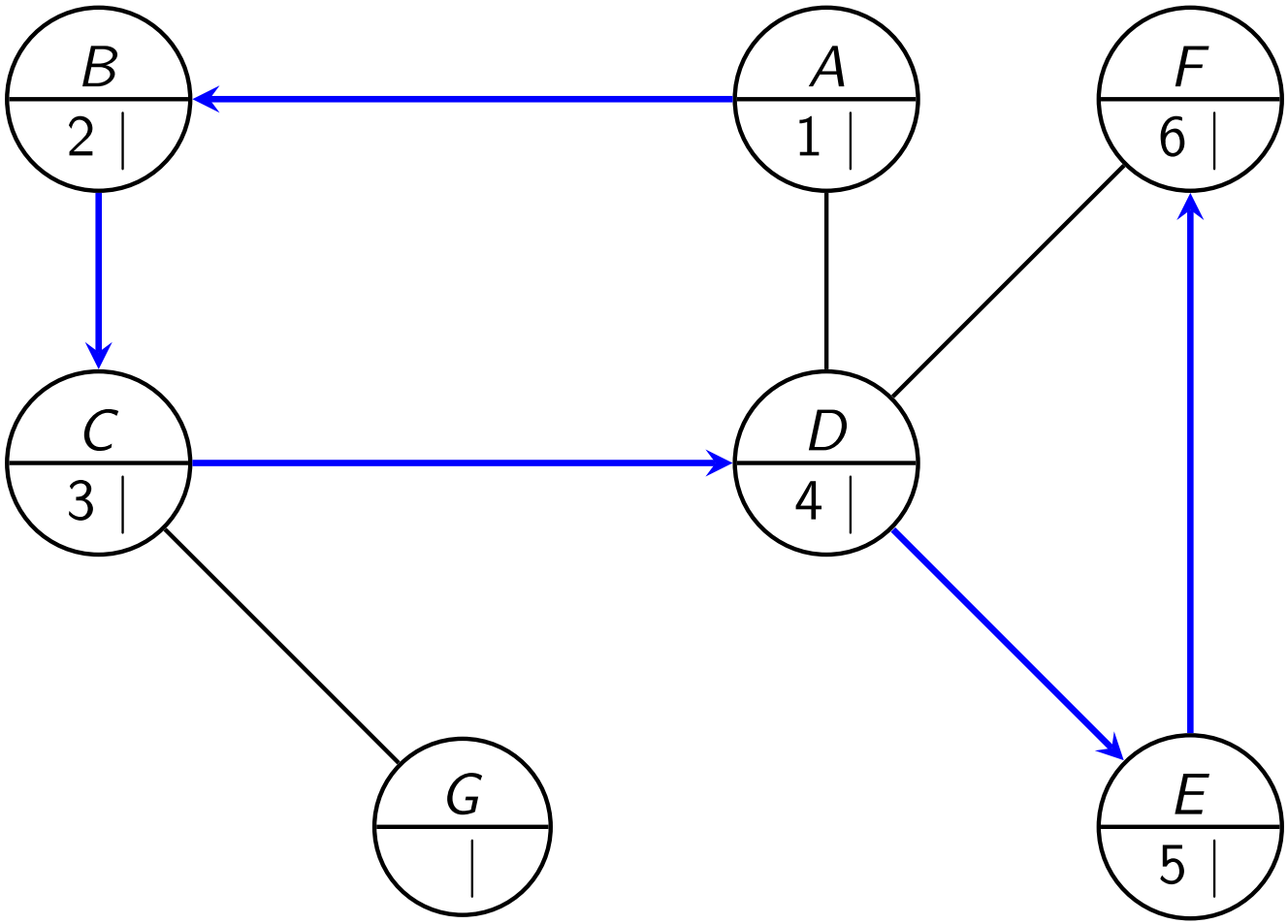
DFS: adding visiting time (4)



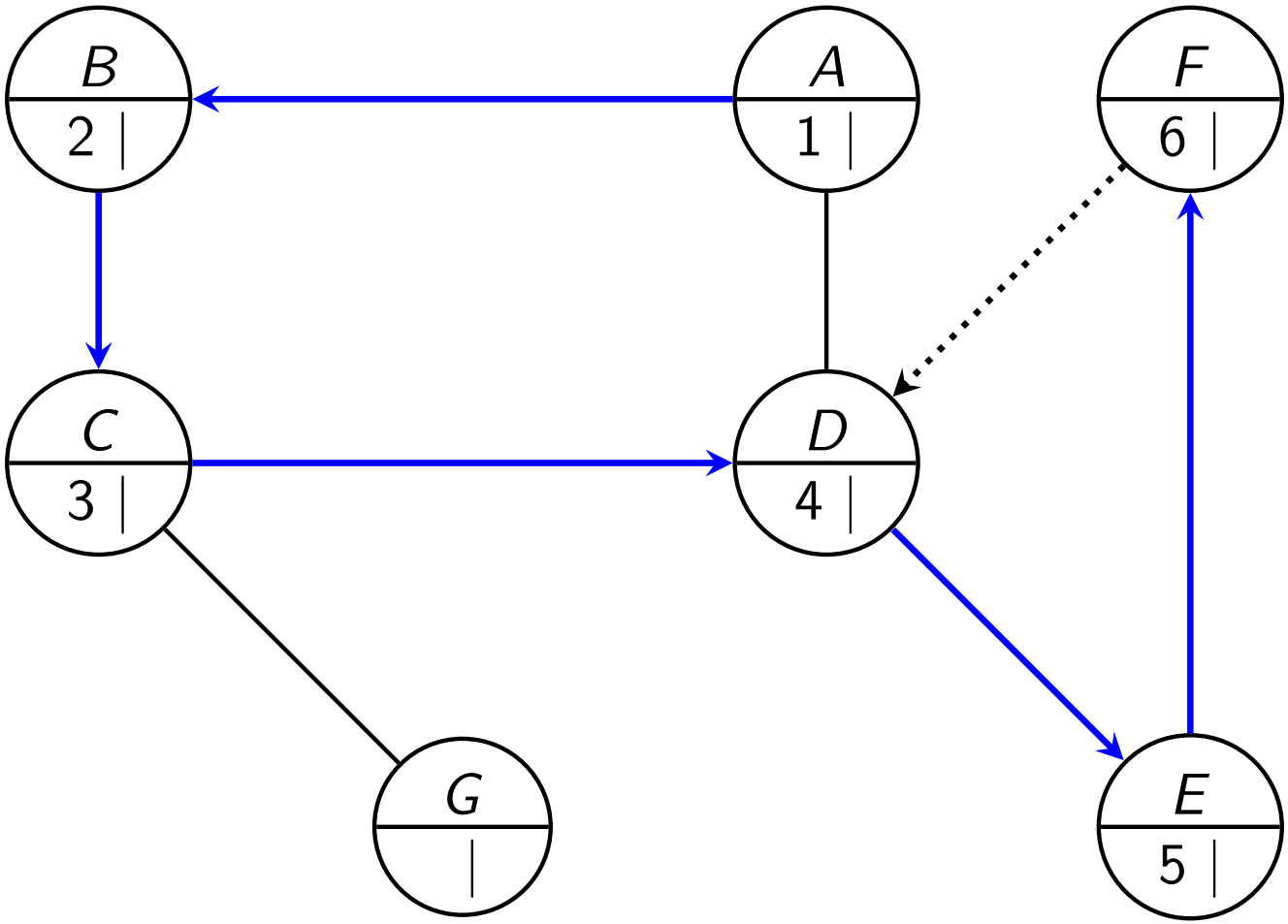
DFS: adding visiting time (5)



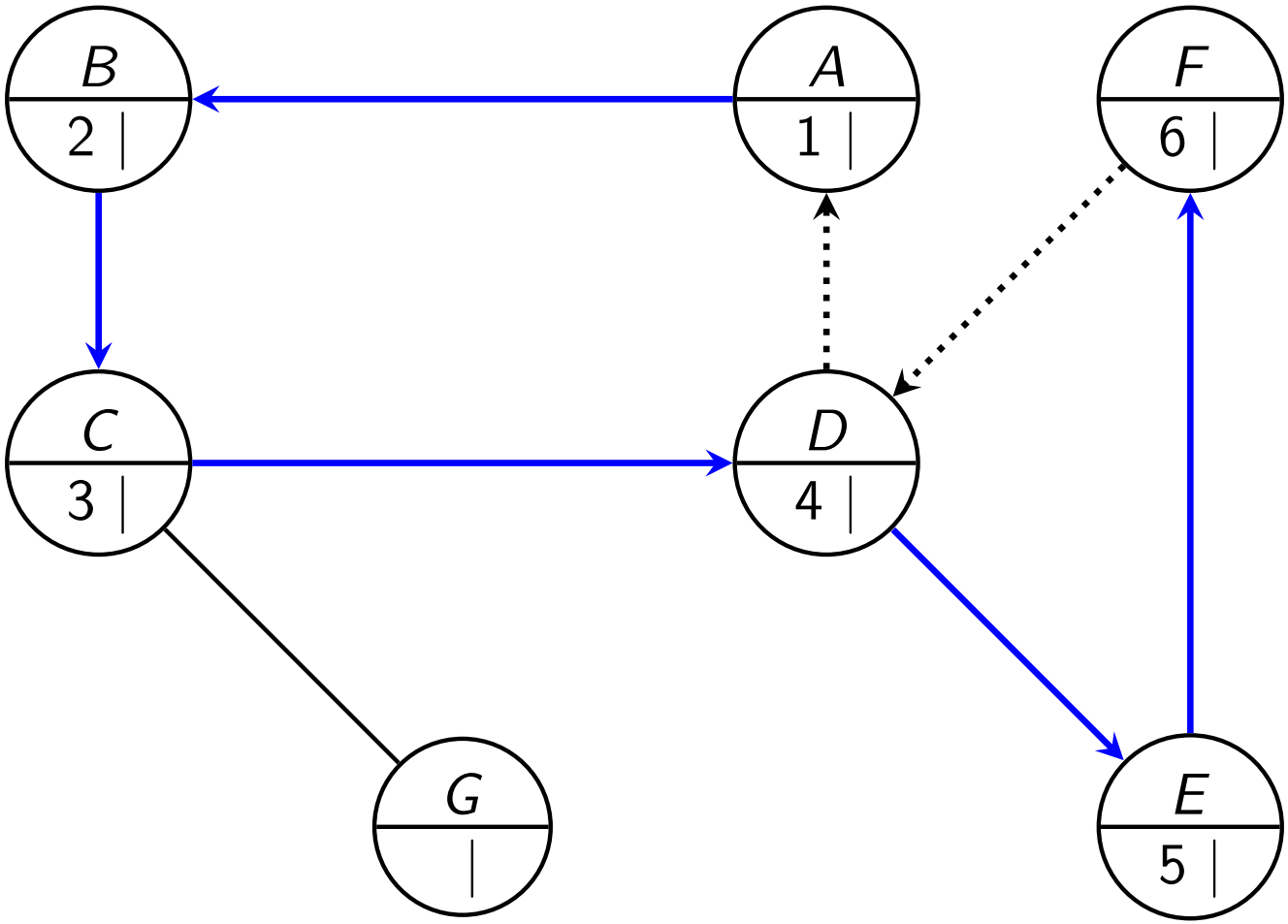
DFS: adding visiting time (6)



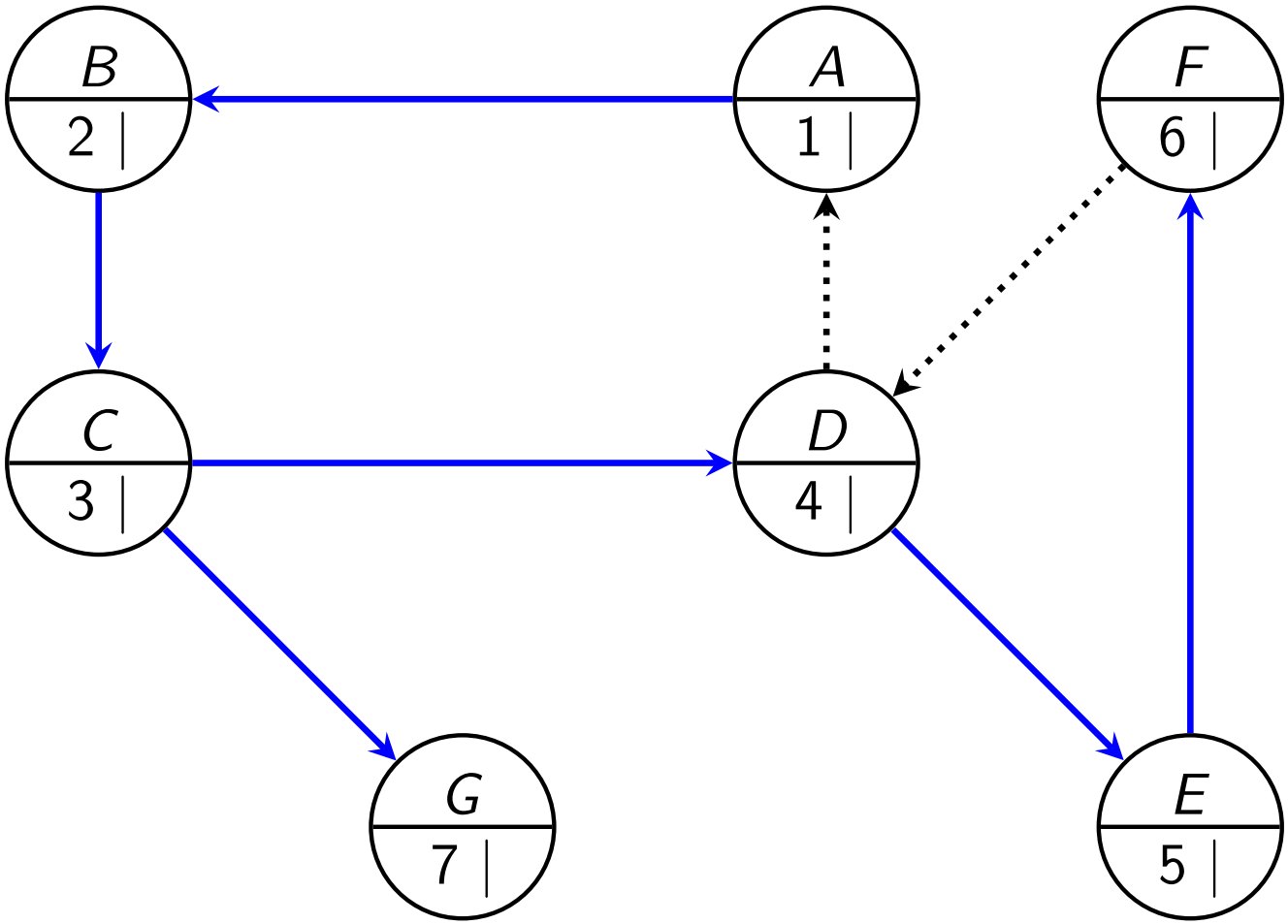
DFS: adding visiting time (7)



DFS: adding visiting time (8)



DFS: adding visiting time (9)



Biconnectivity

Biconnectivity

Definition

En sammenhengende urettet graf er bi-connected hvis det ikke er noen noder som ved fjerning gjør at grafen blir ikke sammenhengende. Slik node heter *cut-vertices* eller *articulation point*.

Egenskapen ved biconnectede grafer:
Det er to disjunkte stier mellom hvilke to noder som helst.

F.eks. Nettverk-feiltoleranse: dersom en ruter går ned, finnes det alltid en alternativ vei å rute datapakkene på.

Viktig å identifisere disse nodene.
Single point to failure!

Biconnectivity

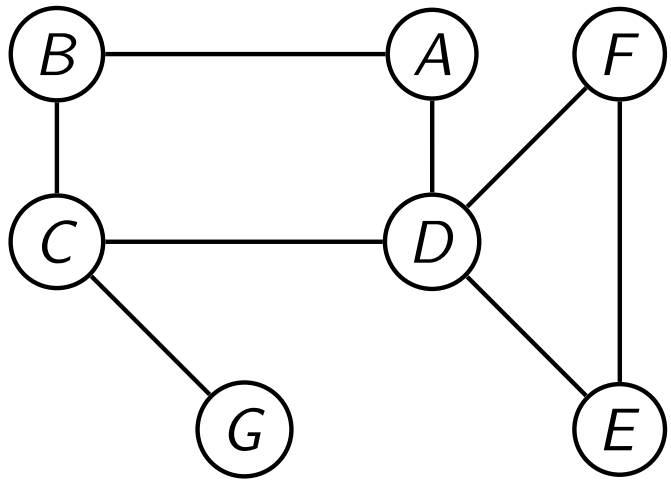
Definition

En sammenhengende urettet graf er bi-connected hvis det ikke er noen noder som ved fjerning gjør at grafen blir ikke sammenhengende. Slik node heter *cut-vertices* eller *articulation point*.

Egenskapen ved biconnectede grafer:
Det er to disjunkte stier mellom hvilke to noder som helst.

F.eks. Nettverk-feiltoleranse: dersom en ruter går ned, finnes det alltid en alternativ vei å rute datapakkene på.

Viktig å identifisere disse nodene.
Single point to failure!



Biconnectivity

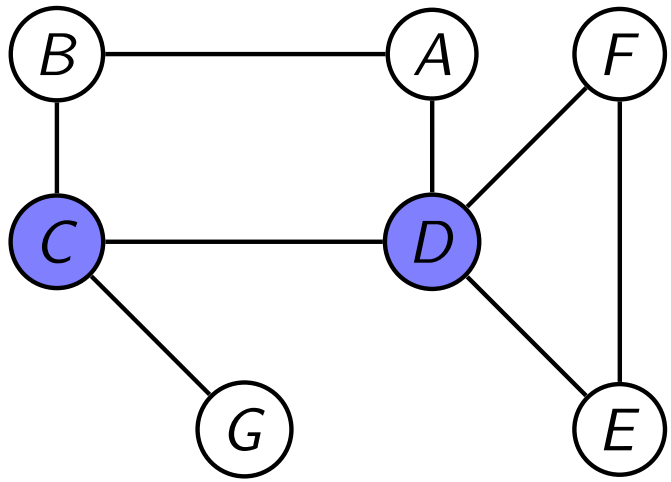
Definition

En sammenhengende urettet graf er bi-connected hvis det ikke er noen noder som ved fjerning gjør at grafen blir ikke sammenhengende. Slik node heter *cut-vertices* eller *articulation point*.

Egenskapen ved biconnectede grafer:
Det er to disjunkte stier mellom hvilke to noder som helst.

F.eks. Nettverk-feiltoleranse: dersom en ruter går ned, finnes det alltid en alternativ vei å rute datapakkene på.

Viktig å identifisere disse nodene.
Single point to failure!

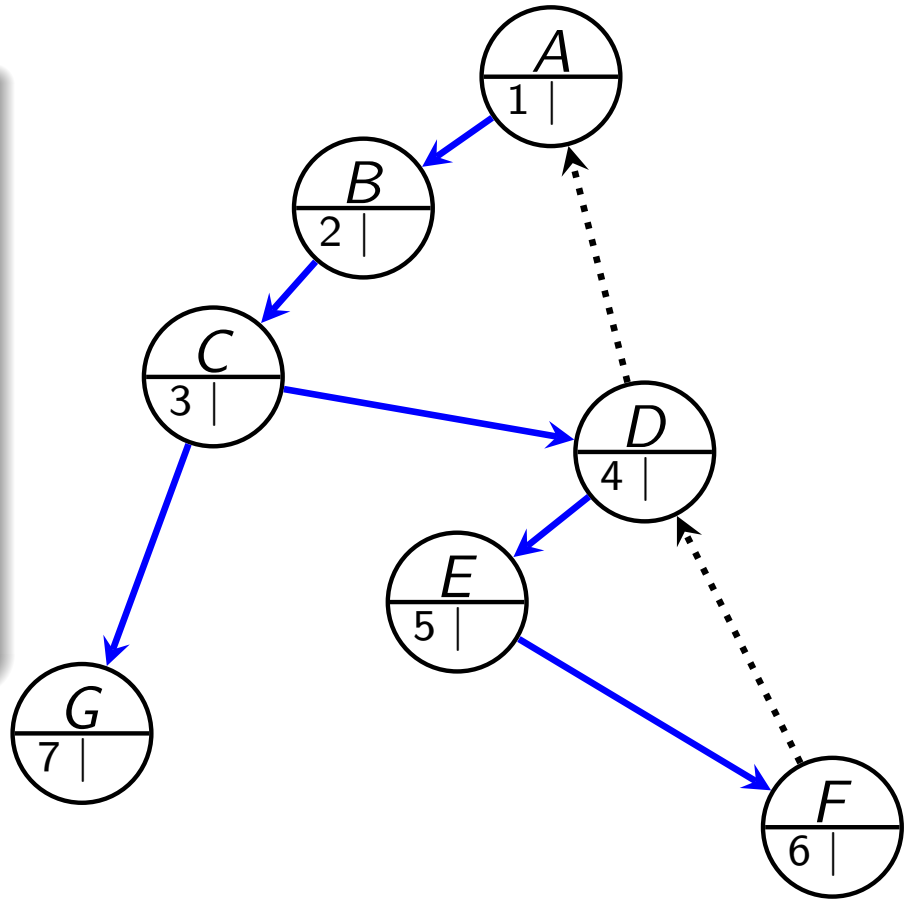


Er grafen bi-connected?

1. DFS-spenntre

Konstruer et dfs spenntre fra en node **v** av **G**

- Alle kantene (**v**, **w**) i **G** er representert i treet som enten en tree edge eller som en back edge
- Nummerer nodene når vi besøker dem

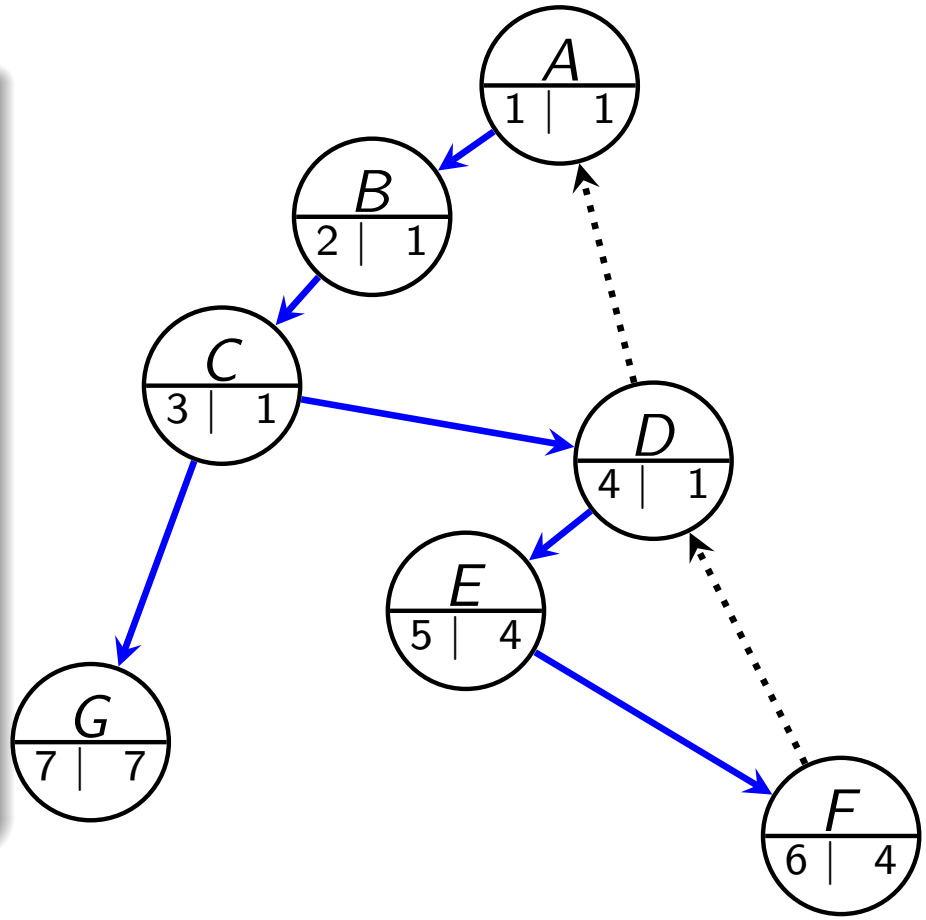


Er grafen bi-connected?

2. Low-number

for hver node i treet: beregn low-nummeret

- laveste noden som kan nås ved å ta 0 eller flere tree edge etterfulgt av 0 eller 1 back edge
- Dette gjør vi like før vi trekker oss tilbake (idet vi er ferdig med kallet)
- note: på dette tidspunktet har funnet low for alle barna til noden

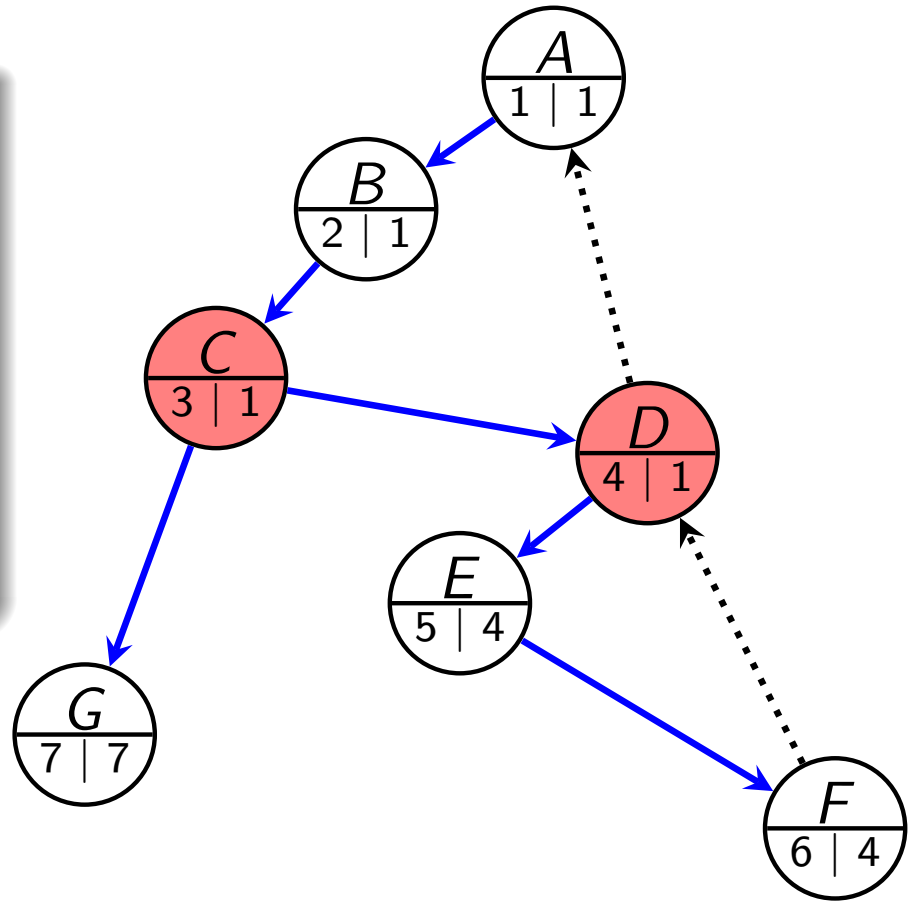


Er grafen bi-connected?

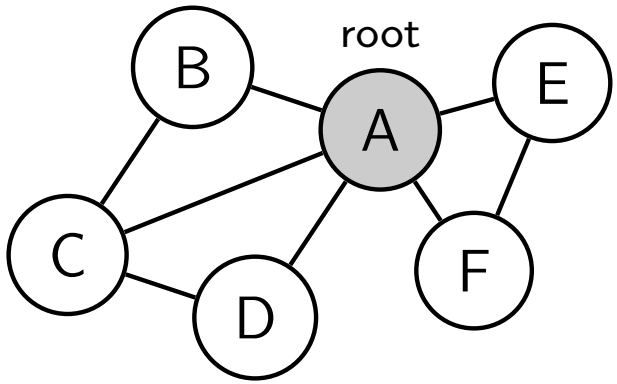
3. Cut-vertex

En node **v** er en cutvertex i **G** hvis:

1. **v** er rotnoden av $DFS_{\mathbf{G}}$-treet og har to eller flere tree edges
2. **v** *ikke* er rotnoden av $DFS_{\mathbf{G}}$ og det finnes en tree edge $(\mathbf{v}, \mathbf{w})$ slik at $\text{low}(w) \geq \text{num}(v)$



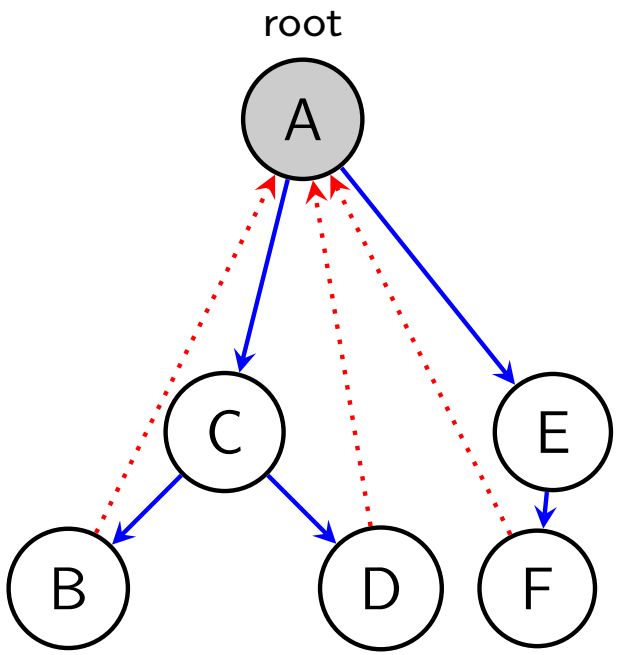
Rot



Roten av DFS-treet er en cutvertex hvis den har to eller flere utgående tree edges.

- tilbaketrekking må gå gjennom rooten for å komme fra den ene til den andre sub-treet.

Rot

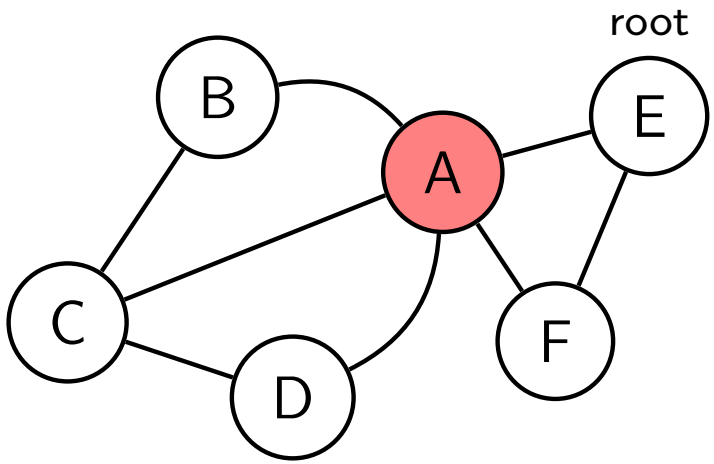


Roten av DFS-treet er en cutvertex hvis den har to eller flere utgående tree edges.

- tilbaketrekking må gå gjennom rooten for å komme fra den ene til den andre sub-treet.

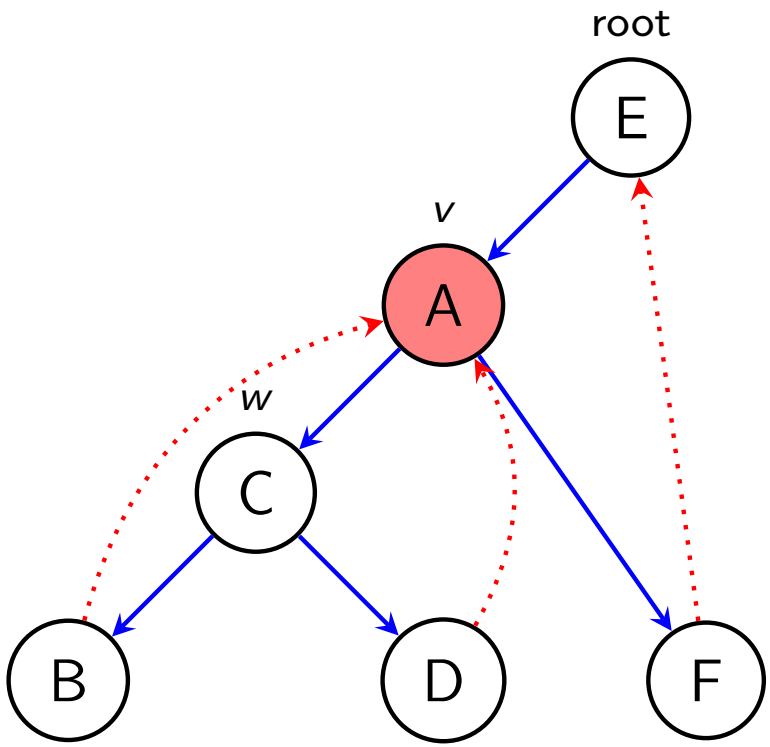
Ikke-rot node

Et ikke-rot node **v** er en cutvertex hvis den har et barn **w** slik at *ingen back edge* som starter i subtreet av **w** når en *predesessornode* til **v**.



Ikke-rot node

Et ikke-rot node **v** er en cutvertex hvis den har et barn **w** slik at *ingen back edge* som starter i subtreet av **w** når en *predesessornode* til **v**.



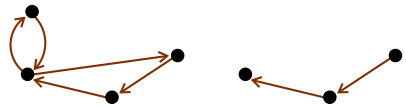
```
void dfs(v)
  {
    v.num    = counter++;          //
    v.status = visited;            // I am visiting now
    for each w adjacent to v
      if w.status ≠ visited
        w.parent  = v;             // remember parent
        dfs (w)                    // recur
  }
```

```
void assignlow(Node v) {
      v.low = v.num;                          // at least num
      for each w adjacent to v  {             // neighbors in G!
        if (w.num > v.num)                    //forward edge
        {
            assignlow(w);
            if (w.low ≥ v.num)
                system.out.println(v +
                        ''an articulation point!'');
            v.low = min(v.low, w.low);
        }
        else                                  // w.num ≤ v.num
            if (v.parent ≠ w) //back edge
                v.low = min (v.low, w.num);
      }
   }
```

- test for root not shown
- the two traversals (dfs + `assignlow`) can be combined into 1 pass.

Strongly Connected Components

rettet graf & DFS



En rettet graf er sterkt sammenhengende hvis og bare hvis vi fra hver eneste node **v** klarer å besøke alle de andre nodene i grafen ved et dybde-først søk fra **v**

Strongly Connected Components (SCC)

partisjonere G i såkalt strongly connected components

Definition (SCC)

Gitt en rettet graf $G = (V, E)$. En *strongly connected component* av G er en maksimal sett av noder $U \subseteq V$ s.t.: for alle $u_1, u_2 \in U$ vi har at $u_1 \rightarrow^* u_2$ and $u_2 \rightarrow^* u_1$.

- Ide: G og G^t har den samme SCC's $\Longrightarrow$ bruk dfs 2 ganger, en gang på G og en gang på den reverserte grafen[1] G^t
- kompleksitet: lineær tid $\mathcal{O}(E + V)$

[1] Gitt $G = (V, E)$ da er $G^t = (V, E^t)$ hvor $(v, u) \in E^t$ iff $(u, v) \in E$

Strongly Connected Components (SCC)

partisjonere G i såkalt strongly connected components

Definition (SCC)

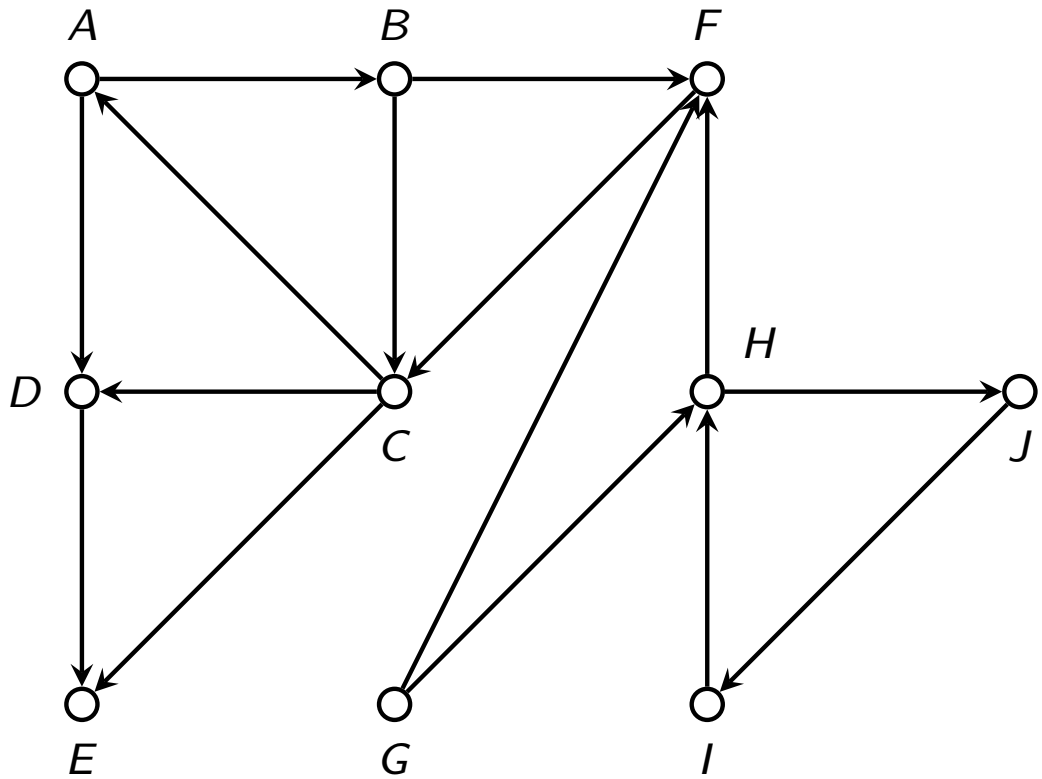
Gitt en rettet graf $G = (V, E)$. En *strongly connected component* av G er en maksimal sett av noder $U \subseteq V$ s.t.: for alle $u_1, u_2 \in U$ vi har at $u_1 \rightarrow^* u_2$ and $u_2 \rightarrow^* u_1$.

- Ide: G og G^t har den samme SCC's $\implies$ bruk dfs 2 ganger, en gang på G og en gang på den reverserte grafen[1] G^t
- kompleksitet: lineær tid $\mathcal{O}(E + V)$

[1] Gitt $G = (V, E)$ da er $G^t = (V, E^t)$ hvor $(v, u) \in E^t$ iff $(u, v) \in E$

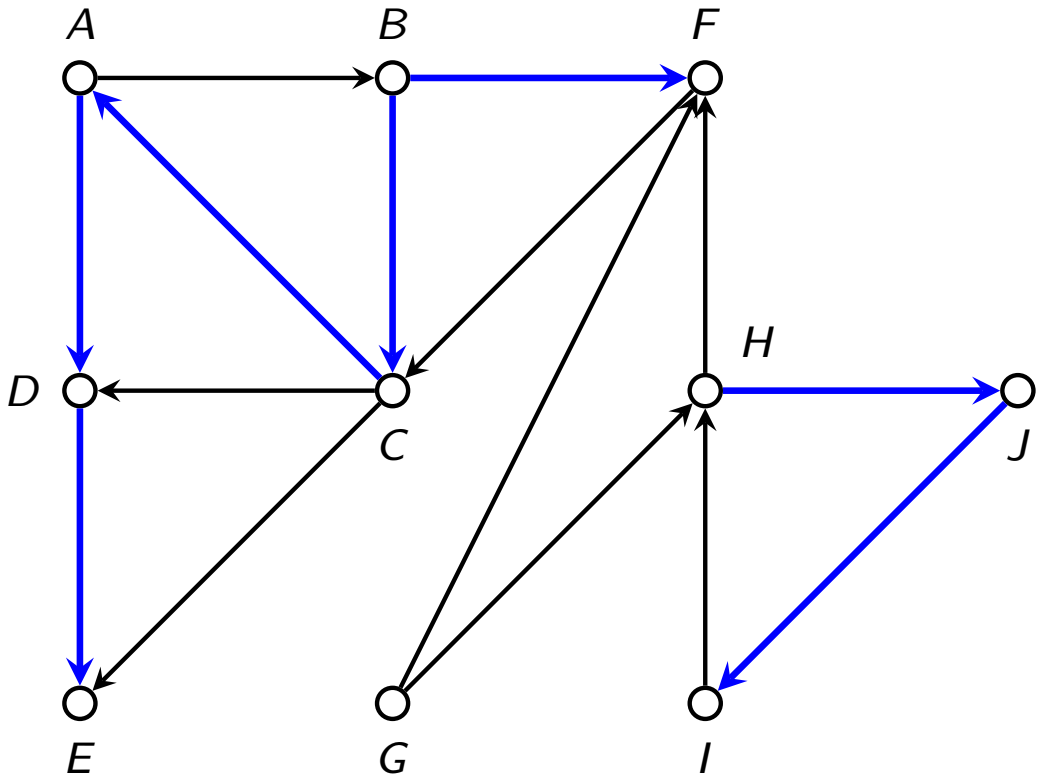
SCC

1. DFS på G
   1. gjør DFS traversering
   2. husker post-order (`finished` time stamp)



SCC

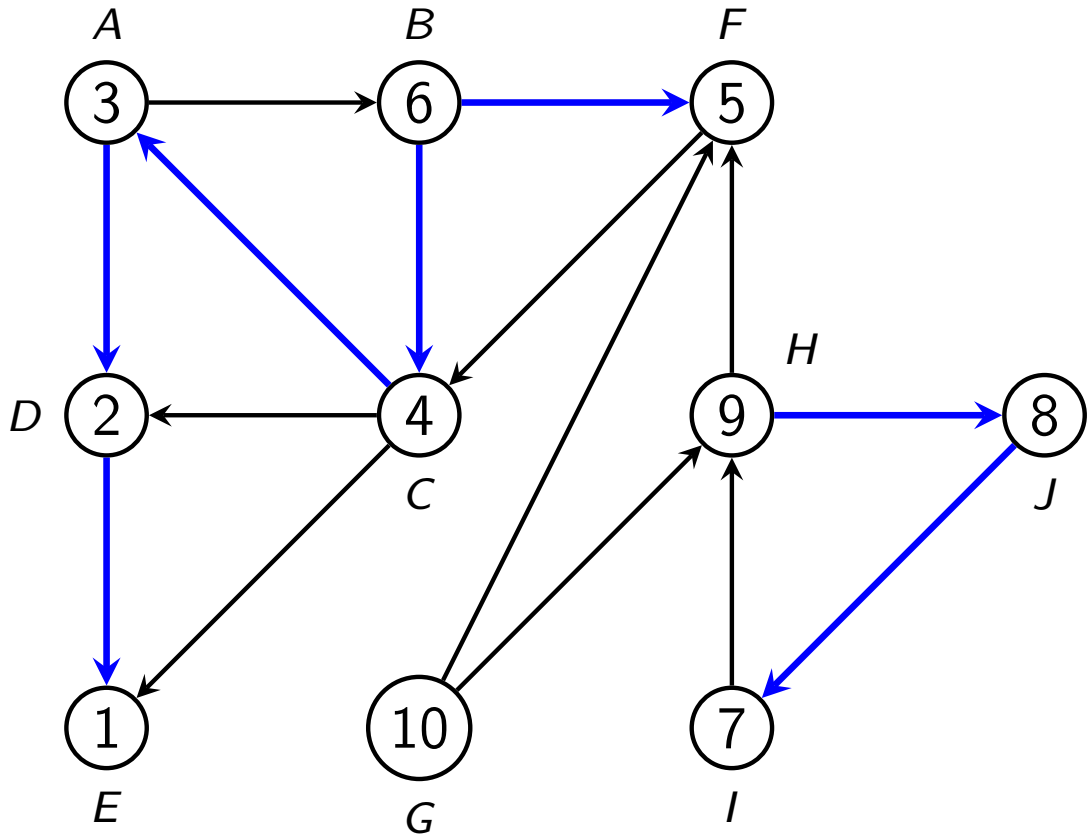
1. DFS på G
   1. gjør DFS traversering
   2. husker post-order (`finished` time stamp)



SCC

1. DFS på G

1. gjør DFS traversering
2. husker post-order (`finished` time stamp)



SCC

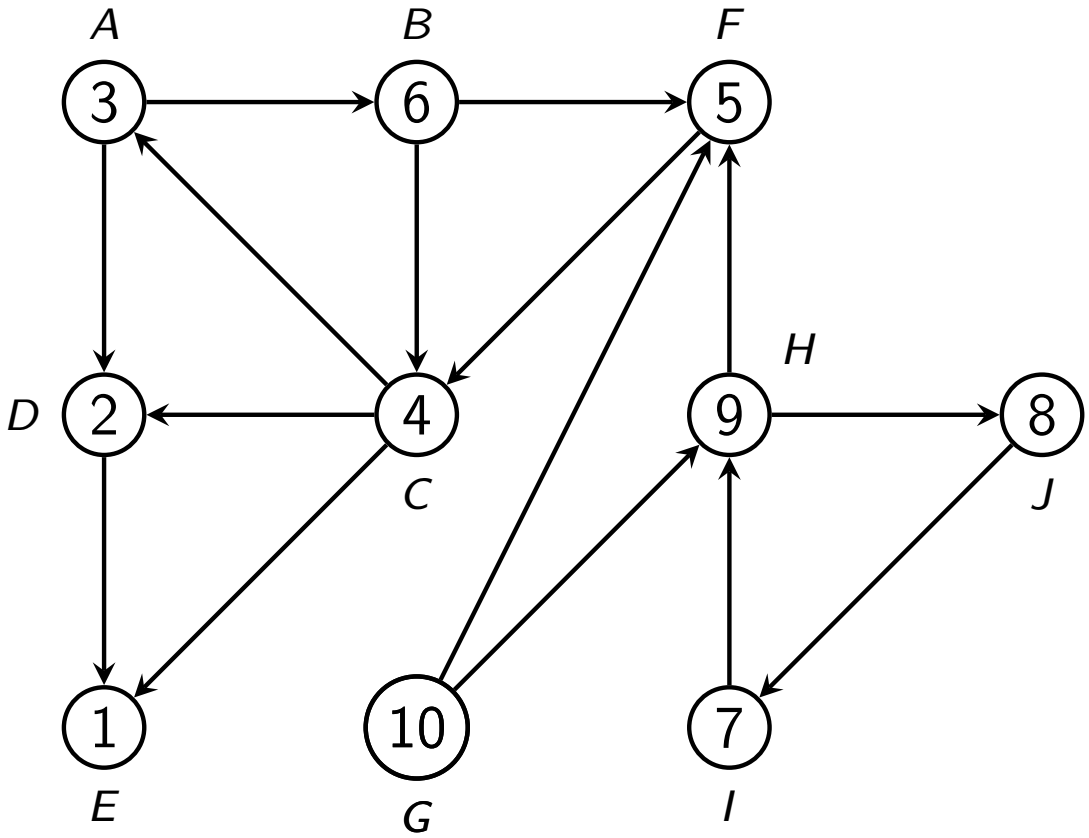
2. Reversere

reversere kantene til G og får G^t

3. DFS på G^t

iterere DFS på G^t i avtagende rekkefølger!

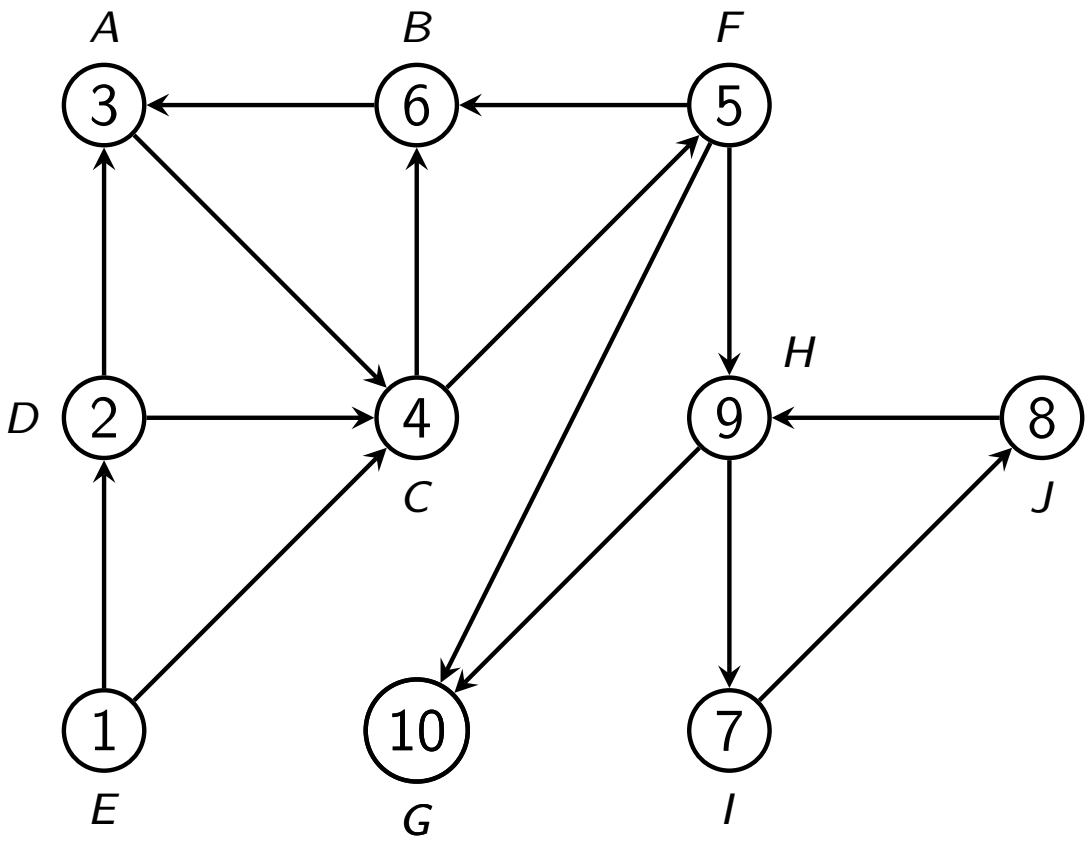
Andre fase (1)



strongly connected components:

$$\{\}$$

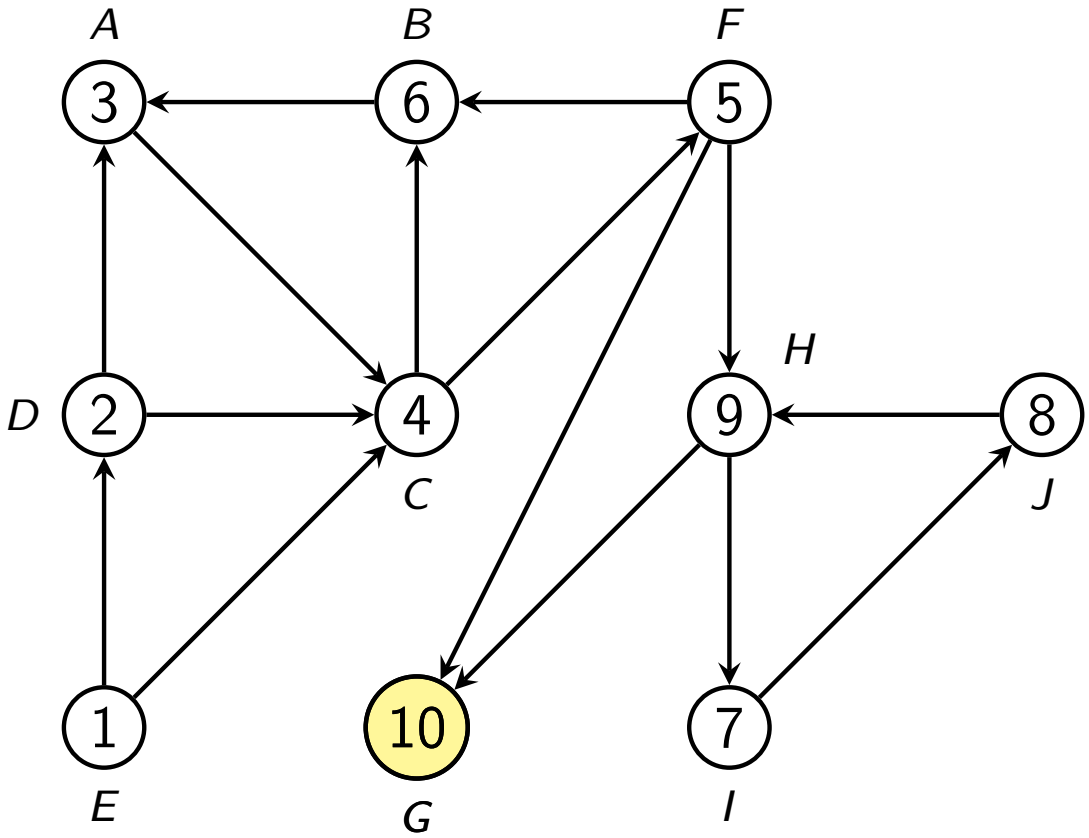
Andre fase (2)



strongly connected components:

{}

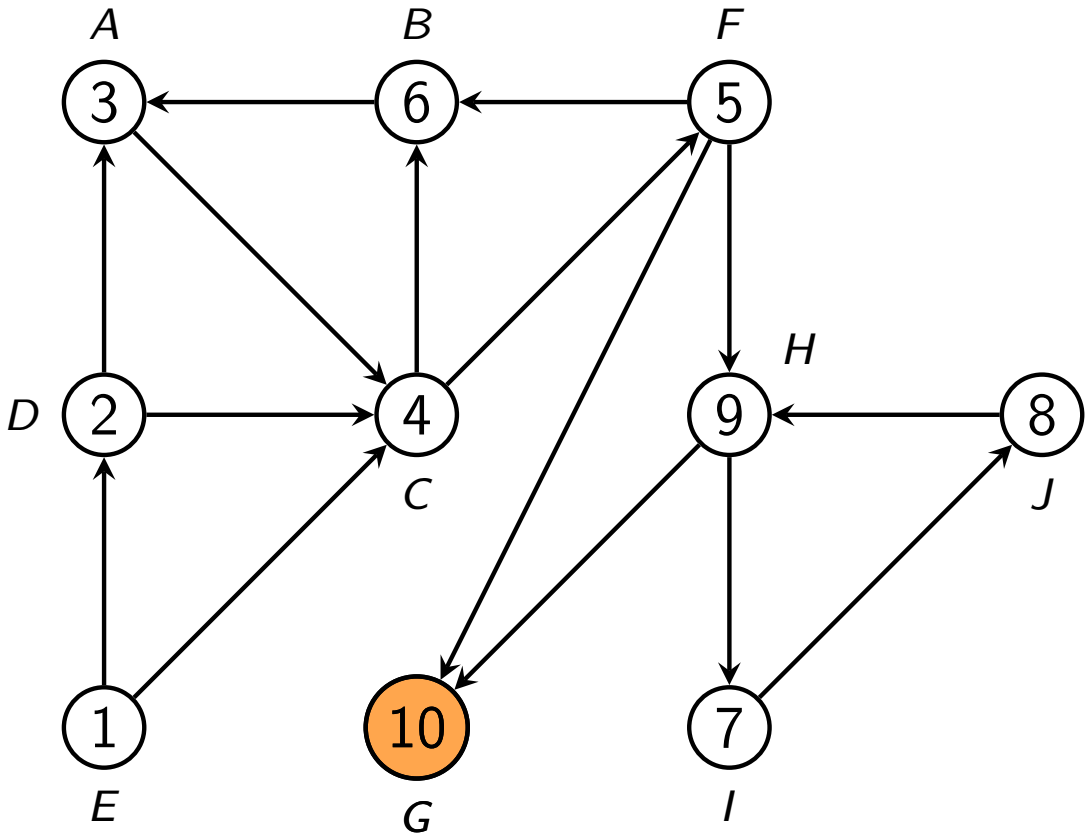
Andre fase (3)



strongly connected components:

$$\{\}$$

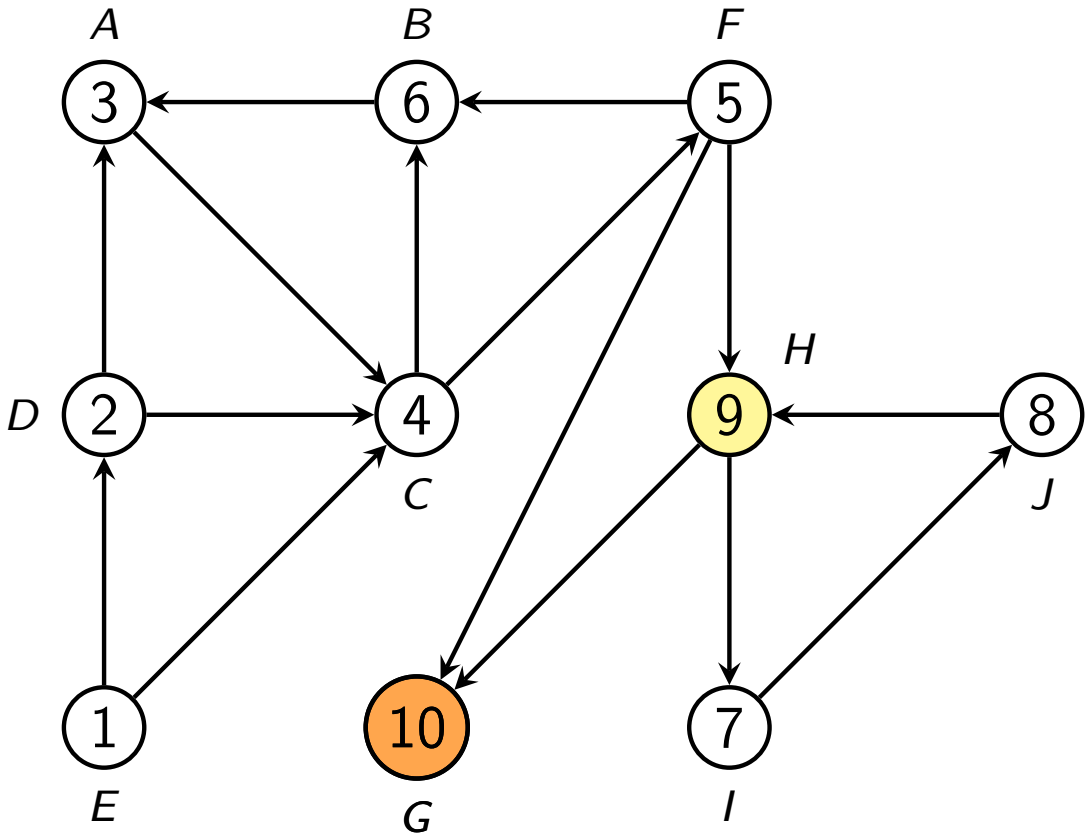
Andre fase (4)



strongly connected components:

$$\{\{G\}\}$$

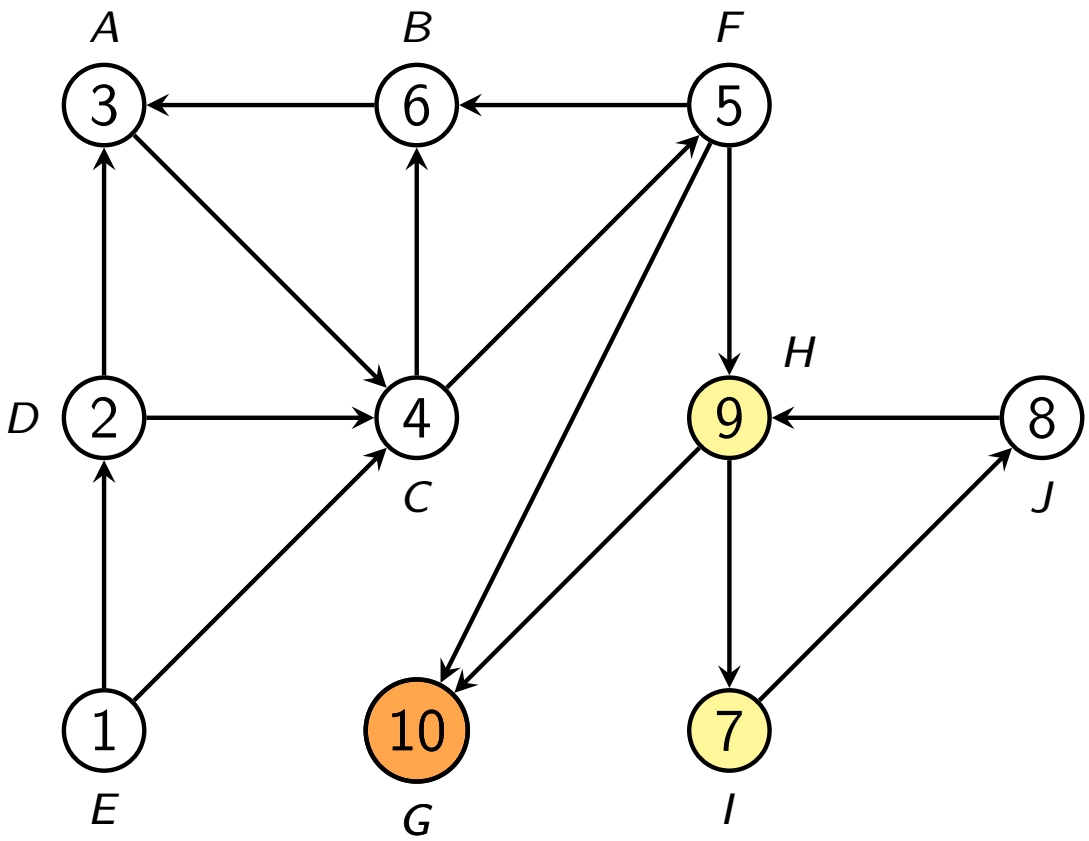
Andre fase (5)



strongly connected components:

$$\{\{G\}\}$$

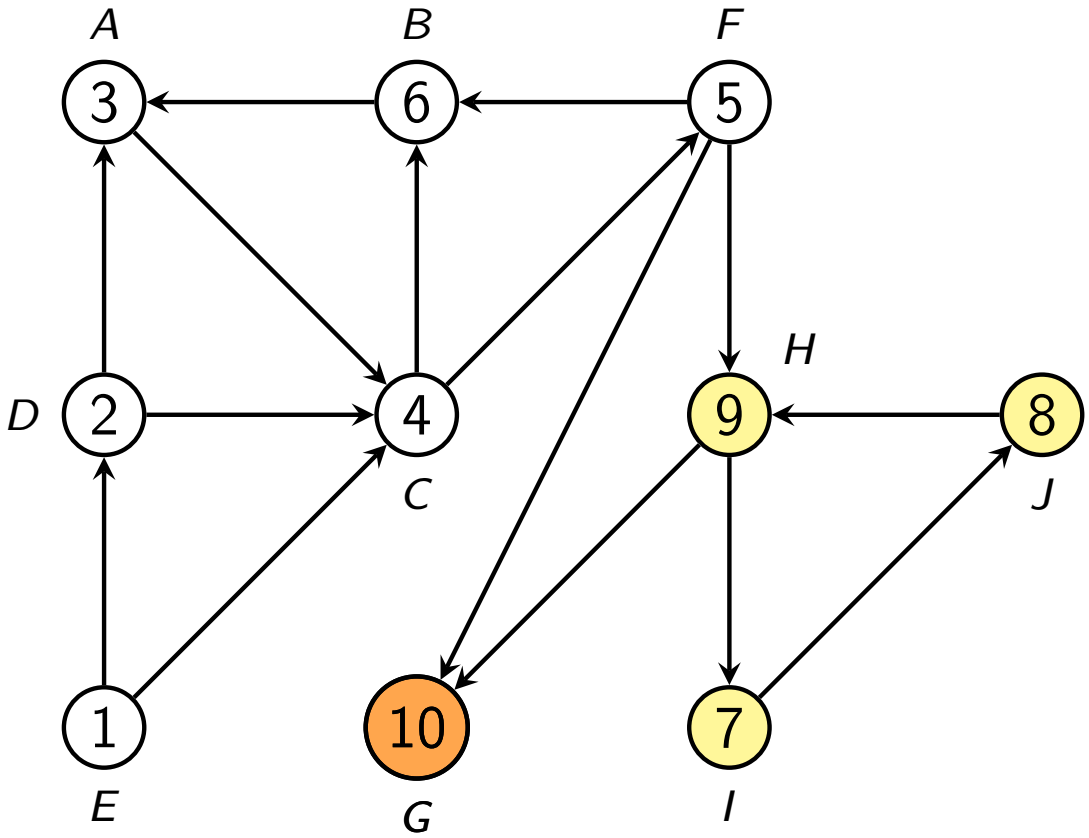
Andre fase (6)



strongly connected components:

$$\{\{G\}\}$$

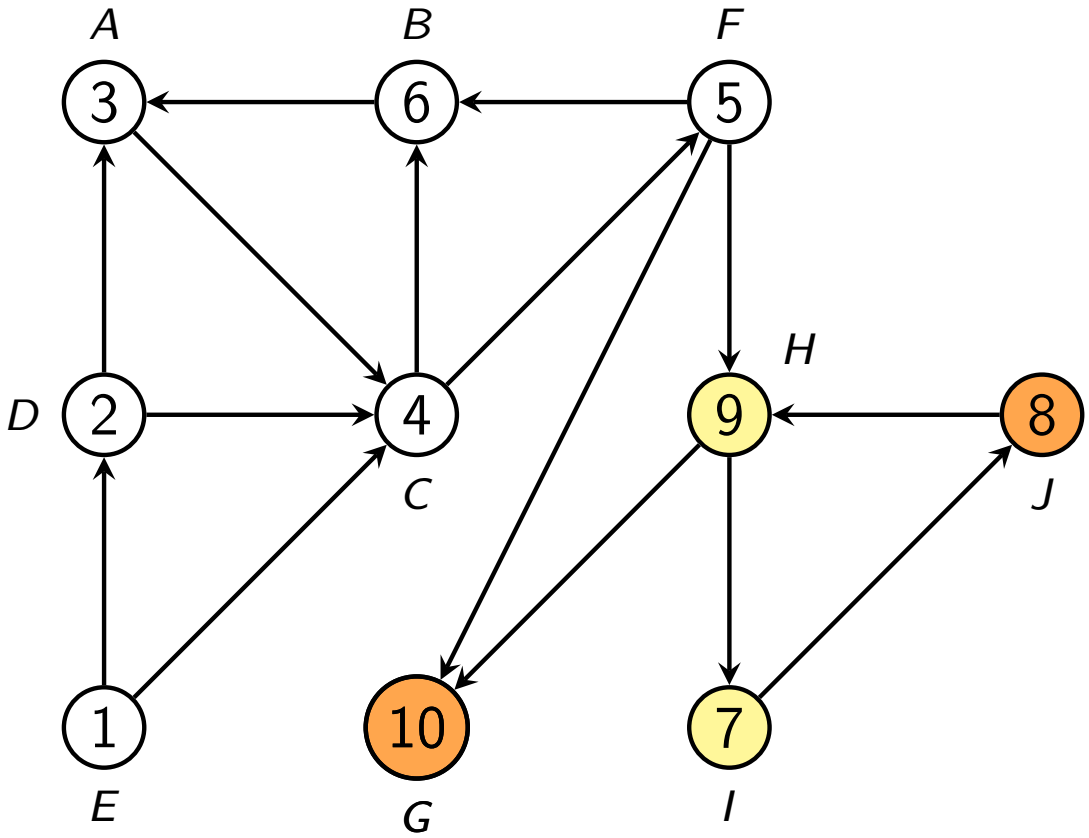
Andre fase (7)



strongly connected components:

$$\{\{G\}\}$$

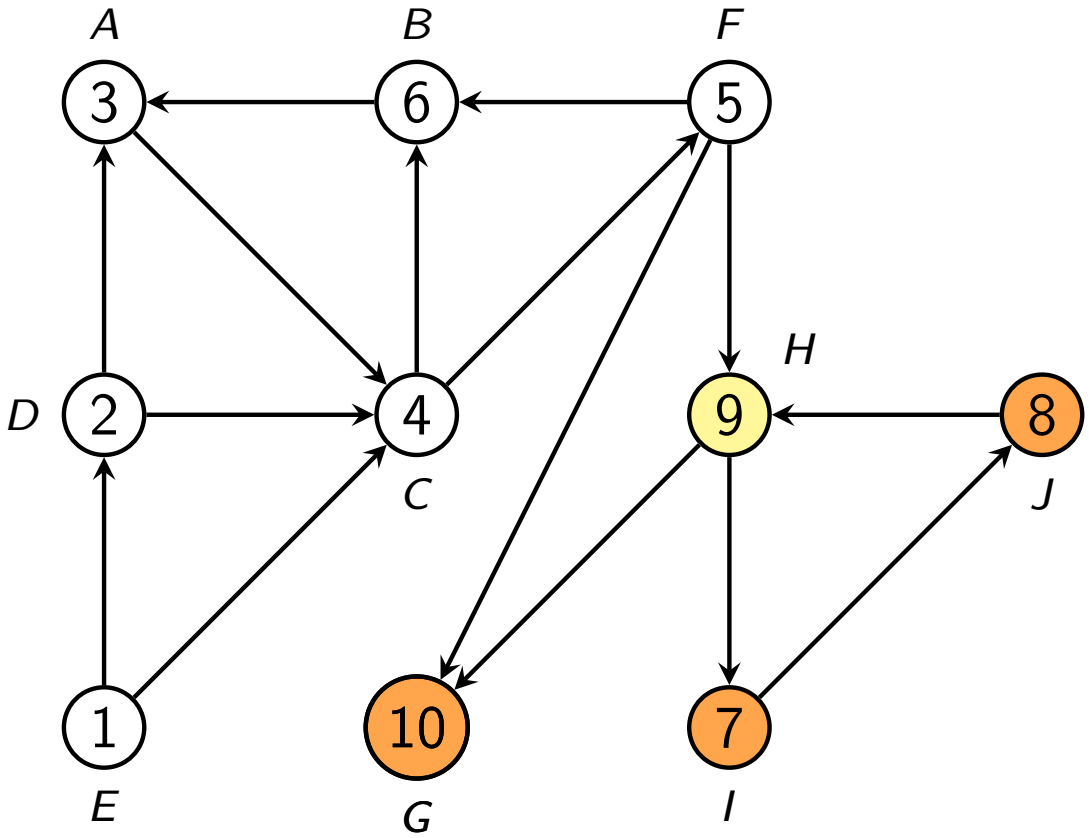
Andre fase (8)



strongly connected components:

$$\{\{G\}\}$$

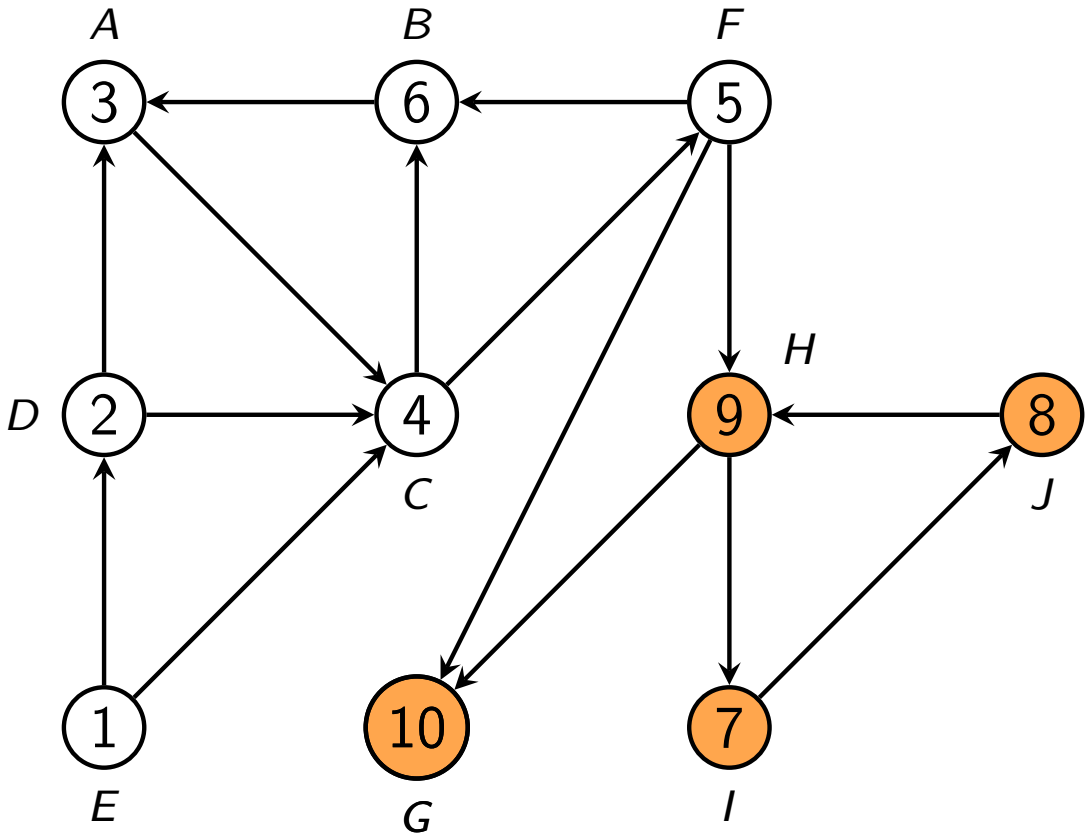
Andre fase (9)



strongly connected components:

$$\{\{G\}\}$$

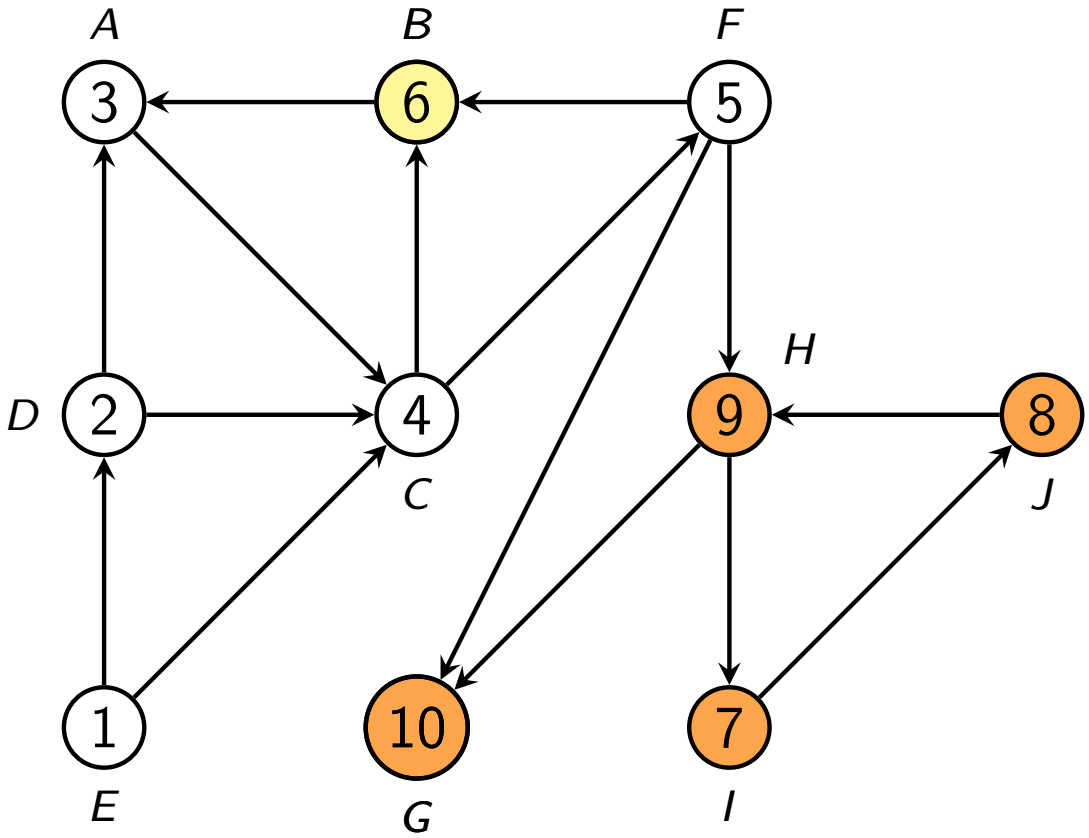
Andre fase (10)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

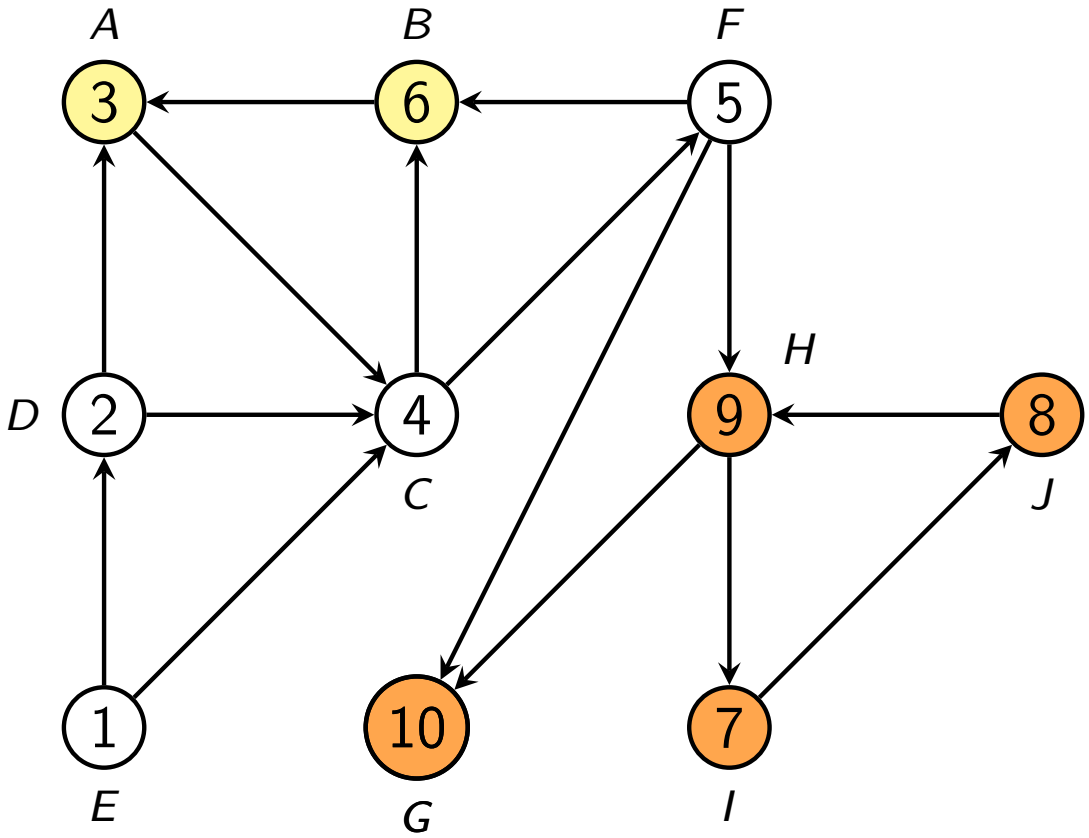
Andre fase (11)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

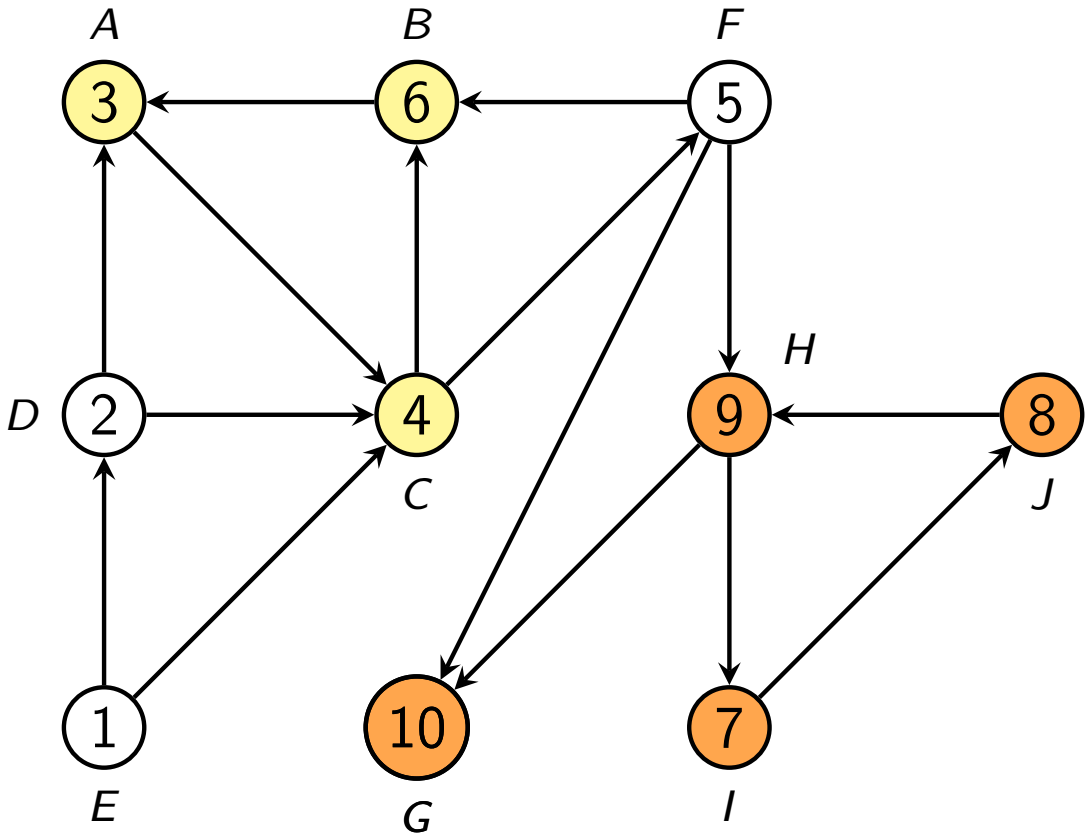
Andre fase (12)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

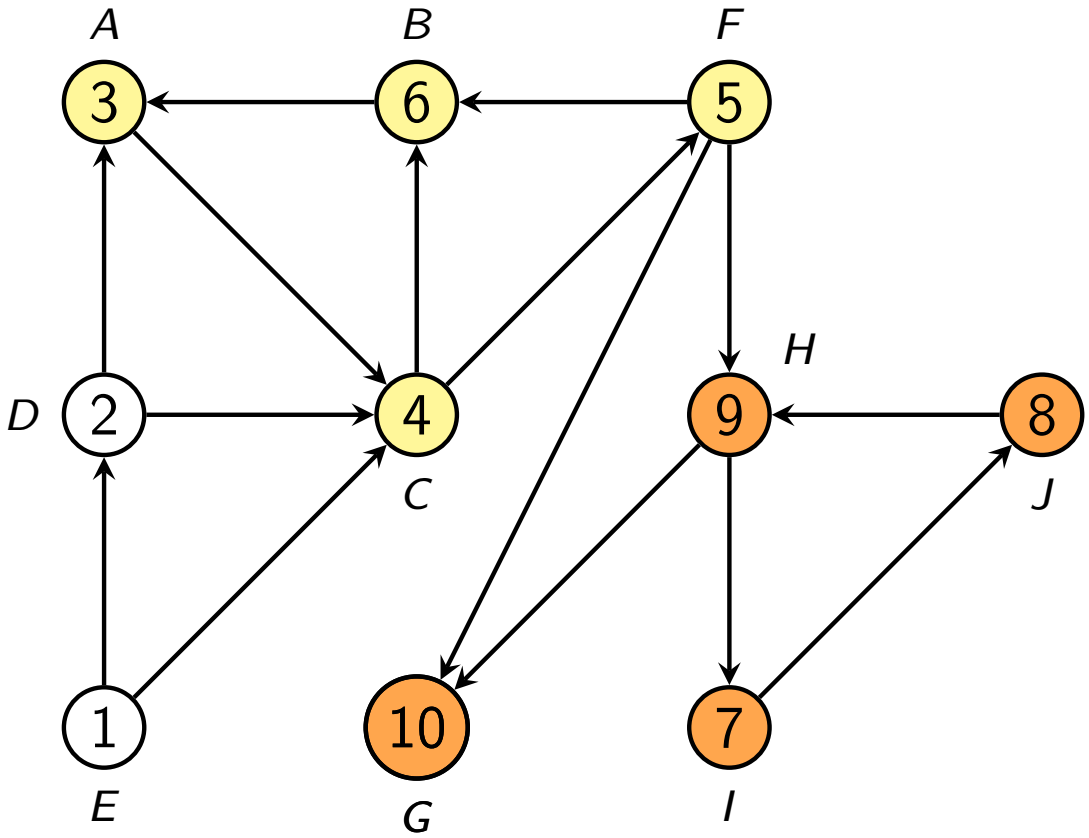
Andre fase (13)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

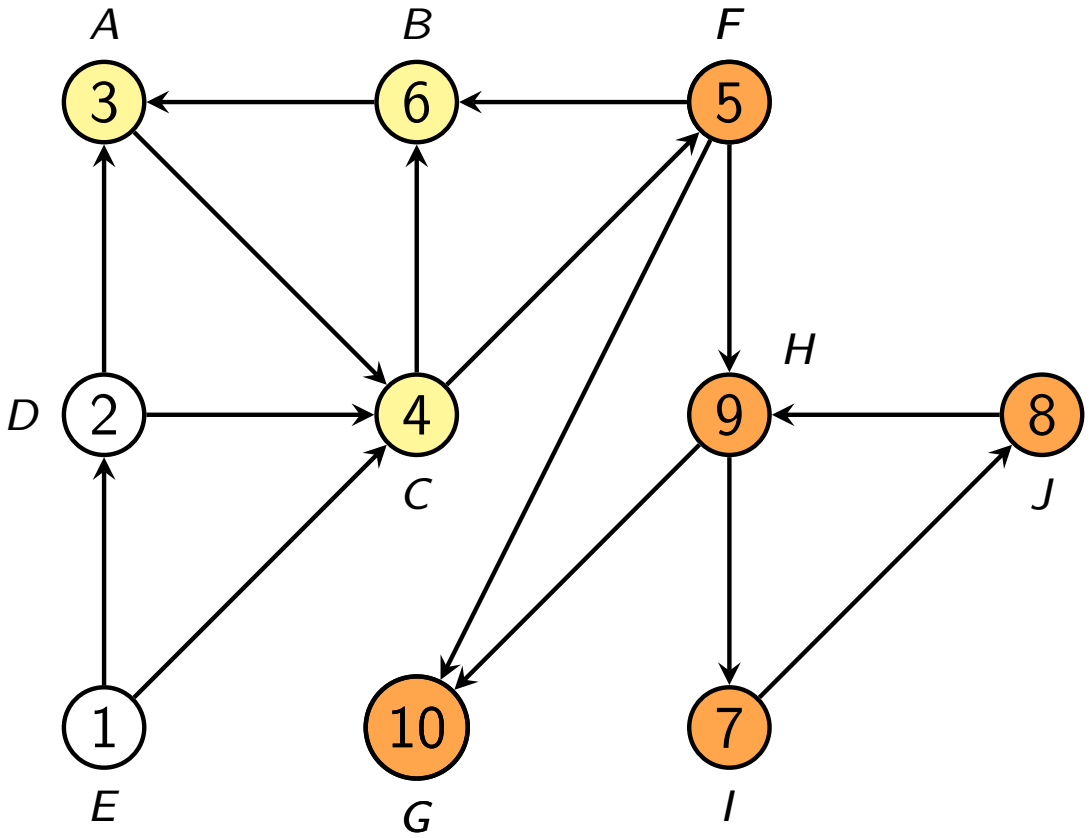
Andre fase (14)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

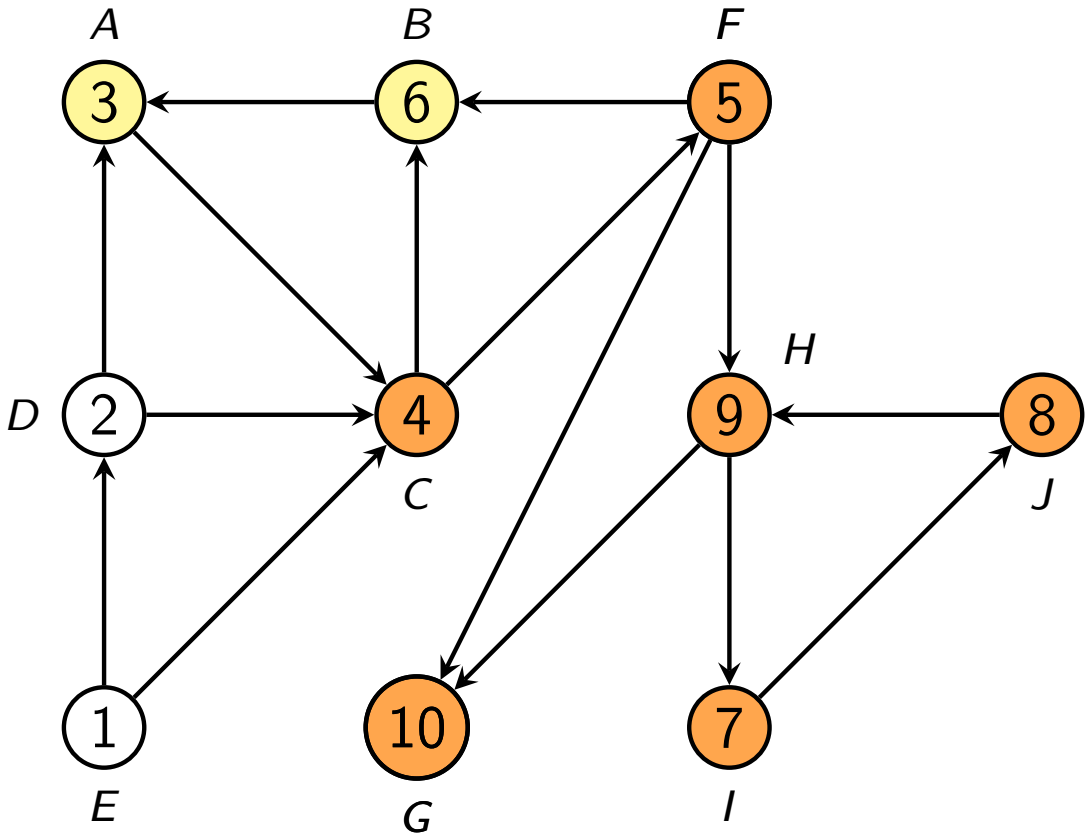
Andre fase (15)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

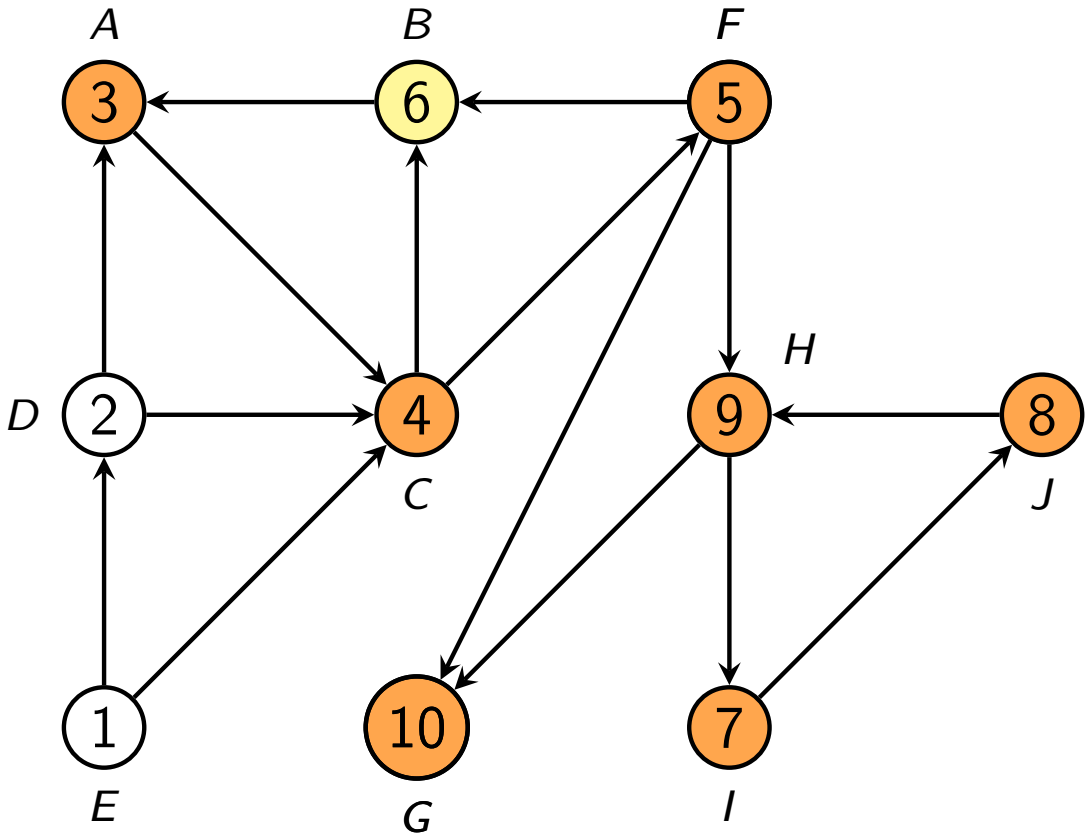
Andre fase (16)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

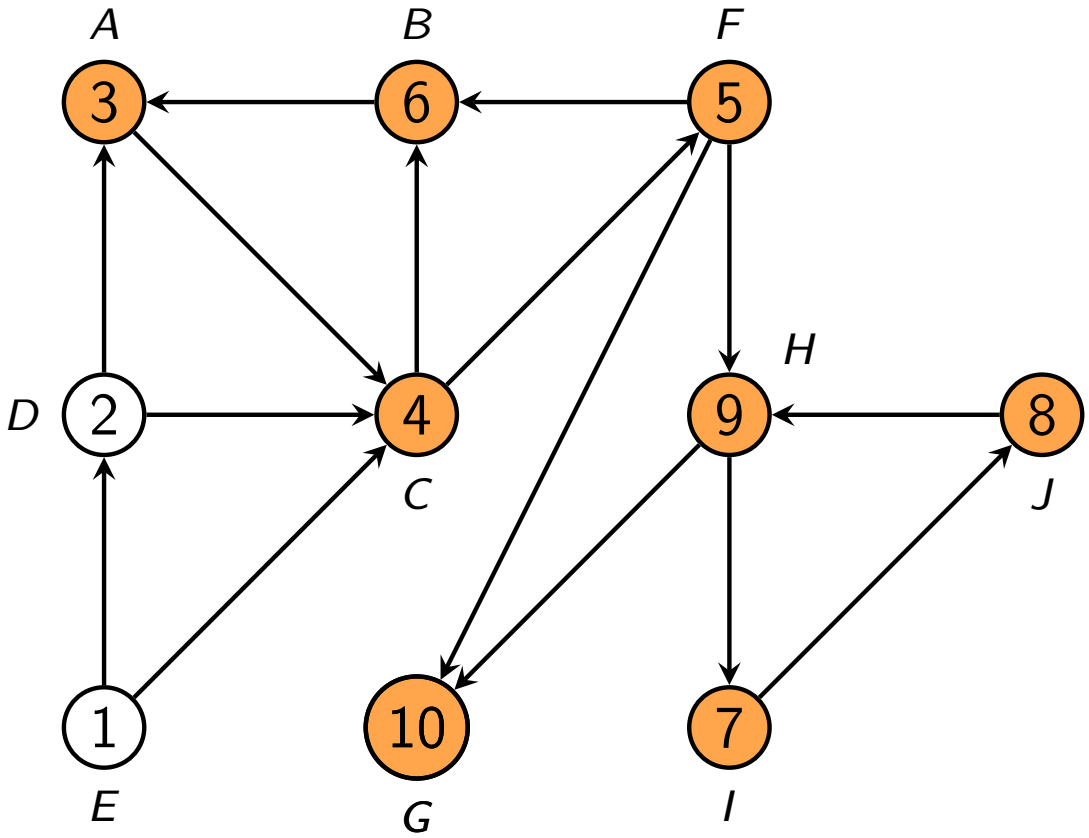
Andre fase (17)



strongly connected components:

$$\{\{G\}, \{H, J, I\}\}$$

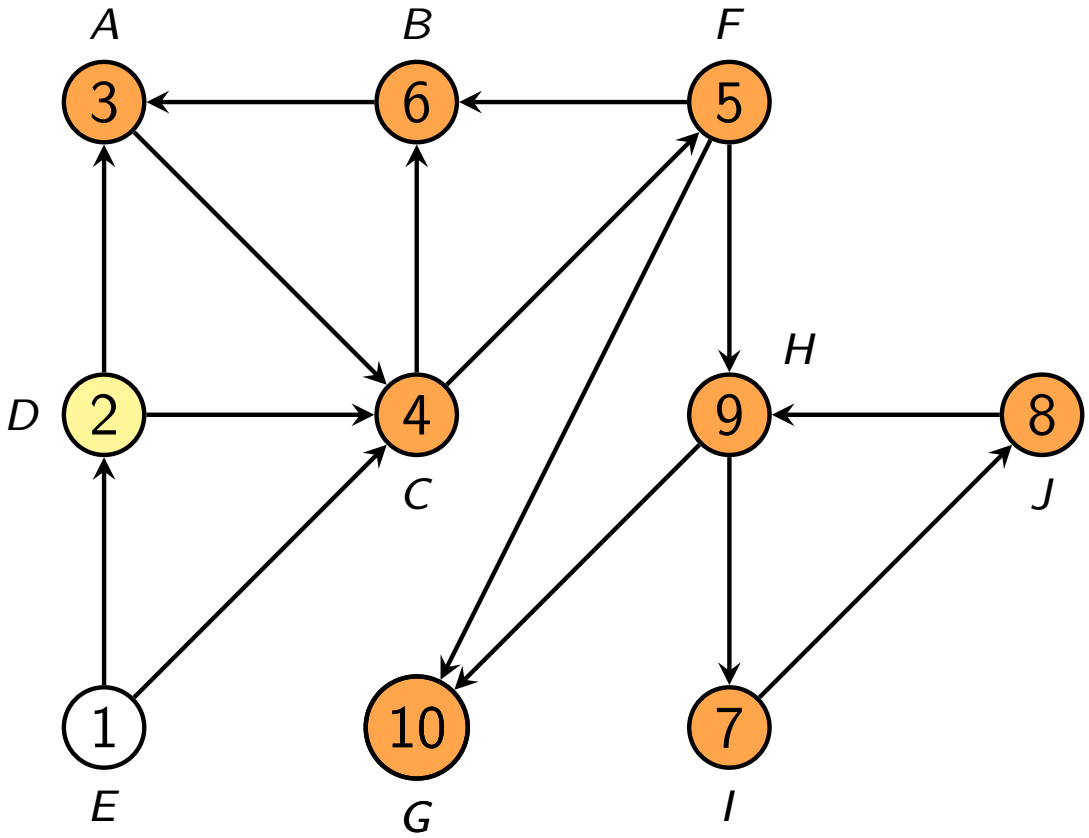
Andre fase (18)



strongly connected components:

$$\{\{G\},\{H,J,I\},\{B,A,C,F\}\}$$

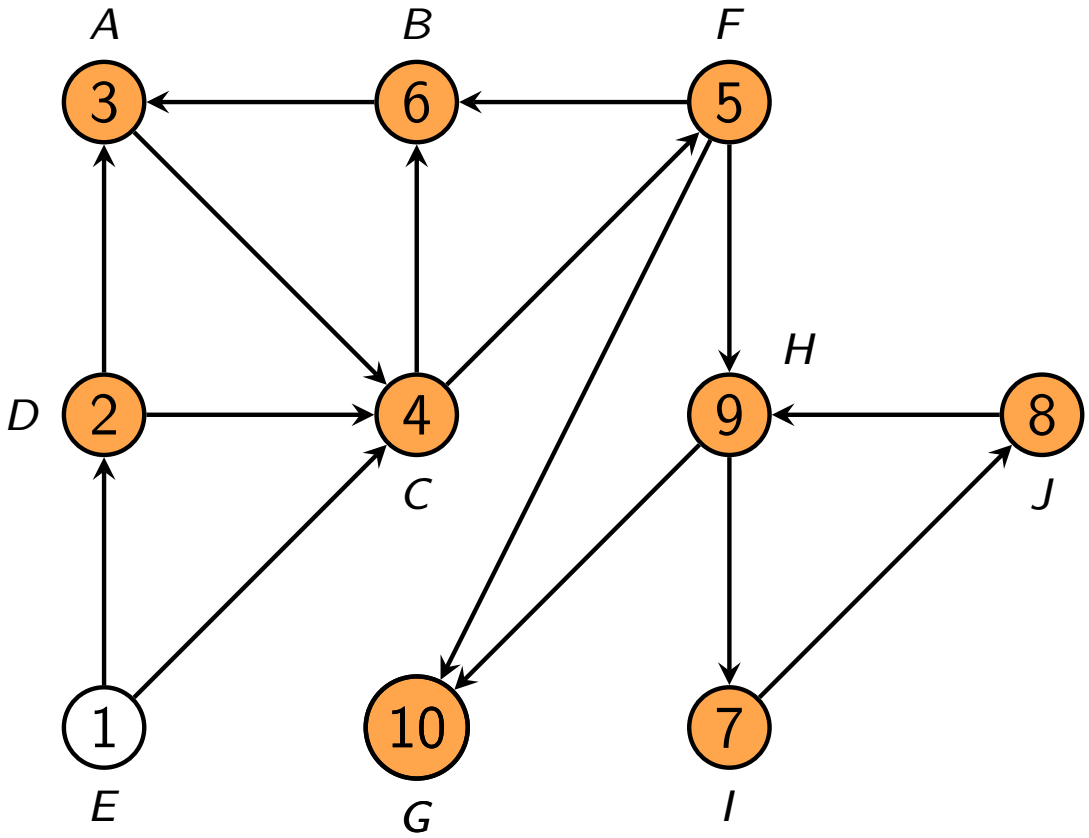
Andre fase (19)



strongly connected components:

$$\{\{G\},\{H,J,I\},\{B,A,C,F\}\}$$

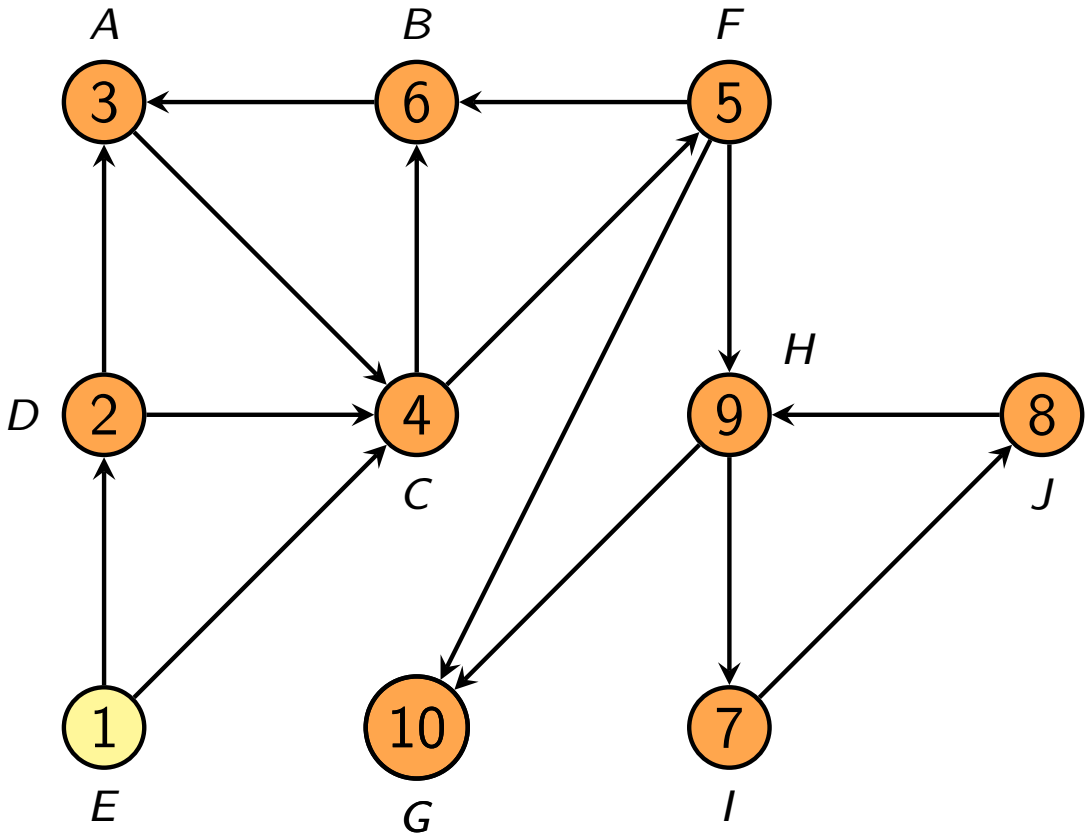
Andre fase (20)



strongly connected components:

$$\{\{G\},\{H,J,I\},\{B,A,C,F\},\{D\}\}$$

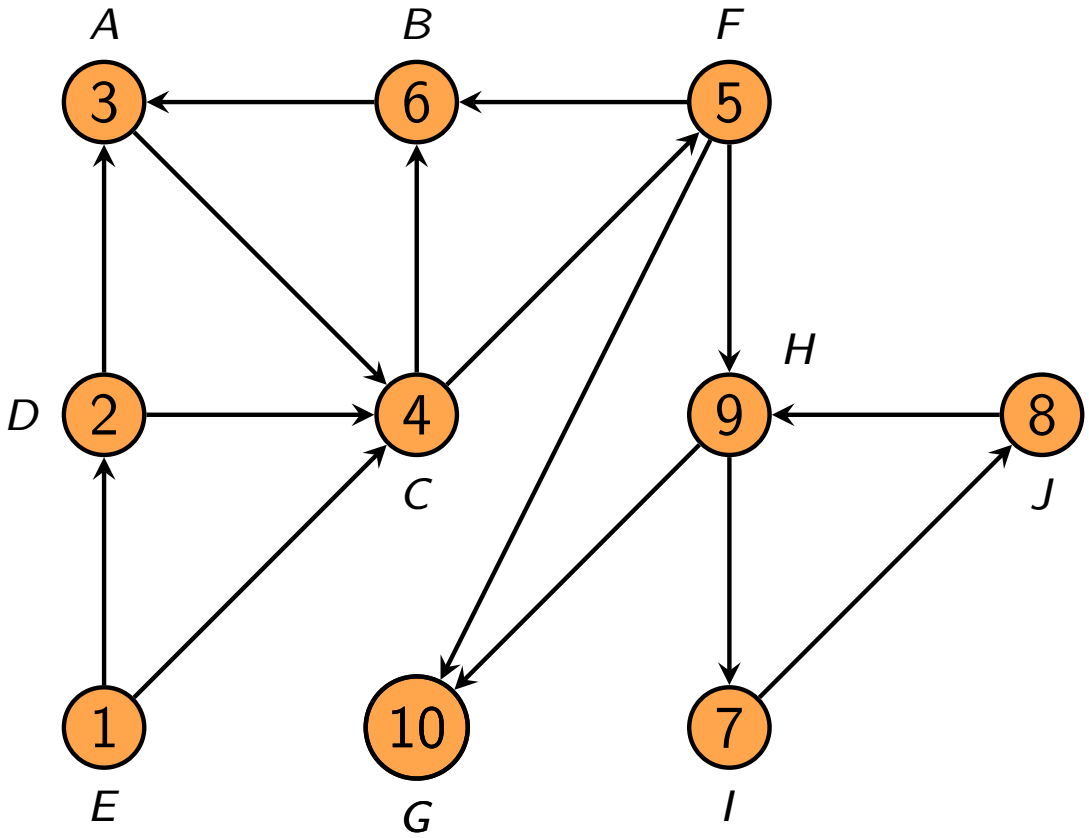
Andre fase (21)



strongly connected components:

$$\{\{G\},\{H,J,I\},\{B,A,C,F\},\{D\}\}$$

Andre fase (22)



strongly connected components:

$$\{\{G\},\{H,J,I\},\{B,A,C,F\},\{D\},\{E\}\}$$

Argument

Trenger å vise

v og w er i den samme DFS treet av G^t. Da har vi $v \longrightarrow^* w \longrightarrow^* v$

Argument

Trenger å vise

La x være roten av en dfs tre av G^t og 2 noder v og w i det samme treet av G^t. Da har vi at

$$x \longrightarrow^* v \longrightarrow^* x \quad \text{and} \quad x \longrightarrow^* w \longrightarrow^* x$$

- x er en rot
  - $\Rightarrow$ $x \longrightarrow^* v$ in G^t
  - $\Rightarrow$ $v \longrightarrow^* x$ in G
- x har høyere post-order nummer enn v
  - $\Rightarrow$ x ble ferdig senere enn v i den første dybde-først traversering av G
  - $\Rightarrow$ v må være en etterkommer av x ellers vil v bli ferdig etter x.
  - $\Rightarrow$

$$x \longrightarrow^* v \in G$$

Oppsummering

Grafer: Oppsummering

- Implementasjon av grafer
- Topologisk sortering
- Korteste vei, uvektet graf
- Dijkstras alg. (korteste vei, vektet graf)
- Prim, Kruskal (minimale spenntrær)
- Dybde-først søk
- biconnectivity, SCC

Neste forelesning: 12. oktober

Kombinatorisk søk og Kompleksitet