北海道大学

トポロジカルソート

大規模知識処理研究室　M2
竹内　文登

ほぼアニメーション

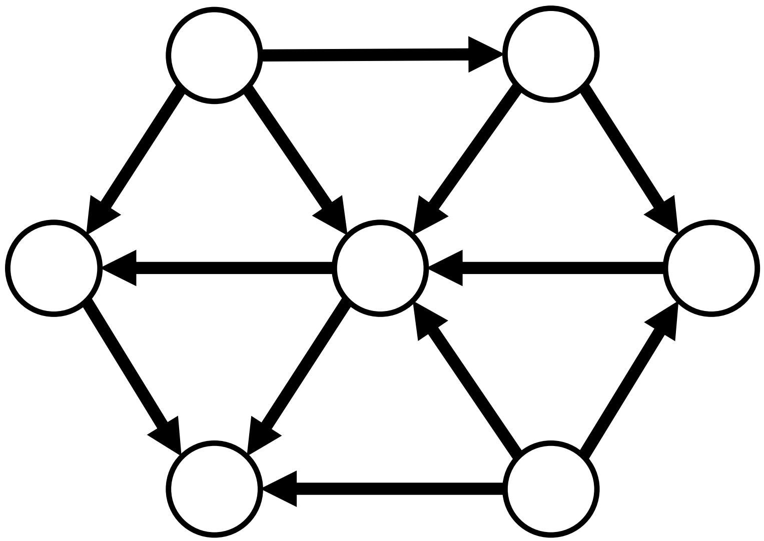
今日の内容

- 有向無閉路グラフ（DAG）
- トポロジカルソートとは？
- トポロジカルソートを求めるアルゴリズム×2

- 参考文献
  - アルゴリズムデザイン
  - アルゴリズムイントロダクション

有向無閉路グラフとは？

- 有向グラフで閉路を持たないグラフのこと
- directed acyclic graphを略してDAGという

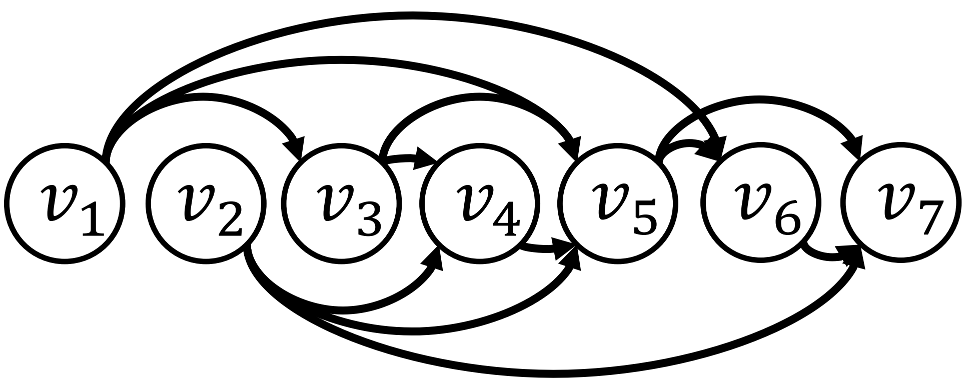


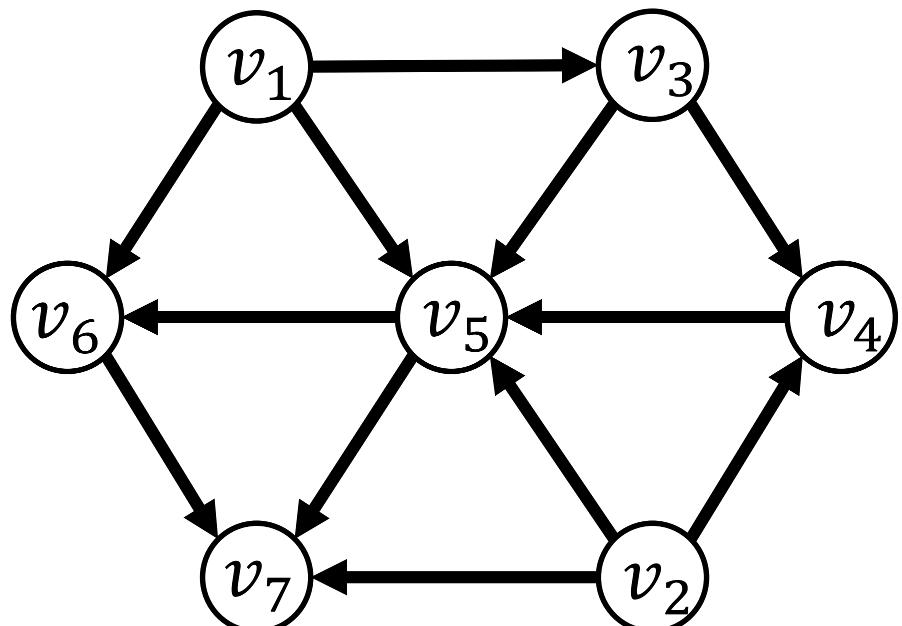
閉路はないよ。

トポロジカルソートとは？

次の性質を持つ頂点の並び $v_1, v_2, \ldots, v_n$

- すべての辺 (v_i, v_j) に対して、$i < j$ が成立
- 言い換えると、すべての辺がその順序で”順方向”に向く

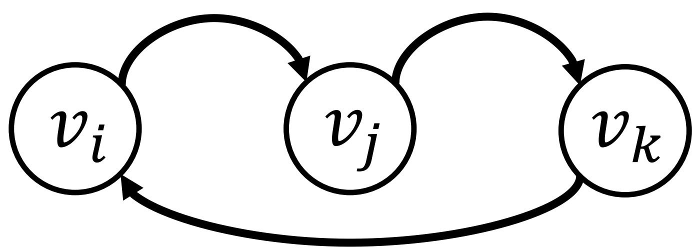




DAGとトポロジカル順序の関係

有向グラフGに対して、
GがDAGである ⇔ Gのトポロジカルソートが存在する

- 証明：GがDAG ⟸ トポロジカルソートが存在
  - トポロジカルソートがあるのに、閉路Cがあると仮定する（背理法）
  - 閉路C内をぐるぐるできる
  - 順序が大きくなり続けなければならない。おかしい。 □

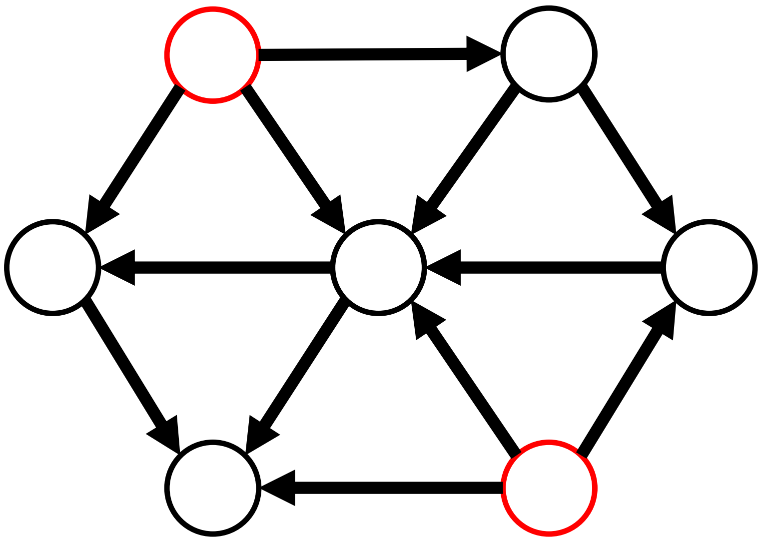


逆の証明

- GがDAG ⟹ Gのトポロジカル順序が存在する
- つまりDAGが与えられるので、トポロジカルソートを求めればよい
- 実際にトポロジカルソートを得るアルゴリズムを構築する

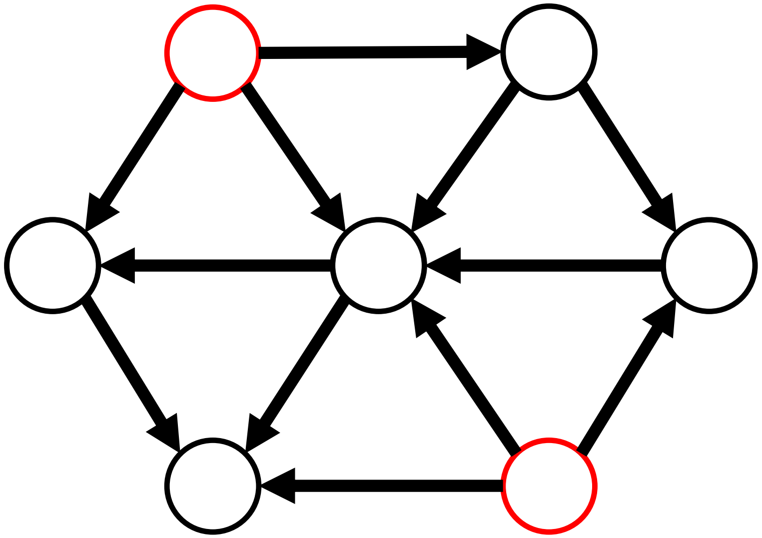
その前にDAGの重要な性質

- DAG Gには入ってくる辺のない頂点 v が存在する



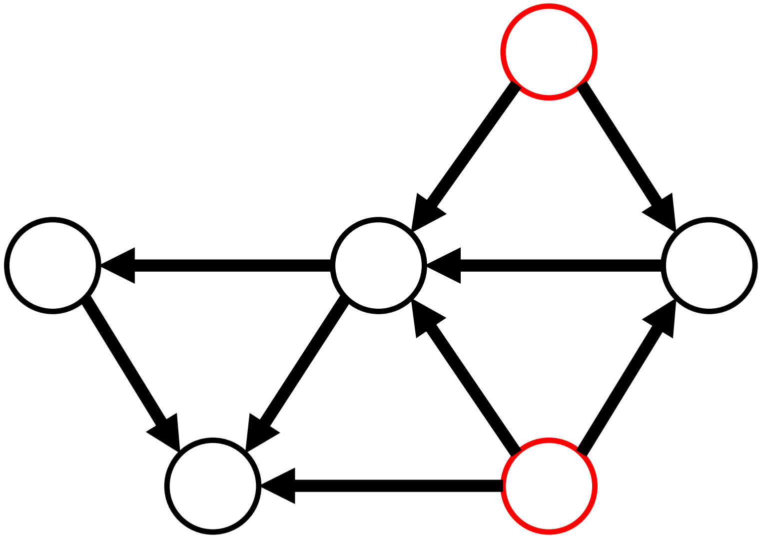
その前にDAGの重要な性質

- DAG Gには入ってくる辺のない頂点 v が存在する
- それらの頂点はトポロジカルソートの最初の頂点になれる



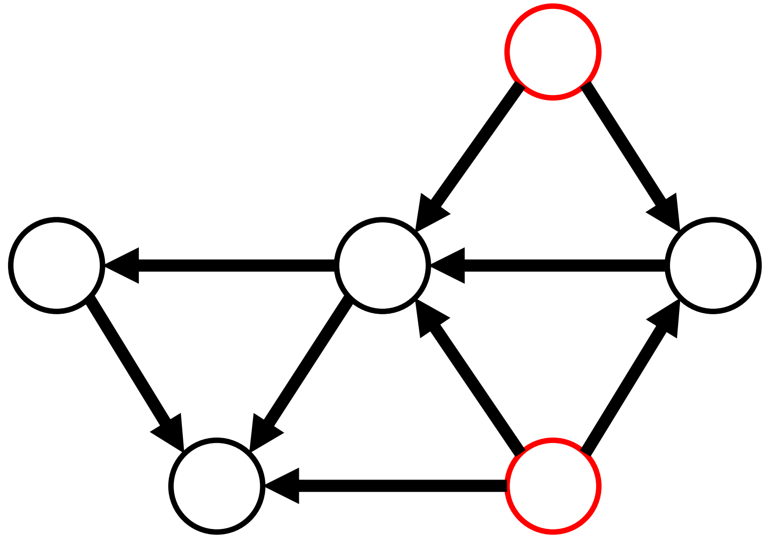
その前にDAGの重要な性質

- DAG Gには入ってくる辺のない頂点 v が存在する
- それらの頂点はトポロジカルソートの最初の頂点になれる
- その頂点を除いたグラフもDAGである



その前にDAGの重要な性質

- DAG Gには入ってくる辺のない頂点 v が存在する
- それらの頂点はトポロジカルソートの最初の頂点になれる
- その頂点を除いたグラフもDAGである
- 以下、繰り返し



トポロジカルソートを求めるアルゴリズム

- 最初の頂点を見つけ、削除して、を繰り返す。

- Gのトポロジカルソートの計算:
  - まず入ってくる辺のない頂点 v を求める
  - Gから v を削除する
  - 再帰的に G – {v} のトポロジカル順序を求め、vの後に繋ぐ

トポロジカルソートを求めるアルゴリズム

- 最初の頂点を見つけ、削除して、を繰り返す。

- Gのトポロジカルソートの計算:
  - まず入ってくる辺のない頂点 v を求める
  - Gから v を削除する
  - 再帰的に G – {v} のトポロジカル順序を求め、vの後に繋ぐ

- 「入ってくる辺のない頂点 v」を効率良く求めると、
- 全体で$\boldsymbol{O(V+E)}$**時間**でトポロジカルソートを求めることができる

トポロジカル順序を求めるアルゴリズム

- アルゴリズム中で、以下を保持する
  - 各頂点の入次数
  - 入次数==0の頂点集合 S

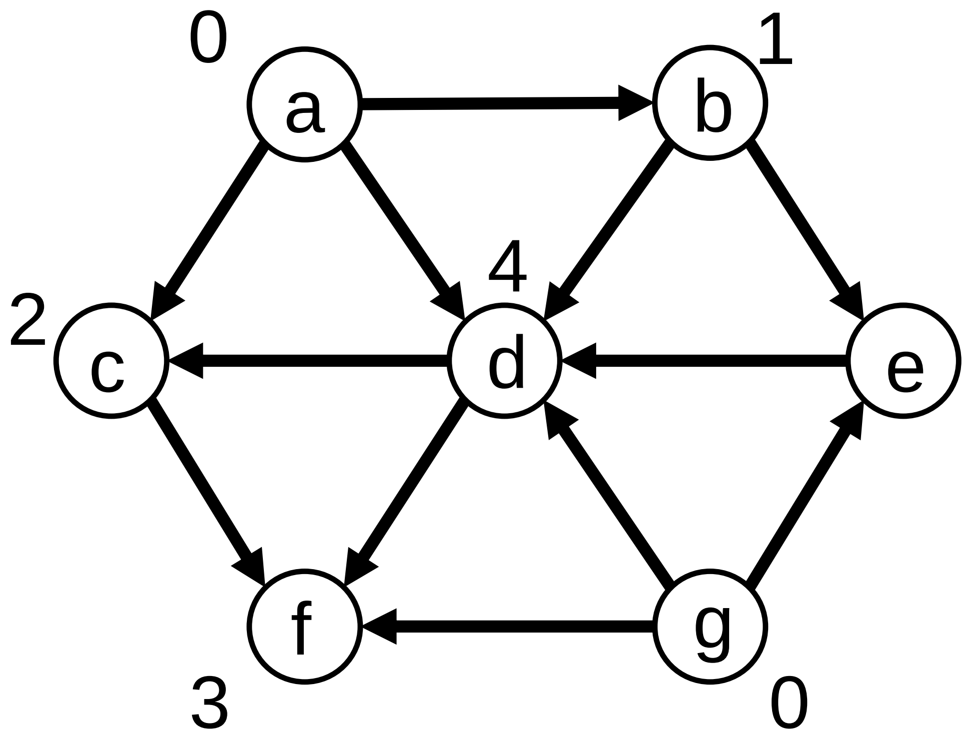
- 具体的なアルゴリズム
  1. すべての頂点の入ってくる辺の個数を数え、その値が0となる頂点の集合をSとする: $O(E)$
  2. Sが空になるまで以下を繰り返す: $O(V)$
     1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
     2. 0ならばSに追加する

アルゴリズムの動作例

1. すべての頂点の入ってくる辺の個数を数え、その値が0となる頂点の集合をSとする

S = {a,g}

ans = {}

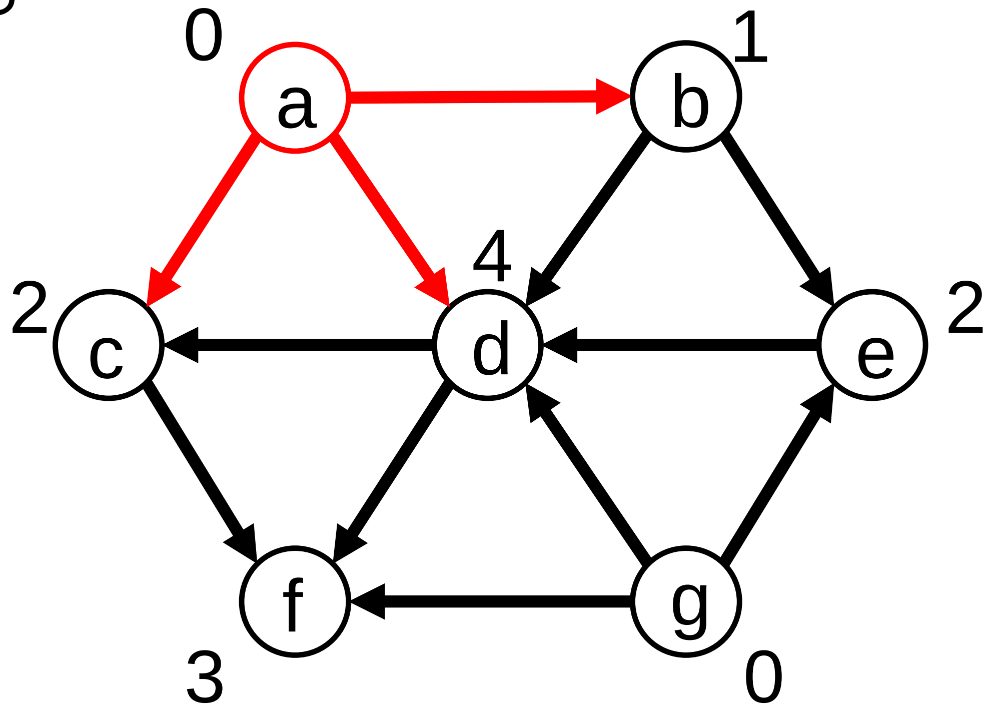


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {a,g}

ans = {}

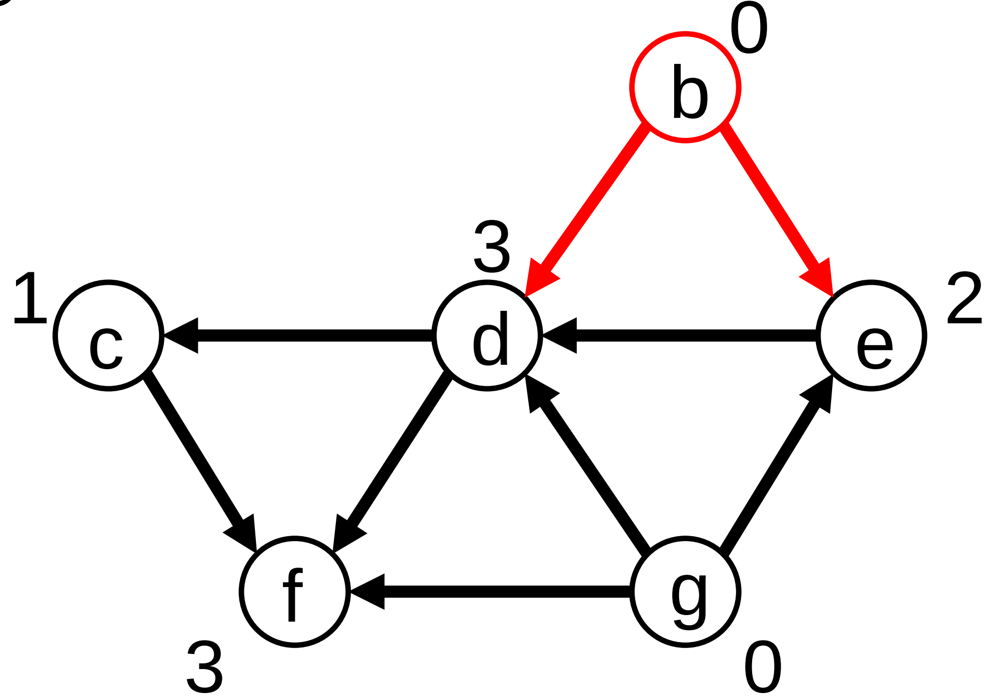


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {b,g}

ans = {a}

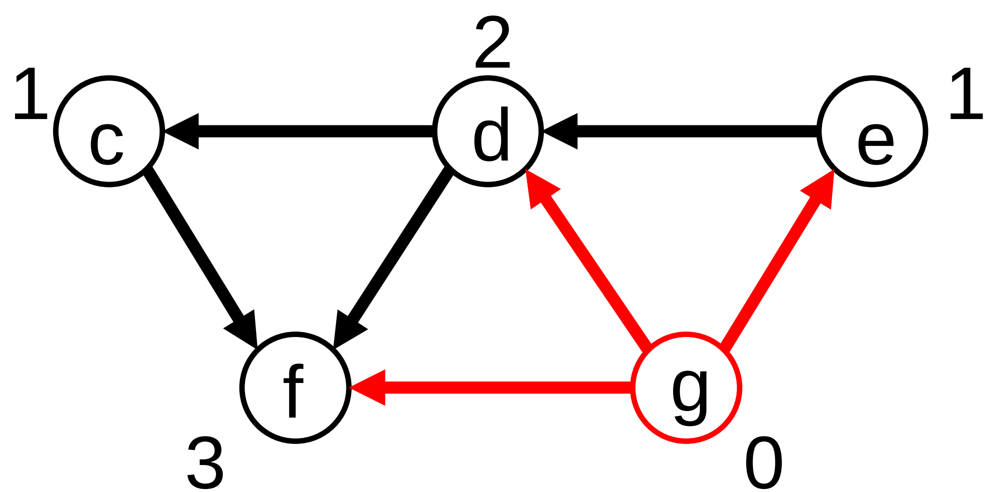


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {g}

ans = {a,b}

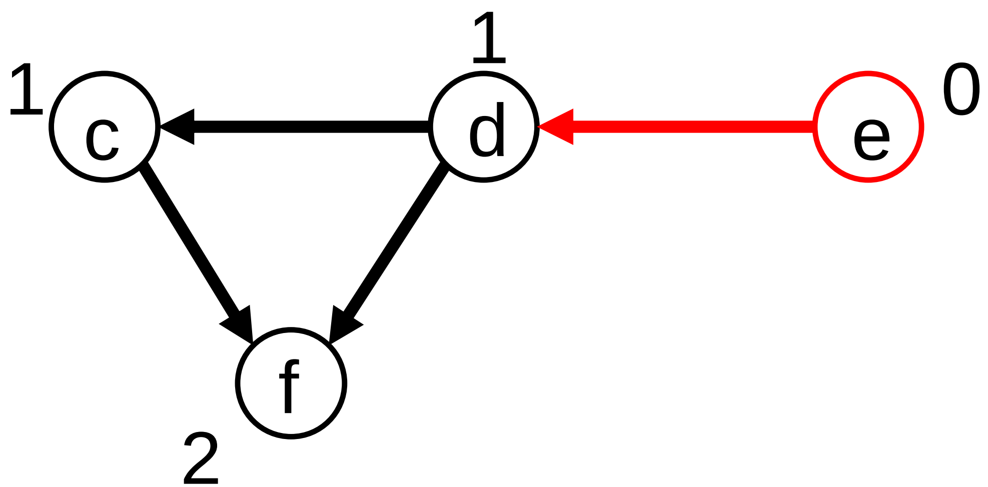


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {e}

ans = {a,b,g}

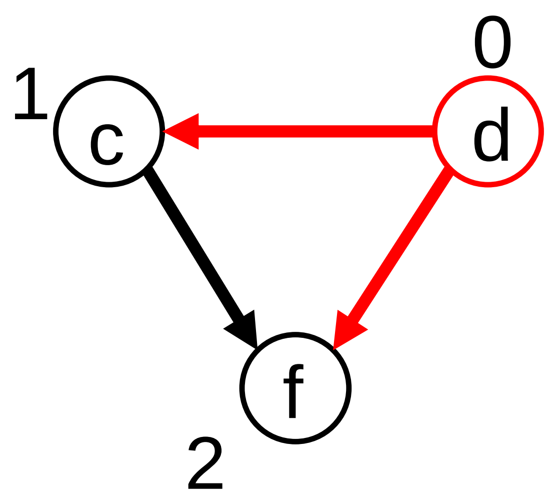


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {d}

ans = {a,b,g,e}

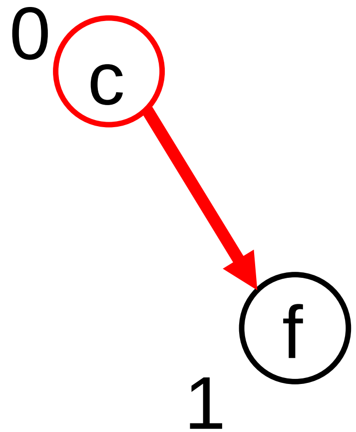


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
    1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
    2. 0ならばSに追加する

S = {c}

ans = {a,b,g,e,d}

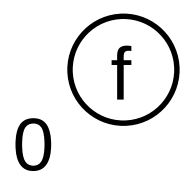


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {f}

ans = {a,b,g,e,d,c}

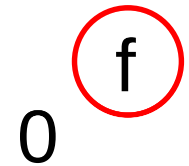


アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {f}

ans = {a,b,g,e,d,c}



アルゴリズムの動作例

2. Sが空になるまで以下を繰り返す:$O(V)$
   1. Sから頂点 v を取り出すたびに、vから出るすべての辺を見て、行き先の頂点の入ってくる辺の個数を1減らす
   2. 0ならばSに追加する

S = {}

ans = {a,b,g,e,d,c,f}

```cpp
vector<int> tsort_Kahn(const vector<vector<int>>& g) { // g:隣接リスト
    const int V = g.size();
    vector<int> indeg(V,0);          // indeg[i] : 頂点iの入次数
    stack<int>  S;                   // indeg[i] == 0 となるiの集合

    for(auto& u_out_edges : g)       // 各頂点の入次数を計算 O(E)
        for(auto& v : u_out_edges)
            indeg[v]++;

    for(int i=0; i<V; ++i)           // 入次数 == 0 の頂点を計算 O(V)
        if( indeg[i] == 0 )
            S.push(i);

    vector<int> ans;
    while( S.size() > 0 ) {          // 入次数0の頂点がなくなるまで...
        int u = S.top(); S.pop(); // 入次数0の頂点 u をひとつ決める
        ans.emplace_back(u);         // u を結果列に挿入
        for(auto& v : g[u]) {        // u から出る辺の行き先 v の入次数を1減らす
            indeg[v]--;
            if( indeg[v] == 0 ) S.push(v); // 入次数が0となる頂点をSに追加
        }
    }
    return ans;
}
```

別のアルゴリズム [Tarjan 1976]

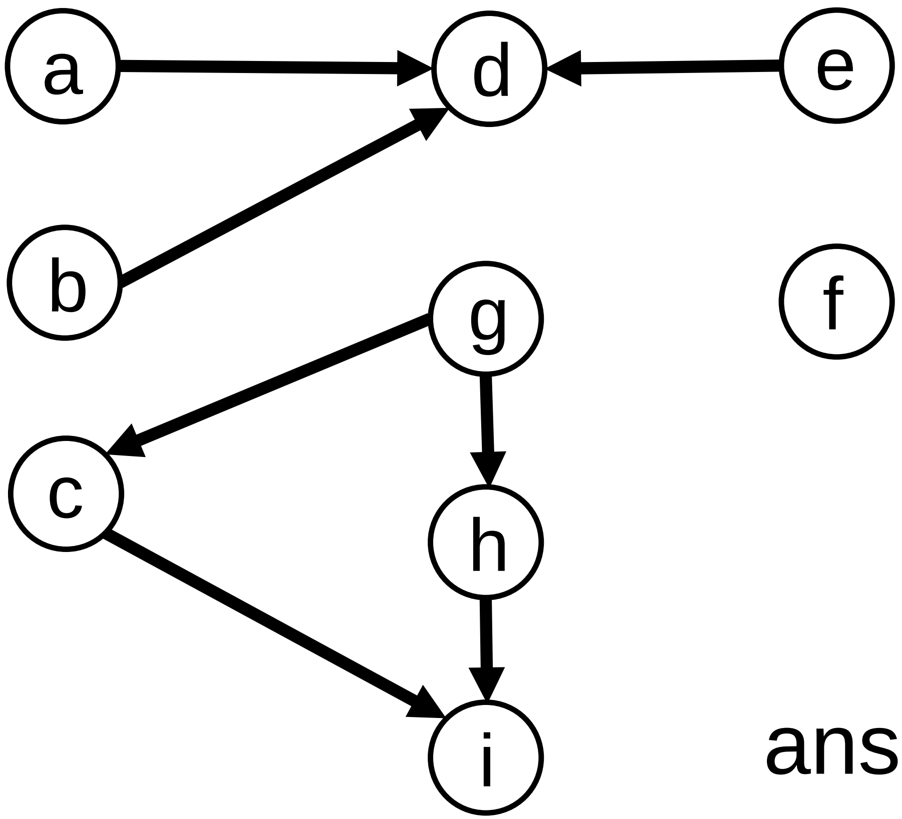
- 各頂点 v からDFSする

- Topological Sort(G) :
  - For all (u in G)　← すべての頂点からDFS
    - Visit (u)

- Visit (u) :
  - If (u が探索済みでないなら)
    - For all (u から出る辺の行き先 v)
      - Visit (v)
    - u を結果列の先頭に挿入　← 帰りがけに結果列に追加

[Tarjan 1976] の動作例

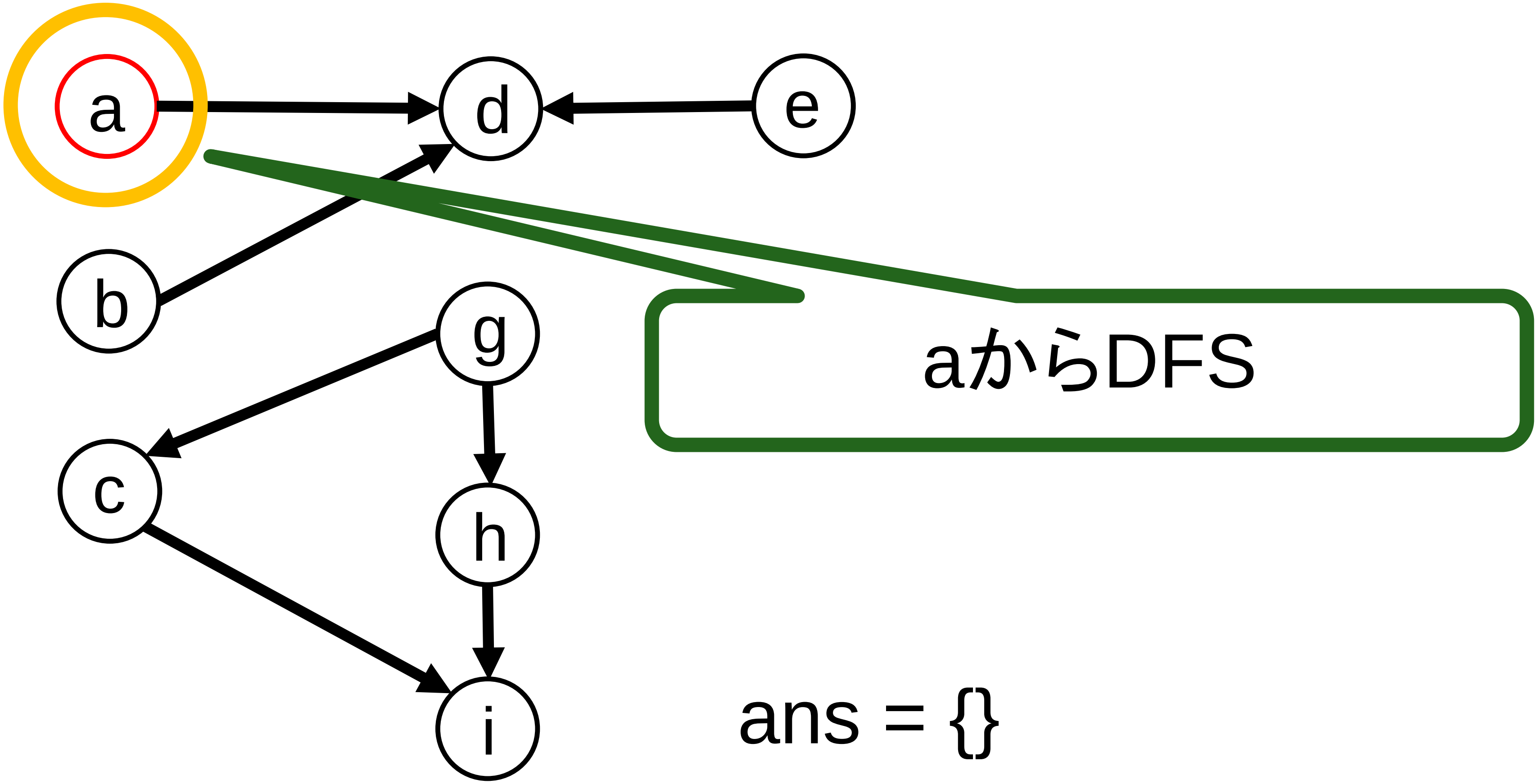
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {}

[Tarjan 1976] の動作例

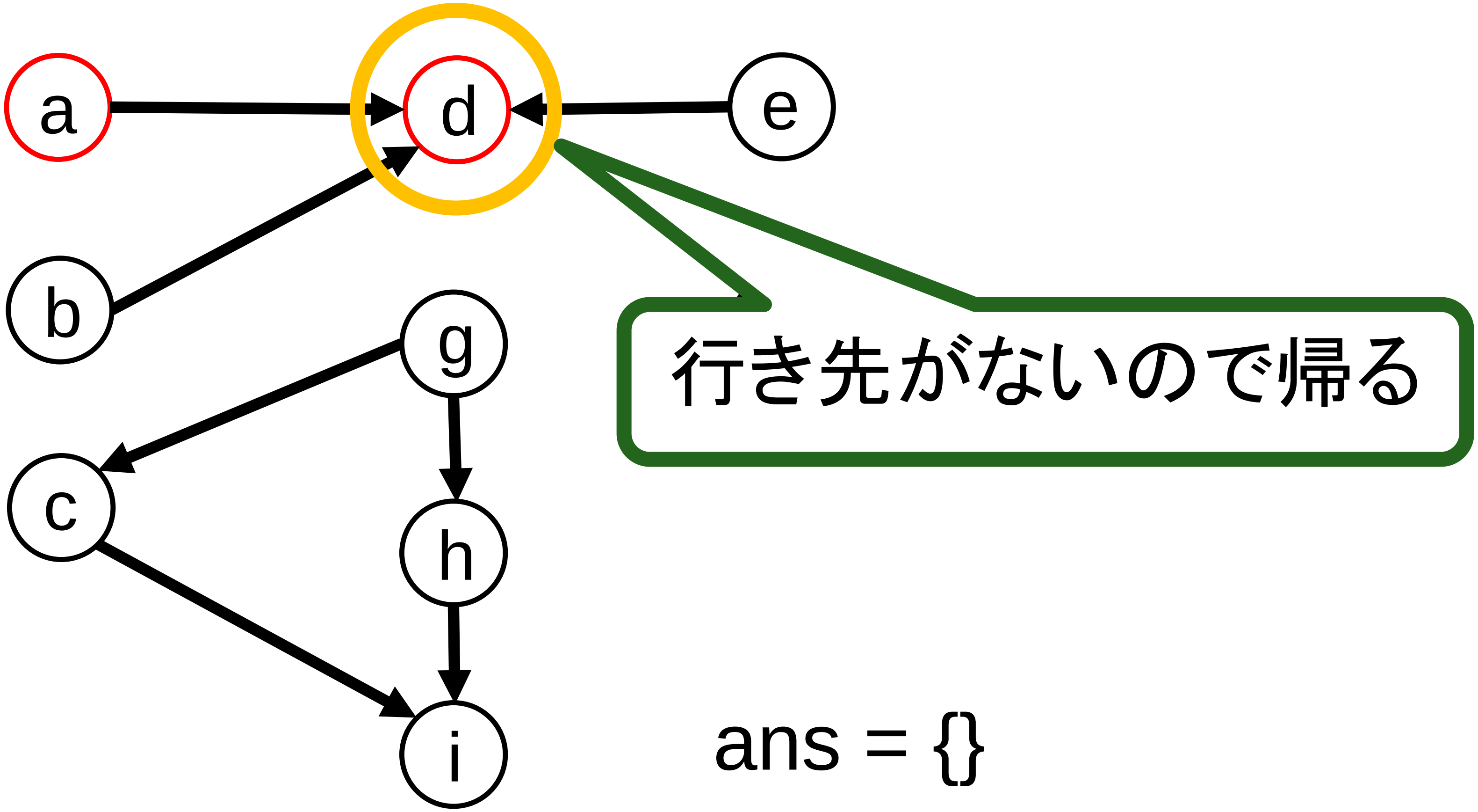
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {}

[Tarjan 1976] の動作例

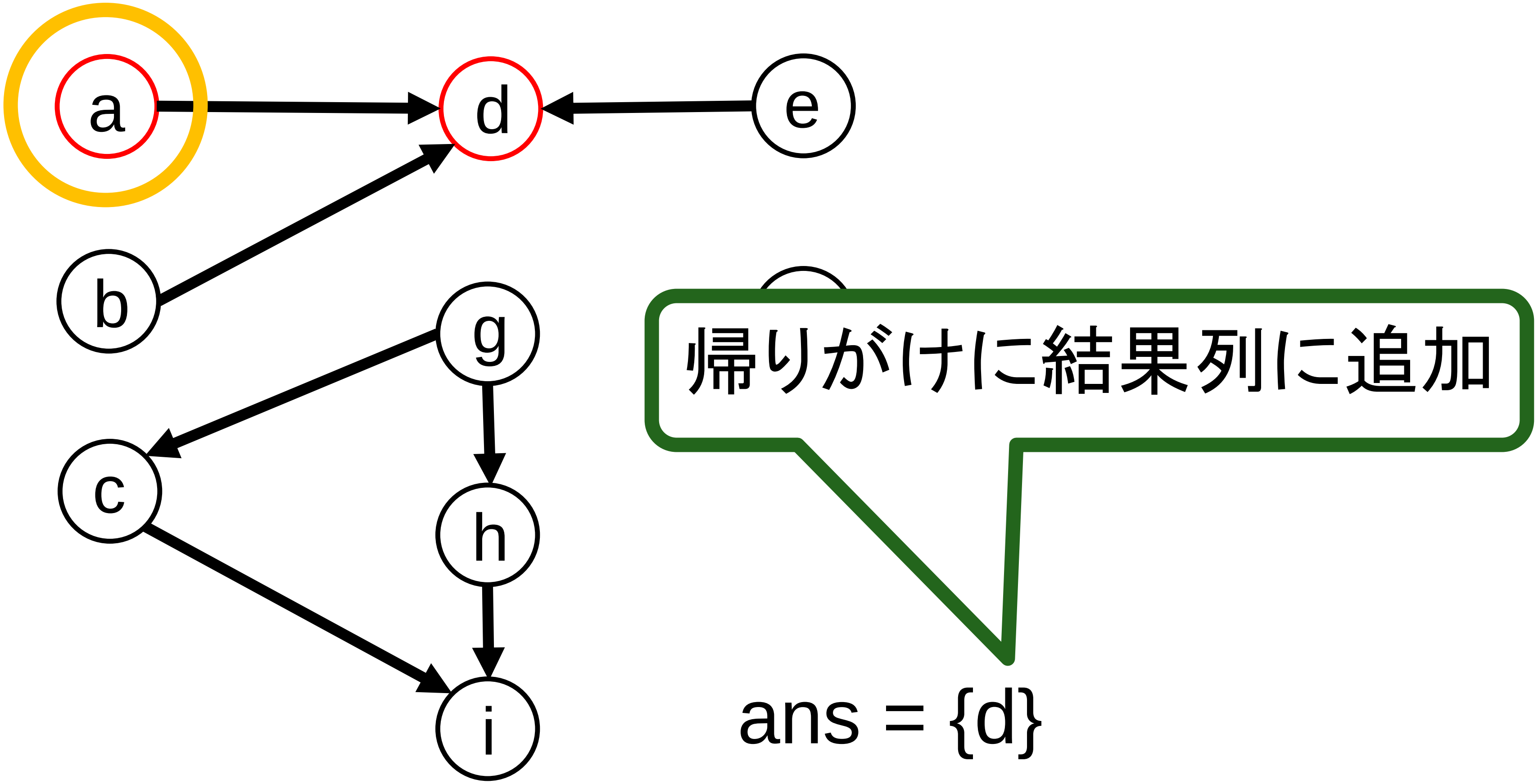
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {}

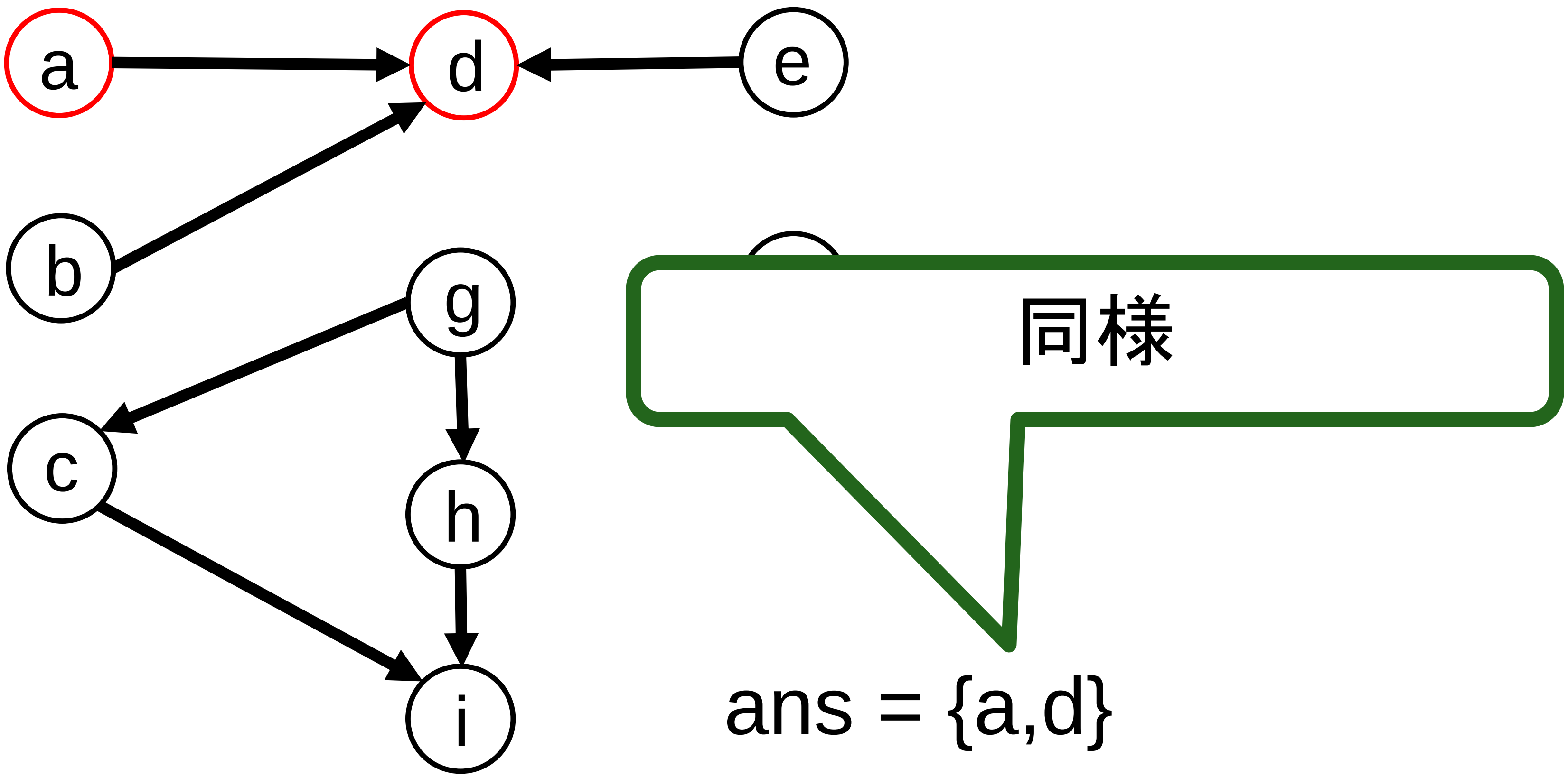
[Tarjan 1976] の動作例

- (どの順でもよいが)a,b,c,d,e,f,g,h,iの順でDFSする



[Tarjan 1976] の動作例

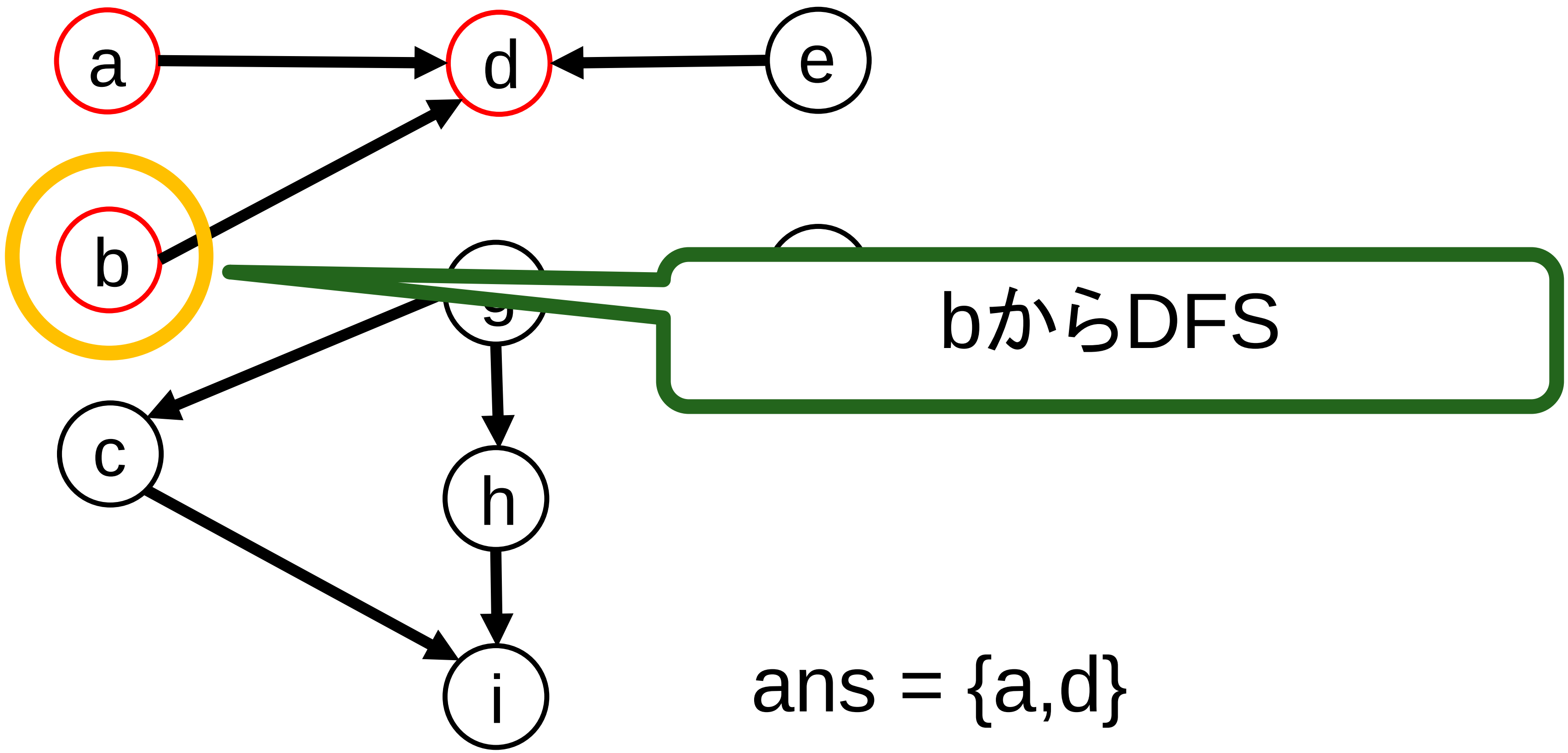
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {a,d}

[Tarjan 1976] の動作例

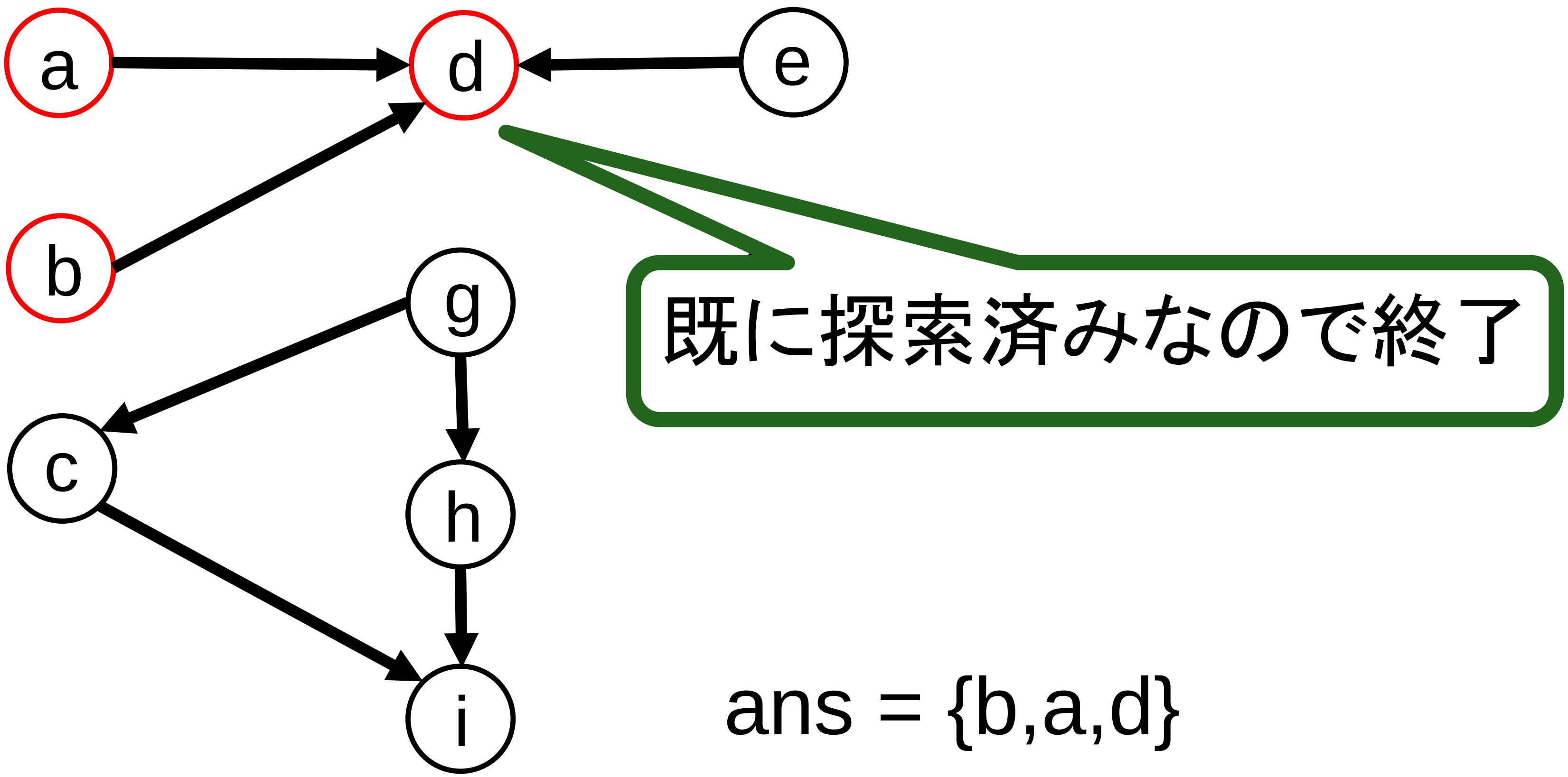
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {a,d}

[Tarjan 1976] の動作例

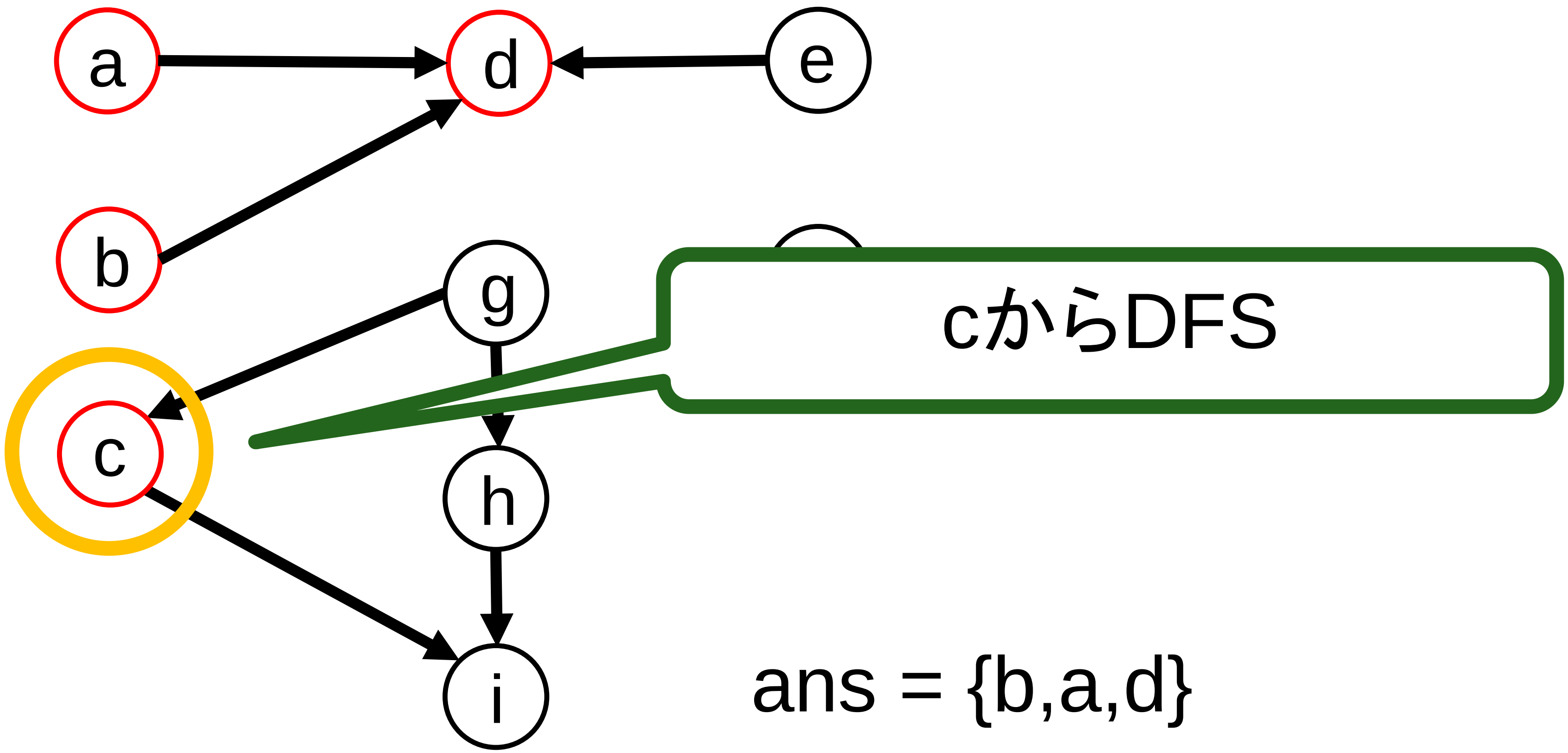
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {b,a,d}

[Tarjan 1976] の動作例

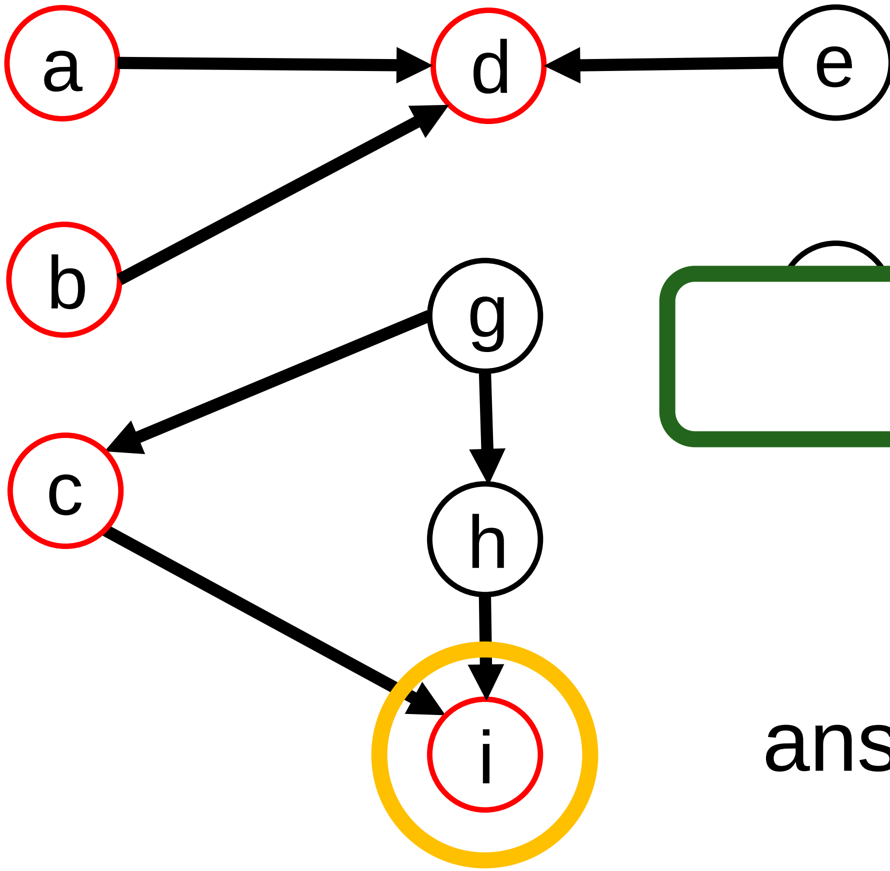
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {b,a,d}

[Tarjan 1976] の動作例

- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする

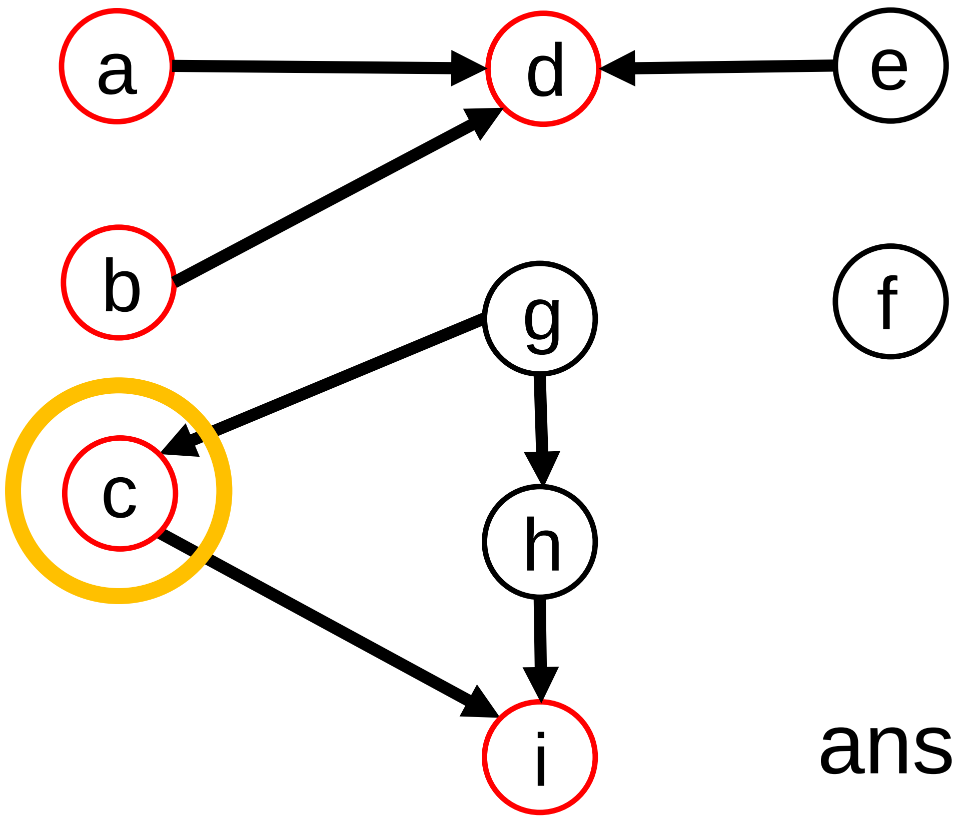


以下、同様

ans = {b,a,d}

[Tarjan 1976] の動作例

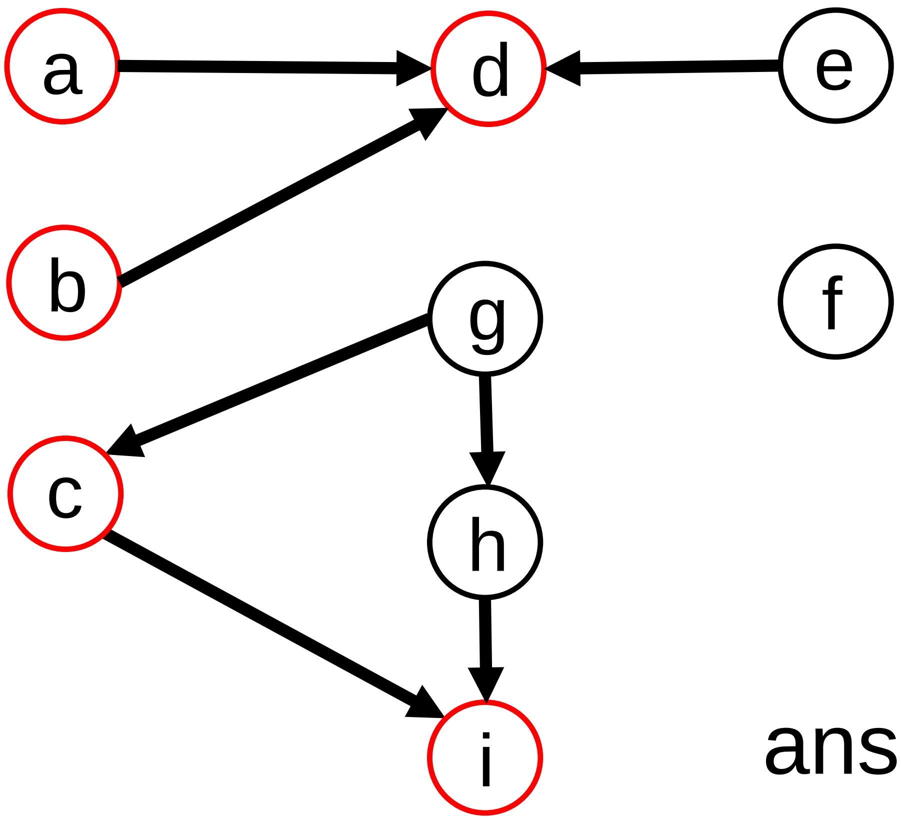
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {i,b,a,d}

[Tarjan 1976] の動作例

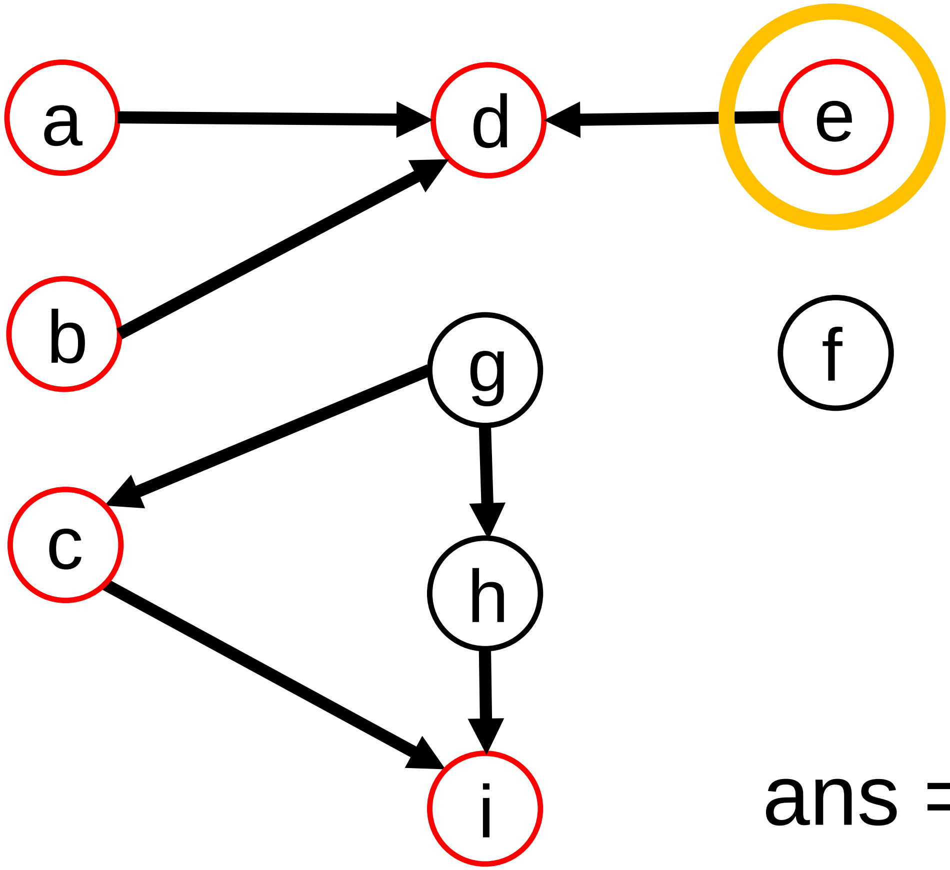
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {c,i,b,a,d}

[Tarjan 1976] の動作例

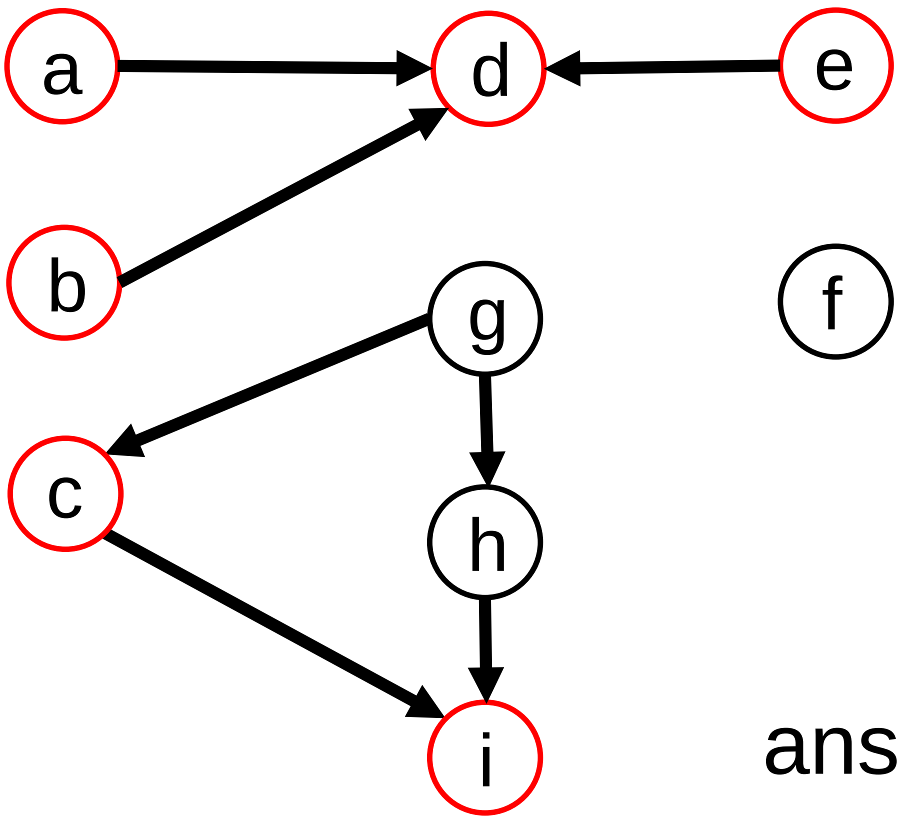
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {c,i,b,a,d}

[Tarjan 1976] の動作例

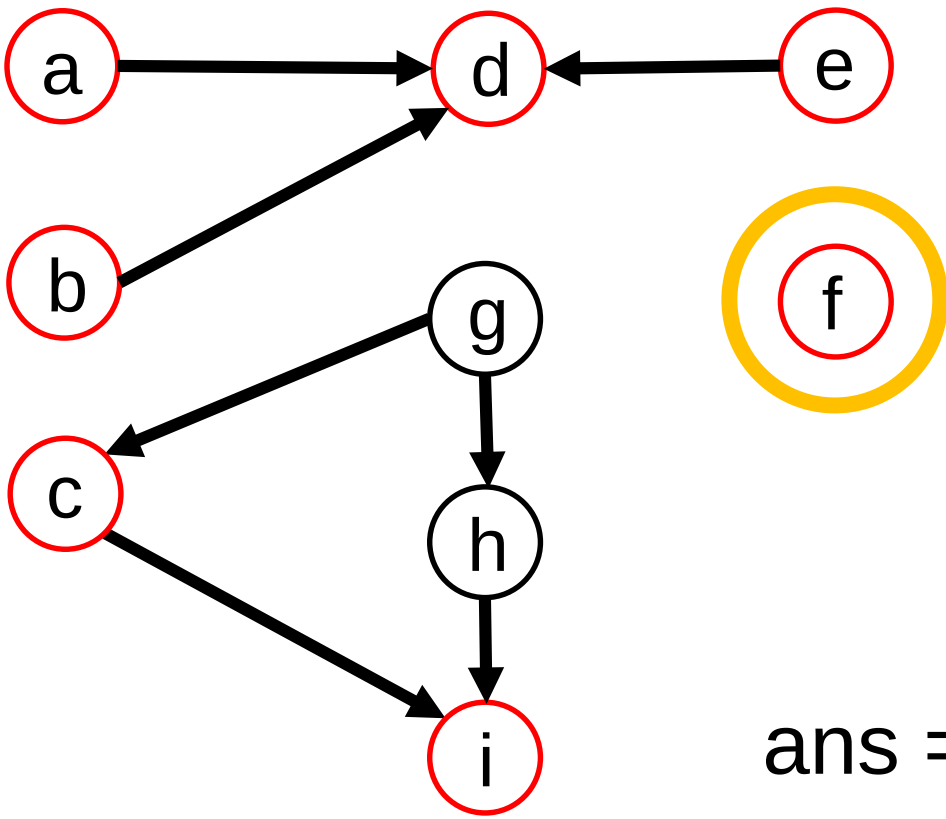
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {e,c,i,b,a,d}

[Tarjan 1976] の動作例

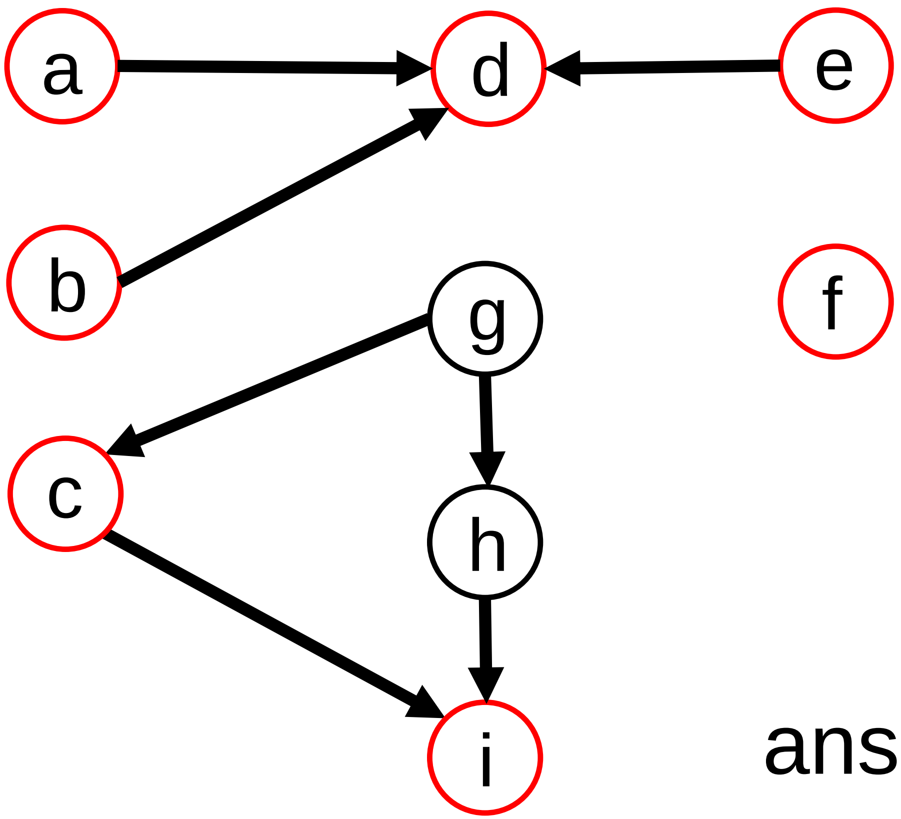
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {e,c,i,b,a,d}

[Tarjan 1976] の動作例

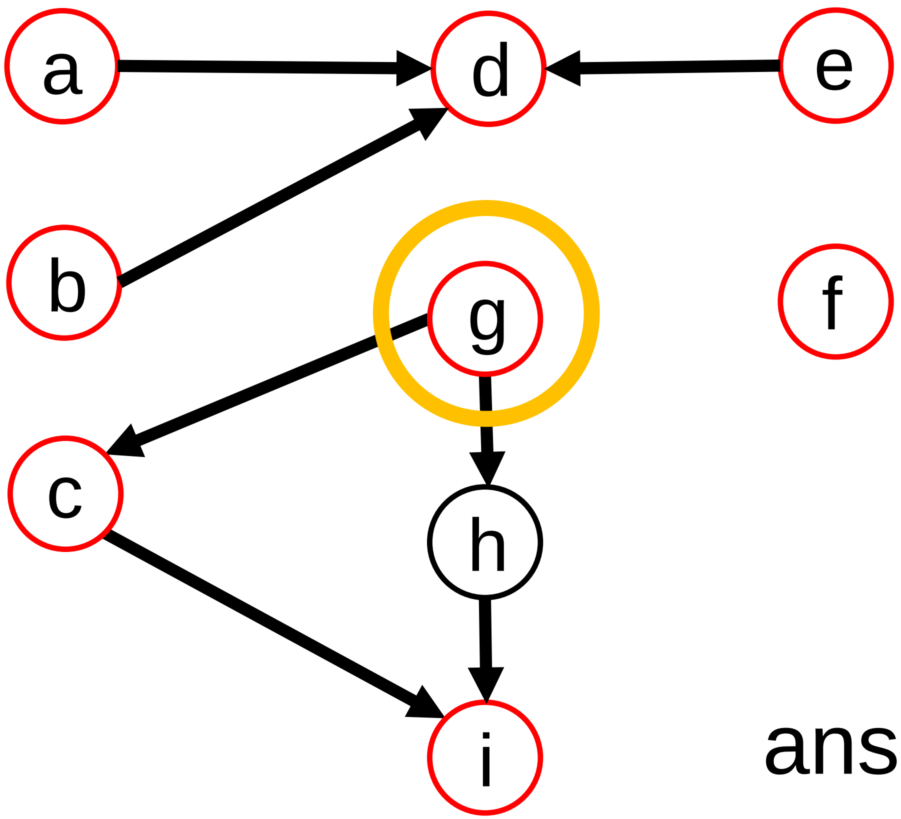
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {f,e,c,i,b,a,d}

[Tarjan 1976] の動作例

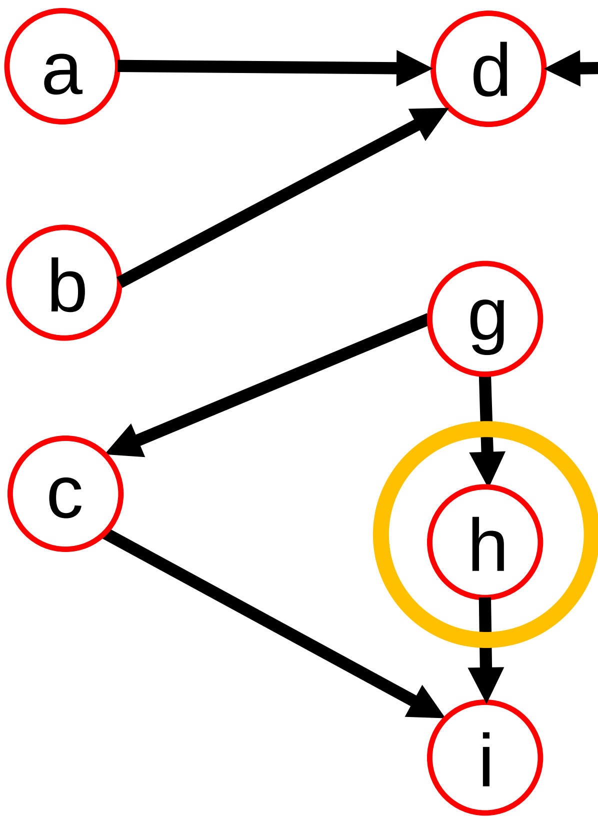
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {f,e,c,i,b,a,d}

[Tarjan 1976] の動作例

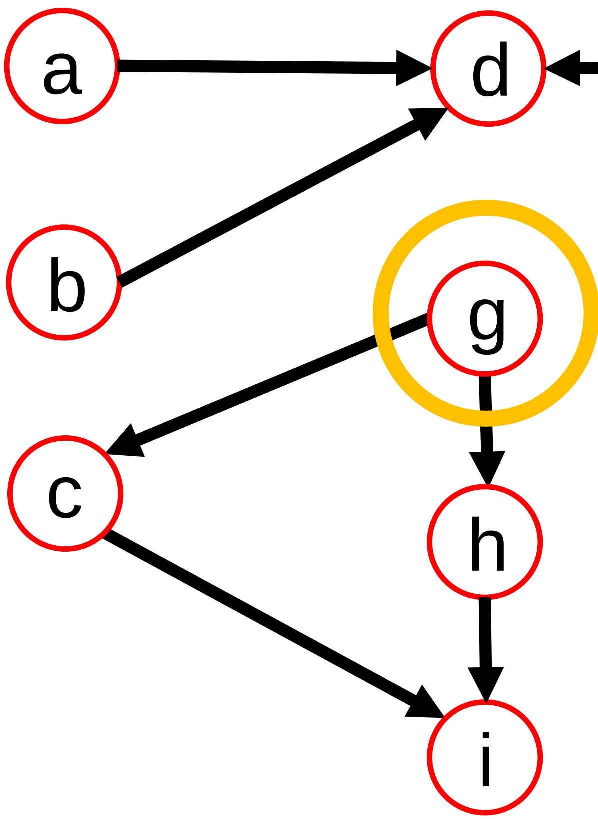
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {f,e,c,i,b,a,d}

[Tarjan 1976] の動作例

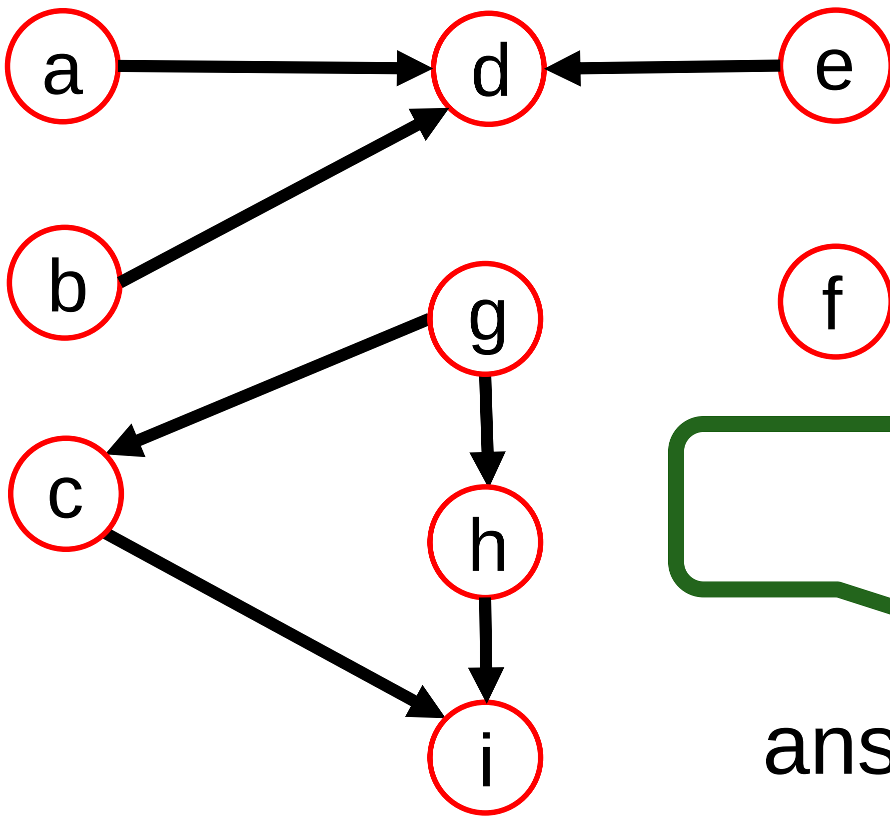
- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



ans = {h,f,e,c,i,b,a,d}

[Tarjan 1976] の動作例

- （どの順でもよいが）a,b,c,d,e,f,g,h,iの順でDFSする



答え

ans = {g,h,f,e,c,i,b,a,d}

[Tarjan 1976]の実装例

```
void visit(const vector<vector<int>>& g, int u, vector<bool>&used, vector<int>& ans) {

    if( used[u] == false ) {
        used[u] = true;
        for(auto& v : g[u]) {
            visit(g, v, used, ans);
        }
        ans.emplace_back(u);
    }
}
vector<int> tsort_Tarjan(const vector<vector<int>>& g) {
    const int V = g.size();

    vector<bool> used(V,false);
    vector<int> ans;
    for (int u=0; u<V; ++u) {
        visit(g, u, used, ans);
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

まとめ

- DAGならトポロジカルソートできる
- トポロジカルソートできるならDAG
- トポロジカルソートを求めるのは$O(V+E)$でできる
  - Kahnのアルゴリズム
  - Tarjanのアルゴリズム