アルゴリズム論

2017年6月19日

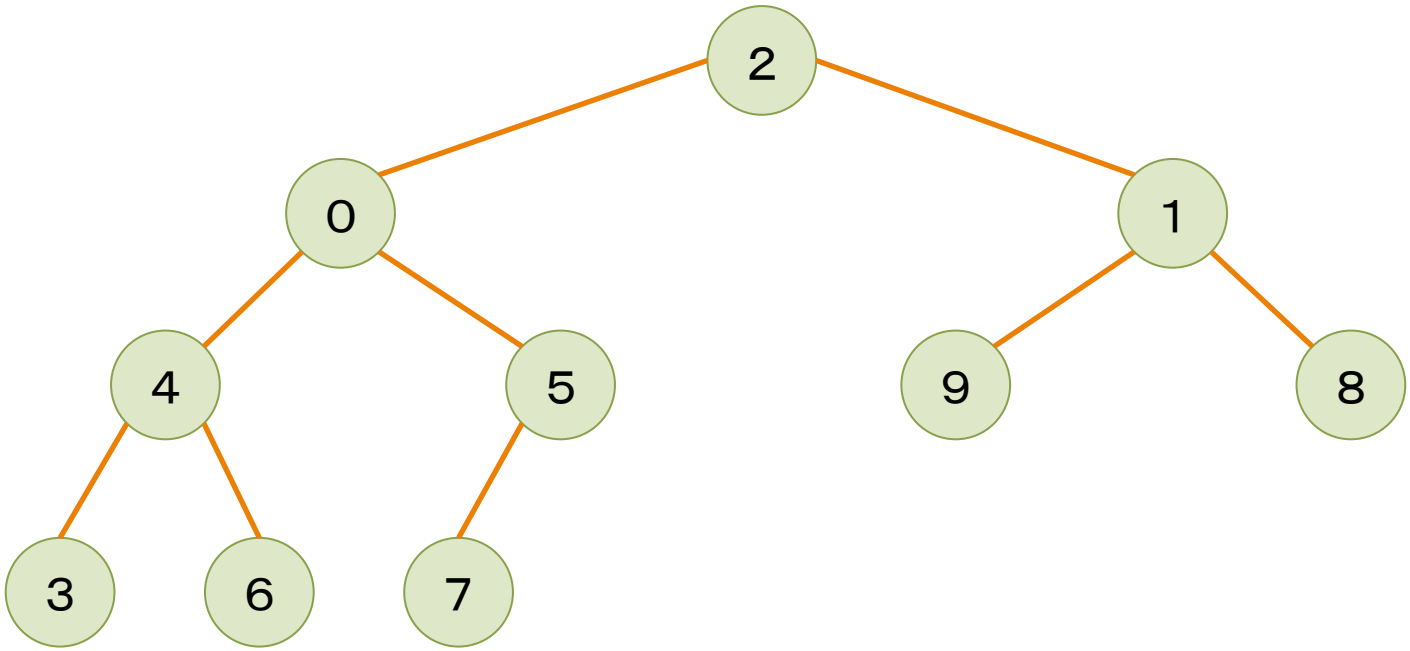
樋口文人

目次

- グラフ
  - 木構造との違い
    - 親が1つとは限らない
    - 閉路（ループ）がある
  - 無向グラフ
  - 有向グラフ
  - 全域木
  - 連結（強連結）
  - 表現方法
    - 隣接行列
    - 隣接リスト
  - グラフの探索（巡回）
    - 深さ優先探索
    - 幅優先探索
  - 問題
    - トポロジカルソート
    - 最小全域木
    - 最短経路
    - 巡回セールスマン問題

グラフ

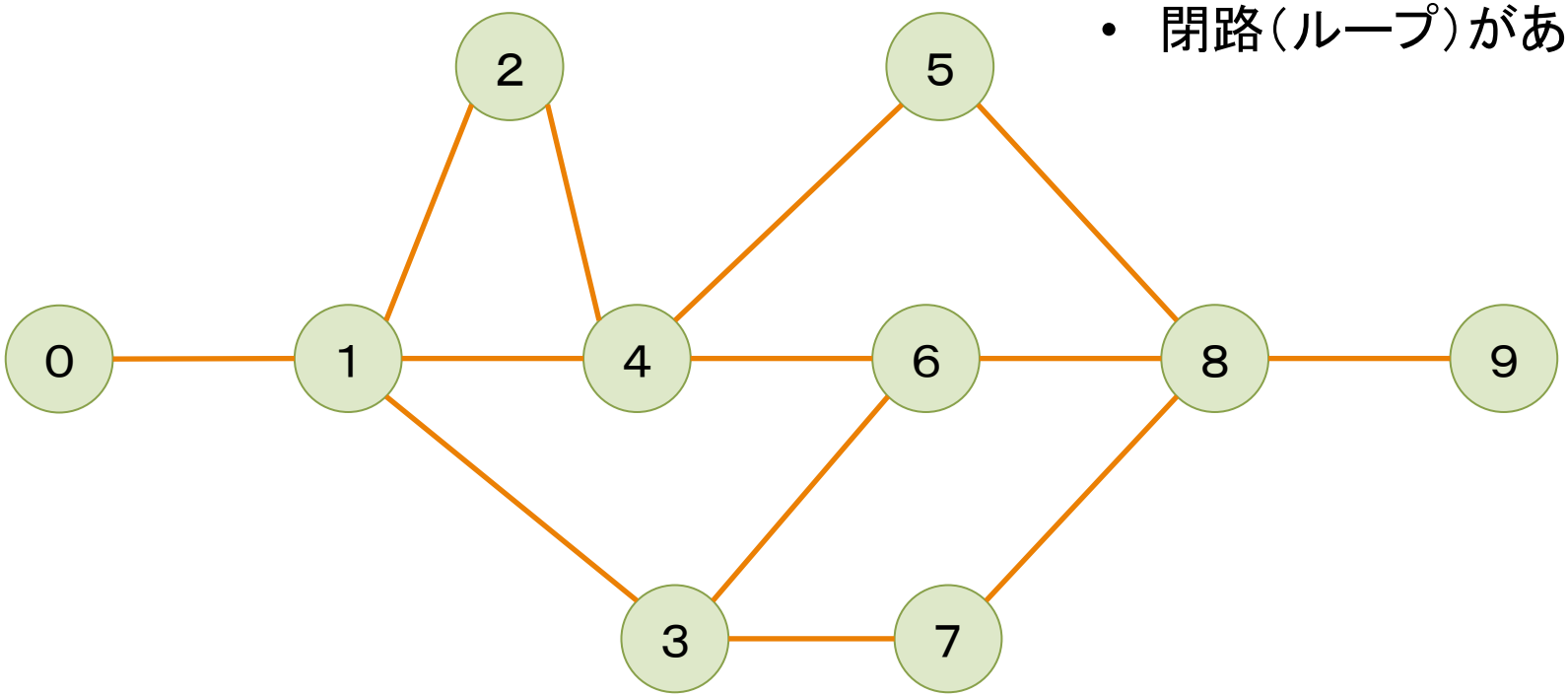
木構造



グラフ

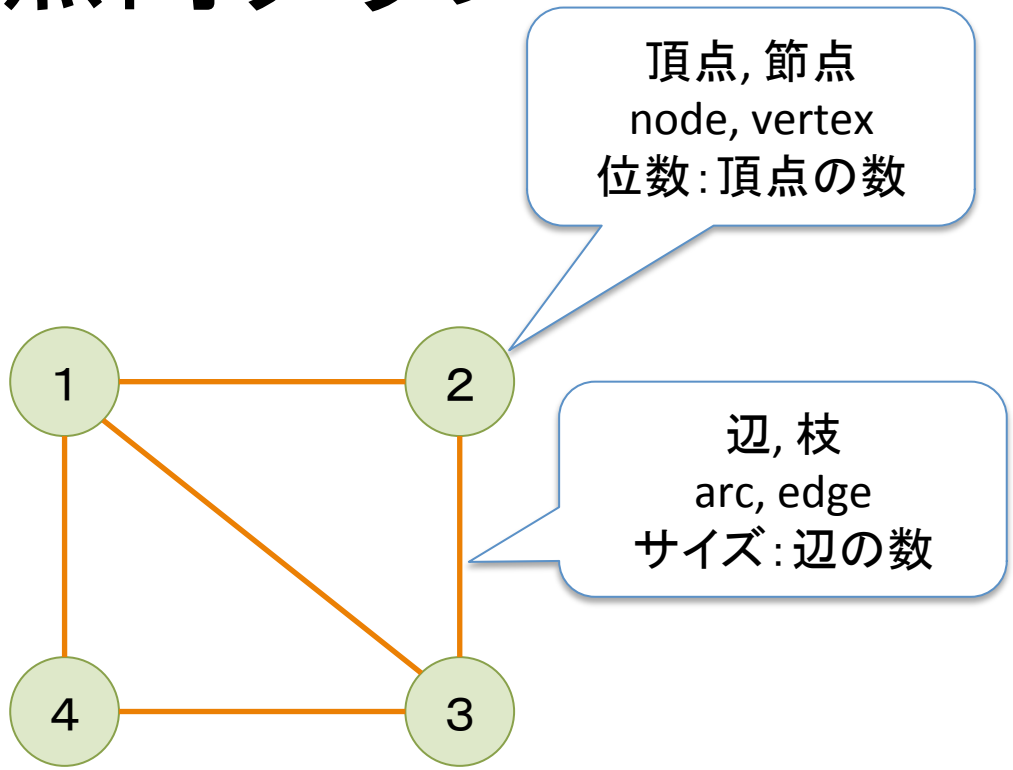
木構造との違い

- 親が1つとは限らない
- 閉路（ループ）がある



グラフに関する用語

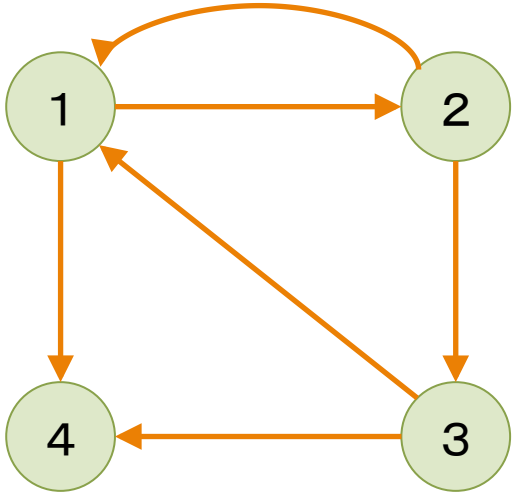
無向グラフ



有向グラフ

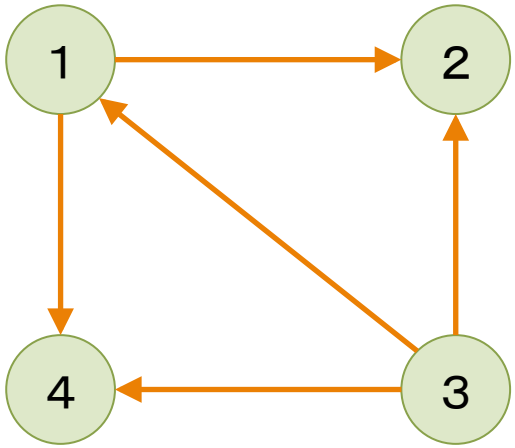
道（path）：辺で結ばれた隣接する頂点の列
例：{1, 2, 3, 4}, {3, 1, 2}, {1, 2, 1, 4}
また，重複する頂点が無いとき単純路という

頂点の次数：接続する辺の数



閉路（cycle）：最初と最後の頂点が同一である道
例：{1, 2, 3, 1}, {1, 2, 1}
また，途中に重複する頂点が無いとき単純閉路という

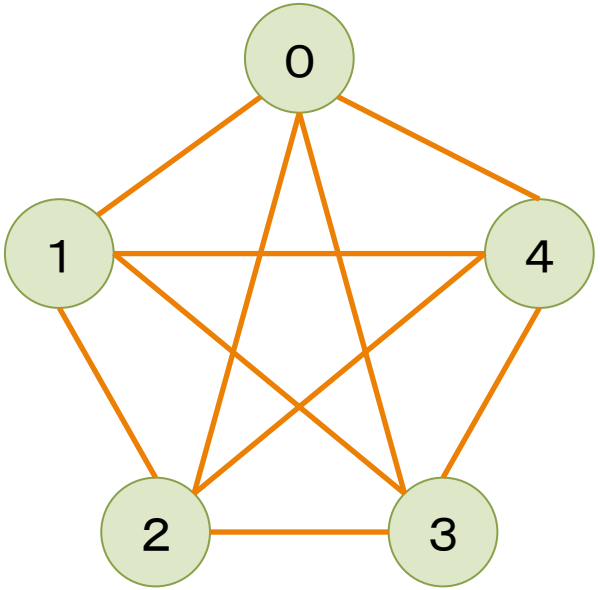
DAG: Directed Acyclic Graph

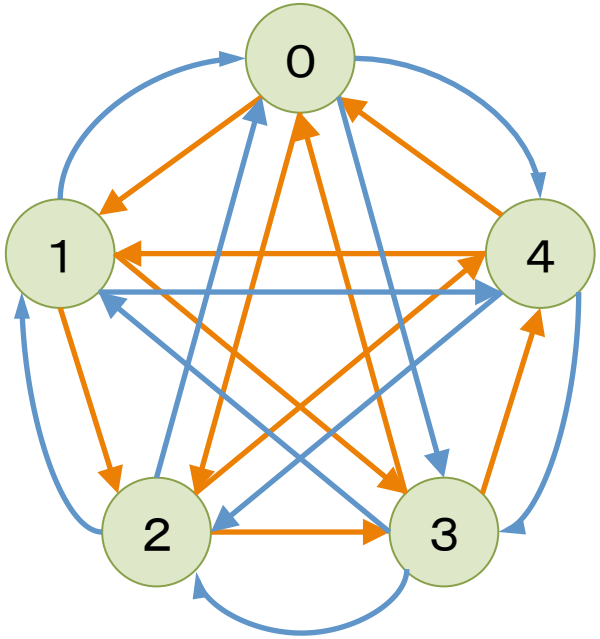


無閉路有向グラフ

完全グラフ

頂点数:n
辺の数:e



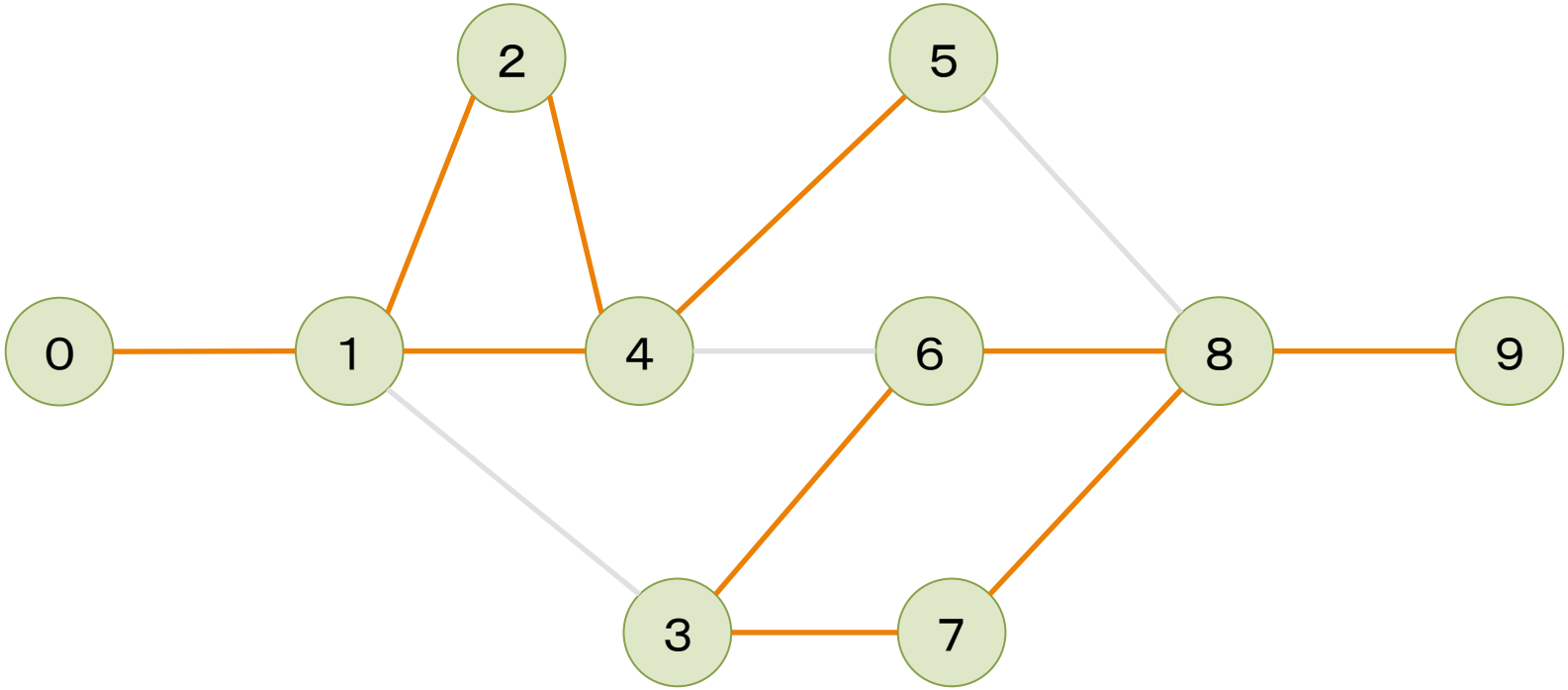


e = n(n-1)/2

e = n(n-1)

連結グラフ

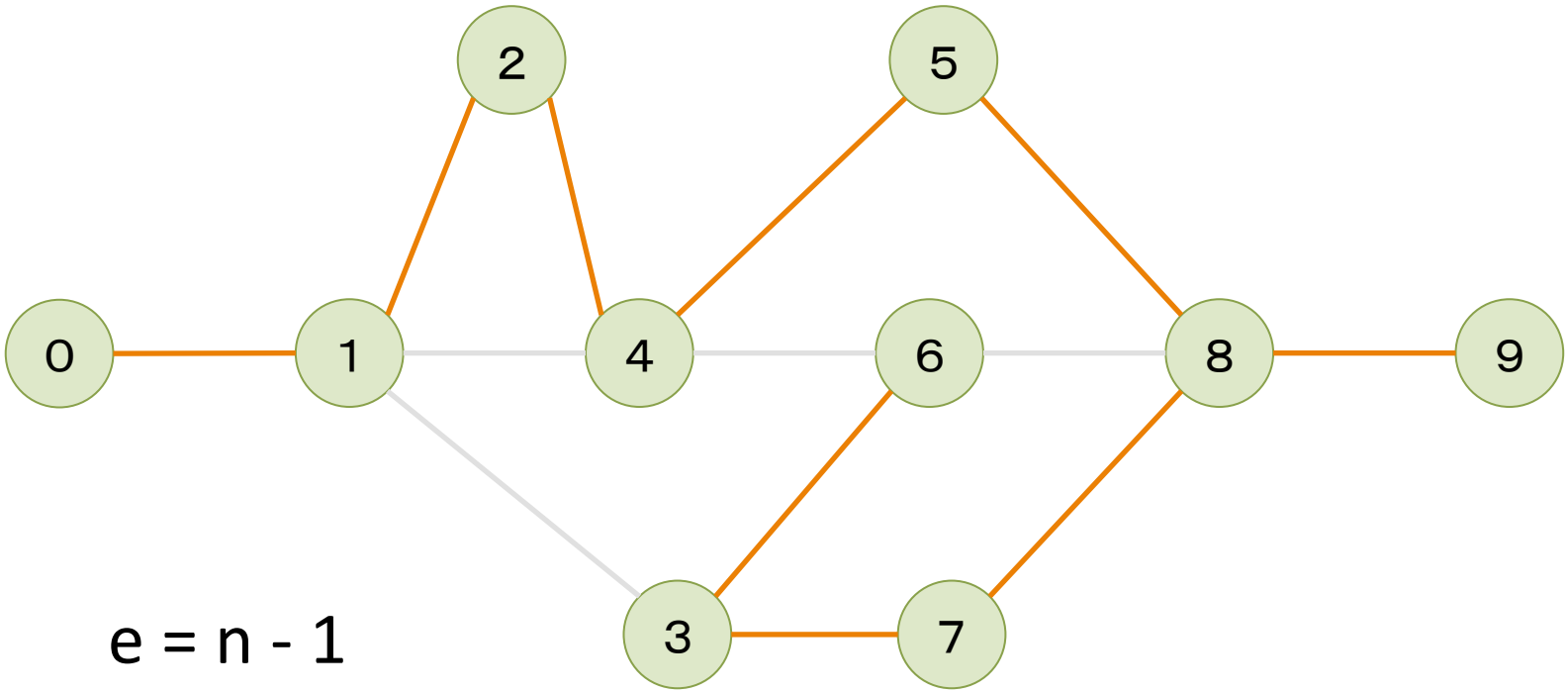
全ての節点から残りの全ての節点への道があるグラフ



強連結：有向グラフで任意の2頂点間に道がある

全域木

連結グラフの部分グラフで，全ての節点を含む単一の木

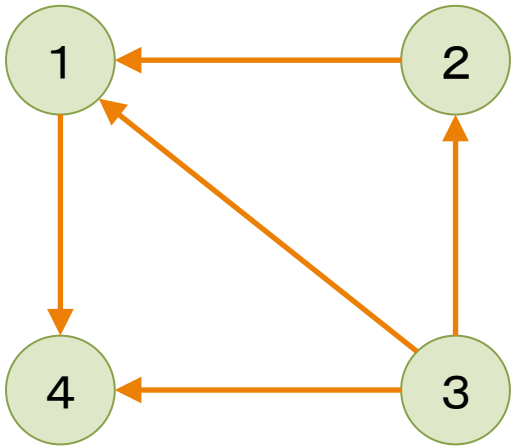


e = n - 1

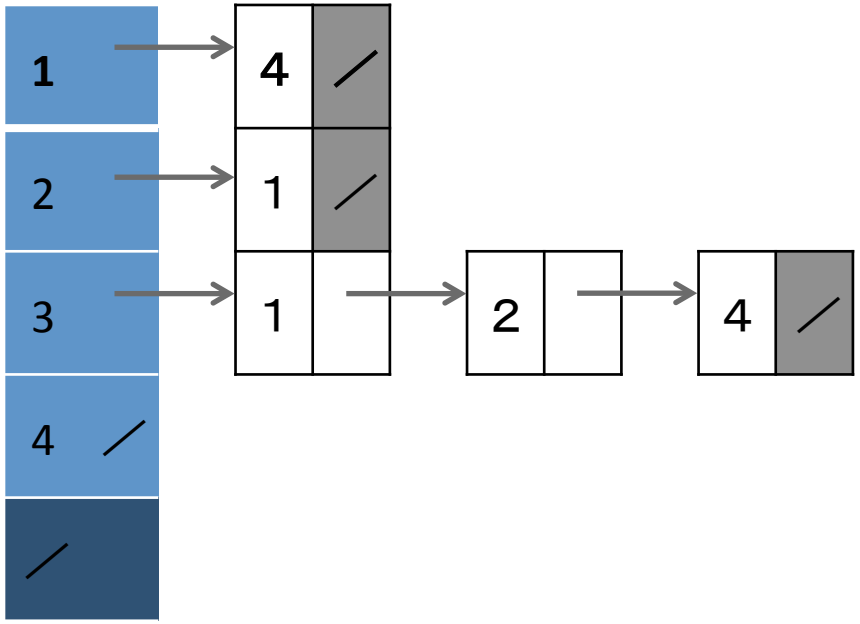
グラフの表現

隣接行列

	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	1	1	0	1
4	0	0	0	0



隣接リスト



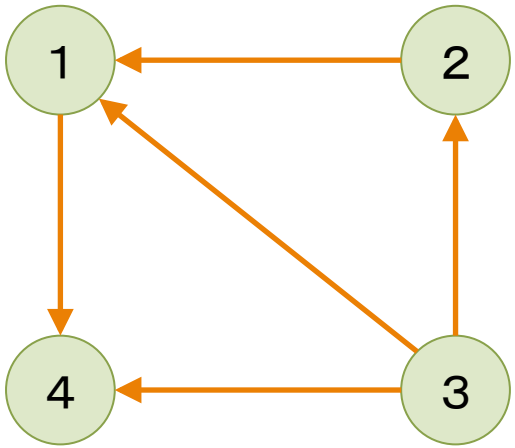
グラフの表現 補足

辺行列

	1	2
1	1	4
2	2	1
3	3	1
4	3	2
5	3	4

接続行列

	1	2	3	4	5
1	1	-1	-1	0	0
2	0	1	0	-1	0
3	0	0	1	1	1
4	-1	0	0	0	-1



表現上の特徴

- 隣接行列
  - 頂点が多く，次数が低いとメモリー効率が悪くなる
  - 無向グラフ
    - 対称行列
  - 有向グラフ
    - （一般に）非対称
    - 転置行列の意味は？
  - 閉路の存在
- 隣接リスト
  - メモリー効率は高い
  - リスト構造
    - 基本的には有向グラフ
    - 閉路は分かりにくい

表現上の特徴 補足

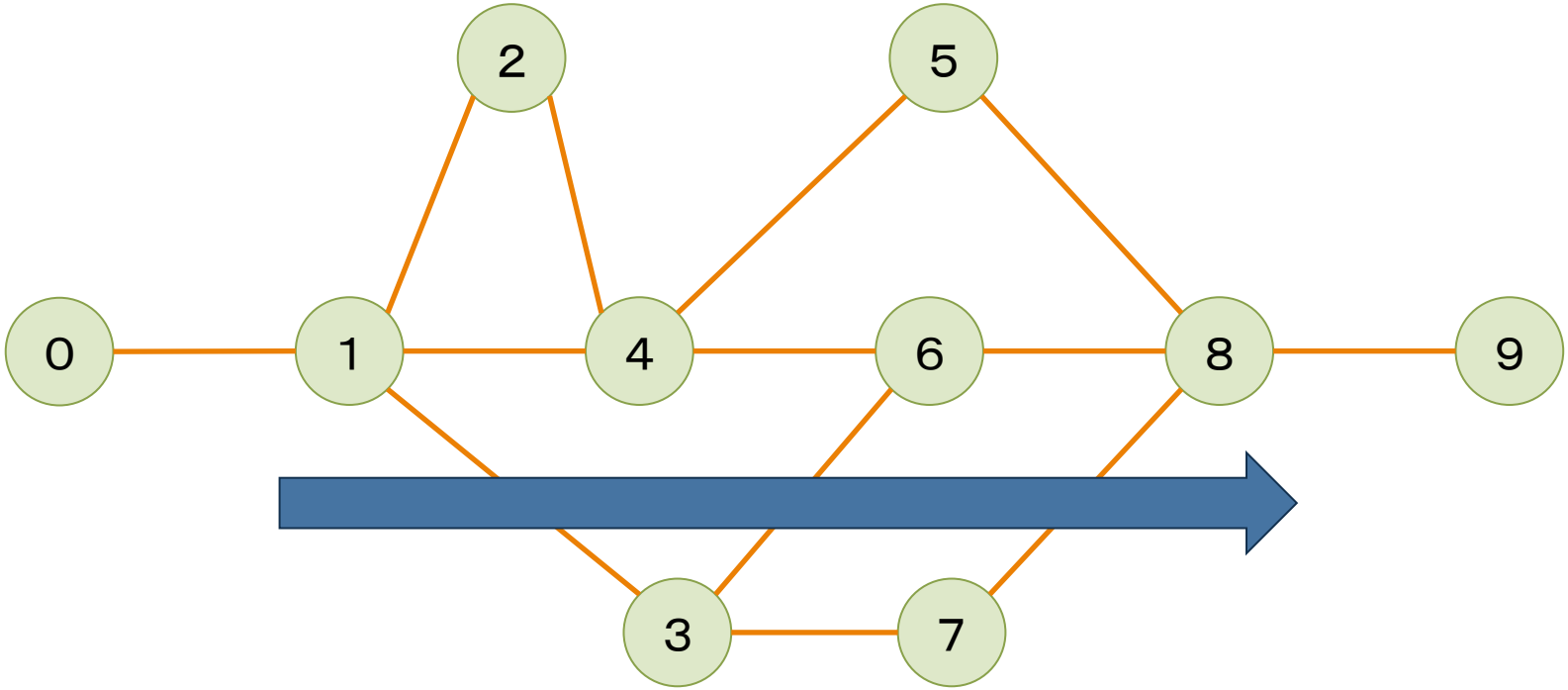
- 隣接行列
  - この行列をAとする
  - 節点iから節点j向かう辺があるとき $a_{ij} = 1$
    それ以外 $a_{ij} = 0$
  - このとき
    A^2 とか A^3 にはどんな意味があるか？
- 連結行列
  - この行列をCとする
  - i = j, あるいは，節点iから節点jへの道があるとき $c_{ij} = 1$
    それ以外 $c_{ij} = 0$
  - m個の接点があるとき
    Cの0成分の位置は主対角成分を除いて
    $A+A^2+A^3+\ldots+A^{m-1}$
    の0成分の位置と一致する

グラフの探索

- 横断ともいう（木構造の巡回）
  - 深さ優先探索 (DFS: Depth First Search)
  - 幅優先探索 (BFS: Breadth First Search)
- cf. 木構造
  - 行き掛け
  - 通り掛け
  - 帰り掛け

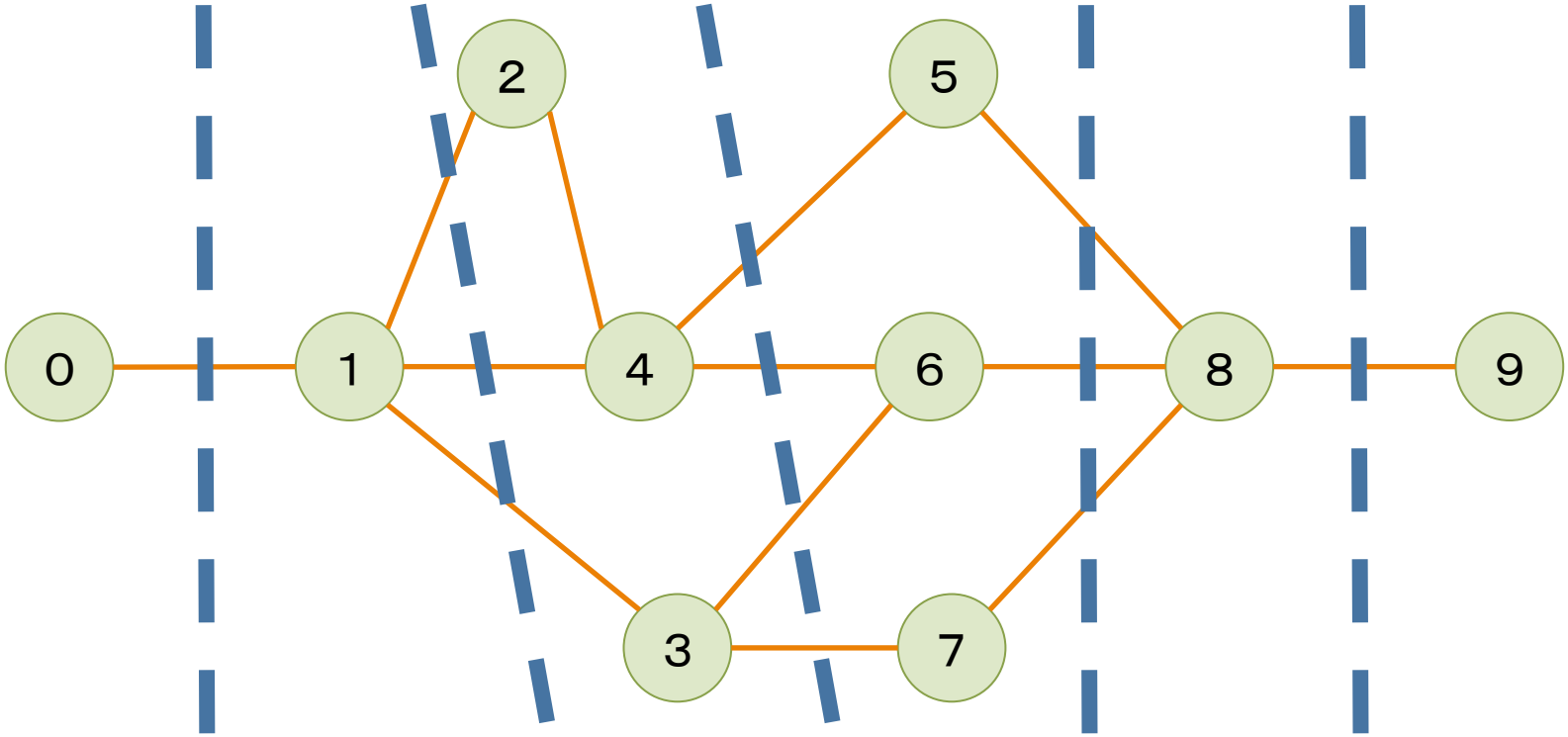
DFS

探索開始頂点を0とすると



BFS

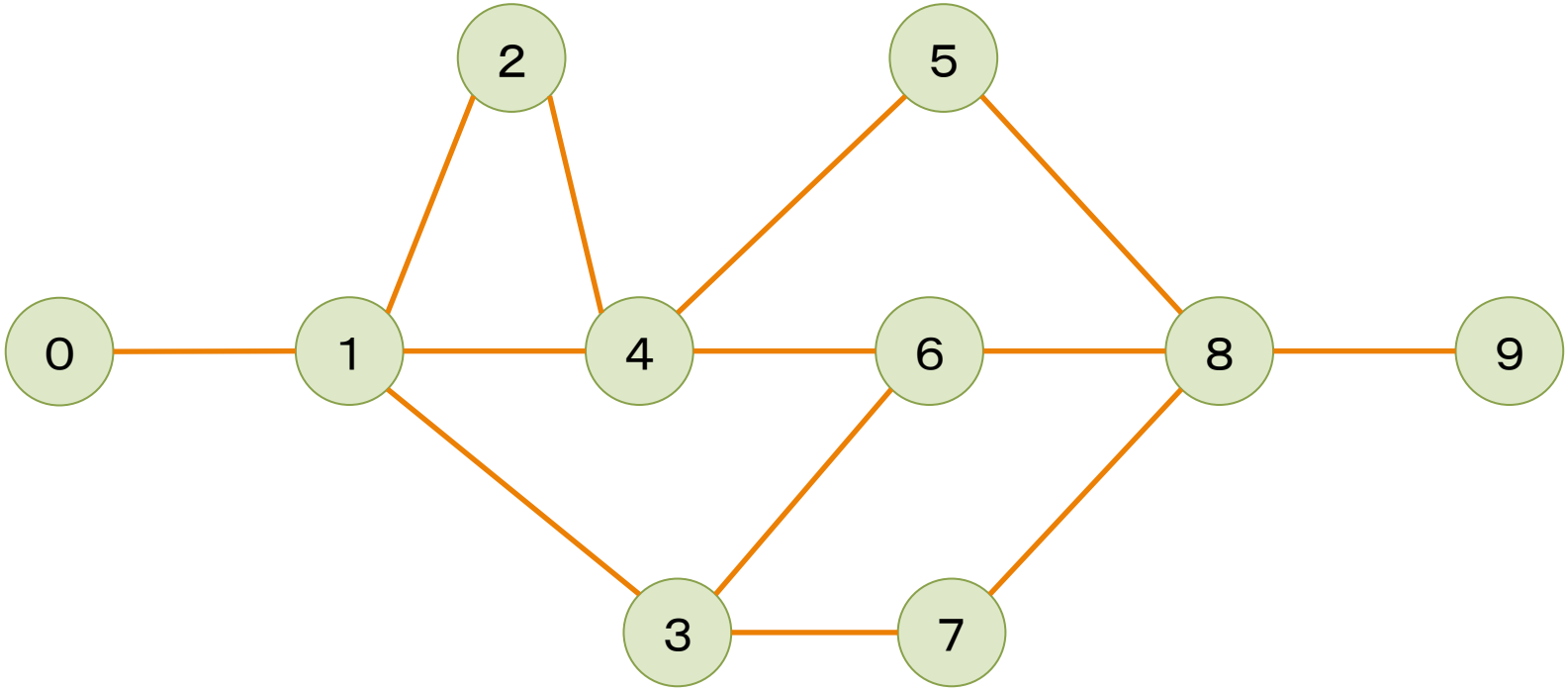
探索開始頂点を0とすると



出発点から同一路長の節を優先

DFS

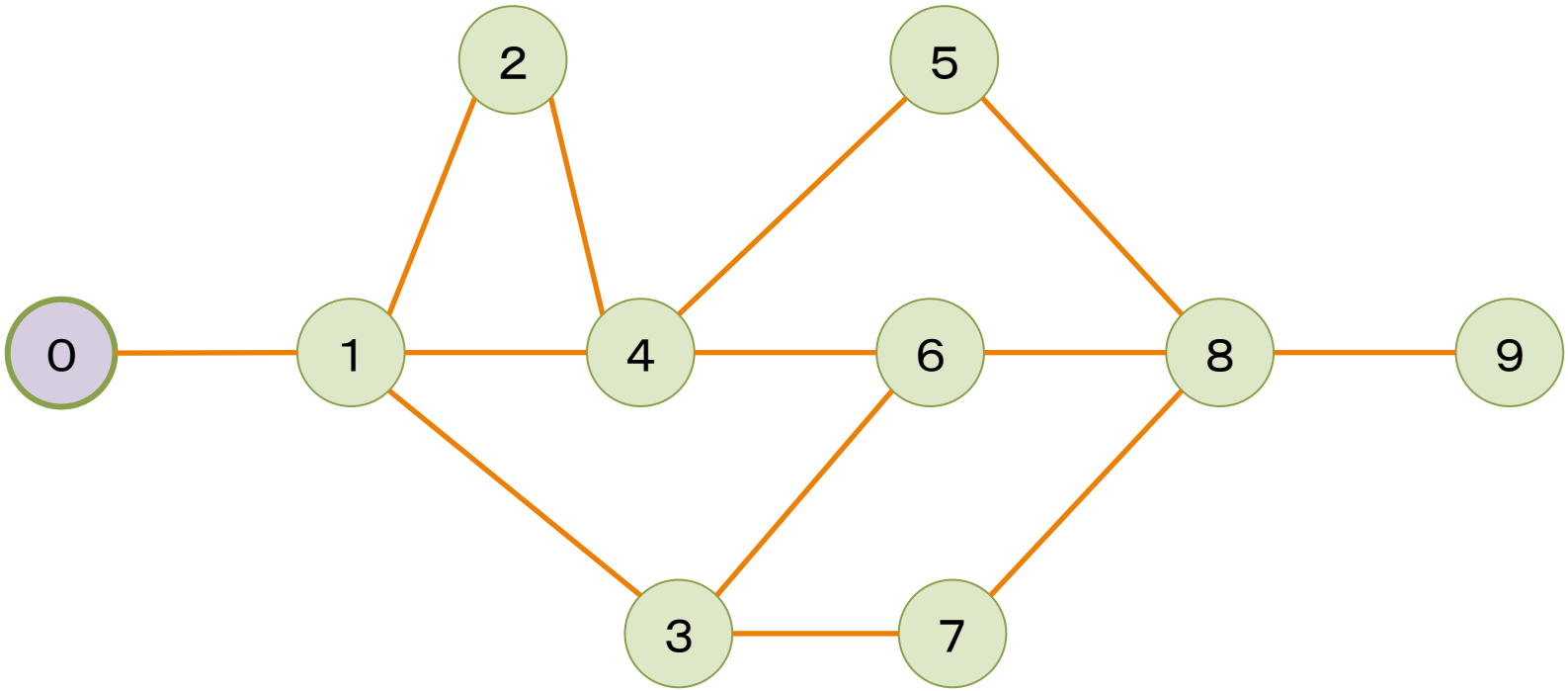
探索開始頂点を0とすると



全ての頂点を未訪問としておく

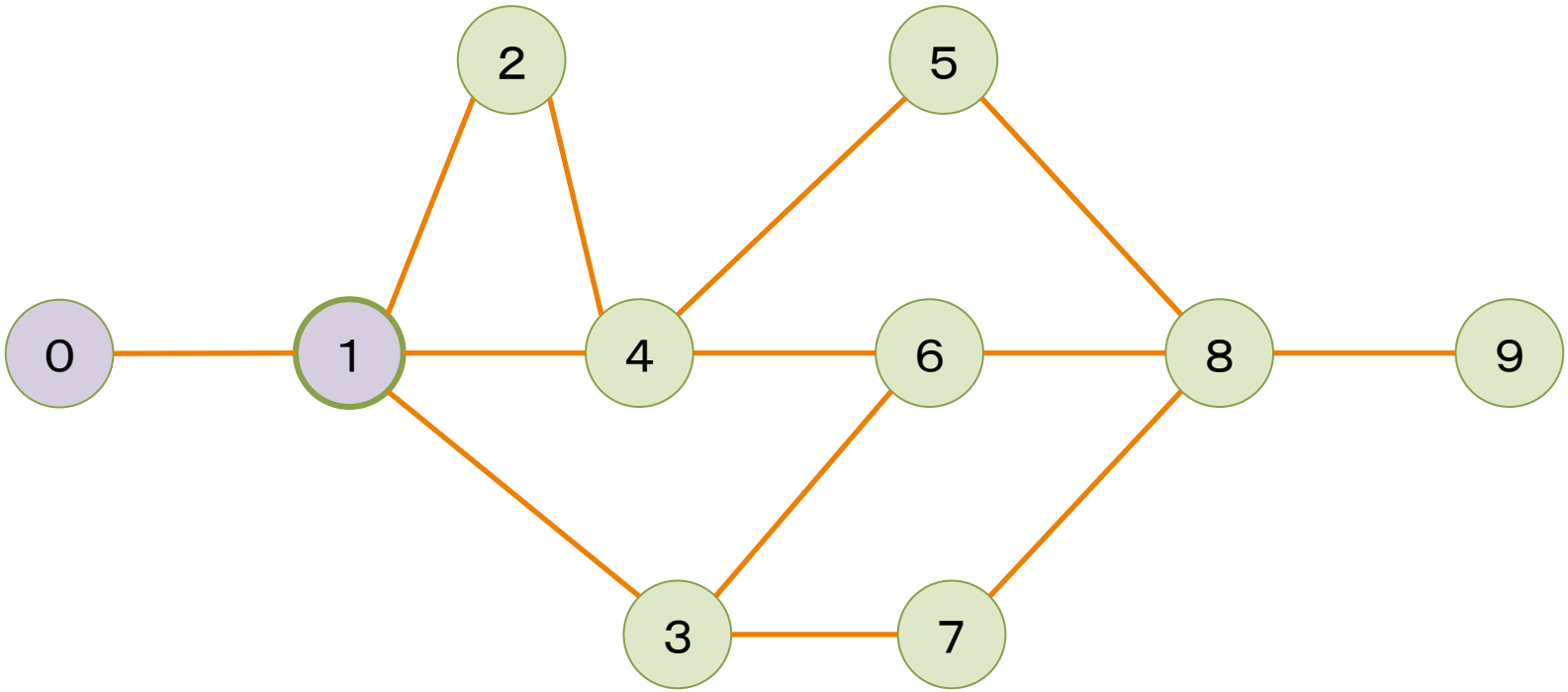
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

DFS step 1



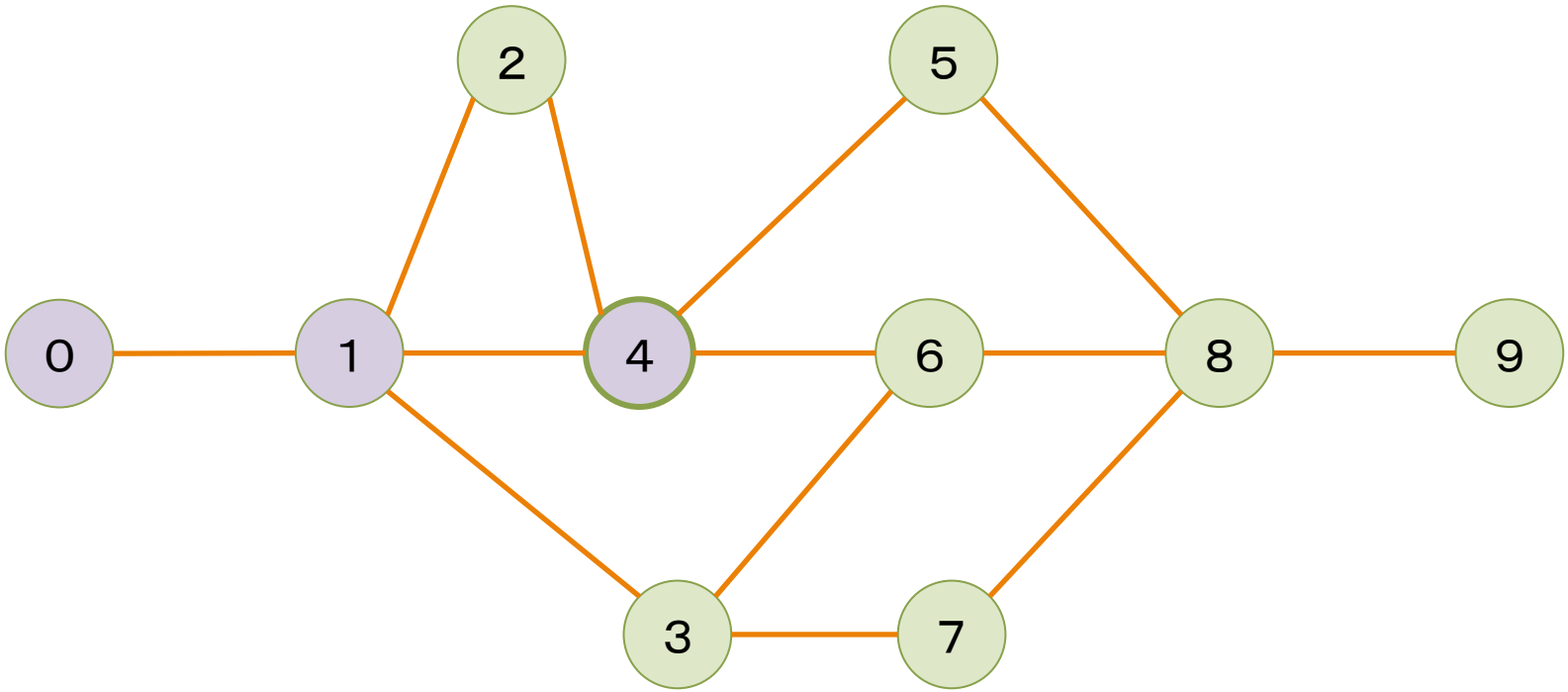
0	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0

DFS step 2



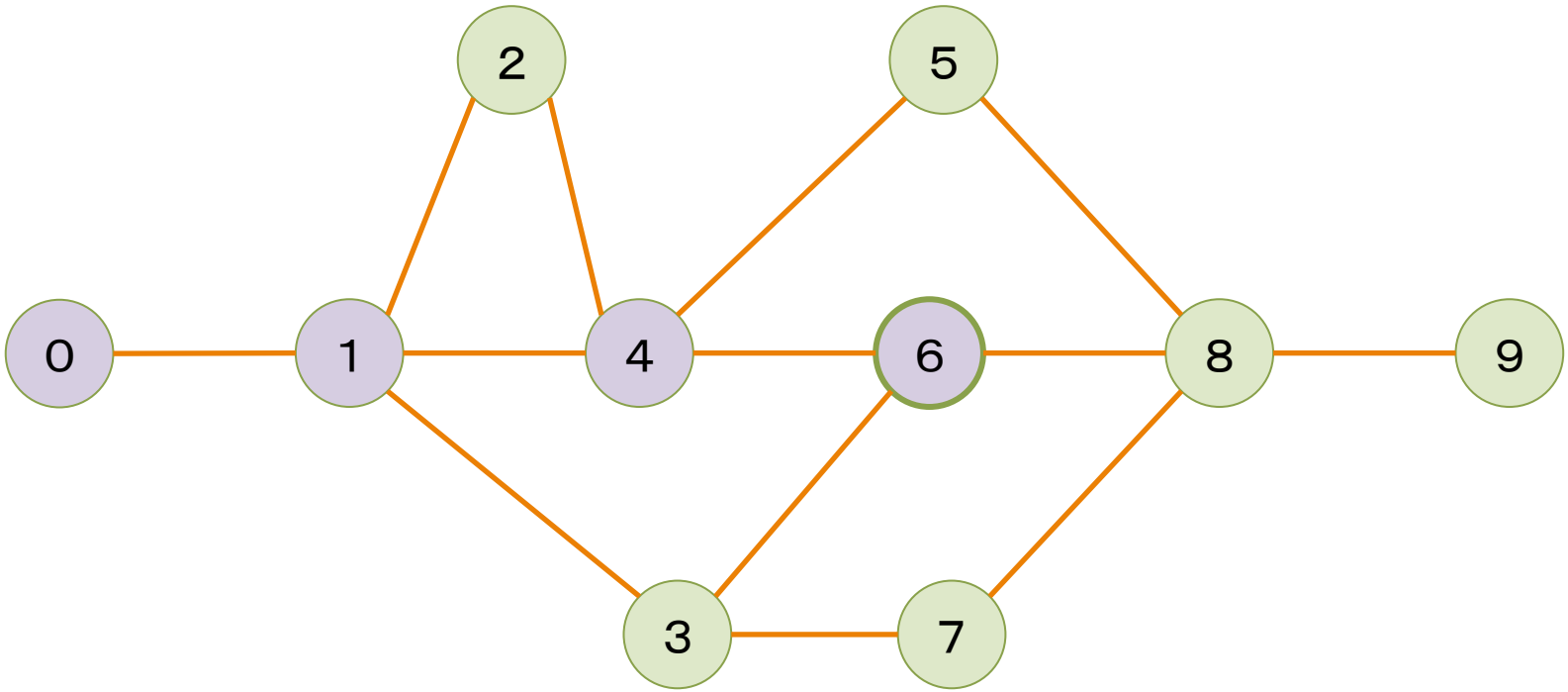
0	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0

DFS step 3



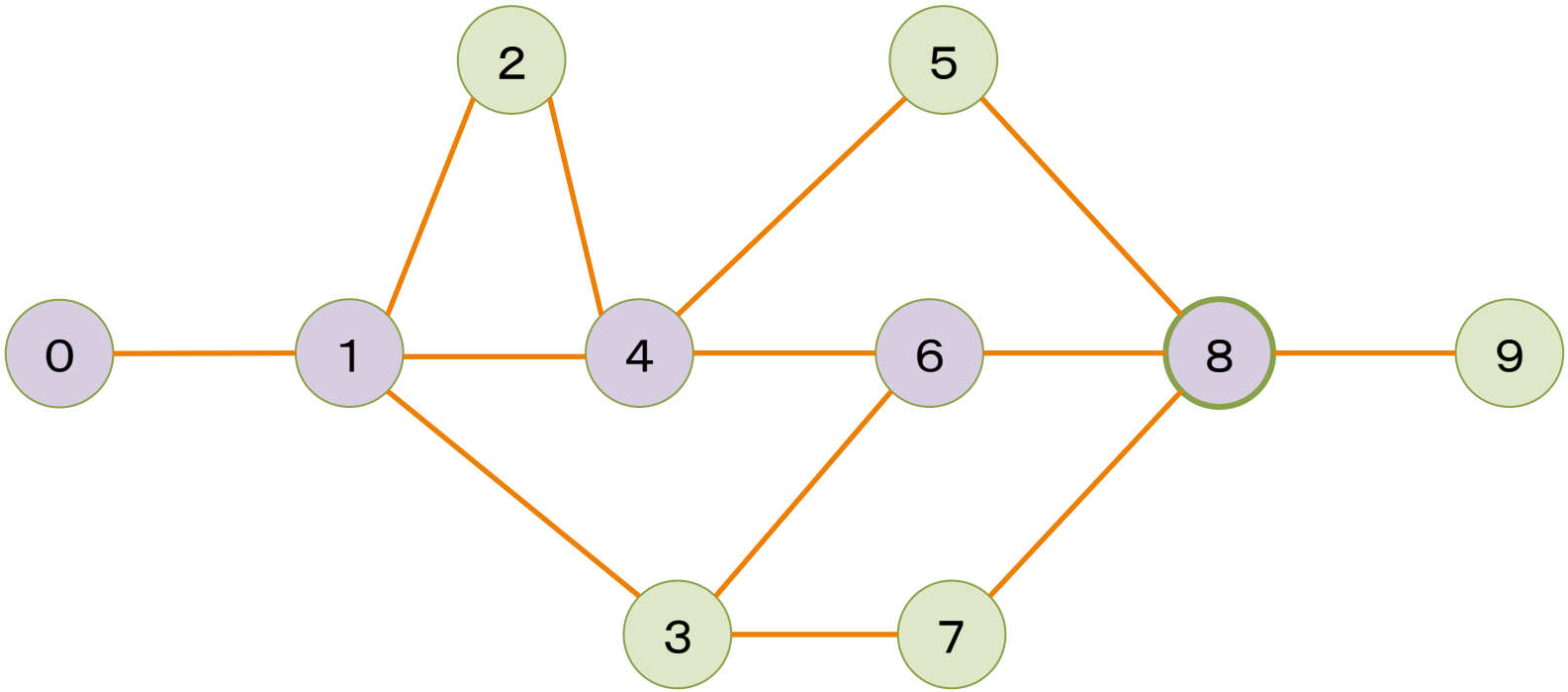
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	0	0	0	0

DFS step 4



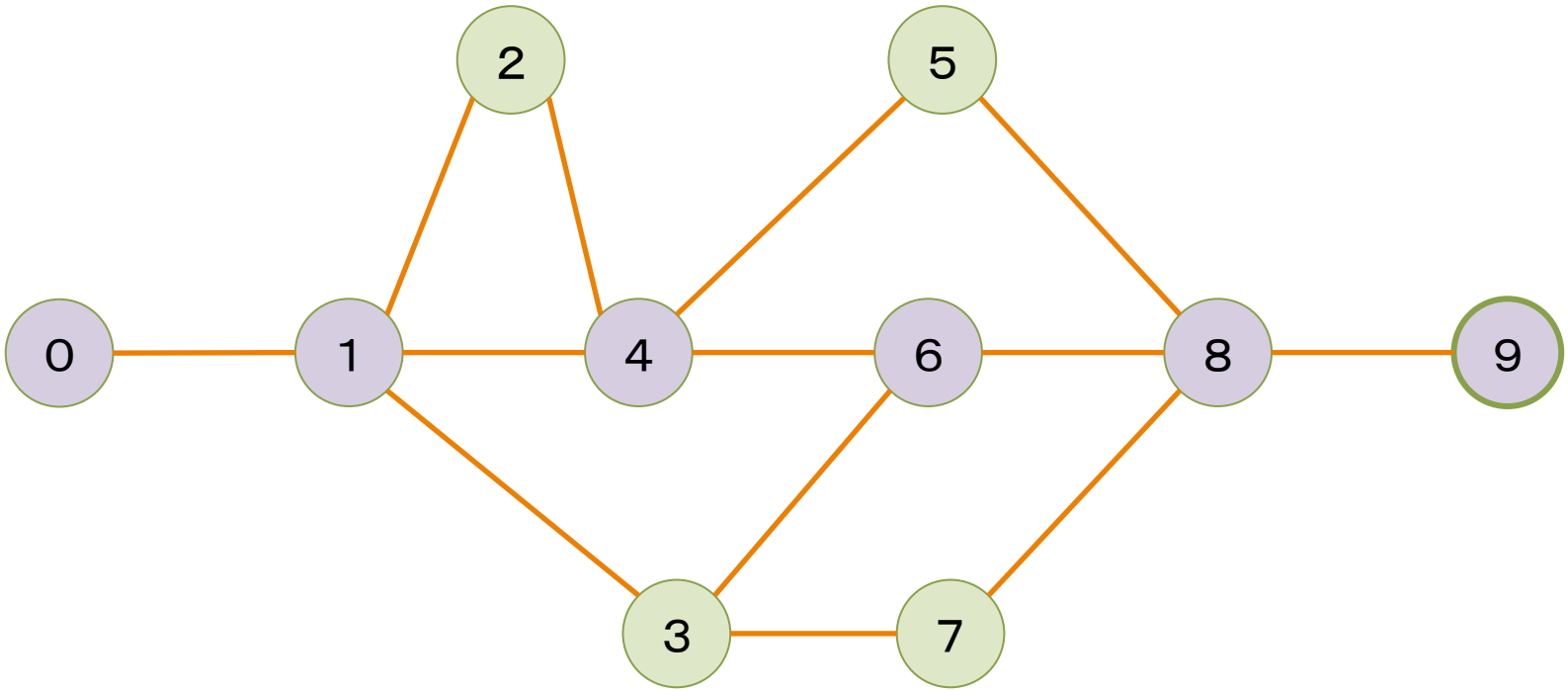
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	0	0	0

DFS step 5



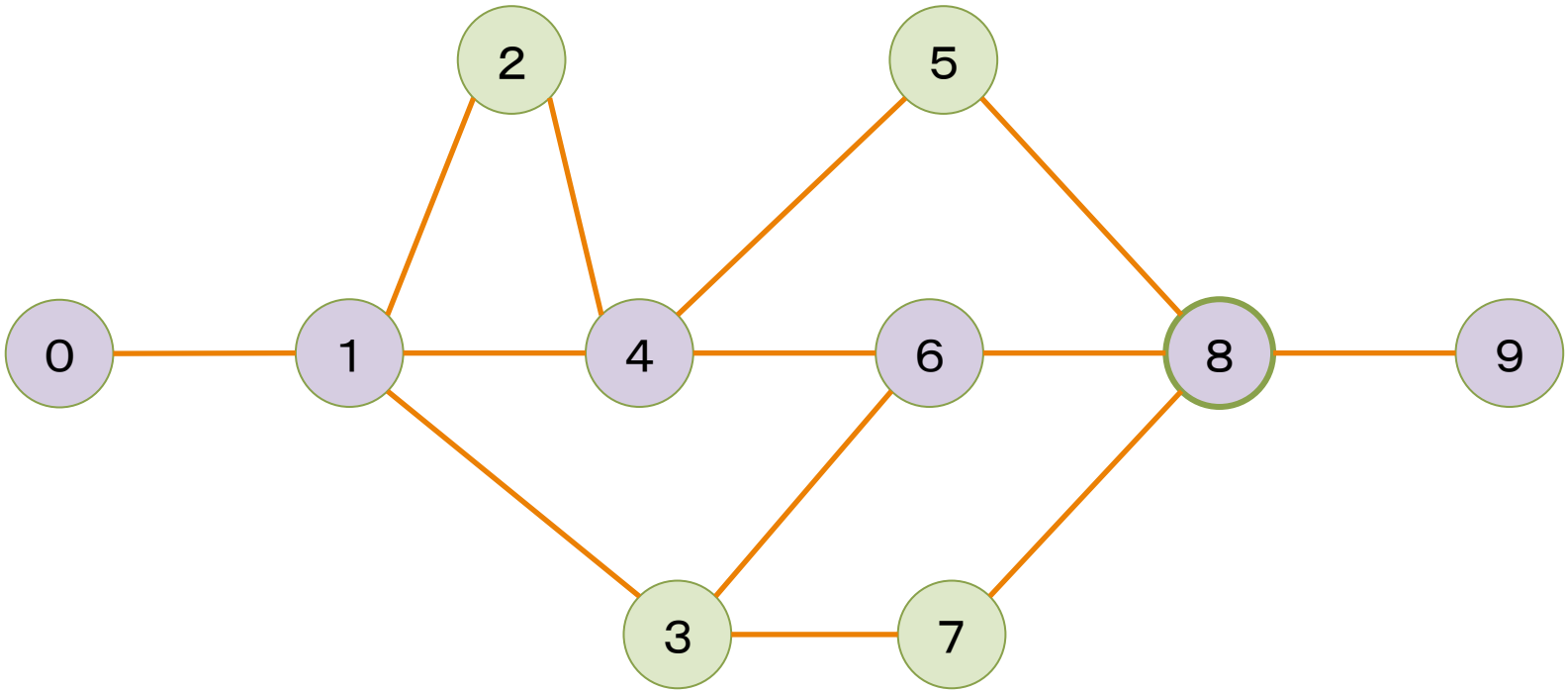
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	0	1	0

DFS step 6



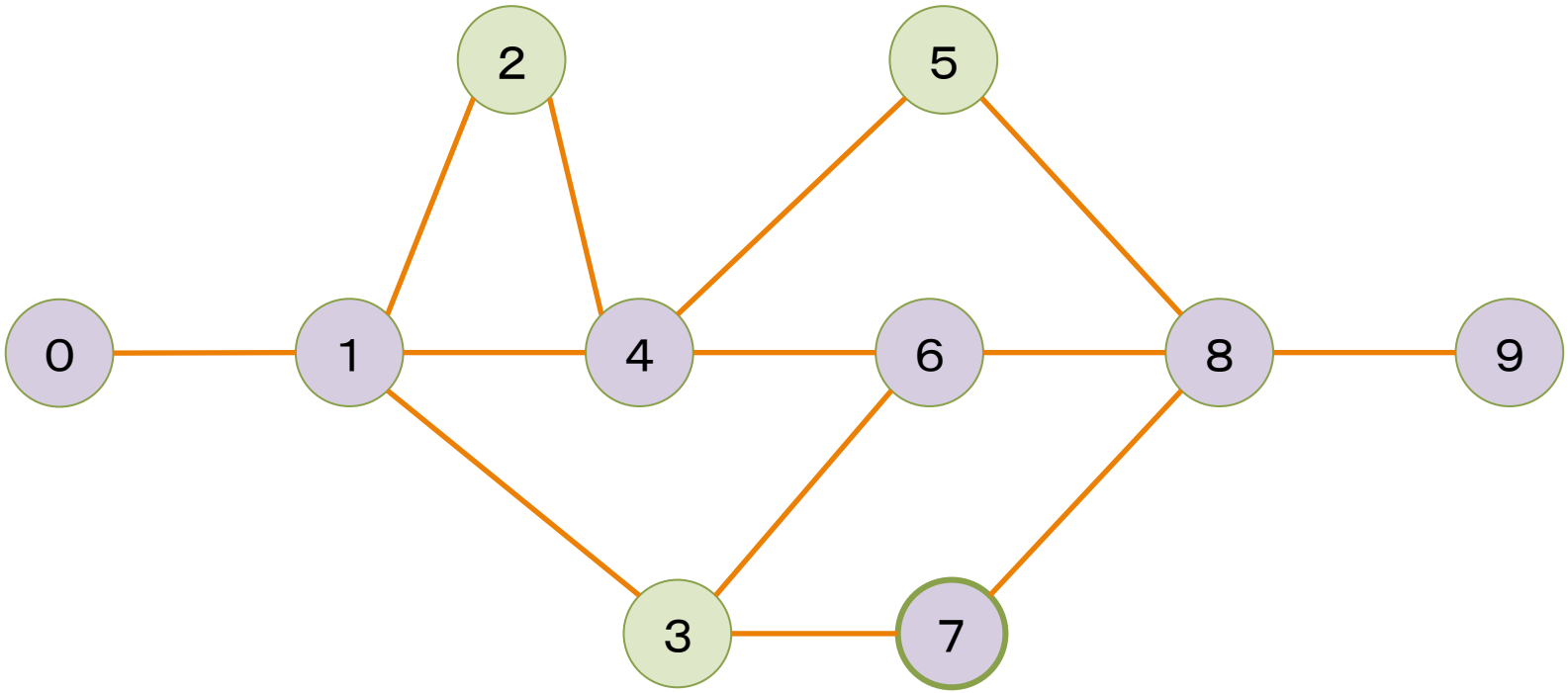
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	0	1	1

DFS step 6.5



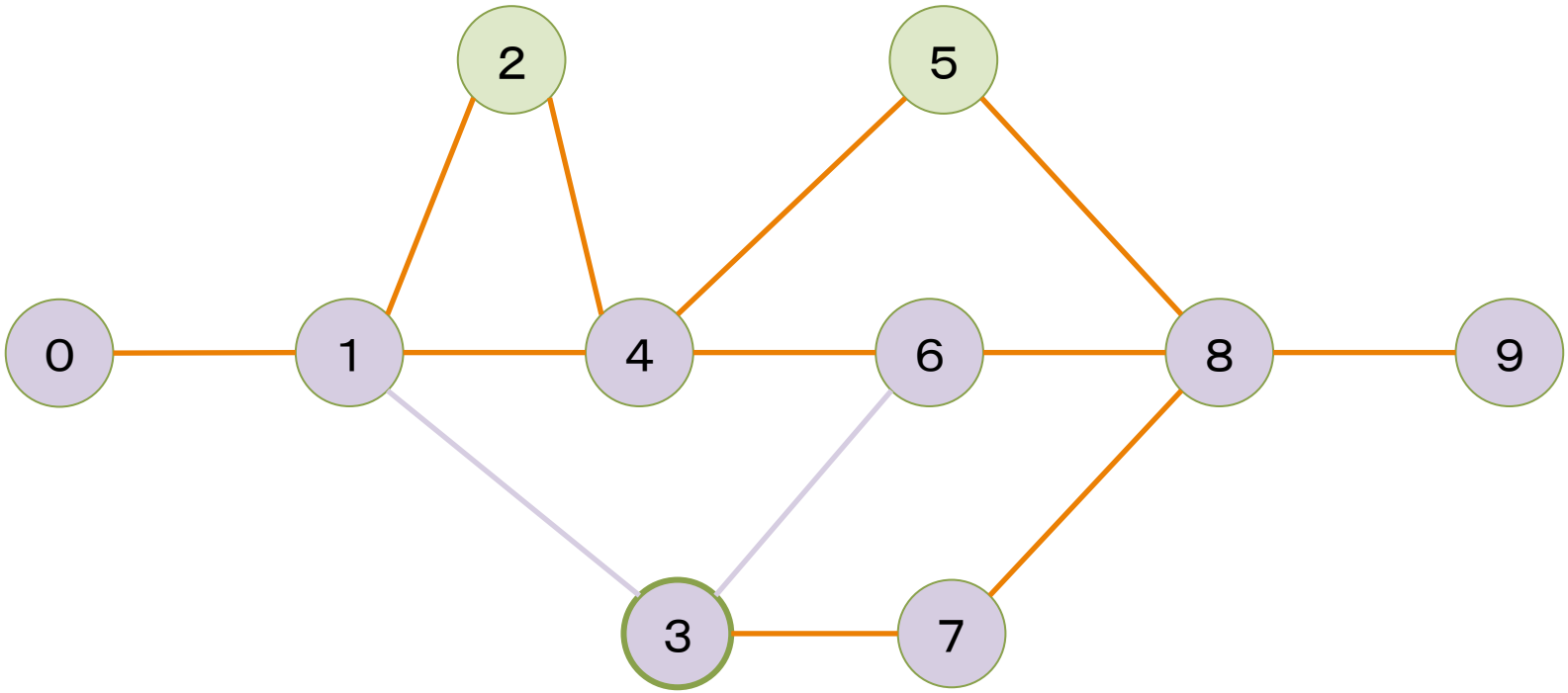
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	0	1	1

DFS step 7



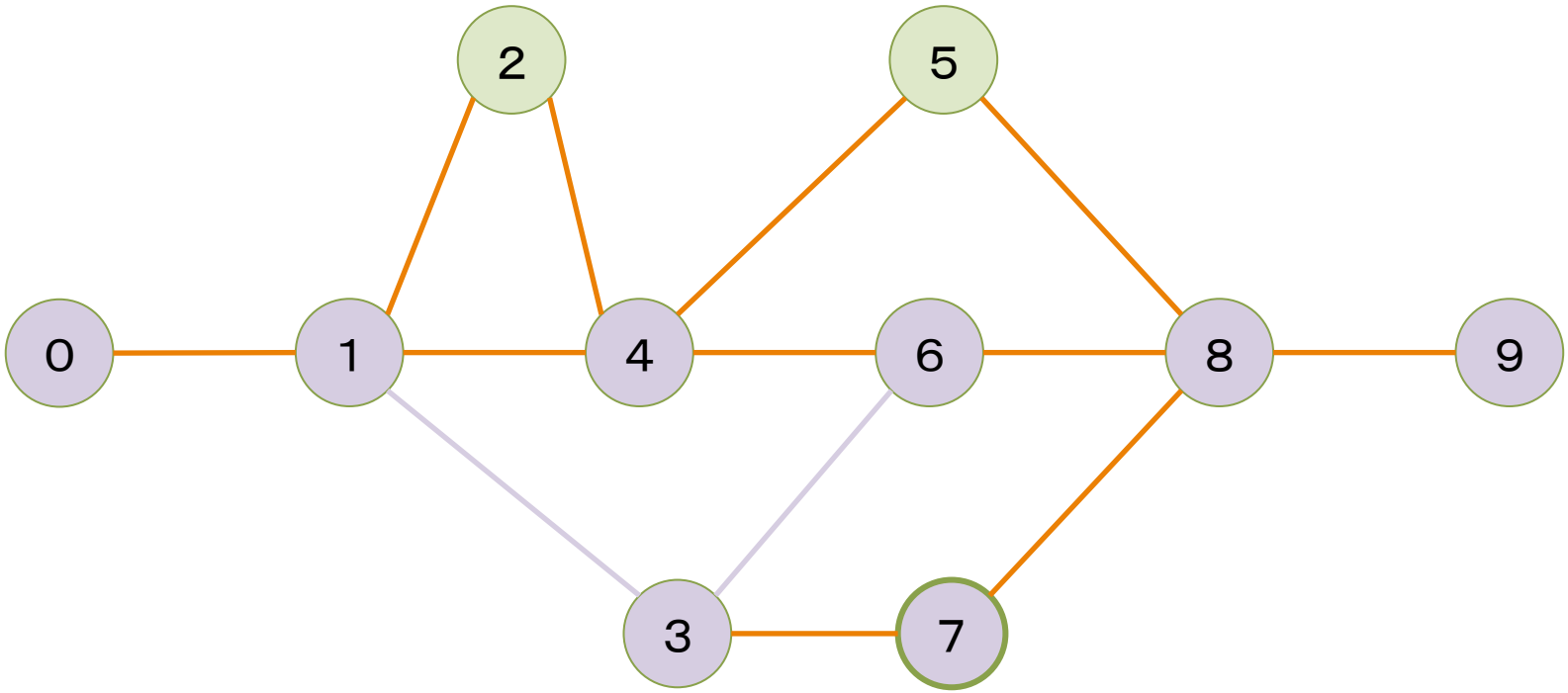
0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	1	1	1

DFS step 8



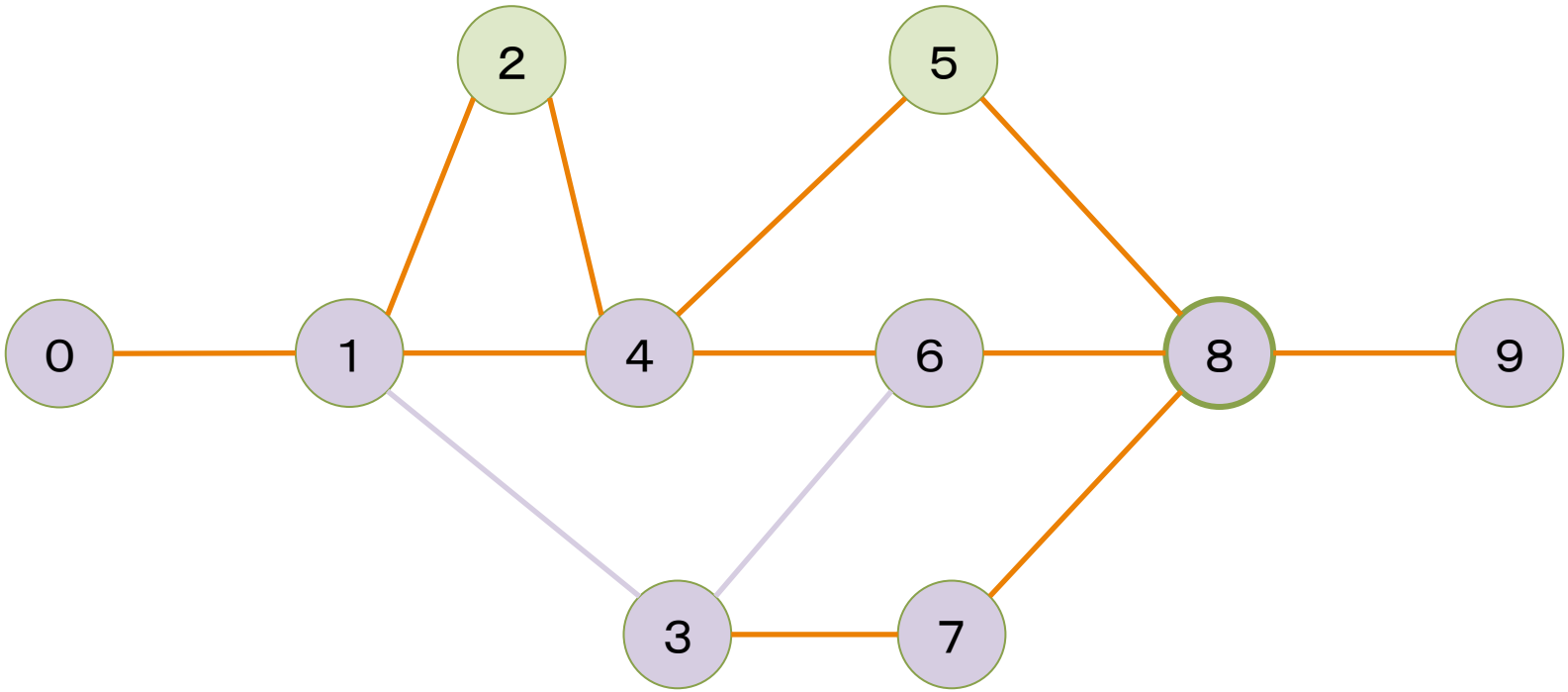
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	0	1	1	1	1

DFS step 8.33…



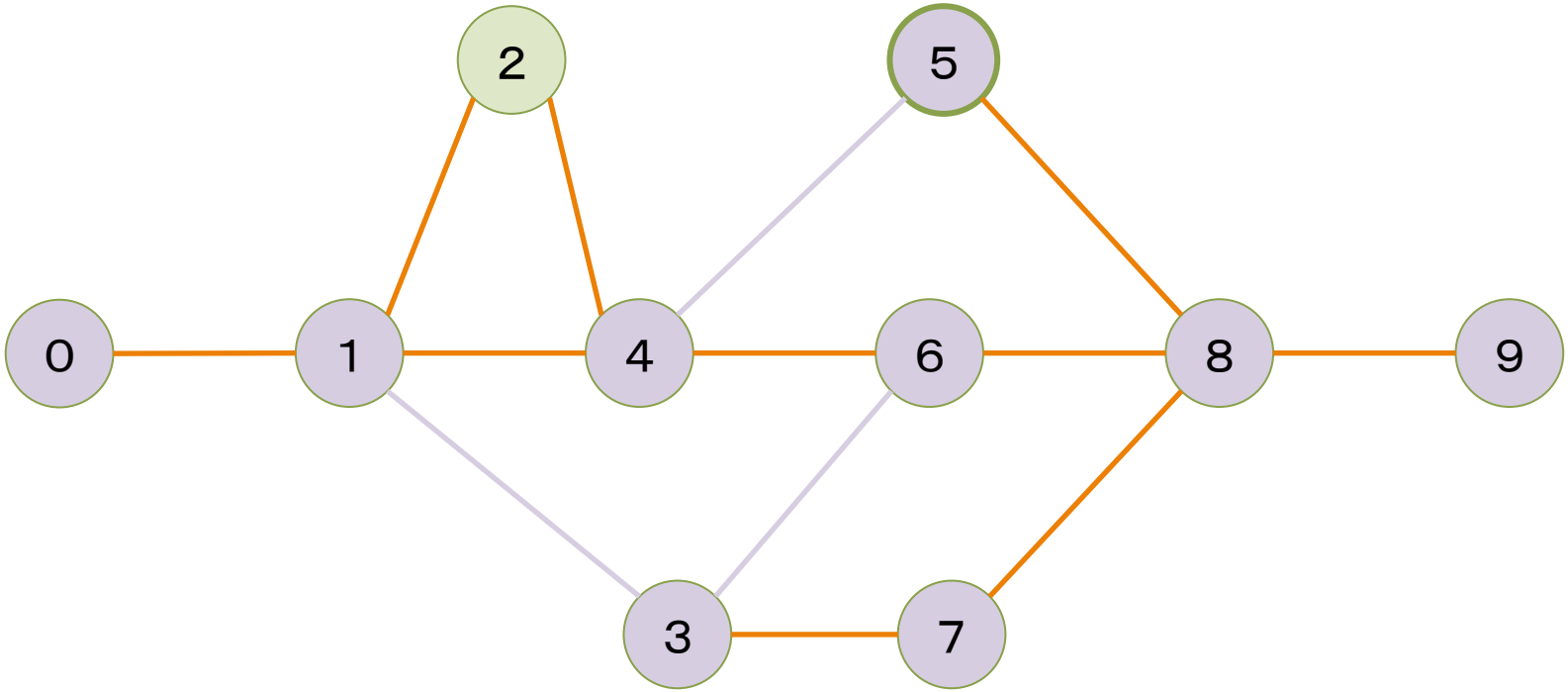
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	0	1	1	1	1

DFS step 8.66…



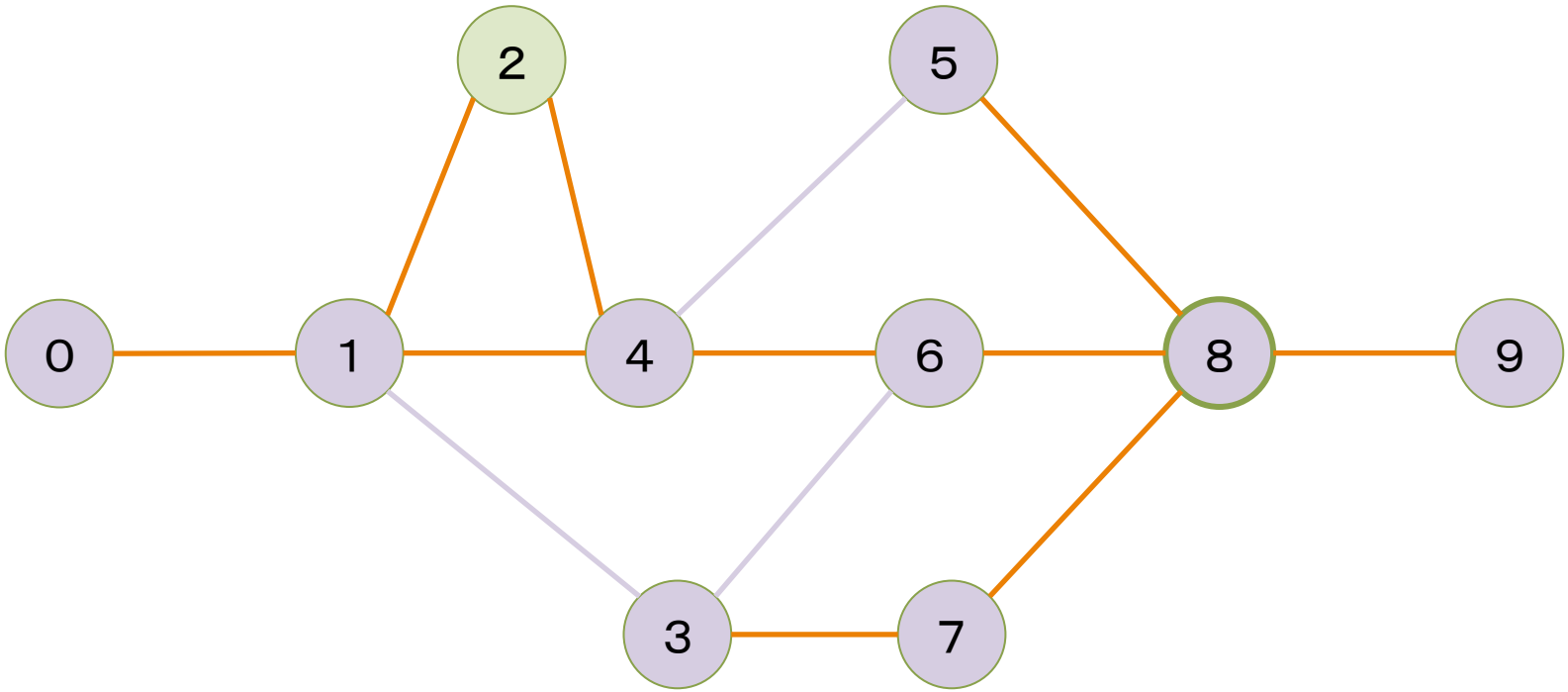
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	0	1	1	1	1

DFS step 9



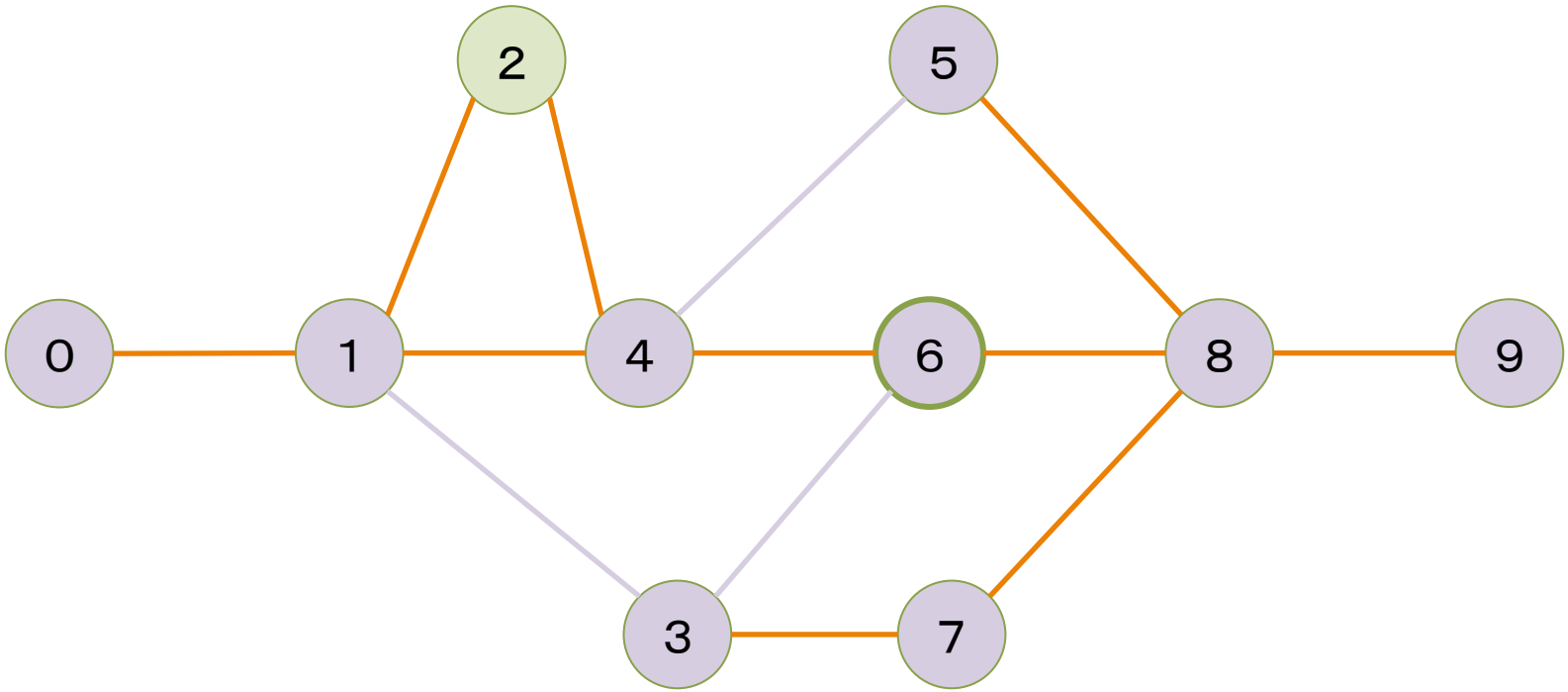
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	1	1	1	1	1

DFS step 9.25



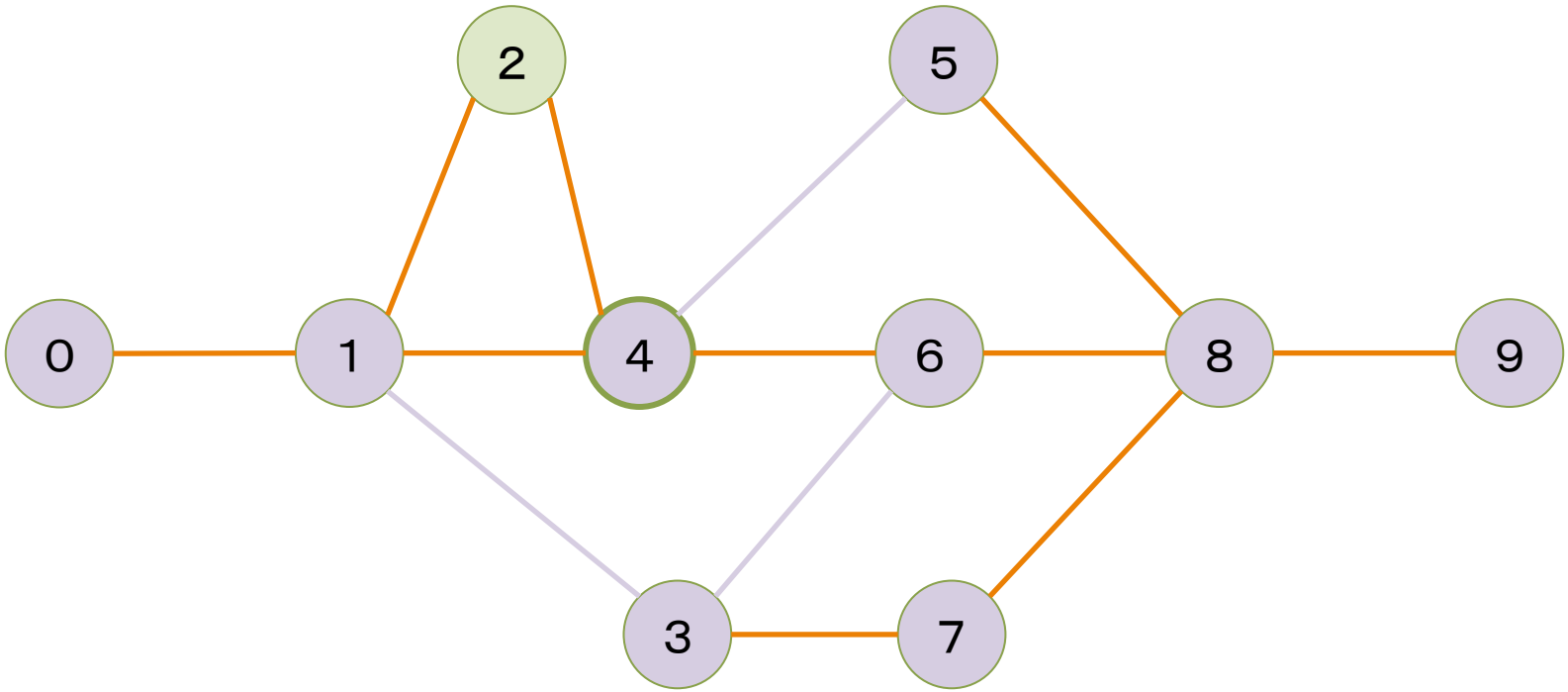
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	1	1	1	1	1

DFS step 9.5



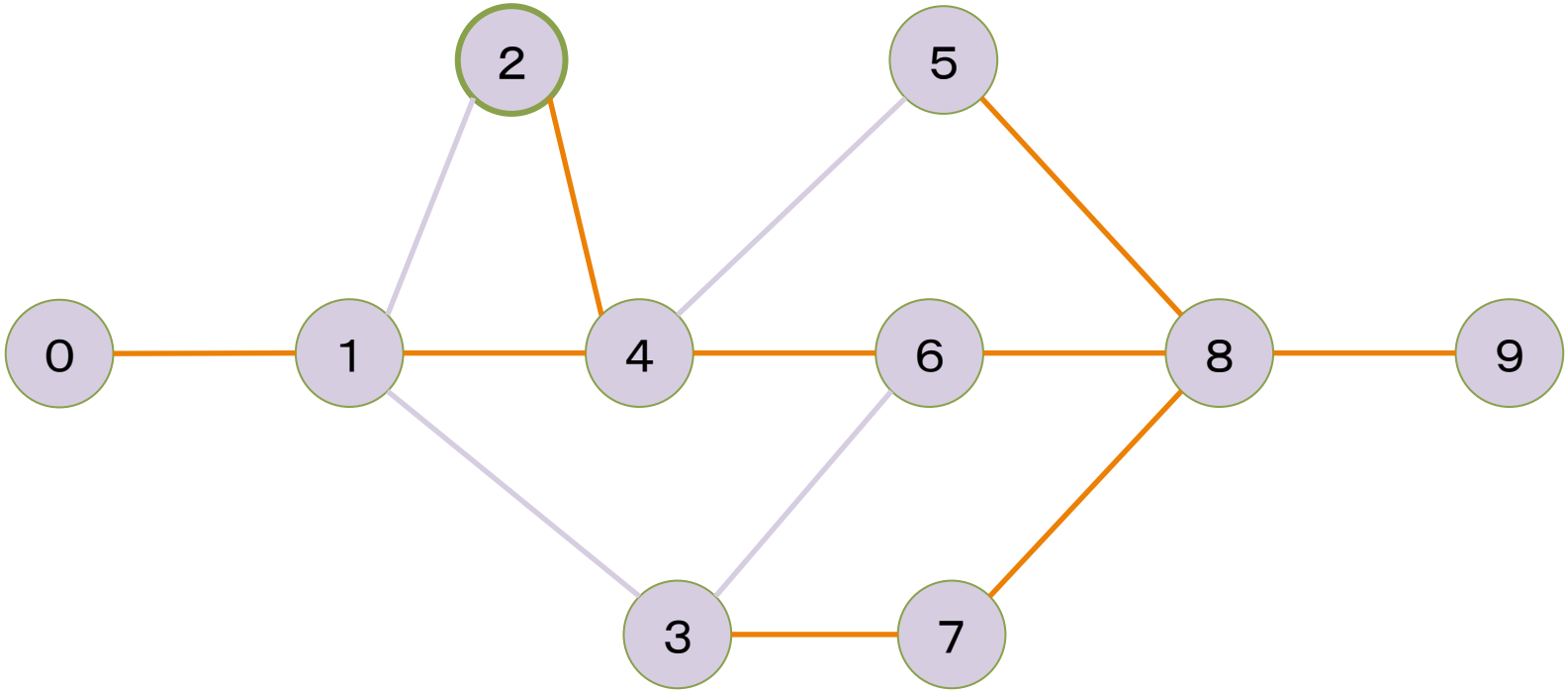
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	1	1	1	1	1

DFS step 9.75



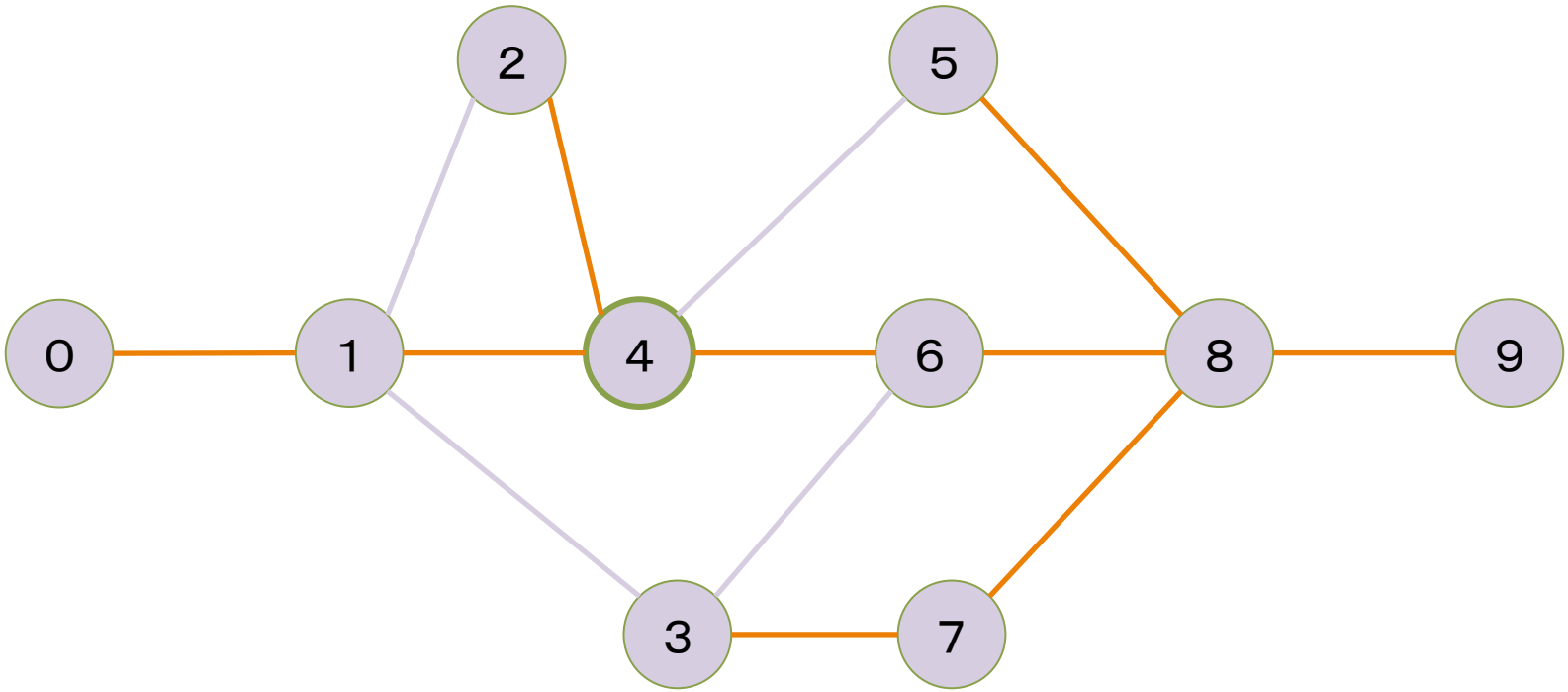
0	1	2	3	4	5	6	7	8	9
1	1	0	1	1	1	1	1	1	1

DFS step 10



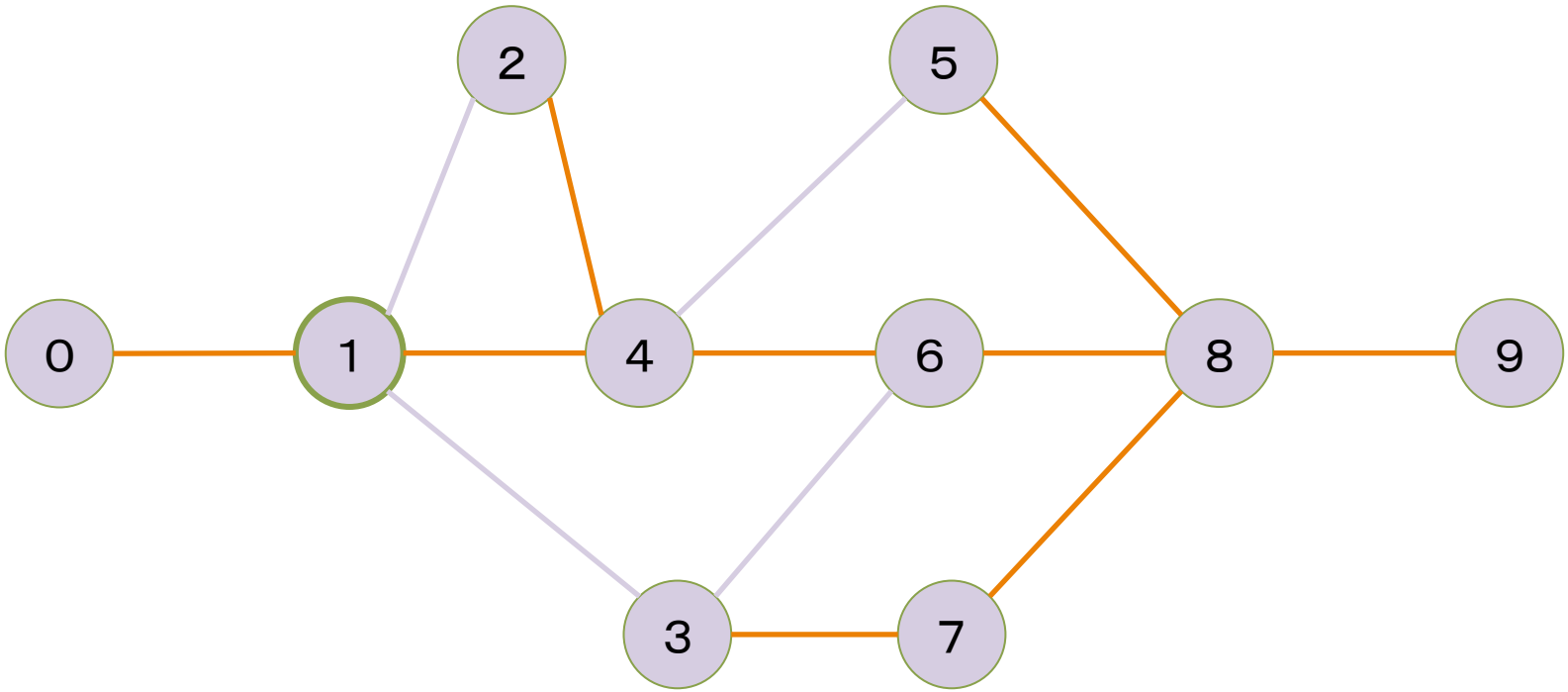
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

DFS step 10.33…



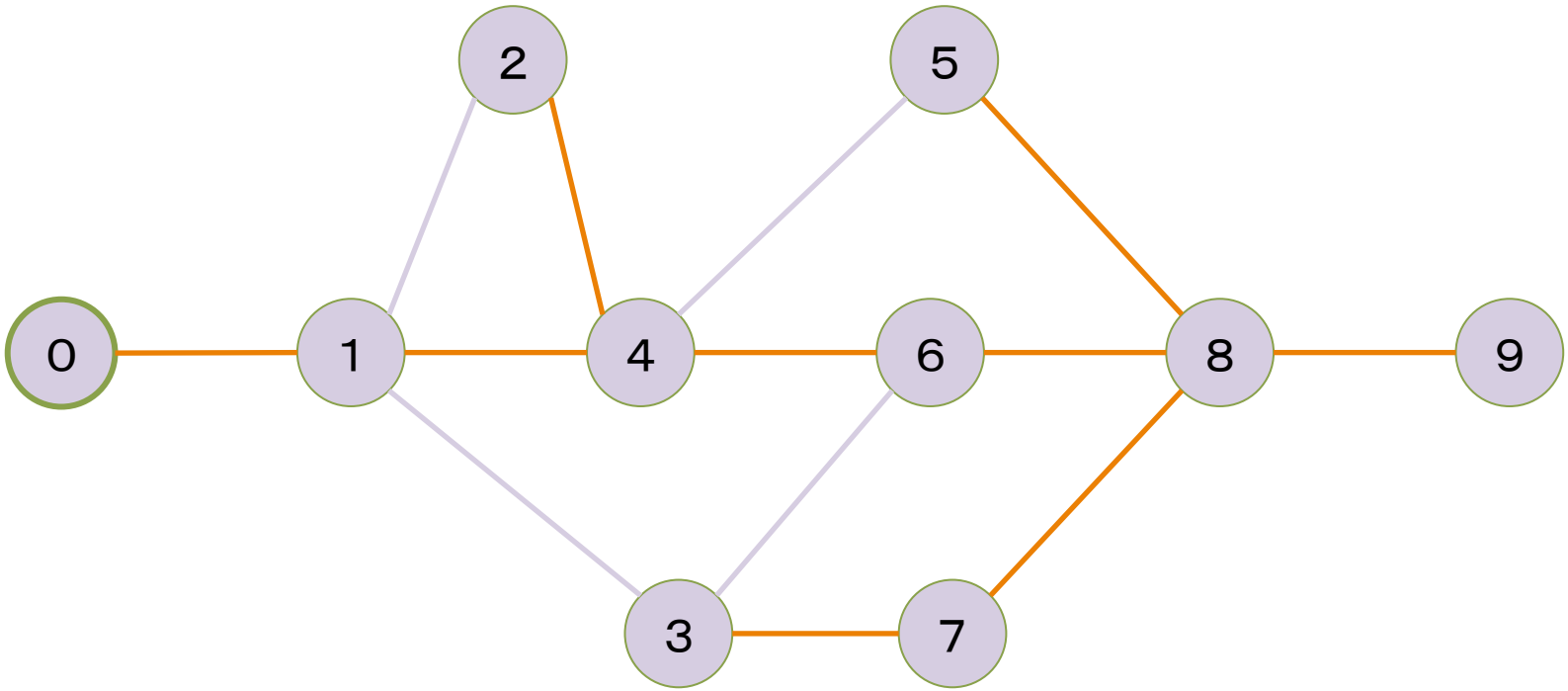
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

DFS step 10.66…



0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

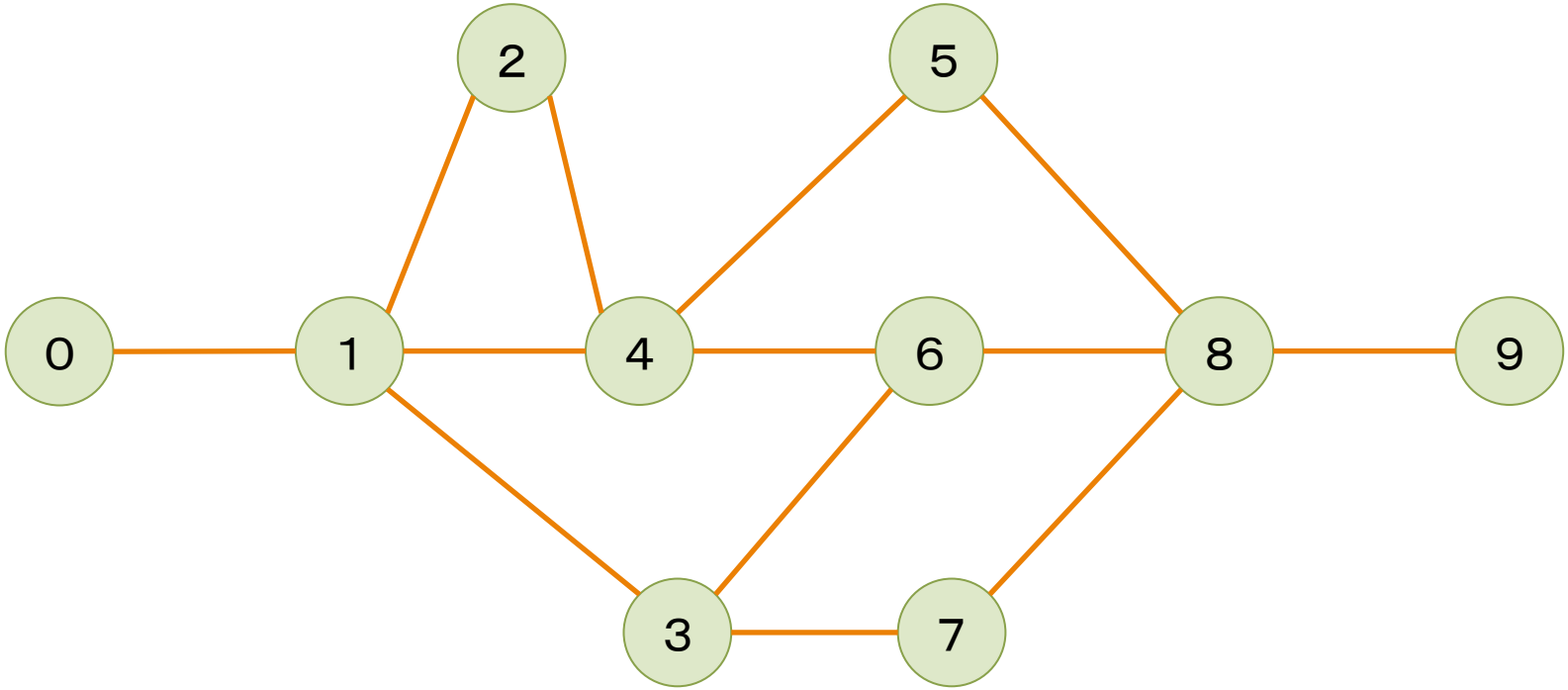
DFS step 11



0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

BFS

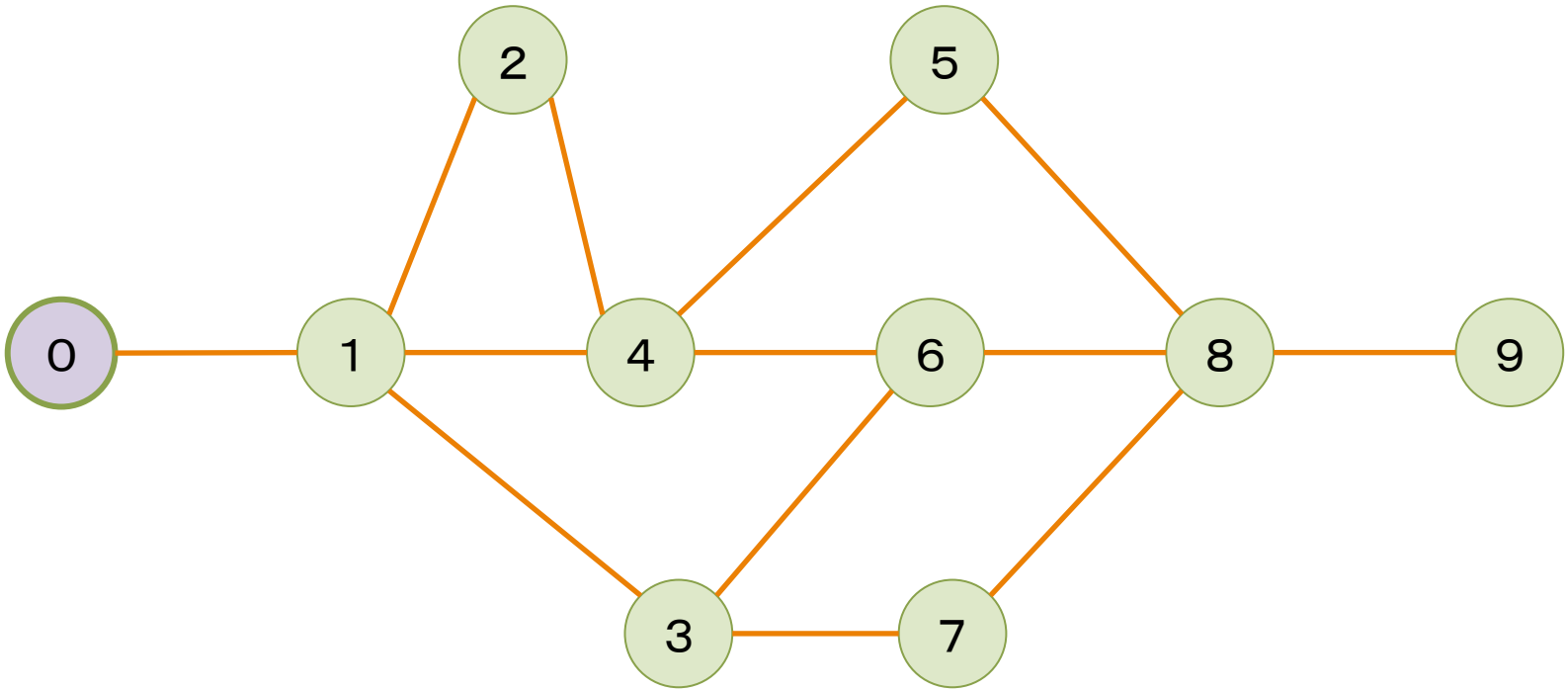
探索開始頂点を0とする：次は0



全ての頂点を未訪問としておく

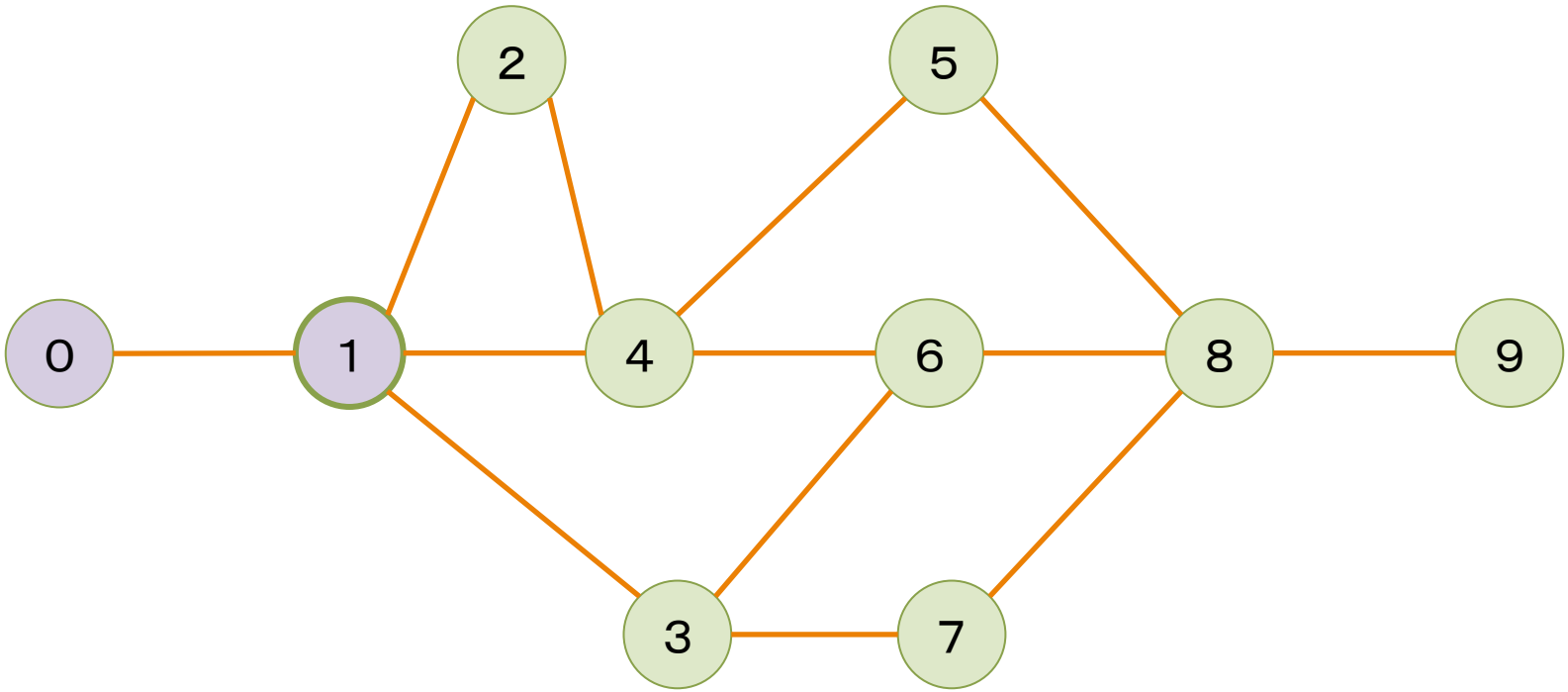
0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

BFS step 1



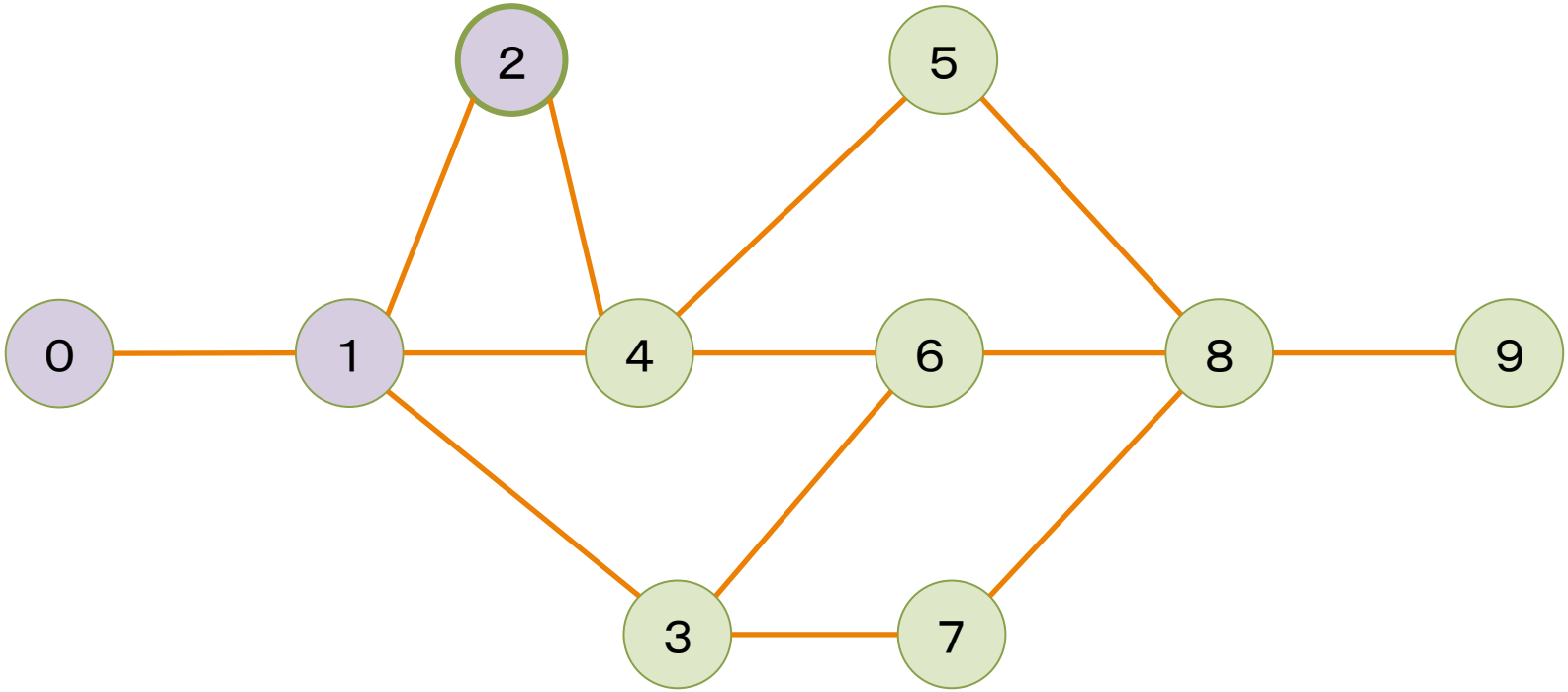
0	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0

BFS step 2



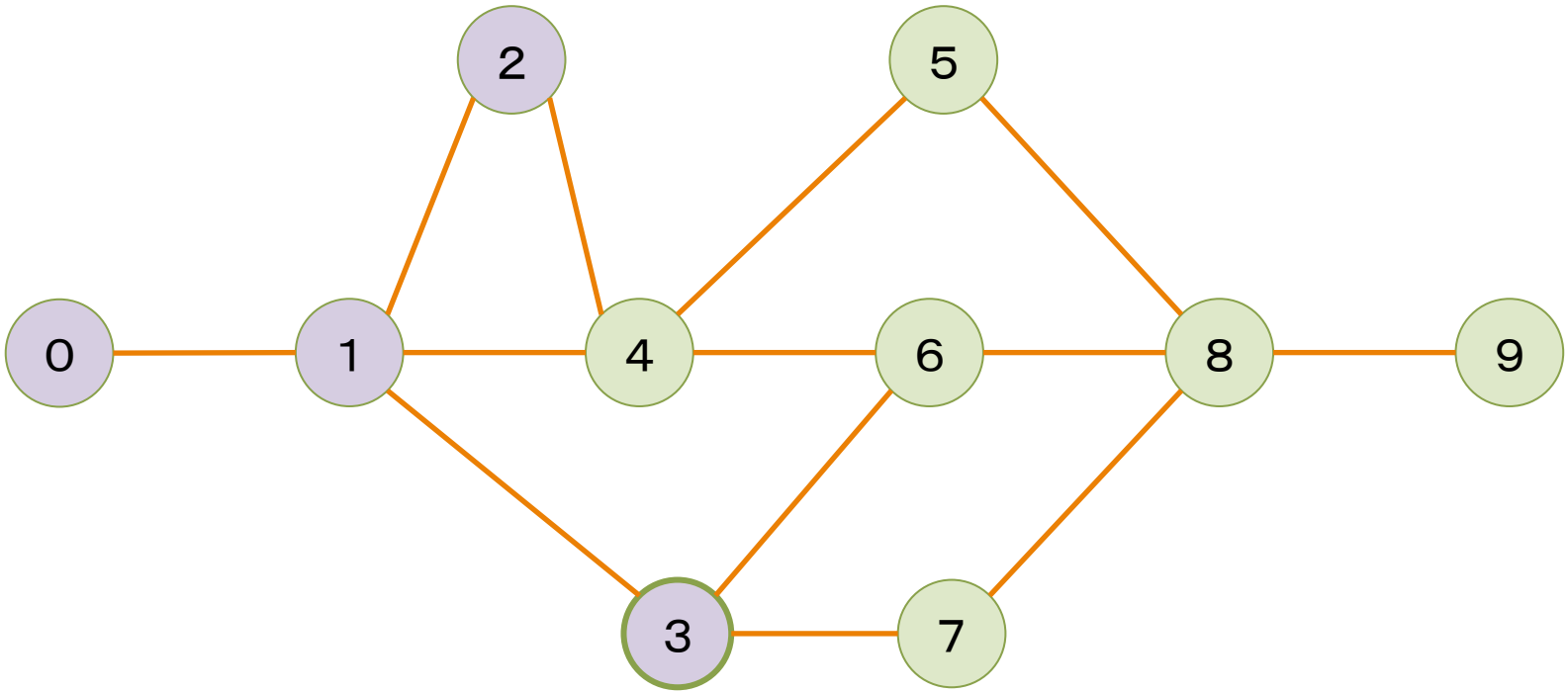
0	1	2	3	4	5	6	7	8	9
1	1	0	0	0	0	0	0	0	0

BFS step 3



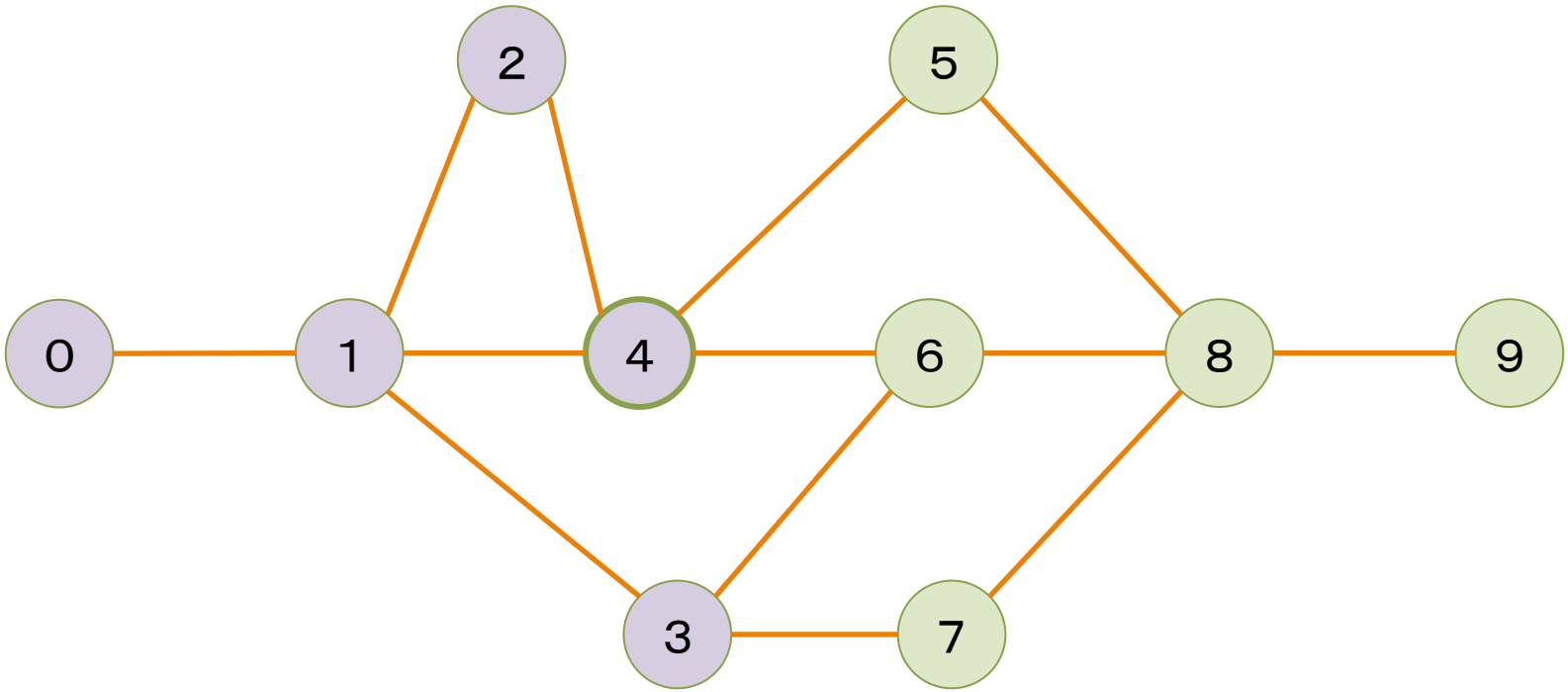
0	1	2	3	4	5	6	7	8	9
1	1	1	0	0	0	0	0	0	0

BFS step 4



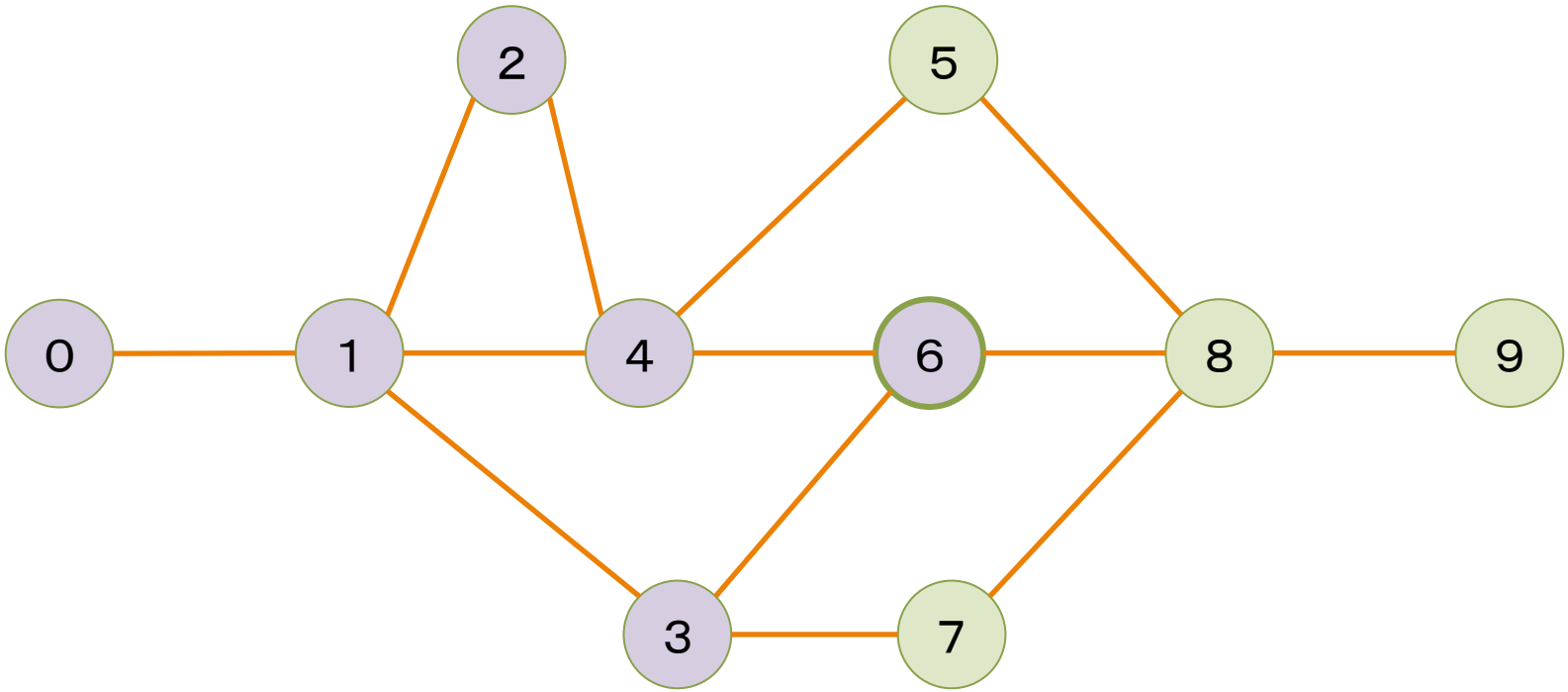
0	1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	0	0

BFS step 5



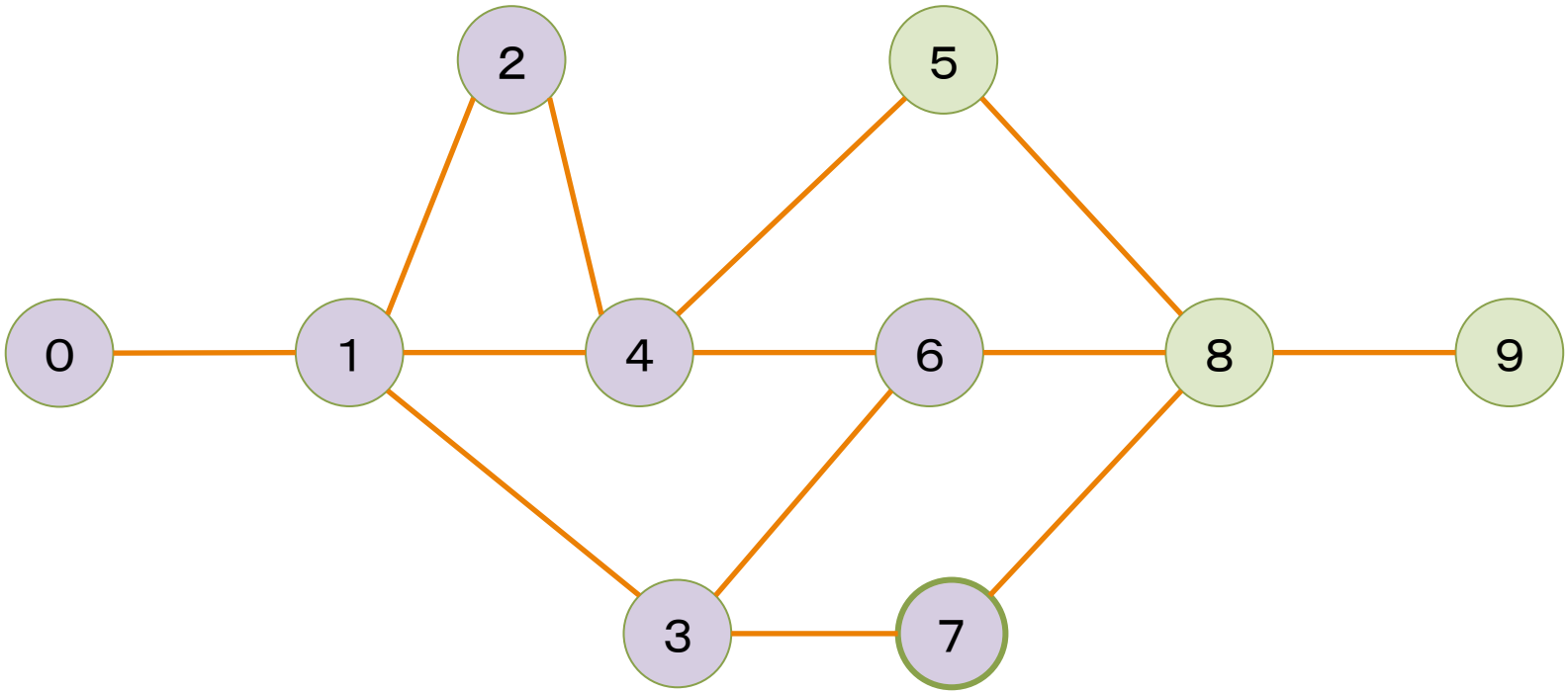
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	0	0	0	0

BFS step 6



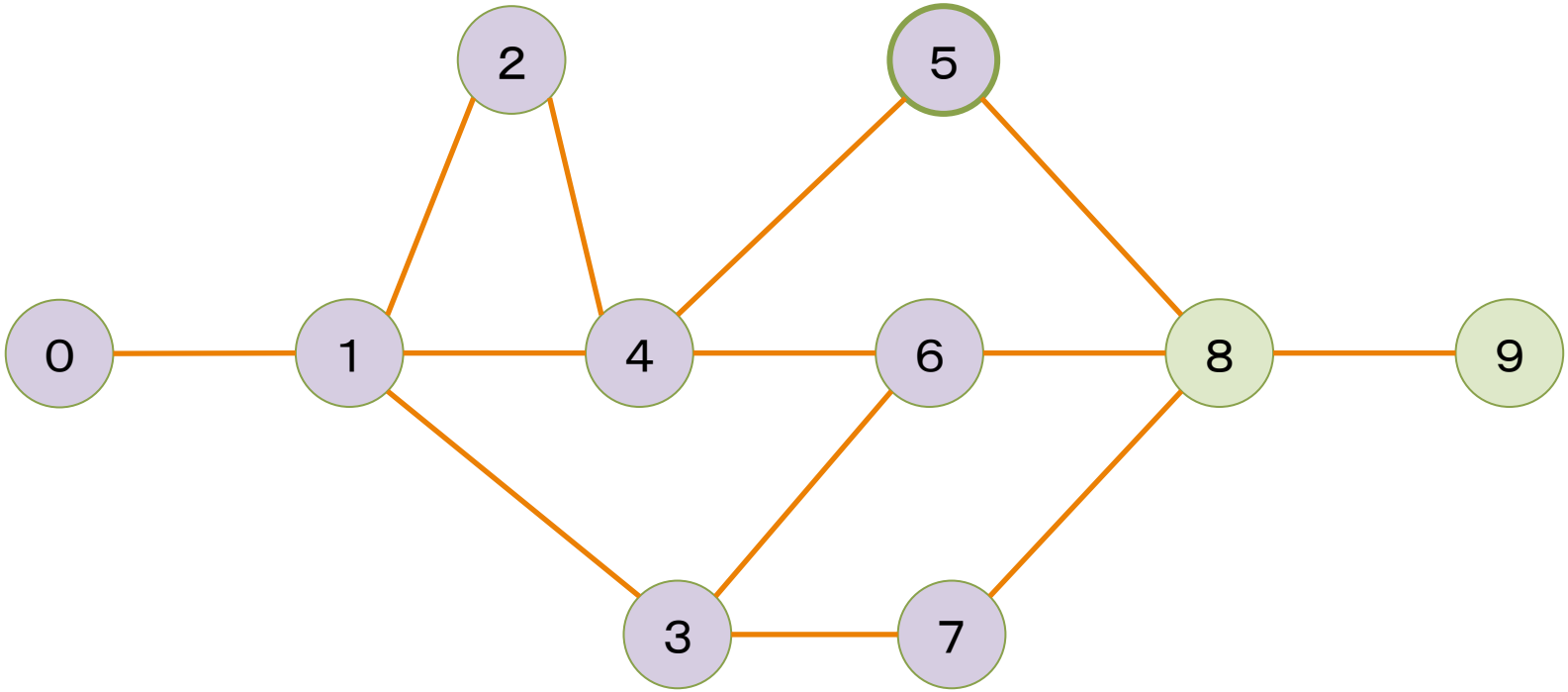
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	1	0	0	0

BFS step 7



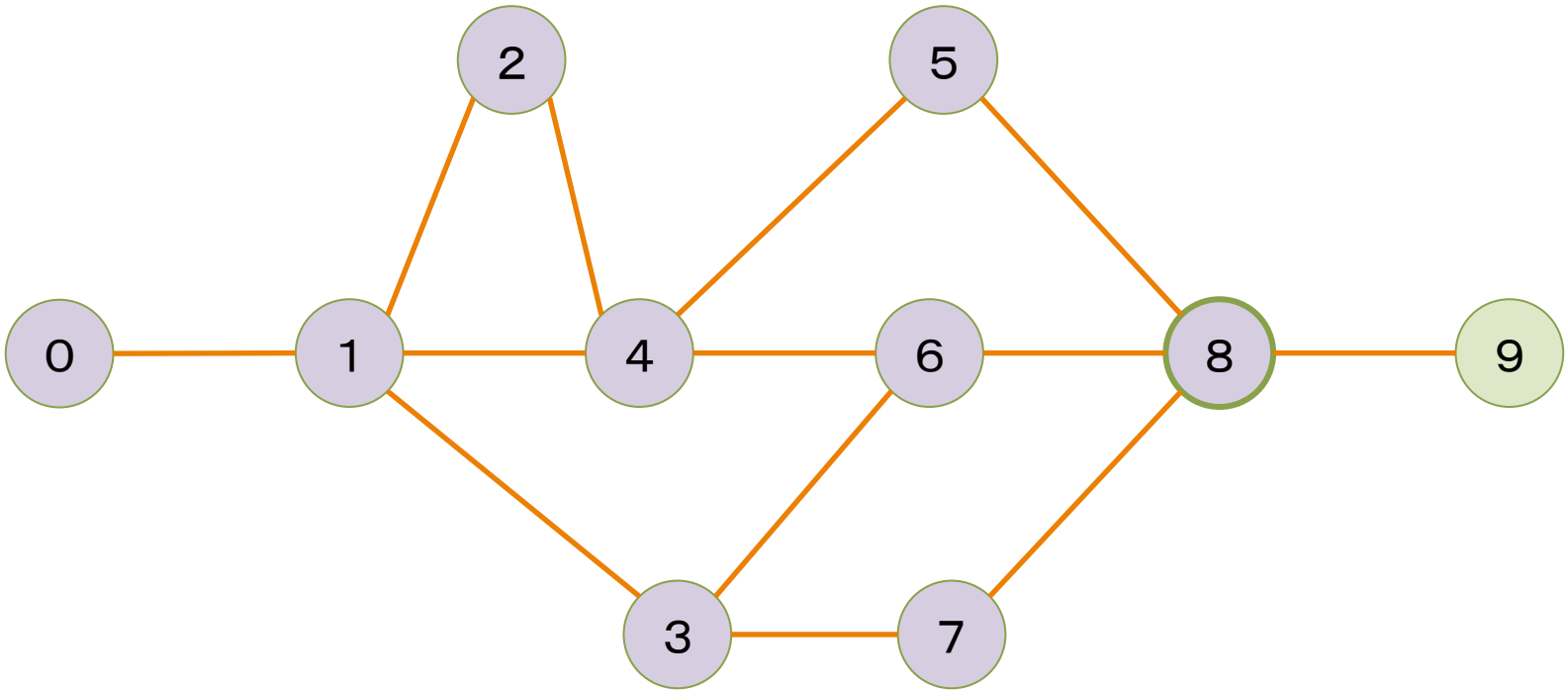
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	1	1	0	0

BFS step 8



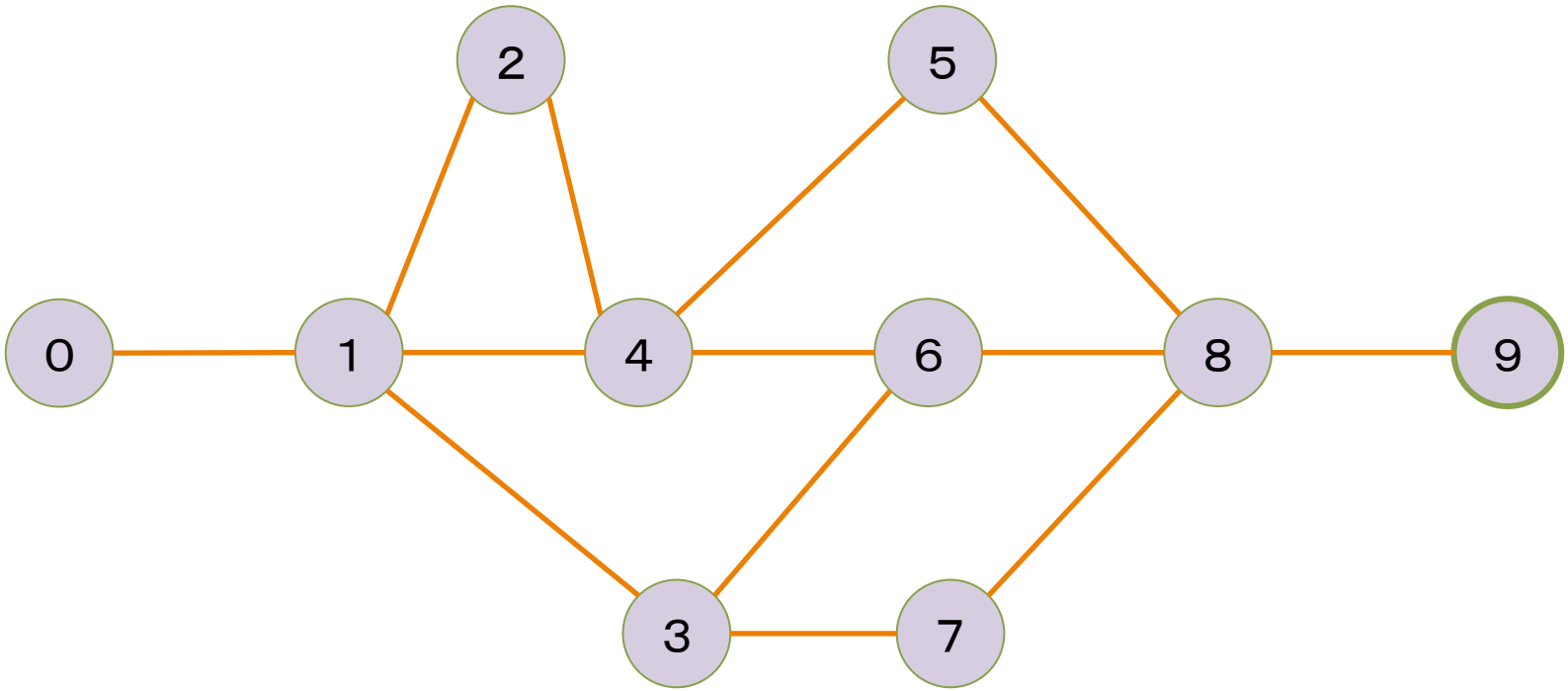
0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	0	0

BFS step 9



0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	0

BFS step 10



0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1

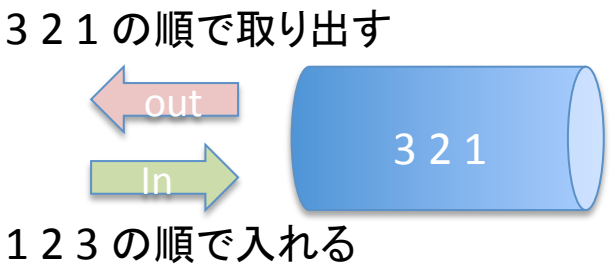
練習問題

スタックとキュー

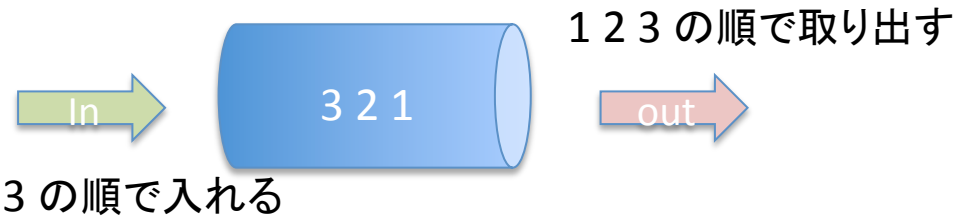
- グラフの探索にスタックとキューを使うことを考える
  1. 訪問する節をスタック（キュー）から取り出す
  2. この節を訪問済みにする
     - 訪問済みを示す行列またはリストに記録
  3. 取り出した節に隣接する節をスタック（キュー）に入れる
     - 複数の節がある場合は隣接行列（リスト）に記載の順に入れる
  - 以上をスタック（キュー）が空になるまで繰り返す
- 注意：
  - スタックとキューへのアクセスはそれぞれの入り口へのデータの追加と出口からの取り出しに限られ，それ以外はデータ残っているかどうかしかわからない

確認

- スタック (stack)
  - LIFO: Last In Last Out

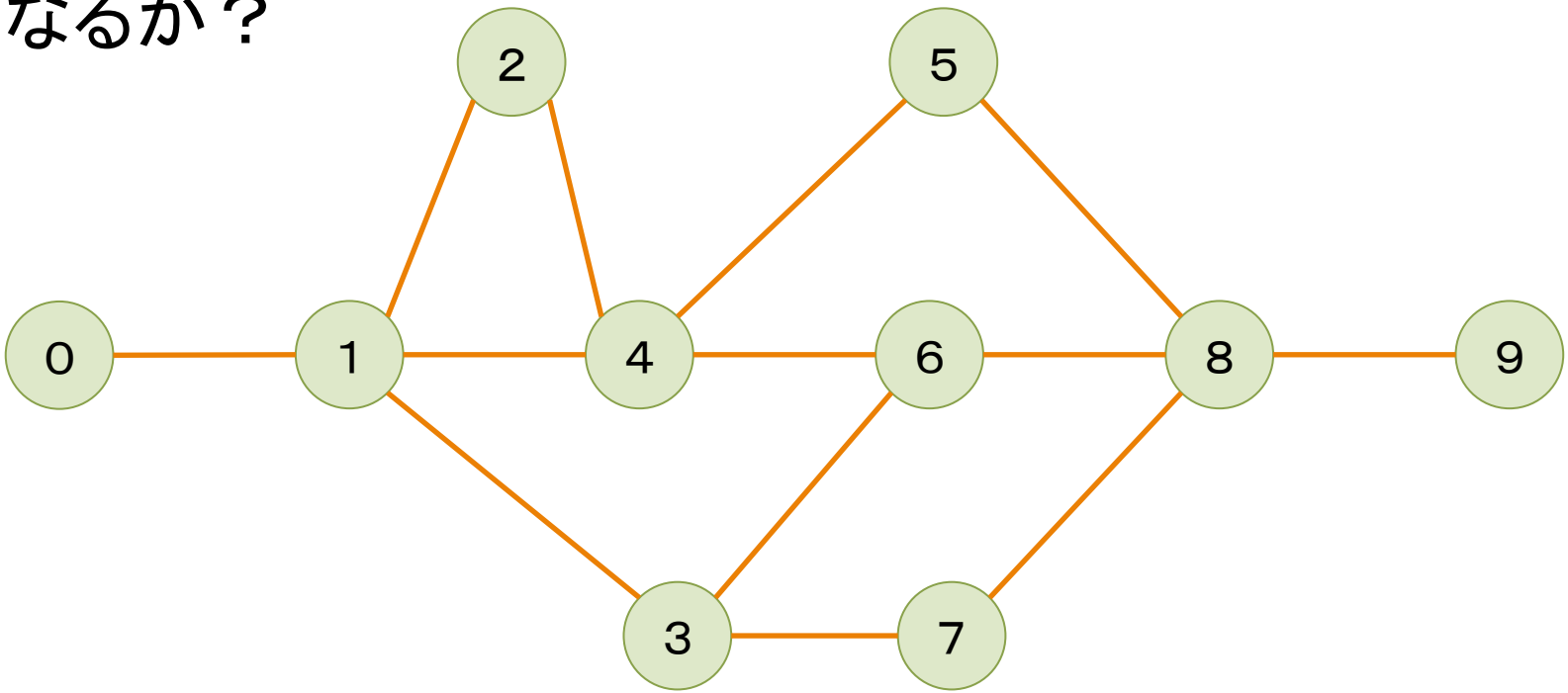


- キュー (queue)
  - FIFO: First In First Out

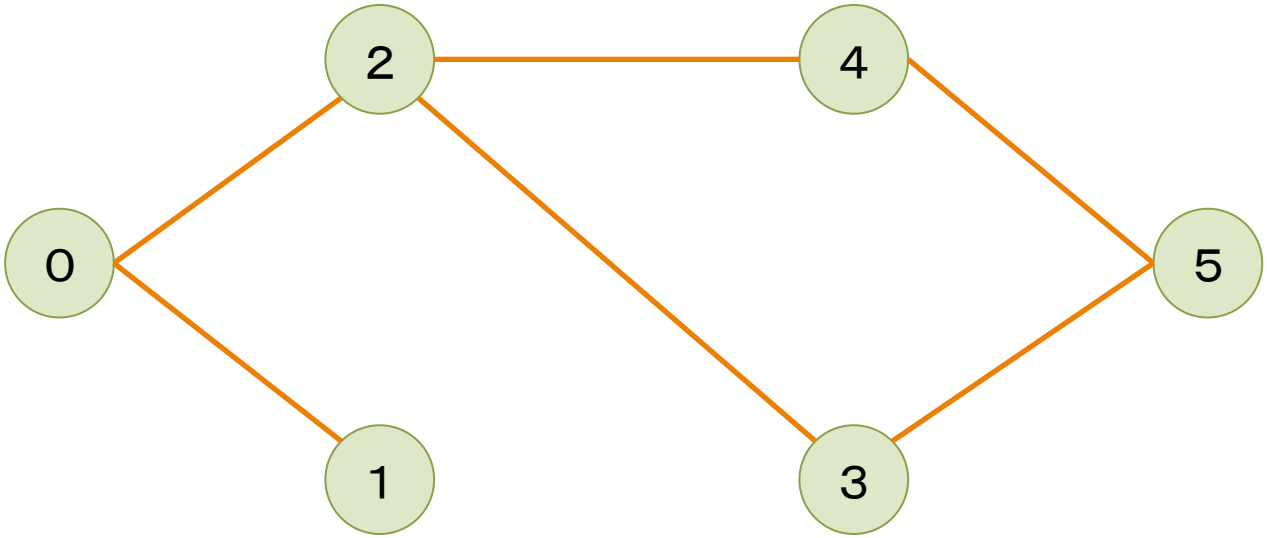


問題

- スタック（キュー）の初期状態として唯一0が入っているとして探索で訪問する節の順番はそれぞれどうなるか？



隣接行列と隣接リストを記せ

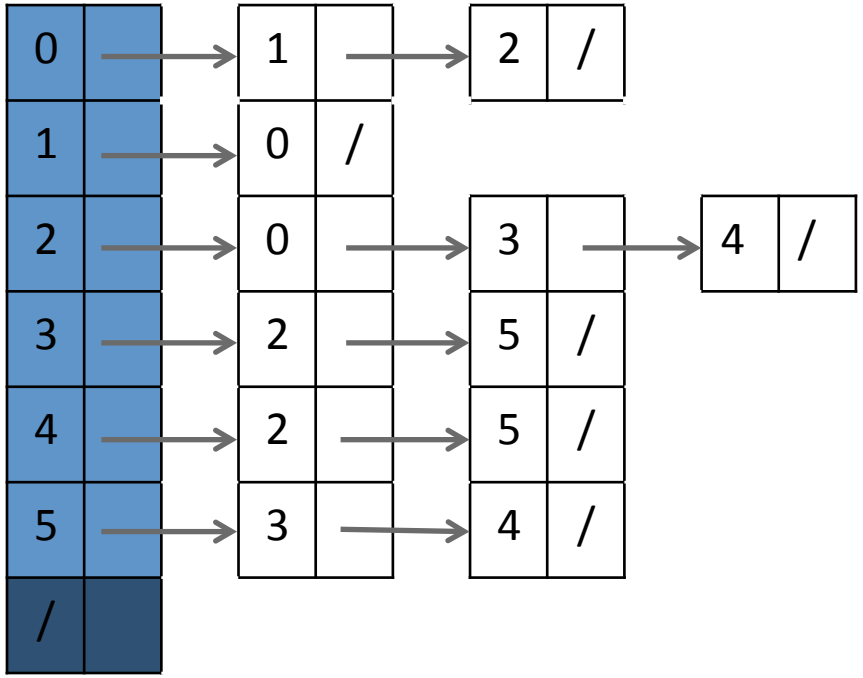


解答

隣接行列

	0	1	2	3	4	5
0	0	1	1			
1	1	0				
2	1		0	1	1	
3			1	0		1
4			1		0	1
5				1	1	0

隣接リスト

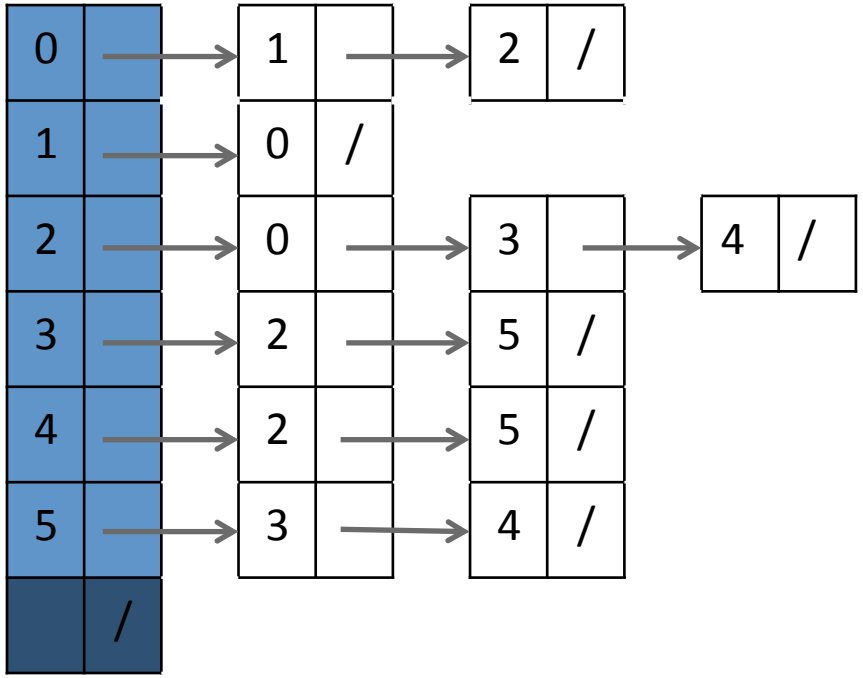


隣接{行列またはリスト}を使い DFSを実行せよ

隣接行列

	0	1	2	3	4	5
0	0	1	1			
1	1	0				
2	1		0	1	1	
3			1	0		1
4			1		0	1
5				1	1	0

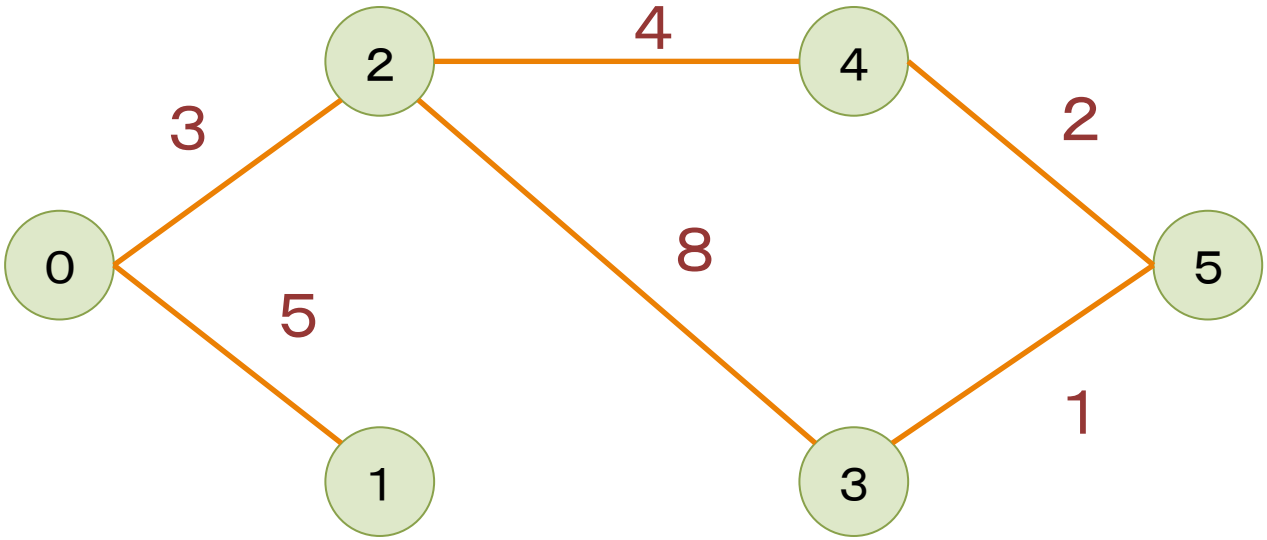
隣接リスト



重みつきグラフ

辺に重みと呼ぶ数値を対応させる

- 距離
- 時間
- コスト



グラフ上の問題

- 最小全域木
- トポロジカルソート
- 最短経路
  - 特定の2頂点間の最短経路
  - 全ての2頂点間の最短経路
- 最短閉路
  - 最短巡回路（巡回セールスマン問題）
  - ハミルトン閉路問題

課題

- C言語で実装したポインターを使った線形リストから，i番目の要素をリストから削除する関数を作成してください
  - 引数を使い、線形リストのi番目の要素を削除する。
    - iが先頭や末尾の要素を指すときの動作に注意。
  - iが線形リストの長さより大きいときの振る舞いも設計すること
  - 削除した要素はfree()関数を使いメモリを解放すること

void free(ポインタ変数)

- #include <stdlib.h> が必要
- 変数として使用しているメモリを解放
  - もう使用しないので（他で）自由に使っても良いことにする
- 引数に解放したいメモリ領域を指すポインタを指定する
- 使用には注意が必要
  - 使わなくなったメモリを解放しないと使えないメモリ領域が増大
  - 使っているメモリ領域を解放してしまうと，どのような動作をし出すか分からない

補足

- 参考プログラムで使用している clock() の精度は 10ms 程度です.
- 時間計測の精度を確保するために，処理時間は 100ms以上になるよう要素数を決定してください.
- 提出物は以下の内容を含むプログラムファイル
  - ソースコード
  - コメントとして:
    - 氏名，学科，学年，クラス，番号
    - 実行結果
    - 感想

提出についての注意

- プログラム
  - 提出はプログラムのソースファイル(.c)のみ
  - higuchi_fumito_ex09 のように氏名と宿題番号にすること
  - higuchi_fumito_c5_ex09 (同姓同名はクラスを付加）
- プログラムの冒頭に氏名、学年、クラス、番号等をコメントとして記入
  - 日本語の文字コードはutf-8が望ましい
  - 参考にした資料の他、簡単な感想も付け加えてください
- Oh-o!Meijiから提出（次回の授業開始までに）

ポインタ変数を使って…

- 独自のデータ構造を実現することが可能
  - 線形リスト，木構造，グラフ構造
  - 動的にデータ構造が変化する状況でメリット大
  - 例えば「疎行列」
    - 大規模な行列だがその殆どの要素が0
      - 1000,000 x 1000,000 といった大きさ
    - 実現方法はいくつかあるが…
      - 線形または木構造に行番号・列番号・データを持たせる
      - ハッシュテーブルでキーに行番号・列番号の組合せを使う
- 理解や操作は面倒
  - 最近のプログラミング言語ではポインタを隠蔽
    - Processing の IntList, FloatList, ArrayList 等
      - IntList の Method を参照のこと

連絡先

樋口文人

wenren@meiji.ac.jp

free()関数の説明

やってみよう：作業

- 探索で1つのノードを訪れたとき、そのノードから辿ることのできるノードを記憶するのにスタックを使うときとキューを使うときの動作の違いを調べてください。
  - それらはDFSですか？
  - またはBFSですか？
  - それとも全く別の動作になりますか？