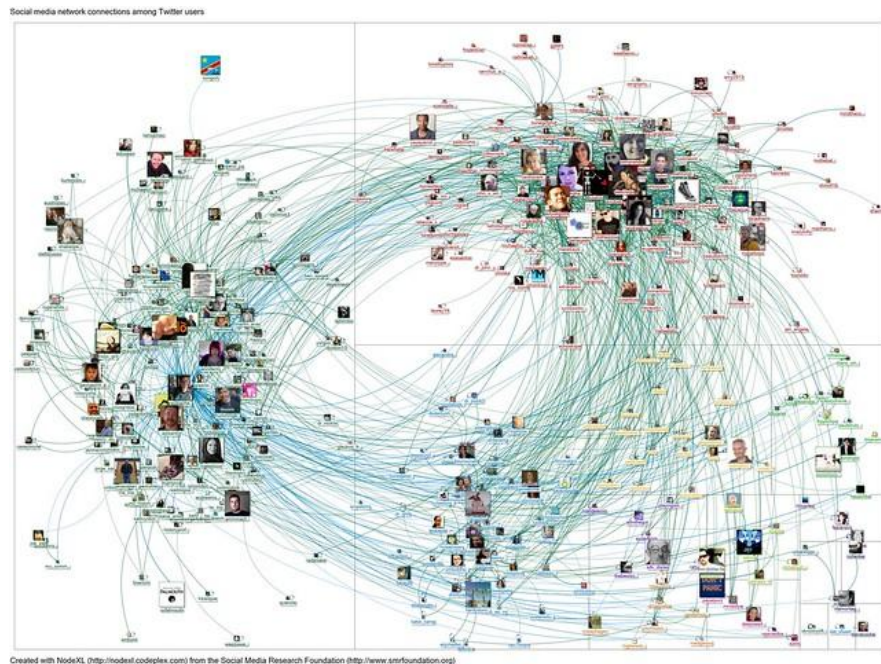


12章 グラフ

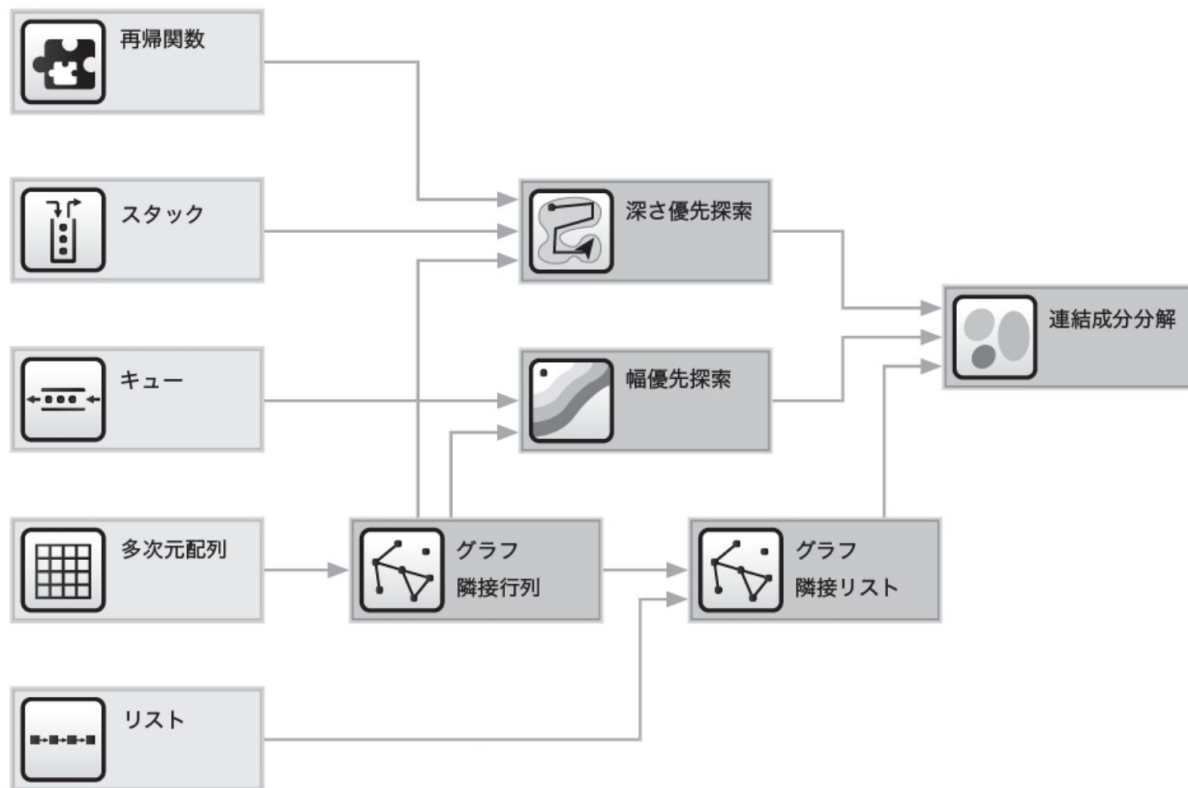
sean

グラフ

モノとそれらの関係を表すデータ構造



必要なスキル，章の内容



目次

- 12.1 グラフ：問題をチャレンジする前に
- 12.2 グラフの表現
- 12.3 深さ優先探索
- 12.4 幅優先探索
- 12.5 連結成分

目次

12.1 グラフ：問題をチャレンジする前に

12.2 グラフの表現

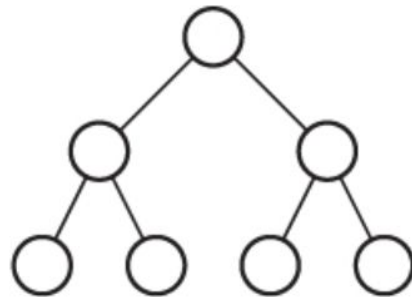
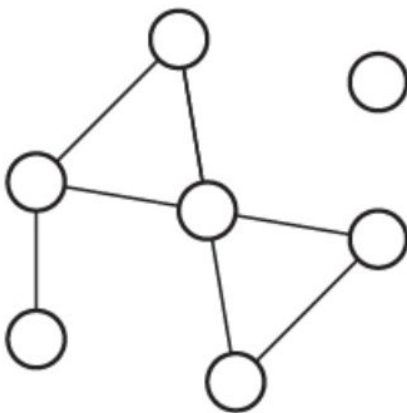
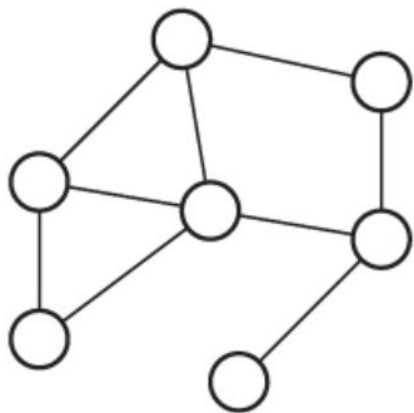
12.3 深さ優先探索

12.4 幅優先探索

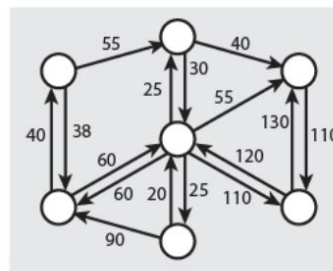
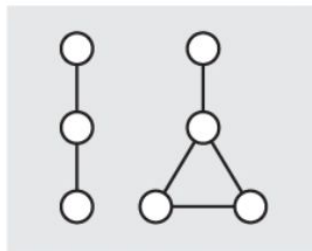
12.5 連結成分

グラフの要素

頂点 (vertex, node) と辺 (edge)を含む

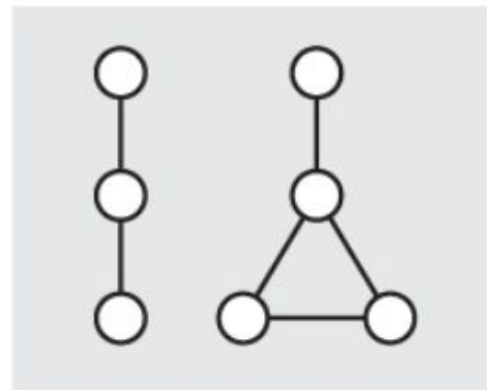
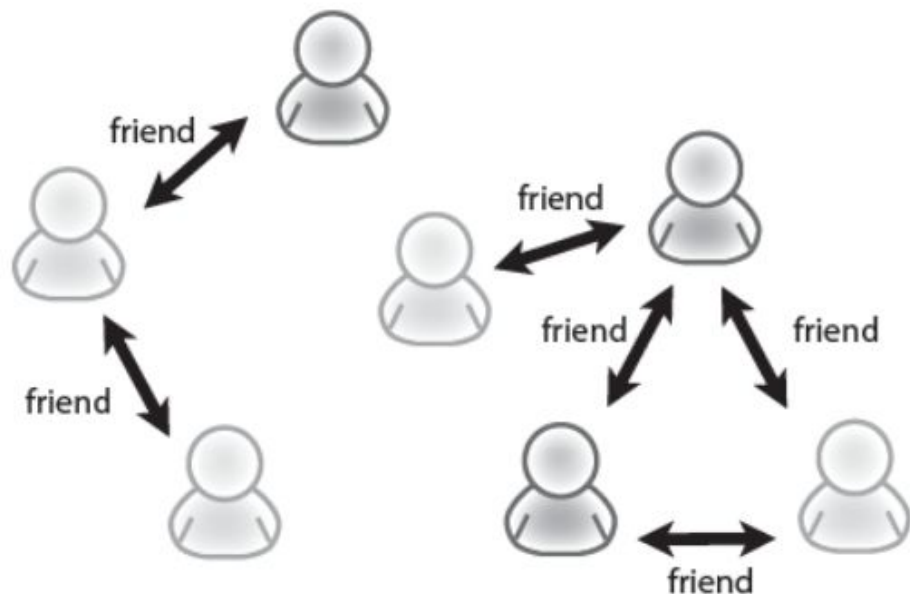


グラフの種類



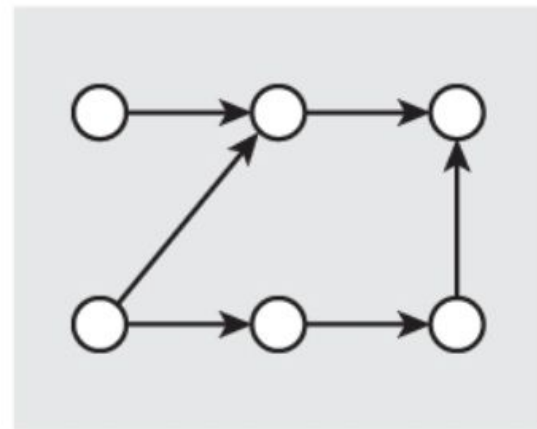
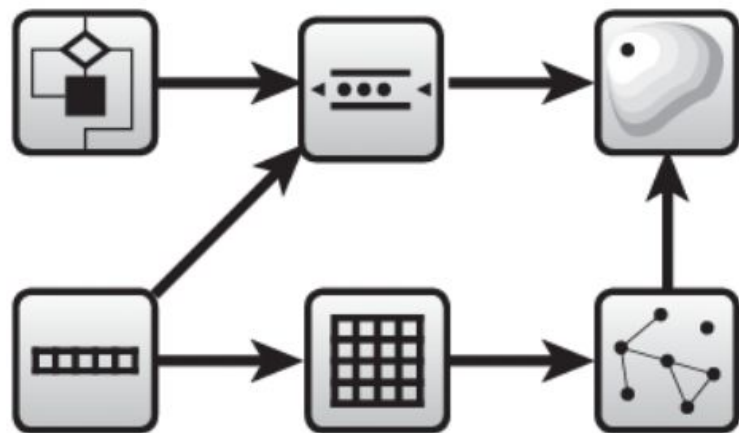
名前	特徴
無向グラフ	エッジに方向がないグラフ
有向グラフ	エッジに方向があるグラフ
重み付き無向グラフ	エッジに重み（値）があり、方向がないグラフ
重み付き有向グラフ	エッジに重み（値）があり、方向があるグラフ

無向グラフの例



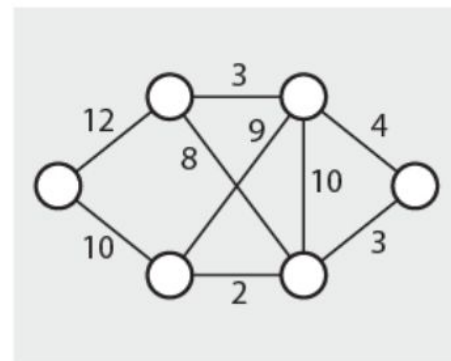
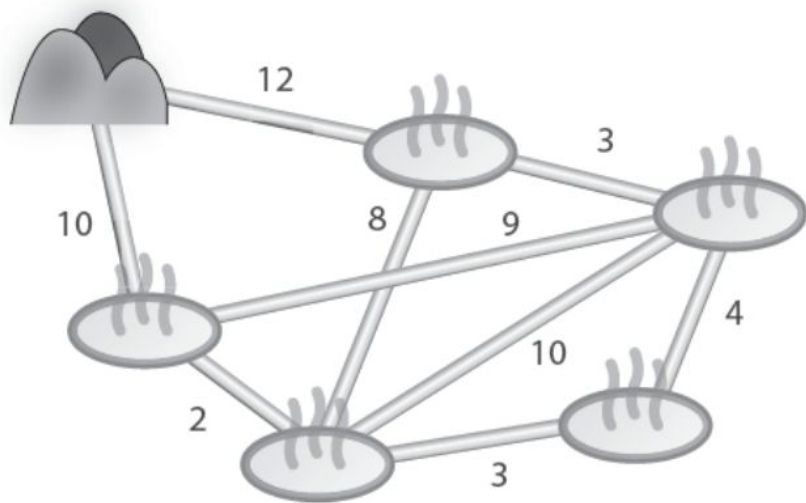
問題例：A君からC君まで辿り着くまで何人経由するか？

有向グラフの例



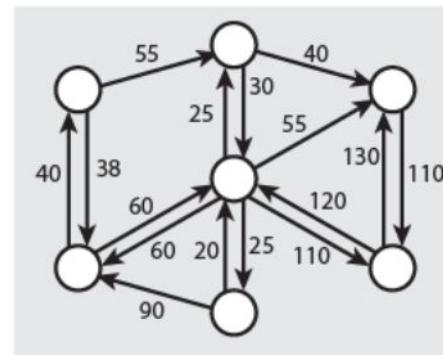
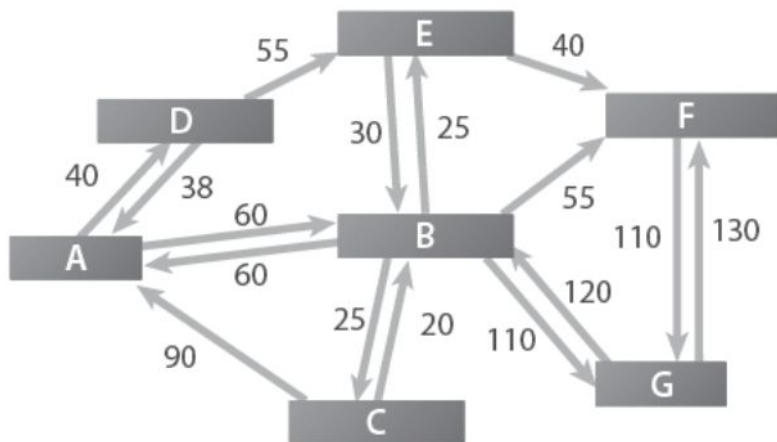
問題例：全てを獲得するにはどの順でやればいいか？

重み付き無向グラフ



問題例：全てを繋げるパイプの最短設定は何か？

重み付き有向グラフ



問題例：AからGの最短経路は？

表記と用語

ノードの集合 V , その総数 $|V|$

エッジの集合 E , その総数 $|E|$

グラフ $G=(V, E)$

2つのノードを結ぶエッジ $e=(u, v)$, その重みは $w(u, v)$

無向の場合 $(u, v)=(v, u)$

無向グラフでの**隣接** (adjacent)と**パス** (ノードの列で表す)

始点と終点が同じパスは**閉路** (cycle)とよぶ

表記と用語

サイクルのない有向グラフ：**DAG**

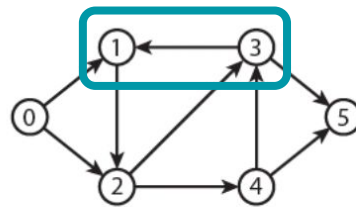
次数 (degree), 有向グラフにおける**入次数**と**出次数**

連結グラフ：

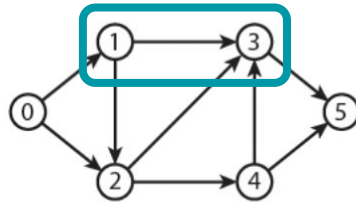
全てのノード同士の間にはパスがある場合

部分グラフ：

G の E と V の部分集合をとったグラフ



(a)



(b)

グラフ探索

グラフにおける基本的なアルゴリズム

あるノードから辿り着けるノードを全て訪問

代表例：

深さ優先探索 (depth first search, DFS)

幅優先探索 (breadth first search, BFS)

動作例

深さ優先探索

<https://www.youtube.com/watch?v=NUgMa5coCoE>

幅優先探索

<https://www.youtube.com/watch?v=x-VTfcmrLEQ>

目次

12.1 グラフ：問題をチャレンジする前に

12.2 グラフの表現

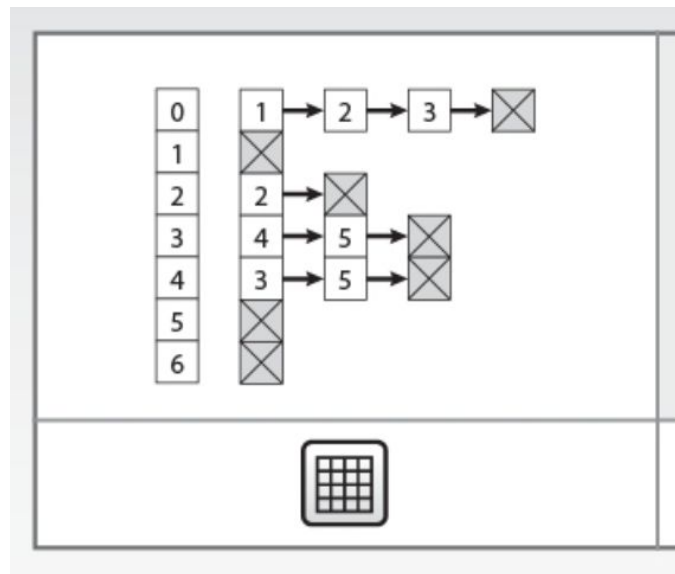
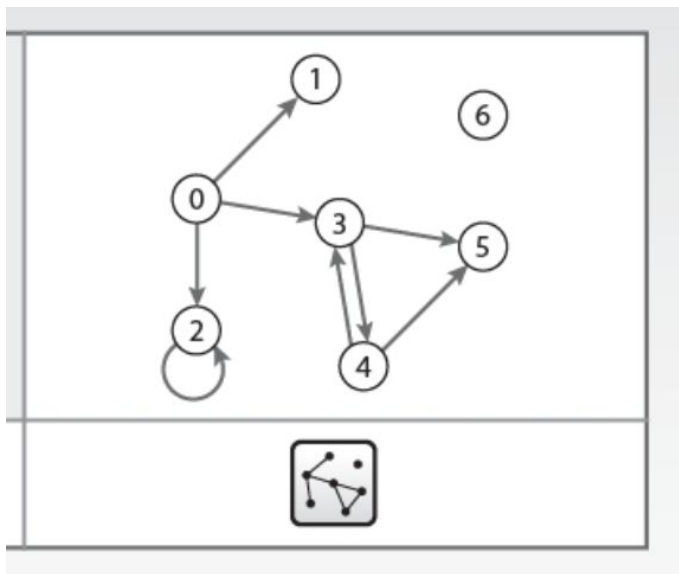
12.3 深さ優先探索

12.4 幅優先探索

12.5 連結成分

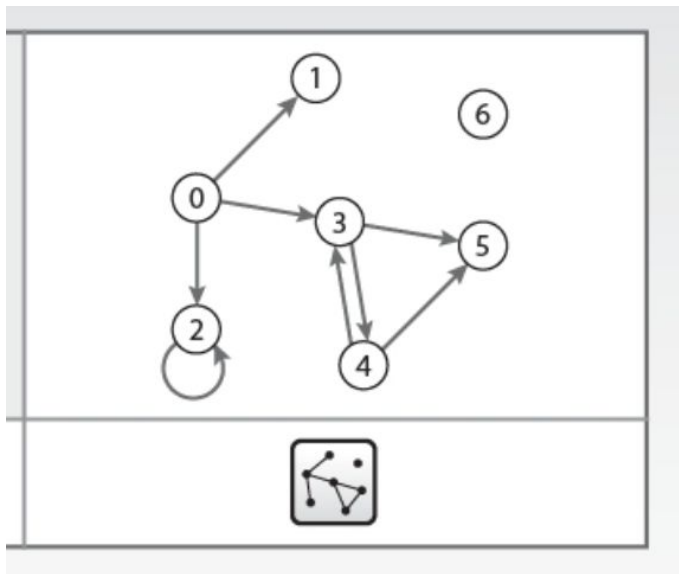
表現1：隣接リスト

$|V|$ 個のリストのリスト． u 番目のリストは $\{v_i \mid (u, v_i) \in E\}$ を含む



表現 2 : 隣接行列

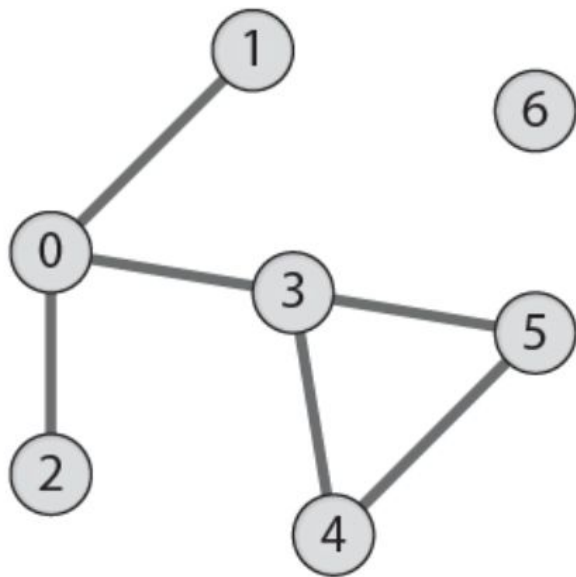
(i, j) がある場合 a_{ij} が1, ない場合0である $|V| \times |V|$ の行列



	0	1	2	3	4	5	6
0	0	1	1	1	0	0	0
1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	1	0	1	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0

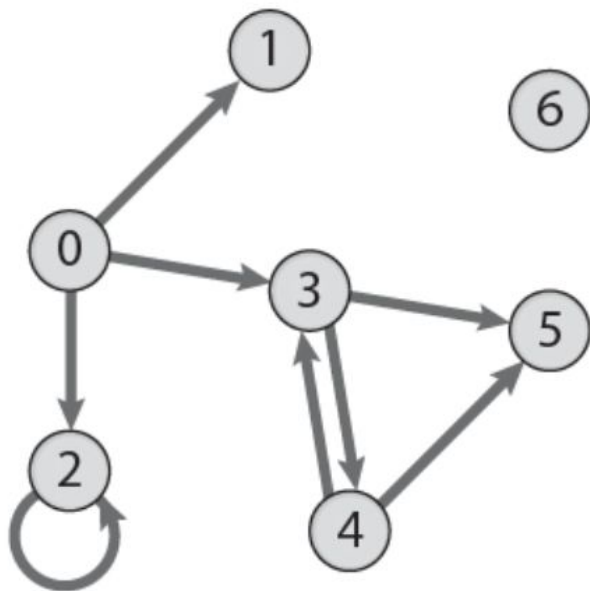
思考★ 実装★

無向グラフの隣接行列



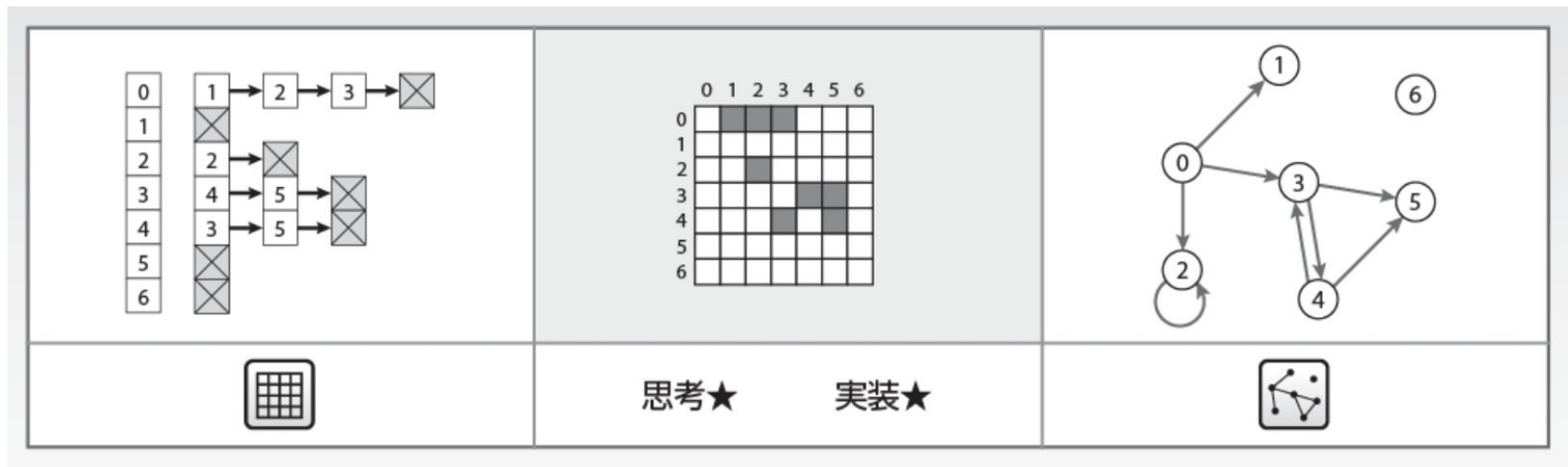
		j						
		0	1	2	3	4	5	6
i	0	0	1	1	1	0	0	0
	1	1	0	0	0	0	0	0
	2	1	0	0	0	0	0	0
	3	1	0	0	0	1	1	0
	4	0	0	0	1	0	1	0
	5	0	0	0	1	1	0	0
	6	0	0	0	0	0	0	0

有向グラフの隣接行列



		j						
		0	1	2	3	4	5	6
i	0	0	1	1	1	0	0	0
	1	0	0	0	0	0	0	0
	2	0	0	1	0	0	0	0
	3	0	0	0	0	1	1	0
	4	0	0	0	1	0	1	0
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0

問題：隣接リスト→隣接行列



入力：隣接リストで定義した有向グラフ

出力：対応する隣接行列を出力

入出力例

制約 $1 \leq n \leq 100$ ノード数

入力例

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

ノード数
ノード番号
ノードの出次数
隣接するノード

出力例

0	1	0	1
0	0	0	1
0	0	0	0
0	0	1	0

隣接行列の長所・短所

長所

エッジの参照，追加，削除を $O(1)$ でできる
($M[i][j]$ を読み書きするだけ)

短所

メモリが n^2 必要

グラフが疎であるほどメモリが勿体無い

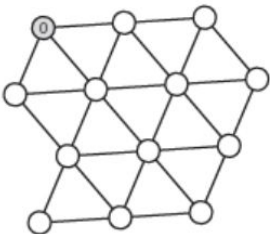
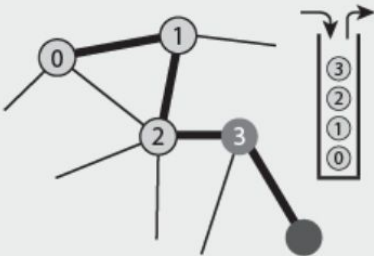
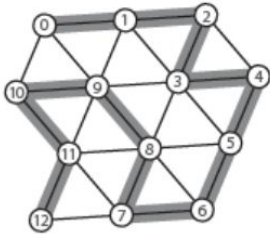



回答例

2次元配列Mの定義, cinとMの書き込み

目次

- 12.1 グラフ：問題をチャレンジする前に
- 12.2 グラフの表現
- 12.3 深さ優先探索**
- 12.4 幅優先探索
- 12.5 連結成分

問題：深さ優先探索

		
 	思考★★ 実装★★	

入力：前回と同じ形式の隣接リスト

出力：全てのノードを訪問し，各ノードの番号，発見時刻と完了時刻を記録

入出力例

制約 $1 \leq n \leq 100$

入力例

```
6
1 2 2 3
2 2 3 4
3 1 5
4 1 6
5 1 6
6 0
```

出力例

1	1	12
2	2	11
3	3	8
4	9	10
5	4	7
6	5	6

ノード番号

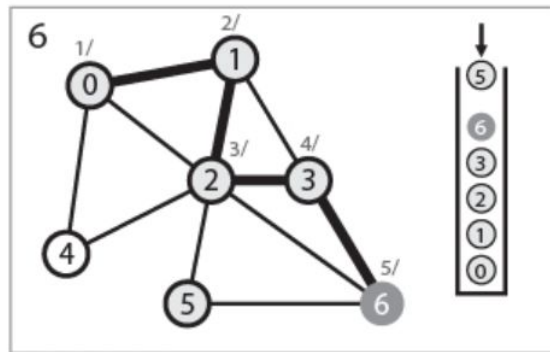
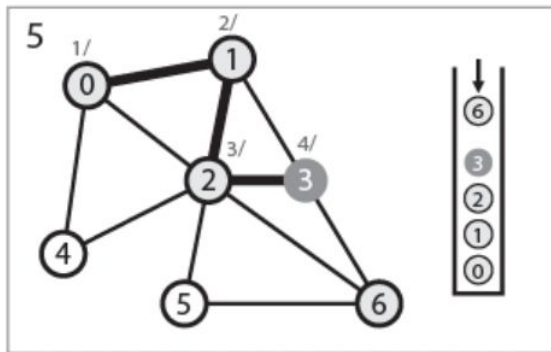
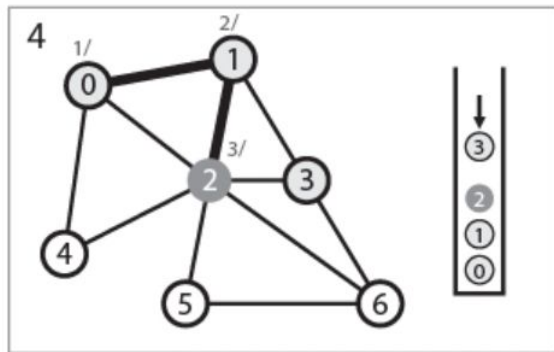
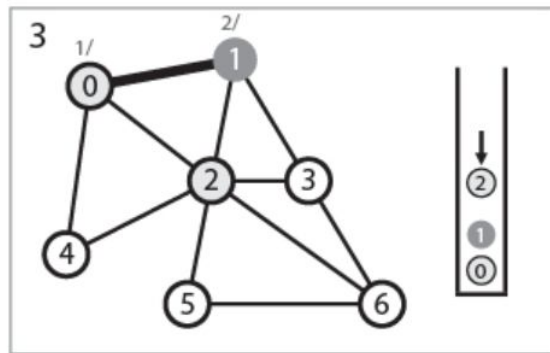
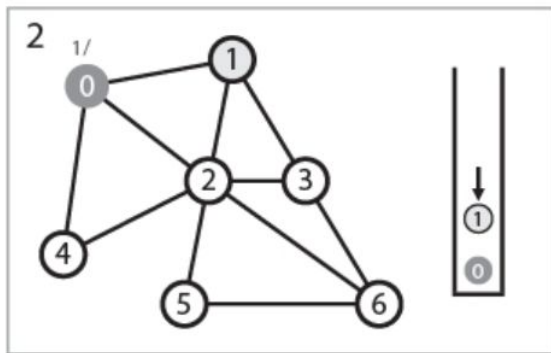
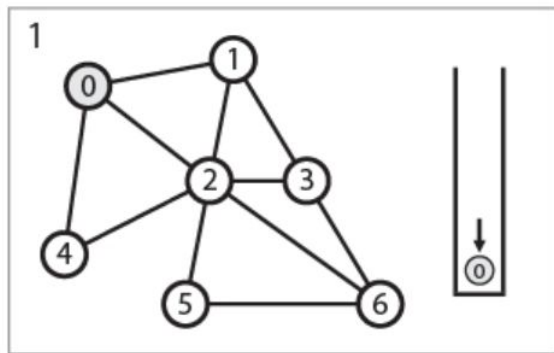
ノードの発見時刻

ノードの完了時刻

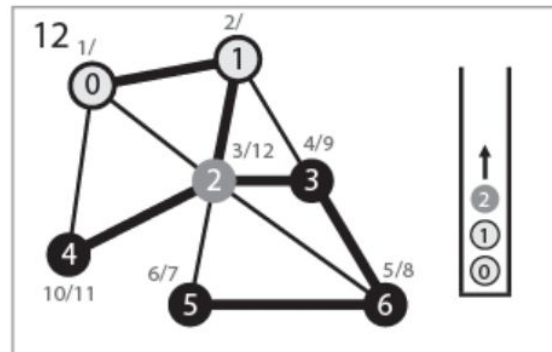
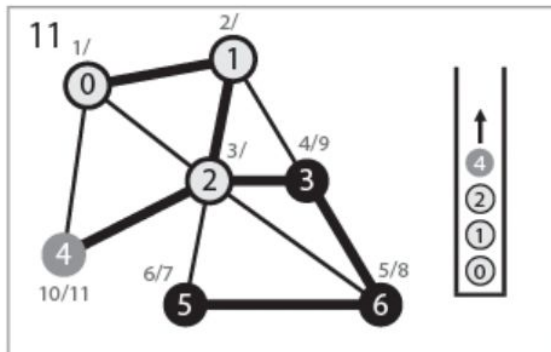
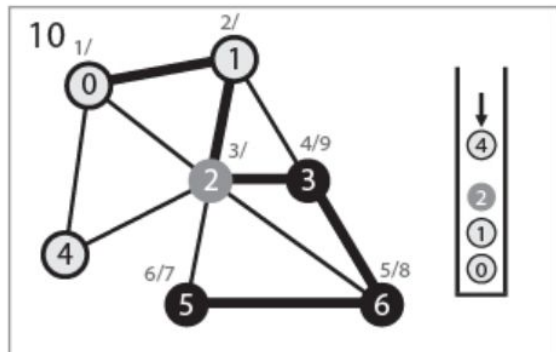
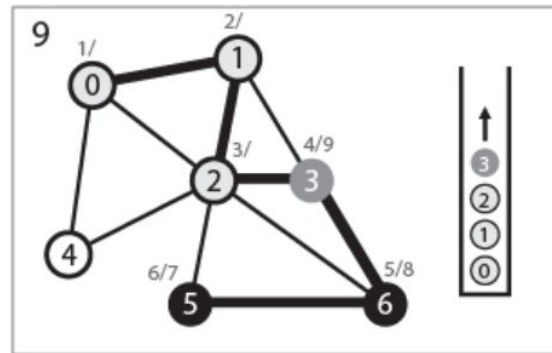
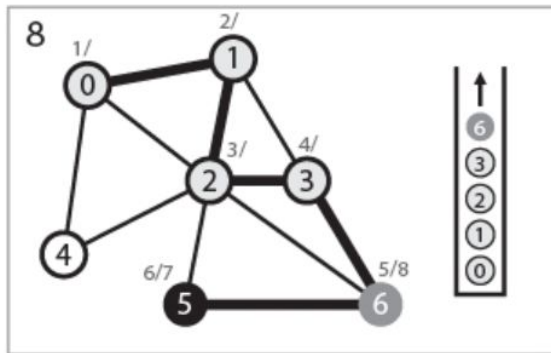
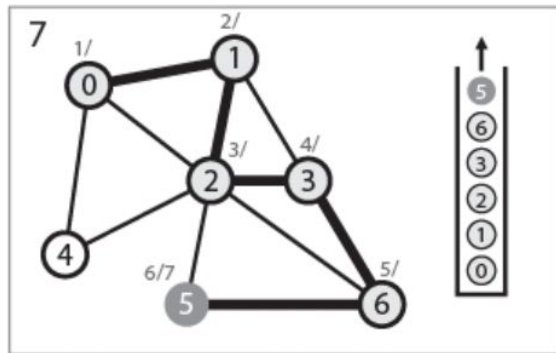
実装アプローチ

1. スタックを用いたDFS
2. 再帰を用いたDFS

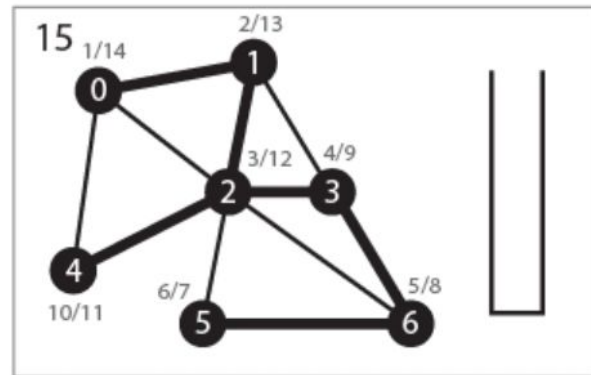
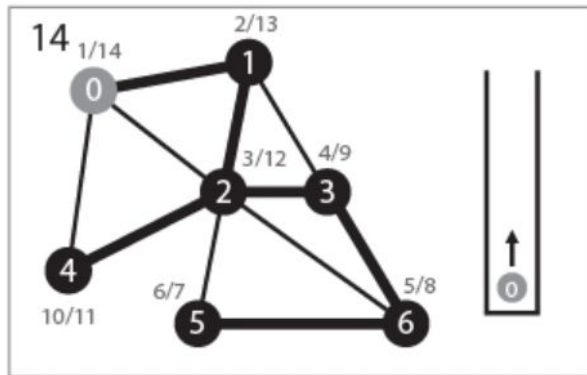
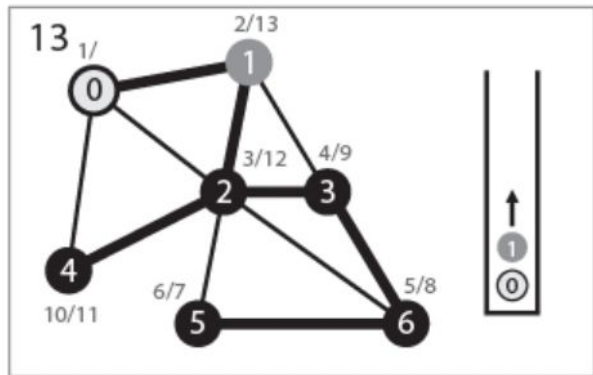
スタックを用いた動作例



スタックを用いた動作例



スタックを用いた動作例



Program 12.1: スタックによる深さ優先探索

WHITE=未訪問

GRAY=訪問した（発見）

BLACK=訪問完了

S: スタック

color[|v|]: ノード状態

d[|v|]: 発見時

f[|v|]: 完了時

```
1  dfs_init() // 頂点番号は 0-オリジン
2      全ての頂点の color を WHITE に設定
3      dfs(0) // 頂点 0 を始点として深さ優先探索
4
5  dfs(u)
6      S.push(u) // 始点 u をスタックに追加
7      color[u] = GRAY
8      d[u] = ++time
9
10     while S が空でない
11         u = S.top()
12         v = next(u) // u に隣接している頂点を順番に取得
13         if v != NIL
14             if color[v] == WHITE
15                 color[v] = GRAY
16                 d[v] = ++time
17                 S.push(v)
18         else
19             S.pop()
20             color[u] = BLACK
21             f[u] = ++time
```


Program 12.2: 再帰による深さ優先探索

```
1  dfs_init() // 頂点番号は 0-オリジン
2      全ての頂点の color を WHITE に設定
3      dfs(0)
4
5  dfs(u)
6      color[u] = GRAY
7      d[u] = ++time
8      for 頂点 v が 0 から |V|-1 まで
9          if M[u][v] && color[v] == WHITE
10             dfs(v)
11      color[u] = BLACK
12      f[u] = ++time
```

考察

各ノードで $|V|$ 個との隣接を確認するので $O(|V|^2)$

「完全グラフ」でなければ無駄が多い

大きいグラフの場合行列ではなく隣接リストが適切 (12.5)

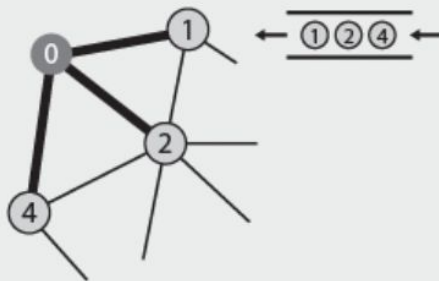
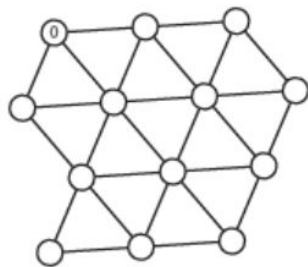
回答例

再帰のほう

目次

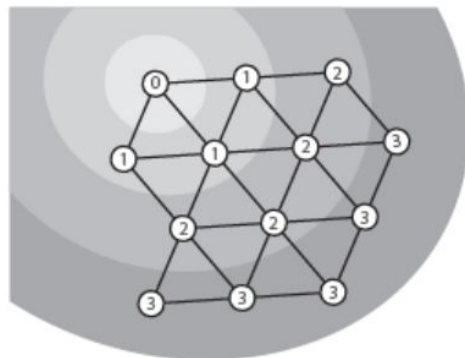
- 12.1 グラフ：問題をチャレンジする前に
- 12.2 グラフの表現
- 12.3 深さ優先探索
- 12.4 幅優先探索**
- 12.5 連結成分

問題：幅優先探索



思考★★

実装★★☆



入力：前回と同じ形式の隣接リスト

出力：ノード 1 から他の各頂点の距離

入出力例

入力例

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

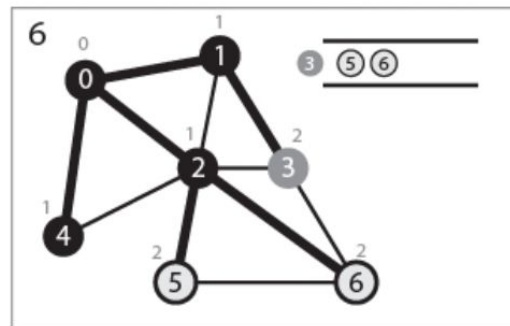
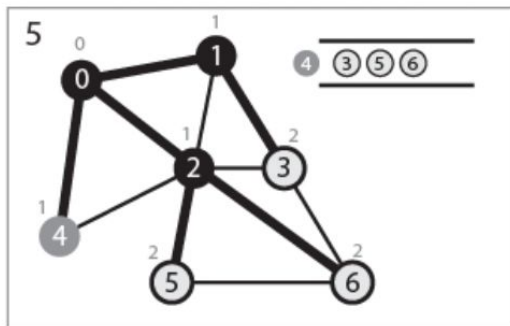
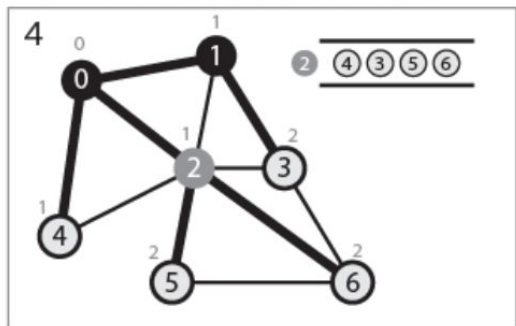
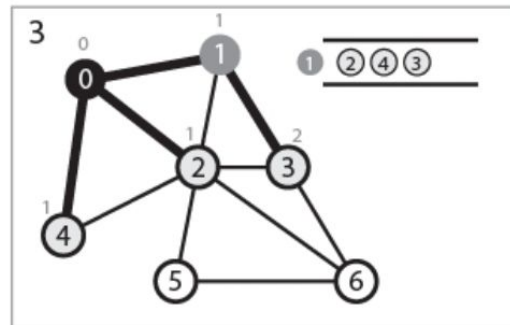
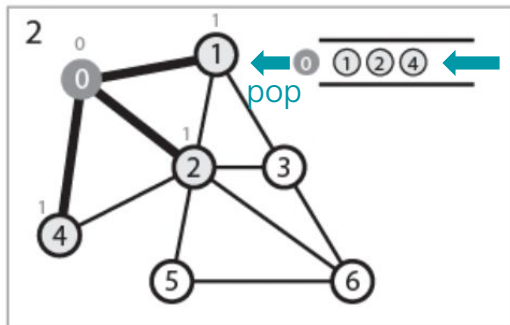
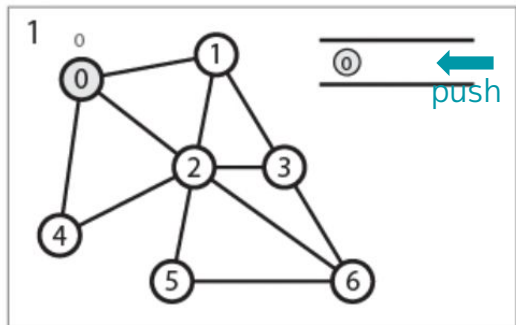
出力例

1	0
2	1
3	2
4	1

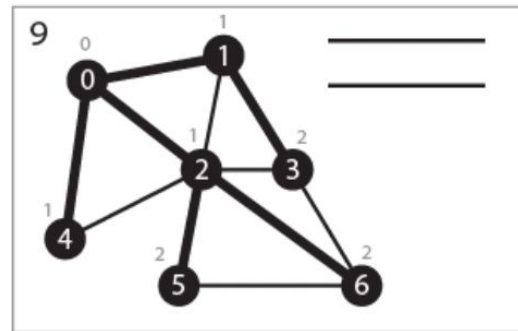
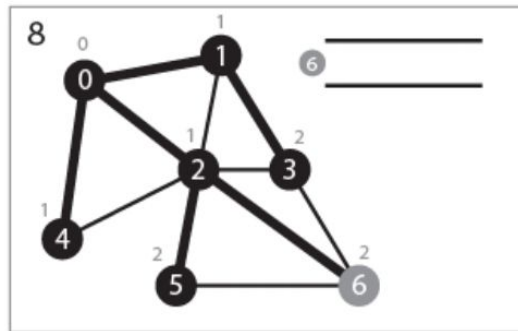
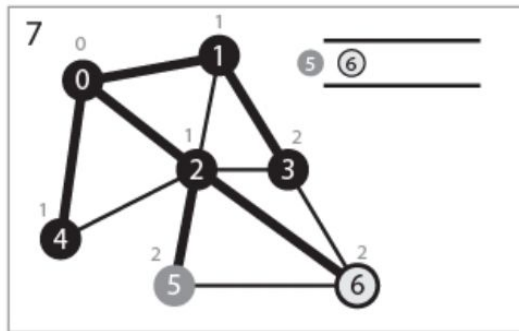
ノード番号

ノード1からの距離

キューを用いた動作例



キューを用いた動作例



WHITE=未訪問

GRAY=訪問した（発見）

BLACK=訪問完了

s: ルートノード

Q: キュー

color[|V|]: ノード状態

d[|V|]: ルートノードからの距離

Program 12.3: 幅優先探索

```
1  bfs() // 頂点番号は 0-オリジン
2      全ての頂点について、color[u] を WHITE に設定
3      全ての頂点について、d[u] を INFTY に設定
4
5      color[s] = GRAY
6      d[s] = 0
7      Q.enqueue(s)
8
9      while Q が空でない
10         u = Q.dequeue()
11         for v が 0 から |V|-1 まで
12             if M[u][v] && color[v] == WHITE
13                 color[v] = GRAY
14                 d[v] = d[u] + 1
15                 Q.enqueue(v)
16         color[u] = BLACK
```

考察

各ノードで $|V|$ 個との隣接を確認するので $O(|V|^2)$

行列を使った深さ優先探索のときと同じ話

回答例

queue<int>でpushやpopを使う

目次

- 12.1 グラフ：問題をチャレンジする前に
- 12.2 グラフの表現
- 12.3 深さ優先探索
- 12.4 幅優先探索
- 12.5 連結成分**

問題：連結成分

		
 	思考★★ 実装★★★★	   

ノードSからノードTへたどり着けるか？

入出力例

制約 $2 \leq n \leq 100,000$ ノード数

$0 \leq m \leq 100,000$ エッジ数

$1 \leq q \leq 10,000$ 質問数

入力例

```
10 9 ノード数 エッジ数
0 1 エッジの定義
0 2
3 4
5 7
5 6
6 7
6 8
7 8
8 9
3 クエリの数
0 1 SからTへ行けるか？
5 9
1 3
```

出力例

```
yes
yes
no
```

隣接リストおさらい

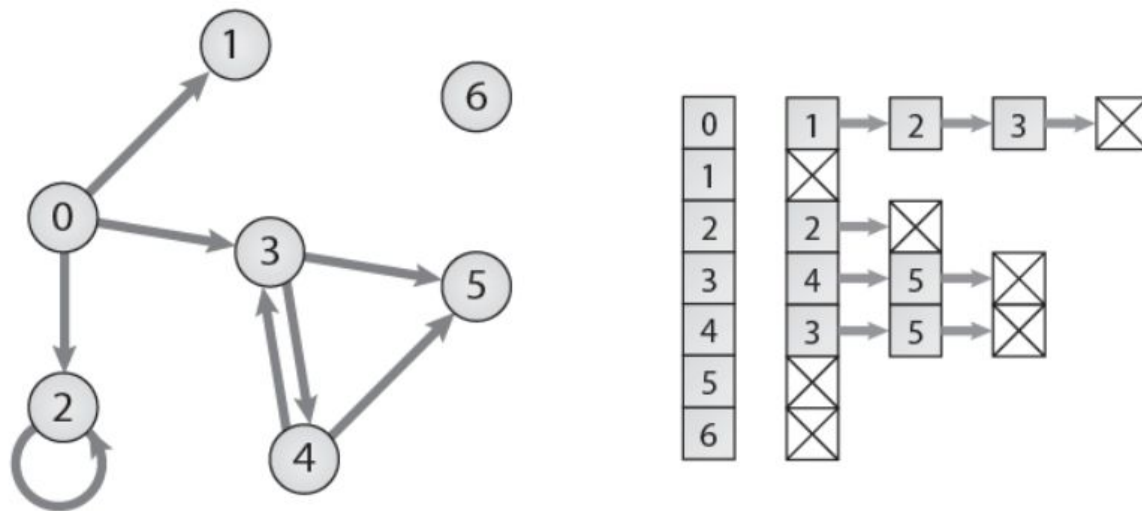


図 12.11: 有向グラフの隣接リスト

隣接リストの実装例

Program 12.4: vector による隣接リスト

```
1  vector<int> G[100]; // 頂点数が 100 のグラフを表す隣接リスト
2  : :
3  G[u].push_back(v); // 頂点 u から頂点 v へ向かって辺を張る
4  : :
5  // 頂点 u に隣接する頂点 v を探索
6  for ( int i = 0; i < G[u].size(); i++ ) {
7      int v = G[u][i];
8      : :
9  }
```


隣接リストの長所・短所

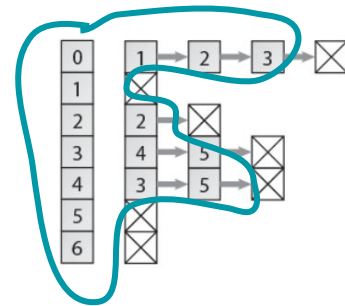
隣接リストを使う場合, DFSとBFSは $O(|V|+|E|)$

長所

メモリの効率 (存在するエッジの数に比例)

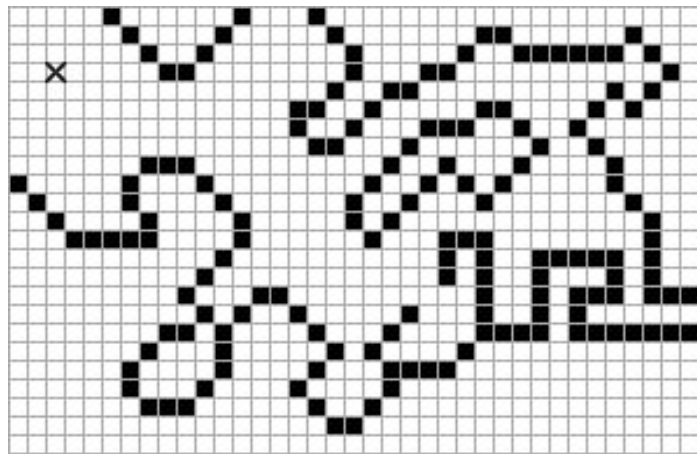
短所

リストの探索・操作効率に依存



解答例

色 (id)の「塗りつぶし」とDPSで実装
塗ったあとSとTが同じ色であれば辿れると判断



まとめ

- グラフの種類
- 隣接リスト vs 隣接行列
- DFS と BFS