

アルゴリズム論 (第5回)



岩手県立大学
Iwate Prefectural University

佐々木研(情報システム構築学講座)
講師 山田敬三

k-yamada@iwate-pu.ac.jp

グラフ探索



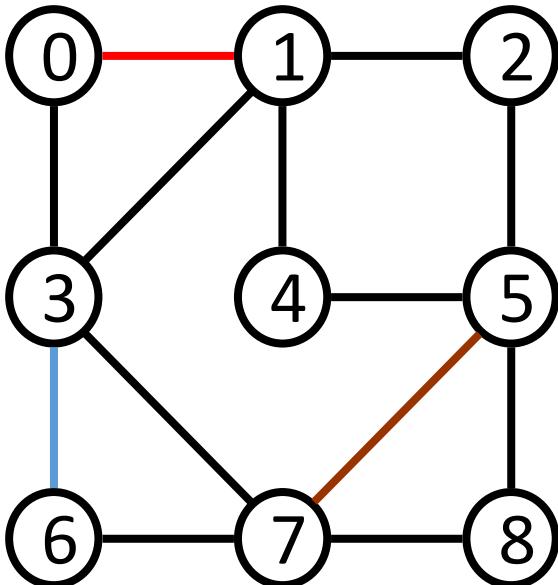
岩手県立大学
Iwate Prefectural University

グラフ探索

- グラフにおいて、ある点からある点まで到達できるかどうかを探索
 - 途中経路は問わない。
- あるノードから到達できる全てのノードを答える
- 結果は始点をルートとする木として表現可能

隣接行列

- 無向グラフのときは対称行列



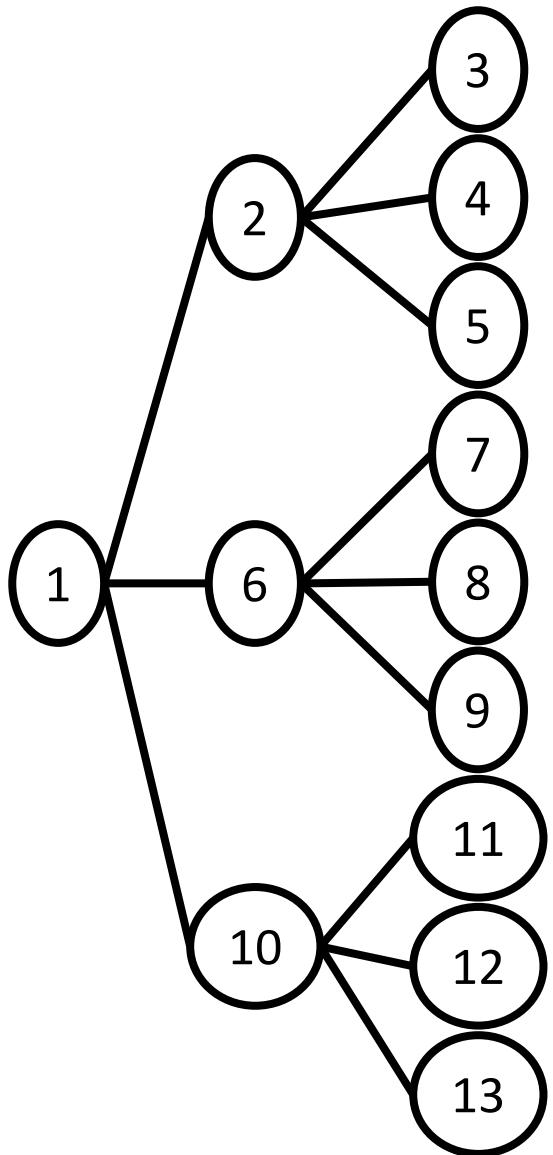
0 1
0 3
1 2
1 3
1 4
2 5
3 6
3 7
4 5
5 7
5 8
6 7
7 8

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

探索方法

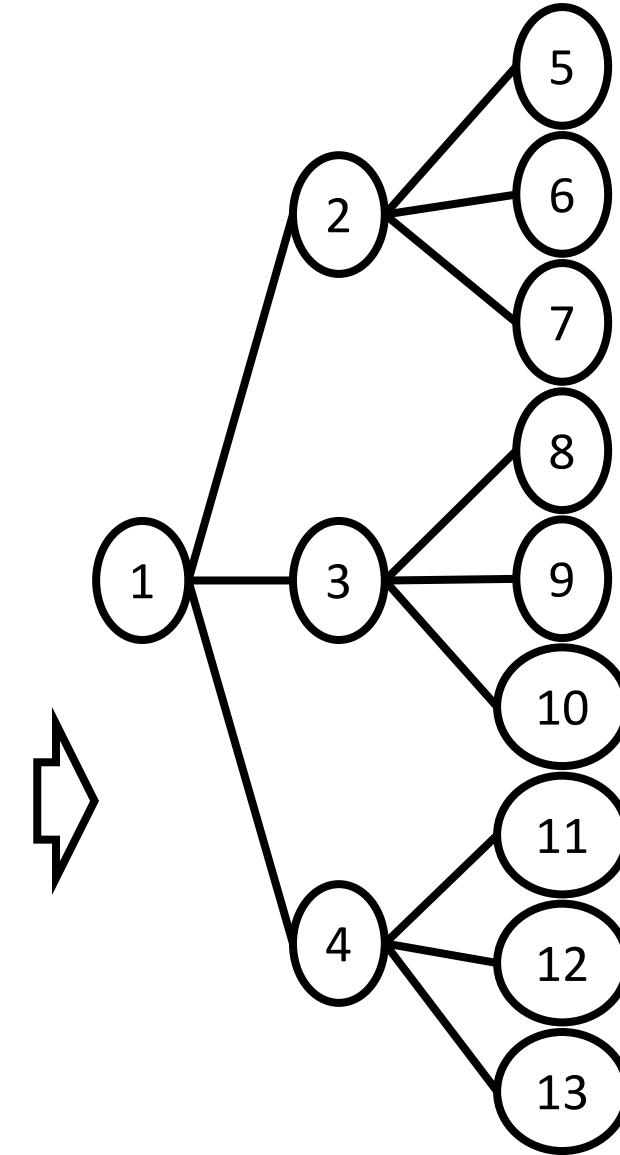
- 縱形探索
 - 深さ優先探索
 - 直列的に処理
 - 記憶領域小
- 橫形探索
 - 幅優先探索
 - 並列的に処理
 - 記憶領域大

探索方法



縦形探索

横形探索



深さ優先探索

- 可能な限り深く探索
- 隣接ノードが全て探索済みになると、探索できる隣接ノードがあるところまで戻って探索
 - バックトラック
- 始点に戻ってきたところで終了
 - すべてのノードの探索が終了

深さ優先探索

1. 探索

- i. 隣接ノード中の未探索ノードを探索
全件探索
- ii. 隣接ノードに未探索ノードがなかったら
次へ

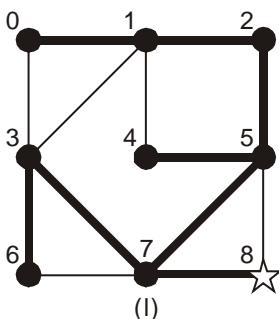
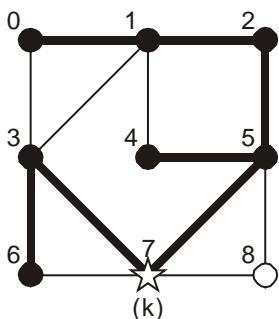
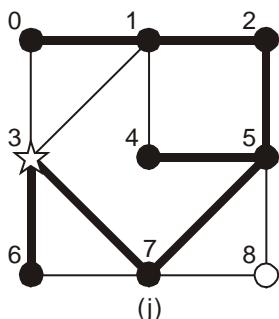
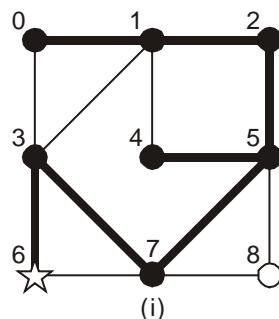
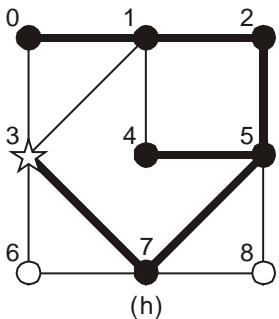
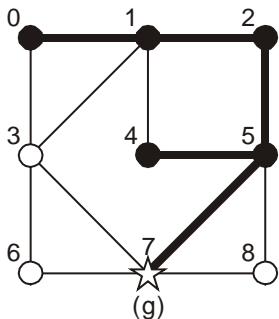
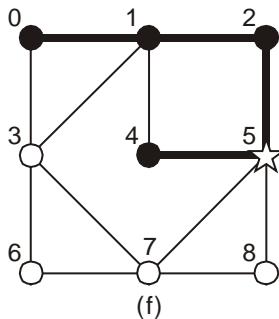
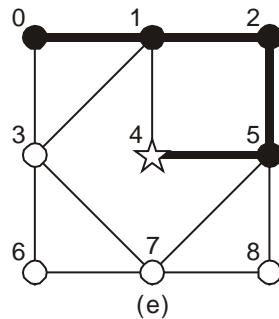
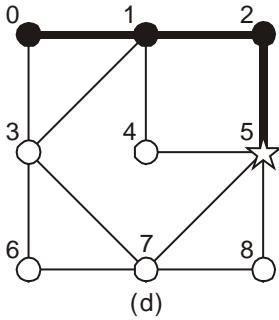
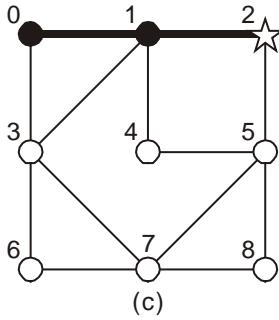
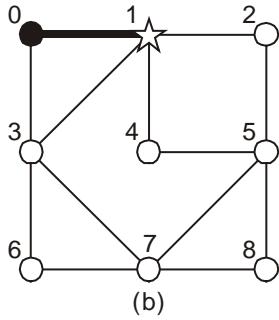
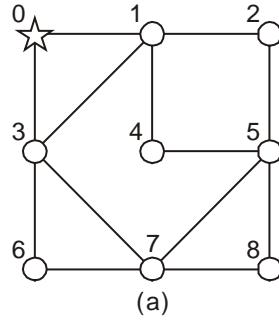
2. 経路の後退(バックトラック)

- i. 未探索隣接ノードが残っているノードを
発見するまで戻る。

3. 探索の繰り返し&終了条件

手順1. 2. を繰り返し、始点に戻ると終了

深さ優先探索



使用するデータ

```
#define NODES 9 /* ノードの数 */

struct edge { /* 辺の両端のノード番号 */
    int start_node;
    int end_node;
};

int matrix[NODES][NODES]; /* 隣接行列 */
int df_flag[NODES]; /* 探索状況 */
struct edge df_tree[NODES-1]; /* 探索木 */
```

処理手順

1. 隣接行列を作成
データを読み込み、隣接行列を作成
2. 初期設定
辺の数を管理するedge_cntを初期化
`df_flag[]`を0で初期化
3. 探索処理
ノード0を始点とする探索
4. 結果を出力

探索処理

`void df_search(int u)`

1. 探索済みの登録

`df_flag[u]`を探索済み(1)にする。

2. 次に探索するノードvの調査

ノードuと隣接するノードを調べ,
`df_flag[]`により次に探索するノー
ドvを決める。

3. ノードu,v間の辺の登録

`df_tree[edge_cnt]`のメンバに
辺(u,v)を登録

探索処理

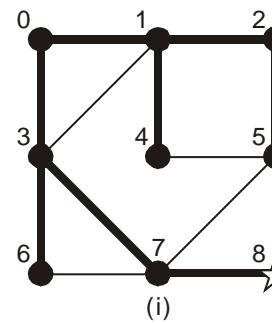
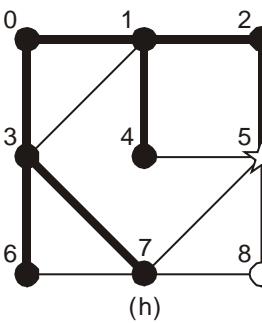
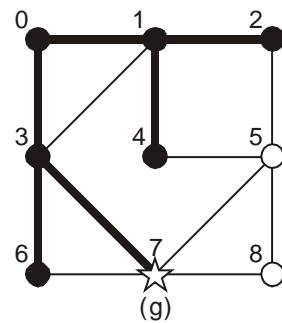
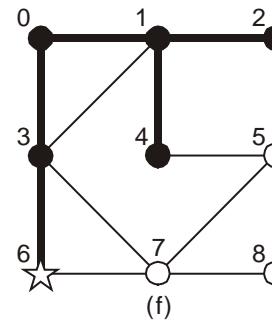
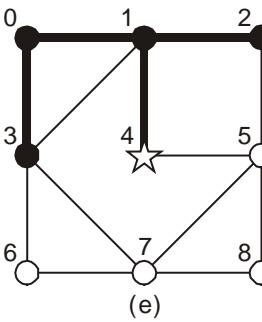
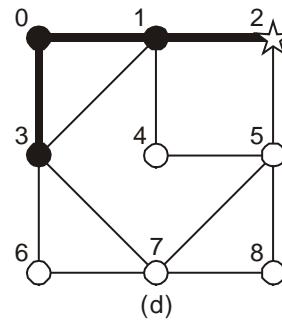
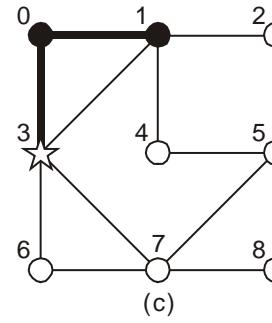
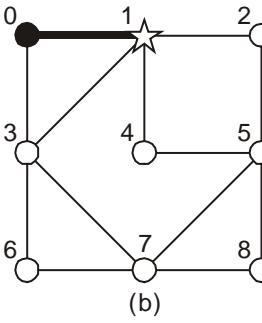
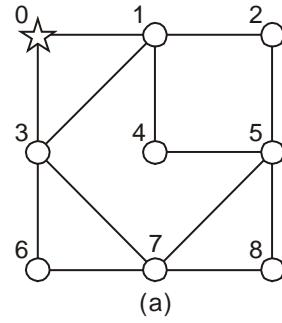
4. ノードvを始点とするグラフ探索
`df_search(v)`を実行することにより,
再帰的に探索
5. 探索の繰り返し
2.～4.を繰り返す

幅優先探索

1. ノード1つ分の探索
現在の探索ノードの隣接ノード**全て**を探索

2. 探索の繰返し&終了
探索が進められた全てのノードに対し1.を繰り返す.
全てのノードが探索されたら終了

幅優先探索



使用するデータ

```
#define NODES 9 /* ノードの数 */

struct edge { /* 辺の両端のノード番号 */
    int start_node;
    int end_node;
};

int matrix[NODES][NODES]; /* 隣接行列 */
int bf_flag[NODES]; /* 探索状況 */
struct edge bf_tree[NODES-1]; /* 探索木 */

int queue[NODES]; /* キュー */
int head=0,tail=0; /* キューの先頭と末尾 */
```

探索処理

`void bf_search(int start)`

1. キューの初期化

次に探索するノードをキューで管理

2. 始点のキューへの登録&探索済みの登録

始点startをキューへ登録

`bf_flag[start]`に探索済みを示す1
を登録

3. 探索ノードの取り出し

次の探索ノードをキューから取り出し, uとする.

探索処理

4. 次に探索するノードvの調査
全てのノードに対し, uとの隣接と
`bf_flag[]`を調査し, 次に探索する
ノードvを調査
5. ノードu,v間の辺の登録
`bf_tree[edge_cnt]`のメンバに辺
(u, v)を登録
6. キューへの登録&探索済みの登録
2.の様にノードvをキューへ登録し, 探索
済みの登録

探索手順

7. ノード u に関する手順の繰返し
4.～6.を次に探索するノードがなくなるまで繰り返す.
8. 次のノードの探索
キューにノードが残っていれば3.に戻り
次のノードの探索
キューが空だったら終了