データ構造とアルゴリズム 第10週

掛下　哲郎
kake@is.saga-u.ac.jp
代講:大月　美佳
mika@is.saga-u.ac.jp

第8～9週のまとめ
ソートアルゴリズムの特徴(その1)

ソートアルゴリズム	特徴
バブルソート	• アルゴリズムが最も単純で実現が容易 • 比較回数，コピー回数が$O(n^2)$必要.
選択ソート	• アルゴリズムが比較的単純 • コピー回数が$O(n)$で済むため，各要素のサイズが大きい場合に効率が良い.
挿入ソート	• データの交換(代入3回)が不要. 移動(代入1回)で済む. • 平均比較回数がバブルソート等の半分で済む. • データがほぼソートされている場合には，ほぼ$O(n)$時間で実行できる.
シェルソート	• 高々$O(n^{3/2})$時間でソートできる. • 多くの場合，挿入ソートより高速. ヒープソートより単純.
ヒープソート	• 最悪の場合でも$O(n \log n)$時間でソートできる. • ヒープを保存するために，$O(n)$の余分な作業領域が必要

第8～9週のまとめ

ソートアルゴリズムの特徴（その2）

ソートアルゴリズム	特徴
マージソート	• 最悪の場合でもO(n log n)時間でソートできる． • マージ処理を実行するために，O(n)の余分な作業領域が必要
クイックソート	• 平均O(n log n)時間だが，最悪$O(n^2)$時間が必要． • 高速ソートアルゴリズムとして有名． • 実際にも良く使われている． • 余分な作業領域は，最悪の場合でもO(log n)で済む．
バケットソート	• データ範囲が1～Nに限定されている． • バケットを保持するために，O(N)の余分な作業領域が必要． • データの相互比較を行わないため，O(n+N)時間で実行できる．

比較回数の下界 ⇒ O(n log n)

講義スケジュール

週	講義計画
1ー2	導入
3	探索問題
4ー5	基本的なデータ構造
6	動的探索問題とデータ構造
7	アルゴリズム演習（第1回）
8ー9	データの整列
10ー11	グラフアルゴリズム
12	文字列照合のアルゴリズム
13	アルゴリズム演習（第2回）
14	アルゴリズムの設計手法
15	計算困難な問題への対応

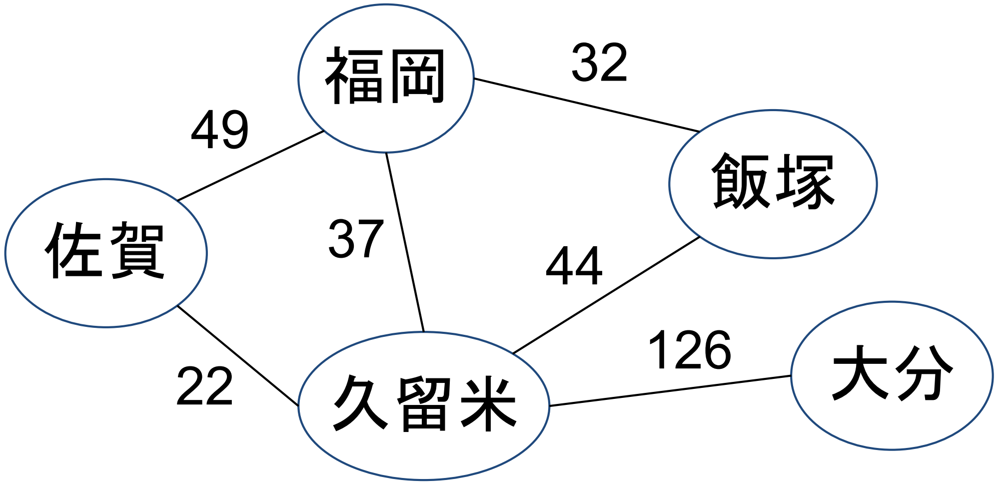
データ構造

アルゴリズム

今日のトピック

グラフアルゴリズム

- グラフの定義
- グラフの表現
- グラフの探索
  - 幅優先探索
  - 深さ優先探索
- 探索の応用
  - 探索木
  - 前置記法，中置記法，後置記法



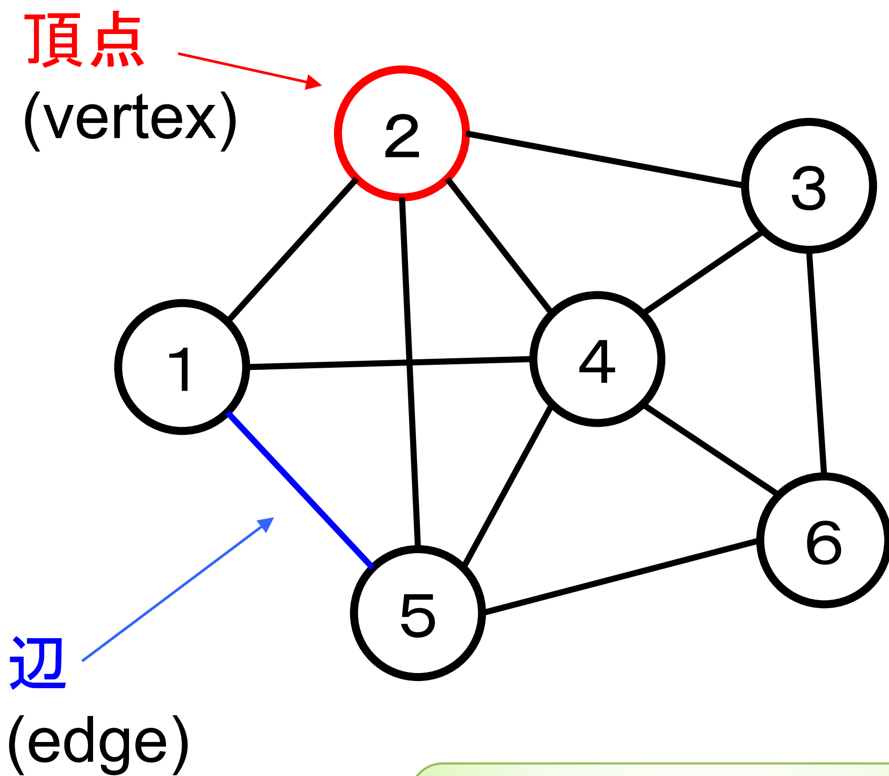
グラフ(graph)の定義

- グラフ $G = (V,E)$
  - 頂点(vertex)の集合Vと，辺(edge)の集合Eとの組
  - $V = \{ v_1, v_2, v_3, \ldots, v_n \}$ ※ 頂点数n
  - $E = \{ e_1, e_2, e_3, \ldots, e_m \}$ ※ 辺の数m

- $e_k = \{v_i,v_j\}$ ⇒ 無向辺
  $e_k = (v_i,v_j)$ ⇒ 有向辺(始点v_i，終点v_j)

要書き込み

グラフの例：無向グラフ



V = {1, 2, 3, 4, 5, 6}

E = {

{1,2}, {1,4}, {1,5},

{2,3}, {2,4}, {2,5},

{3,4}, {3,6}, {4,5},

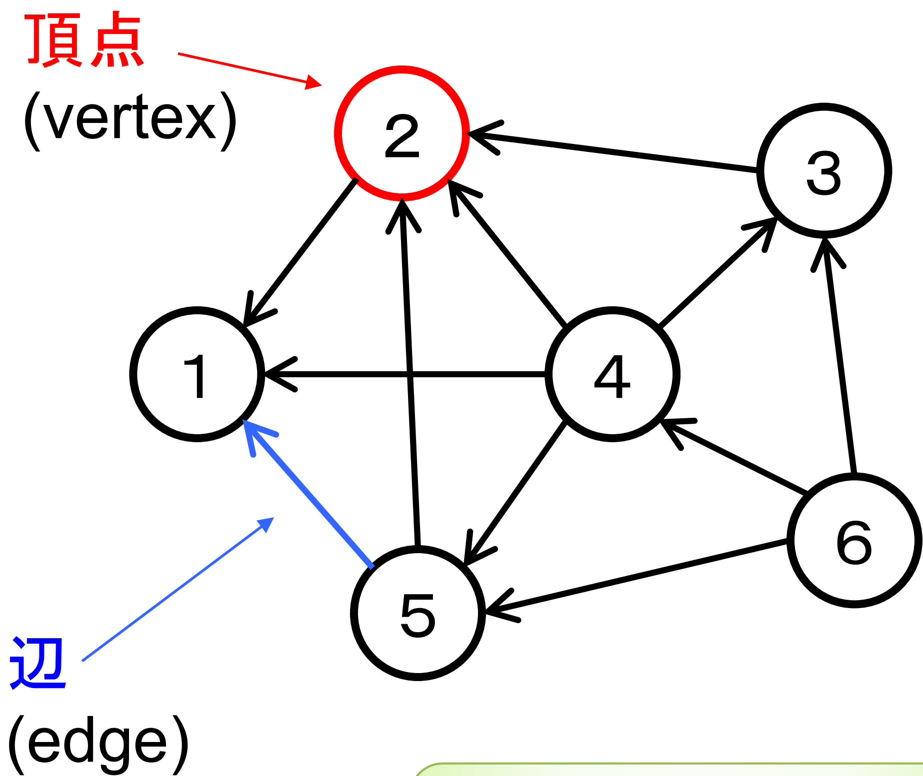
{4,6}, {5,6}

}

辺には向きがない
（無向辺）

要書き込み

グラフの例：有向グラフ



辺に向きがある
（有向辺）

V = {1, 2, 3, 4, 5, 6}

E = {

(2,1), (3,2),(4,1),

(4,2), (4,3), (4,5),

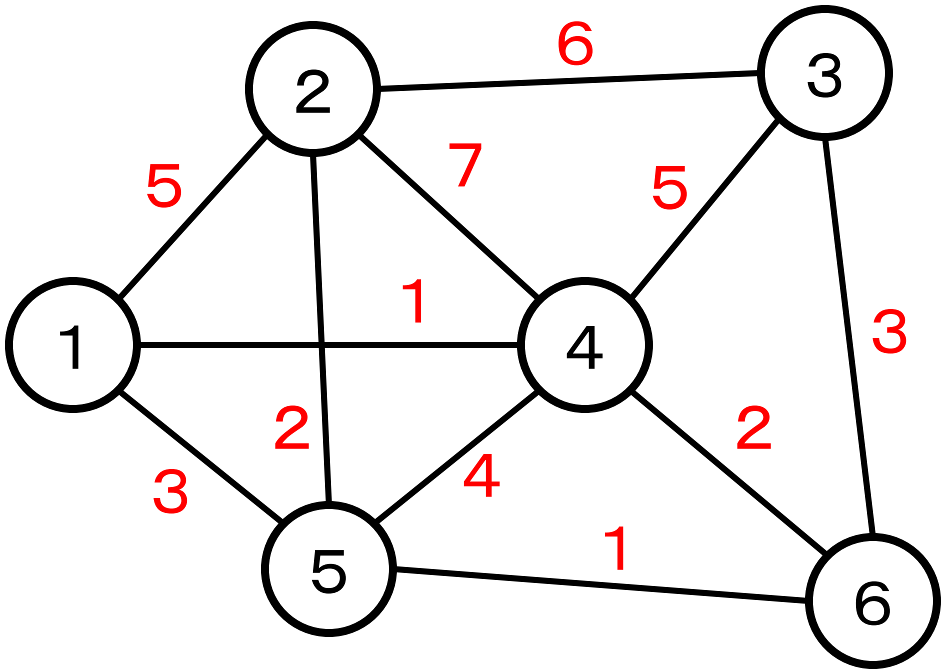
(5,1), (5,2), (6,3),

(6,4), (6,5)

}

要書き込み

グラフの例：重み付き無向グラフ

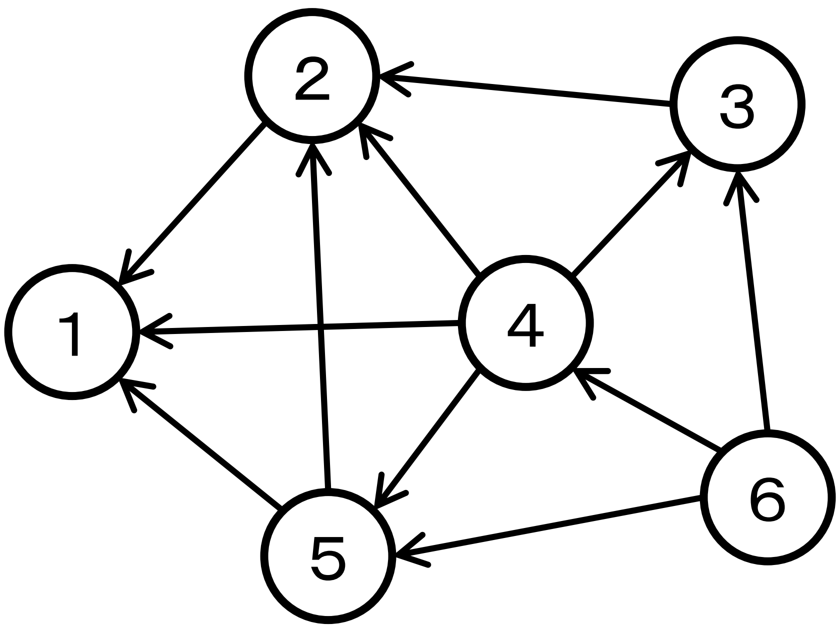


「重み」の例
料金,
所要時間,
容量,
重要度,
コスト
などとして定義

- 辺に重みがある
- 重み付き有向グラフも同様に定義できる

要書き込み

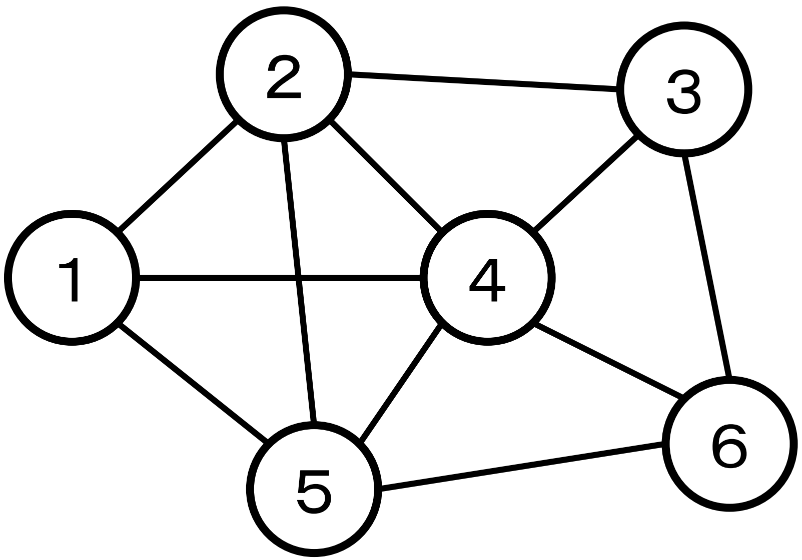
グラフを表現するデータ構造（その1）
隣接行列（adjacency matrix）



(i,j) ∈ E ⇔ i行j列が1

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

グラフを表現するデータ構造（その1）
隣接行列（adjacency matrix）



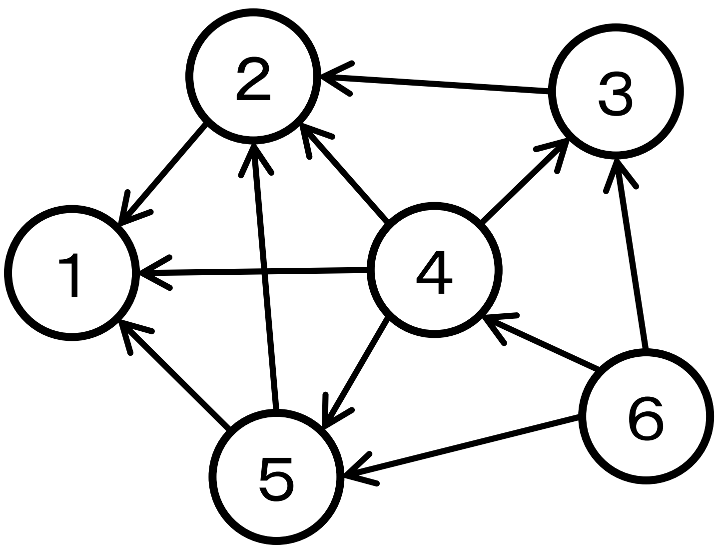
	1	2	3	4	5	6
1	0	1	0	1	1	0
2	1	0	1	1	1	0
3	0	1	0	1	0	1
4	1	1	1	0	1	1
5	1	1	0	1	0	1
6	0	0	1	1	1	0

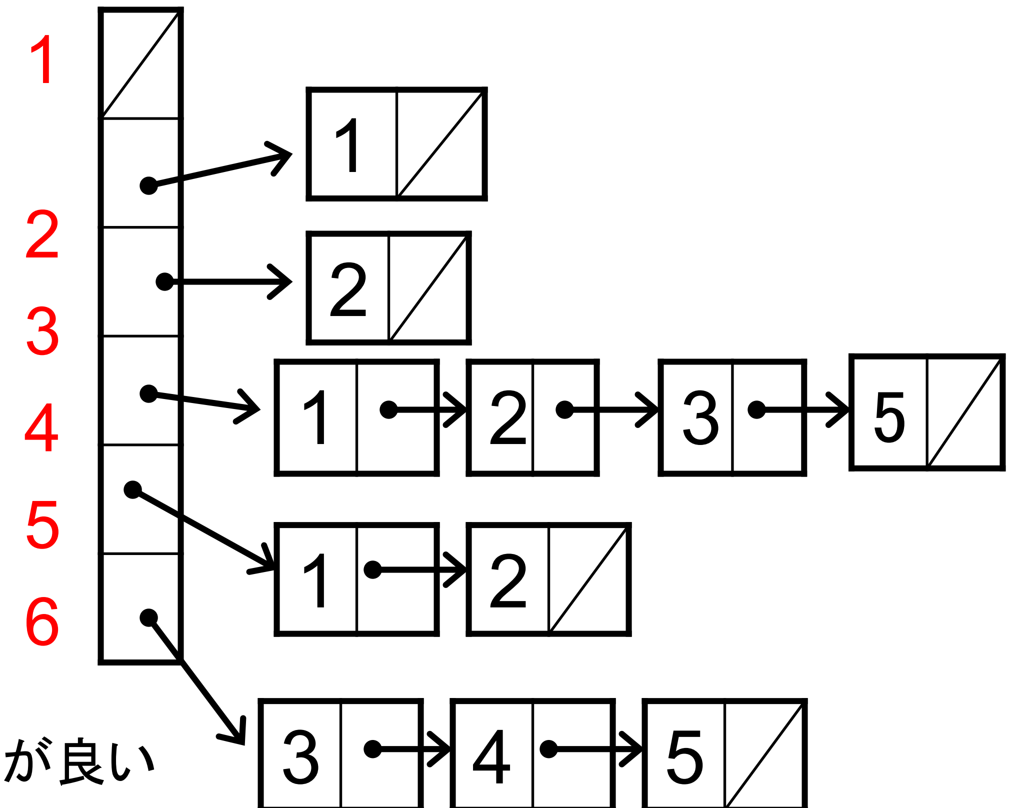
- 記憶領域O(n^2)
  ⇒ グラフが疎のとき領域が無駄
- 各辺のアクセスがO(1) ⇒ 速い
- グラフが密の場合に向く

無向グラフの場合は
対称行列

要書き
込み

グラフを表現するデータ構造（その2）
隣接リスト（adjacency list）





- 記憶領域O(n+m) ⇒ 記憶効率が良い
- 辺は逐次探索 ⇒ グラフが密のとき低速
- グラフが疎の場合に向く

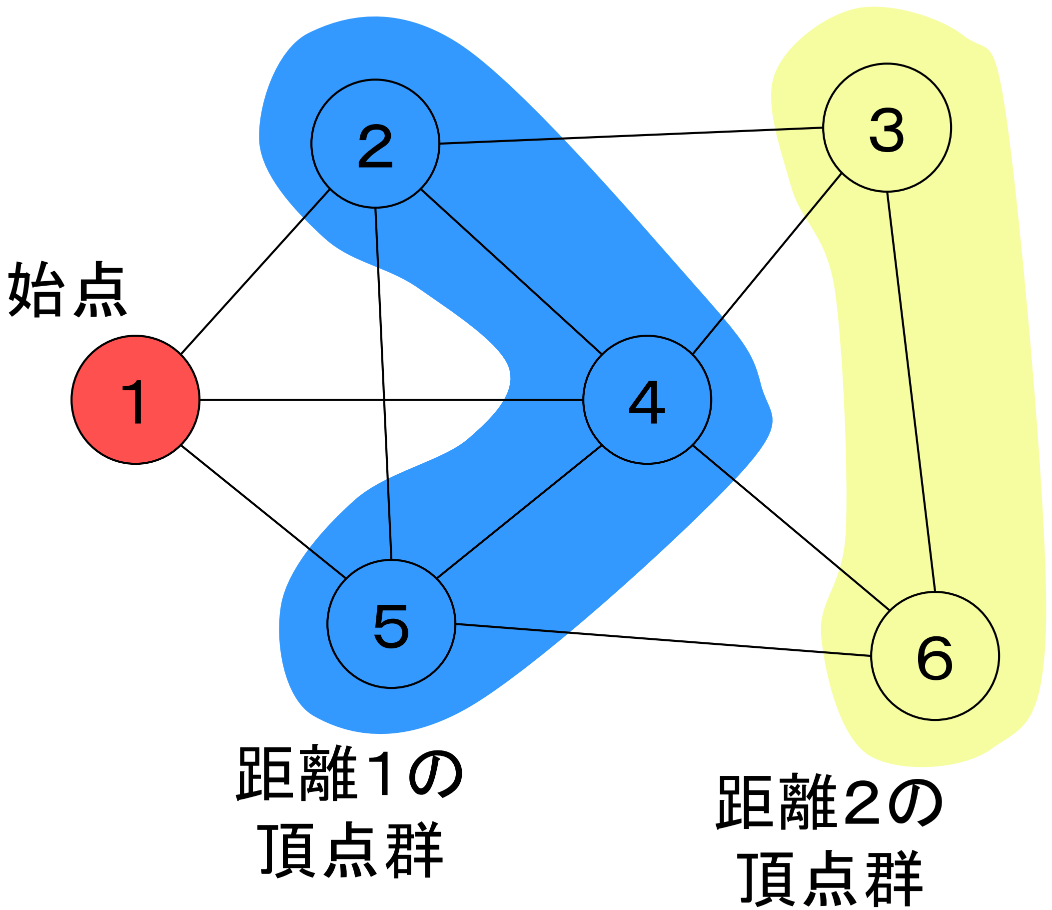
グラフの探索

- ある頂点vを出発点として、vの連結成分の全ての頂点を調べる
- どの順番で頂点を調べるか？
- 頂点を「訪問」していく順番 --> ２つの戦略
  - 幅優先探索 : Breadth First Search (BFS)
    - 出発点から近い順に訪問
  - 深さ優先探索 : Depth First Search (DFS)
    - 行けるところまで突き進みながら訪問

要書き込み

幅優先探索の例

Breadth First Search (BFS)



頂点を

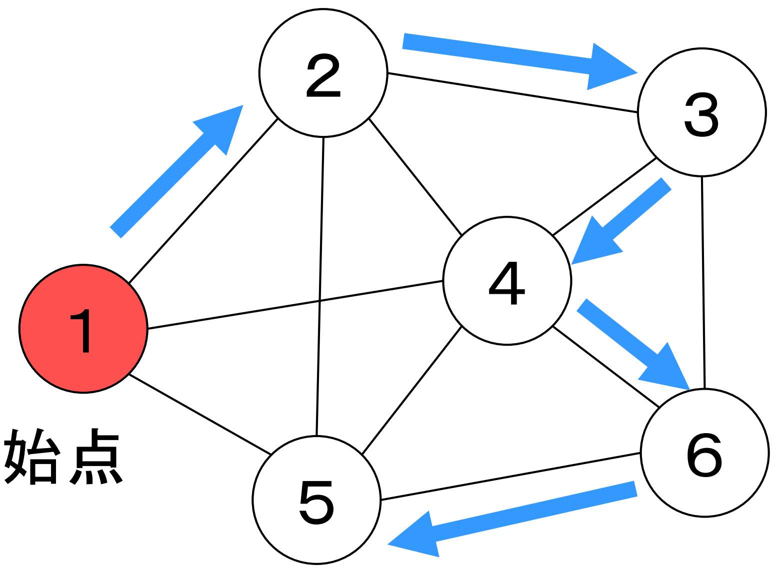
1
↓
2, 4, 5
↓
3, 6

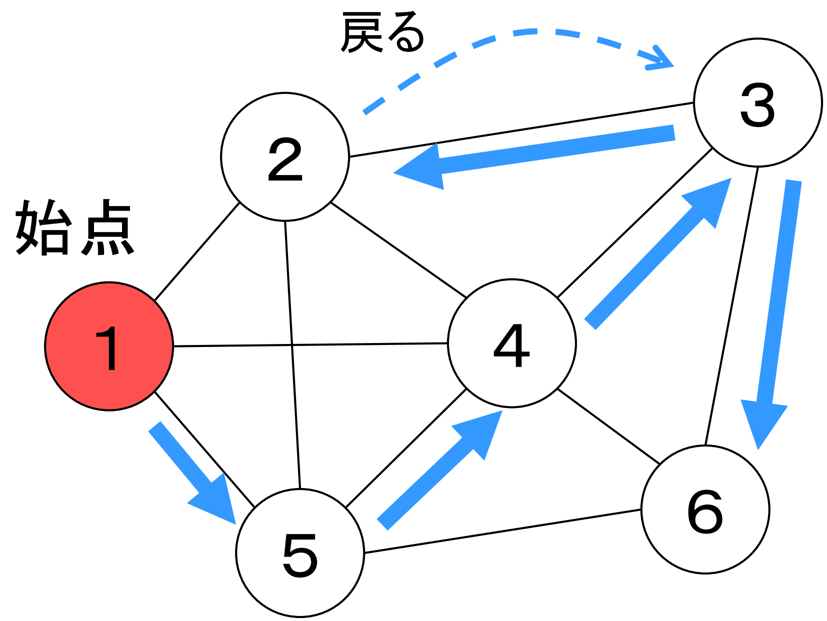
の順に訪問

例1: 1→2→4→5→3→6
例2: 1→4→2→5→6→3

深さ優先探索の例

Depth First Search
(DFS)





例1: 1→2→3→4→6→5
例2: 1→5→4→3→2→6

今日のトピック

- グラフの定義
- グラフの表現
- グラフの探索
  - 幅優先探索（BFS）
  - 深さ優先探索（DFS）
- グラフ探索の応用
  - 探索木
  - 前置記法，中置記法，後置記法

渡河問題

- 一人の男が，狼，山羊，紙を連れて河を渡ろうとしている．川にはボートがあるが，男の他には1つのものだけしか積めない．
- しかし，男がいなければ狼は山羊を食べてしまうし，山羊は紙を食べてしまう．
- このようなことが起こらないように河を渡るには，どのようにすれば良いか．可能な解を一つだけ求めよ．
- なお，河を渡る際には，ボートに男が乗っていなければならない．しかし，男がいなくても動物は逃げないとする．

幅優先探索の応用

探索木：可能な状態の列挙

L：河の左岸
R：河の右岸

初期状態

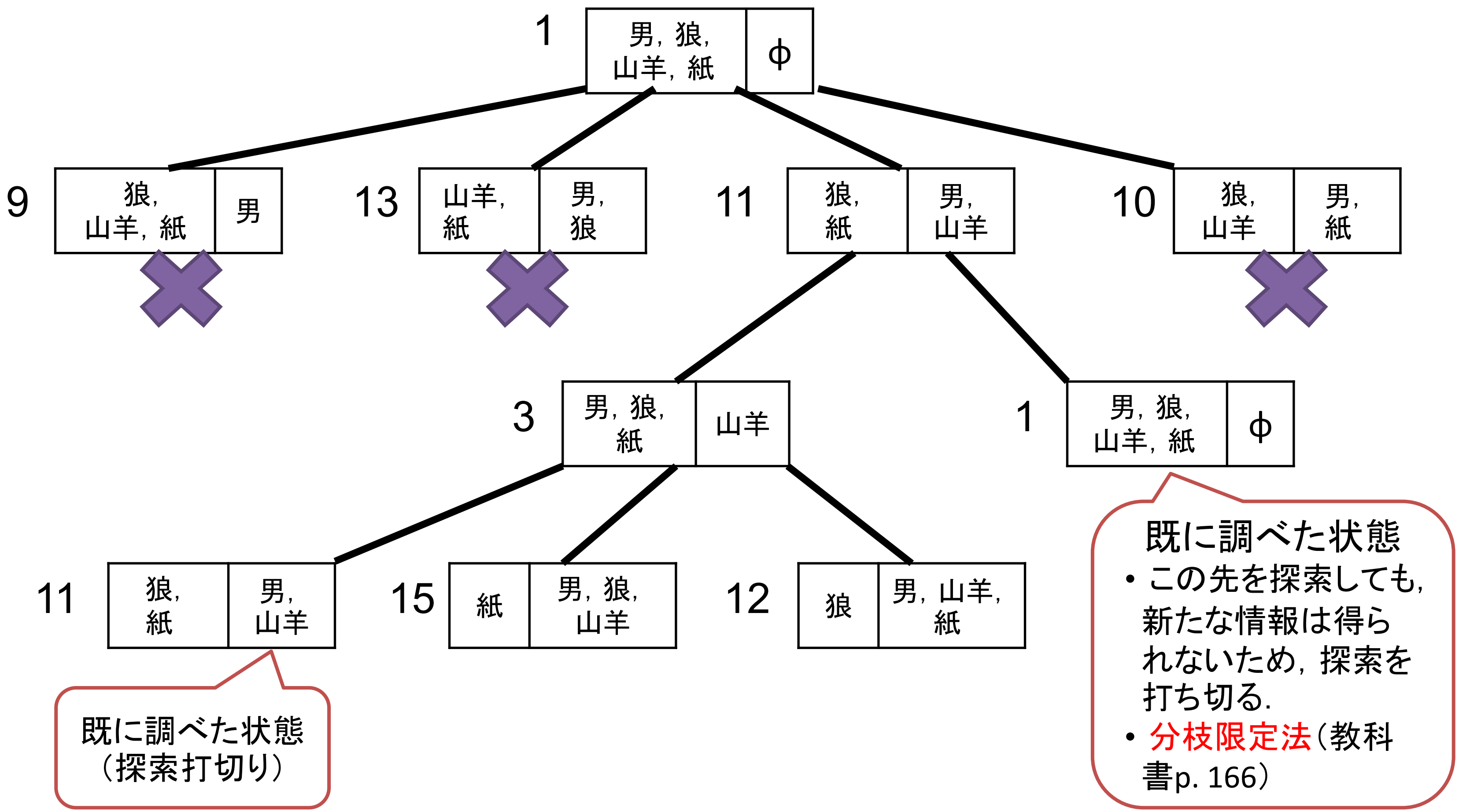
状態番号	男	狼	山羊	紙		
1	L	L	L	L	男，狼，山羊，紙	φ
2	L	L	L	R	男，狼，山羊	紙
3	L	L	R	L	男，狼，紙	山羊
4	L	L	R	R	~~男，狼~~	~~山羊，紙~~
5	L	R	L	L	男，山羊，紙	狼
6	L	R	L	R	男，山羊	狼，紙
7	L	R	R	L	~~男，~~	~~狼，山羊~~
8	L	R	R	R	~~男~~	~~狼，山羊，紙~~

不正な状態

状態番号	男	狼	山羊	紙		
9	R	L	L	L	~~狼，山羊，紙~~	~~男~~
10	R	L	L	R	~~狼，山羊~~	~~男，紙~~
11	R	L	R	L	狼，紙	男，山羊
12	R	L	R	R	狼	男，山羊，紙
13	R	R	L	L	~~山羊，紙~~	~~男，狼~~
14	R	R	L	R	山羊	男，狼，紙
15	R	R	R	L	紙	男，狼，山羊
16	R	R	R	R	φ	男，狼，山羊，紙

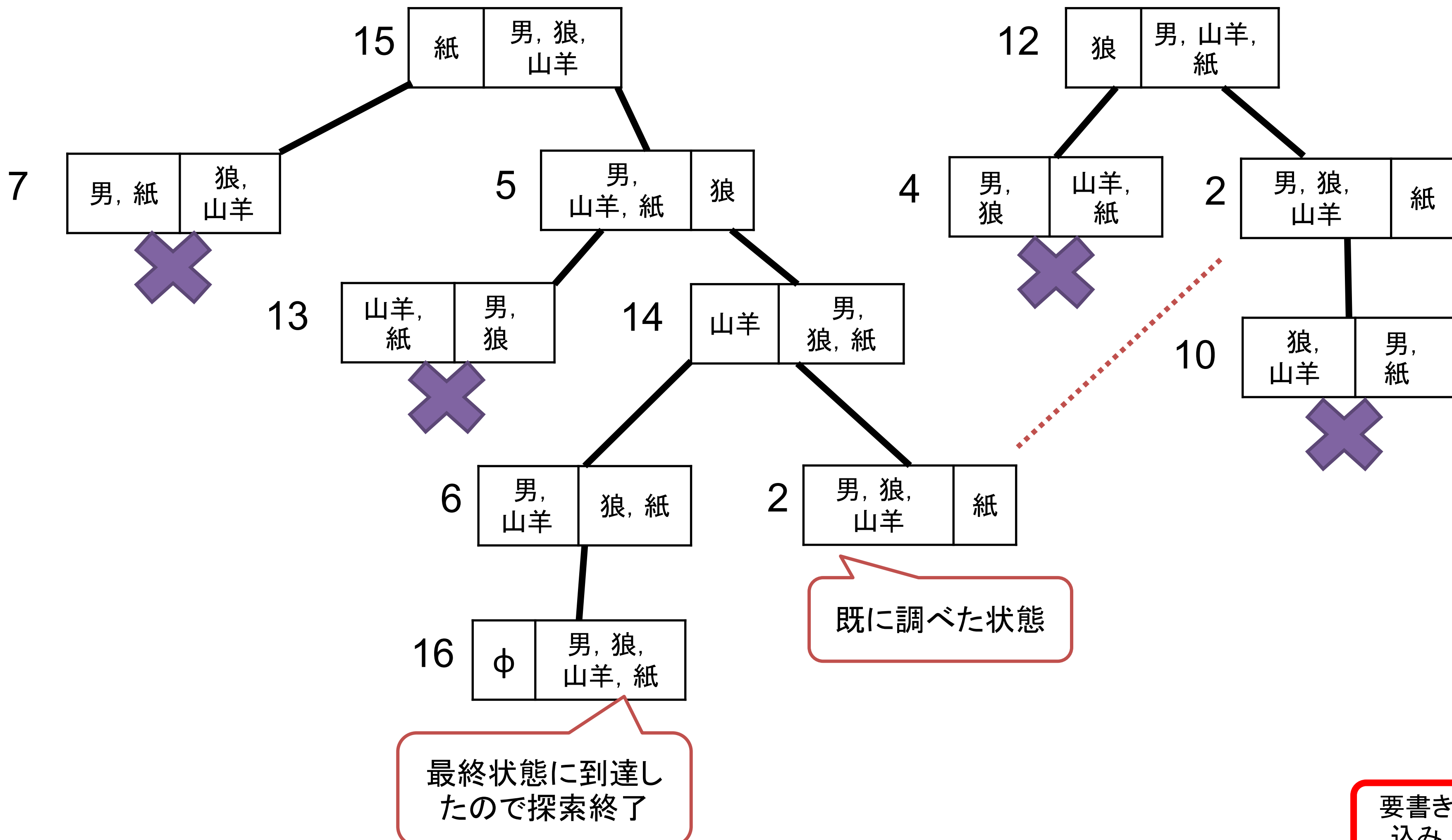
最終状態

幅優先探索の応用
探索木：状態遷移の探索（その1）



要書き込み

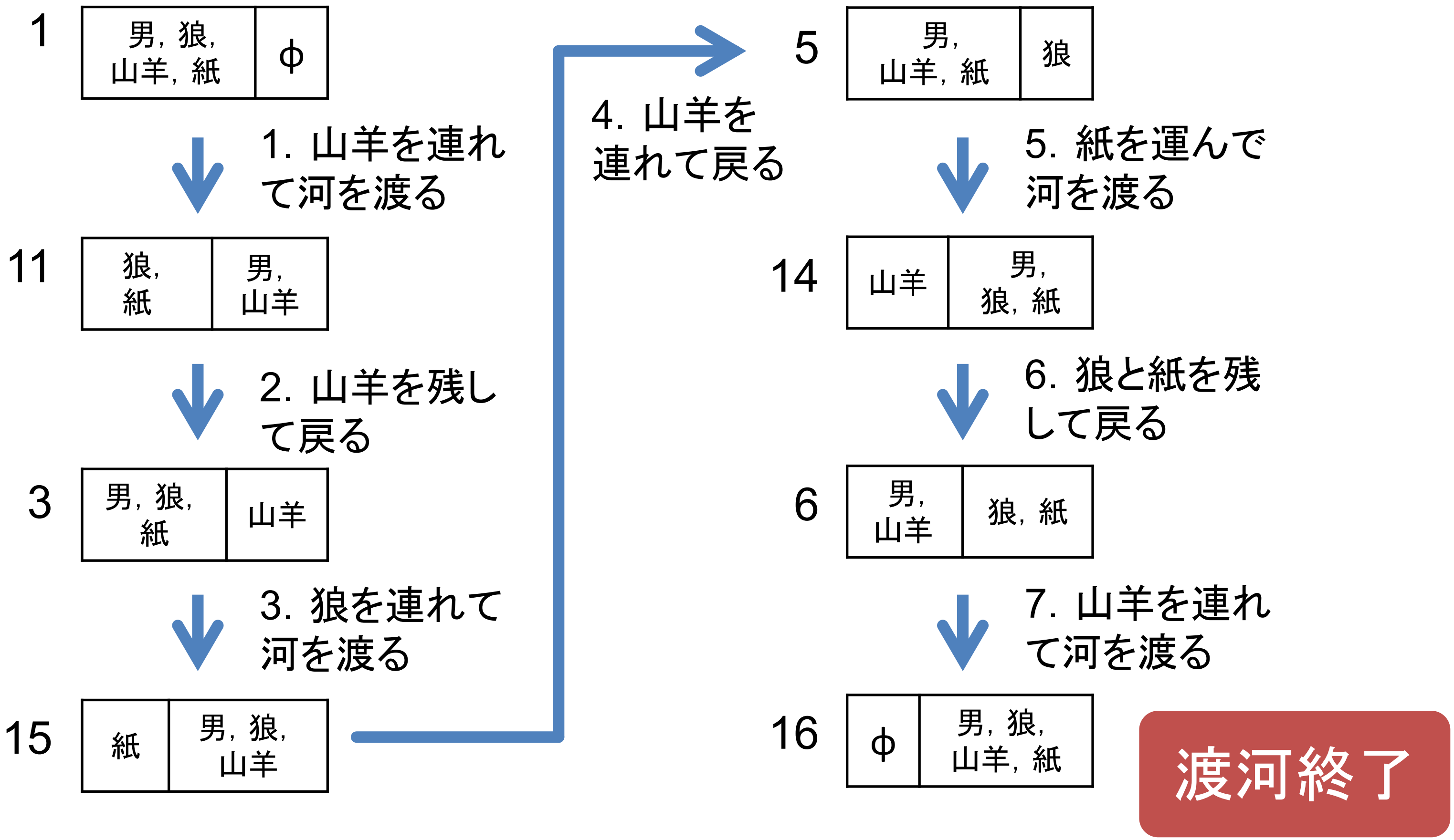
幅優先探索の応用
探索木：状態遷移の探索(その2)



要書き込み

幅優先探索の応用

探索木：渡河問題の解

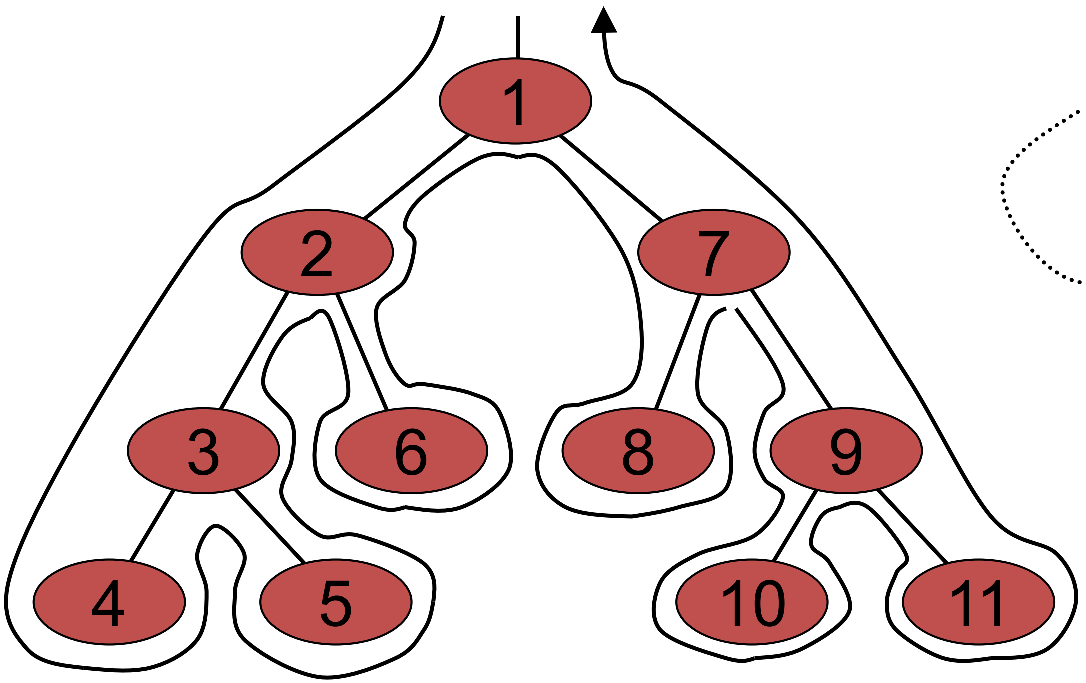


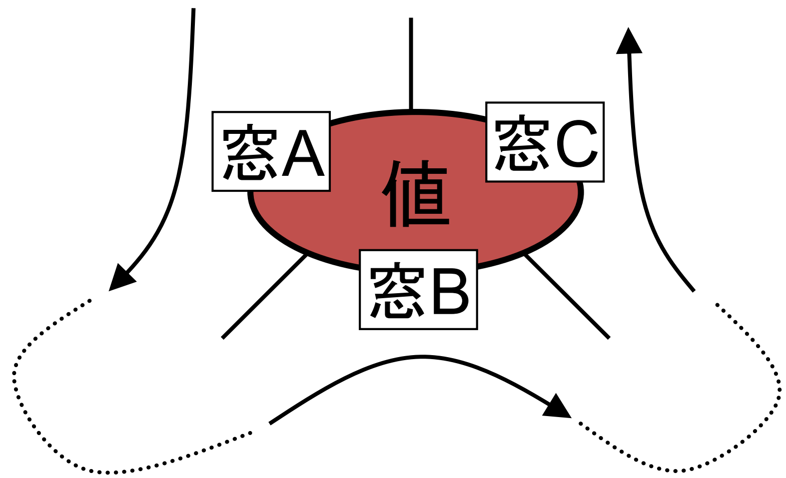
要書き込み

深さ優先探索の応用

木構造走査の概念

木構造とそれを巡る経路

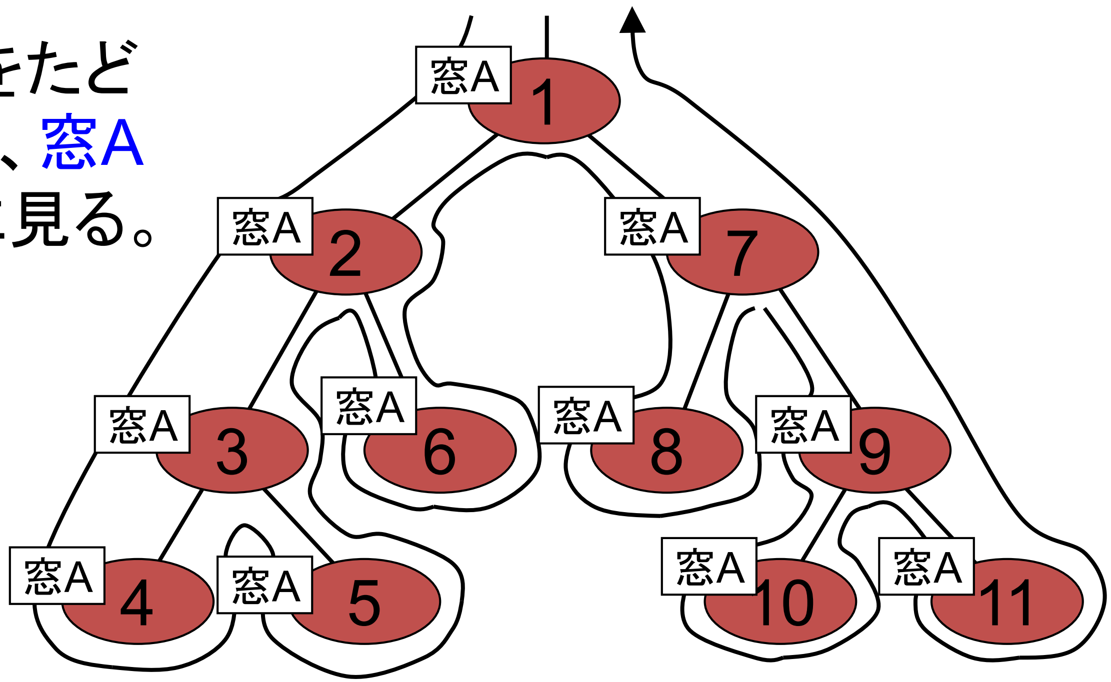




節点の構造と窓

深さ優先探索の応用
木構造の走査：Pre Order

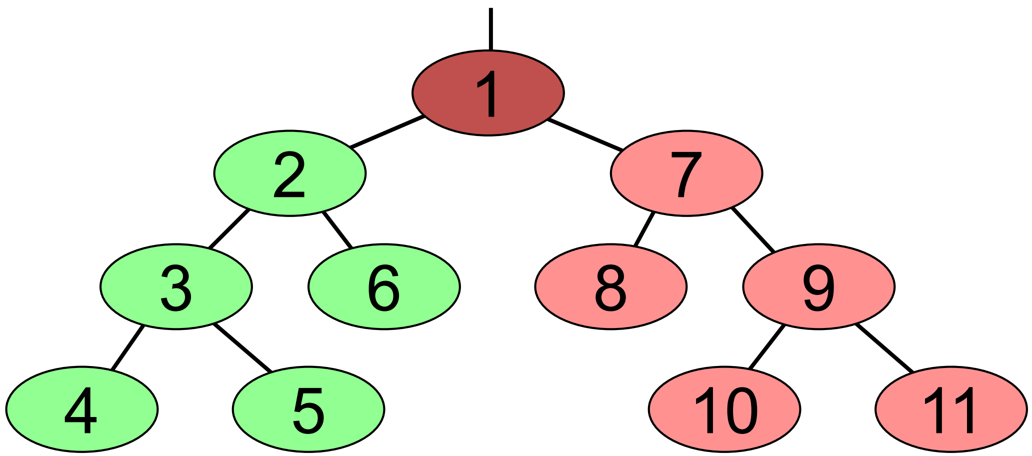
木構造をたどりながら、窓Aを順番に見る。



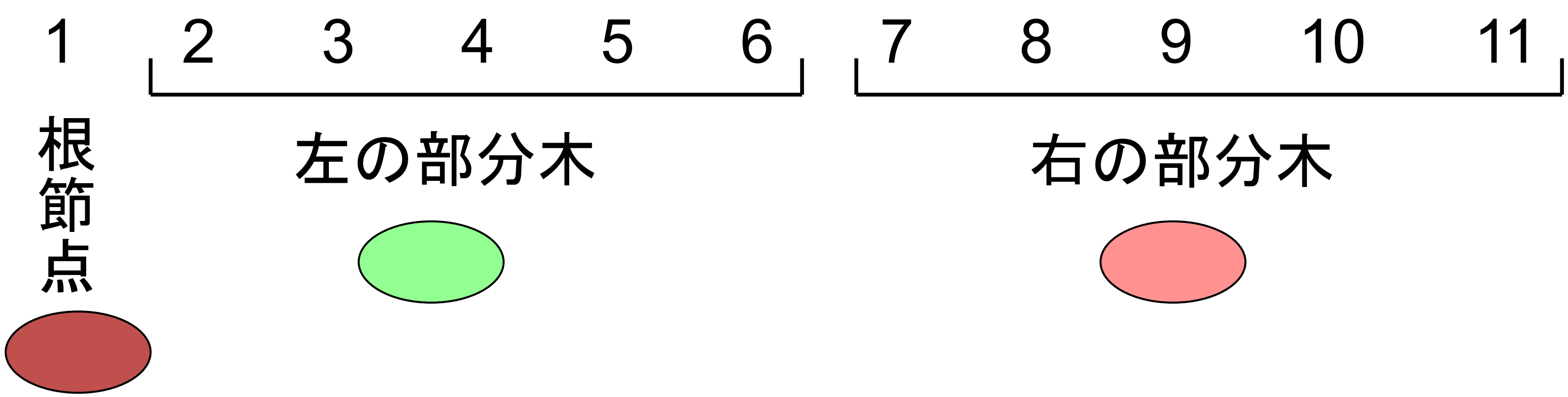
1　2　3　4　5　6　7　8　9　10　11

見えた順に節点を並べる。

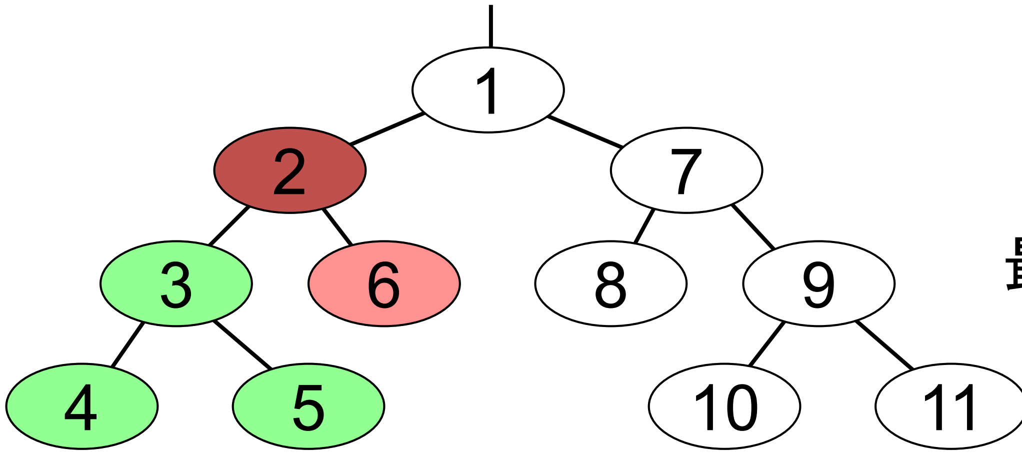
深さ優先探索の応用
Pre Orderの再帰構造(その1)



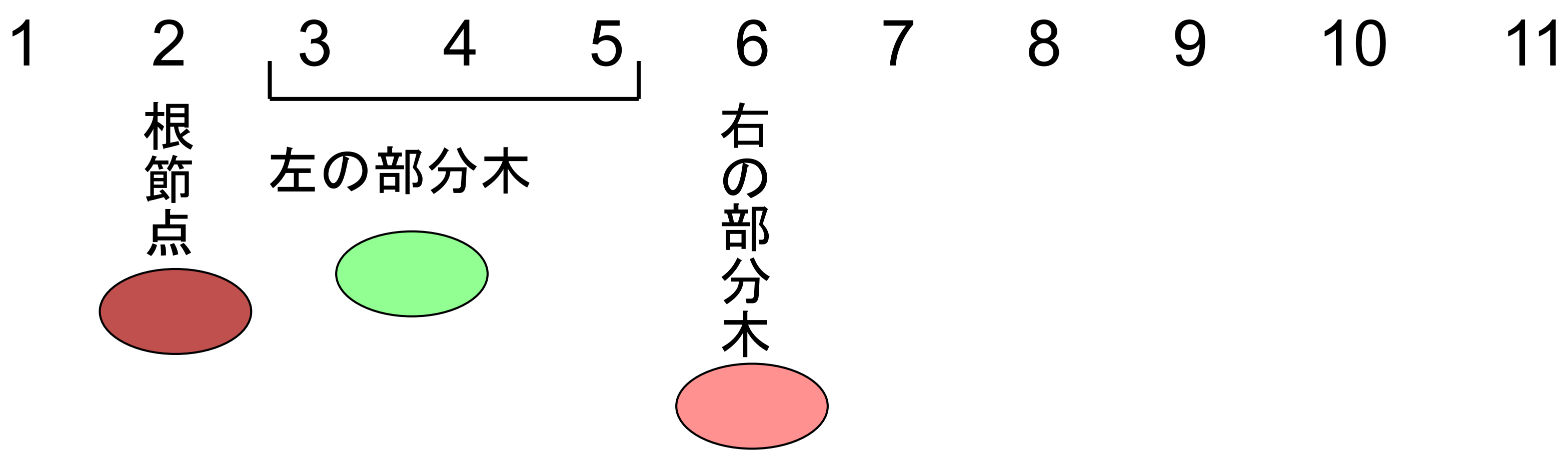
根節点を最初に見る。
次に左の部分木を見る。
最後に右の部分木を見る。



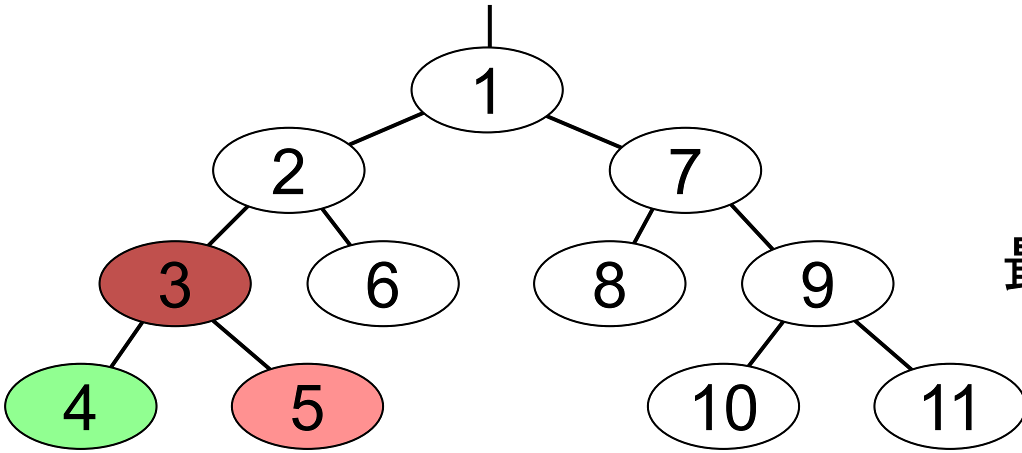
深さ優先探索の応用
Pre Orderの再帰構造(その2)



根節点を最初に見る。
次に左の部分木を見る。
最後に右の部分木を見る。



深さ優先探索の応用
Pre Orderの再帰構造（その3）



根節点を最初に見る。
次に左の部分木を見る。
最後に右の部分木を見る。

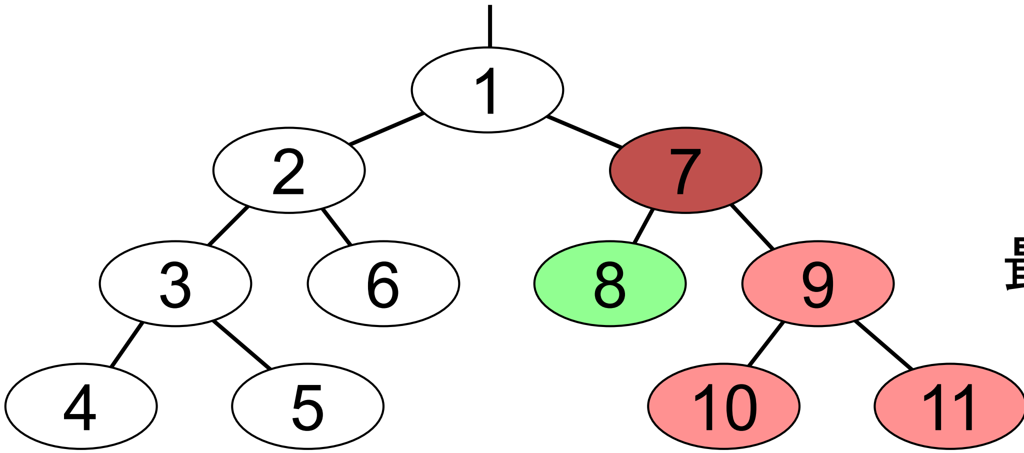
1 2 3 4 5 6 7 8 9 10 11

根節点

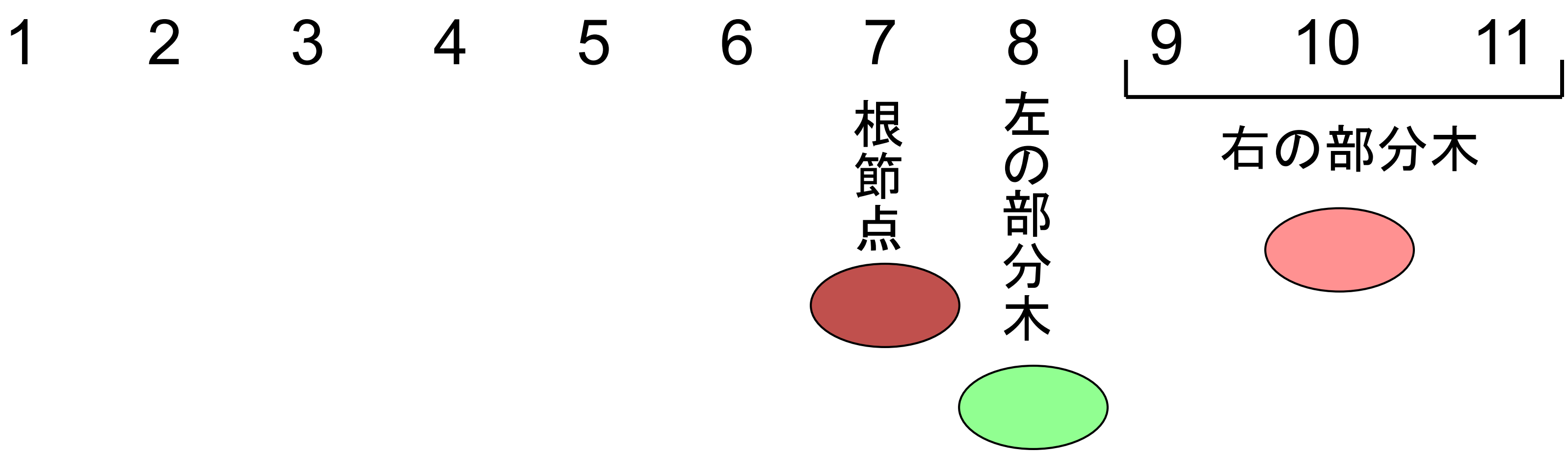
左の部分木

右の部分木

深さ優先探索の応用
Pre Orderの再帰構造(その4)

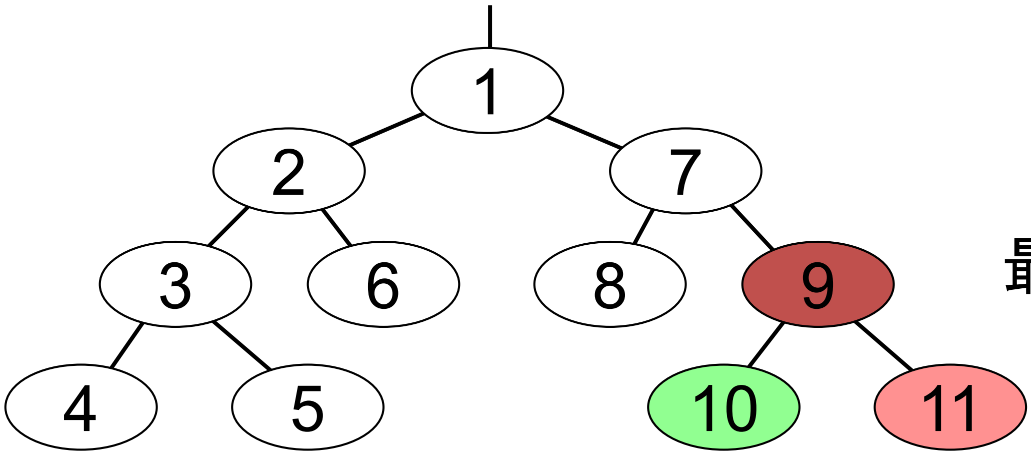


根節点を最初に見る。
次に左の部分木を見る。
最後に右の部分木を見る。



深さ優先探索の応用

Pre Orderの再帰構造(その5)



根節点を最初に見る。
次に左の部分木を見る。
最後に右の部分木を見る。

木構造の走査アルゴリズム
pre-orderの場合

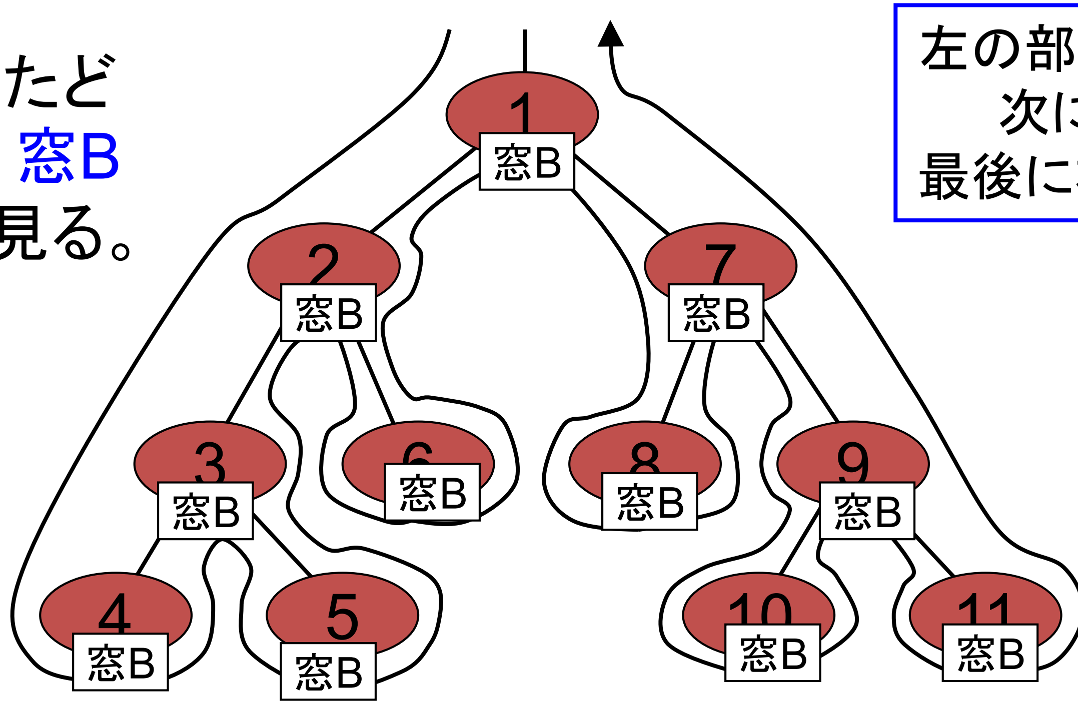
アルゴリズムpreOrder

1. 木が空ならば終了する。
2. そうでなければ、以下の処理を実行する。
   2.1 木の根節点を探索する。
   2.2 左の部分木を再帰的に探索する。
   2.3 右の部分木を再帰的に探索する。

要書き込み

木構造の走査:In Order

木構造をたどりながら、窓Bを順番に見る。



左の部分木を最初に見る。
次に根節点を見る。
最後に右の部分木を見る。

4 3 5 2 6 1 8 7 10 9 11

見えた順に節点を並べる。

木構造の走査アルゴリズム
in-orderの場合

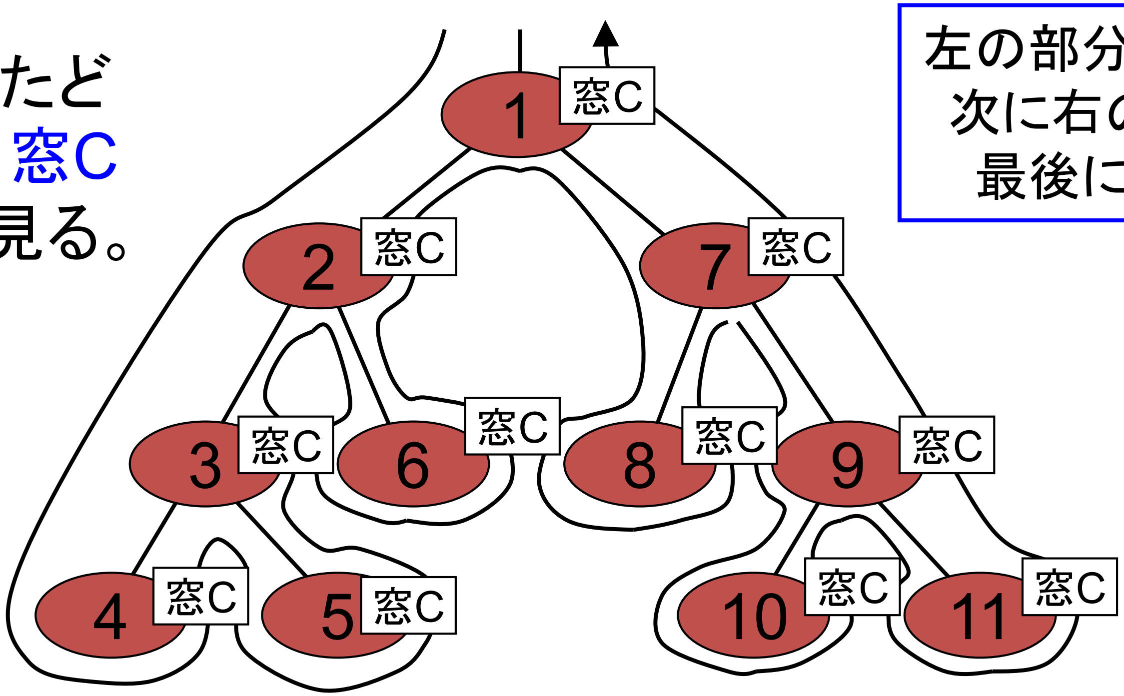
アルゴリズムinOrder
1. 木が空ならば終了する。
2. そうでなければ、以下の処理を実行する。
   2.1 左の部分木を再帰的に探索する。
   2.2 木の根節点を探索する。
   2.3 右の部分木を再帰的に探索する。

要書き込み

木構造の走査：Post Order

木構造をたどりながら、窓Cを順番に見る。

左の部分木を最初に見る。
次に右の部分木を見る。
最後に根節点を見る。



4 5 3 6 2 8 10 11 9 7 1

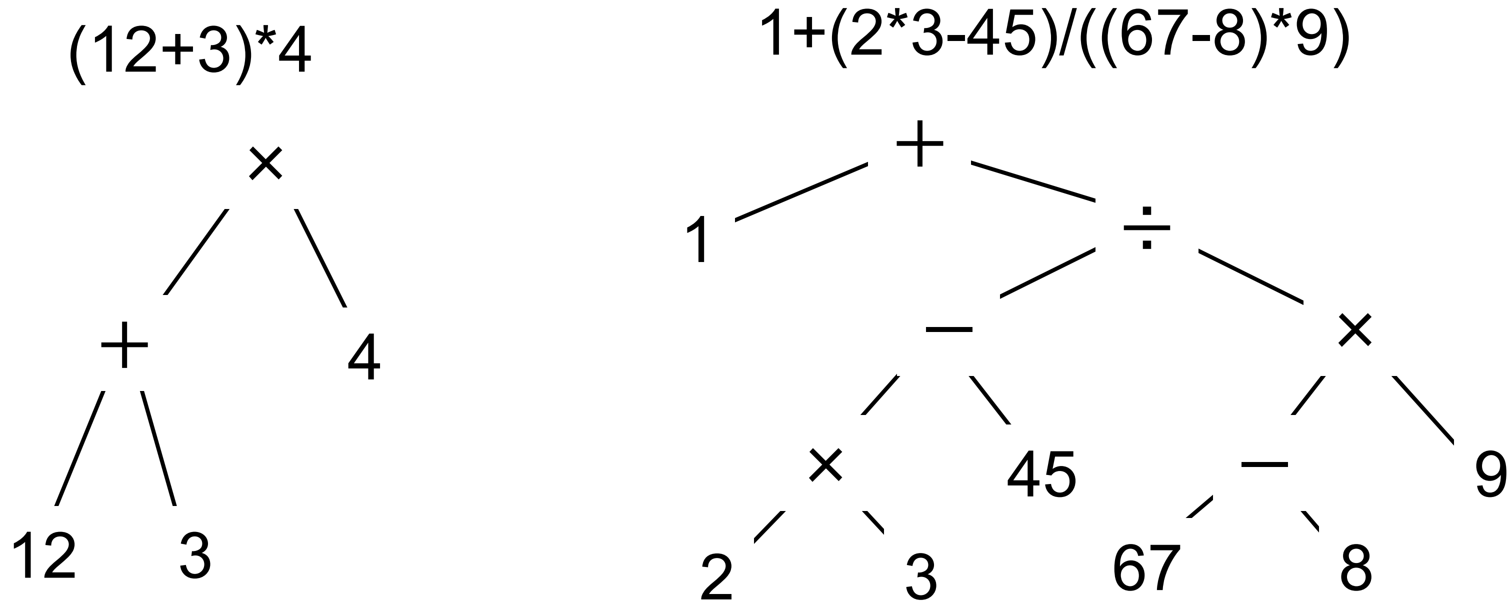
見えた順に節点を並べる。

木構造の走査アルゴリズム
post-orderの場合

アルゴリズムpostOrder
1. 木が空ならば終了する。
2. そうでなければ、以下の処理を実行する。
   2.1 左の部分木を再帰的に探索する。
   2.2 右の部分木を再帰的に探索する。
   2.3 木の根節点を探索する。

要書き込み

後置記法で表現された式



式の木構造をpost orderで走査すると、後置記法が得られる。

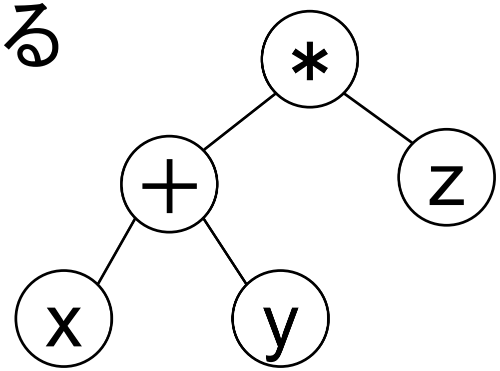
逆ポーランド記法（RPN, Reverse Polish Notation）

計算式の記法

- 演算子を置く場所で、3つの記法がある
  - 中置記法（infix notation）
    (x + y) * z
  - 前置記法（prefix notation）
    * (+ x y) z
  - 後置記法（postfix notation, 別名 逆ポーランド記法）
    (x y +)z *

関数呼び出しは前置記法で記述

後置記法の式はスタックを用いて簡単に計算できる



- 前置記法や後置記法では、括弧を省略可
  - 省略しても、式の意味が一意に定まる（曖昧さが無い）

後置記法の式の計算(その1)

後置記法の式 12 3 + 4 * の値を計算する。

入力	最初	12	3	+	4	*	最後
スタックの状態	空	12	3 12	15	4 15	60	空
出力							60

後置記法の式の計算（その2）

後置記法の式の値を計算する規則

トークンを順に読みながら，トークンの種類毎に以下の処理を行う.

トークン	トークンの処理
数値	スタックに数値をpushする。
演算子	1. スタックから2つの値をpopして、演算を行なう。 2. 演算結果をスタックにpushする。
トークンが残っていない	スタックから値をpopしてそれを出力する。 ※ 出力した値が式の値

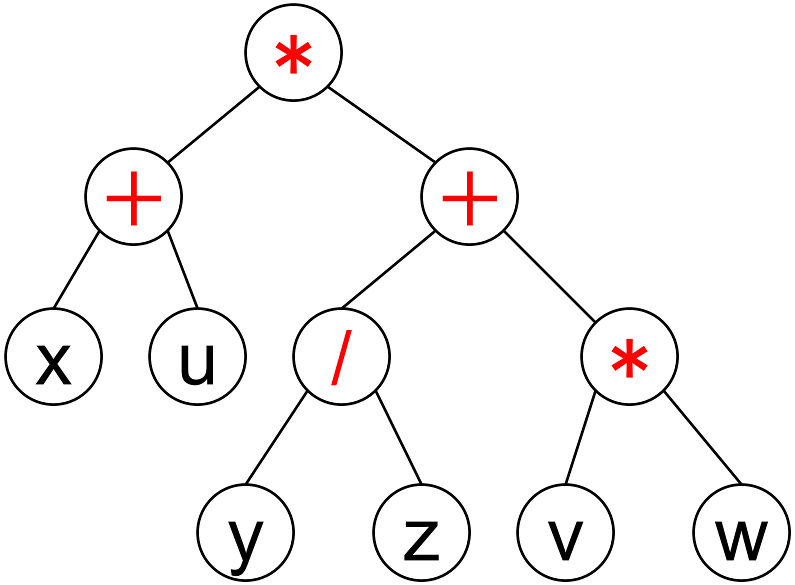
まとめ

グラフアルゴリズム

- グラフの定義
  - 無向グラフ，有向グラフ，重み付きグラフ
- グラフの表現
  - 隣接行列，隣接リスト
- グラフの探索
  - 幅優先探索（BFS），深さ優先探索（DFS）
- 探索の応用
  - 探索木
  - 前置記法，中置記法，後置記法

確認テスト（第10回）

1. 右の木を＊からスタートして幅優先で探索した場合、どの順番で頂点を訪れるか示せ
2. 右の木を計算木とみる。このとき、木が表している式を、前置記法、中置記法、後置記法でそれぞれ示せ



確認テスト（第10回）

- n人の女王（n Queen）問題
  - n×nのチェス盤と女王（Queen）のコマがn個ある．女王は，自分の位置から見て上下左右および斜め方向に好きなだけ移動できる．チェス盤上にn人の女王が互いに干渉しないように配置せよ．
- 例：n=5の場合の解の例

				Q
	Q			
			Q	
Q				
		Q		

n=5の場合の解を全て求める探索木を作成せよ．ただし，オレンジ色のマスのいずれかには女王を配置すること．