

# 部分グラフ集合を扱う DDアルゴリズム

川原 純

2022/7/29 版

# 本資料の目的

- 部分グラフ集合を扱う、DD (ZDD) を用いたアルゴリズムについて解説
  - DD そのものについても解説
- その他情報
  - [dd documents](#): DD に関する情報を集約
  - 部分グラフ集合を扱うDD アルゴリズム: 本資料
  - [DD ライブラリ入門](#): DD を扱うライブラリを紹介
  - [DD の再帰演算のカタログ](#): DD の再帰演算の一覧

# ZDD

- Zero-suppressed Binary Decision Diagram
- [Minato 1993] によって提案
- 集合族をコンパクトに効率良く記憶

## 集合族

{2, 3, 5}, {1, 2, 3, 4}, {1, 3}, {3, 6}, {2, 5, 6, 7},  
{1, 2, 6, 7}, {1, 6, 7}, {1, 2, 5, 7}, {2, 3, 6},  
{2, 5, 6, 7}, {1, 2, 4, 5, 6, 7}, {1, 4}, {1, 5, 6},  
{1, 2, 3, 5, 7}, {1, 2, 3, 6}, {1, 2}, {1, 6, 7},  
{1, 2, 4, 7}, {2, 5, 6, 7}, {1, 3, 4, 5, 6}, {1, 3},  
{5, 6, 7}, {1, 4, 5, 6, 7}, {3, 6, 7}, {3, 4, 7}, {1},  
{2}, {6, 7}, {1, 2, 5}, {7}, {2, 5, 7}, {2, 6},  
{1, 5, 7}, {3, 5, 7}, {1, 2, 6, 7}, {2, 3, 5, 6, 7},  
{2, 5}, {2, 3, 4, 6}, {}, {2, 3}, {1, 6}, {1, 2, 4},  
{2, 3, 5, 7}, {2, 3, 6, 7}, {3, 5, 6, 7}, {1, 5, 6},  
{3}, {2, 6, 7}, {3, 4}, {2, 4, 6, 7}, {1, 2, 3, 4},  
{2, 3, 5}, {1, 2, 3, 6, 7}, {1, 2, 3, 4, 6}, {5, 7},  
{5}, {2, 5, 6, 7}, {1, 3, 4, 6}, {1, 2, 5, 6},  
{2, 3, 4, 5, 6}, {3, 4, 5, 6}, {3, 4, 7}, {1, 5, 7},  
{3, 4, 5, 7}

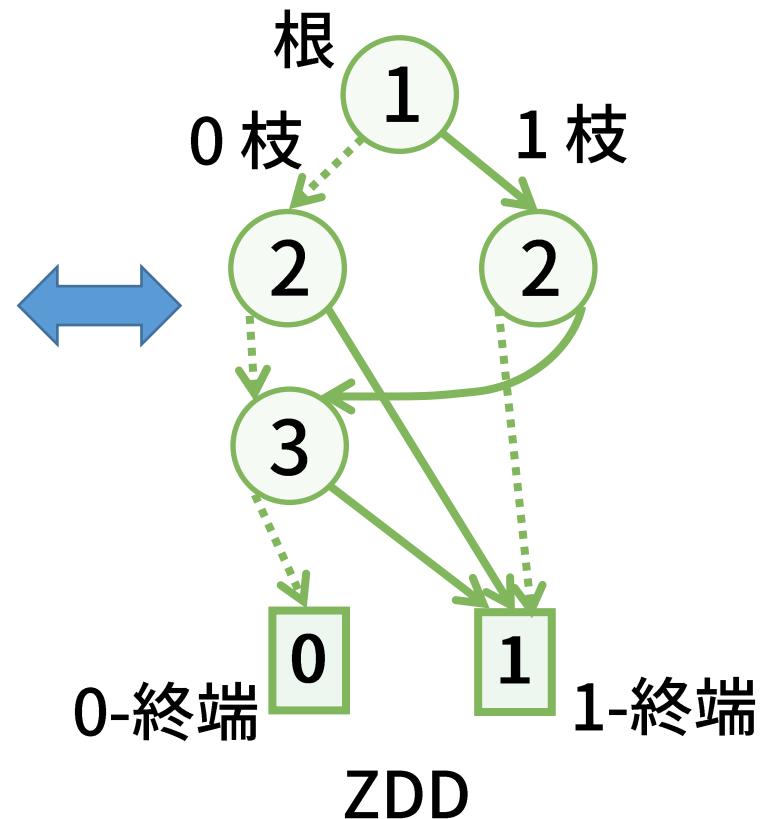
特殊な形をしたグラフ  
(directed acyclic graph)  
**ZDD**



# ZDD の読み方

{ {1}, {2}, {3}, {1, 2, 3} }

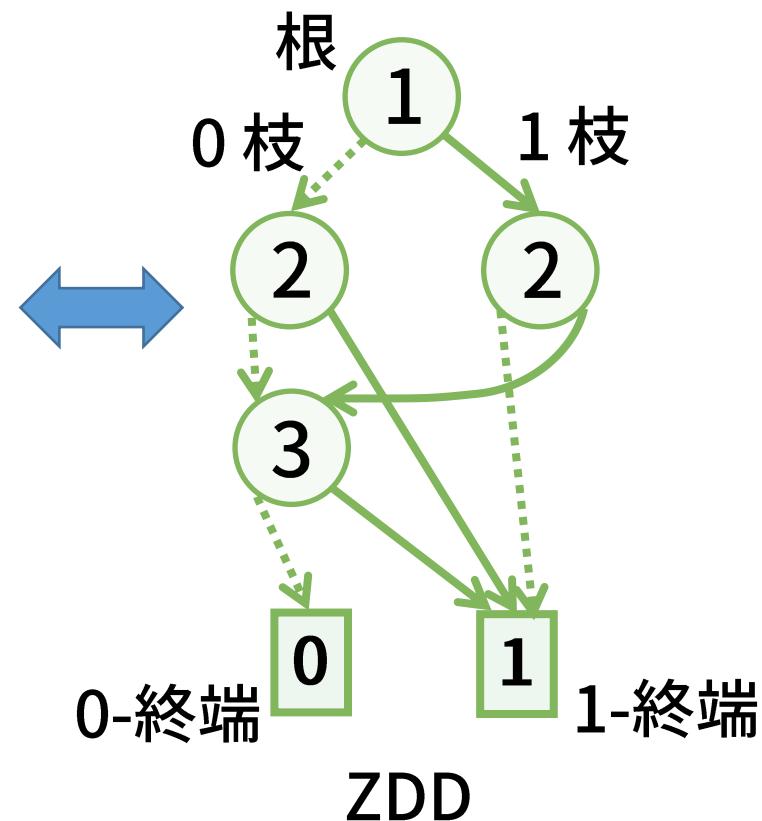
集合族



# ZDD の読み方

{ {1}, {2}, {3}, {1, 2, 3} }

集合族



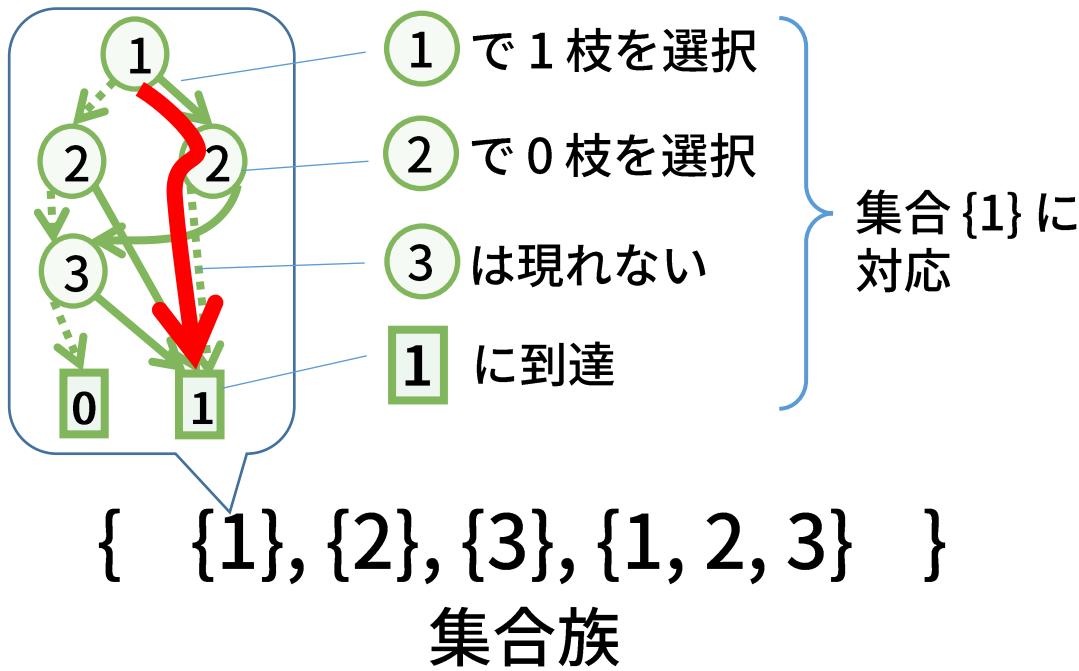
: ノード ラベル  $i$  をもつ  
0 枝と 1 枝を 1 個ずつもつ  
(点線) (実線)



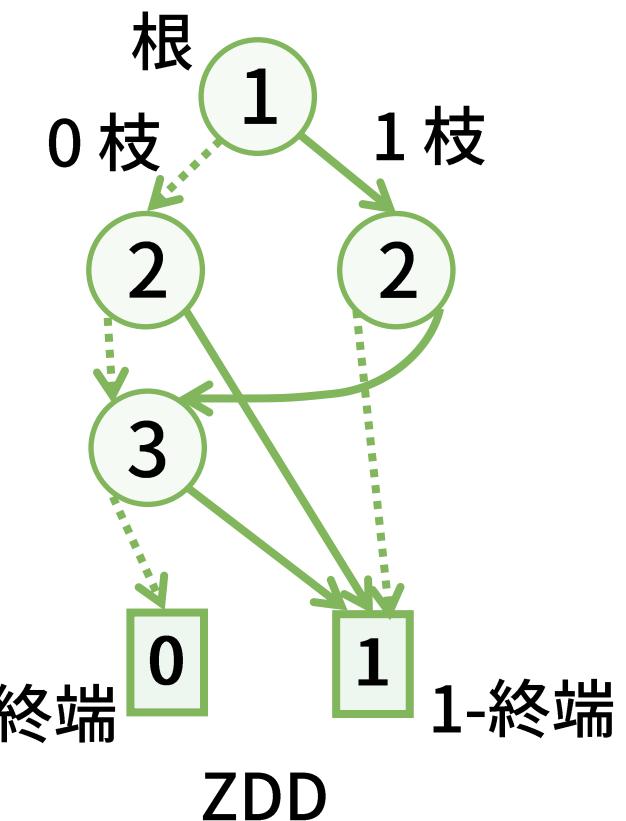
$i$  が  $j$  を指すなら  
 $i < j$

# ZDD の読み方

根から **1** までの経路1本が集合1つに対応



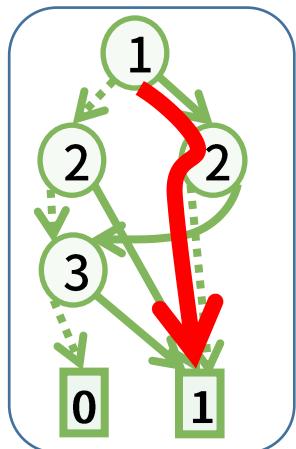
: ノード ラベル  $i$  をもつ  
0 枝と 1 枝を 1 個ずつもつ  
(点線) (実線)



$i$  が  $j$  を指すなら  
 $i < j$

# ZDD の読み方

根から **1** までの経路1本が集合1つに対応

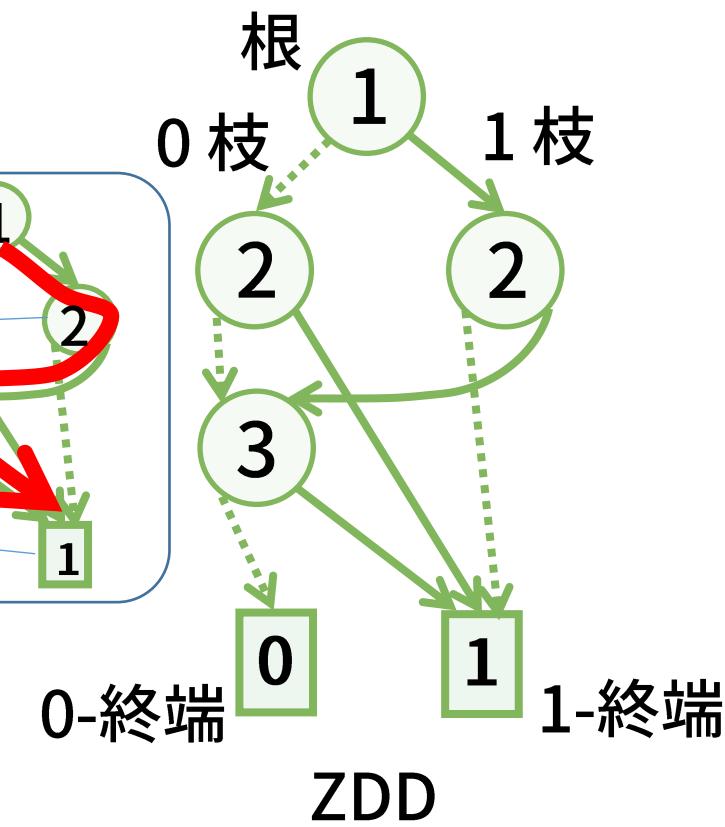
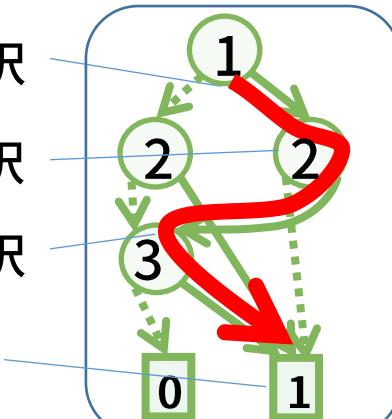


{ {1}, {2}, {3}, {1, 2, 3} }

集合族

集合  
 $\{1, 2, 3\}$  に対応

- 1 で 1 枝を選択
  - 2 で 1 枝を選択
  - 3 で 1 枝を選択
- 1 に到達



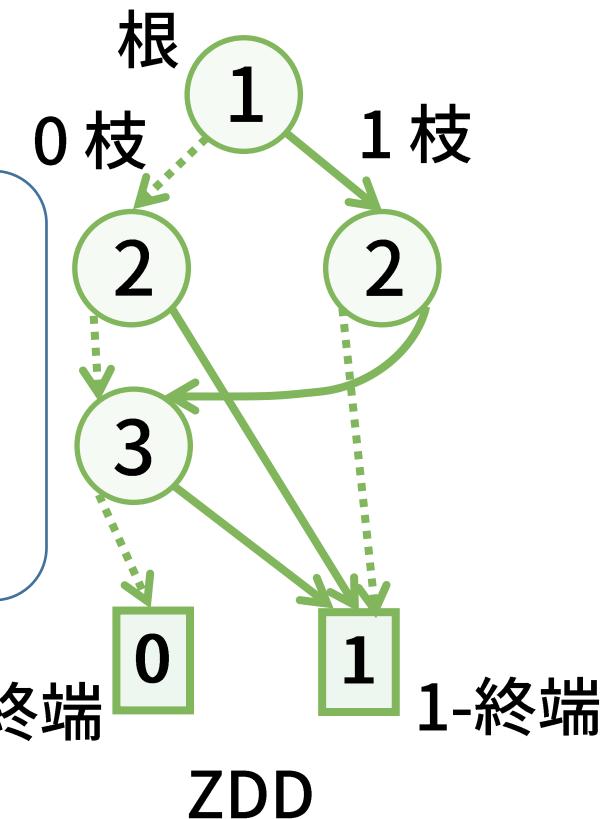
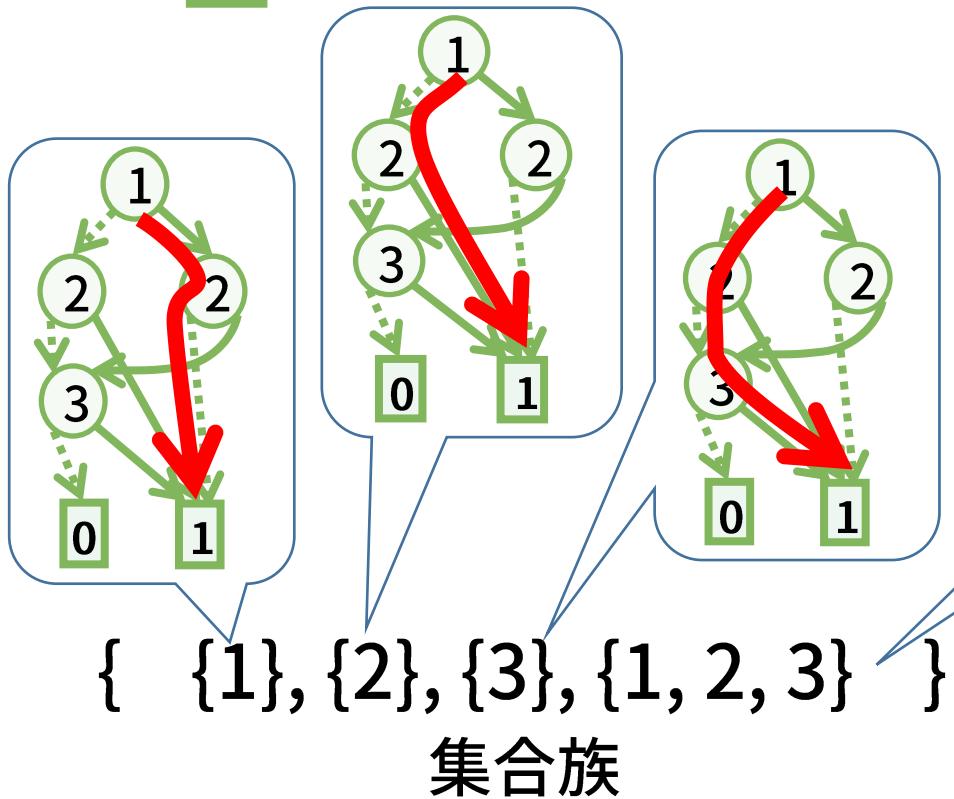
: ノード ラベル  $i$  をもつ  
0 枝と 1 枝を 1 個ずつもつ  
(点線) (実線)



$i$  が  $j$  を指すなら  
 $i < j$

# ZDD の読み方

根から **1** までの経路1本が集合1つに対応

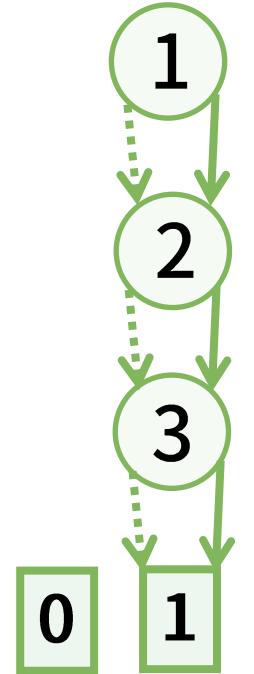


: ノード ラベル  $i$  をもつ  
0 枝と 1 枝を1個ずつもつ  
(点線) (実線)

$i$  が  $j$  を指すなら  
 $i < j$

## ZDD の特徴

- (ときには) 集合族を指数的に小さく圧縮できる
    - 応用で現れる集合族を小さく表せやすい
  - 集合族に対して、カノニカルな表現を持つ
    - 2つの ZDD が同じ集合族を表すかを根を指すポインタ同士の比較により可能
  - 集合族に対する豊富な「演算」 (family algebra)
    - 和集合、共通部分集合、super/subset、フィルタリン  
線形重み最大・最小化、ランダムサンプリングなど
    - カノニカル表現により効率的な演算を実現

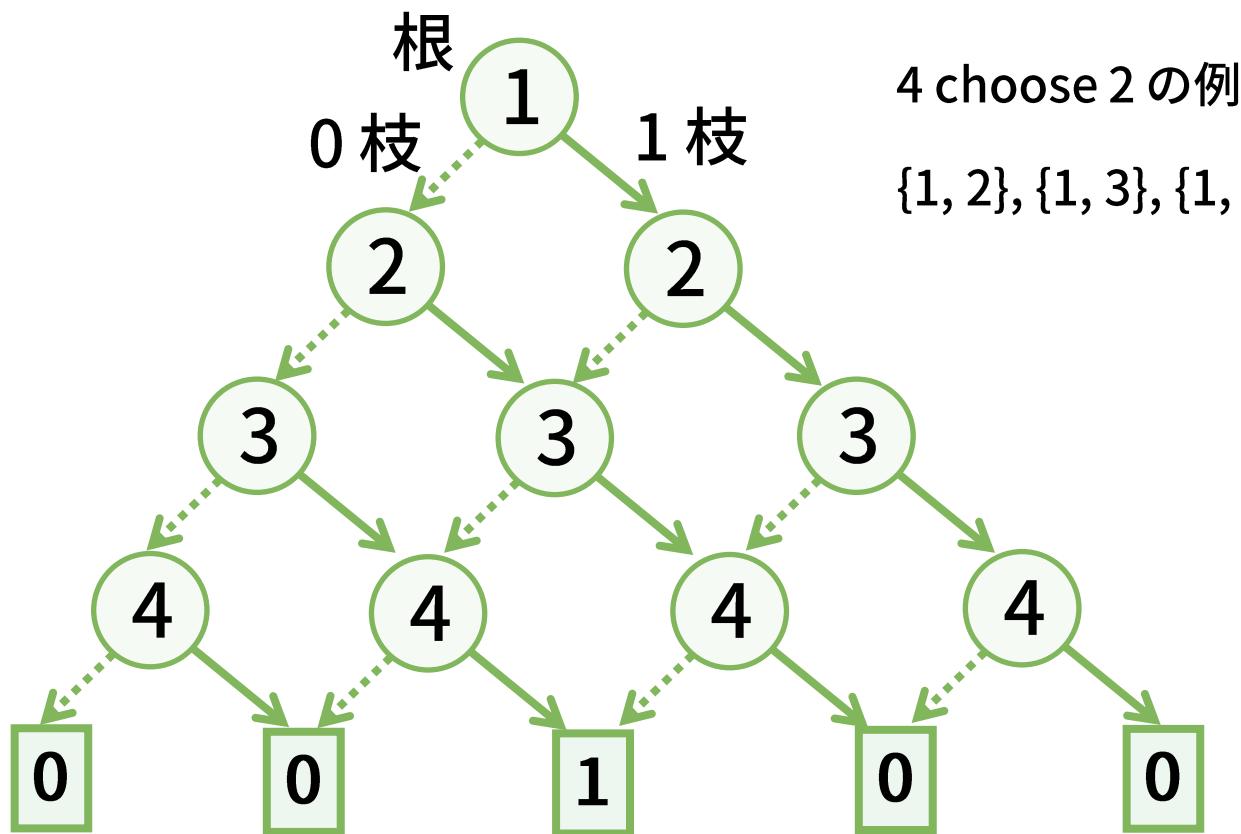


# ZDD の構築法

- ZDD の構築法を 2 つ紹介する
  - トップダウン構築法
  - ボトムダウン構築法

# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築



0-終端、1-終端は実際には1つずつしかない

# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

根から順番に作成



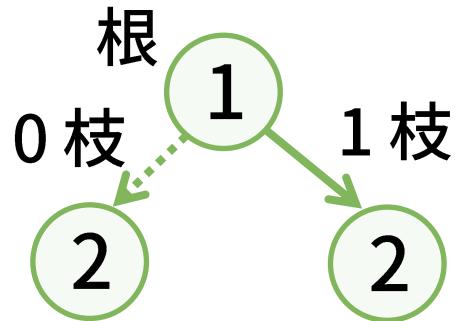
4 choose 2 の例

{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}

# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

根から順番に作成



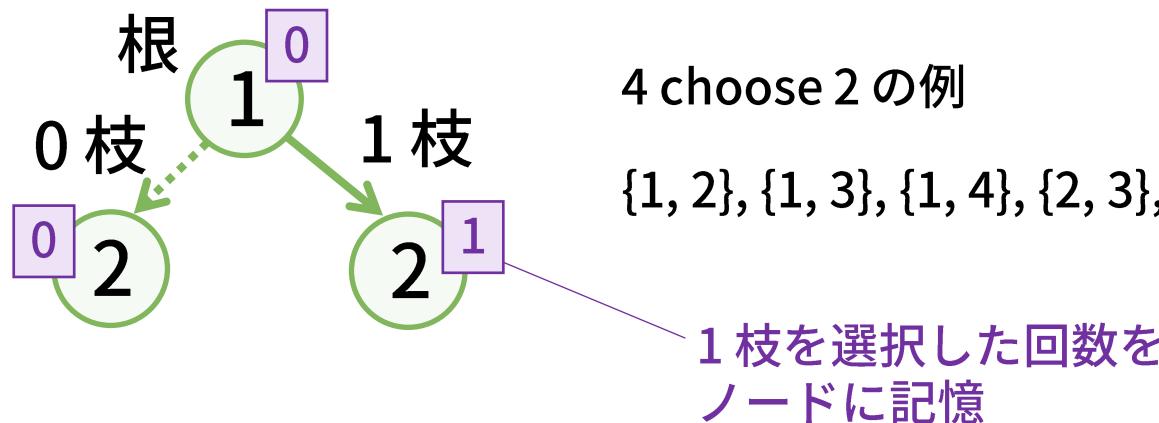
$4 \text{ choose } 2$  の例

$\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}$

# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

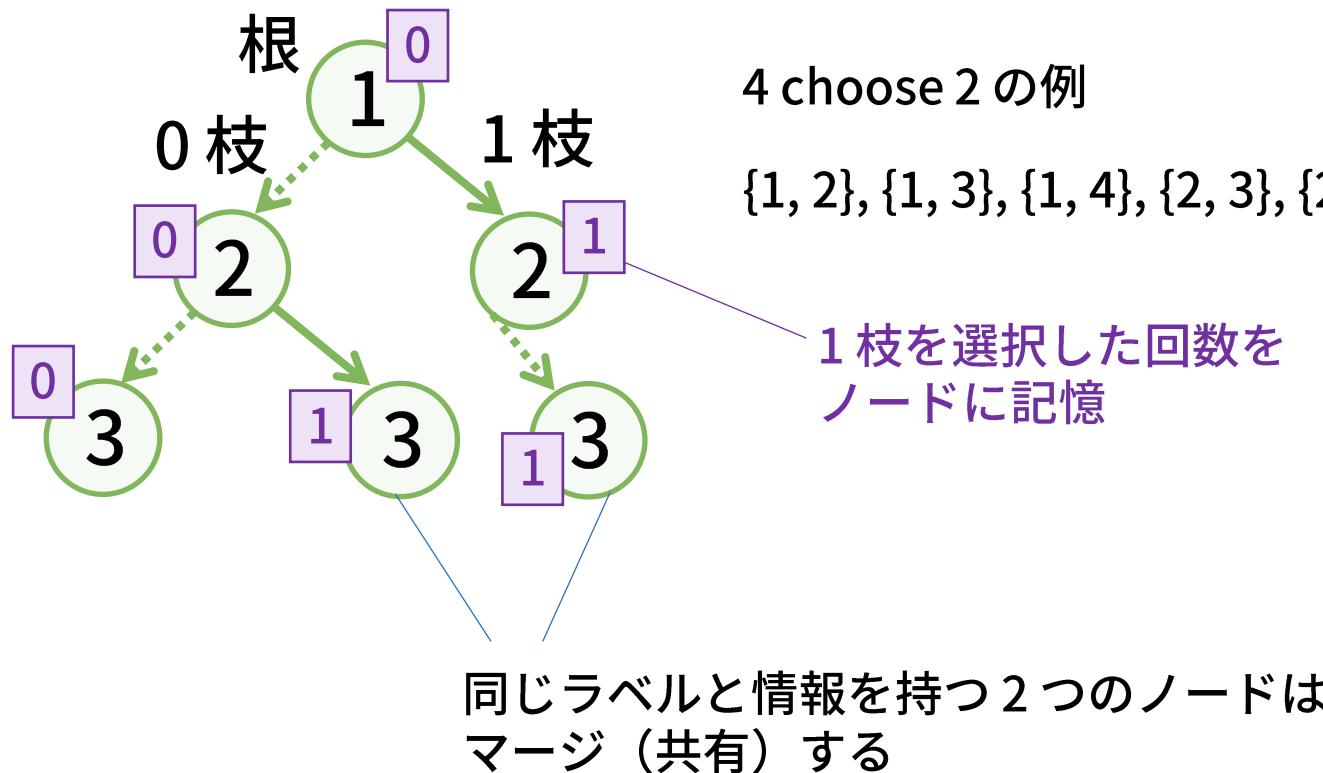
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

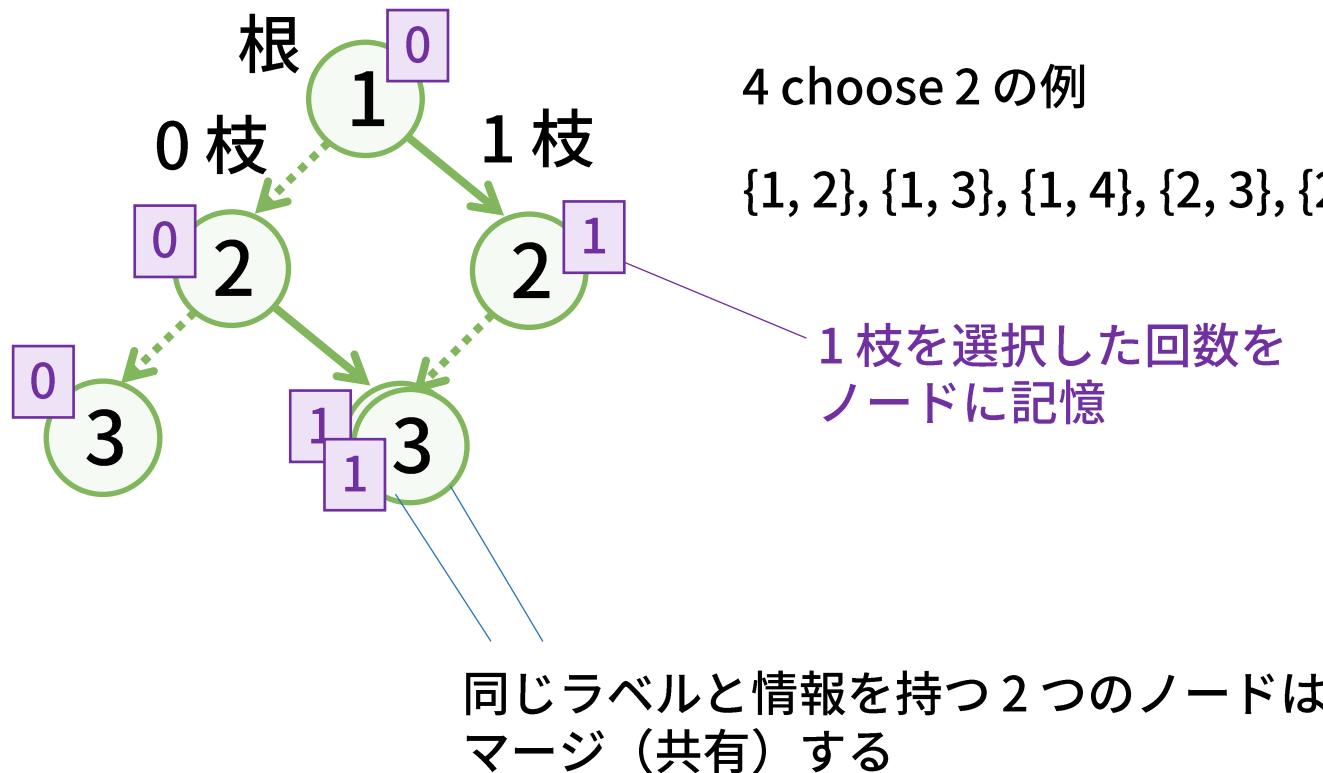
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

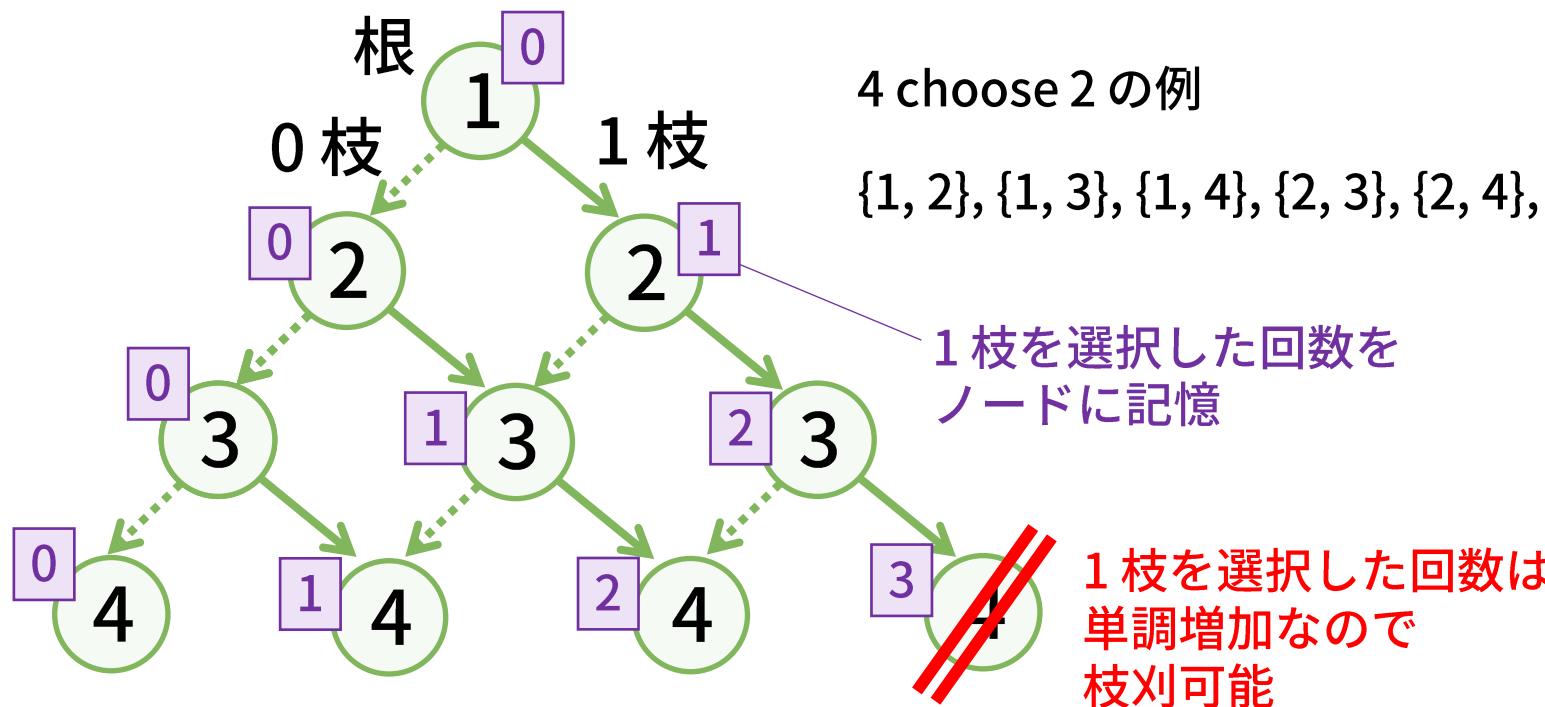
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

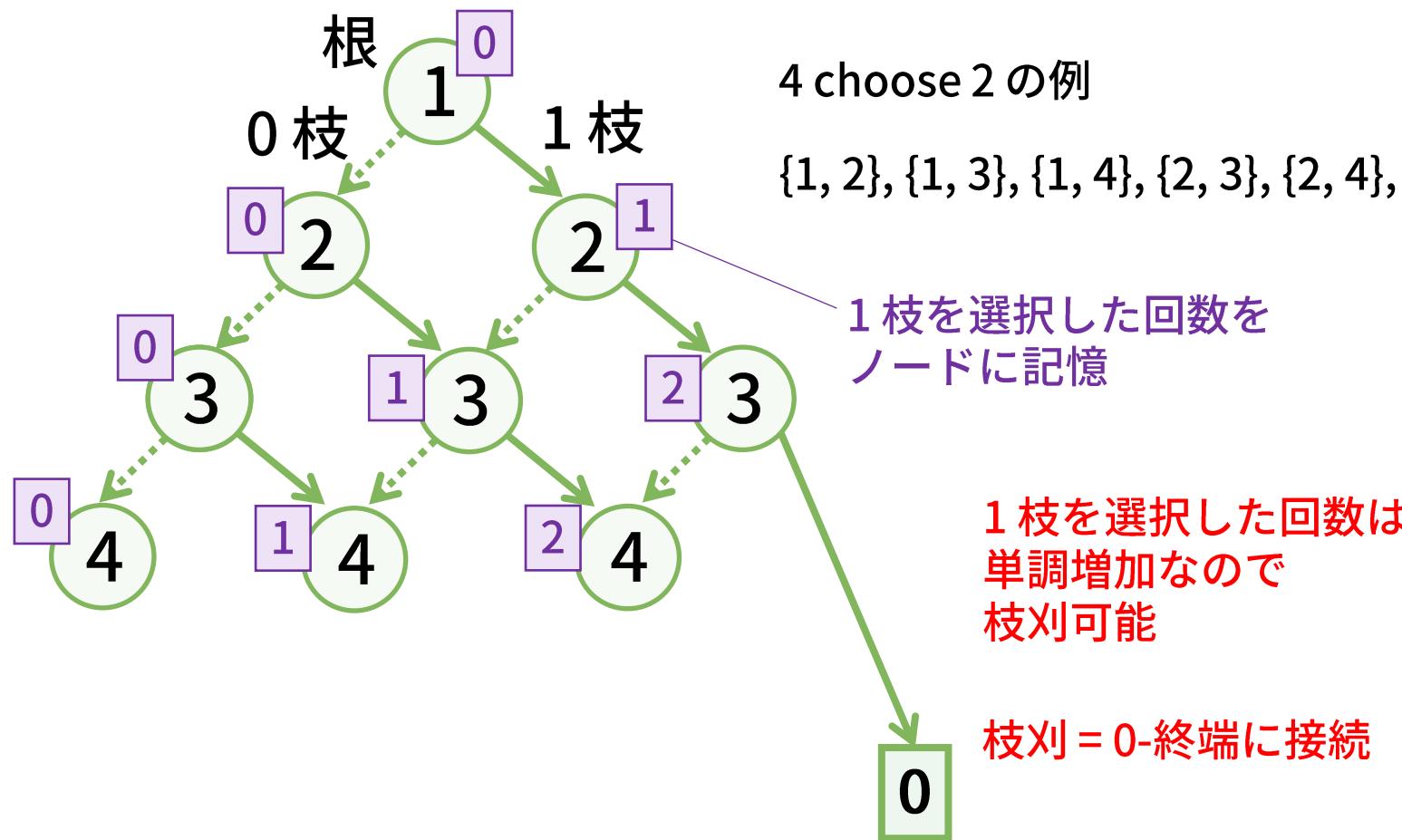
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

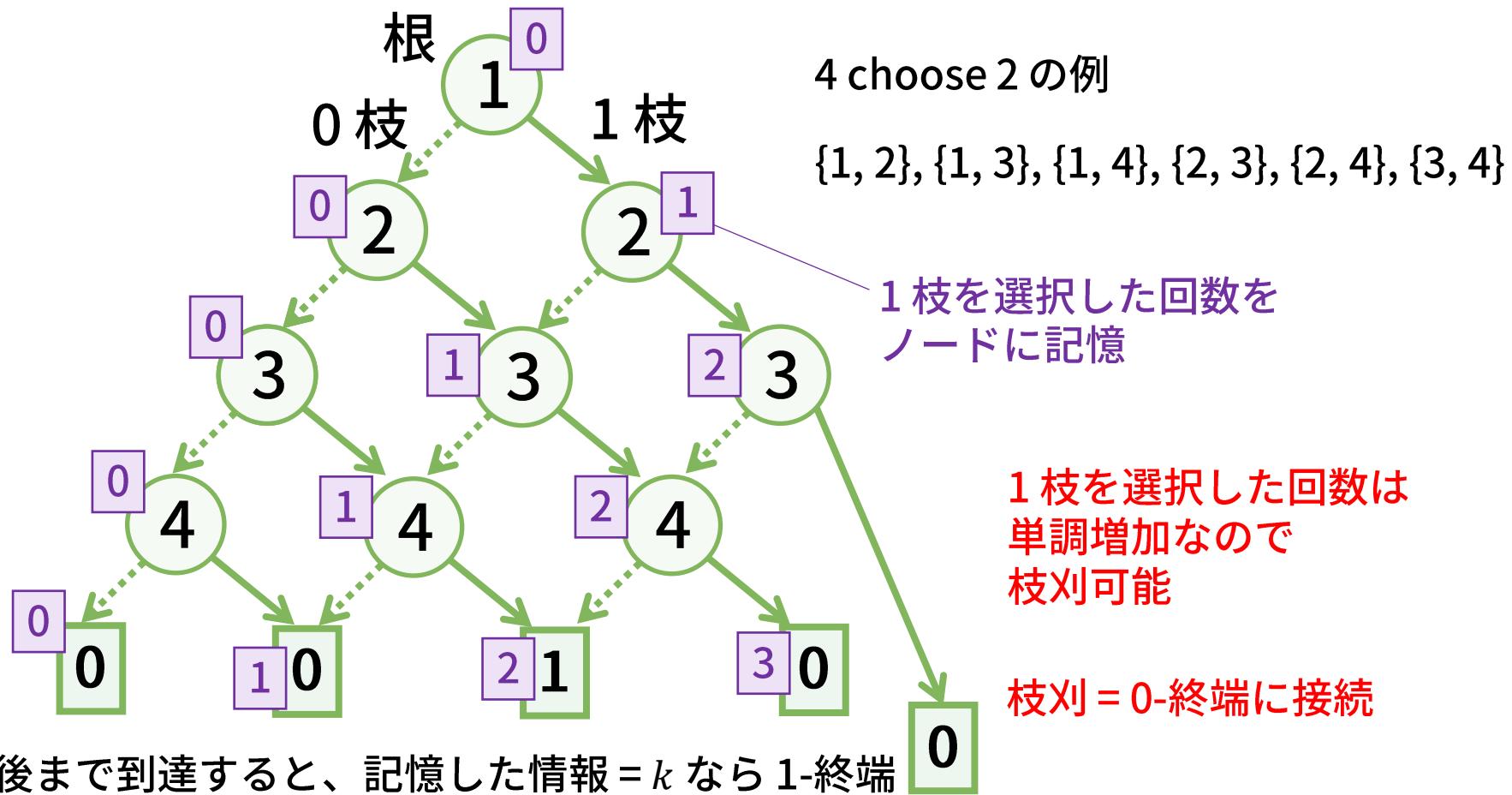
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

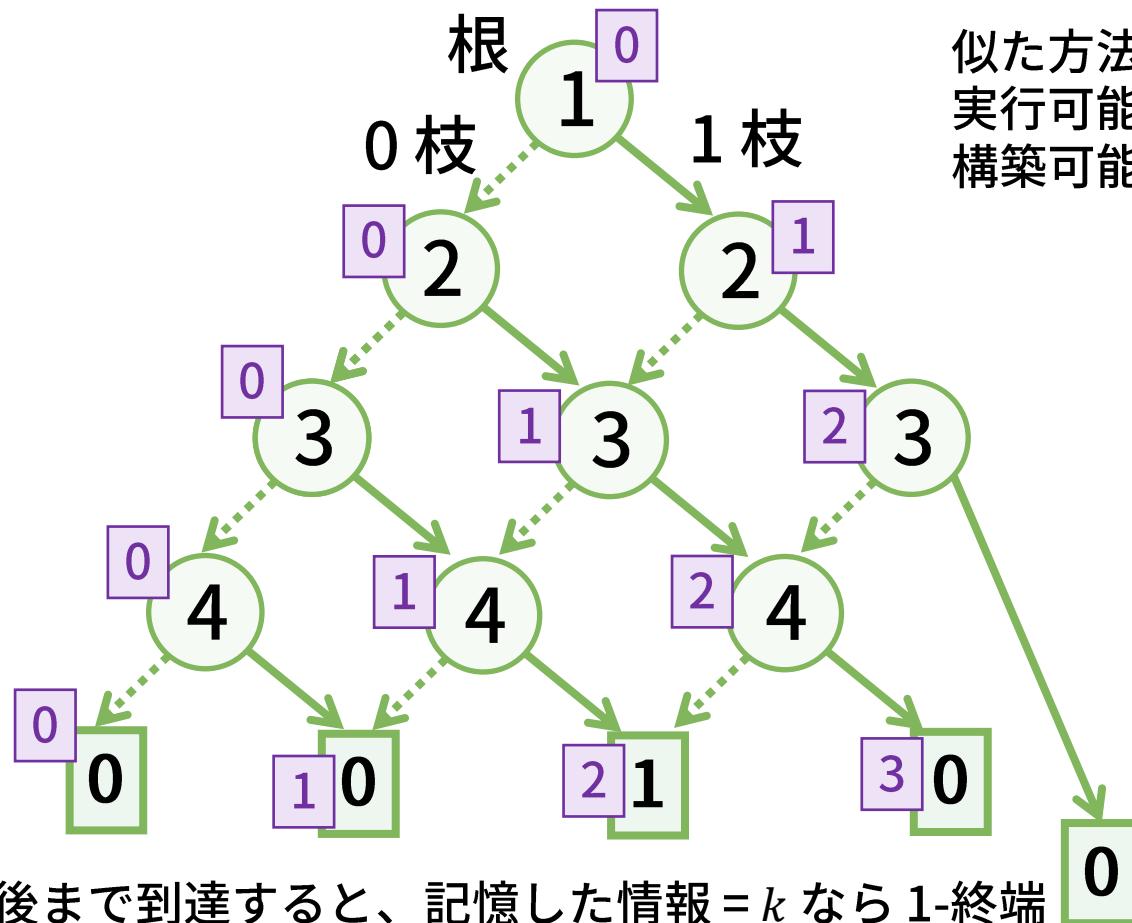
根から順番に作成



# ZDD の構築法: トップダウン

例:  $n \text{ choose } k$  ( $k$  個の要素からなる集合すべて) の集合族を表す ZDD 構築

根から順番に作成



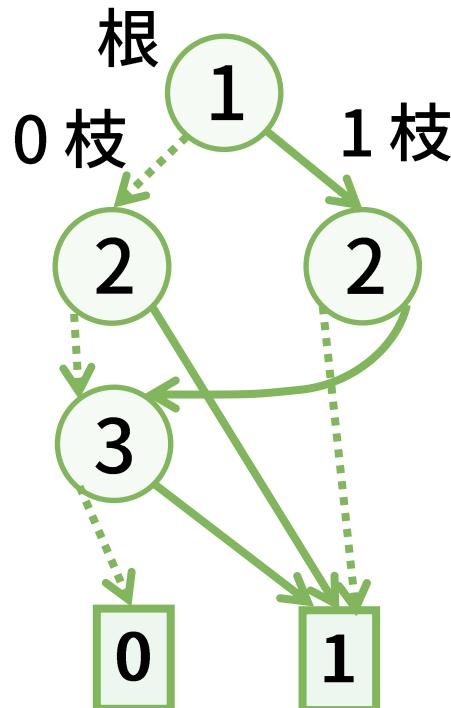
# ZDD の構築法: トップダウン

- トップダウン構築法のフレームワーク
  - ノードに何らかの情報を記憶させる
  - 2つのノードのラベルと情報が一致する場合、ノードのマージを行う
  - ノードの情報を基に枝刈を行う

正しくノードマージや枝刈が可能なように記憶させる情報を設計する

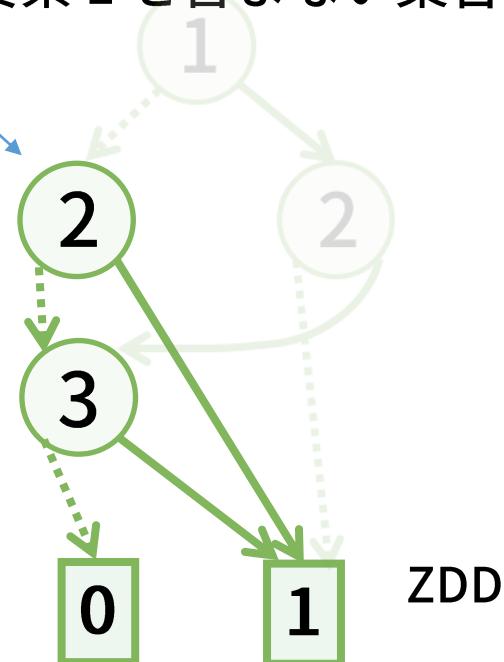
# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 根の 0 枝の先から到達可能なノードもまた ZDD とみなせる



{ {1}, {2}, {3}, {1, 2, 3} }

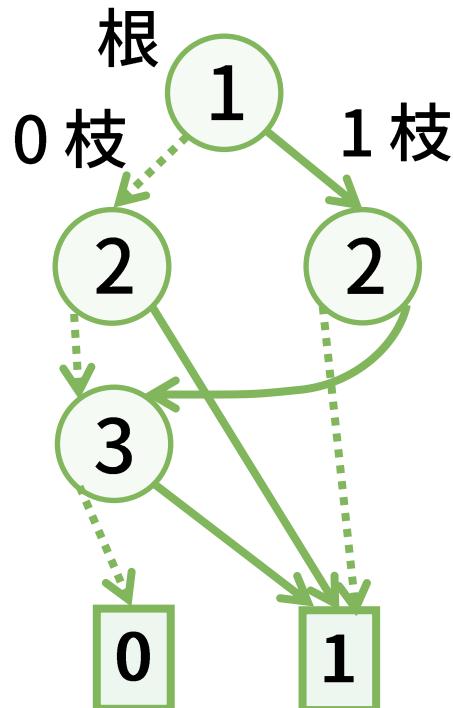
これは要素 1 を含まない集合の族



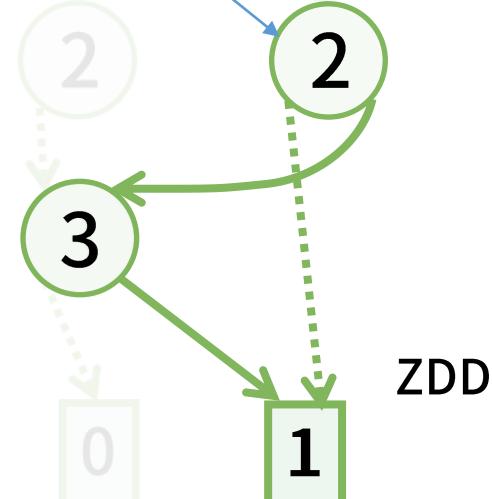
{ {2}, {3} }

# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 同様に、根の 1 枝の先から到達可能なノードもまた ZDD とみなせる



これは要素 1 を含む集合の族。  
ただし、要素 1 自体は削除される

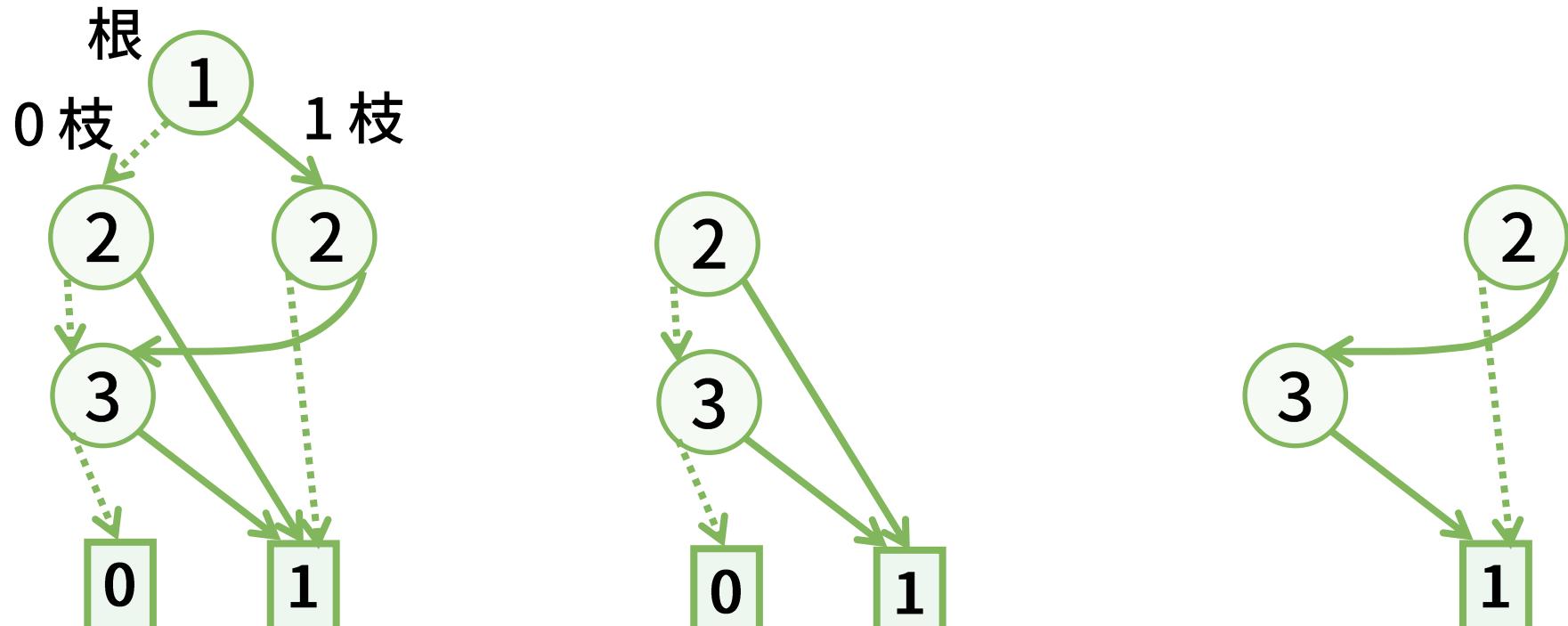


{ {1}, {2}, {3}, {1, 2, 3} }

{ {}, {2, 3} }

# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 集合族を 1 を含むものと  
含まないものに分解できる

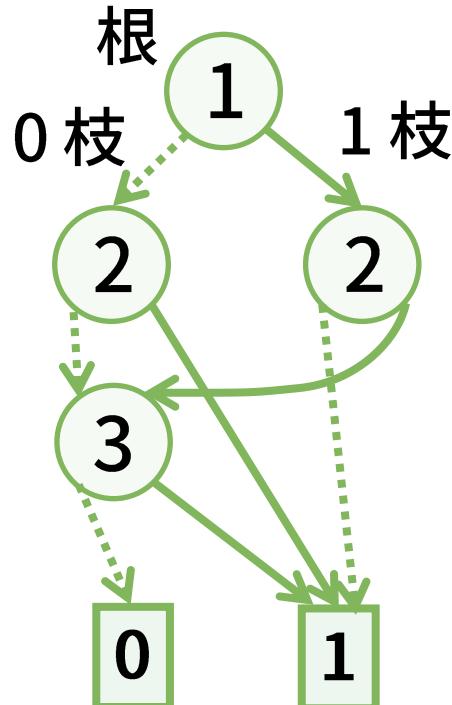


$$\{\{1\}, \{2\}, \{3\}, \{1, 2, 3\}\} = \{\{2\}, \{3\}\} \cup \{\{1\}\} \times \{\{\}, \{2, 3\}\}$$

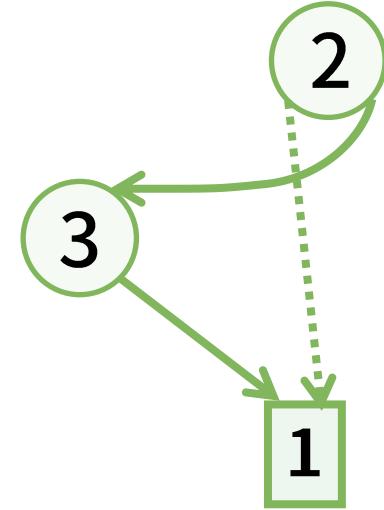
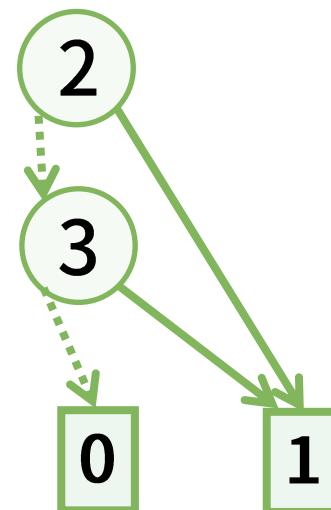
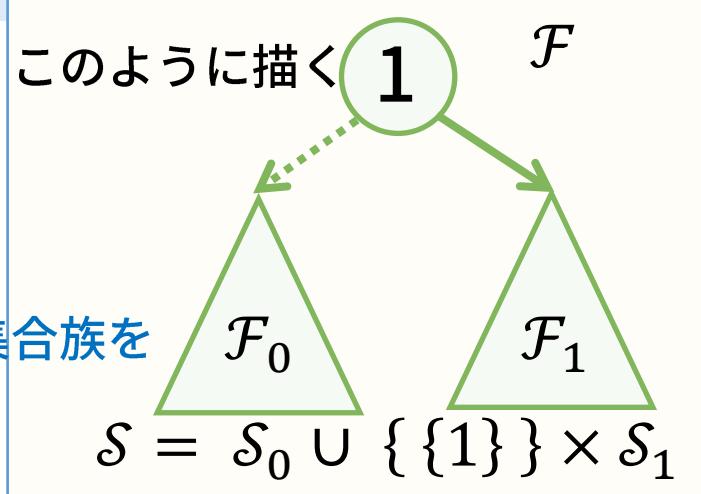
演算子導入 24

# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 集合族を 1 を含むものと  
含まないものに分解できる



$\mathcal{F}, \mathcal{F}_0, \mathcal{F}_1$  が表す集合族を  
 $\mathcal{S}, \mathcal{S}_0, \mathcal{S}_1$  とする



$$\{ \{1\}, \{2\}, \{3\}, \{1, 2, 3\} \} = \{ \{2\}, \{3\} \} \cup \{ \{1\} \} \times \{ \{ \}, \{2, 3\} \}$$

演算子導入 25

# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 再帰構造の末端



{ {3} }

= { }

U

{ {3} } × { {} }

空集合  
Ø とも書く

3 を含まない集合は  
無いので、空集合になる

空集合1つだけから  
なる集合族  
{Ø} とも書く

3 を含む集合は {3} だけ。  
要素 3 を削除すると Ø になる

# ZDD の構築法: ボトムアップ: ZDD の再帰構造

- 再帰構造の末端



$$\{ \emptyset, \{3\} \} = \{\emptyset\} \cup \{ \{3\} \} \times \{\emptyset\}$$

# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

- 2つの集合族の共通部分を計算

$$\begin{aligned} & \{\{1\}, \{2\}, \{3\}, \{1, 2, 3\}\} \cap \{\{1\}, \{1, 2\}, \{1, 2, 3\}\} \\ &= \{\{1\}, \{1, 2, 3\}\} \end{aligned}$$

# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

- 2つの集合族の共通部分を計算

$$\begin{aligned} & \{\{1\}, \{2\}, \{3\}, \{1, 2, 3\}\} \cap \{\{1\}, \{1, 2\}, \{1, 2, 3\}\} \\ &= \{\{1\}, \{1, 2, 3\}\} \end{aligned}$$

考え方: 再帰構造を用いる

$\mathcal{S}_0$

1を含まない集合

$\{\{1\}\} \times \mathcal{S}_1$

1を含む集合

$\mathcal{S}$

$\mathcal{T}_0$

1を含まない集合

$\{\{1\}\} \times \mathcal{T}_1$

1を含む集合

$\mathcal{T}$

$\mathcal{S}_0 \cap \mathcal{T}_0$

1を含まない集合

$\{\{1\}\} \times (\mathcal{S}_1 \cap \mathcal{T}_1)$

1を含む集合

$\mathcal{S} \cap \mathcal{T}$

# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

集合族  $\mathcal{S}, \mathcal{T}$  を表す ZDD を  $\mathcal{F}, \mathcal{G}$  とする

- $\text{Intersec}(\mathcal{F}, \mathcal{G})$ :  $\mathcal{S} \cap \mathcal{T}$  を表す ZDD を構築

$\mathcal{S}_0$

1を含まない集合

$\{\{1\}\} \times \mathcal{S}_1$

1を含む集合

$\mathcal{S}$

$\mathcal{T}_0$

1を含まない集合

$\{\{1\}\} \times \mathcal{T}_1$

1を含む集合

$\mathcal{T}$

$\mathcal{S}_0 \cap \mathcal{T}_0$

1を含まない集合

$\{\{1\}\} \times (\mathcal{S}_1 \cap \mathcal{T}_1)$

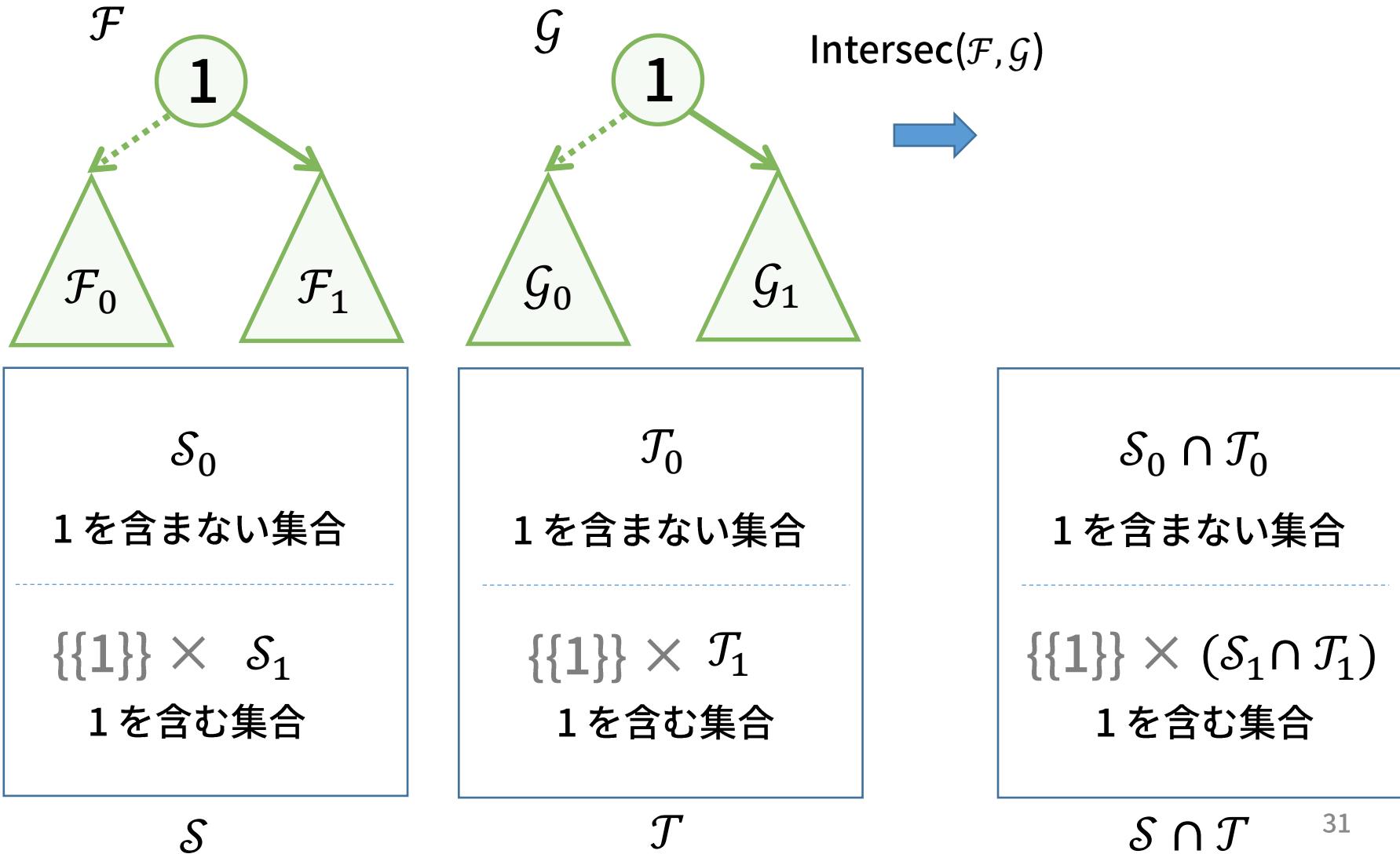
1を含む集合

$\mathcal{S} \cap \mathcal{T}$

# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

集合族  $\mathcal{S}, \mathcal{T}$  を表す ZDD を  $\mathcal{F}, \mathcal{G}$  とする

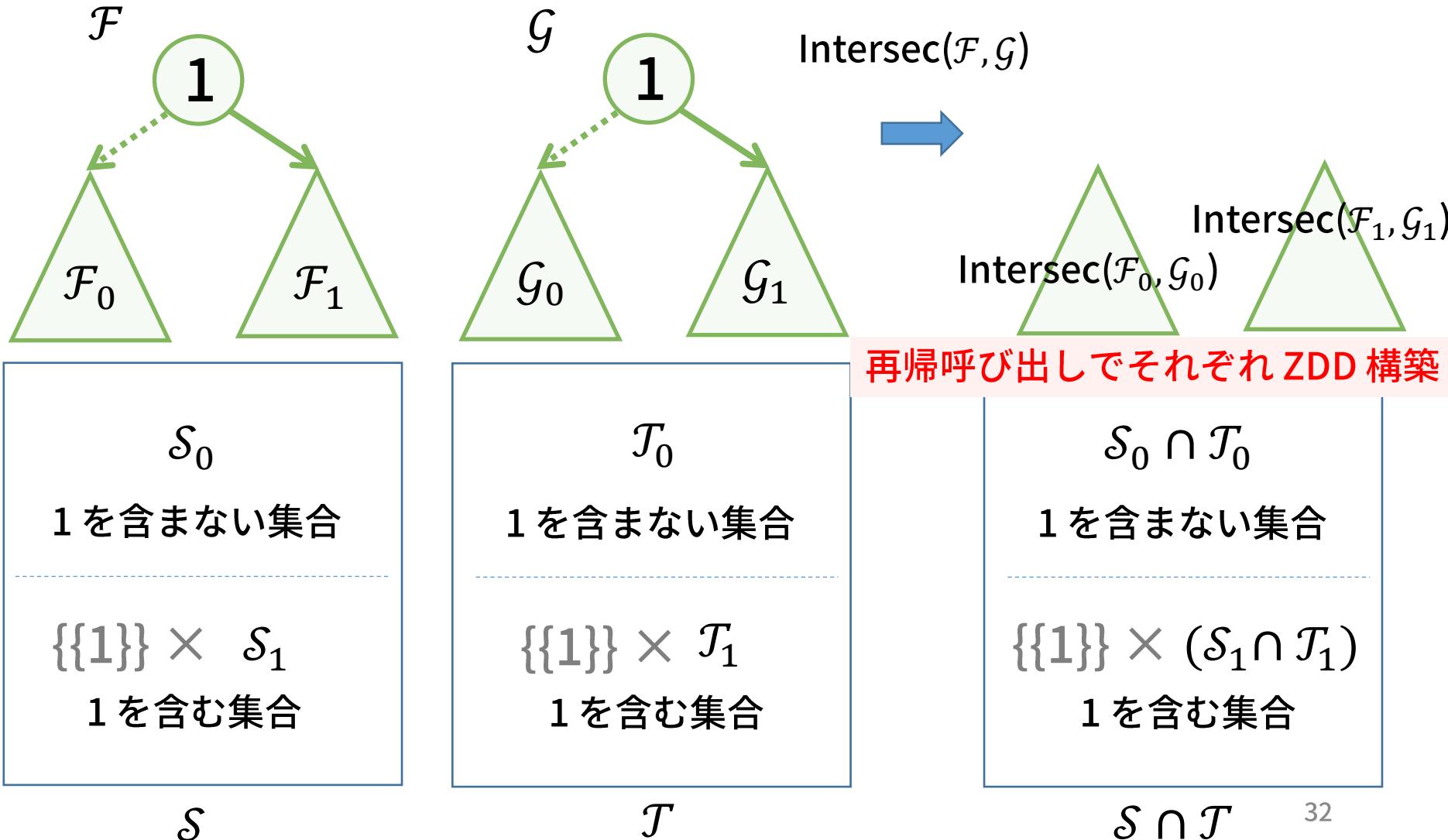
- $\text{Intersec}(\mathcal{F}, \mathcal{G})$ :  $\mathcal{S} \cap \mathcal{T}$  を表す ZDD を構築



# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

集合族  $\mathcal{S}, \mathcal{T}$  を表す ZDD を  $\mathcal{F}, \mathcal{G}$  とする

- $\text{Intersec}(\mathcal{F}, \mathcal{G})$ :  $\mathcal{S} \cap \mathcal{T}$  を表す ZDD を構築

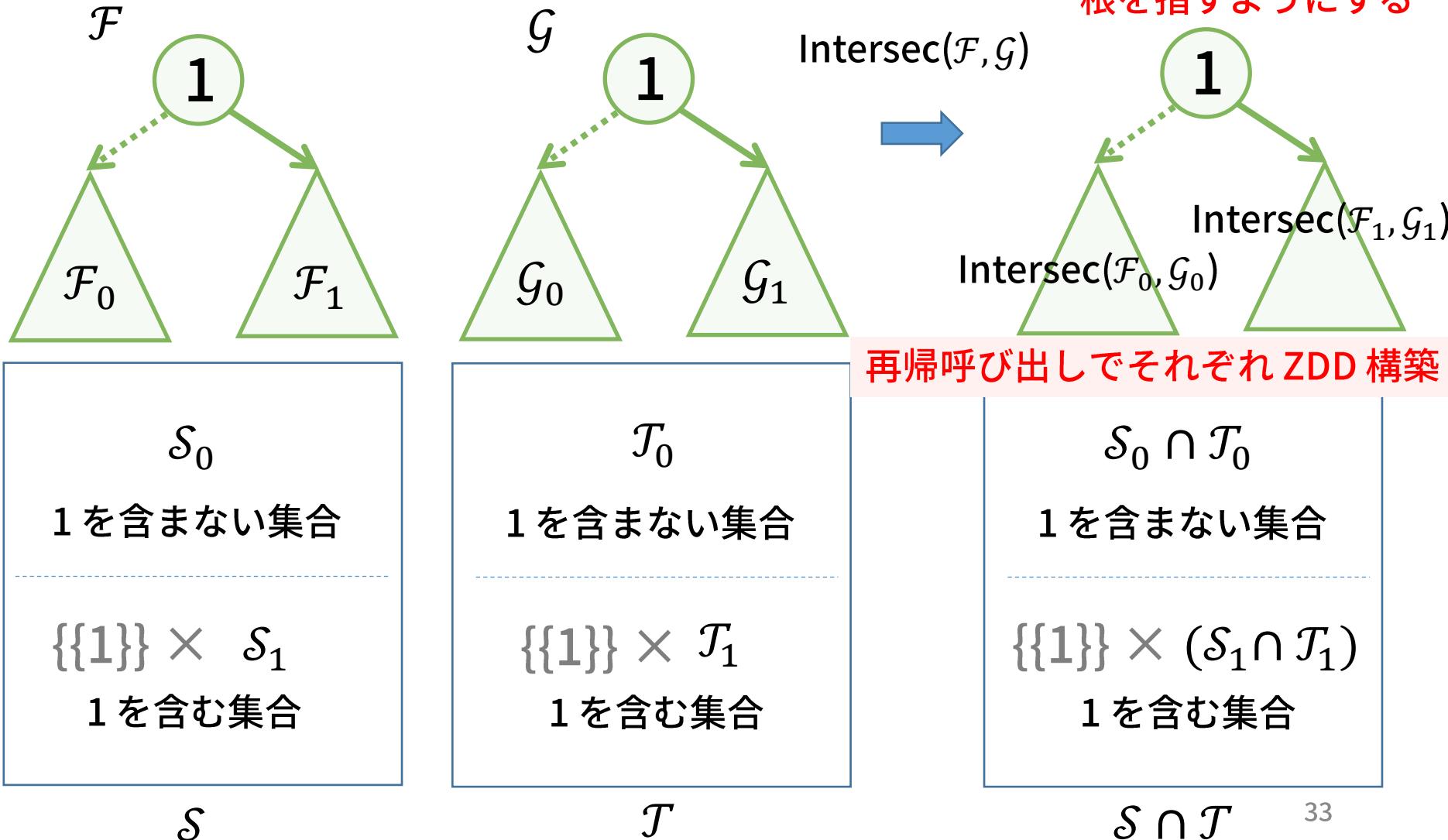


# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

集合族  $\mathcal{S}, \mathcal{T}$  を表す ZDD を  $\mathcal{F}, \mathcal{G}$  とする

- $\text{Intersec}(\mathcal{F}, \mathcal{G})$ :  $\mathcal{S} \cap \mathcal{T}$  を表す ZDD を構築

根ノードを作成して  
0/1枝を構築 ZDD の  
根を指すようにする

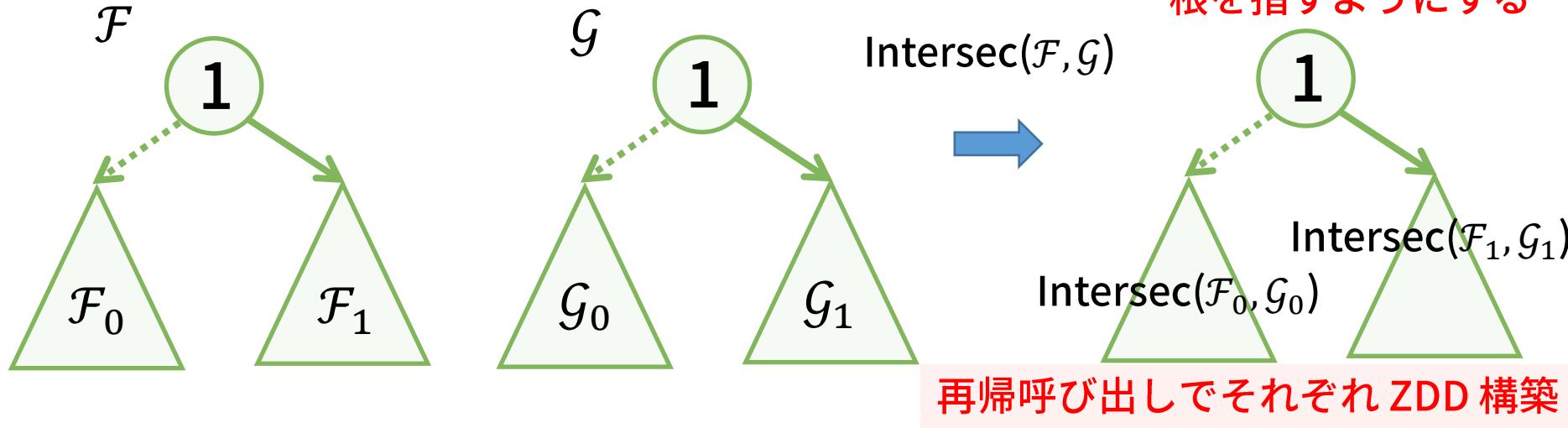


# ZDD の構築法: ボトムアップ: ZDD の共通部分演算 (Intersection)

集合族  $\mathcal{S}, \mathcal{T}$  を表す ZDD を  $\mathcal{F}, \mathcal{G}$  とする

- $\text{Intersec}(\mathcal{F}, \mathcal{G})$ :  $\mathcal{S} \cap \mathcal{T}$  を表す ZDD を構築

根ノードを作成して  
0/1枝を構築 ZDD の  
根を指すようにする



再帰の末端

$$\text{Intersec}(\boxed{0}, \mathcal{G}) = \boxed{0} \quad \text{for all } \mathcal{G}$$

**0** は空集合  $\emptyset$  を表す

空集合との共通部分は空集合

$$\text{Intersec}(\boxed{1}, \boxed{1}) = \boxed{1}$$

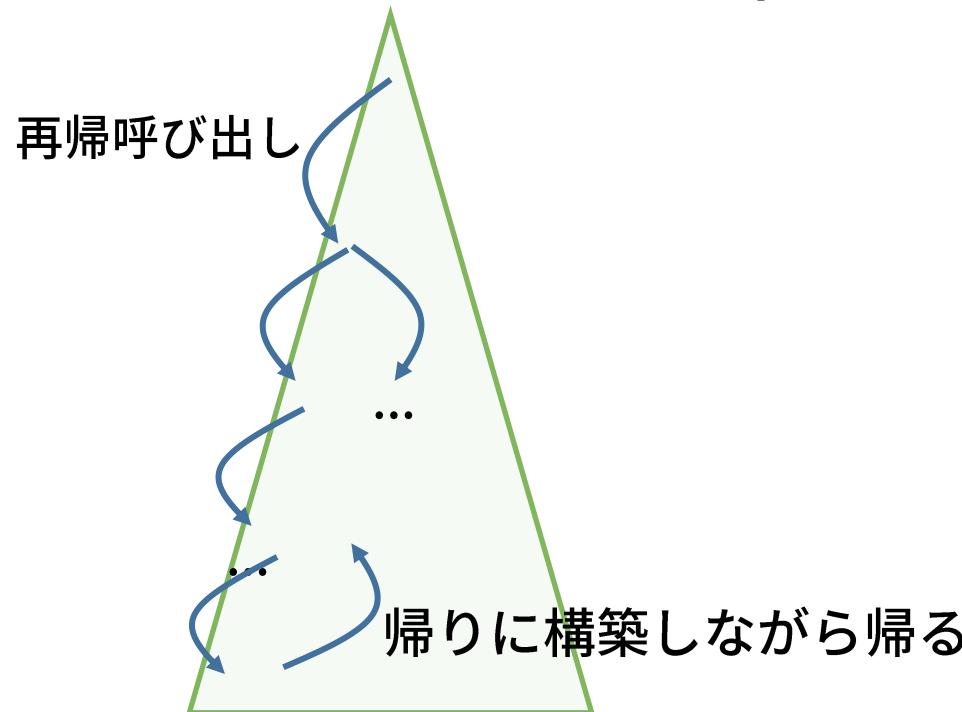
**1** は空集合からなる集合  $\{\emptyset\}$

$$\{\emptyset\} \cap \{\emptyset\} = \{\emptyset\}$$

など

# ZDD の構築法: ボトムアップ

- ZDD の再帰構造を利用した ZDD 構築法を  
ボトムアップ構築法、または Apply 演算という

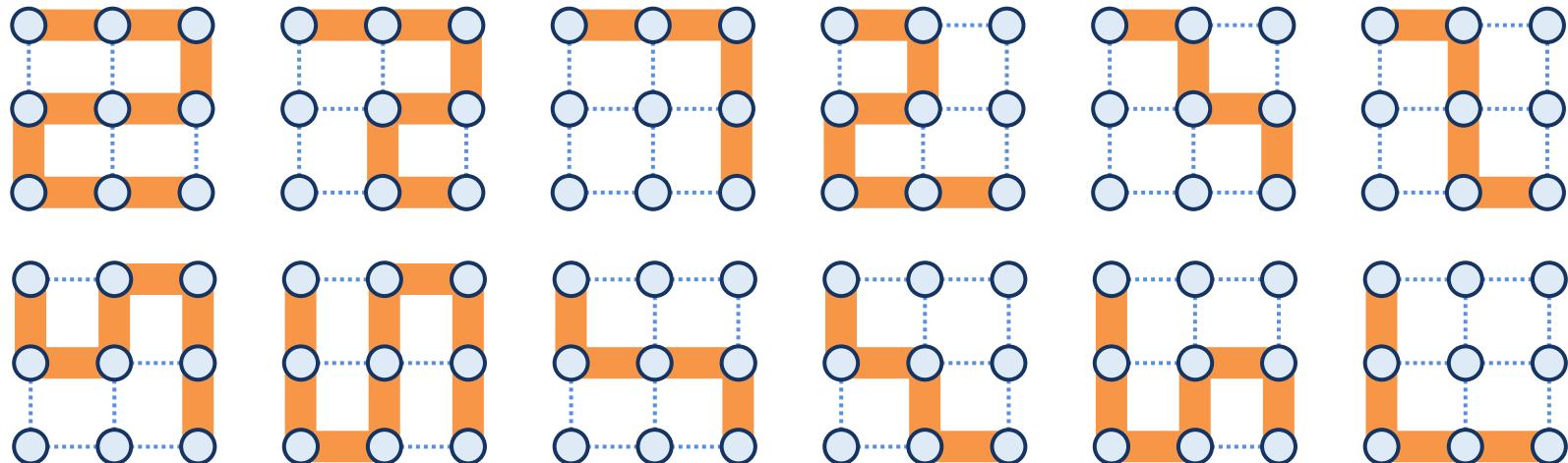
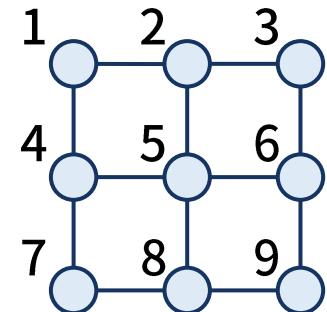


ボトムアップ構築法の例:

和集合  
差集合  
共通部分集合  
superset  
subset  
hitting set

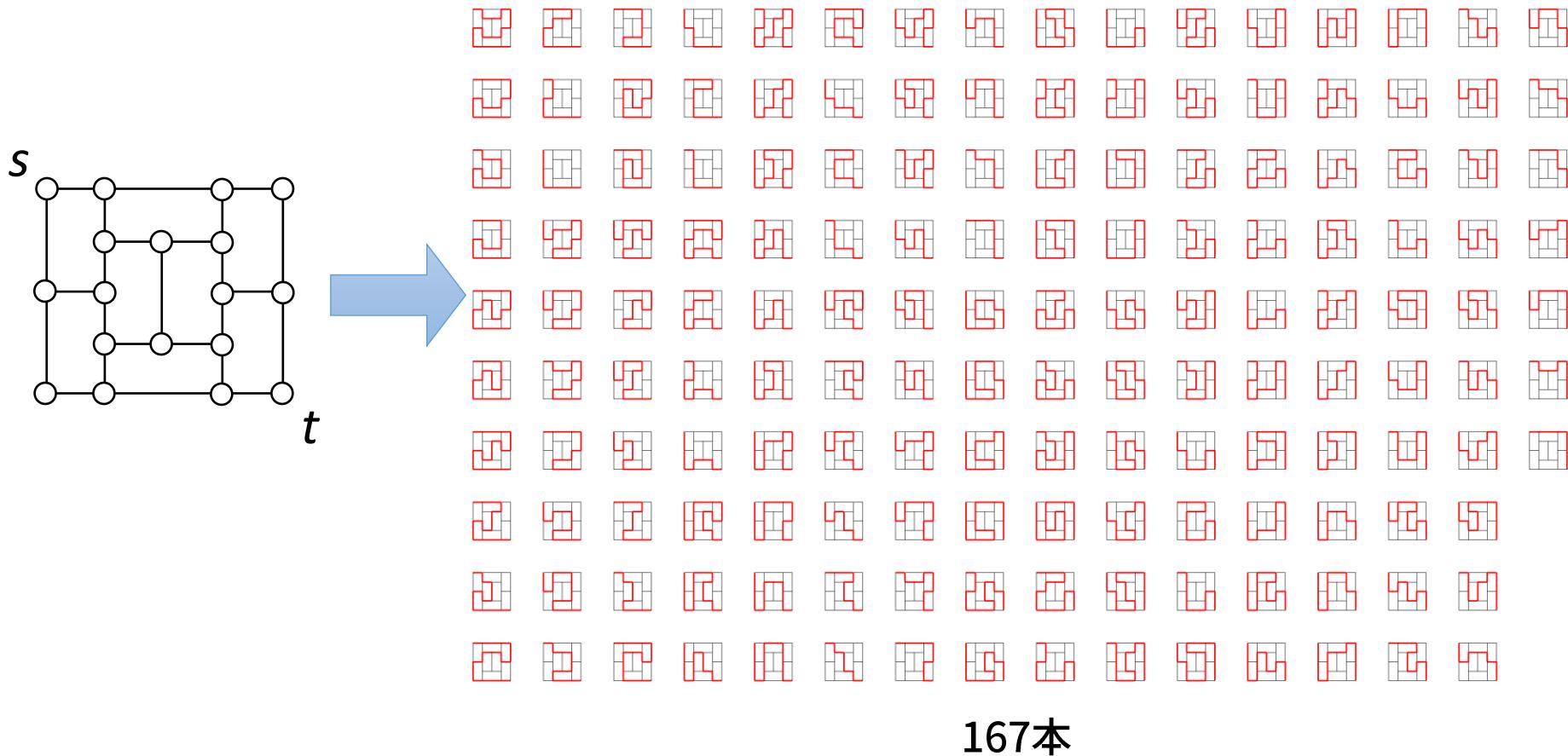
# ZDD のグラフ問題への応用: $s-t$ パス列挙問題

- ・入力グラフと、頂点  $s, t$  が与えられる
- ・ $s-t$  パス（同じ頂点を2度通らない）を  
**すべて求めたい**



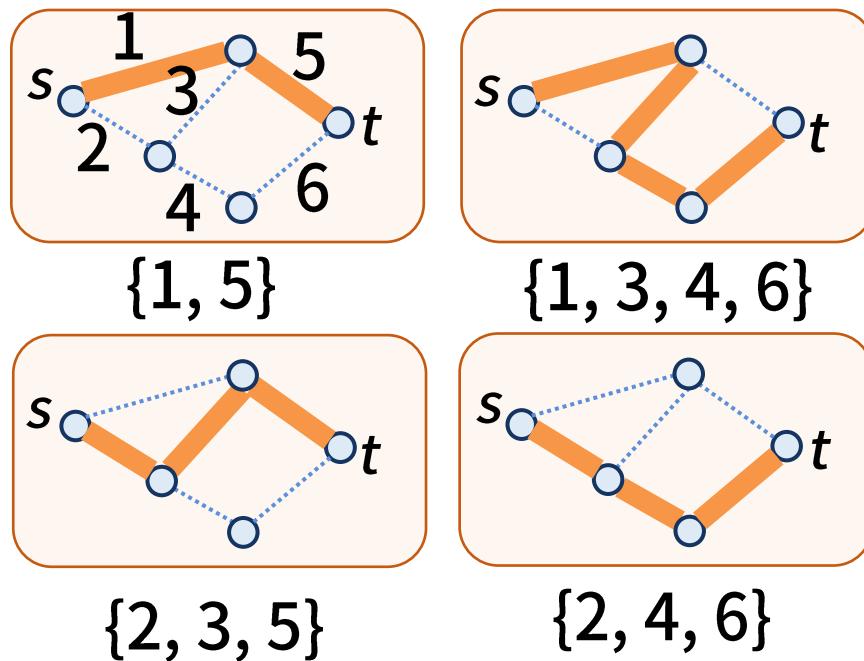
# $s-t$ パス列挙問題

- ・少しだけ大きな例

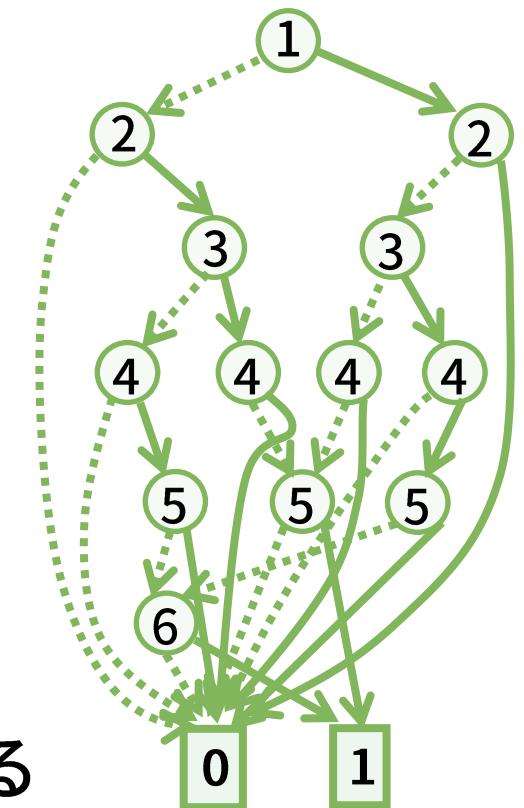
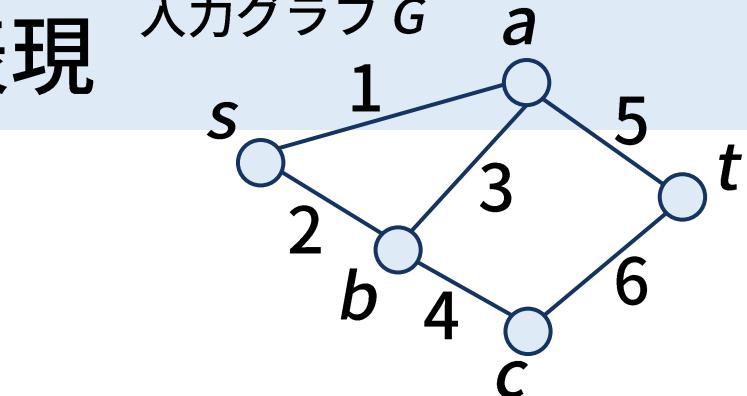


# パス集合の集合族による表現

- ・入力グラフ  $G$  を固定する
- ・パスを辺の集合とみなせる



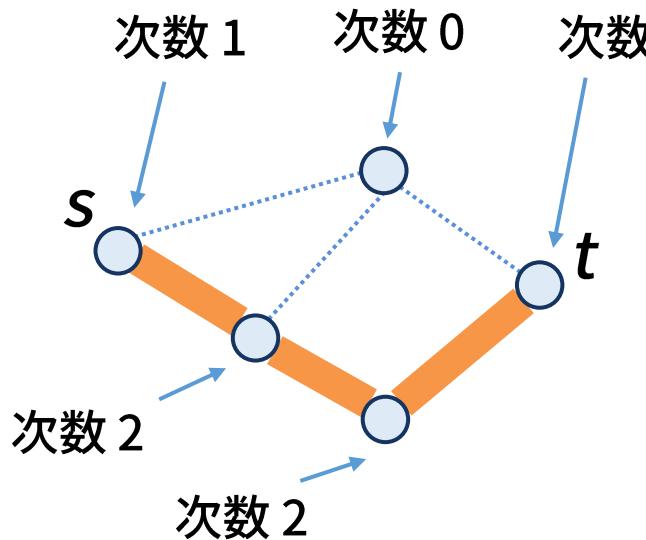
入力グラフ  $G$



- ・パス集合を辺の集合の集合とみなせる
  - ・ZDDで表現できる

# パスの性質

- ・パスの（部分グラフとしての）次数を考える



$s$  と  $t$  の次数は 1  
パスの内点の次数は 2  
パスを構成しない頂点の次数は 0

- ・「 $s$  と  $t$  は次数 1、それ以外は次数 0 または 2 であり、次数 2 のすべての頂点は  $s, t$  と連結」となる  
部分グラフは  $s$ - $t$  パス



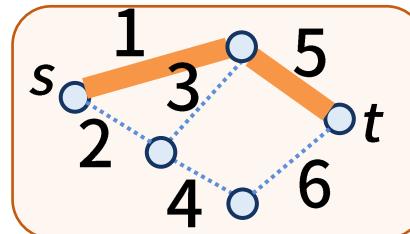
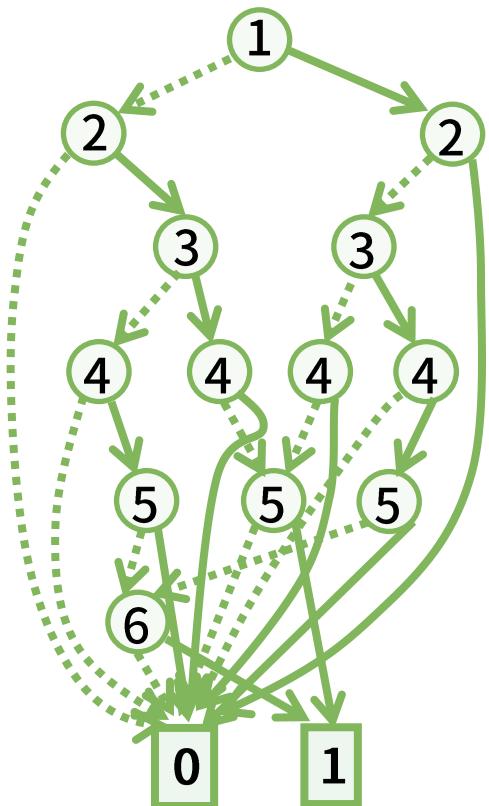
サイクルが存在できないので  
単独の  $s$ - $t$  パスになる

# パス集合を表す ZDD の構築

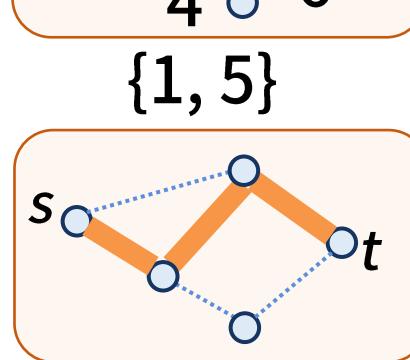
- Knuth は著書 「The art of computer programming 4A」 (2011) で、以下のアルゴリズムを示した。

- 入力グラフと頂点  $s, t$  に対して、 $s-t$  パス集合を表す ZDD を **直接** 構築する

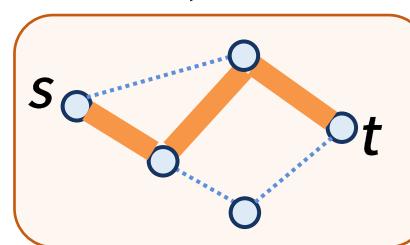
ZDD を構築できれば、要素の列挙は可能  
以下では ZDD 構築を目指す



{1, 5}



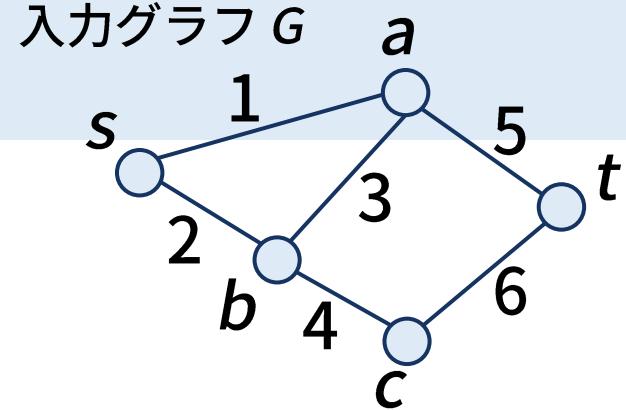
{1, 3, 4, 6}



{2, 3, 5}



{2, 4, 6}

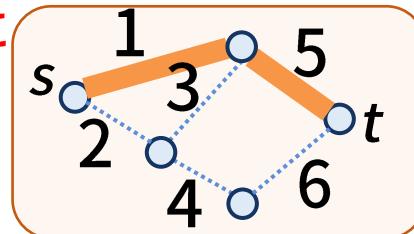
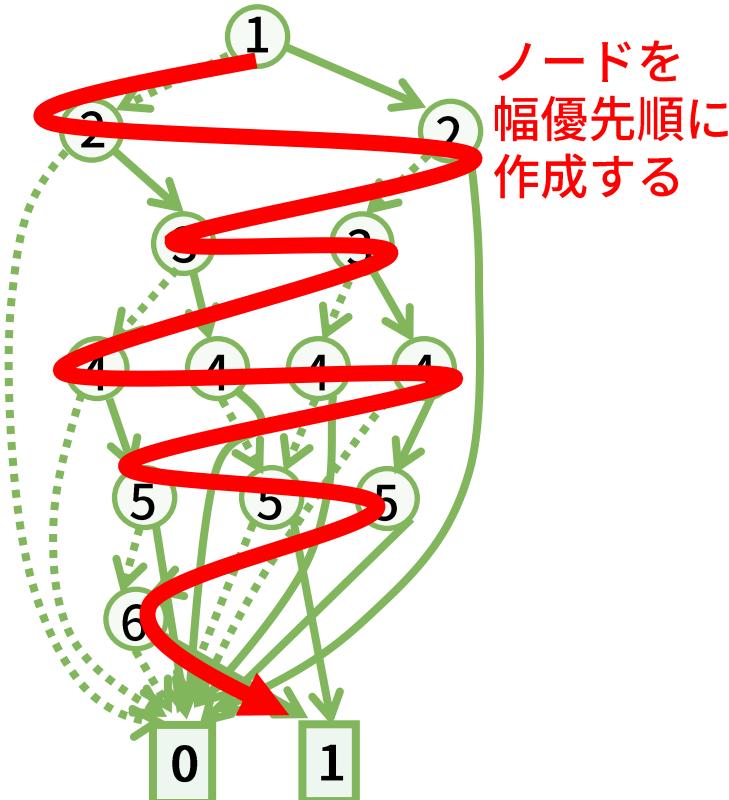


# パス集合を表す ZDD の構築

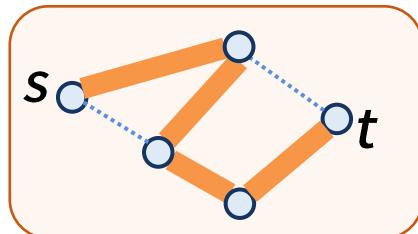
- Knuth は著書 「The art of computer programming 4A」 (2011) で、以下のアルゴリズムを示した。

- 入力グラフと頂点  $s, t$  に対して、 $s-t$  パス集合を表す ZDD を **直接構築**する

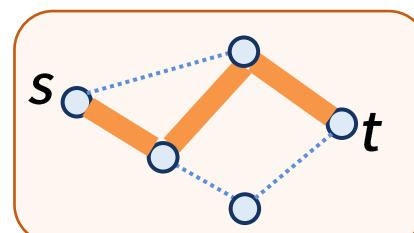
ZDD を構築できれば、要素の列挙は可能  
以下では ZDD 構築を目指す



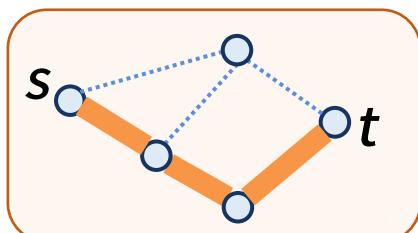
$\{1, 5\}$



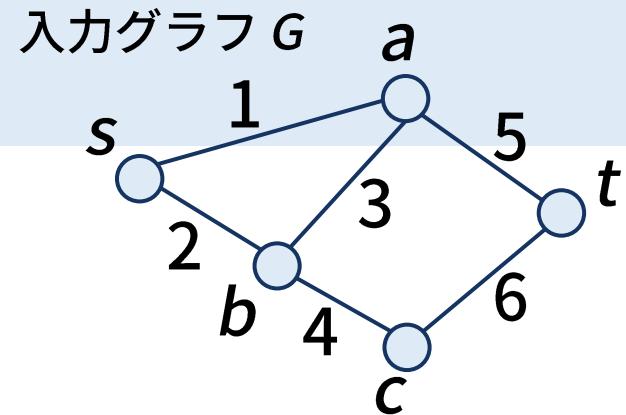
$\{1, 3, 4, 6\}$



$\{2, 3, 5\}$

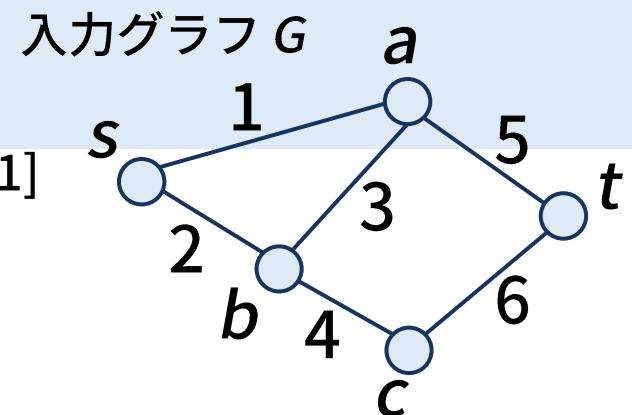


$\{2, 4, 6\}$



# パス集合を表す ZDD の構築

[Knuth 2011]



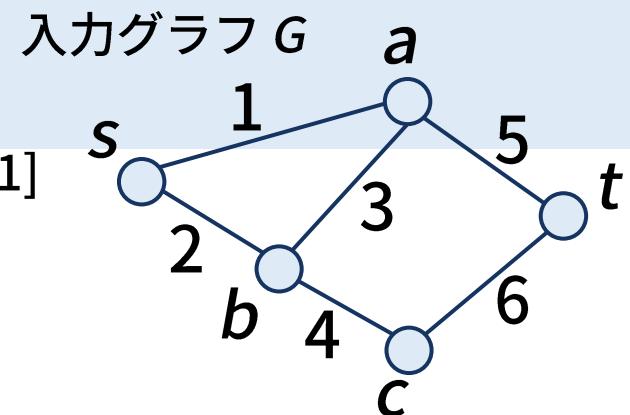
$s-t$  パスの連結性はとりあえず無視する。

グラフの辺を、何らかの方法で順序付ける（詳細略）。  
辺の順を  $1, 2, 3, \dots$  とする。

# パス集合を表す ZDD の構築

[Knuth 2011]

1



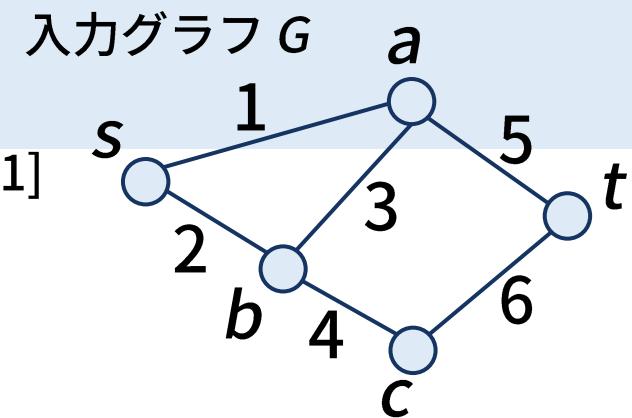
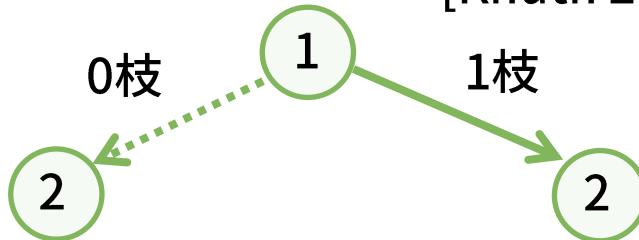
$s-t$  パスの連結性はとりあえず無視する。

グラフの辺を、何らかの方法で順序付ける（詳細略）。  
辺の順を  $1, 2, 3, \dots$  とする。

最初に根ノードを作成する。

# パス集合を表す ZDD の構築

[Knuth 2011]



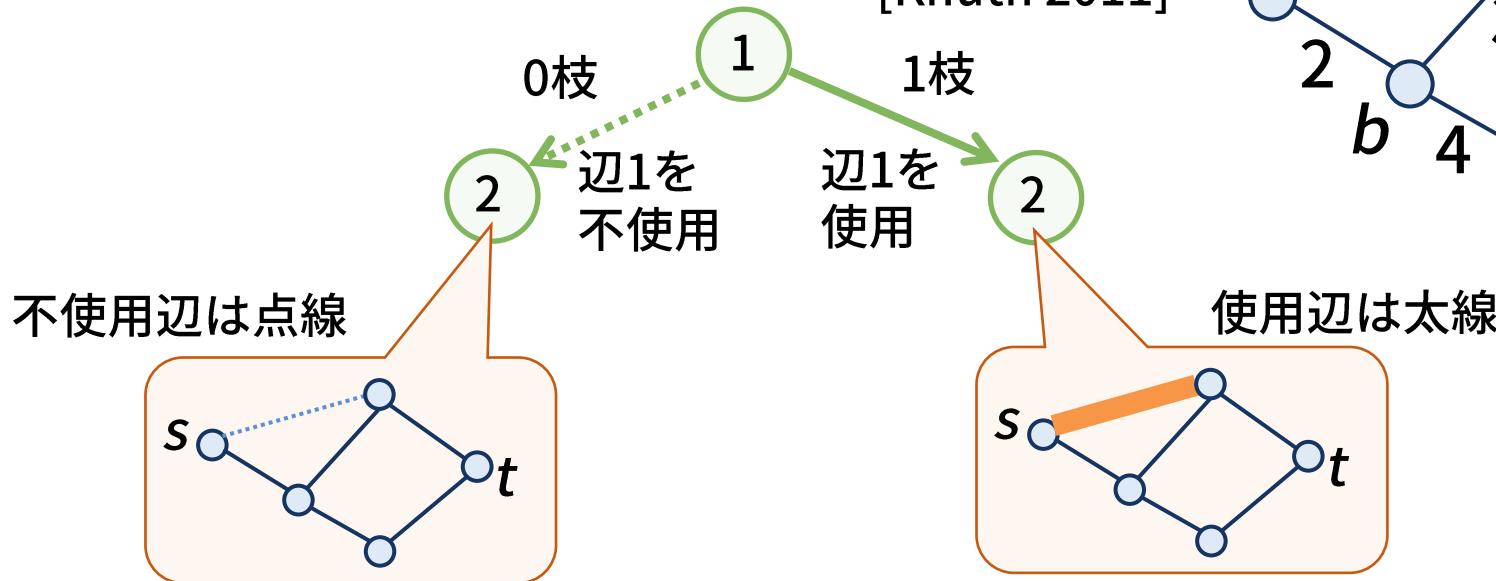
$s-t$  パスの連結性はとりあえず無視する。

グラフの辺を、何らかの方法で順序付ける（詳細略）。  
辺の順を  $1, 2, 3, \dots$  とする。

最初に根ノードを作成する。

根ノードから 0 枝と 1 枝を伸ばし、子ノードを作成する。

# パス集合を表す ZDD の構築



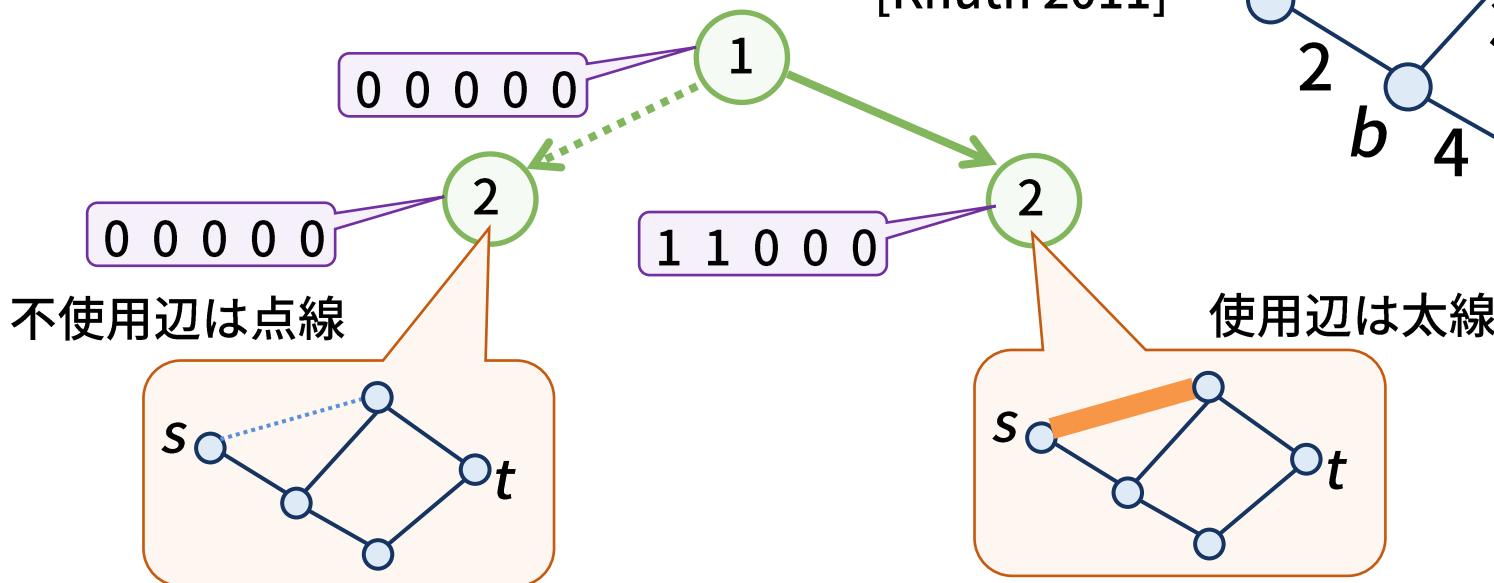
グラフの辺を、何らかの方法で順序付ける（詳細略）。  
辺の順を  $1, 2, 3, \dots$  とする。

最初に根ノードを作成する。

根ノードから 0枝と 1枝を伸ばし、子ノードを作成する。

これは、辺1をパスの辺として不使用/使用を場合分けしているとみなせる。

# パス集合を表す ZDD の構築



グラフの辺を、何らかの方法で順序付ける（詳細略）。  
辺の順を  $1, 2, 3, \dots$  とする。

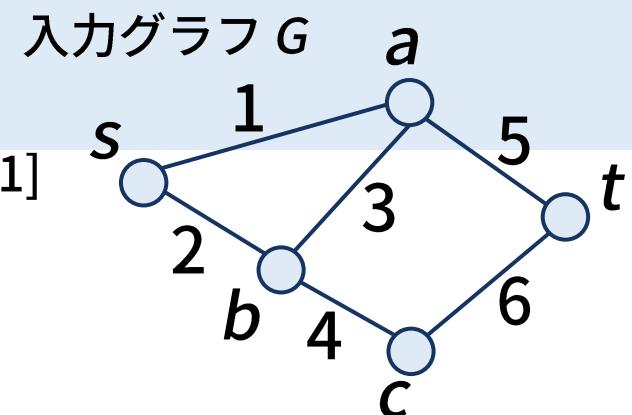
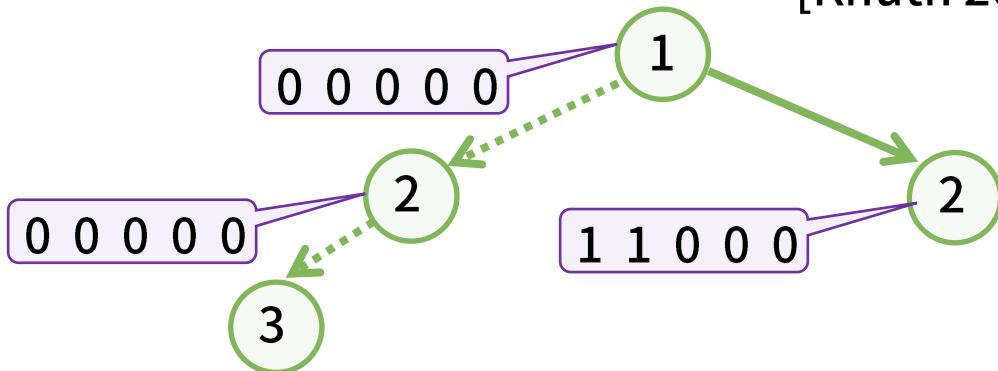
最初に根ノードを作成する。

根ノードから 0 枝と 1 枝を伸ばし、子ノードを作成する。

ノードには各頂点の次数を記憶する。

次数	$s$	$a$	$b$	$c$	$t$
0	0	0	0	0	0

# パス集合を表す ZDD の構築



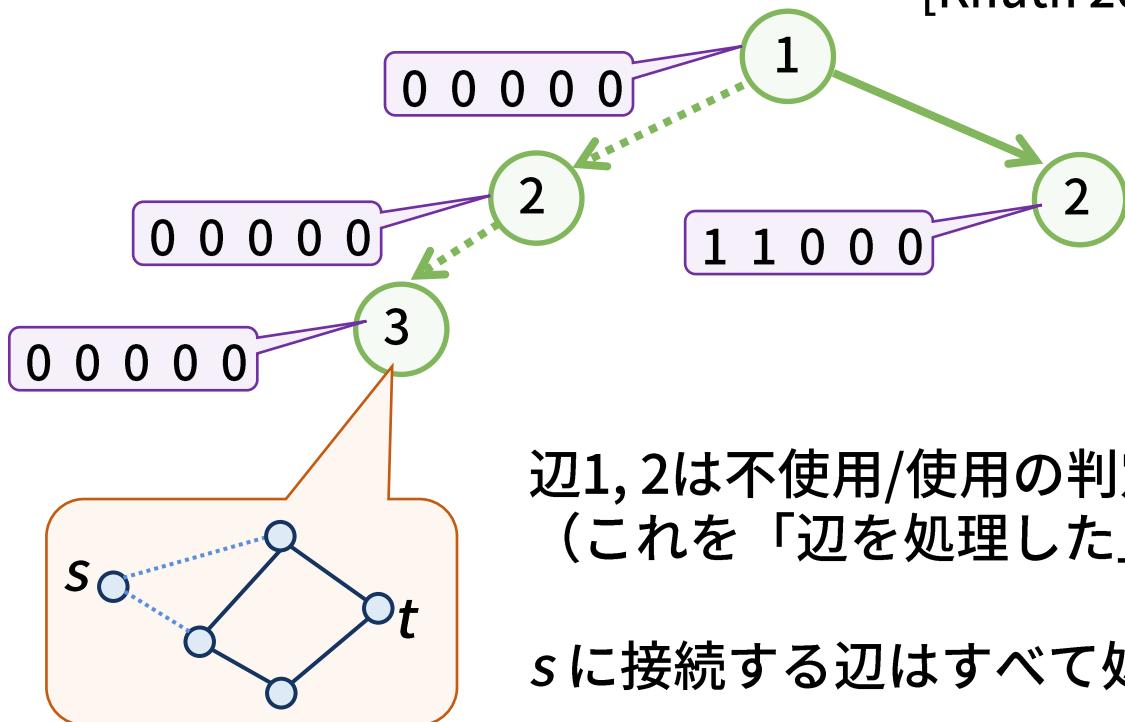
以降、子ノードを幅優先的に作成していく。

ノードには各頂点の次数を記憶する。

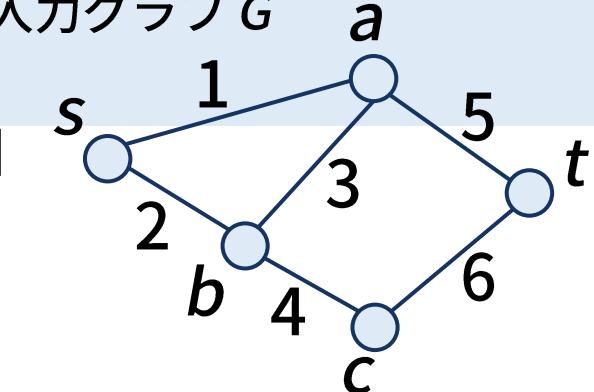
次数	$s$	$a$	$b$	$c$	$t$
0 0 0 0 0					

# パス集合を表す ZDD の構築

[Knuth 2011]



## 入力グラフ $G$

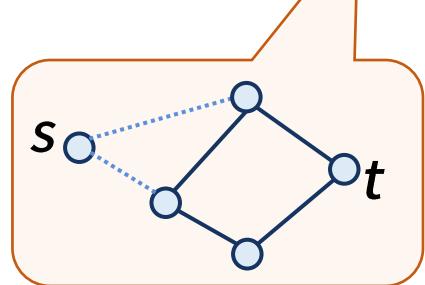
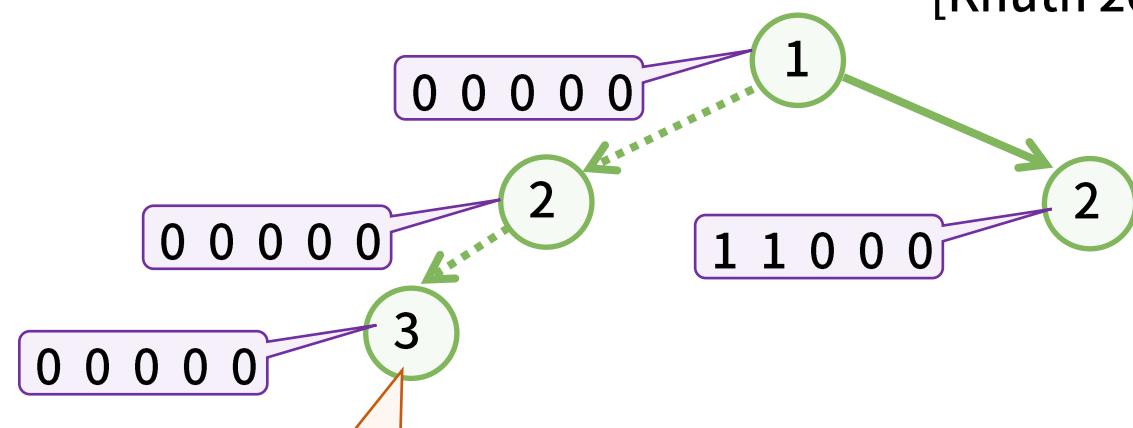


辺1, 2は不使用/使用的判定がされた。  
(これを「辺を処理した」と言う)

$s$  に接続する辺はすべて処理された。

次数
s a b c t
0 0 0 0 0

# パス集合を表す ZDD の構築



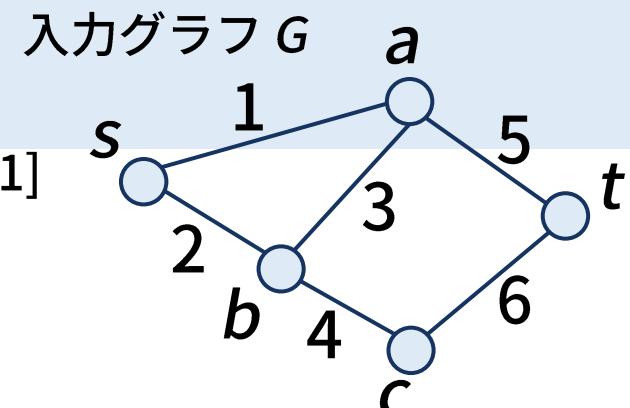
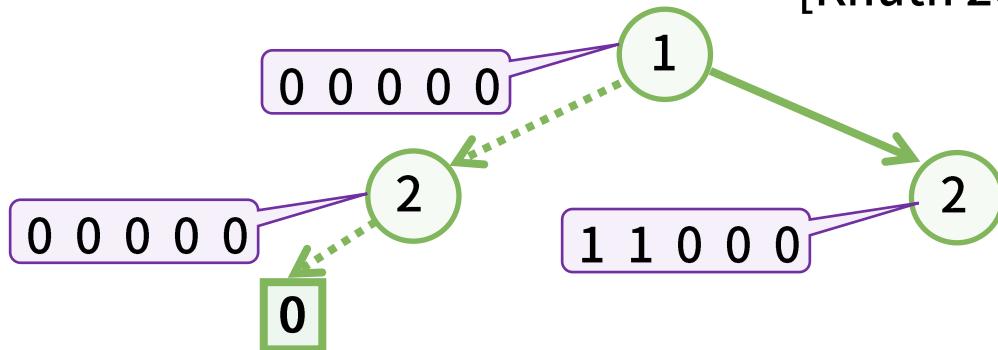
辺1, 2は不使用/使用的判定がされた。  
(これを「辺を処理した」と言う)

$s$ に接続する辺はすべて処理された。

$s$ の次数は、以降の辺の処理によって変化しない  
→  $s-t$  パスが完成する見込みが無くなつたので枝刈り

次数	$s$	$a$	$b$	$c$	$t$
	0	0	0	0	0

# パス集合を表す ZDD の構築



辺1, 2は不使用/使用的判定がされた。  
(これを「辺を処理した」と言う)

$s$  に接続する辺はすべて処理された。

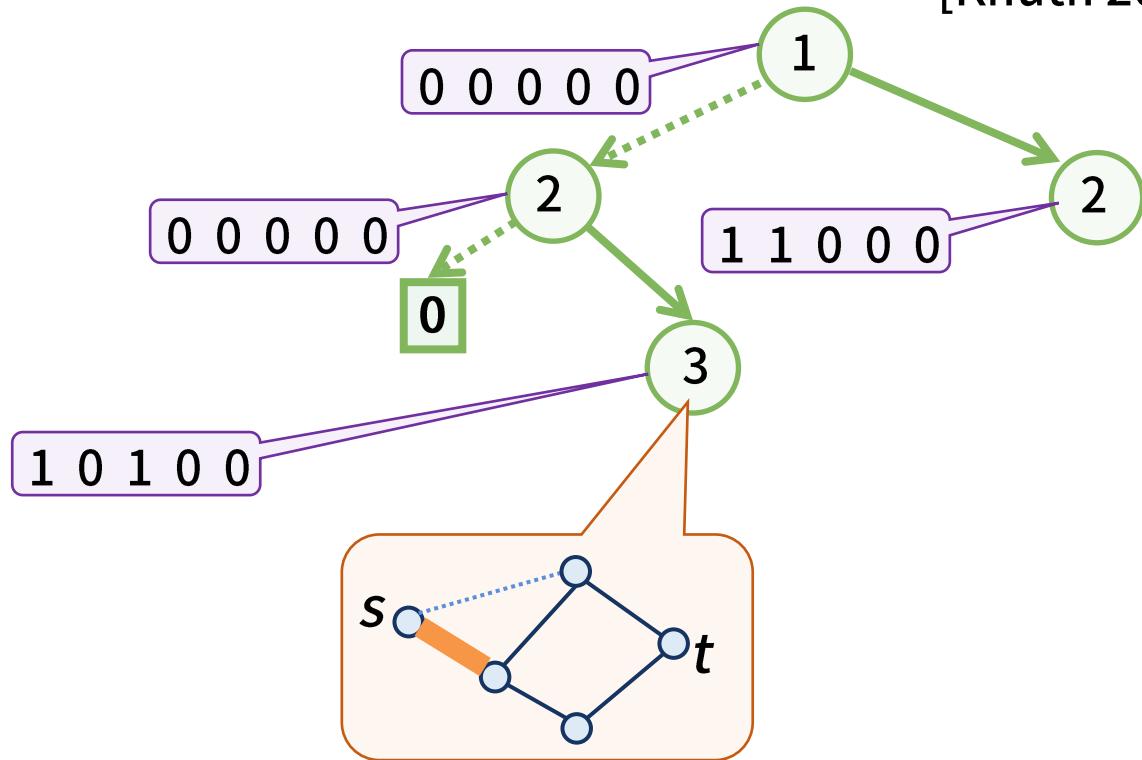
$s$  の次数は、以降の辺の処理によって変化しない  
→  $s-t$  パスが完成する見込みが無くなつたので枝刈り

0 を接続する (0-終端を接続することを枝刈りと言う)

次数	$s$	$a$	$b$	$c$	$t$
0 0 0 0 0					

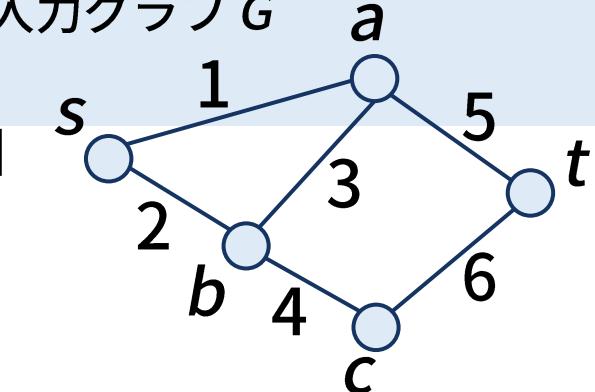
# パス集合を表す ZDD の構築

[Knuth 2011]



$s$  に接続する辺はすべて処理された。  
 $s$  の次数は 1 なので問題なし。

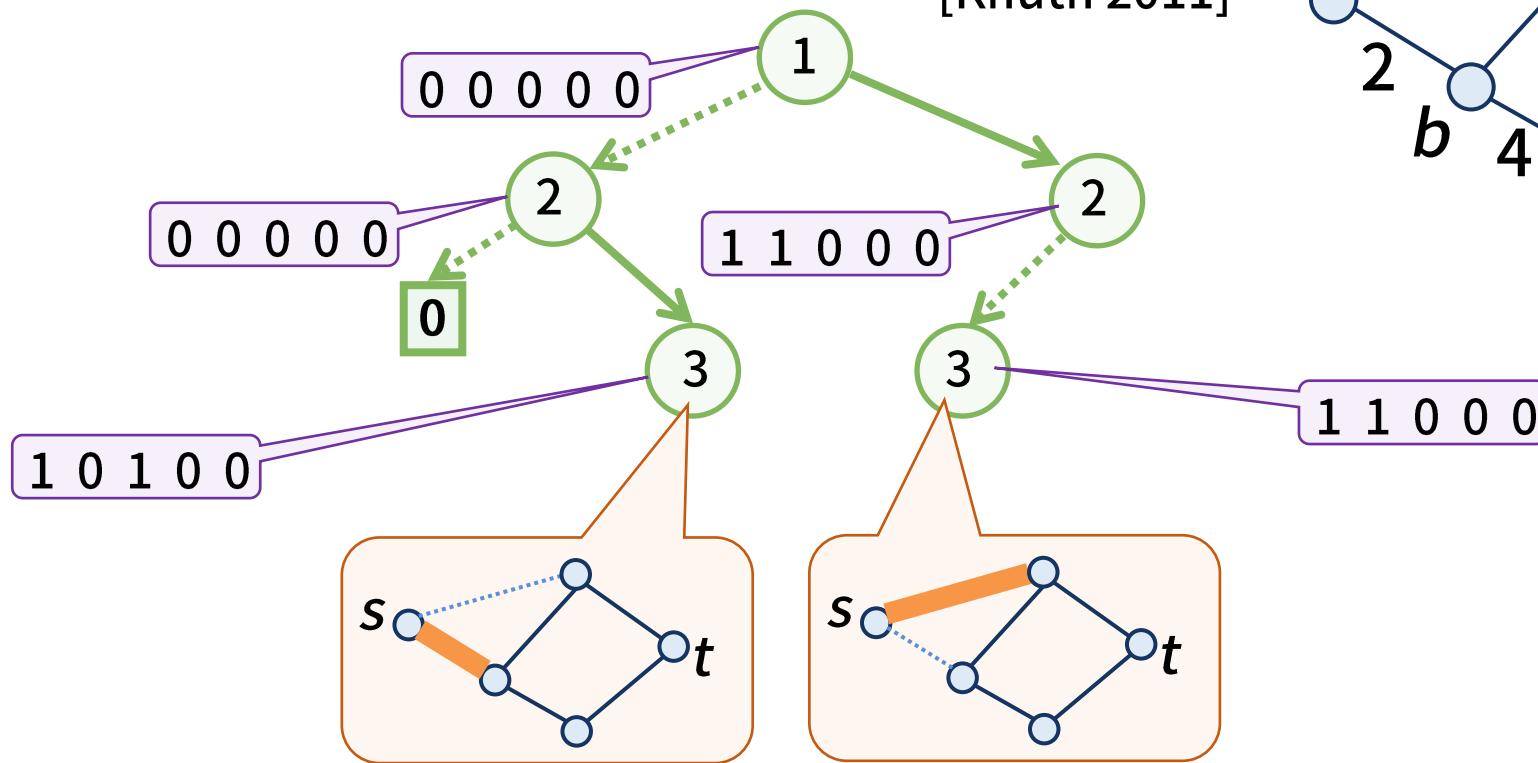
入力グラフ  $G$



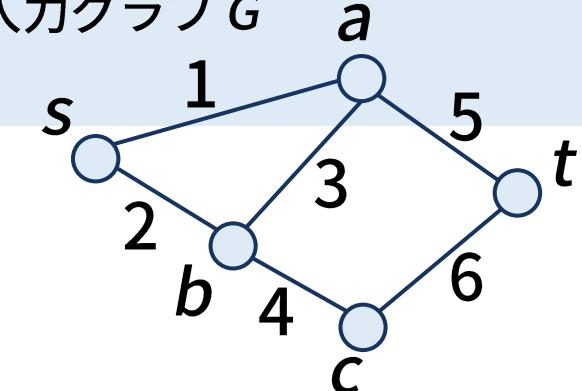
次数	$s$	$a$	$b$	$c$	$t$
0	0	0	0	0	0

# パス集合を表す ZDD の構築

[Knuth 2011]



入力グラフ  $G$

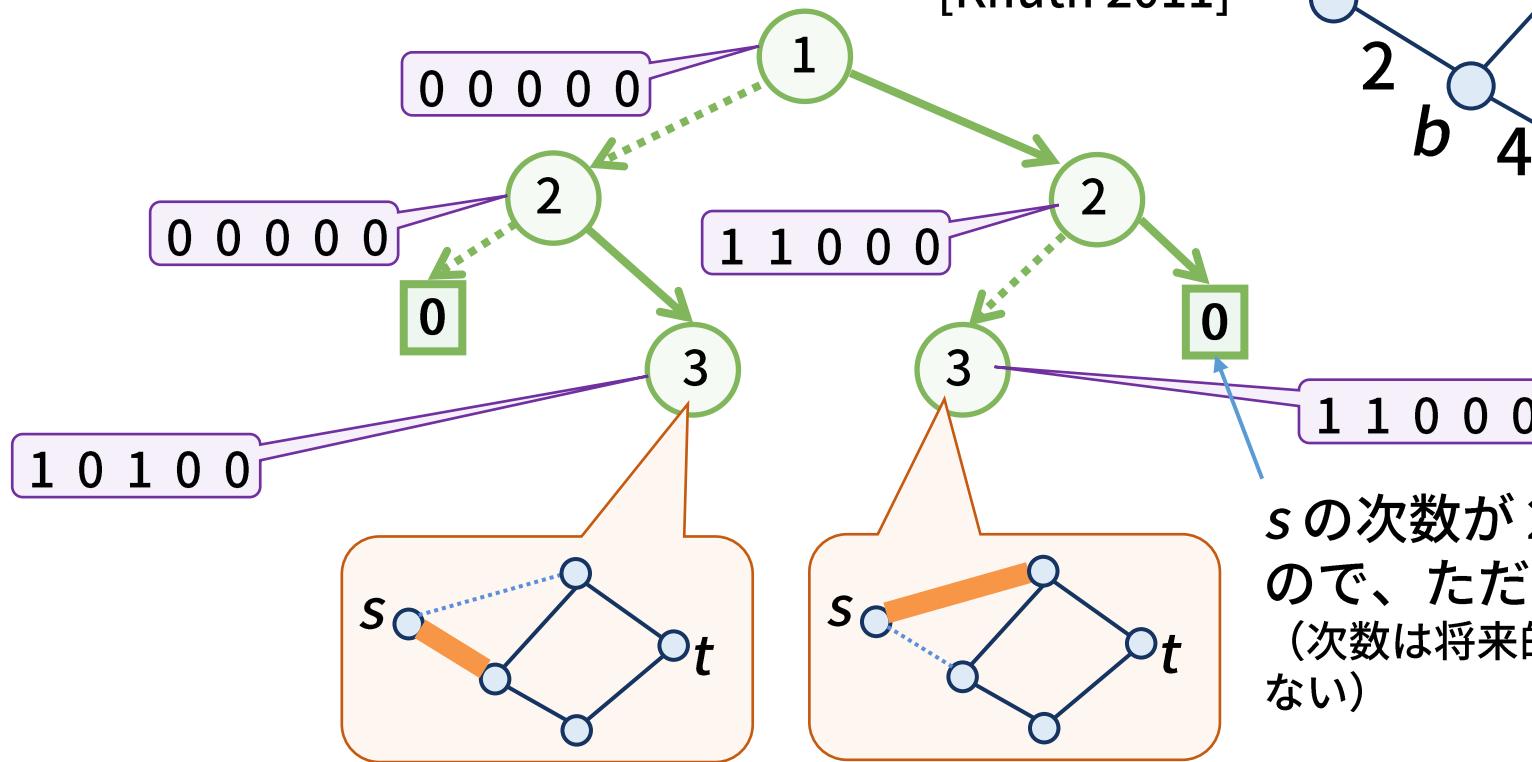


$s$  に接続する辺はすべて処理された。  
 $s$  の次数は 1 なので問題なし。

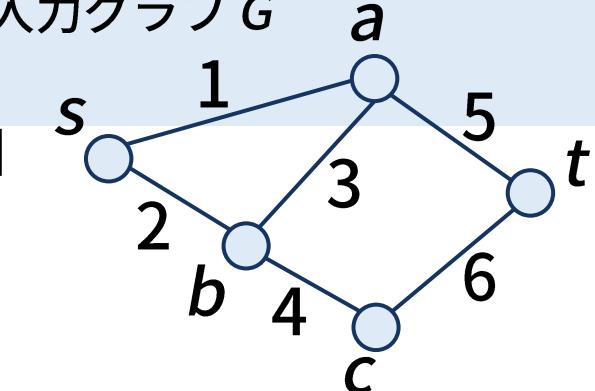
次数	$s$	$a$	$b$	$c$	$t$
0 0 0 0 0	0	0	0	0	0

# パス集合を表す ZDD の構築

[Knuth 2011]



入力グラフ  $G$

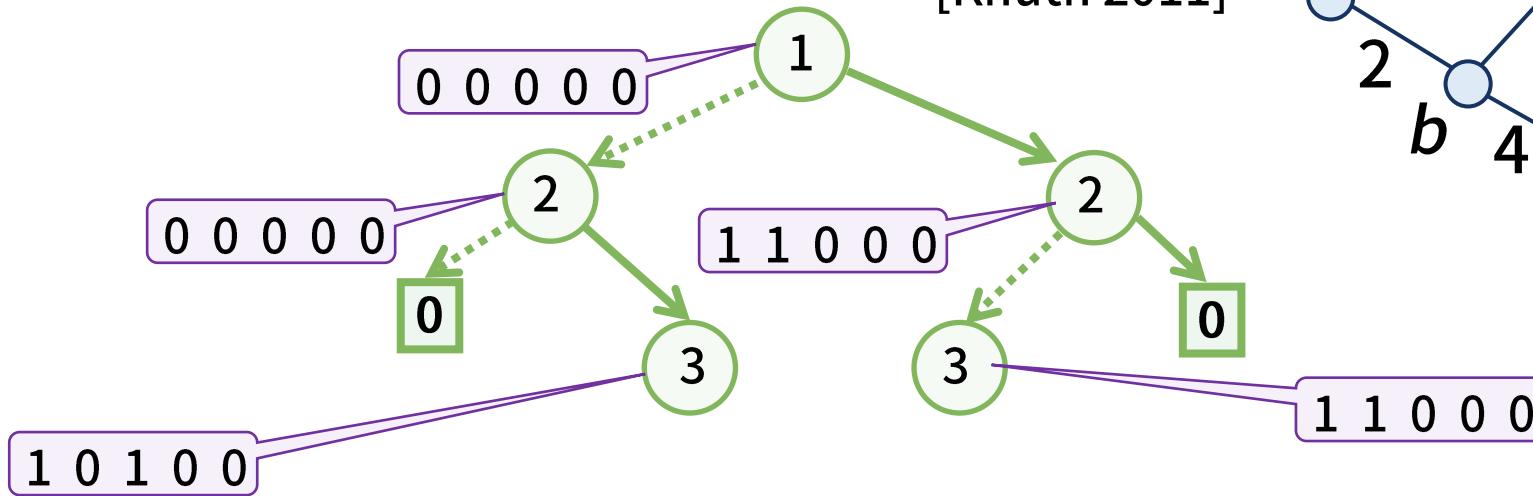


$s$  に接続する辺はすべて処理された。  
 $s$  の次数は 1 なので問題なし。

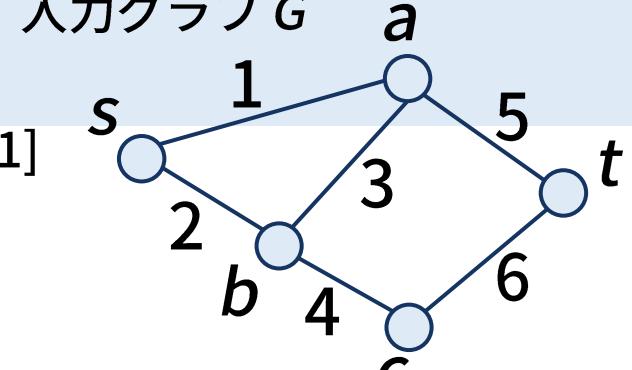
次数	$s$	$a$	$b$	$c$	$t$
0 0 0 0 0					

# パス集合を表す ZDD の構築

[Knuth 2011]



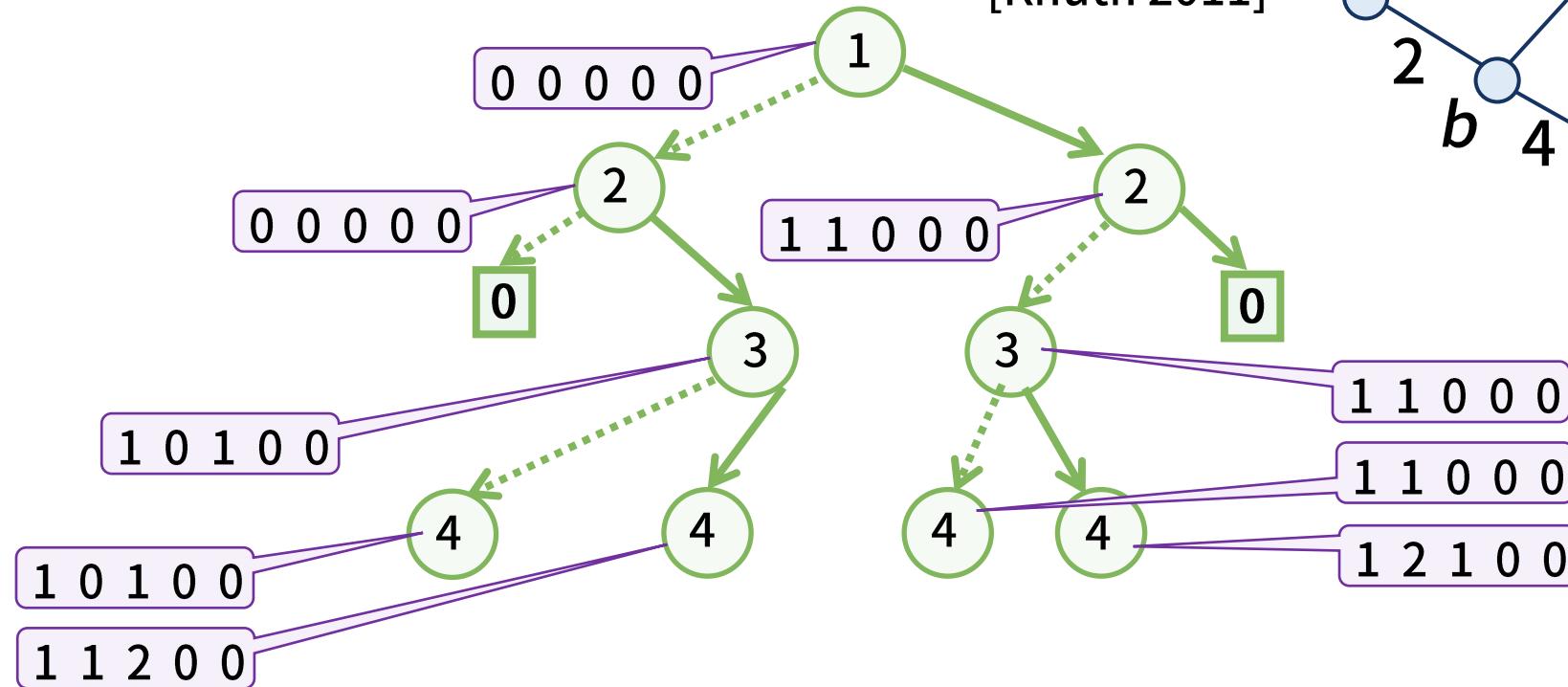
入力グラフ  $G$



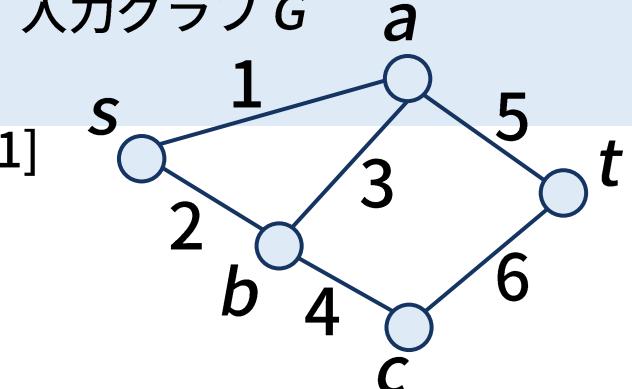
次数	$s$	$a$	$b$	$c$	$t$
00000	0	0	0	0	0

# パス集合を表す ZDD の構築

[Knuth 2011]



入力グラフ  $G$

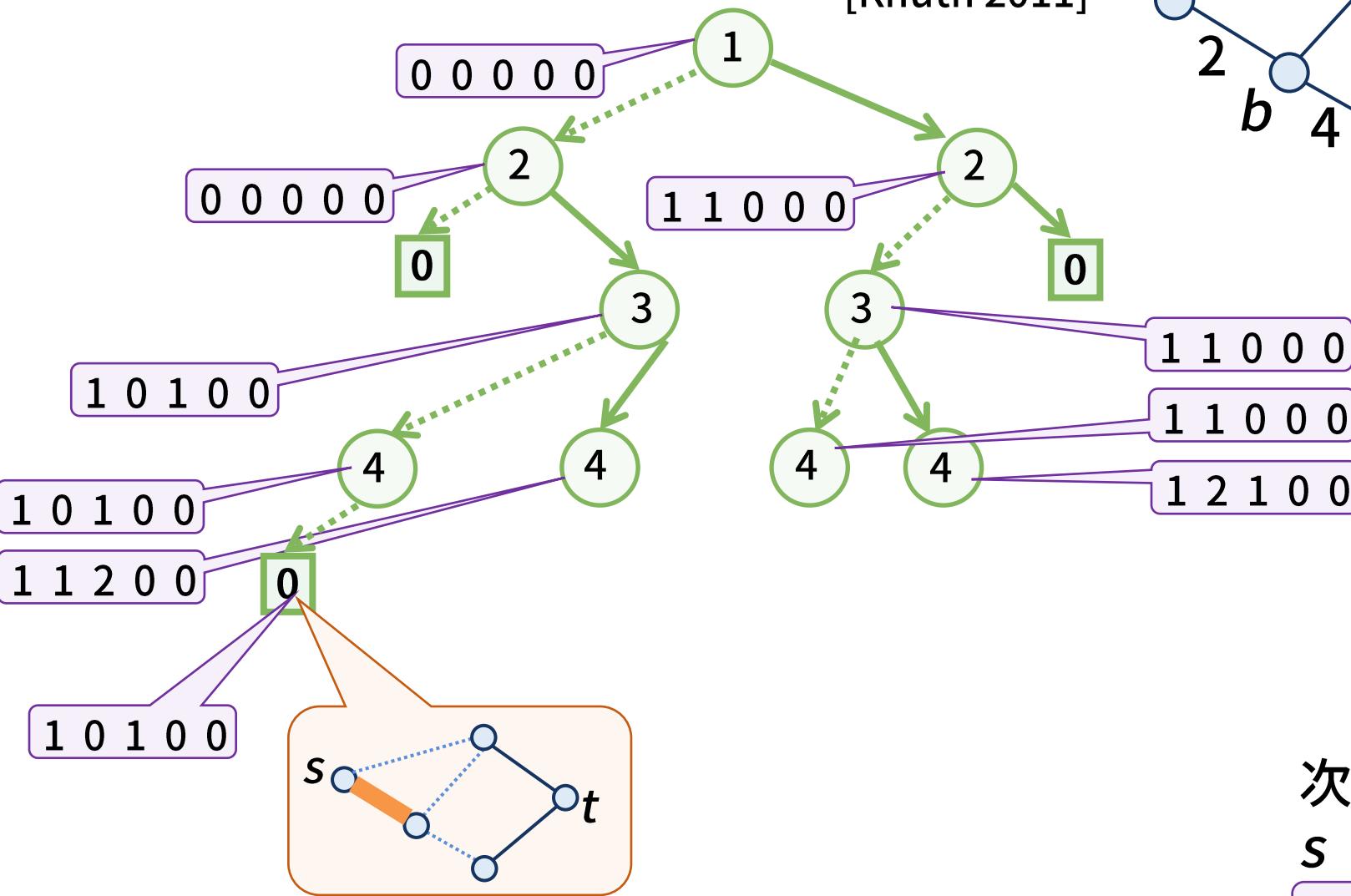
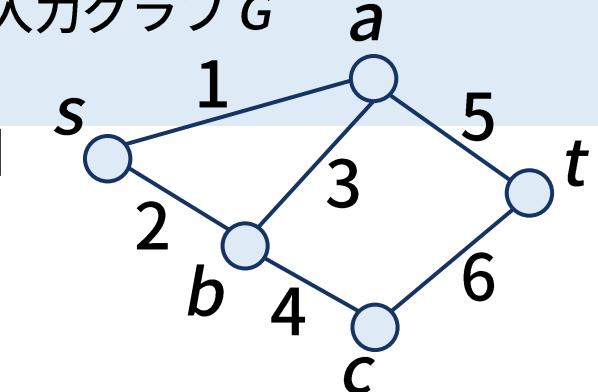


次数	$s$	$a$	$b$	$c$	$t$
00000	0	0	0	0	0

# パス集合を表す ZDD の構築

[Knuth 2011]

入力グラフ  $G$

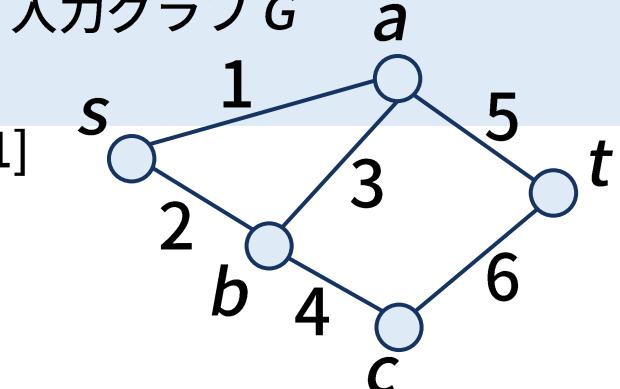


$b$  の次数が 1 に確定  $\rightarrow$  枝刈り

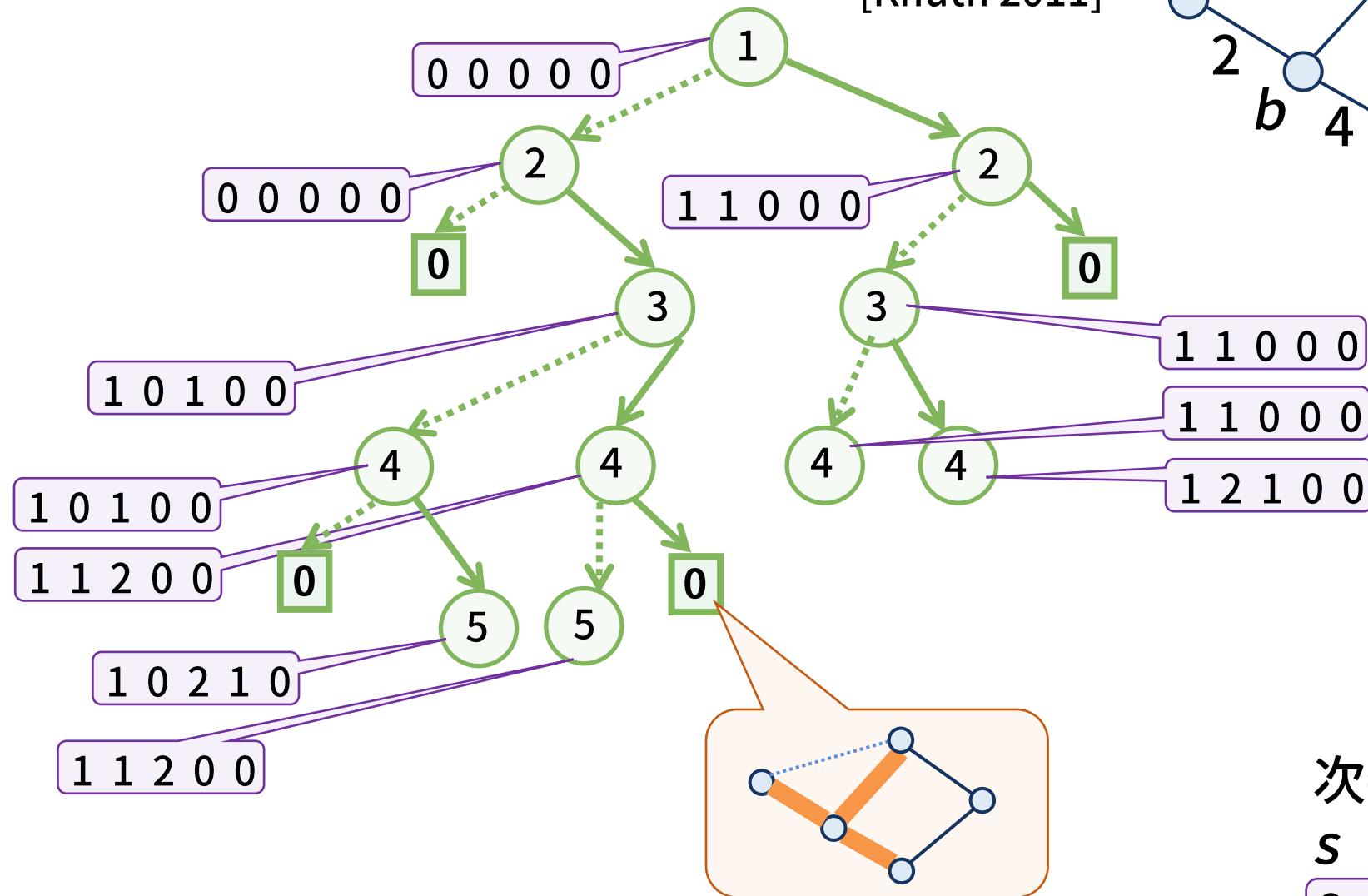
次数  
 $s \ a \ b \ c \ t$   
 $0 \ 0 \ 0 \ 0 \ 0$

# パス集合を表す ZDD の構築

## 入力グラフ $G$



[Knuth 2011]



$b$  の次数が 3 になった → 枝刈り

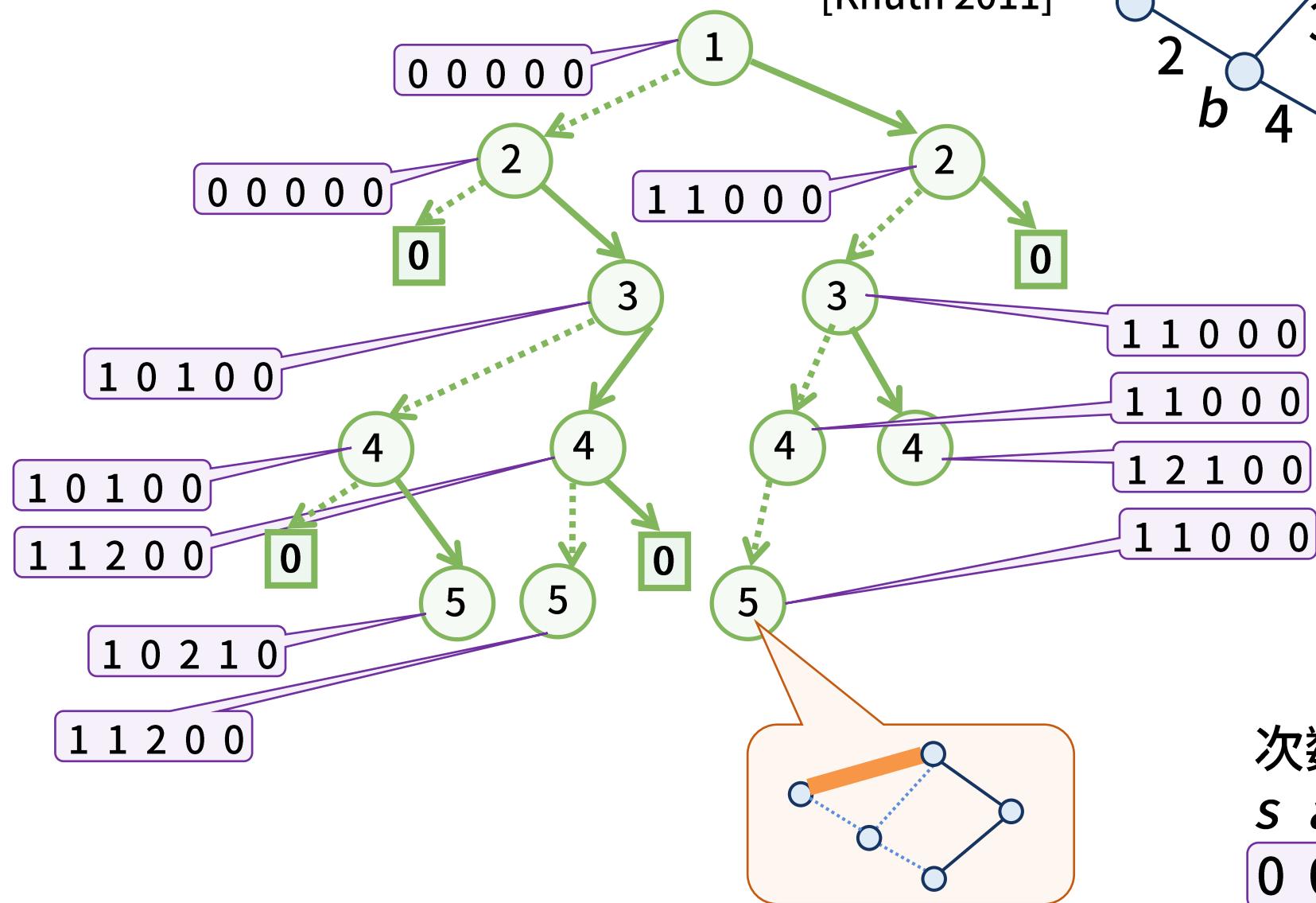
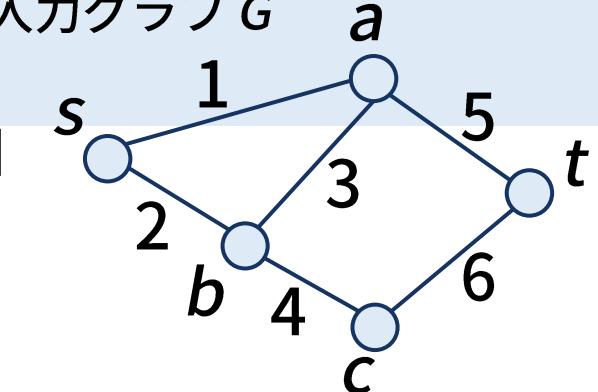
次数
s a b c t

0 0 0 0 0
-----------

# パス集合を表す ZDD の構築

[Knuth 2011]

入力グラフ  $G$

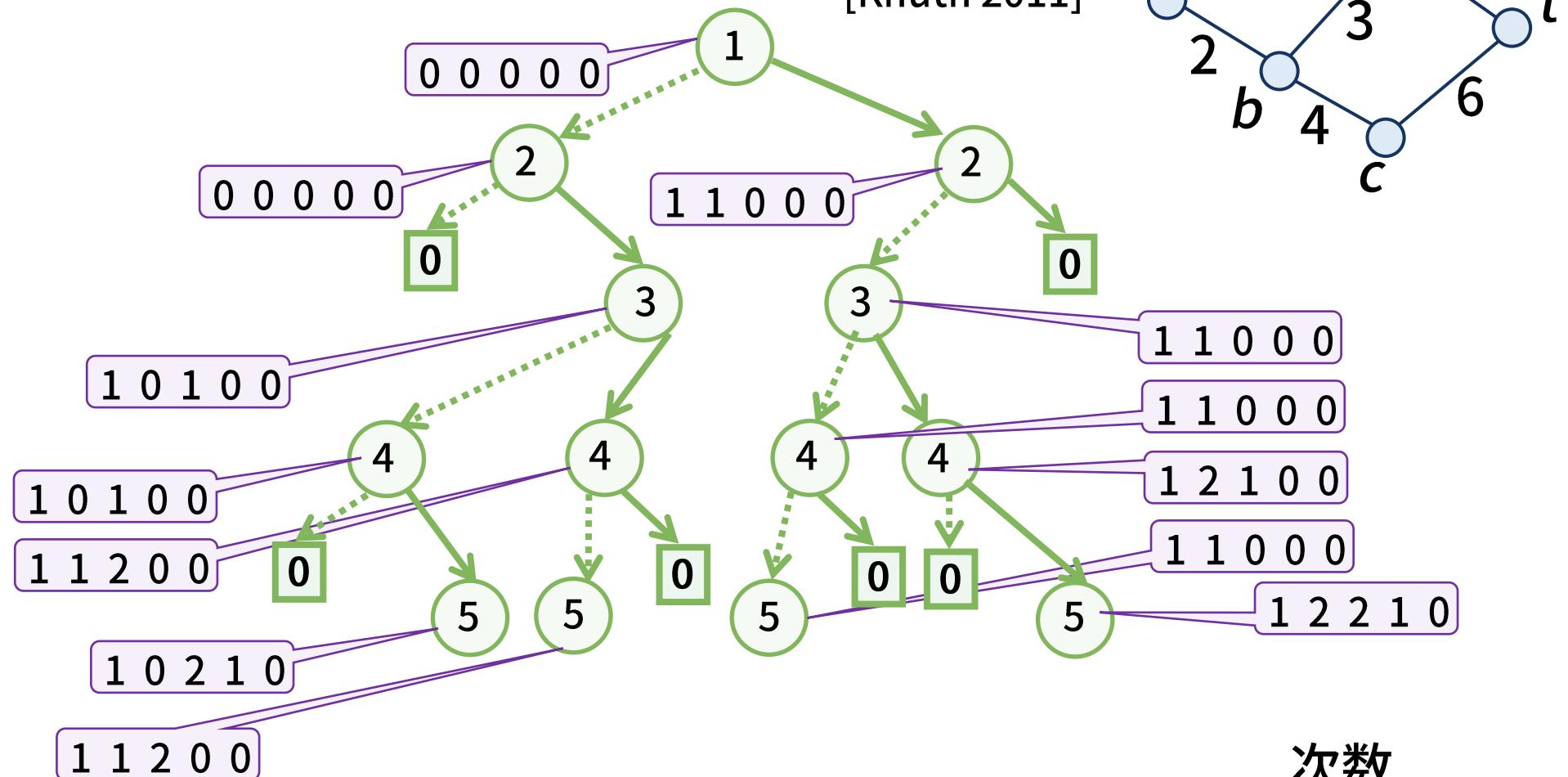


次数  
 $s \ a \ b \ c \ t$   
0 0 0 0 0

$b$  の次数が 0 に確定 → 問題なし

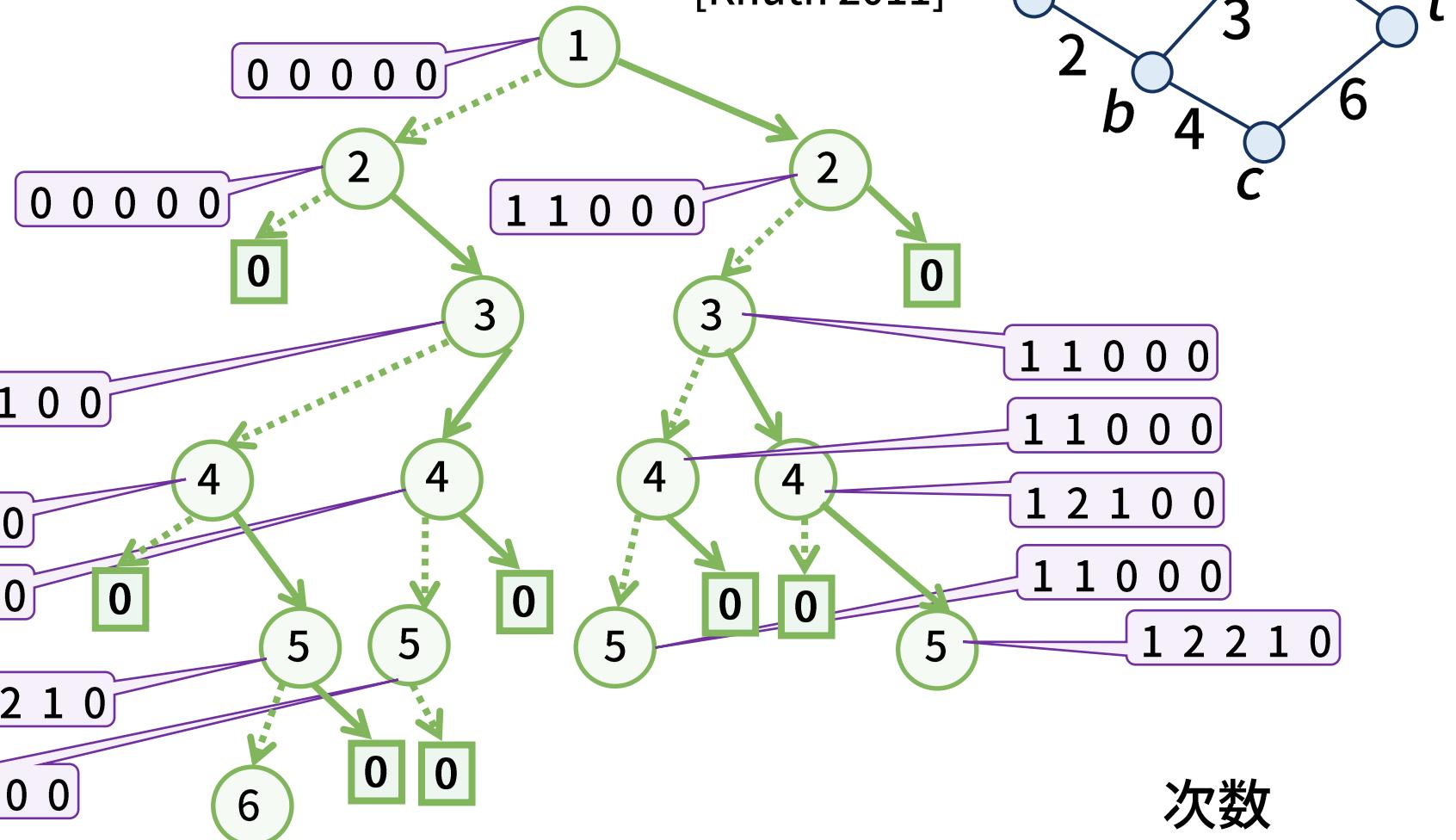
# パス集合を表す ZDD の構築

[Knuth 2011]



# パス集合を表す ZDD の構築

[Knuth 2011]

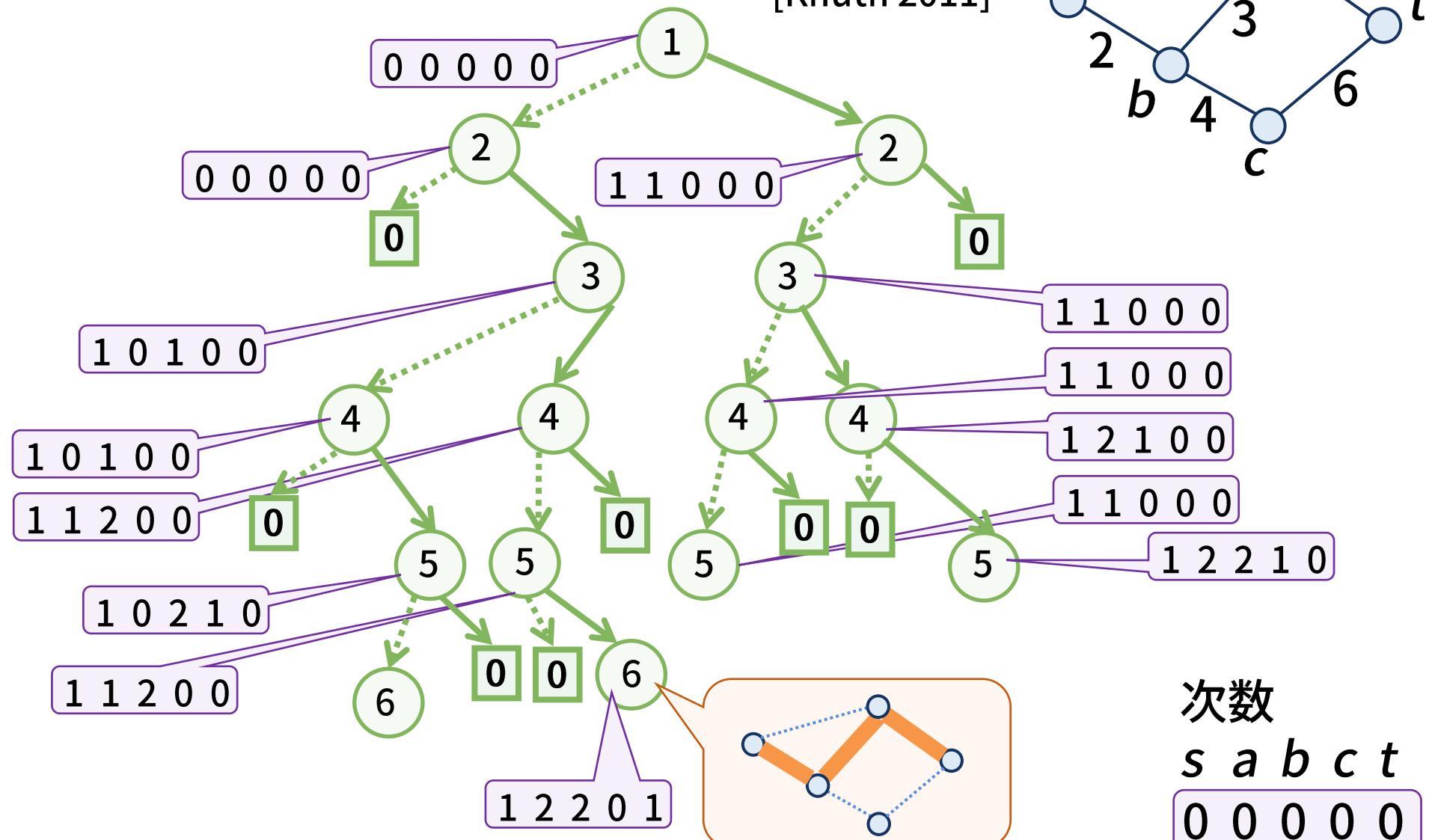


次数  
 $s \ a \ b \ c \ t$

0 0 0 0 0

# パス集合を表す ZDD の構築

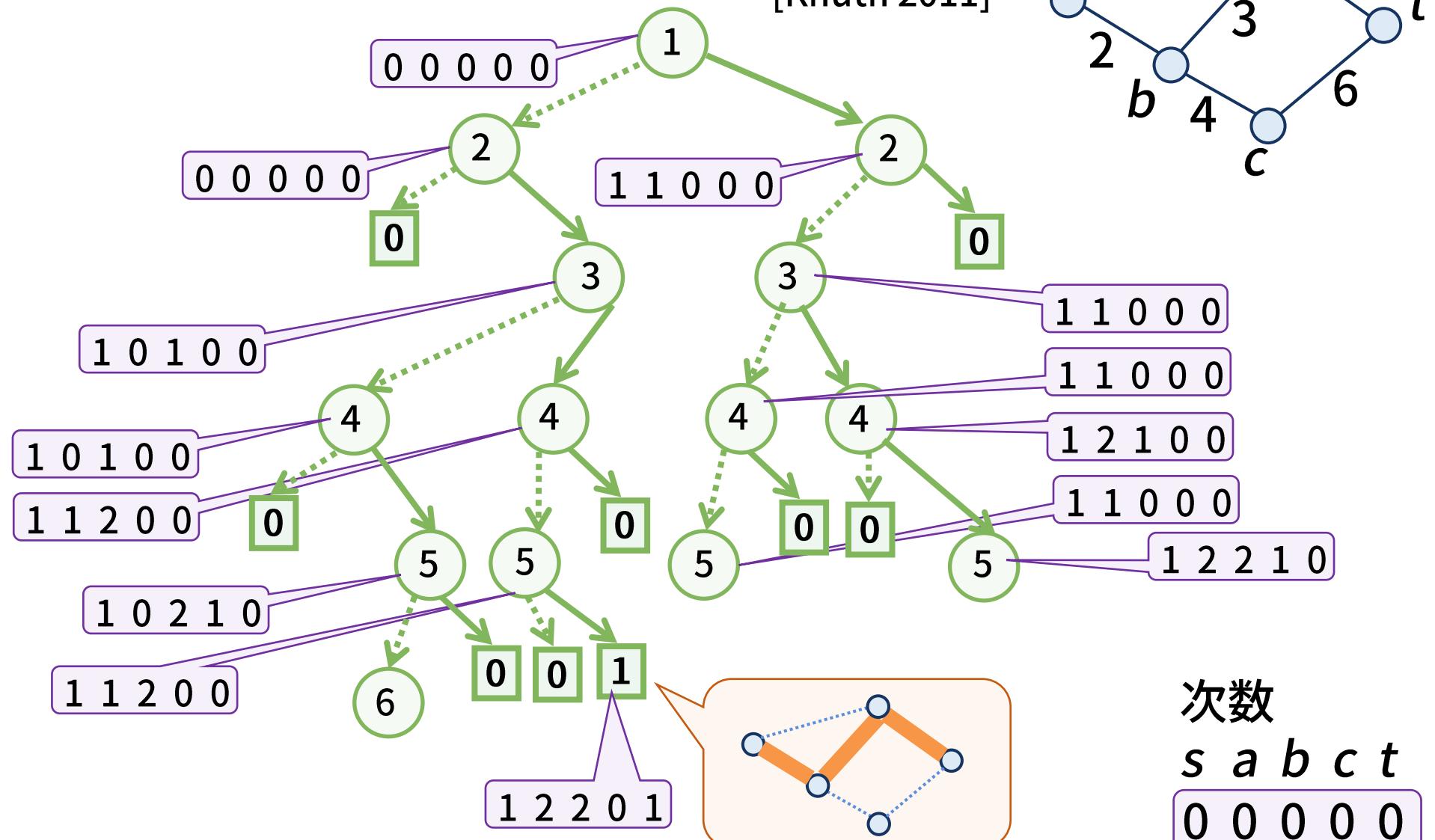
[Knuth 2011]



$s-t$  パスが完成  $\rightarrow$  1-終端につなぐ

# パス集合を表す ZDD の構築

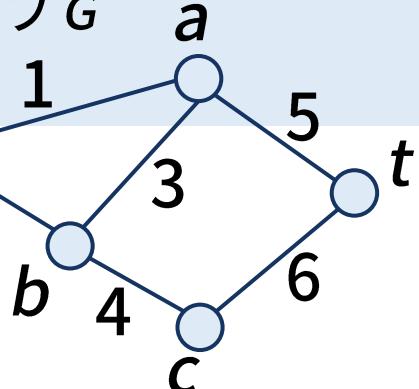
[Knuth 2011]



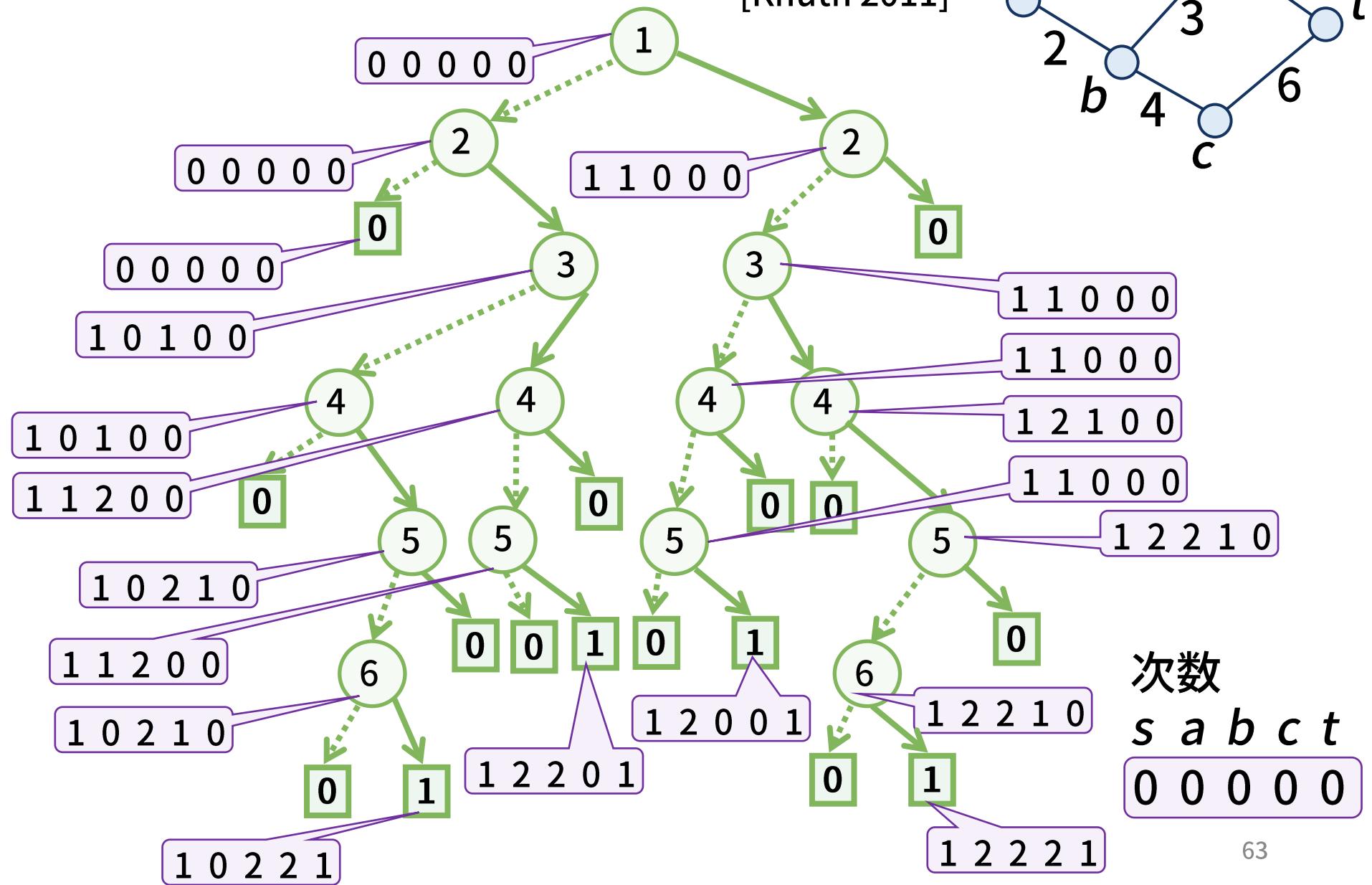
$s-t$  パスが完成  $\rightarrow$  1-終端につなぐ

# パス集合を表す ZDD の構築

## 入力グラフ $G$



[Knuth 2011]



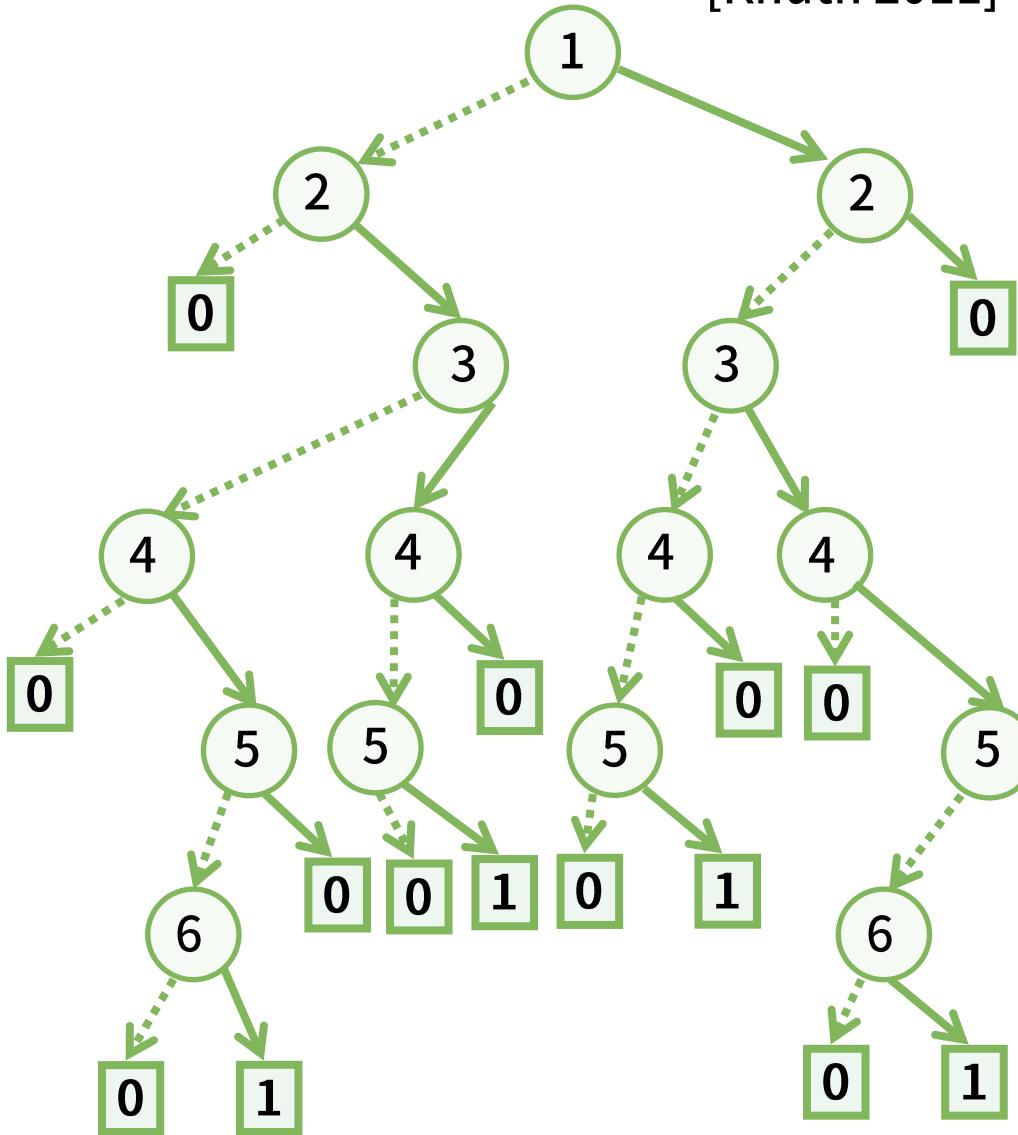
次数  
s a b c t

0 0 0 0 0

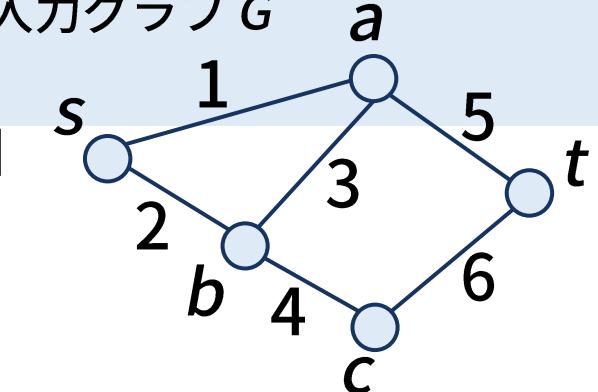
© 2010 Pearson Education, Inc. All Rights Reserved. May not be reproduced without permission.

# パス集合を表す ZDD の構築

[Knuth 2011]



入力グラフ  $G$



木が完成した。

これも一応 ZDD と  
みなせる  
(0-終端、1-終端は  
実際には 1 つ)

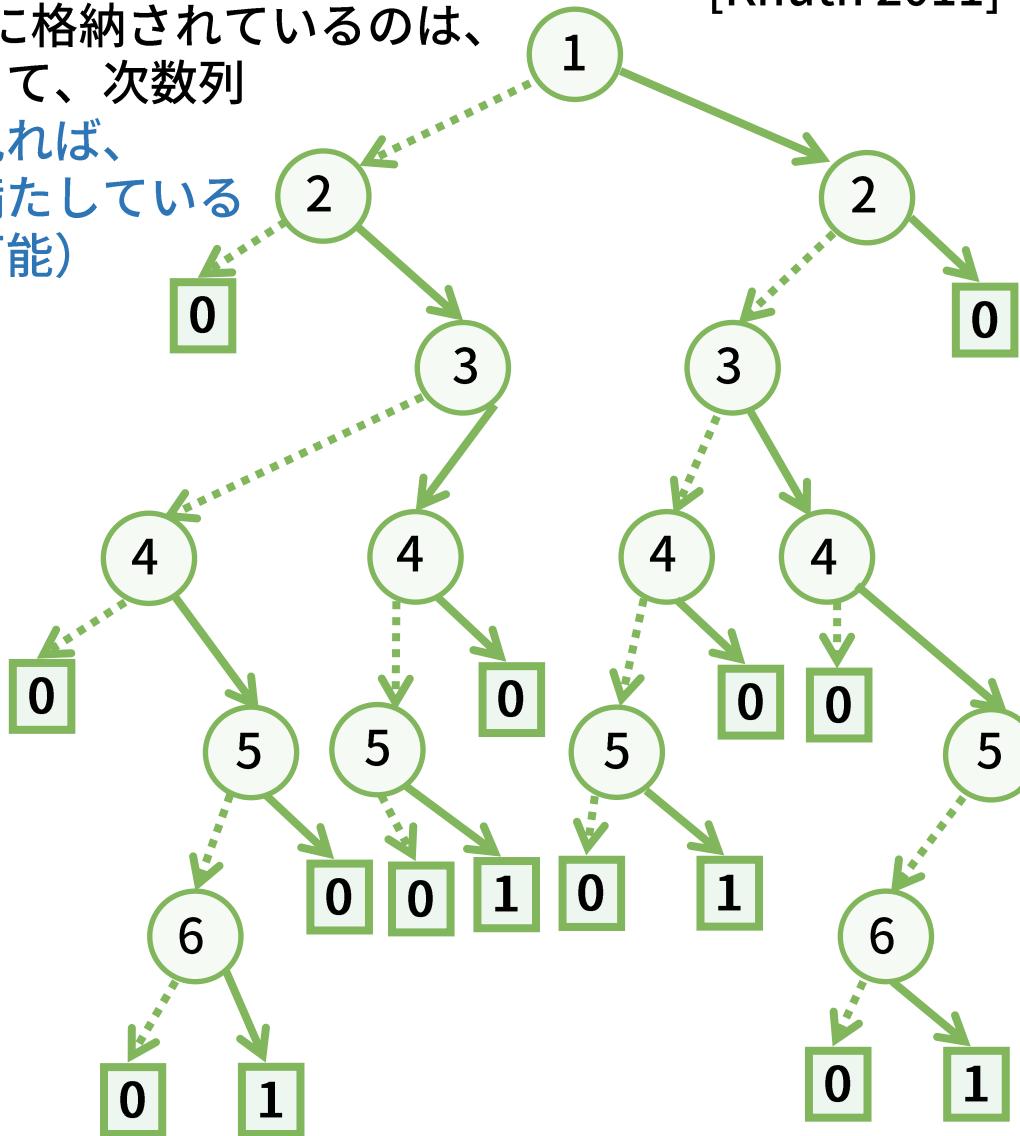
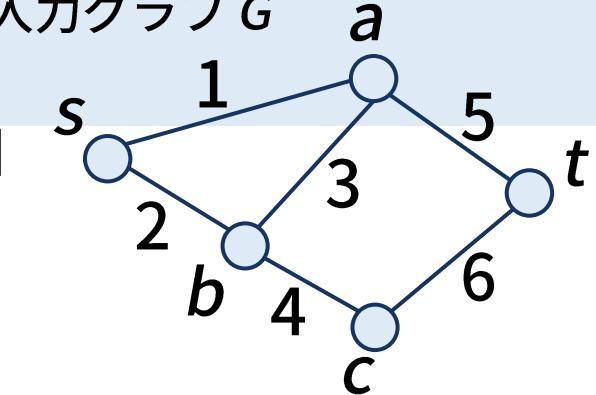
# パス集合を表す ZDD の構築

観察: ノードに格納されているのは、  
パスではなくて、次数列

(次数列を見れば、  
次数条件を満たしている  
ことを判定可能)

[Knuth 2011]

入力グラフ  $G$



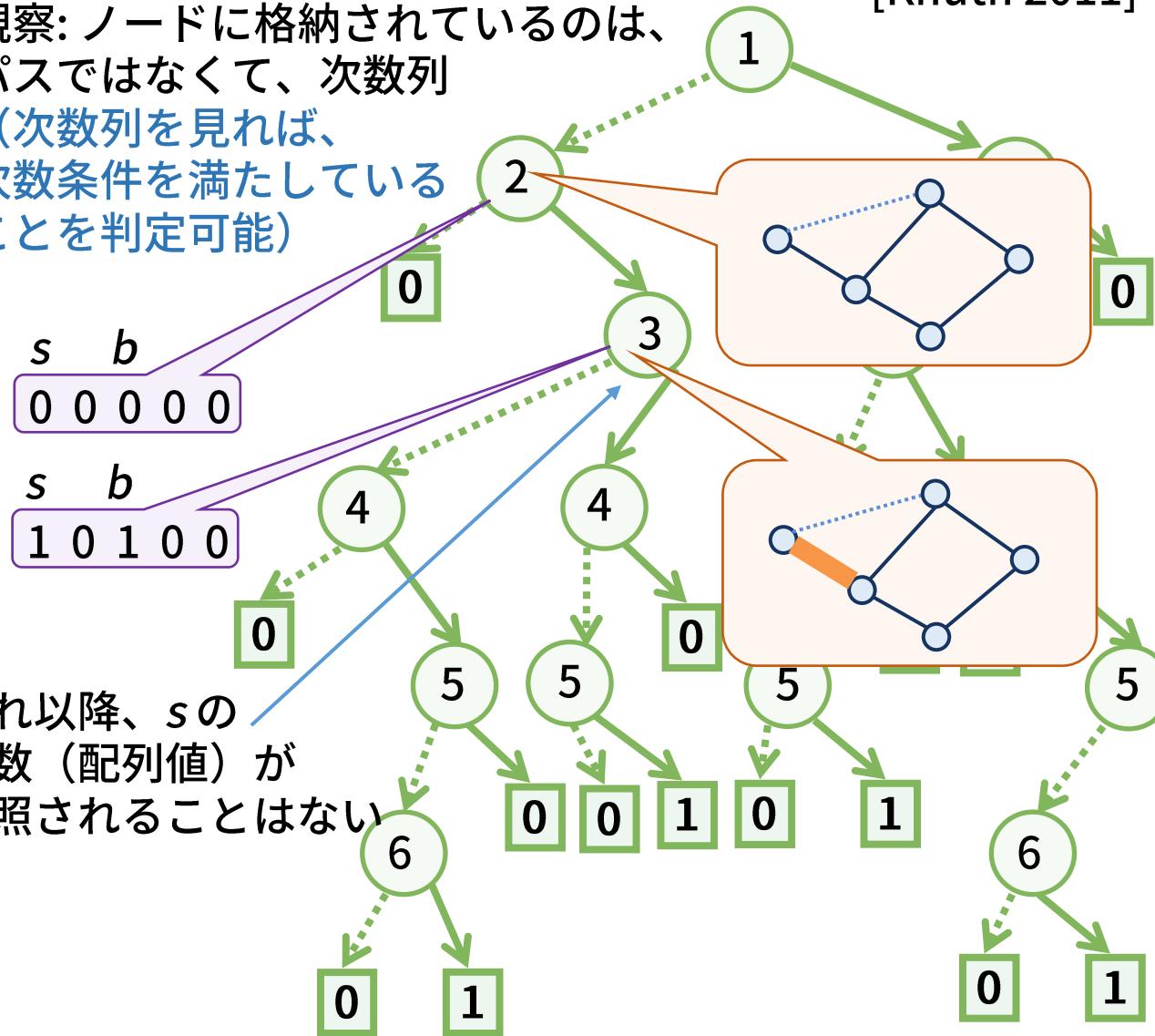
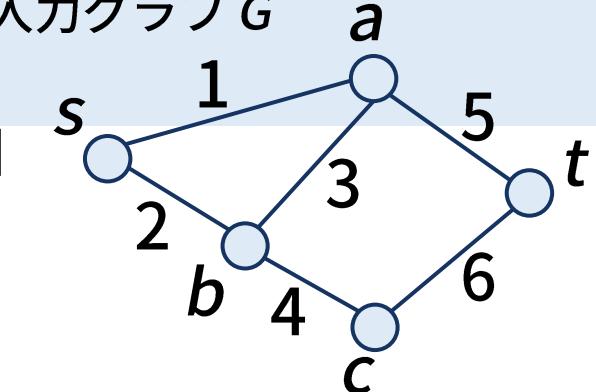
# パス集合を表す ZDD の構築

観察: ノードに格納されているのは、  
パスではなくて、次数列

(次数列を見れば、  
次数条件を満たしている  
ことを判定可能)

[Knuth 2011]

入力グラフ  $G$



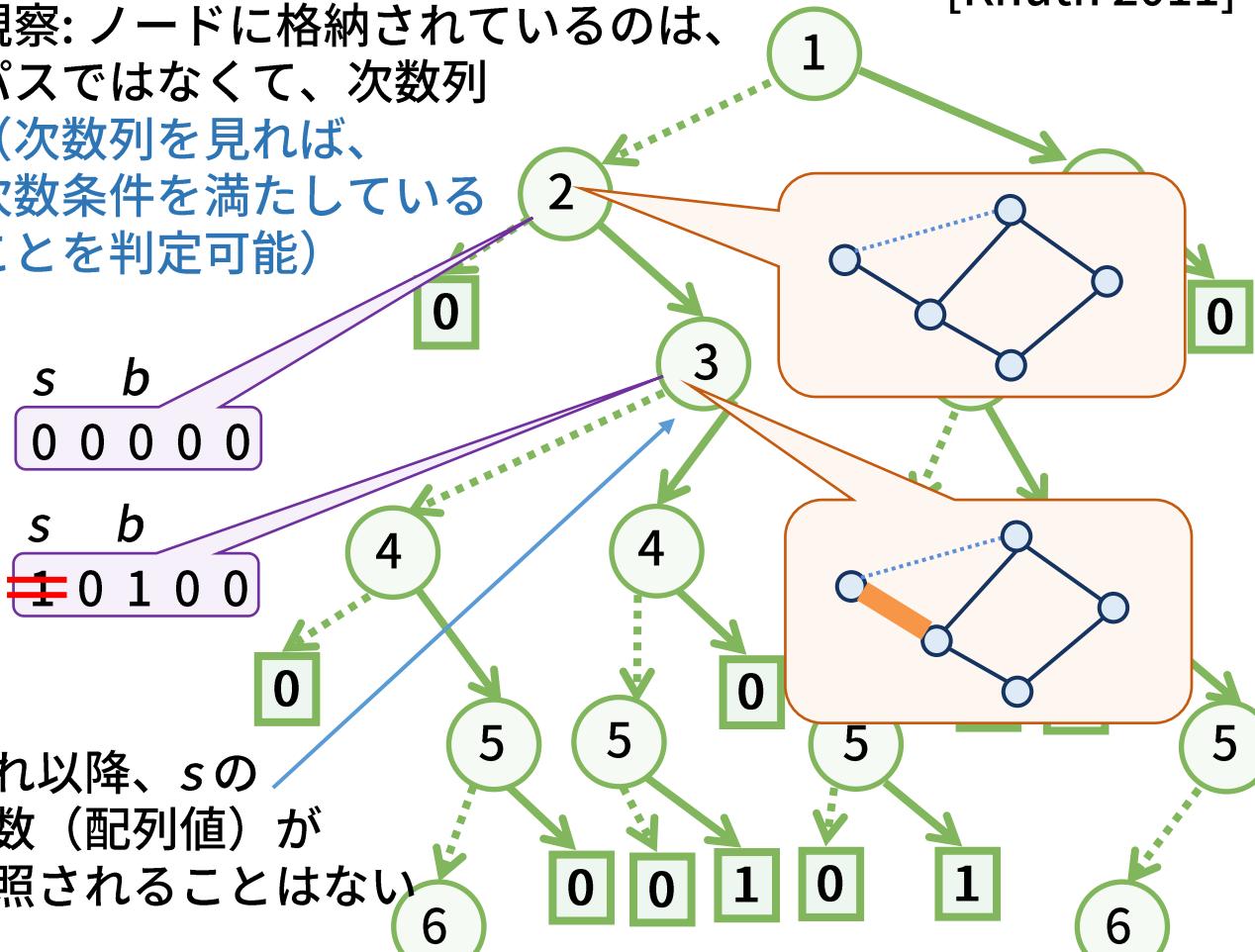
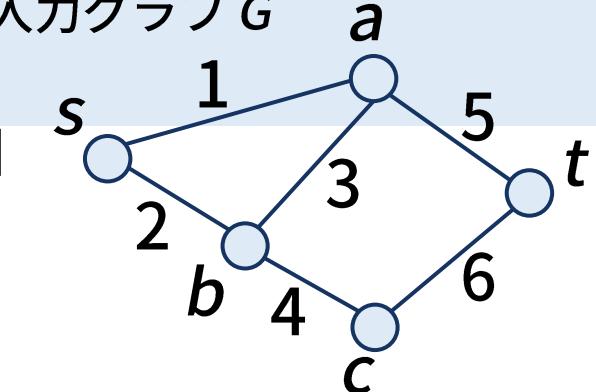
# パス集合を表す ZDD の構築

観察: ノードに格納されているのは、  
パスではなくて、次数列

(次数列を見れば、  
次数条件を満たしている  
ことを判定可能)

[Knuth 2011]

入力グラフ  $G$



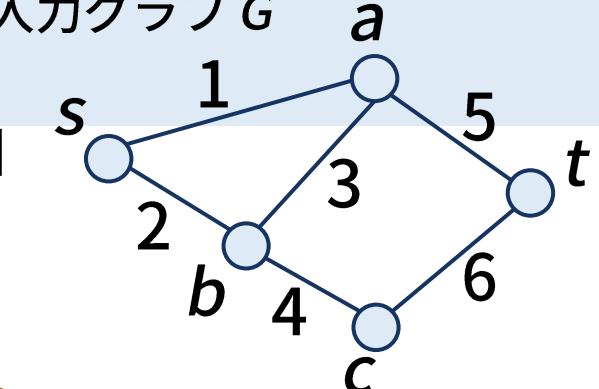
これ以降、 $s$ の  
次数（配列値）が  
参照されることはない

したがって、  
 $s$ に接続する辺が  
すべて処理された時点で、 $s$ の値を忘れて（削除して）よい

# パス集合を表す ZDD の構築

[Knuth 2011]

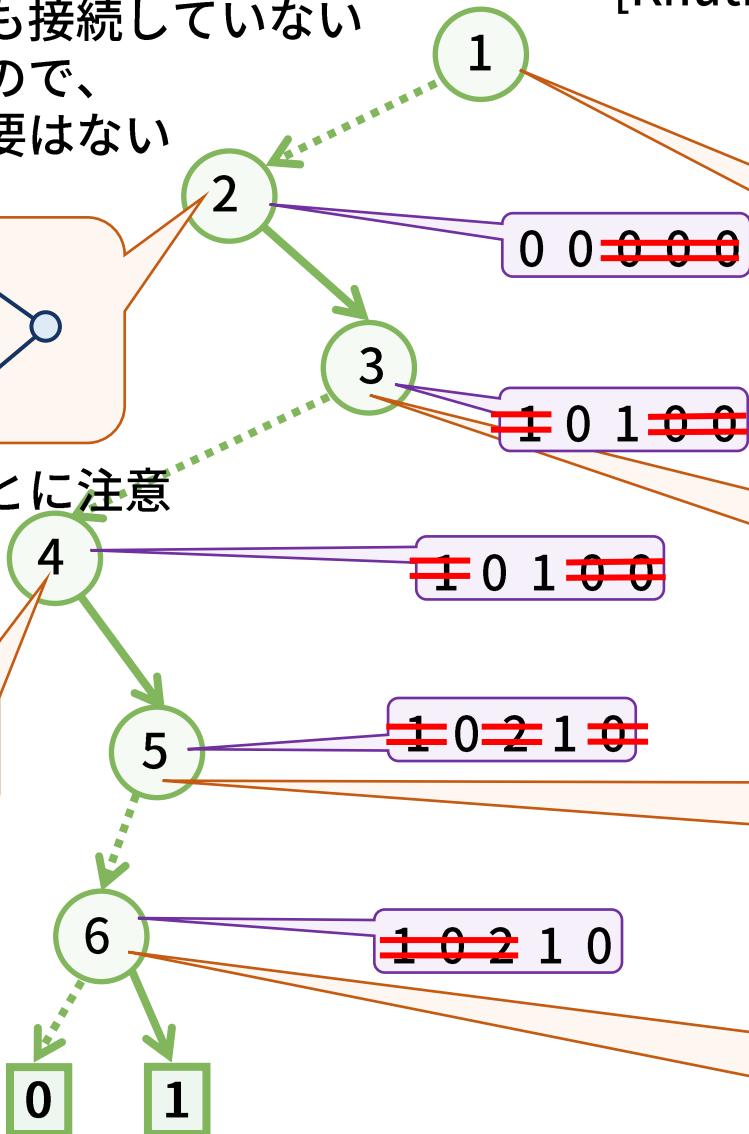
入力グラフ  $G$



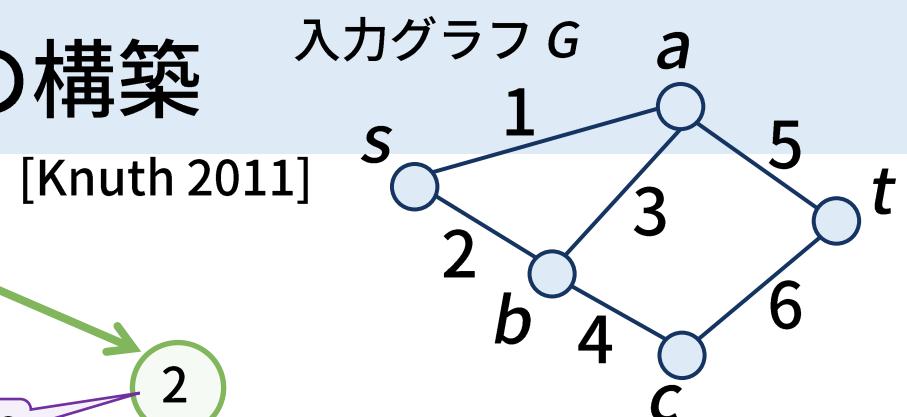
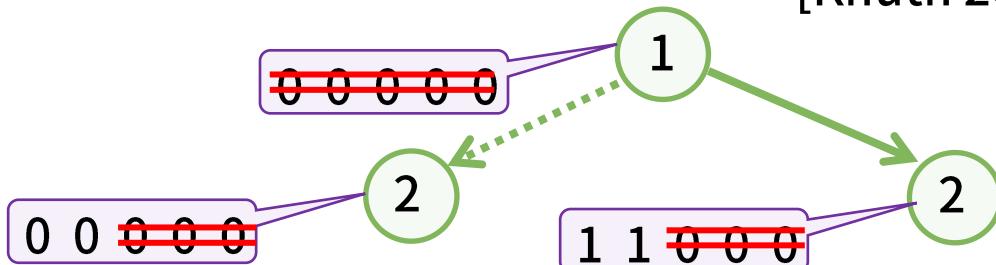
処理した辺が 1 つも接続していない場合、次数は 0 なので、これも記憶する必要はない

パスは実際には格納していないことに注意

$s$  に接続する辺がすべて処理された時点で、 $s$  の値を忘れて（削除して）よい



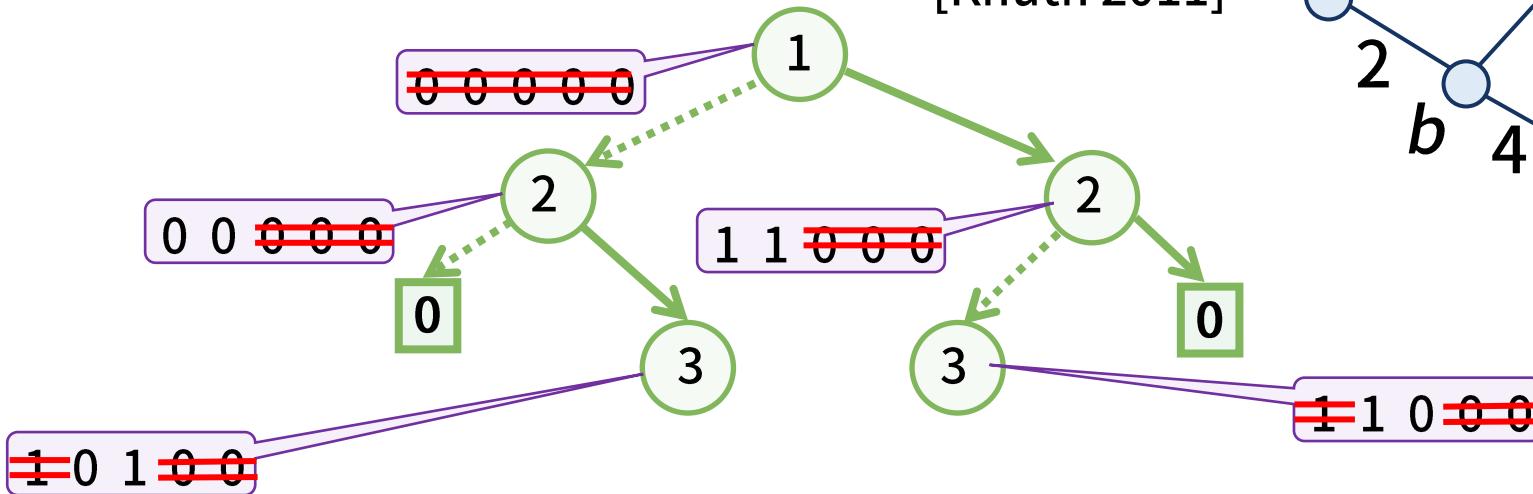
# パス集合を表す ZDD の構築



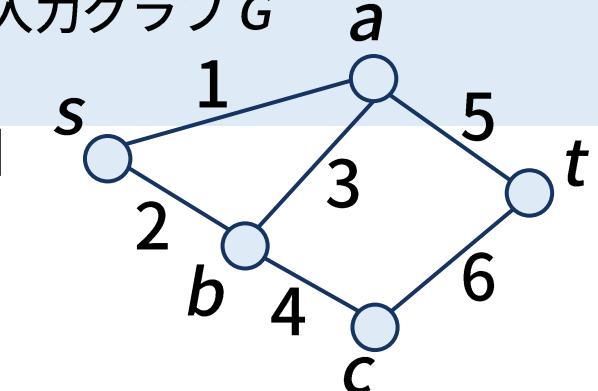
次数  
 $s \ a \ b \ c \ t$   
0 0 0 0 0

# パス集合を表す ZDD の構築

[Knuth 2011]



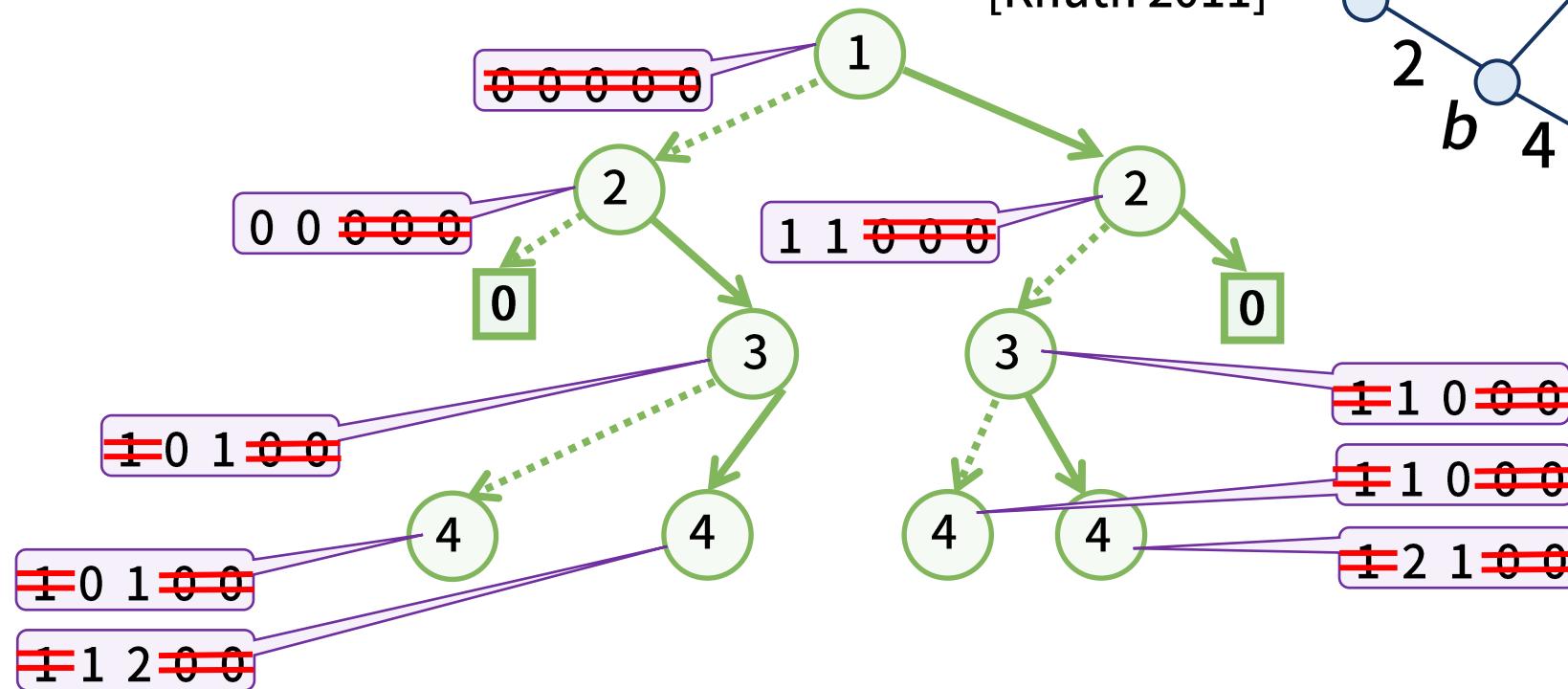
入力グラフ  $G$



次数	$s$	$a$	$b$	$c$	$t$
00000	0	0	0	0	0

# パス集合を表す ZDD の構築

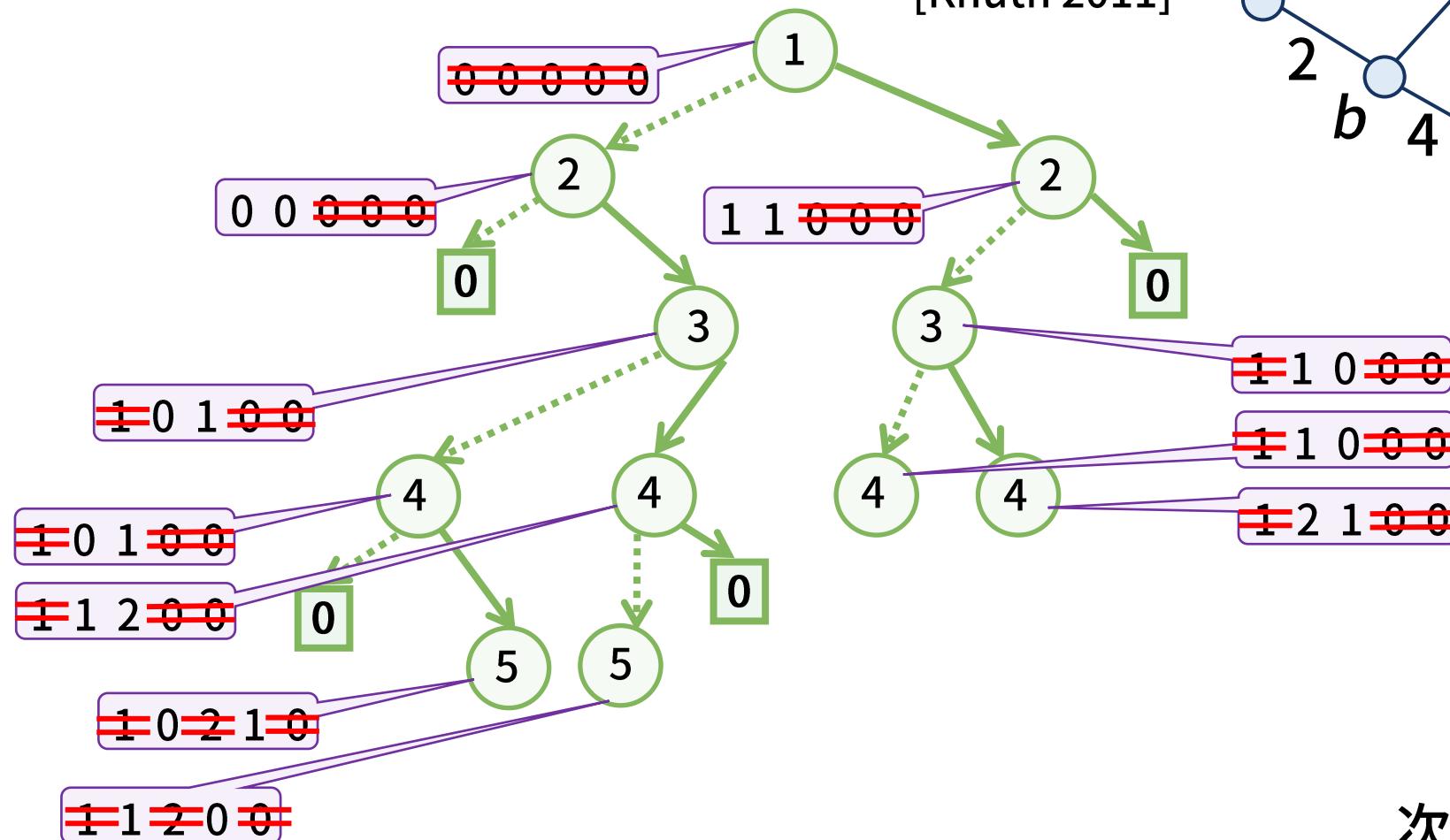
[Knuth 2011]



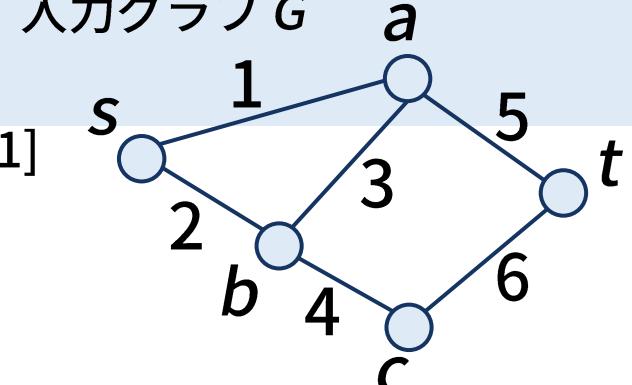
次数	$s$	$a$	$b$	$c$	$t$
$0\ 0\ 0\ 0\ 0$	0	0	0	0	0

# パス集合を表す ZDD の構築

[Knuth 2011]



入力グラフ  $G$



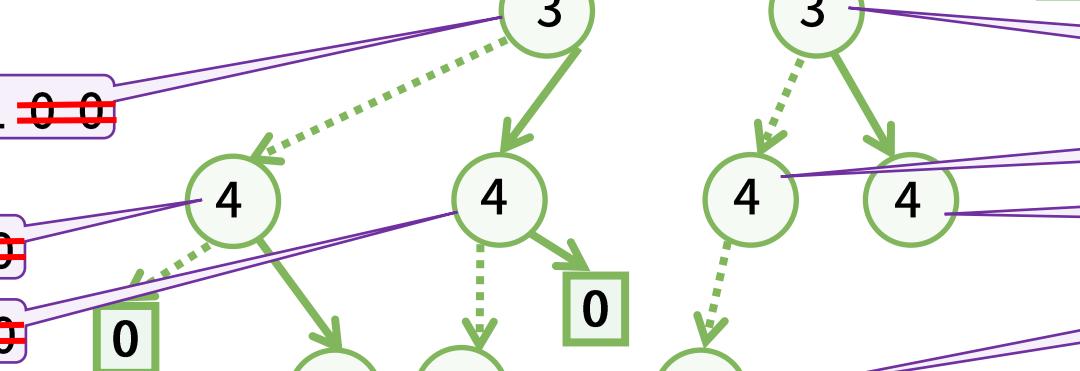
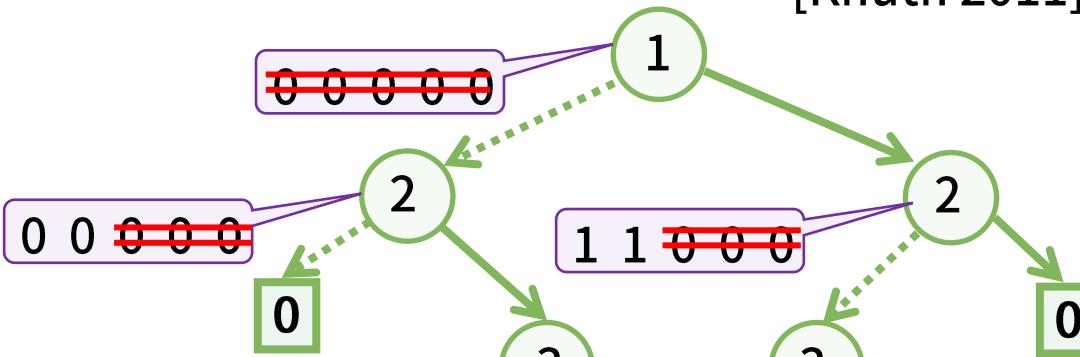
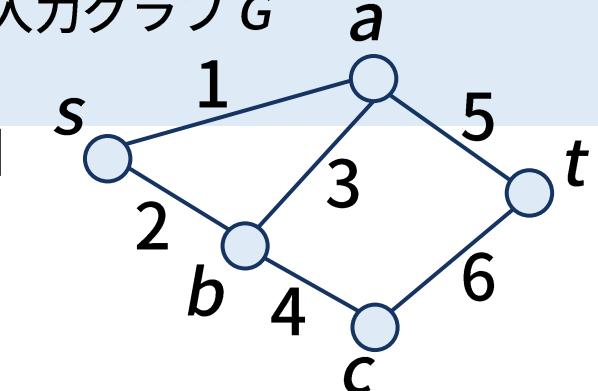
次数  
 $s \ a \ b \ c \ t$

0 0 0 0 0

# パス集合を表す ZDD の構築

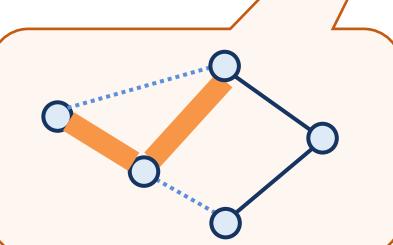
[Knuth 2011]

入力グラフ  $G$



この 2 つのノードに着目。  
配列値（削除後）が一致

パスは異なるが、  
どちらも辺 5 を使用、  
辺 6 が不使用だと  
パスが完成

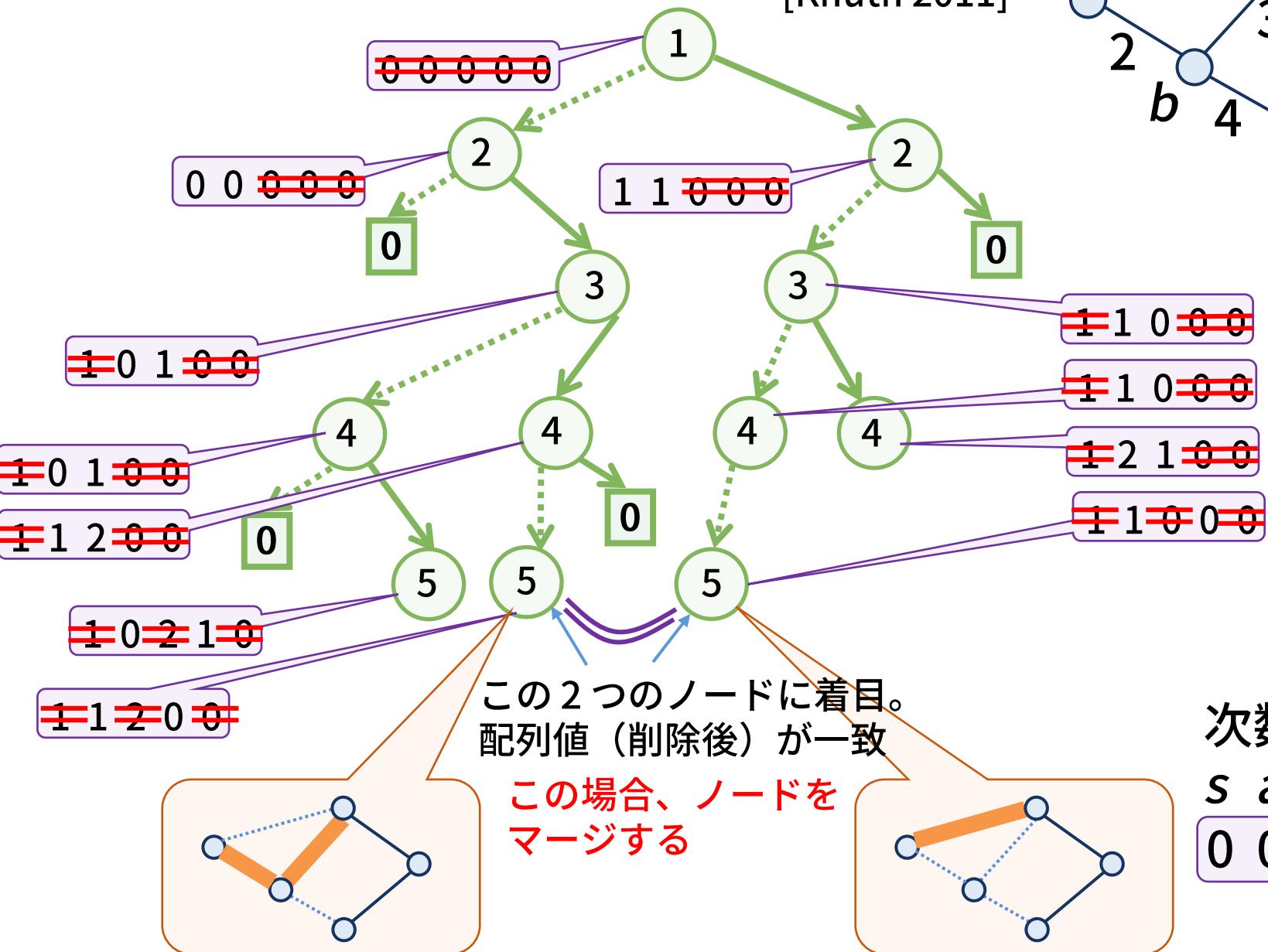
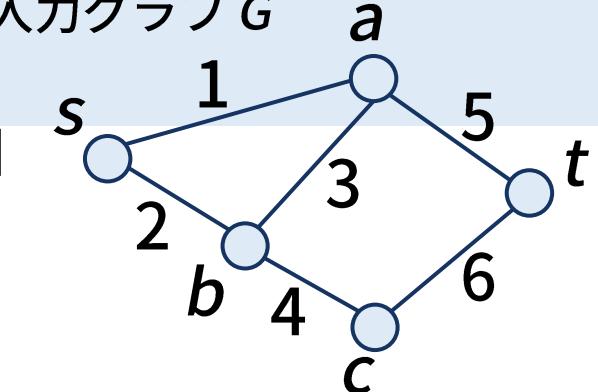


次数  
 $s\ a\ b\ c\ t$   
 $0\ 0\ 0\ 0\ 0$

# パス集合を表す ZDD の構築

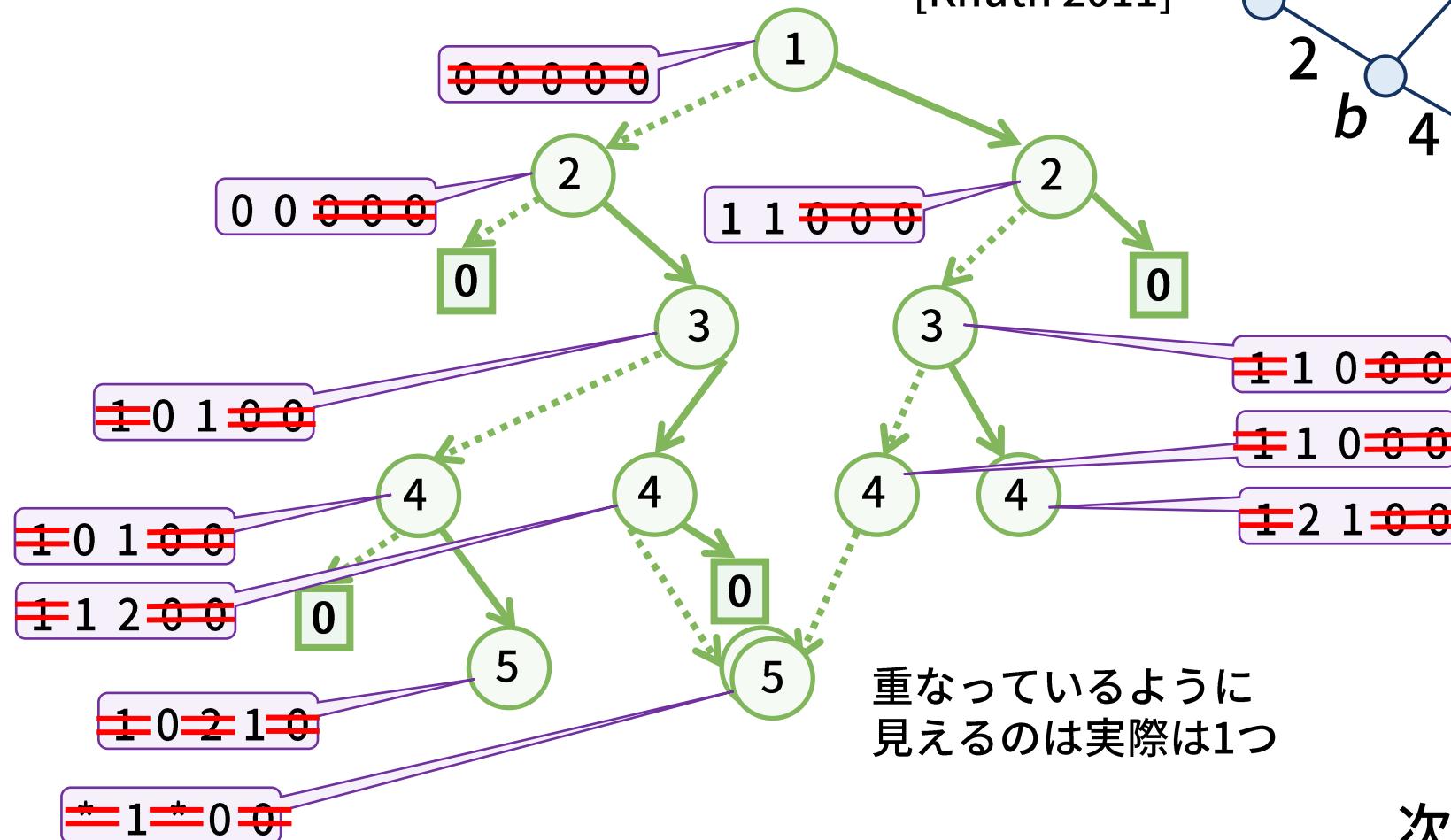
[Knuth 2011]

入力グラフ  $G$

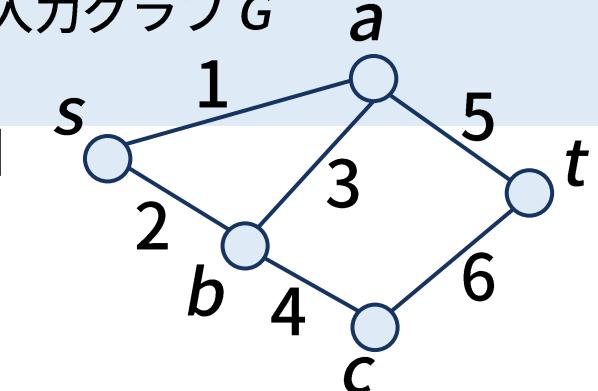


# パス集合を表す ZDD の構築

[Knuth 2011]



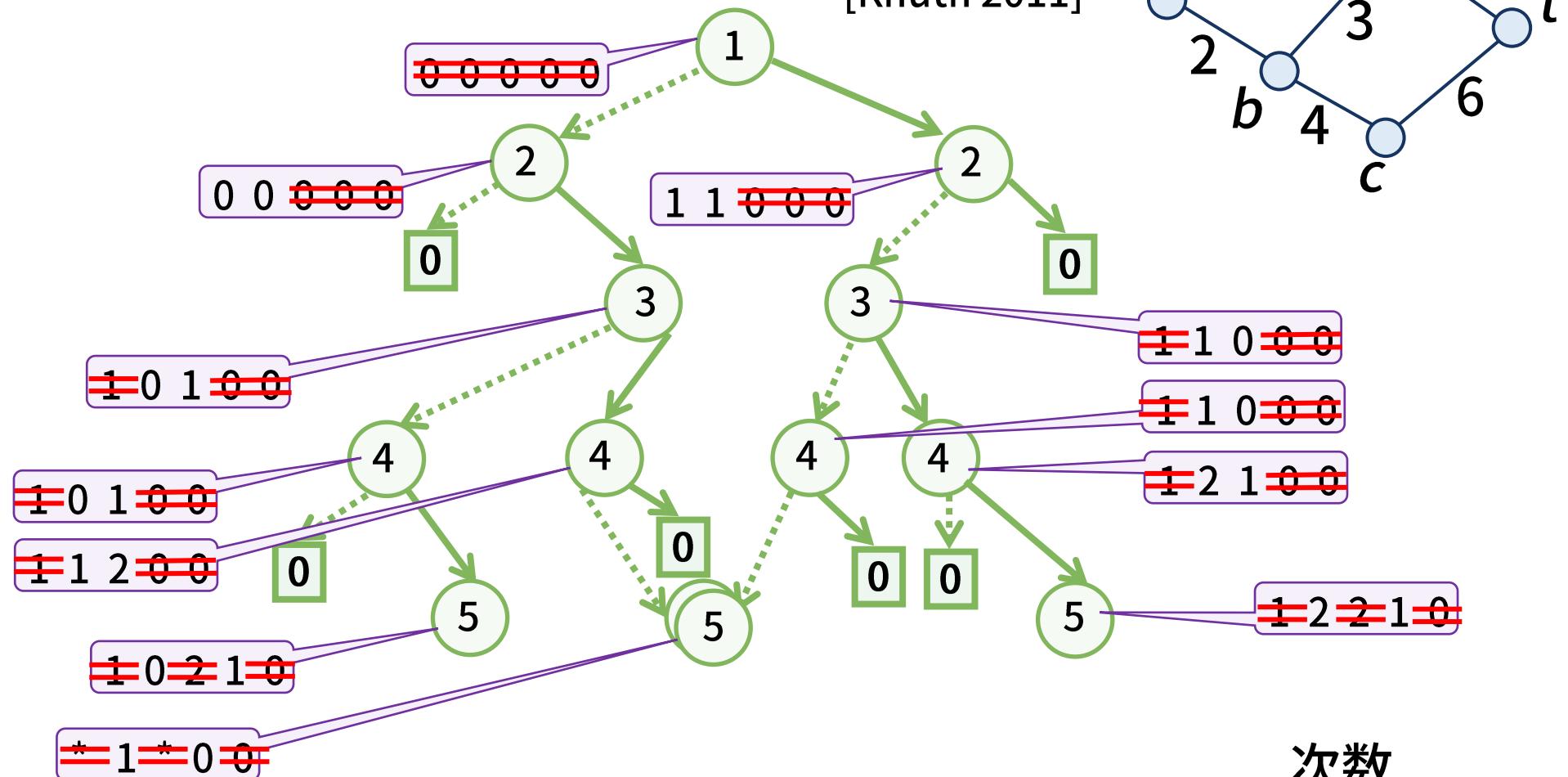
入力グラフ  $G$



次数  
 $s \ a \ b \ c \ t$   
 0 0 0 0 0

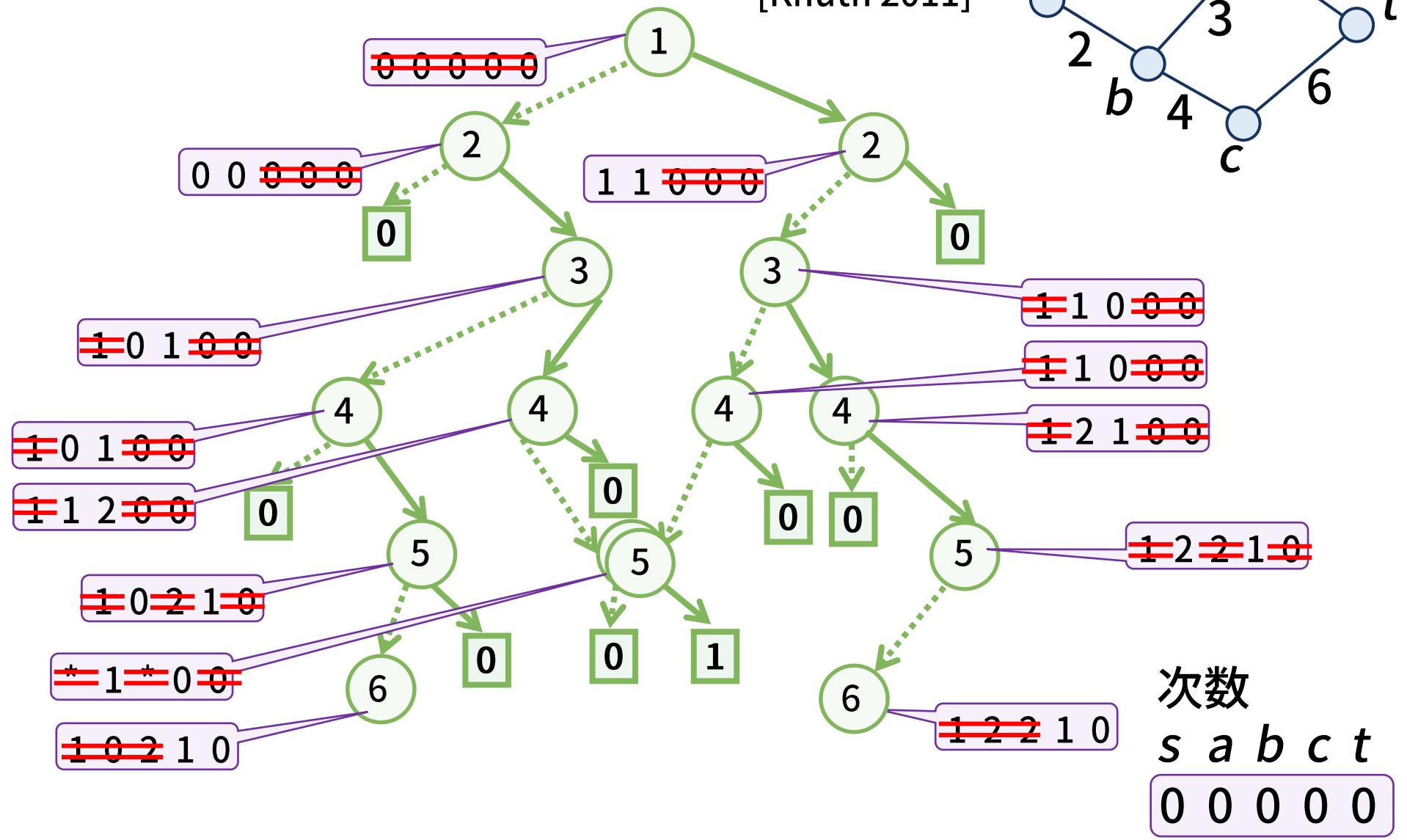
# パス集合を表す ZDD の構築

[Knuth 2011]



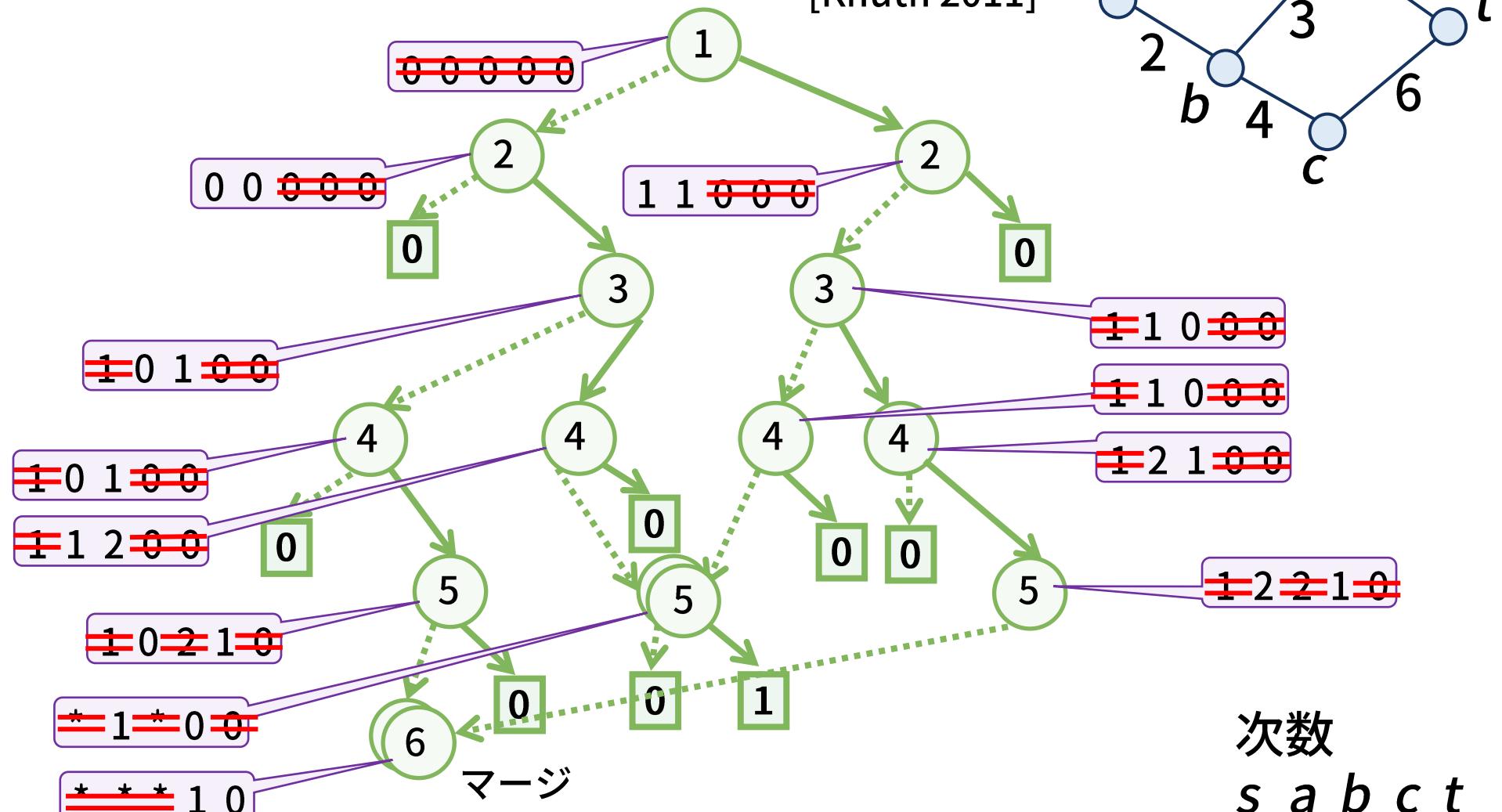
# パス集合を表す ZDD の構築

[Knuth 2011]



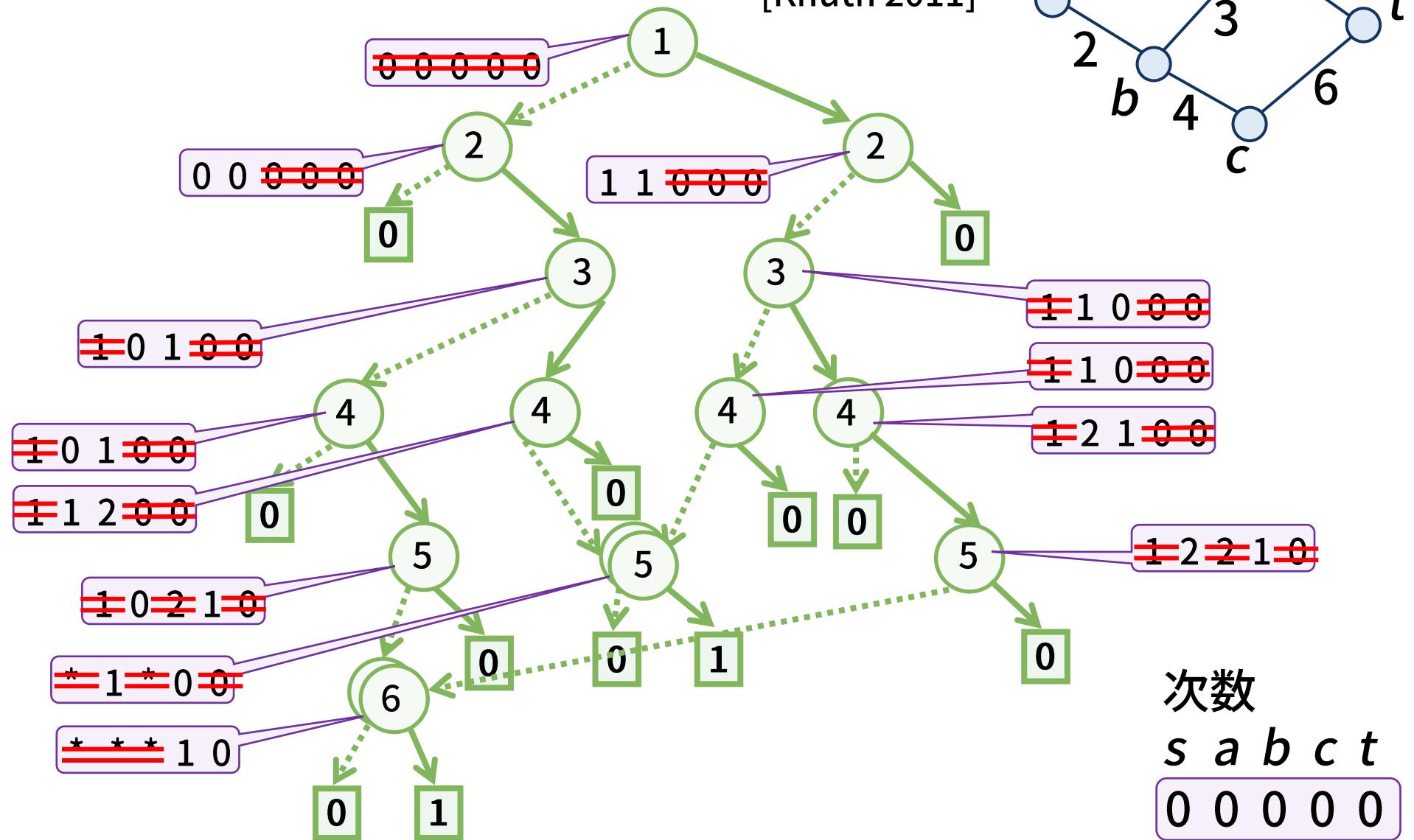
# パス集合を表す ZDD の構築

[Knuth 2011]



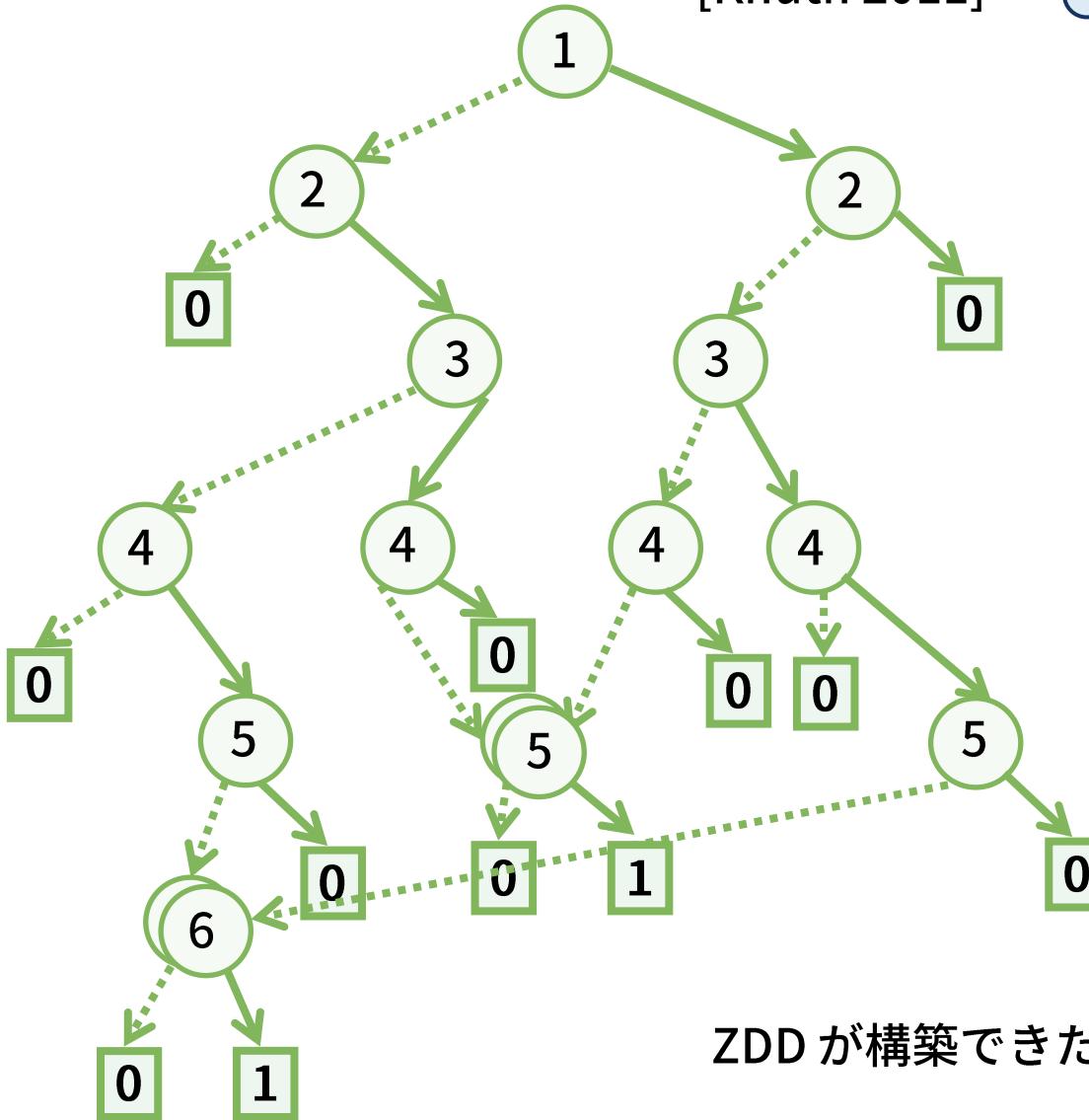
# パス集合を表す ZDD の構築

[Knuth 2011]



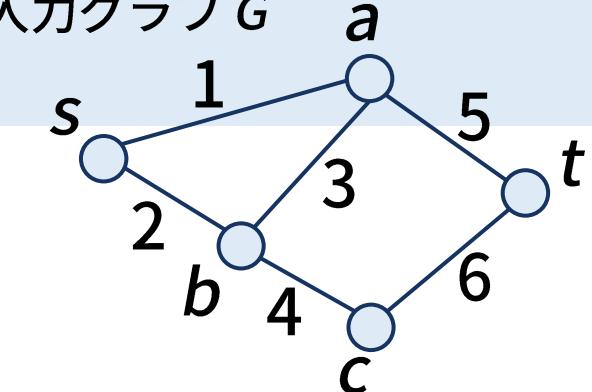
# パス集合を表す ZDD の構築

[Knuth 2011]



## ZDD が構築できた

## 入力グラフ $G$

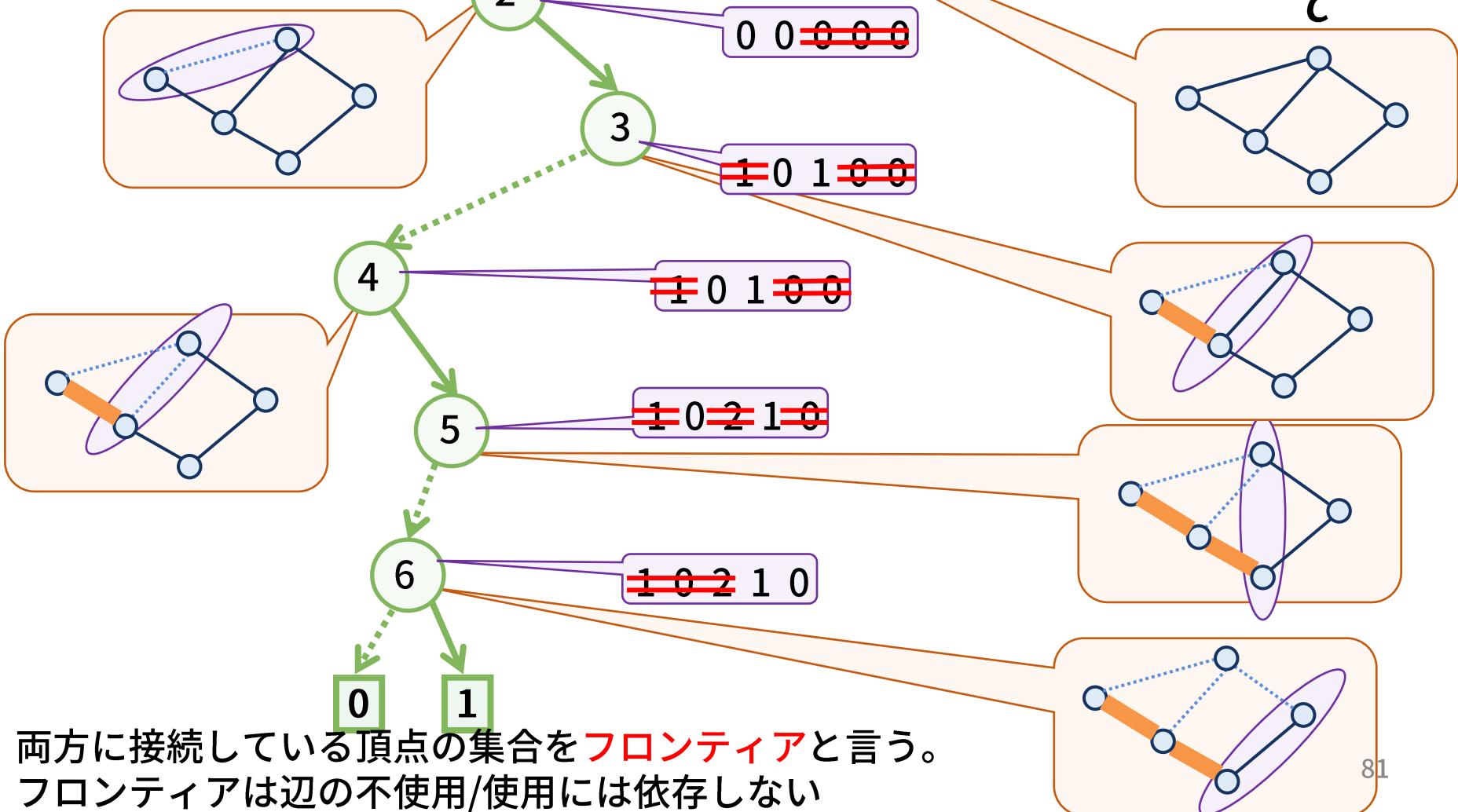
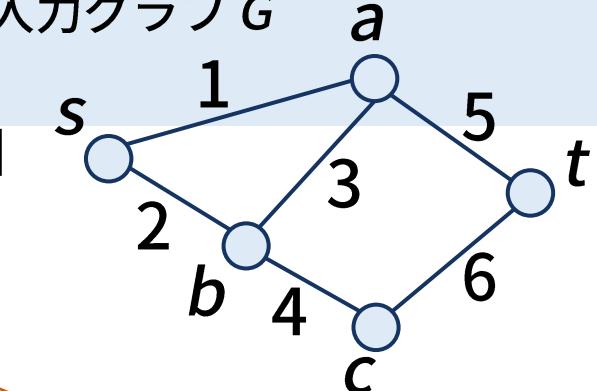


# パス集合を表す ZDD の構築

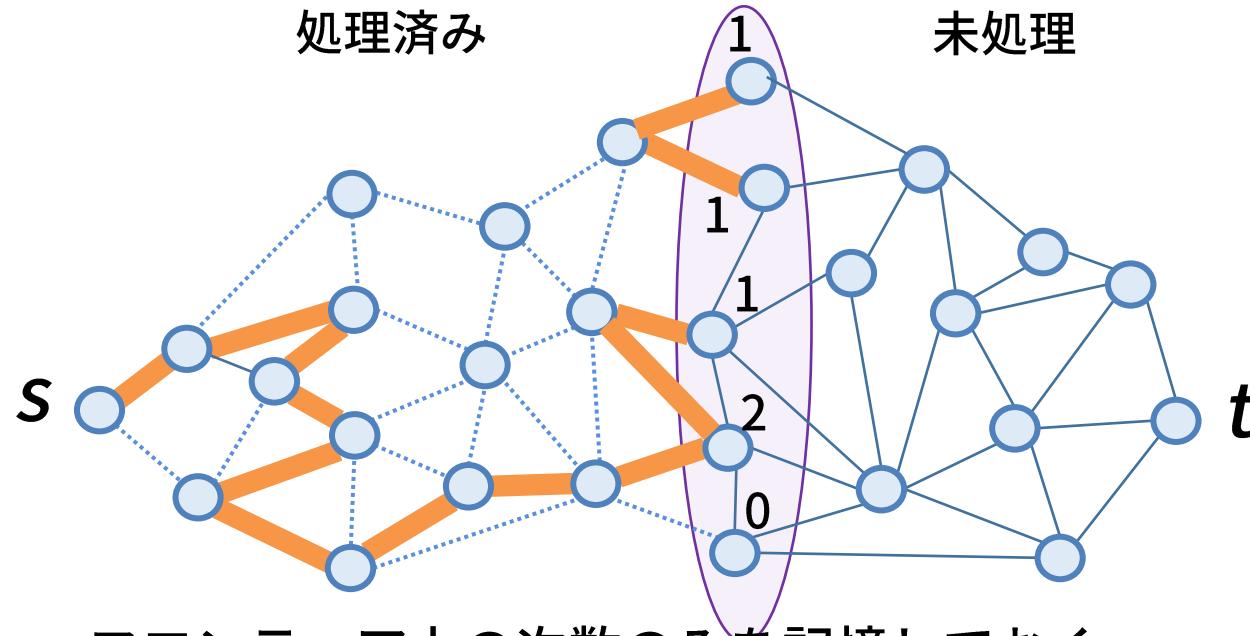
処理済みの辺（不使用/使用どちらでも）  
と未処理の辺の  
両方に接続している頂点を  
紫の楕円で示す

[Knuth 2011]

入力グラフ  $G$



# フロンティア



フロンティア上の次数のみを記憶しておく。

フロンティア上の次数が同じ2つのノードについて、  
フロンティアの前面（未処理側）の辺の選び方は、  
フロンティアの後面（処理済み側）がどのように  
選ばれているかには影響されず、フロンティア上の  
次数のみに影響される。

次数 2 の頂点にはこれ以上つなげられない。

次数 1 の頂点には後1本つなげなければならない。

次数 0 の頂点は、そのままにするか、または 2本つなげる。

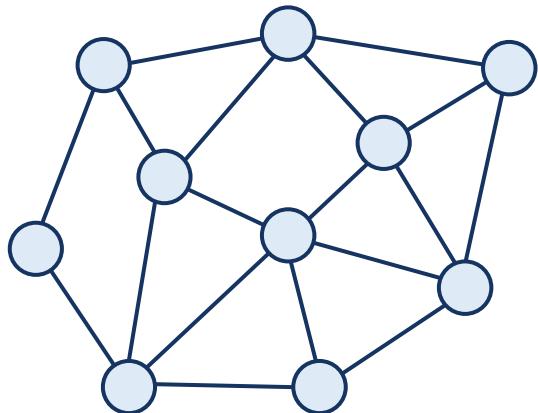
連結性は  
まだ無視している

# パスの連結性の保証

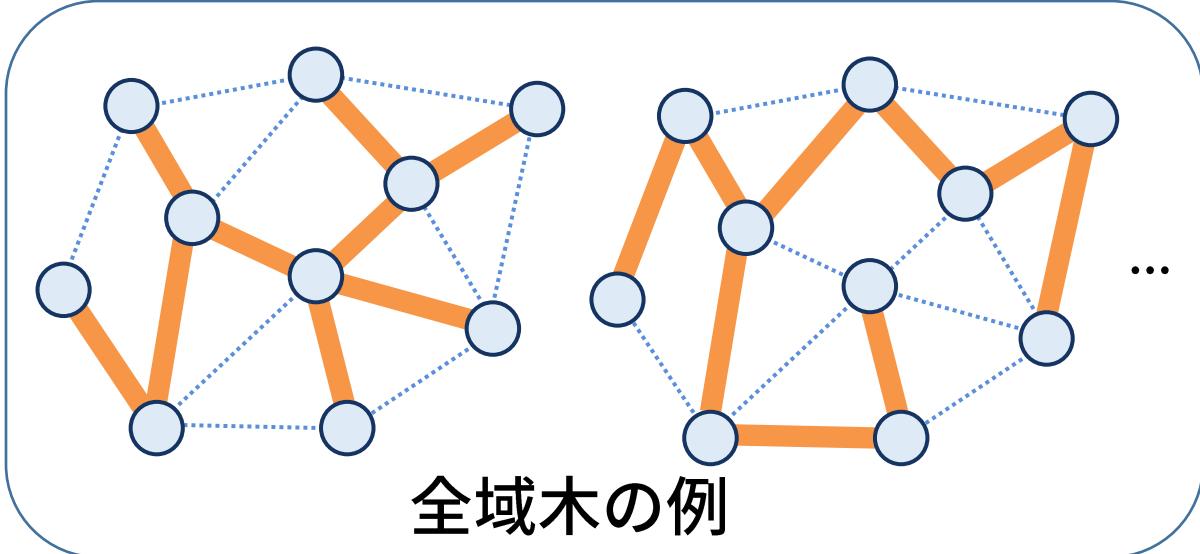
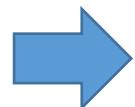
- $s-t$  パスが連結である（余分なサイクルが無い）ことを保証するには、「連結性」の情報を ZDD ノードに格納する必要がある
- $s-t$  パスは一旦置いておいて、先に、全域木の集合を ZDD で表す方法を説明する

# 全域木集合のZDD構築

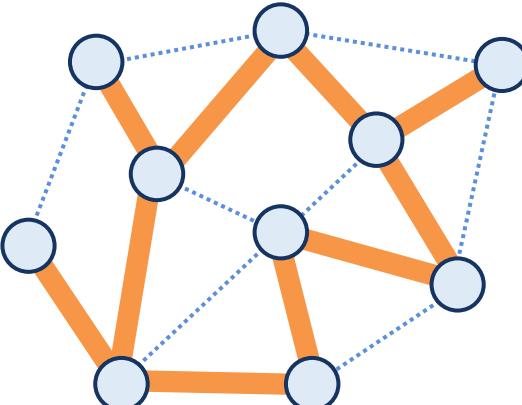
(本質的には [Sekine et al. 1995])



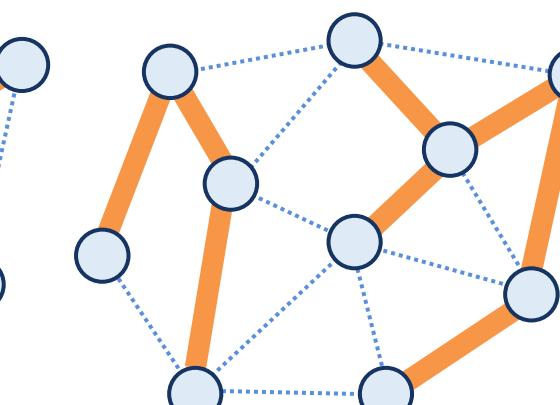
入力グラフの例



全域木の例



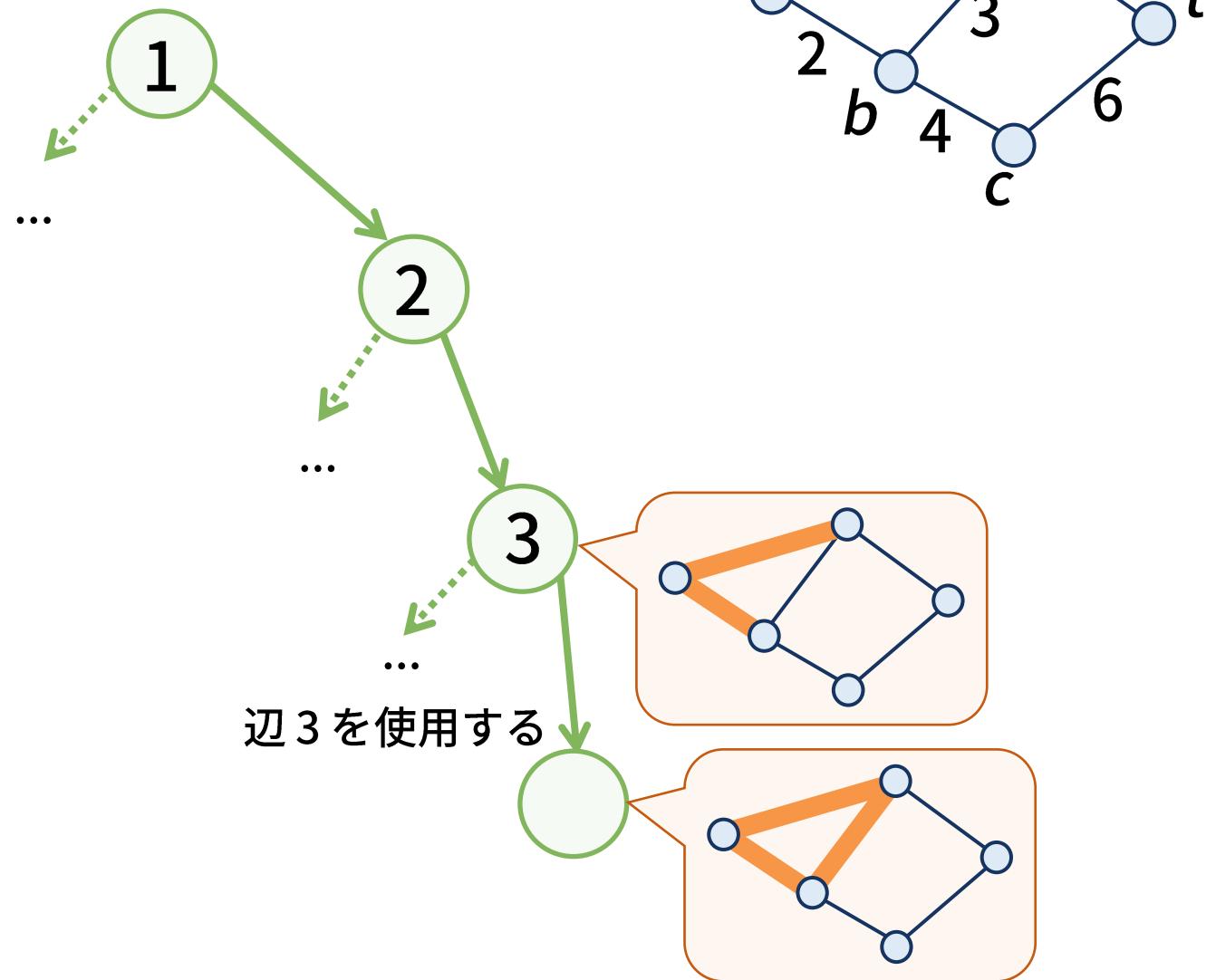
サイクル発生



連結成分が2つ以上

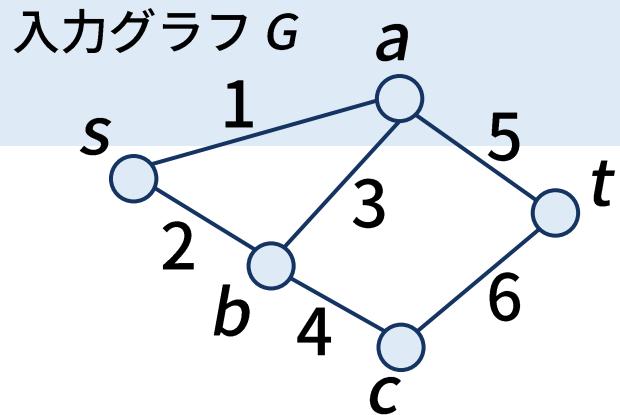
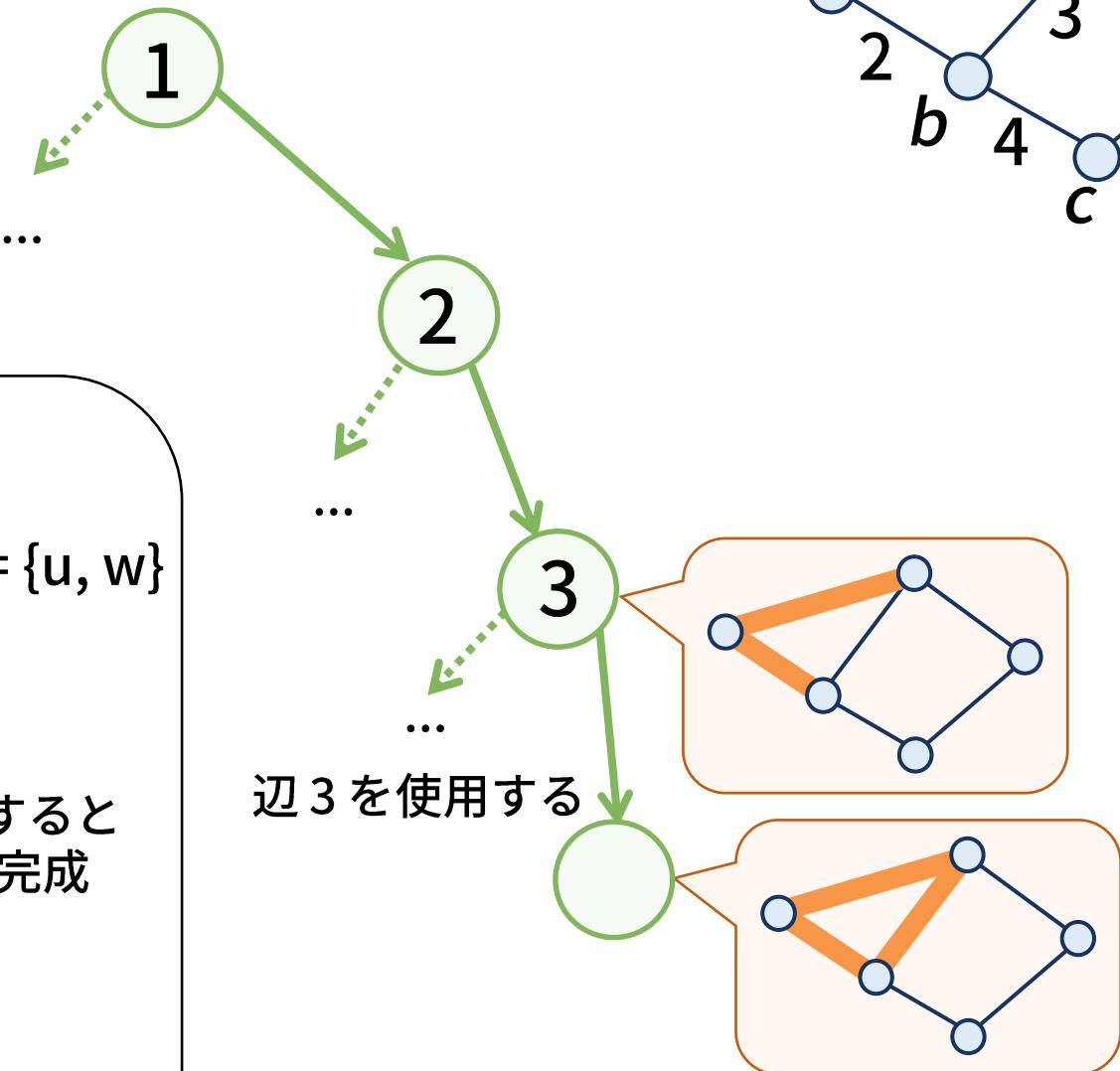
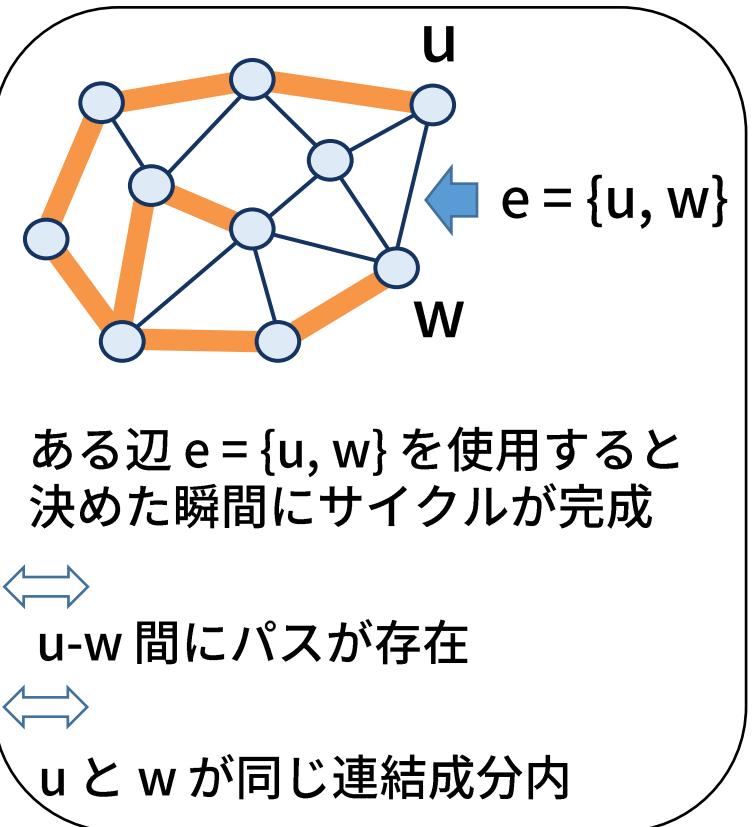
全域木ではない例

# 全域木集合のZDD構築



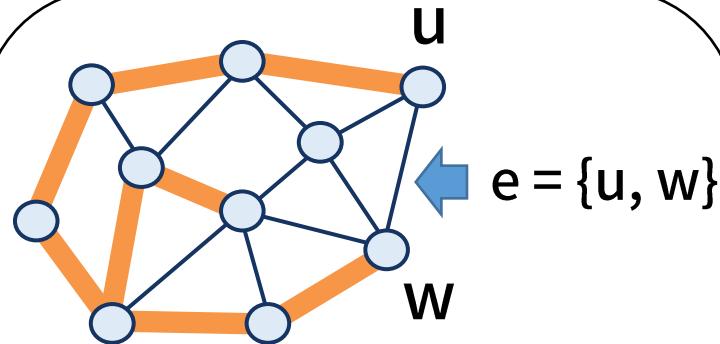
辺 3 を使用するとサイクルが生じる。  
この状況で枝刈りを行いたい

# 全域木集合のZDD構築



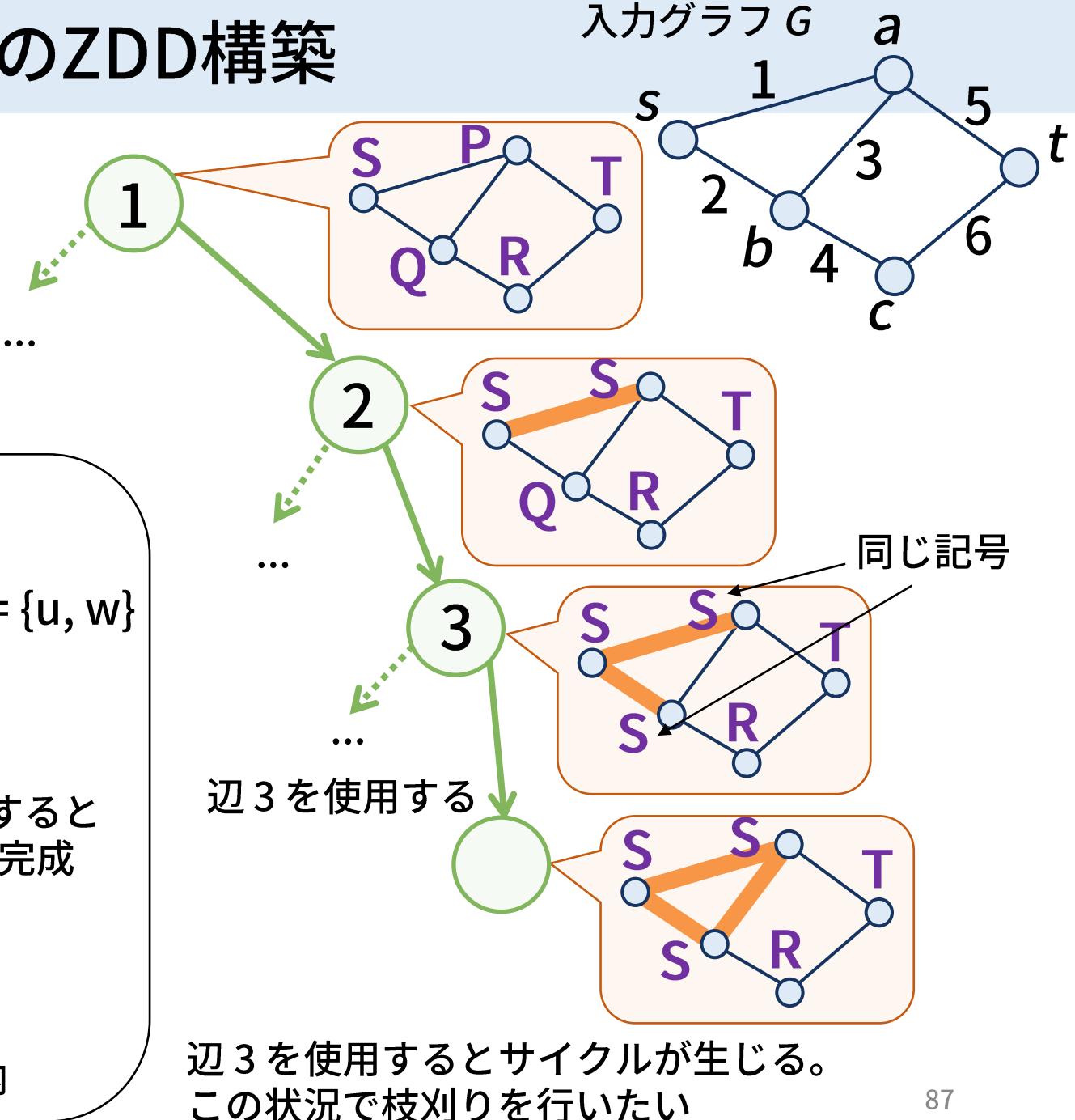
# 全域木集合のZDD構築

ZDD ノードに各頂点の連結成分を記憶させる



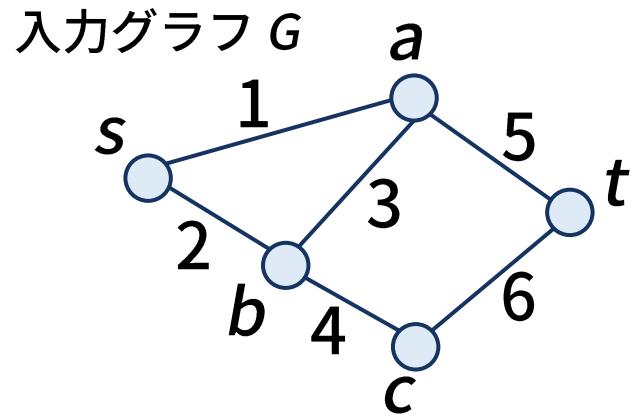
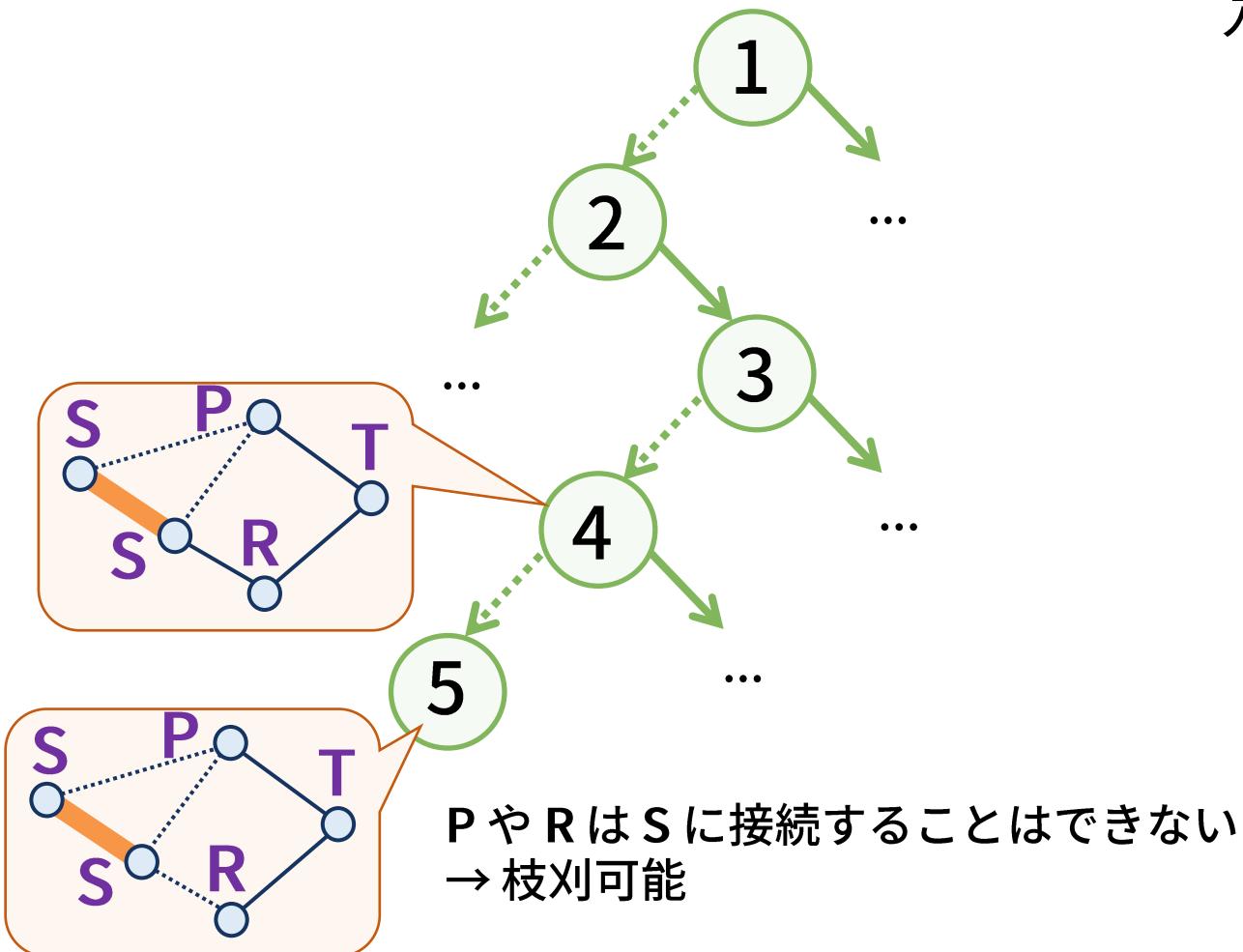
$\leftrightarrow$   
u-w 間にパスが存在

$\leftrightarrow$   
u と w が同じ連結成分内



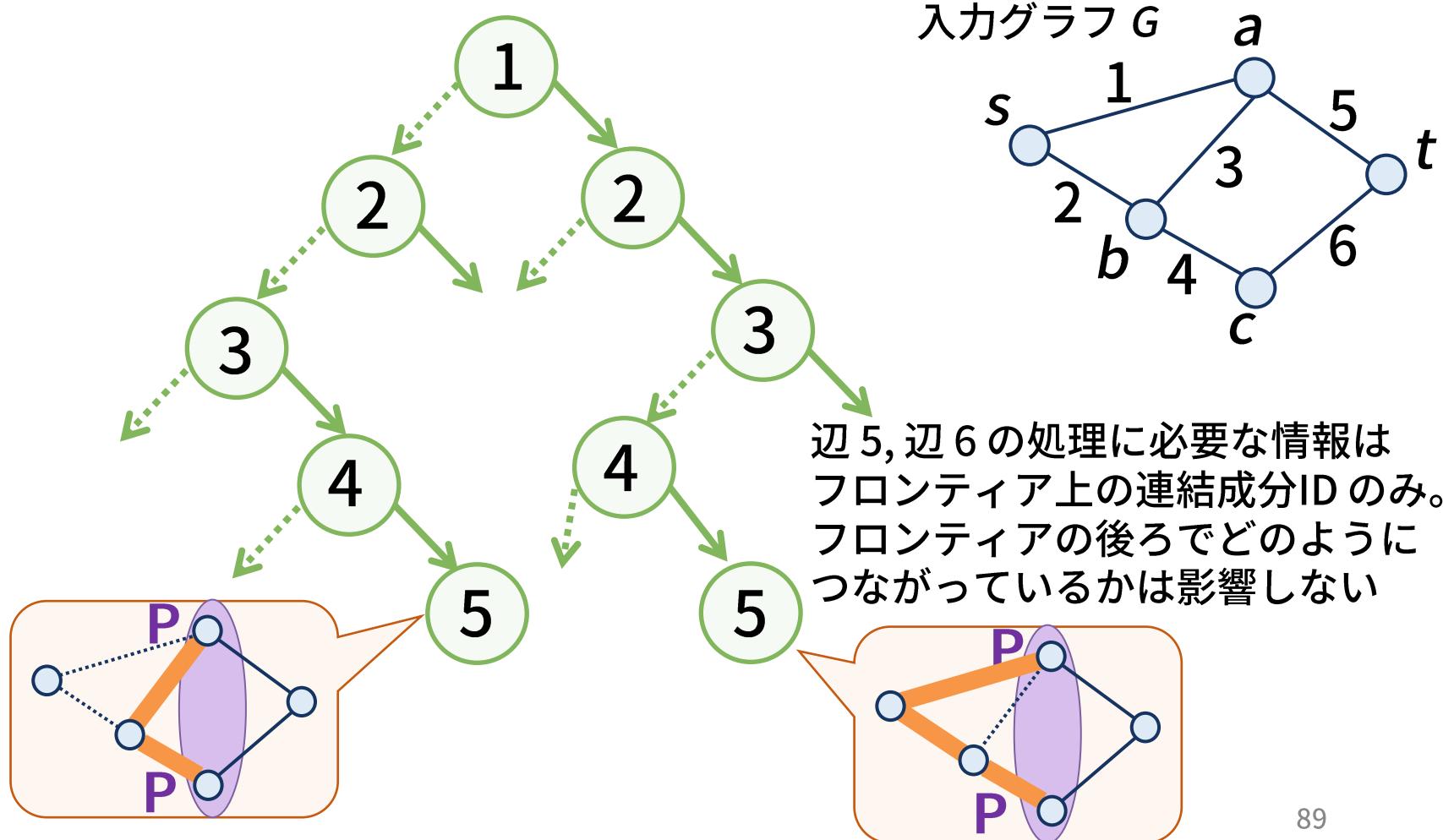
# 全域木集合のZDD構築

- ZDDノードに連結成分を記憶することで、「2つ以上の連結成分の発生」が検出可能



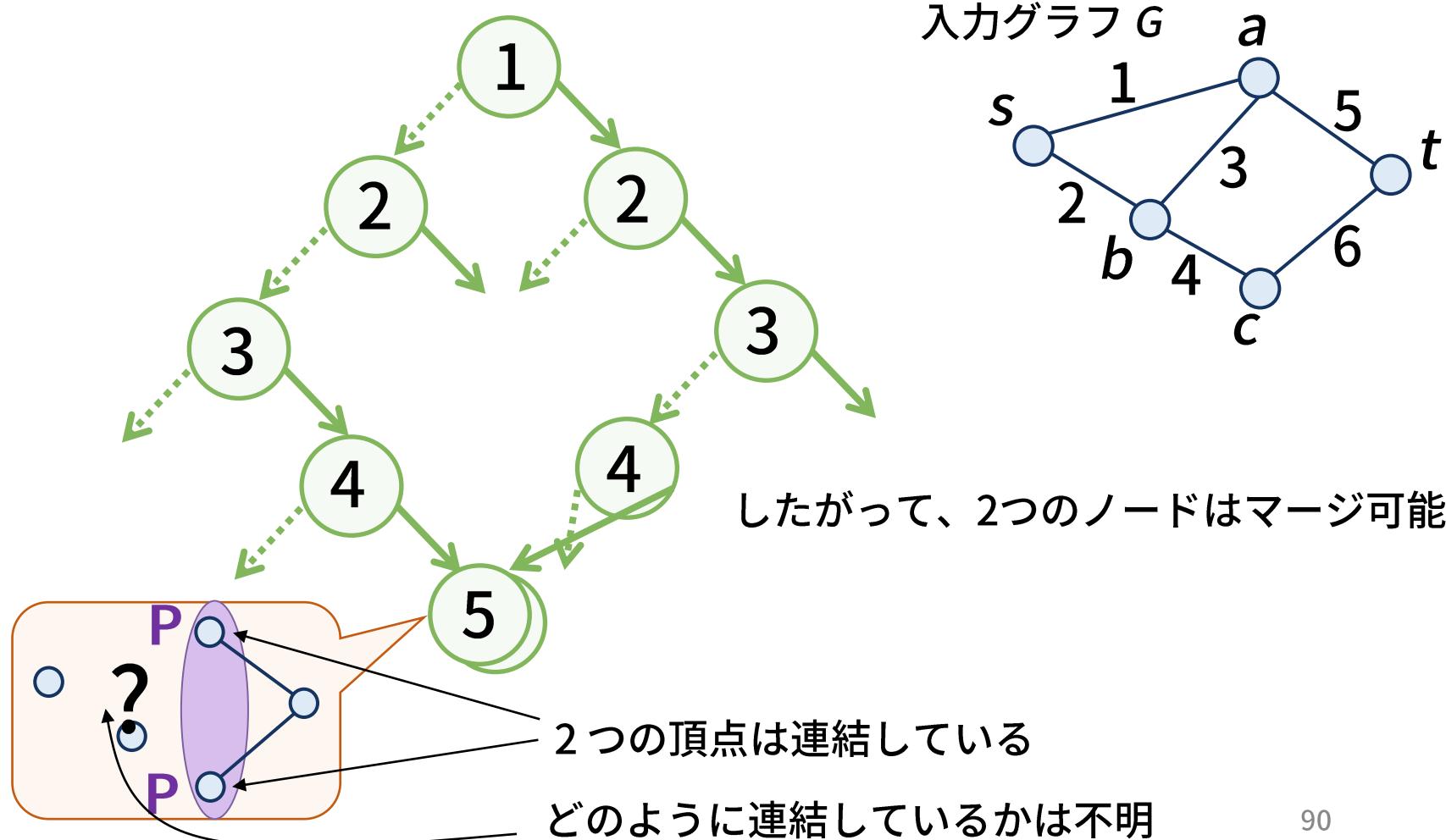
# 全域木集合のZDD構築

- 連結成分IDはフロンティア上の頂点のみ記憶すれば十分（であることが証明できる）



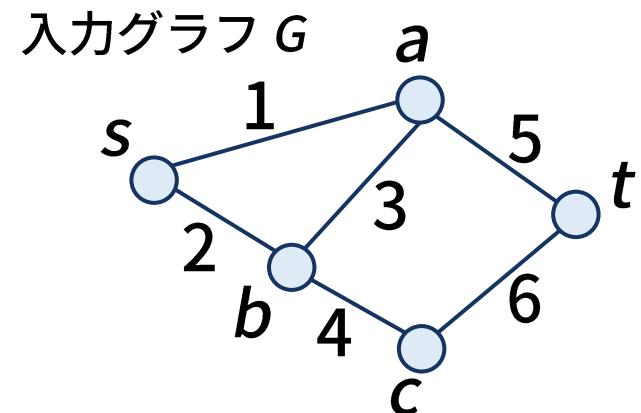
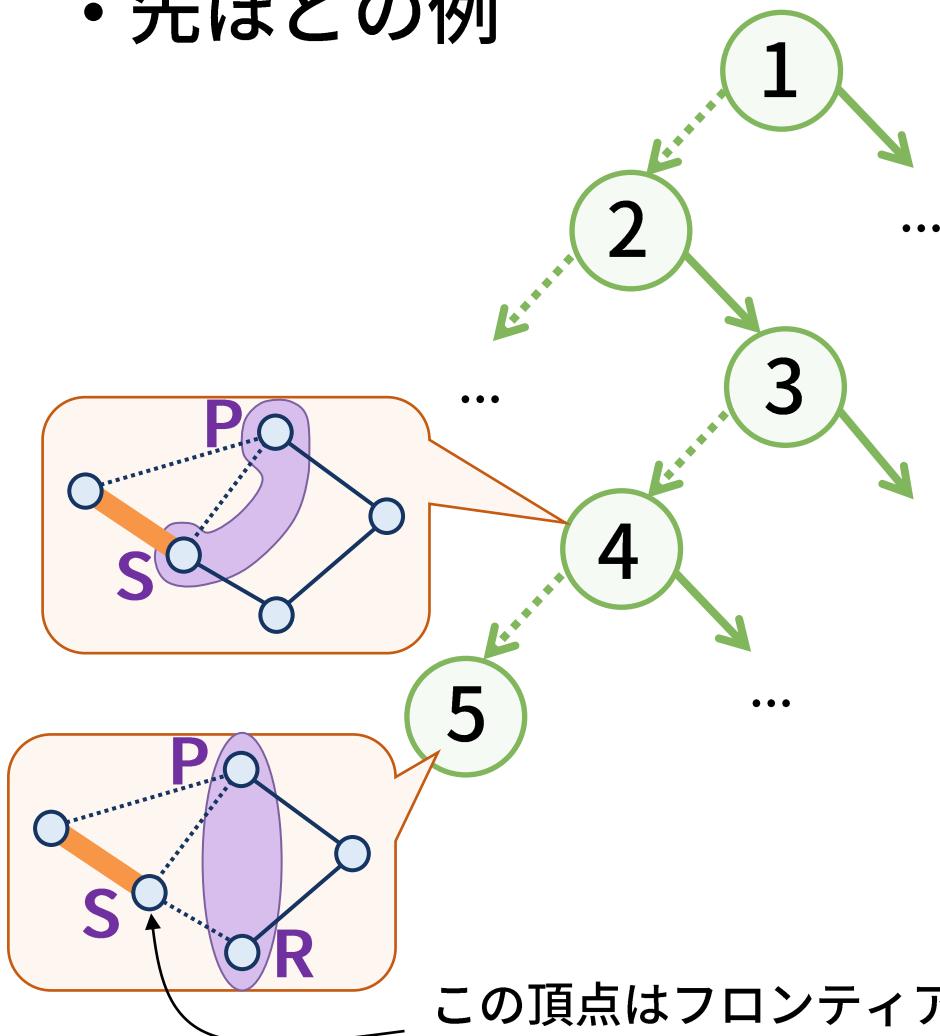
# 全域木集合のZDD構築

- 連結成分IDはフロンティア上の頂点のみ記憶すれば十分（であることが証明できる）



# 全域木集合のZDD構築

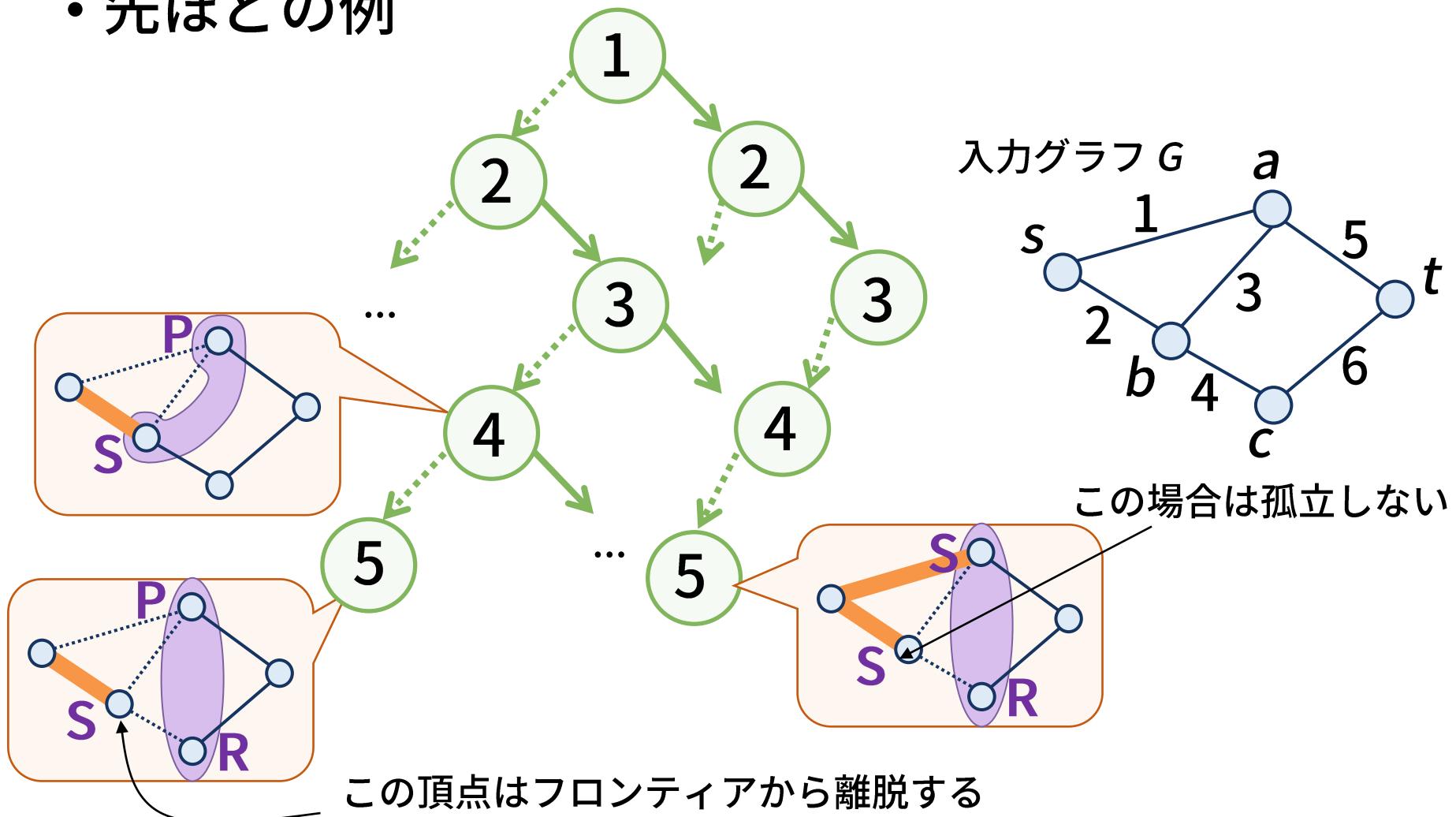
- 先ほどの例



この頂点はフロンティアから離脱する  
(これを「頂点はフロンティアから去る」と言う)。  
このとき、フロンティア上に他に  $S$  が無ければ、  
連結成分  $S$  は孤立する → 枝刈する

# 全域木集合のZDD構築

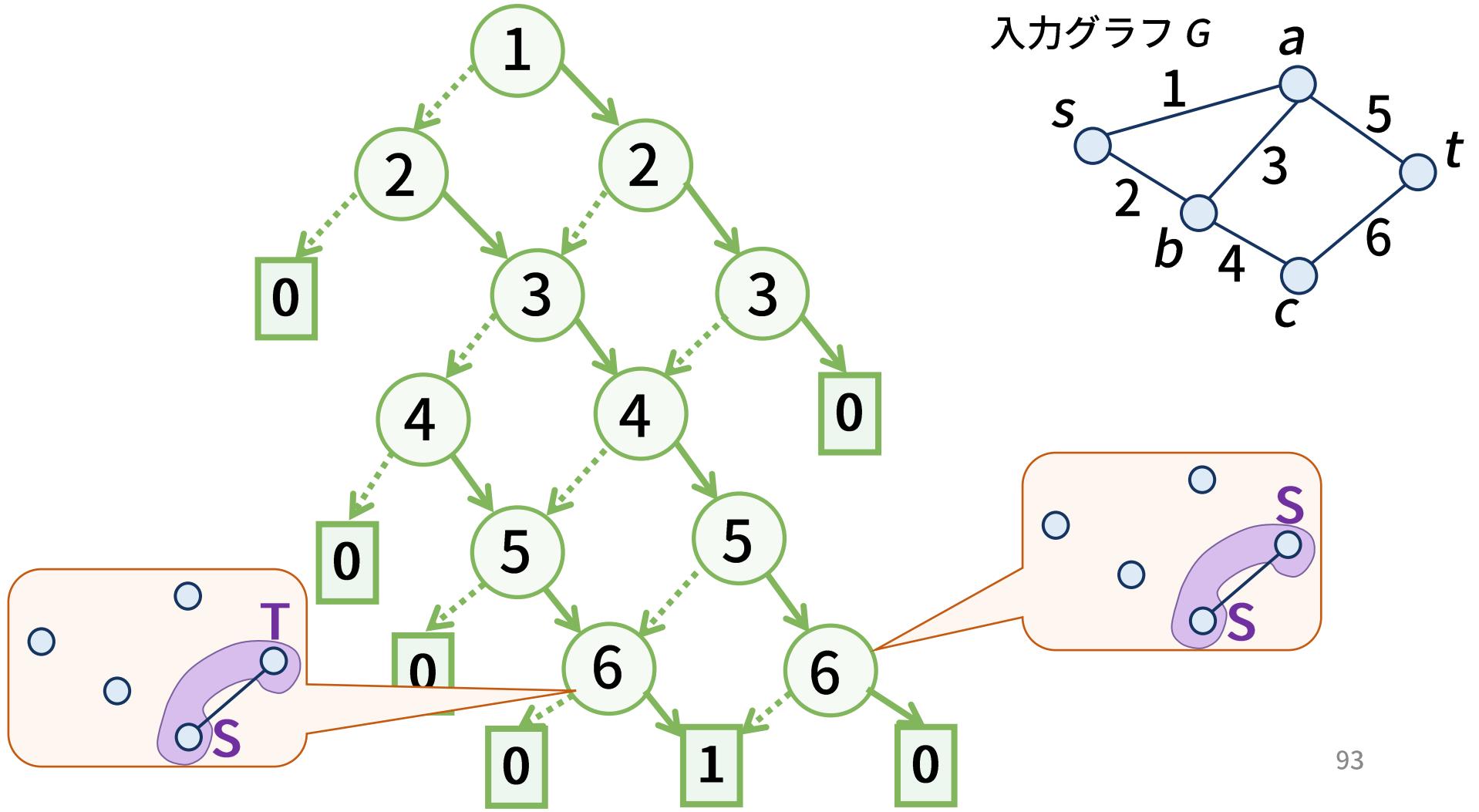
- 先ほどの例



この頂点はフロンティアから離脱する  
(これを「頂点はフロンティアから去る」と言う)。  
このとき、フロンティア上に他に  $S$  が無ければ、  
連結成分  $S$  は孤立する → 枝刈りする

# 全域木集合のZDD構築

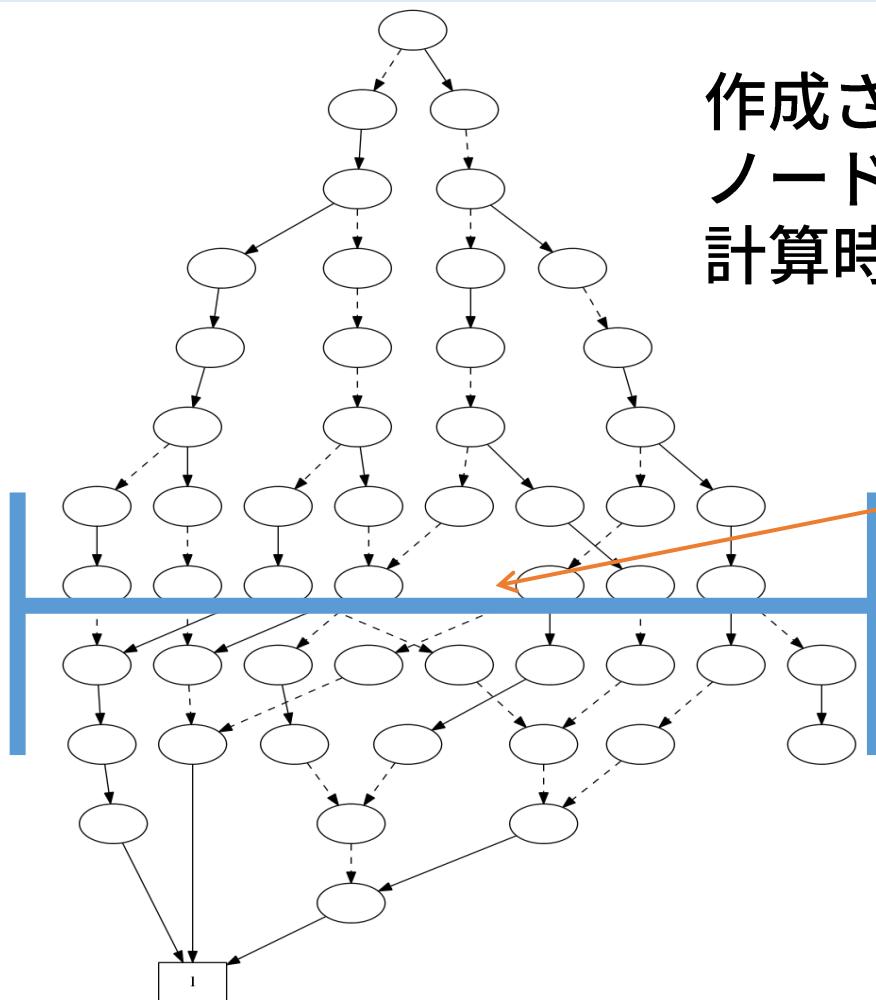
- 枝刈が行われずに最後まで到達すると、  
1-終端に接続



# $s$ - $t$ パス集合ZDD構築再考

- 全域木における連結性の保証の手法を、  
 $s$ - $t$  パスの場合にも適用することで、  
 $s$ - $t$  パスの連結性も保証できる
  - ただし、次数が 0 の頂点は連結でなくてもよい  
(これを扱う方法は省略)

# アルゴリズムの計算時間



作成されたZDDの  
ノード数に比例する  
計算時間がかかる

作成されるZDDの幅の上界値は

平面グラフ

$$C_{s+2} - 2C_{s+1}$$

$$C_k \approx 4^k / k^{3/2}$$

Catalan数

$$B_k + 2B_{k-1}$$

$$B_k \approx 2 \times 3^k$$

Ballot 数

$k \times k$  グリッドグラフ

$$2^{0.48n \log n + O(n)}$$

一般グラフ

K. Sekine, H. Imai, and S. Tani, "Computing the Tutte polynomial of a graph of moderate size," In Proc. of the 6th International Symposium on Algorithms and Computation (ISAAC), pp. 224--233, 1995.

高野圭司, "フロンティア法から生成される ZDD の幅解析,"  
数理解析研究所講究録, vol. 1849, pp. 77--82, 2013.

# ZDD構築可能な部分グラフ集合

- ・様々な部分グラフ集合を扱える

[Sekine+ 1995][Knuth 2008]

[Kawahara+ 2017]

で扱える部分グラフの種類

$s-t$ パス

サイクル

木、森、全域木

マッチング

クリーク

次数指定グラフ

...

グラフ分割 [Kawahara+ 2017]

禁止誘導部分グラフで特徴づけられる部分グラフの種類

弦グラフ

区間グラフ

真区間グラフ

...

[Kawahara+ 2019]

禁止マイナーで特徴づけられる部分グラフの種類

[Nakahata+ 2020]

平面グラフ

外平面グラフ

直並列グラフ

カクタスグラフ

...

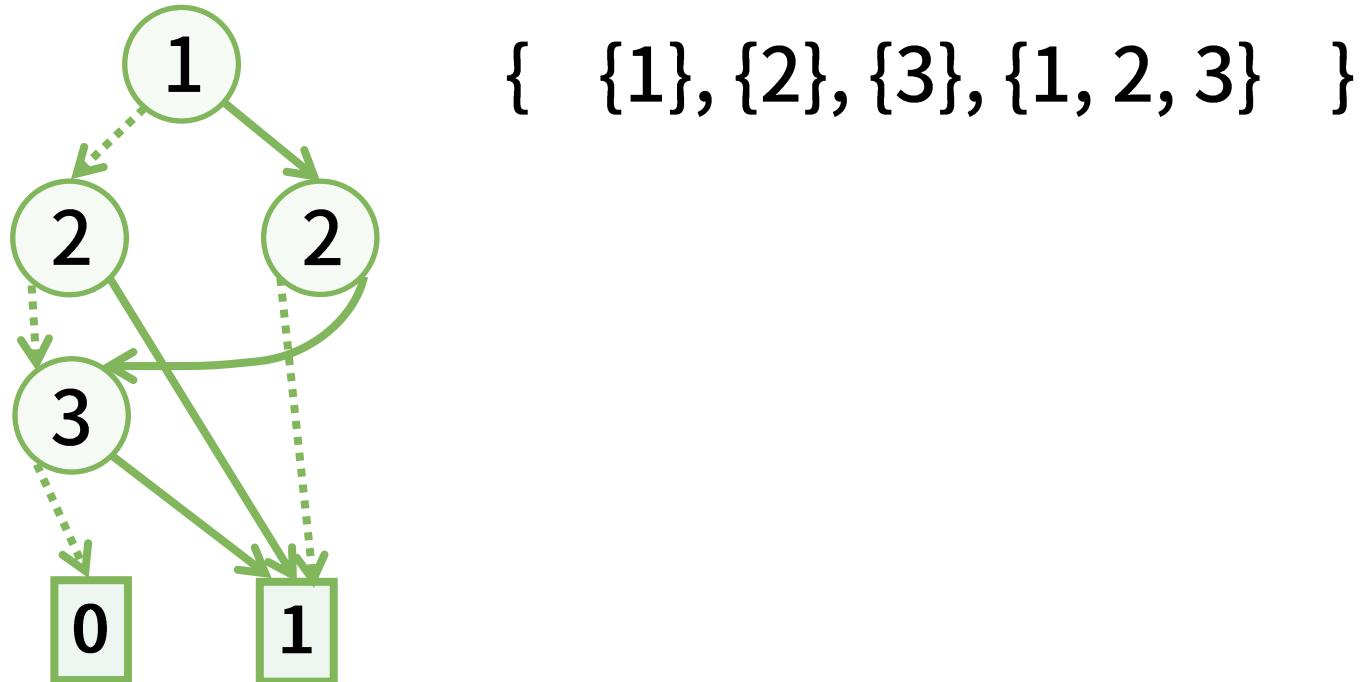
...

# 構築したZDDの活用法

- ・構築した ZDD に対して以下の操作が可能
  - ・集合族中の要素の数え上げ
  - ・一様ランダムサンプリング
  - ・線形重み最小・最大化
  - ・集合族から要素  $x$  を含む集合を抽出
  - ・集合族から大きさがちょうど  $k$  の集合を抽出

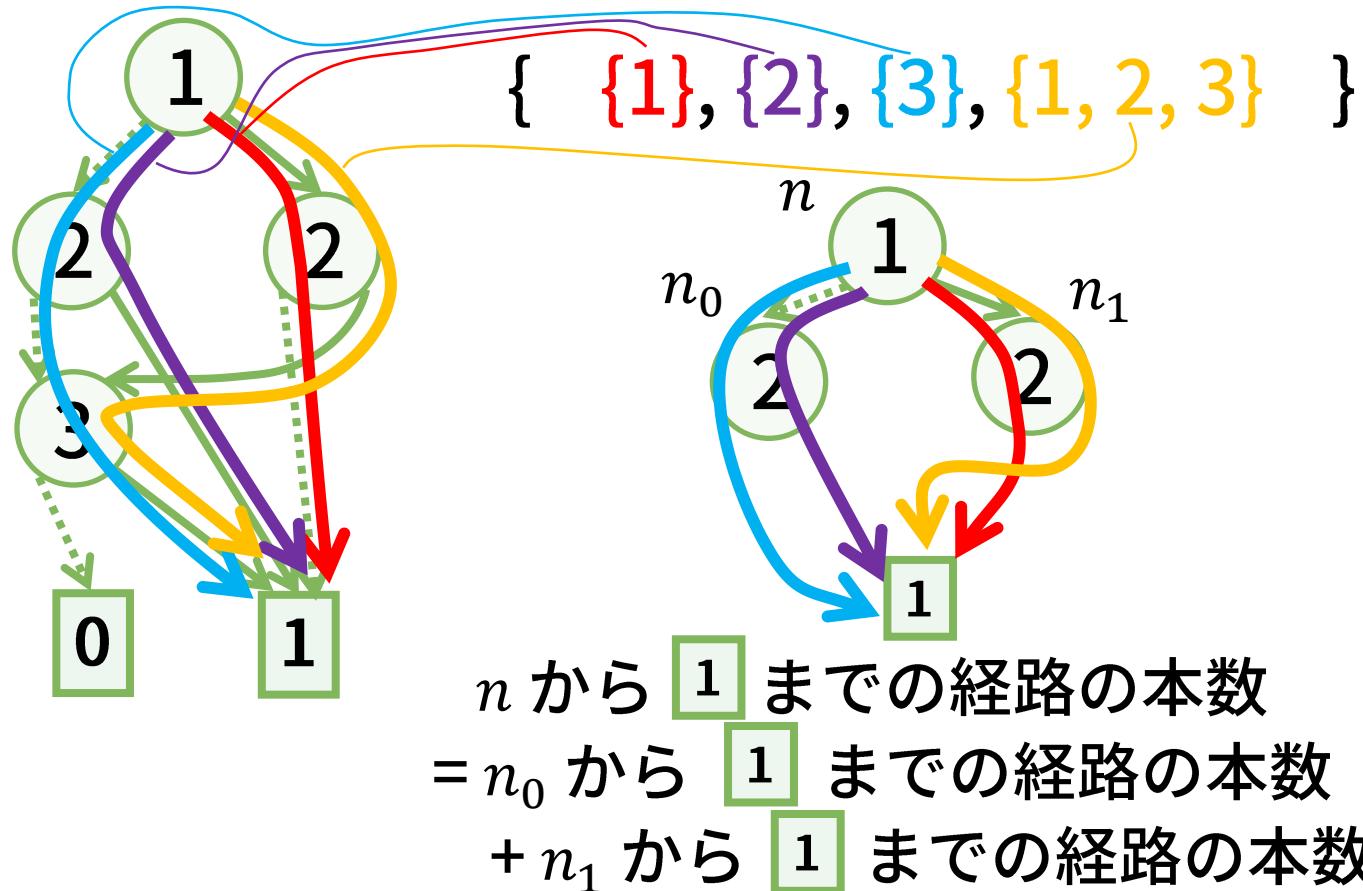
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から1-終端までの経路の本数が集合の個数



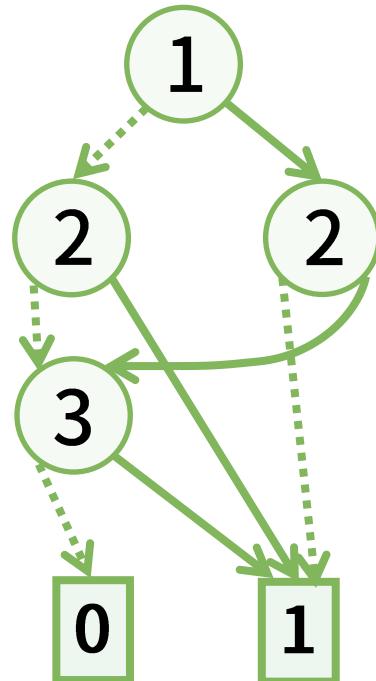
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数

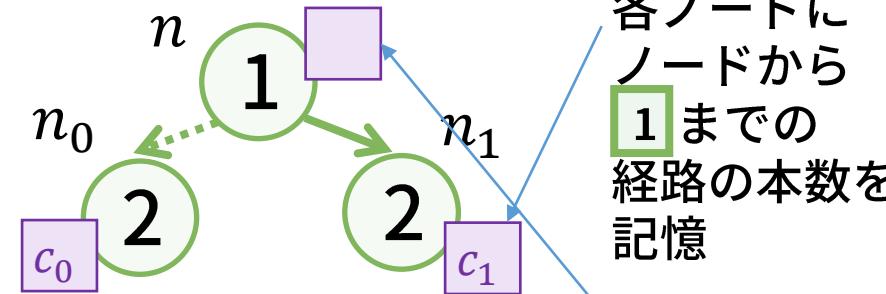


# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数



{ {1}, {2}, {3}, {1, 2, 3} }



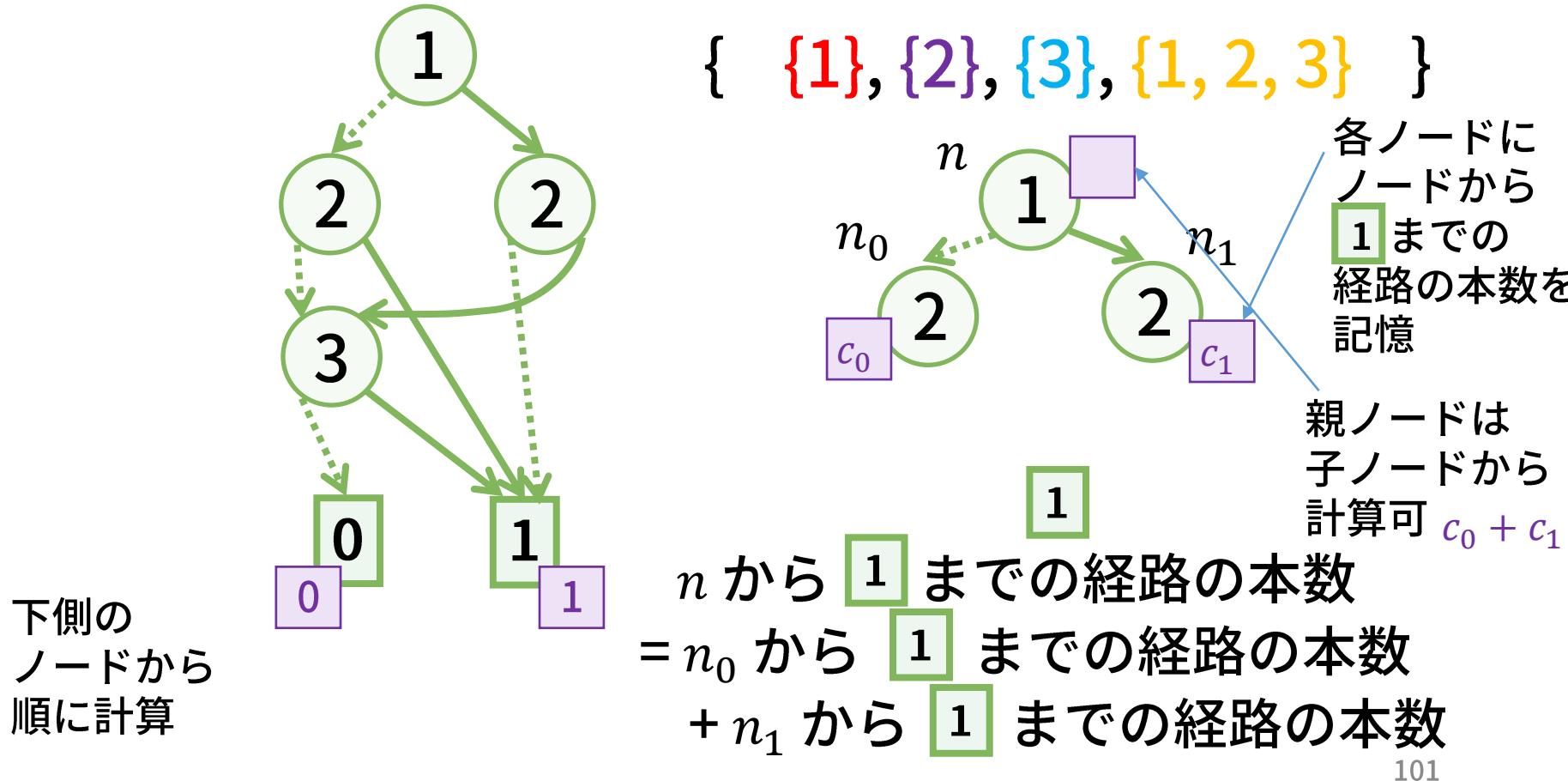
各ノードに  
ノードから  
1 までの  
経路の本数を  
記憶

親ノードは  
子ノードから  
計算可  $c_0 + c_1$

$$\begin{aligned} & n \text{ から } 1 \text{ までの経路の本数} \\ &= n_0 \text{ から } 1 \text{ までの経路の本数} \\ &\quad + n_1 \text{ から } 1 \text{ までの経路の本数} \end{aligned}$$

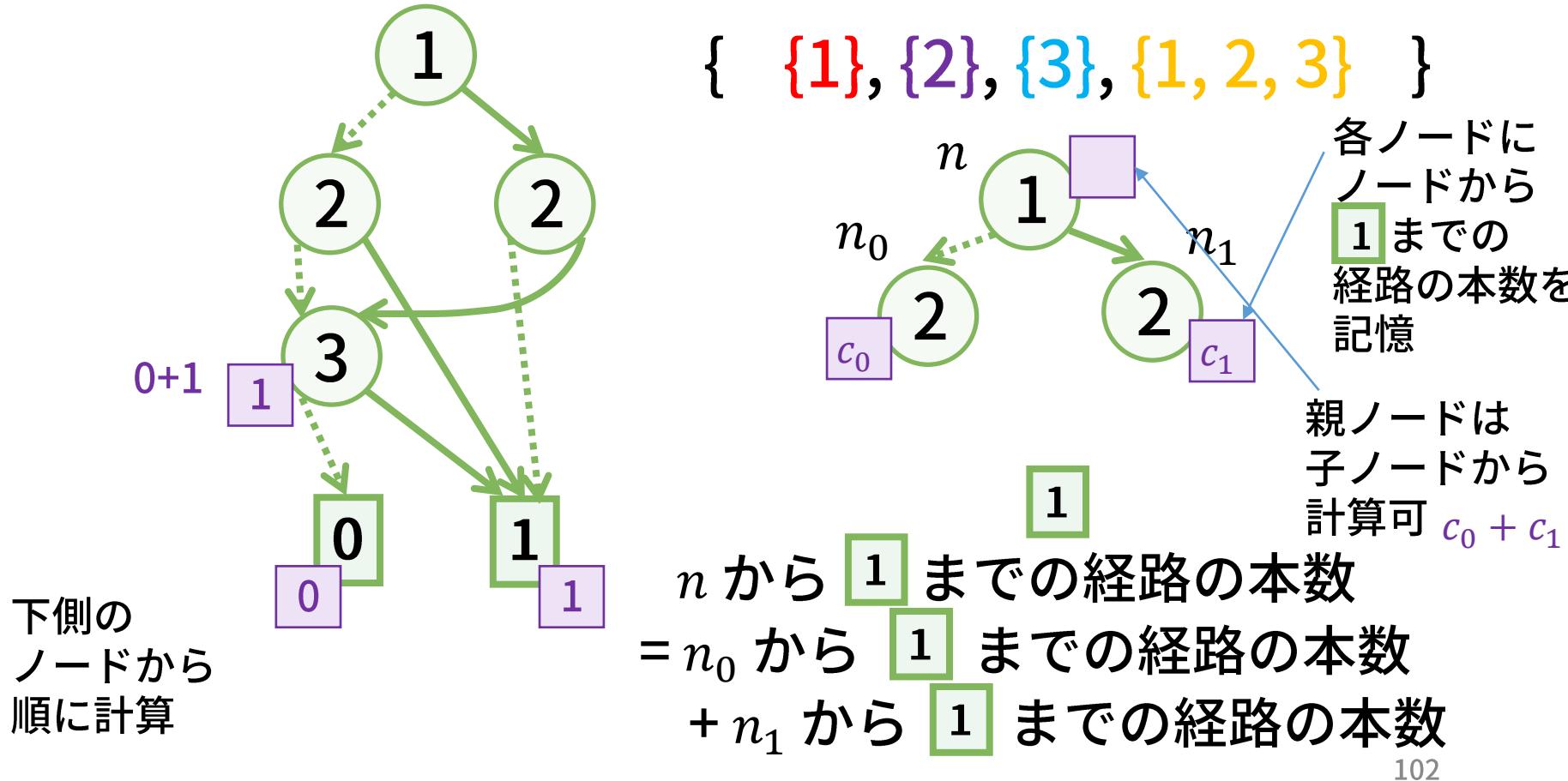
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数



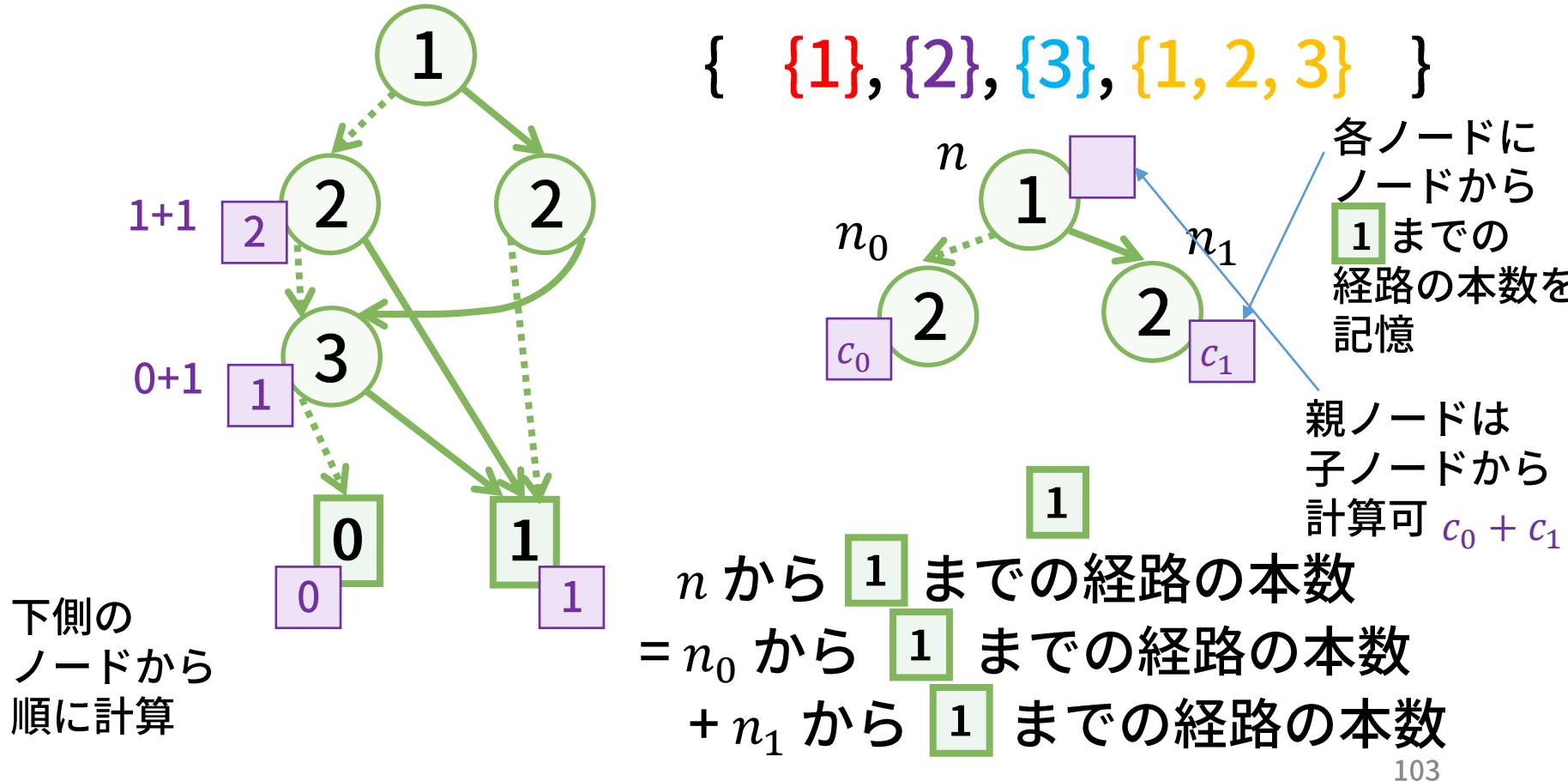
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数



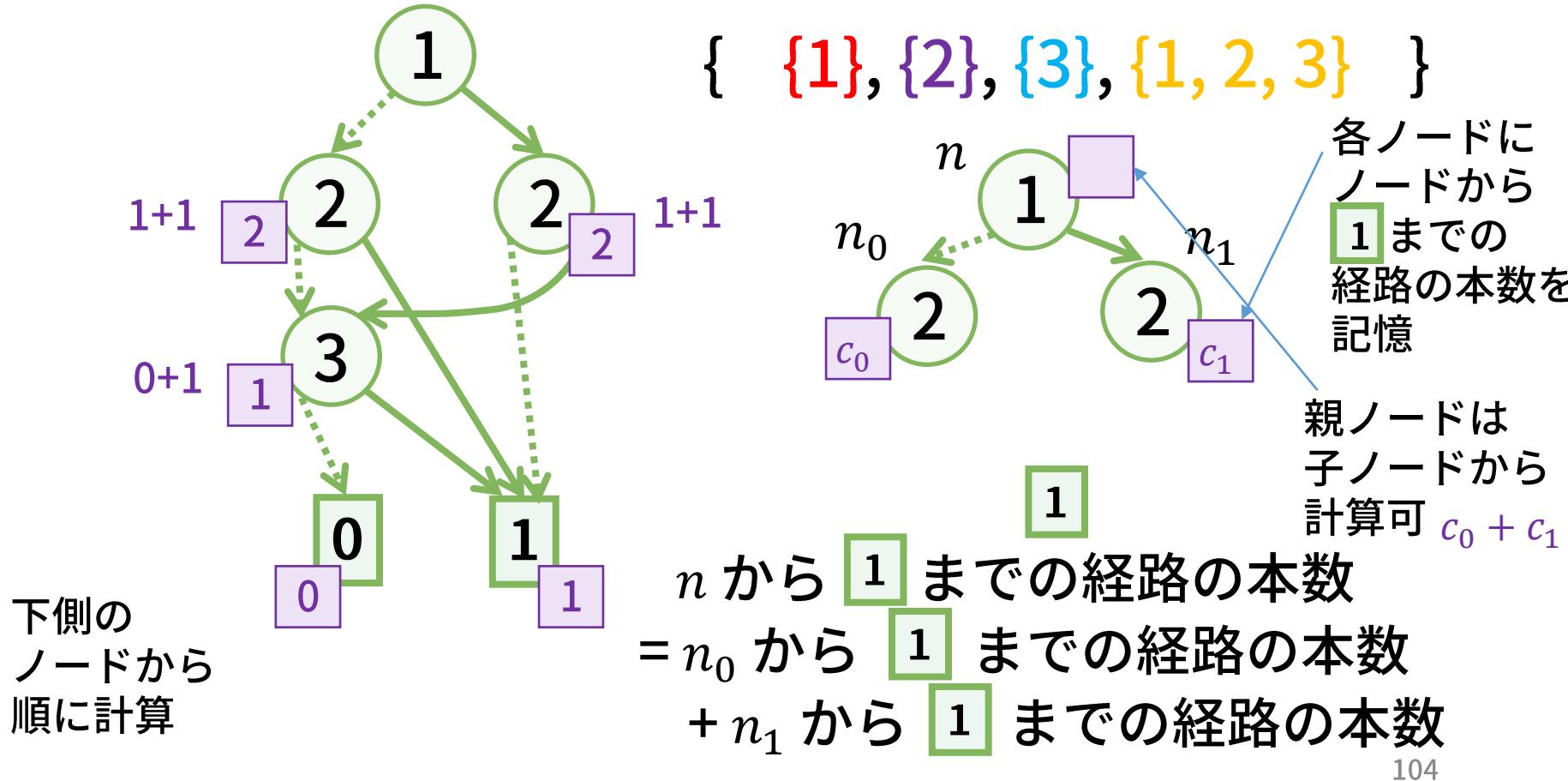
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数



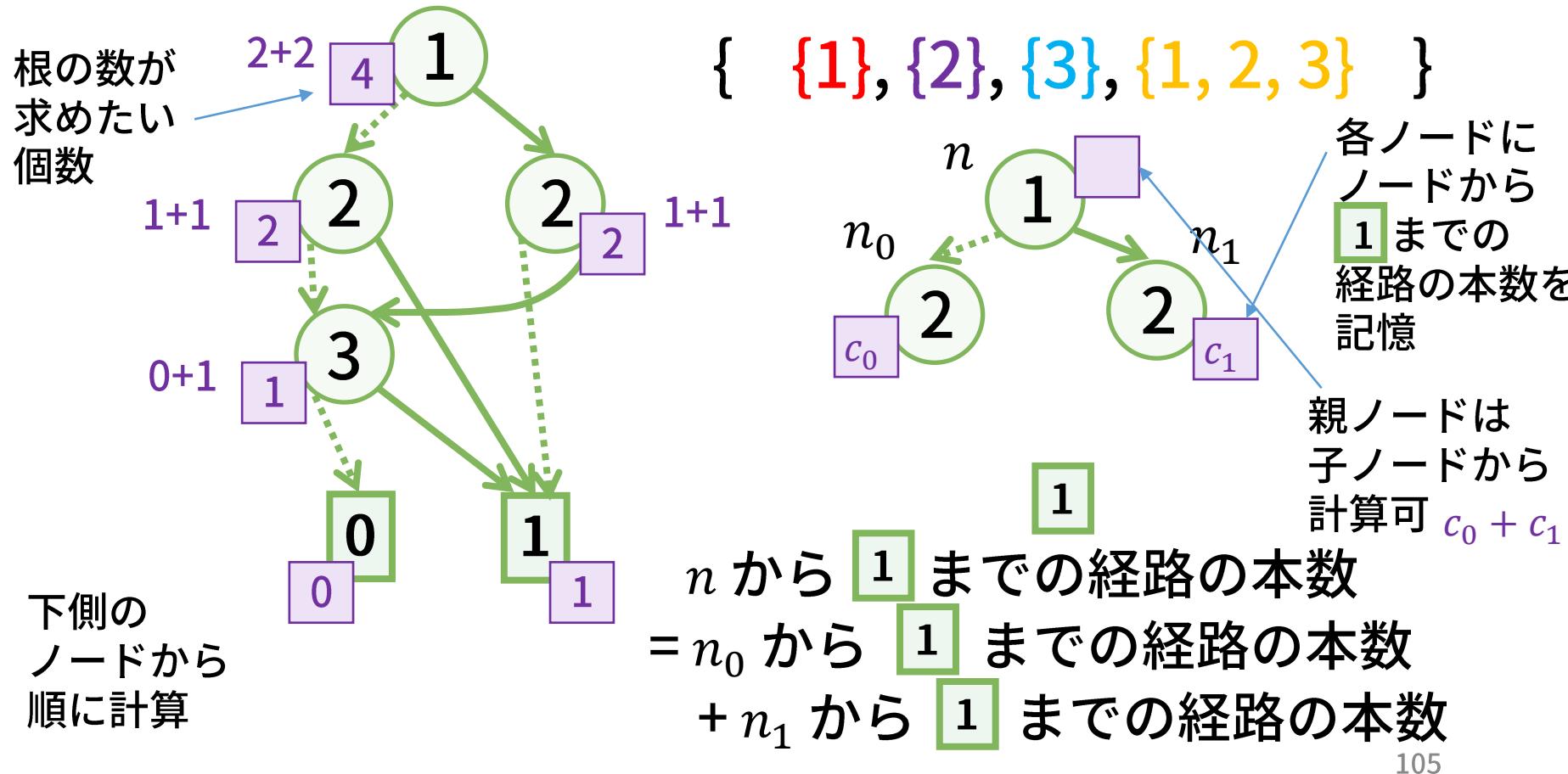
# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数



# 構築したZDDの活用: 要素の数え上げ

- ZDD が構築できれば、対応する集合族内の集合の個数が計算できる
- ZDD の根から 1-終端までの経路の本数が集合の個数

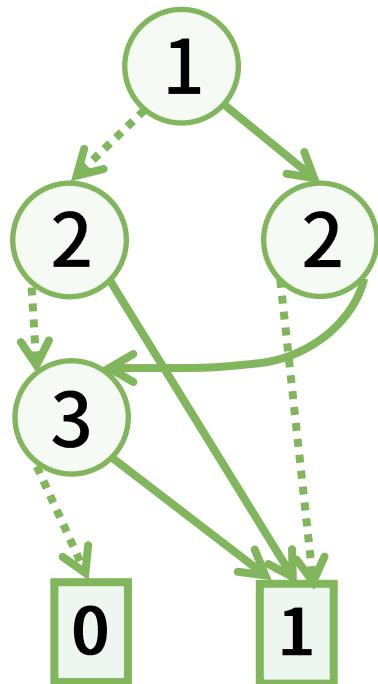


# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

4個の要素（集合）から  
等確率（1/4）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }



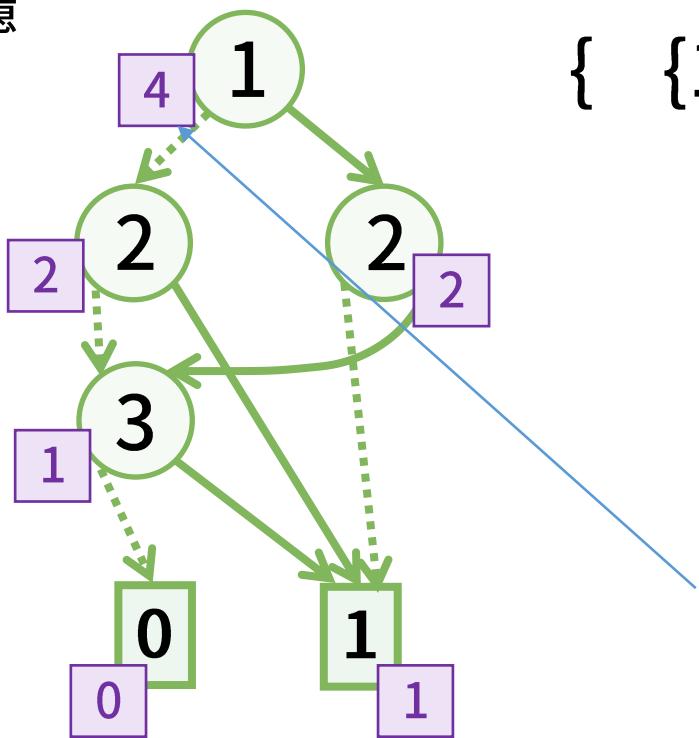
# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶

4個の要素（集合）から  
等確率（1/4）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }

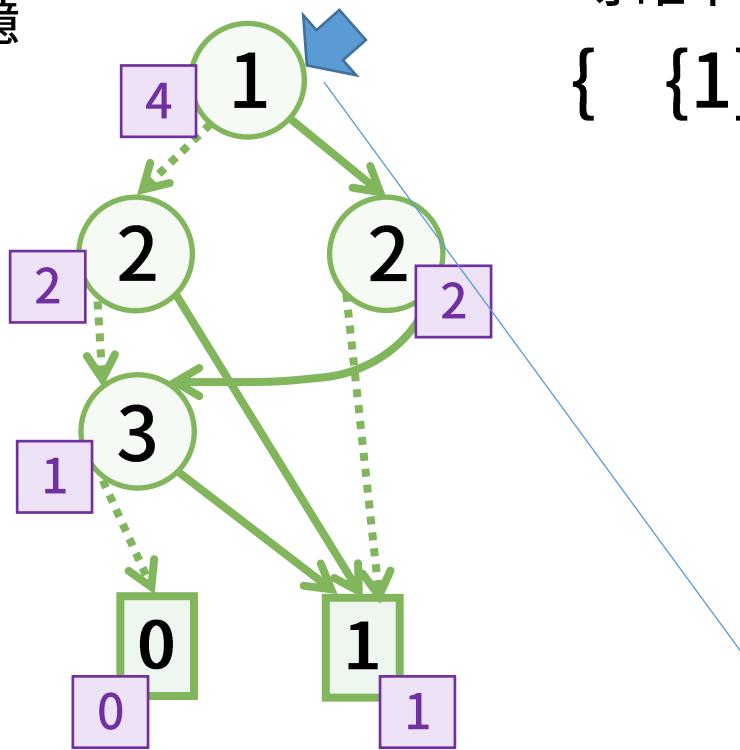


4本ある経路から、等確率で  
1本選ぶ操作を行う

# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶



4個の要素（集合）から  
等確率（1/4）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }

根ノードからスタート  
確率 2/4 で 0 枝側  
確率 2/4 で 1 枝側に進む

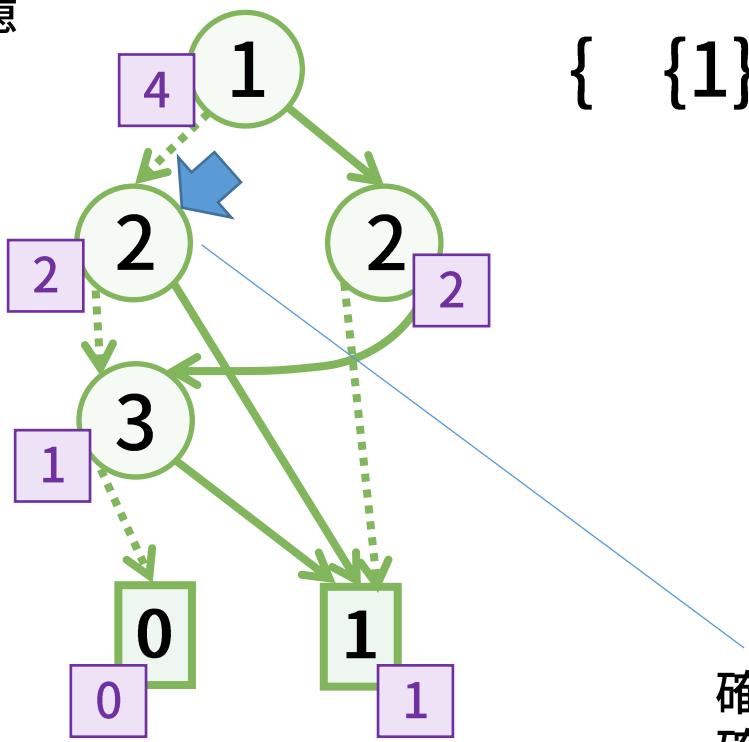
# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶

4個の要素（集合）から  
等確率（ $1/4$ ）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }



確率  $1/2$  で 0 枝側  
確率  $1/2$  で 1 枝側に進む

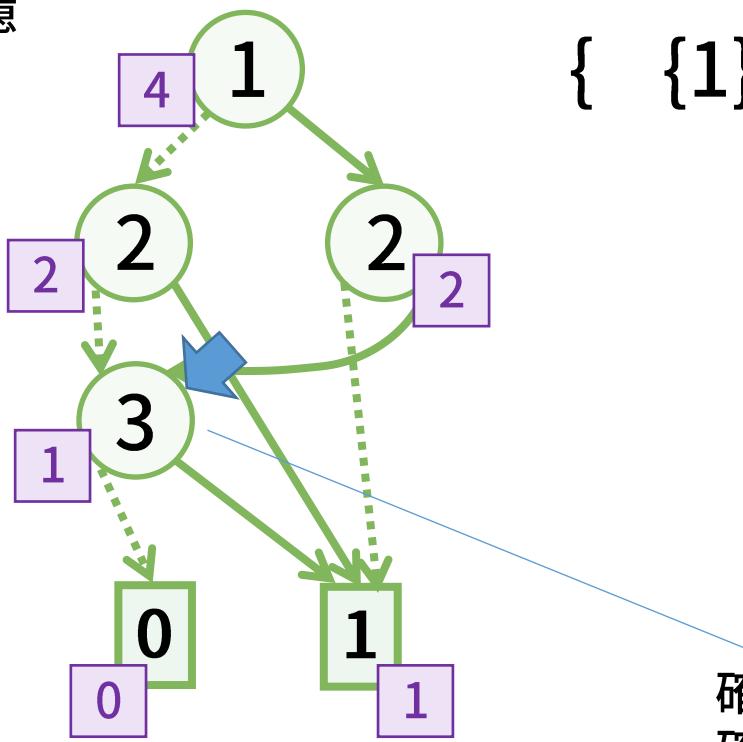
# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶

4個の要素（集合）から  
等確率（1/4）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }



確率 0/1 で 0 枝側  
確率 1/1 で 1 枝側に進む

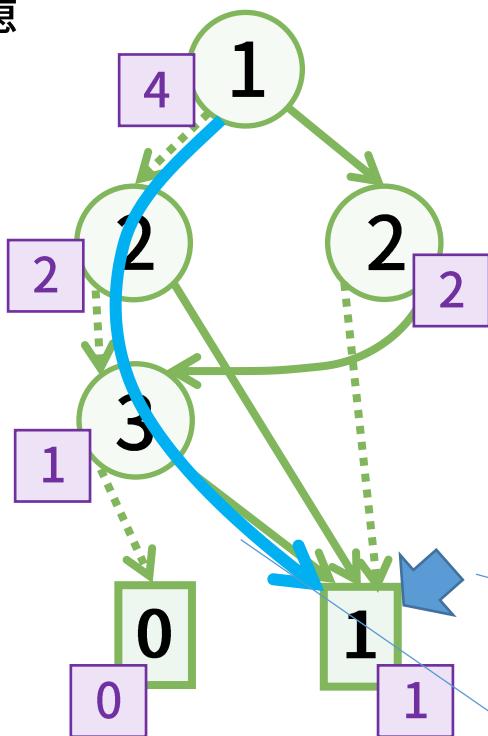
# 構築したZDDの活用: ランダムサンプリング

- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶

4個の要素（集合）から  
等確率（1/4）でランダムに選択

{ {1}, {2}, {3}, {1, 2, 3} }



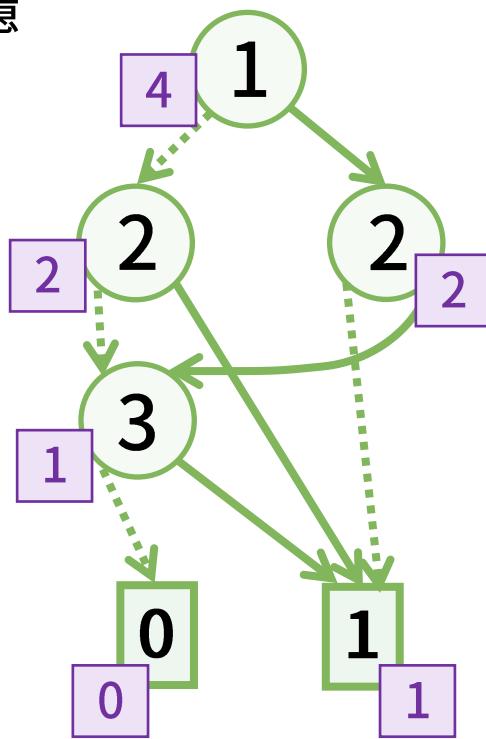
必ず1-終端に到着  
通った経路に対応する集合が  
選ばれた集合

{3}

# 構築したZDDの活用: ランダムサンプリング

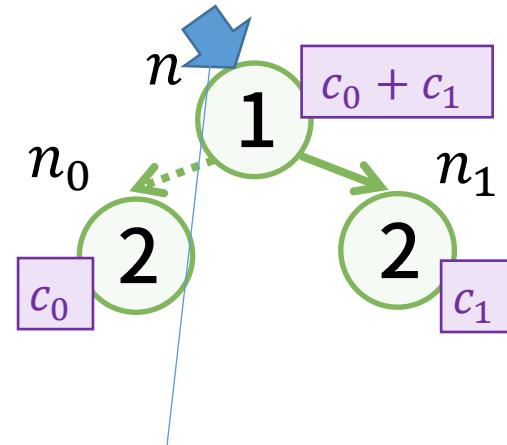
- 集合族内の各集合を一様ランダムサンプリング

要素の数え上げアルゴリズムで  
各ノードから1-終端までの  
経路数を記憶



4個の要素（集合）から  
等確率（1/4）でランダムに選択

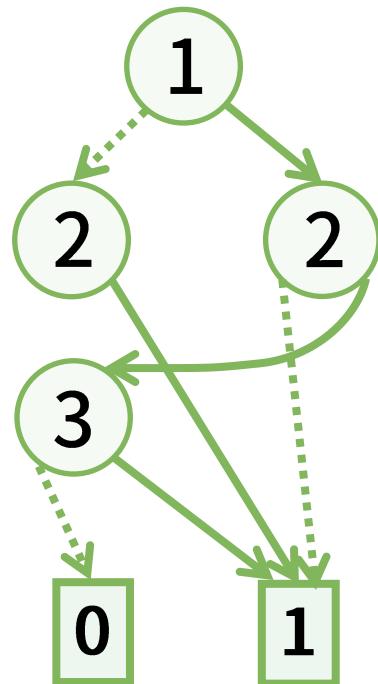
{ {1}, {2}, {3}, {1, 2, 3} }



一般に  
確率  $c_0/(c_0 + c_1)$  で0枝側  
確率  $c_1/(c_0 + c_1)$  で1枝側  
を選ぶ操作を繰り返す

# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい

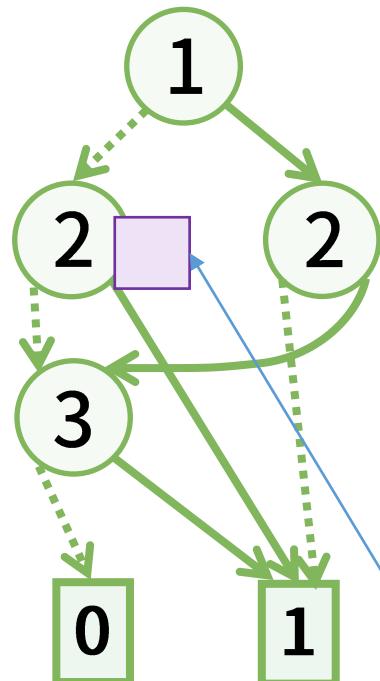


重み  
 $w_1: 2$   
 $w_2: 4$   
 $w_3: -3$

{	{1}, 2	
{2}, 4	最大	
{3}, -3	最小	
{1, 2, 3}	}	
$2 + 4 - 3 = 3$		

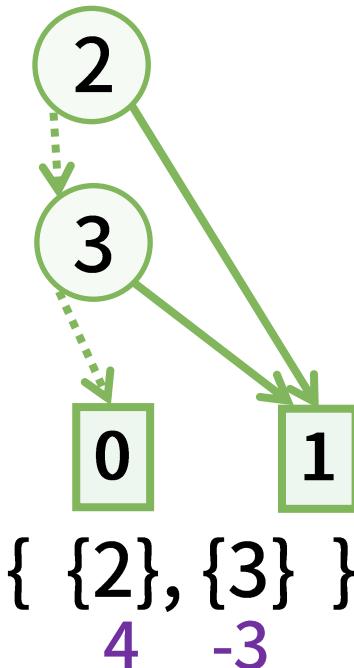
# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



重み  
 $w_1: 2$   
 $w_2: 4$   
 $w_3: -3$

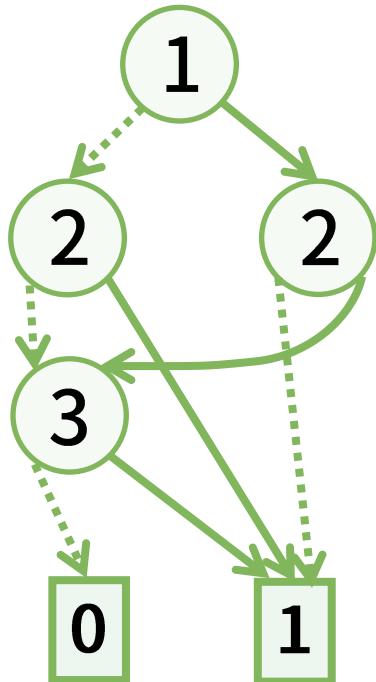
各 ZDD ノードには  
そのノードを根とする  
ZDD が表す集合族の  
最大重みを記憶する



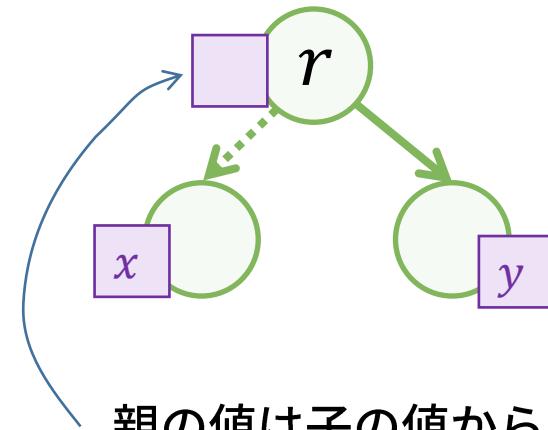
最大重みは 4 なので 4 を記憶

# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



重み  
 $w_1: 2$   
 $w_2: 4$   
 $w_3: -3$



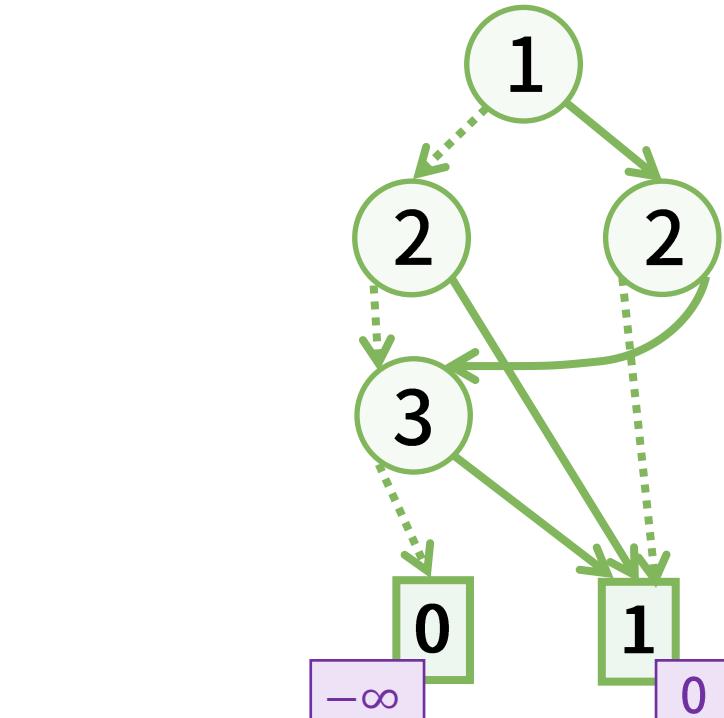
親の値は子の値から  
計算できる  
最大値をとる集合は  
0枝側から選ぶか、  
1枝側から選び、要素  $r$  を  
加えたもののどちらか

$$\max \{ x, y + w_r \}$$

$r$  の重み

# 構築したZDDの活用: 線形重み最適化

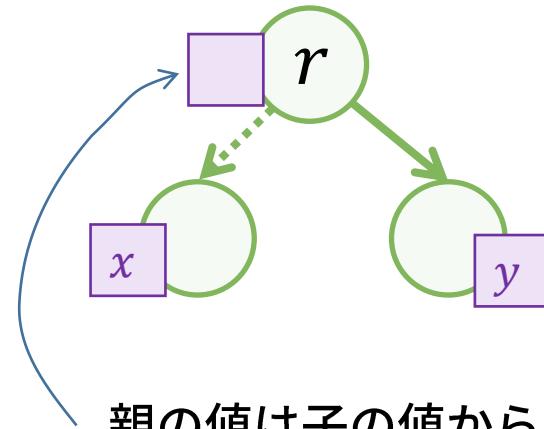
- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



$-\infty$  (決して選ばれない)  
にする

$\{\{\}\}$  の最大重みは 0

重み  
 $w_1: 2$   
 $w_2: 4$   
 $w_3: -3$



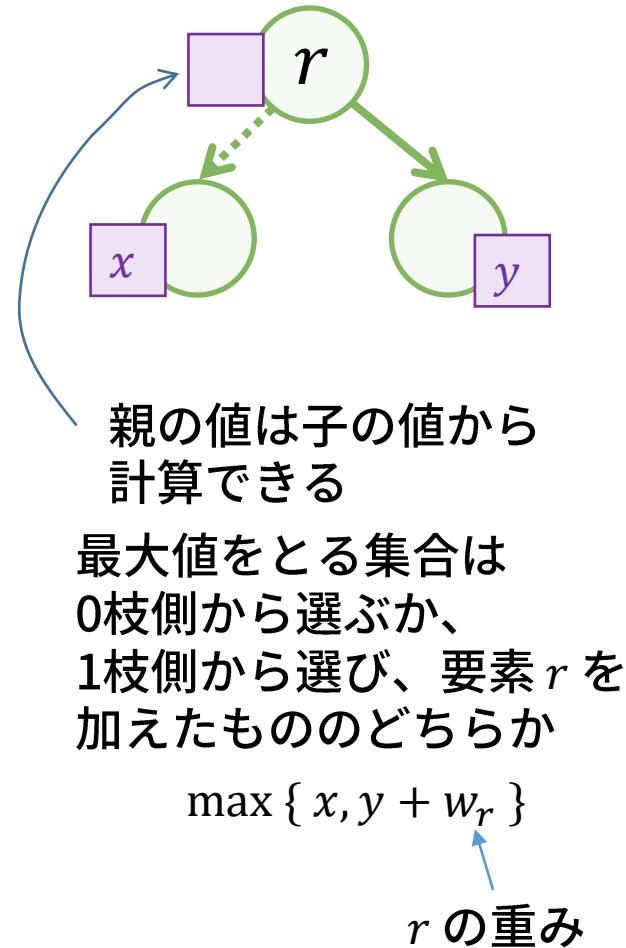
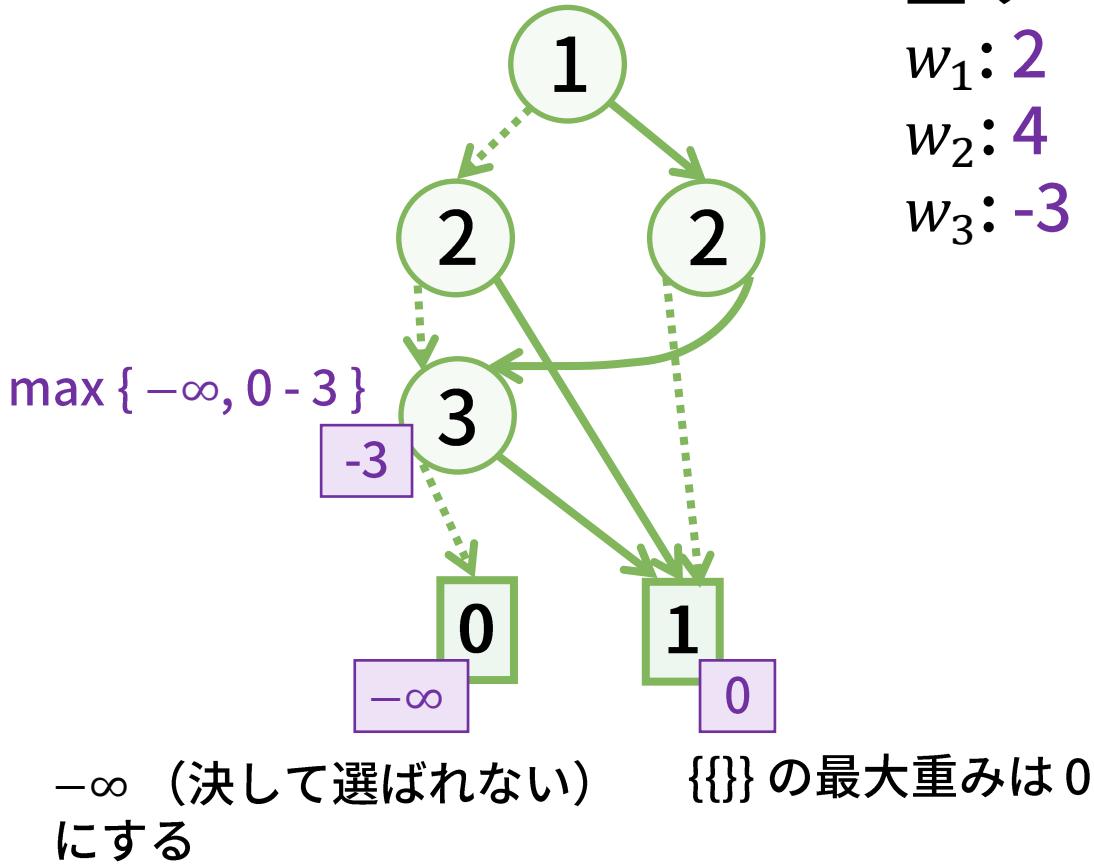
親の値は子の値から  
計算できる  
最大値をとる集合は  
0枝側から選ぶか、  
1枝側から選び、要素  $r$  を  
加えたもののどちらか

$$\max \{ x, y + w_r \}$$

$r$  の重み

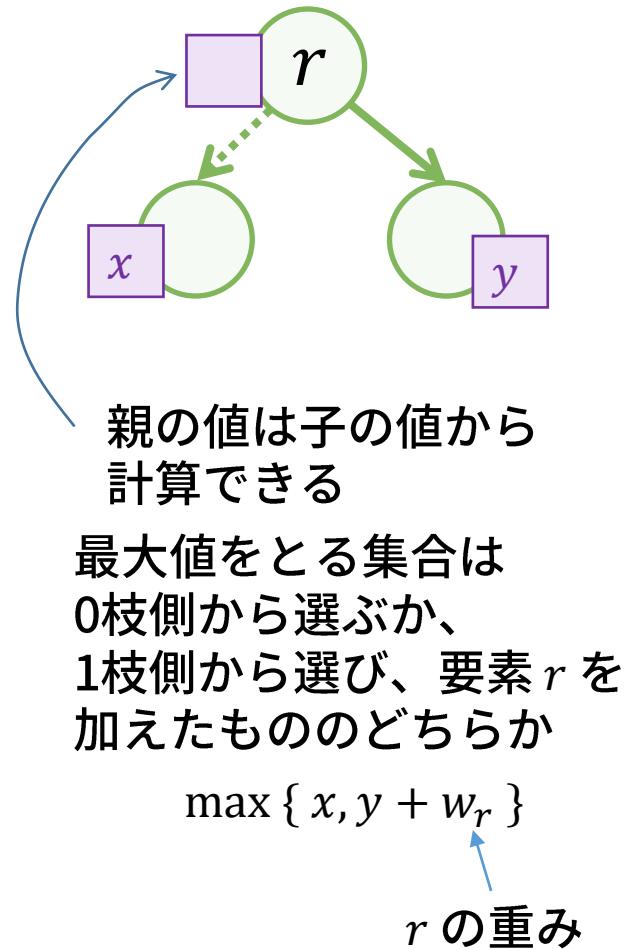
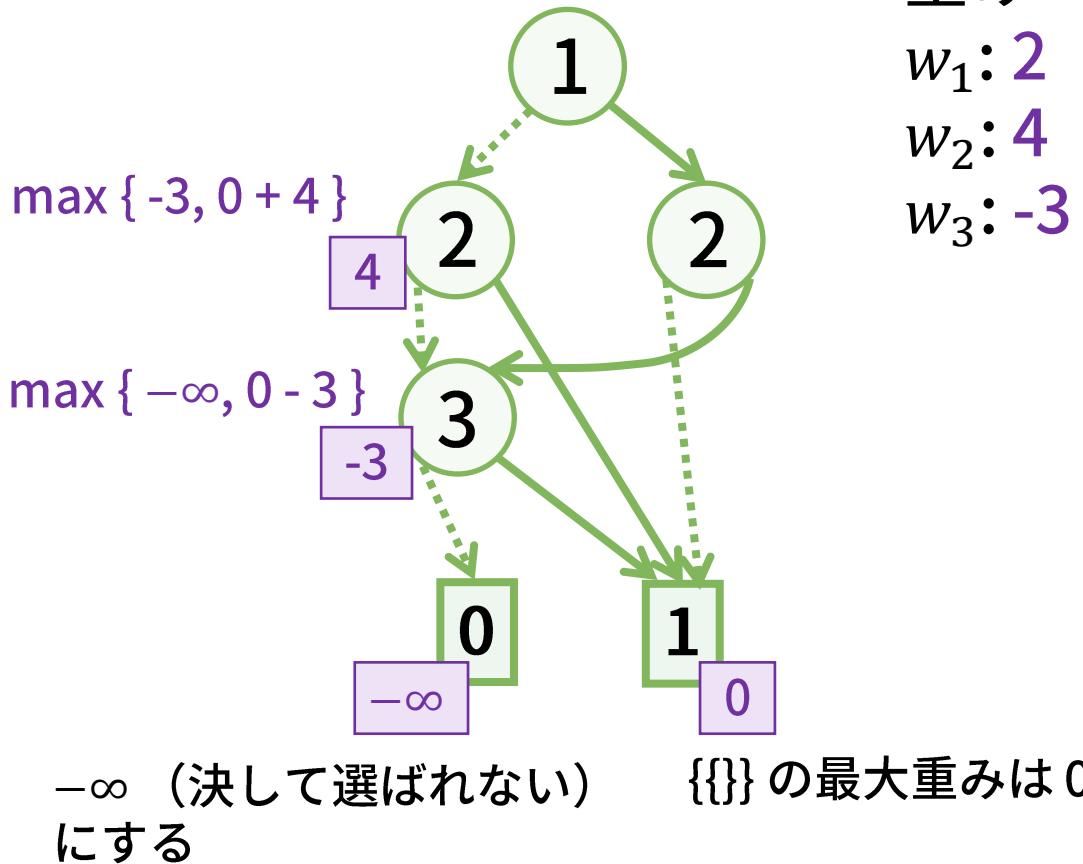
# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



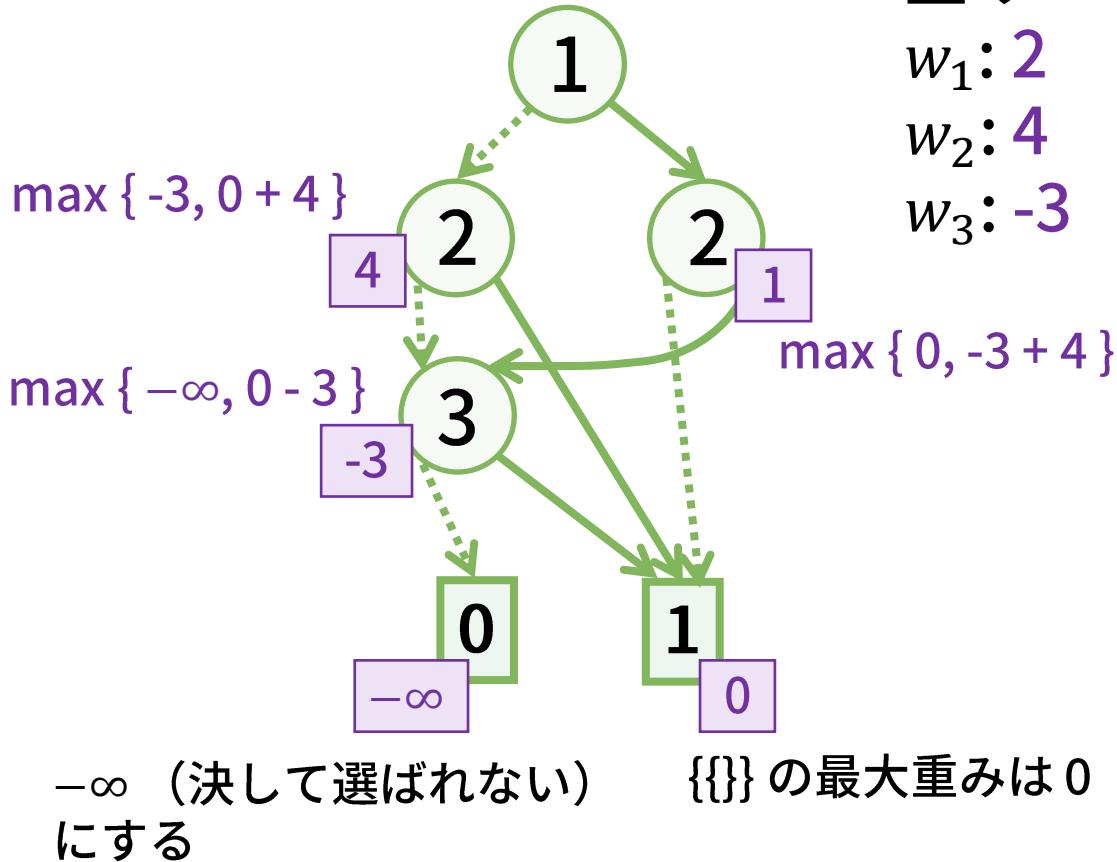
# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



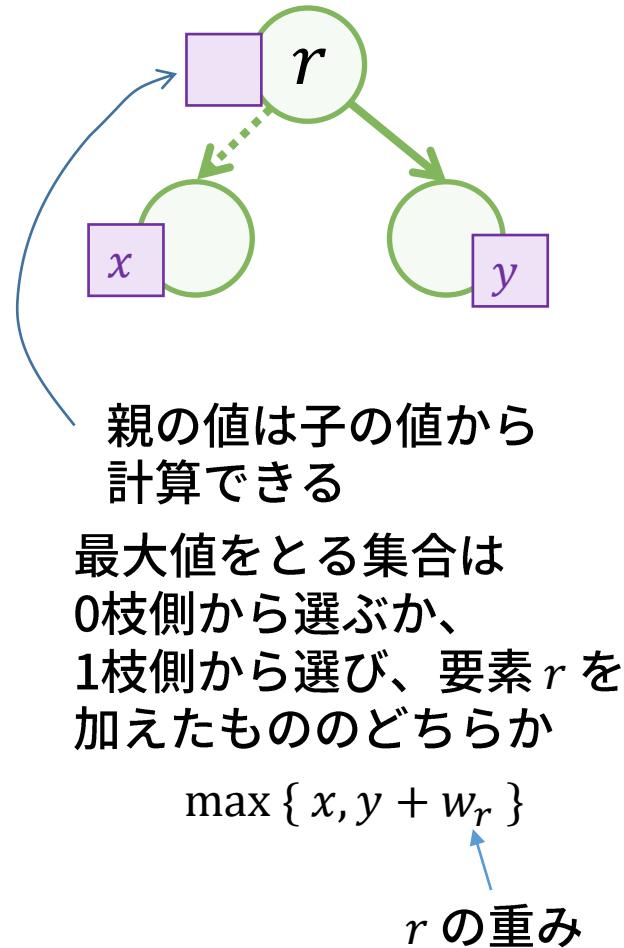
# 構築したZDDの活用: 線形重み最適化

- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい



重み  
 $w_1: 2$   
 $w_2: 4$   
 $w_3: -3$

$\{\{\}\}$  の最大重みは 0



# 構築したZDDの活用: 線形重み最適化

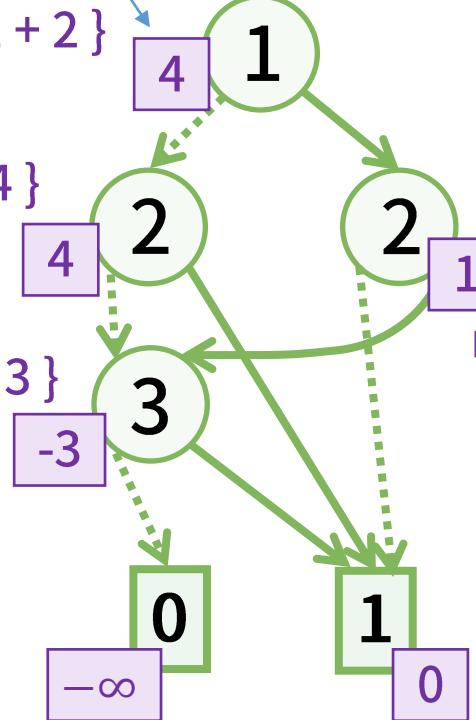
- 集合の各要素に重みを与えるとき、重みが最大・最小の集合を求めたい

根の値が最大値

$$\max \{ 4, 1 + 2 \}$$

$$\max \{ -3, 0 + 4 \}$$

$$\max \{ -\infty, 0 - 3 \}$$



最大値をとる集合を得るには  
バ ckトラックすればよい

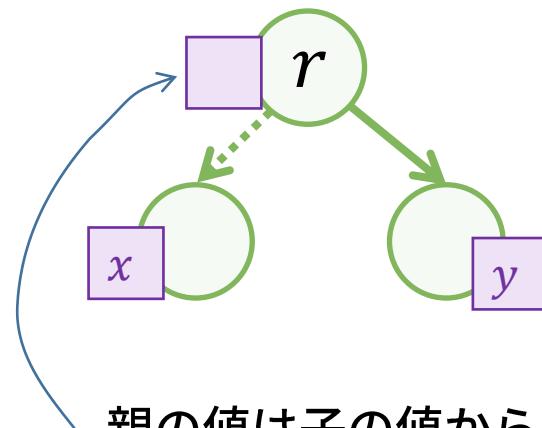
重み

$$w_1: 2$$

$$w_2: 4$$

$$w_3: -3$$

$$\max \{ 0, -3 + 4 \}$$



親の値は子の値から  
計算できる  
最大値をとる集合は  
0枝側から選ぶか、  
1枝側から選び、要素  $r$  を  
加えたもののどちらか

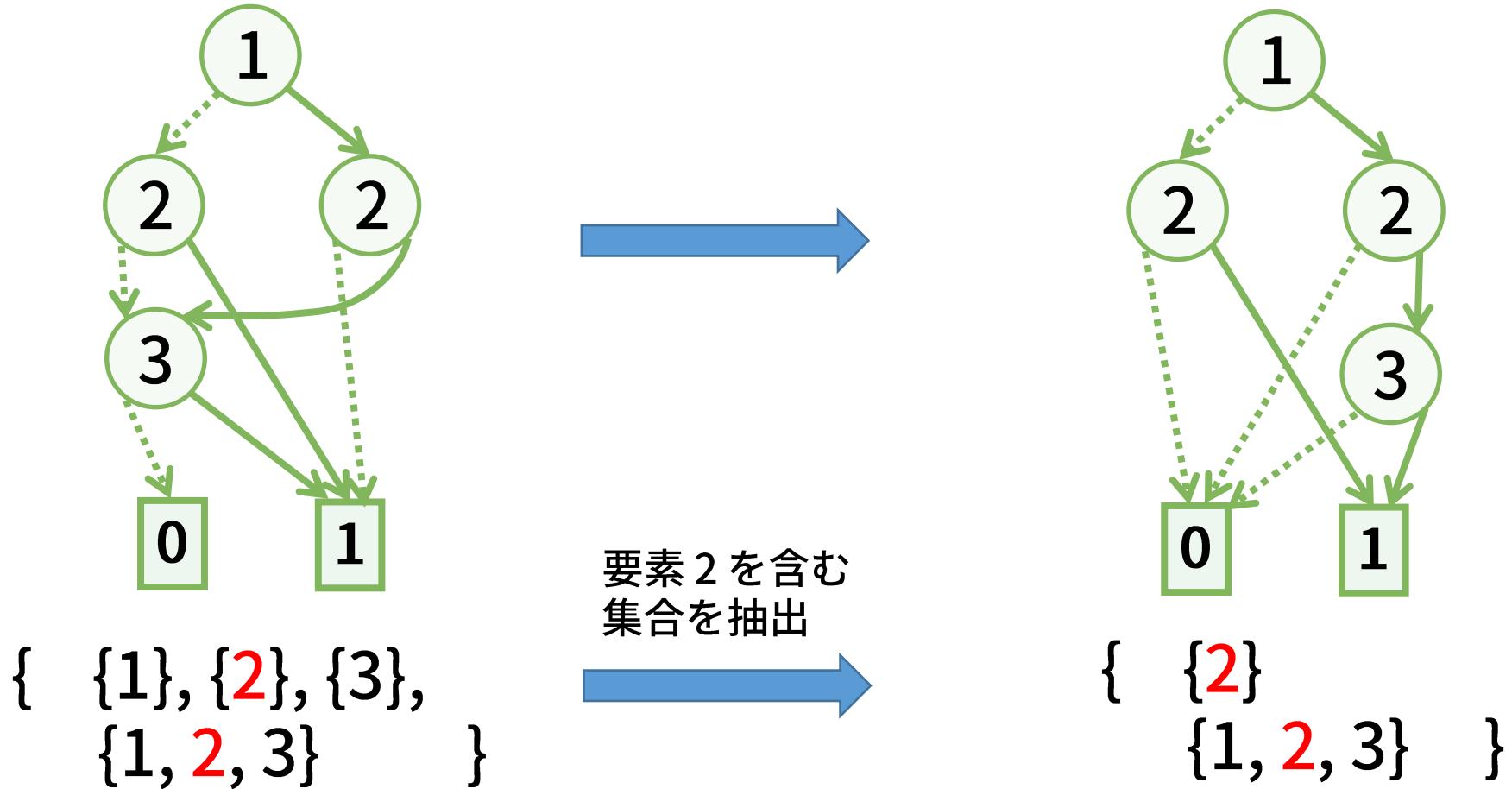
$$\max \{ x, y + w_r \}$$

$r$  の重み

最小値を求める場合も  
同様に可能

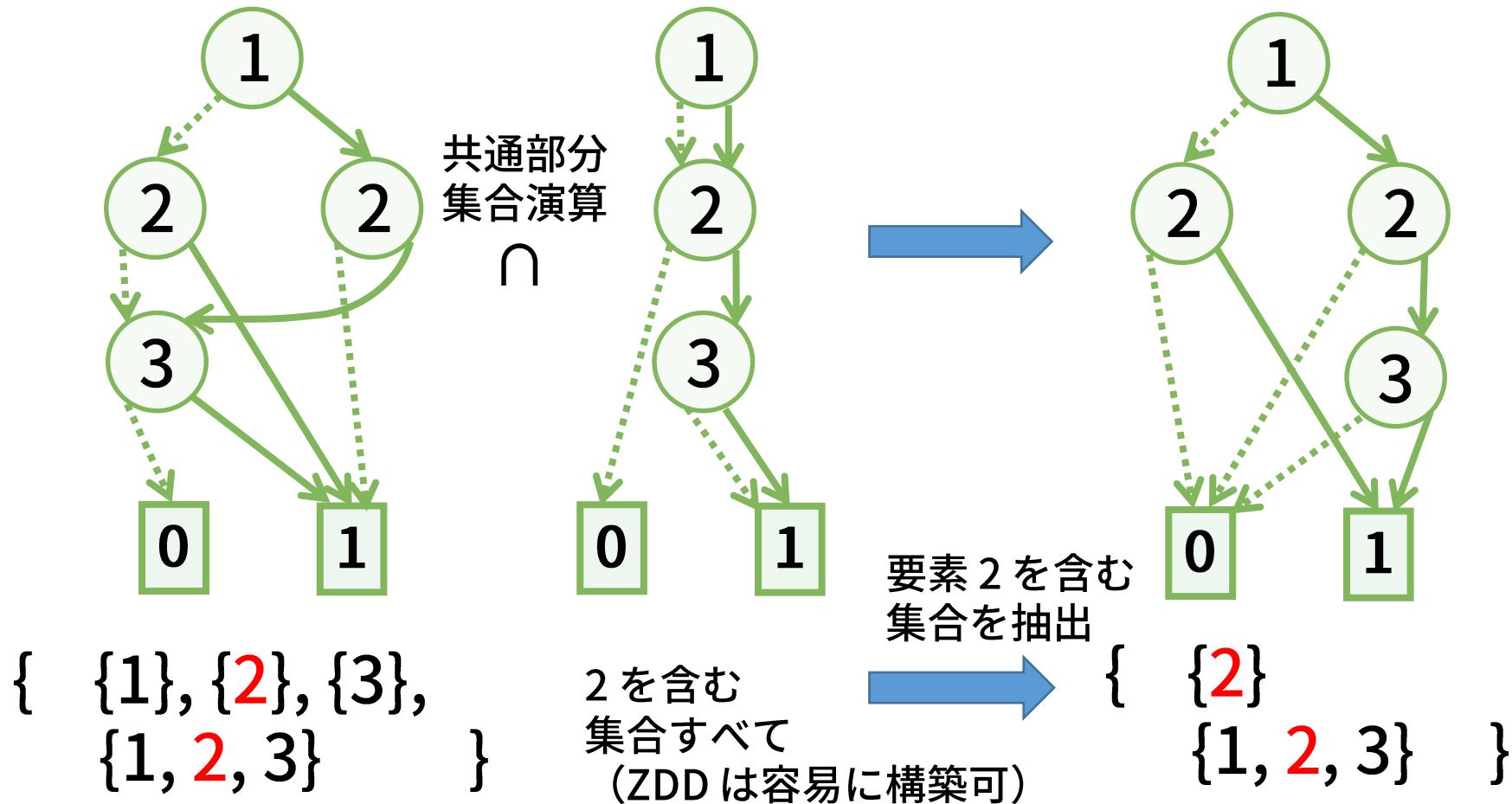
# 構築したZDDの活用: 要素の抽出

- 集合族から要素  $x$  を含む集合を抽出



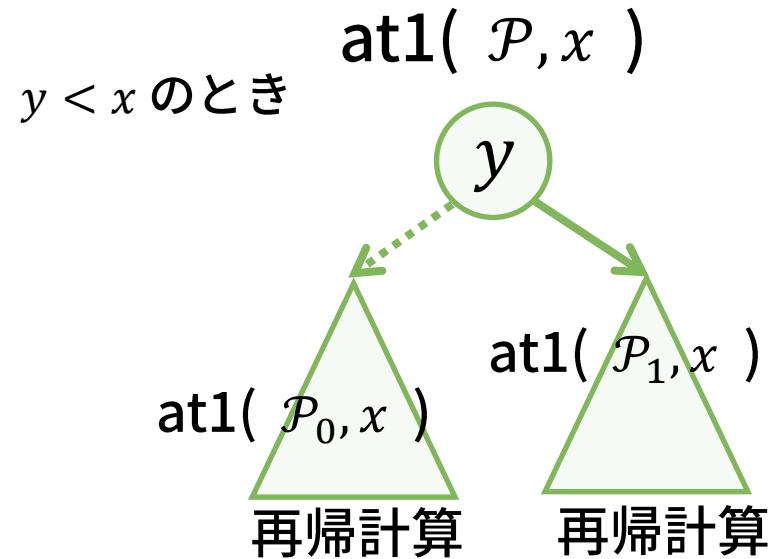
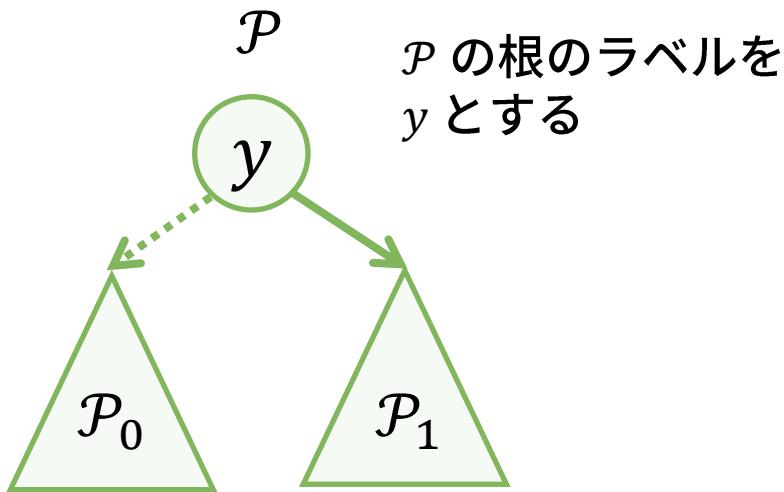
# 構築したZDDの活用: 要素の抽出: 方法 1

- 集合族から要素  $x$  を含む集合を抽出
  - 方法 1: 共通部分集合演算 (Intersec) を用いる



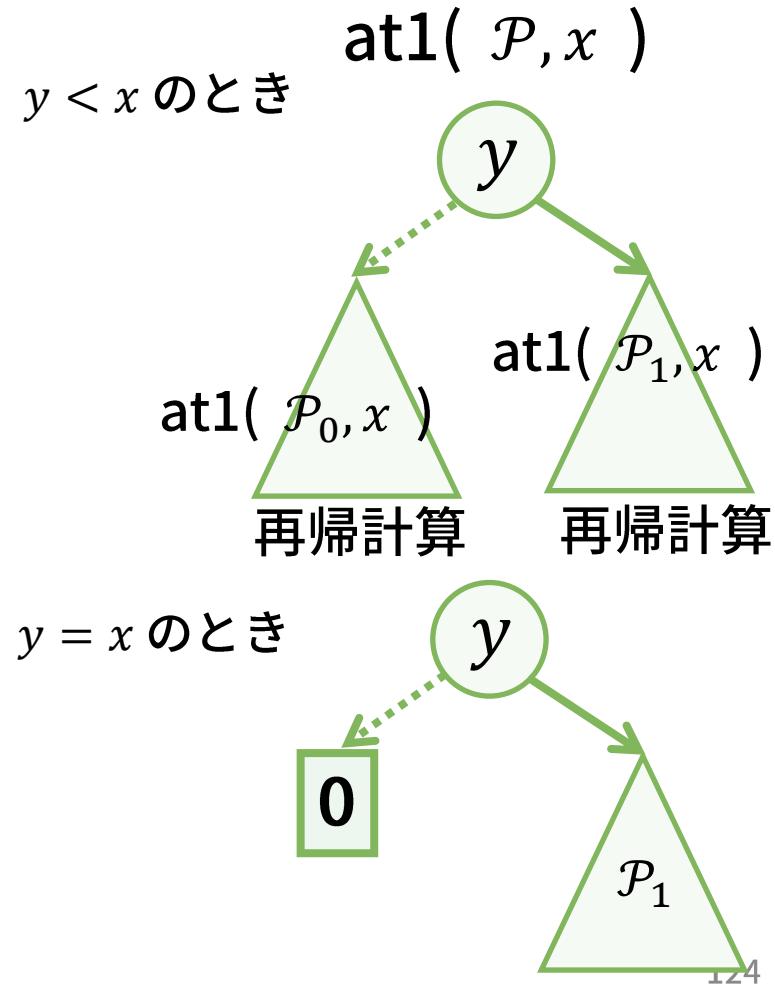
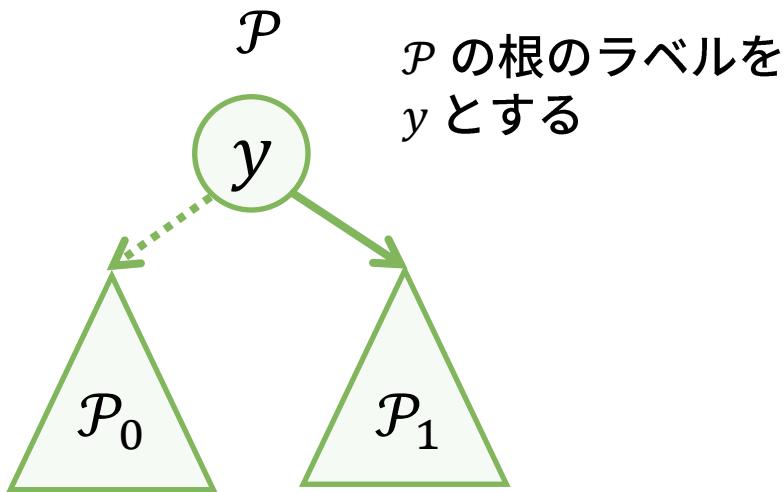
# 構築したZDDの活用: 要素の抽出: 方法 2

- 集合族から要素  $x$  を含む集合を抽出
  - 方法 2: ボトムアップ演算  $\text{at1}$  を用いる



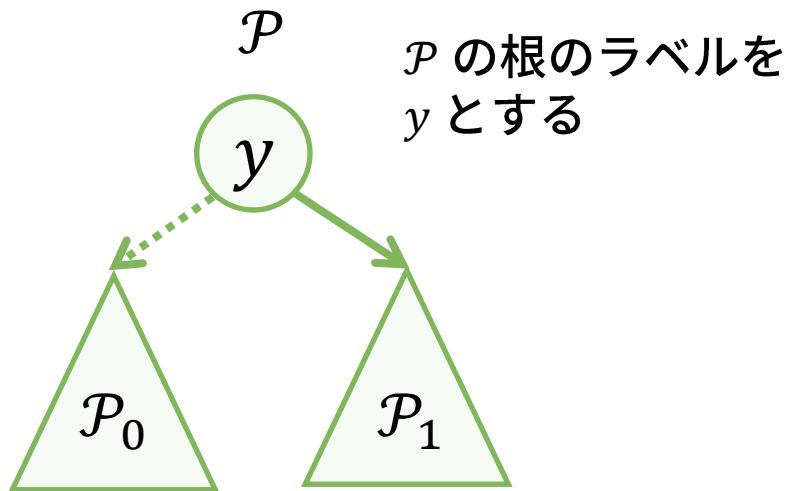
# 構築したZDDの活用: 要素の抽出: 方法 2

- 集合族から要素  $x$  を含む集合を抽出
  - 方法 2: ボトムアップ演算  $\text{at1}$  を用いる

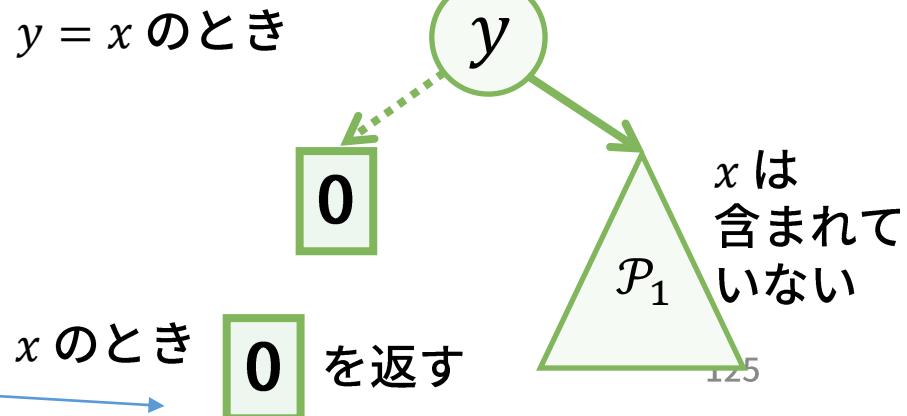
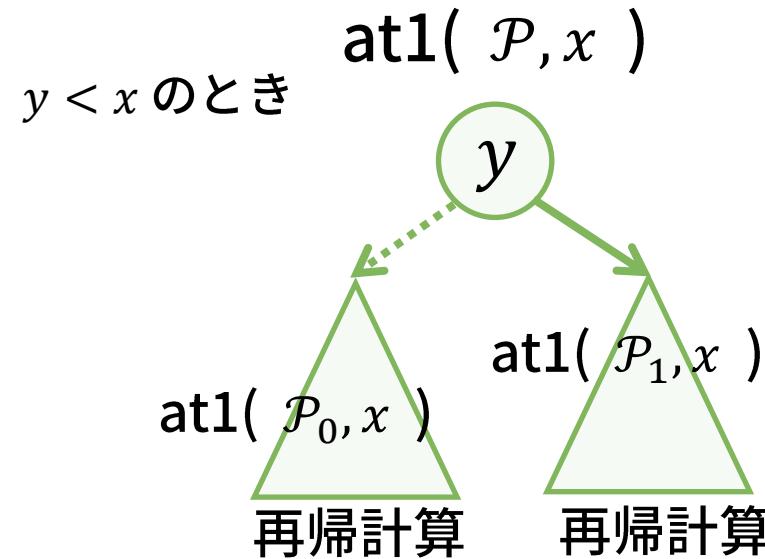


# 構築したZDDの活用: 要素の抽出: 方法 2

- 集合族から要素  $x$  を含む集合を抽出
  - 方法 2: ボトムアップ演算  $\text{at1}$  を用いる

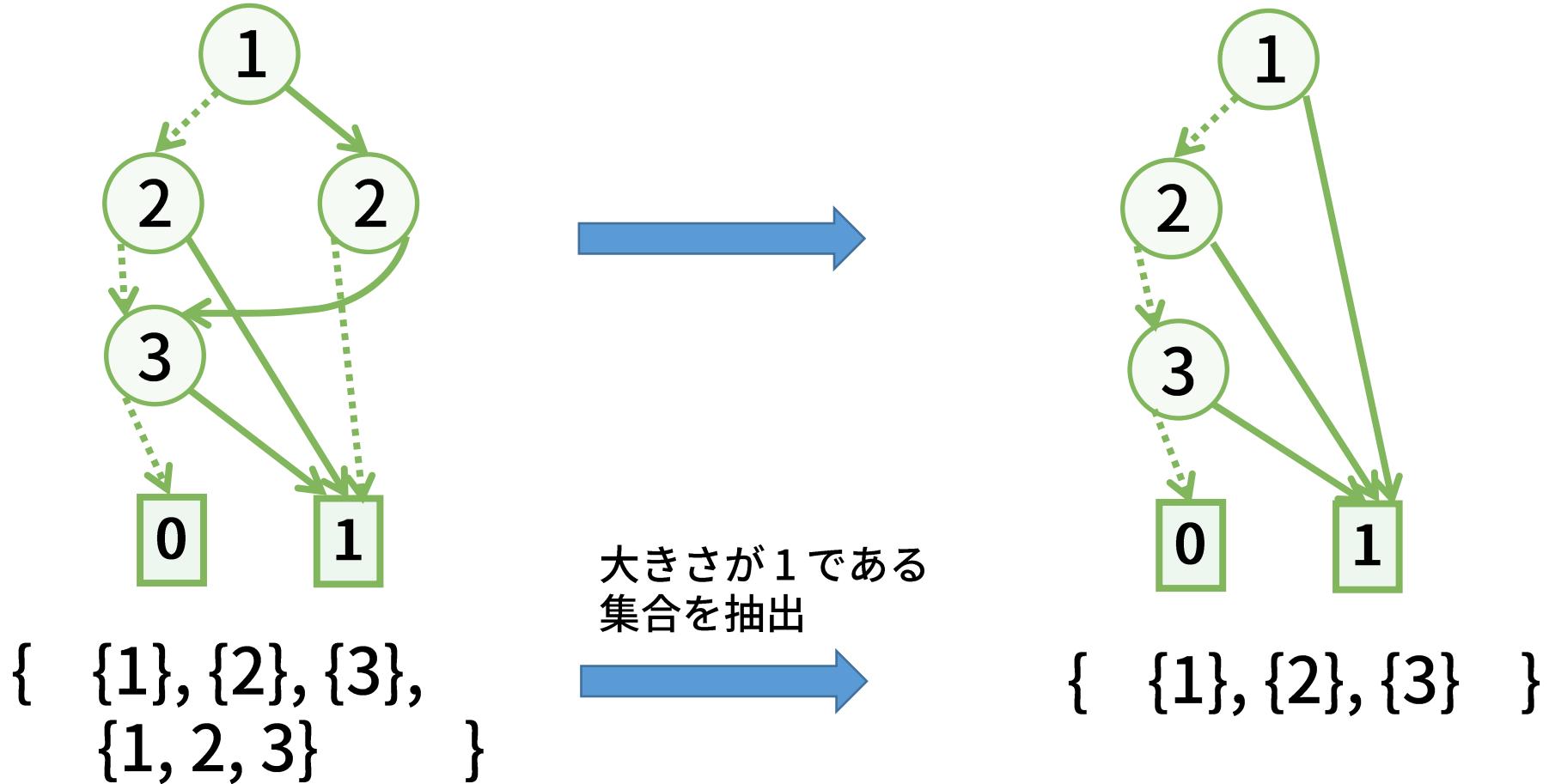


( $y > x$  なので、 $P$  は  $x$  を含む  
集合を一切含んでいない)



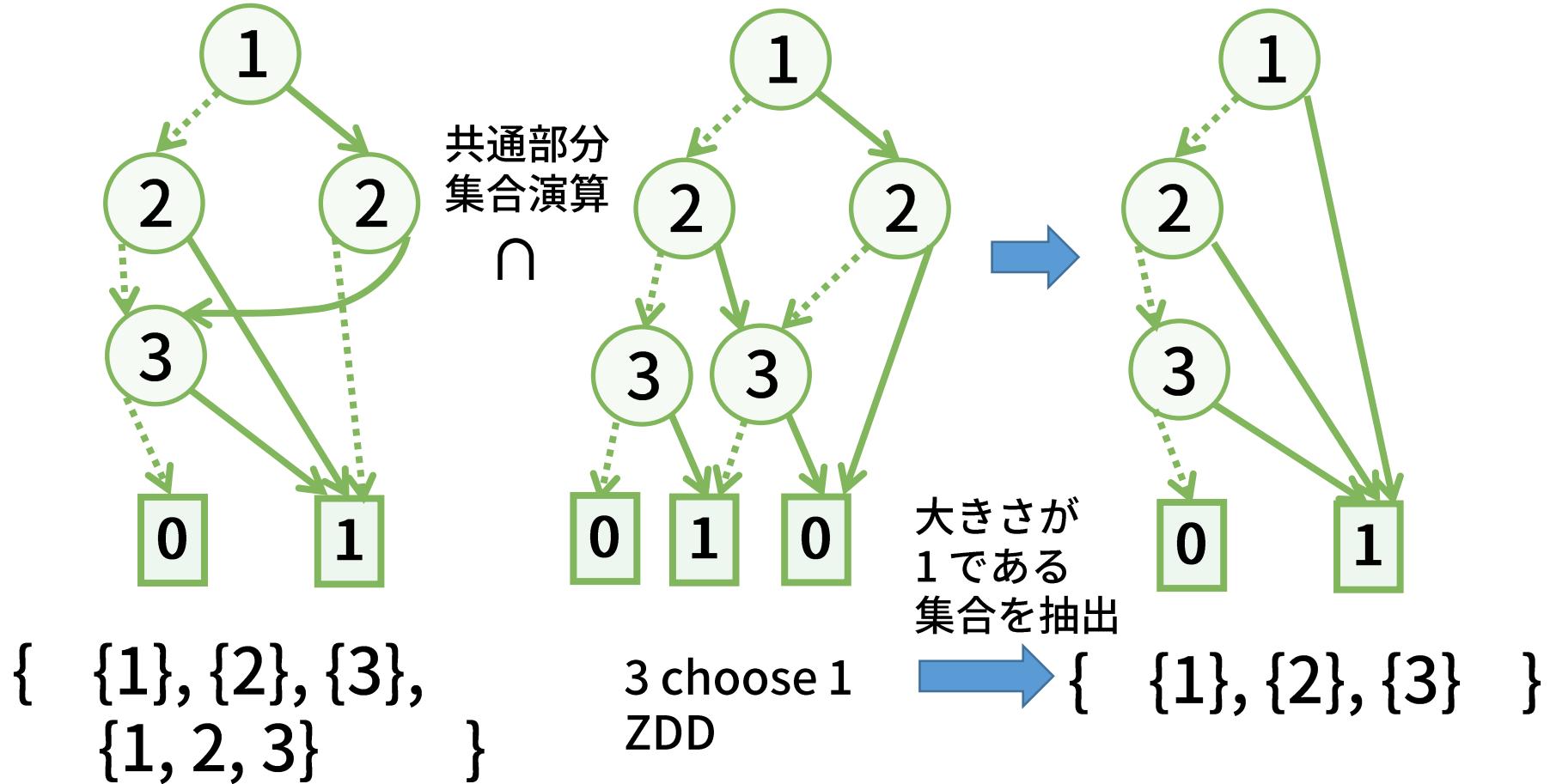
# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出



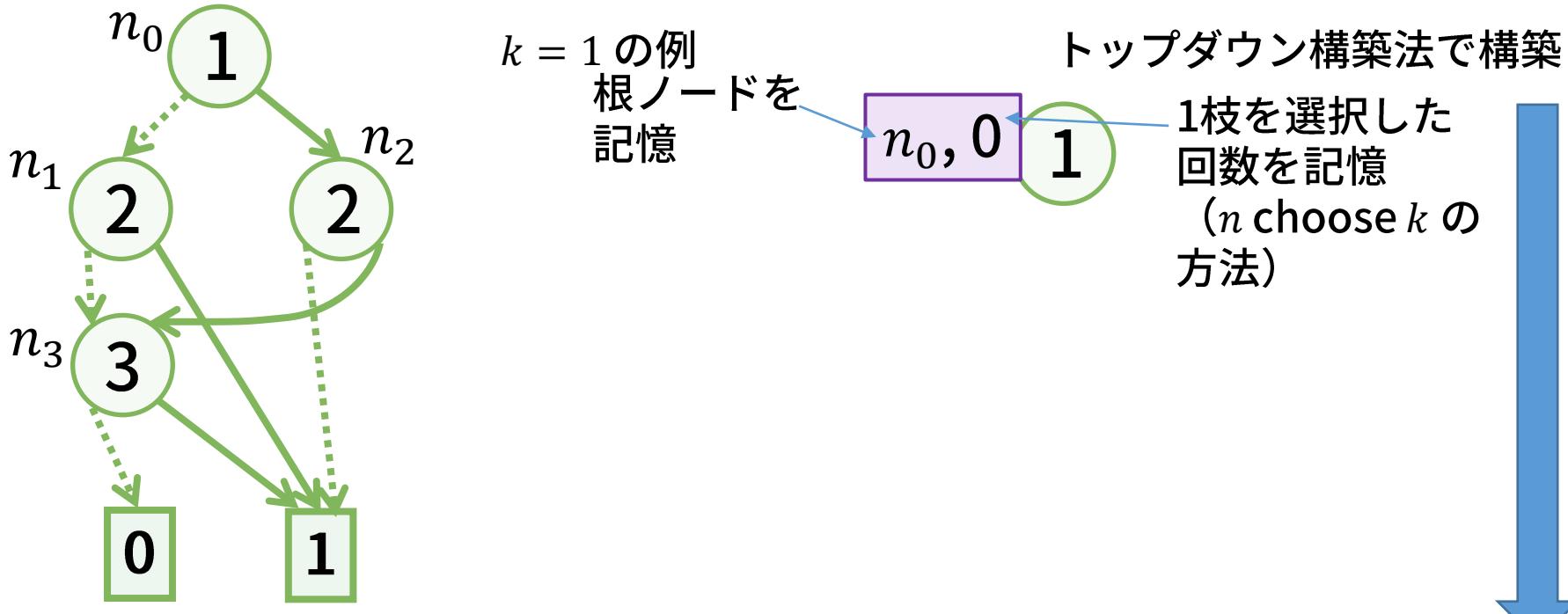
# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 1:  $n$  choose  $k$  の ZDD との共通部分集合演算



# 構築したZDDの活用: 大きさ $k$ の要素の抽出

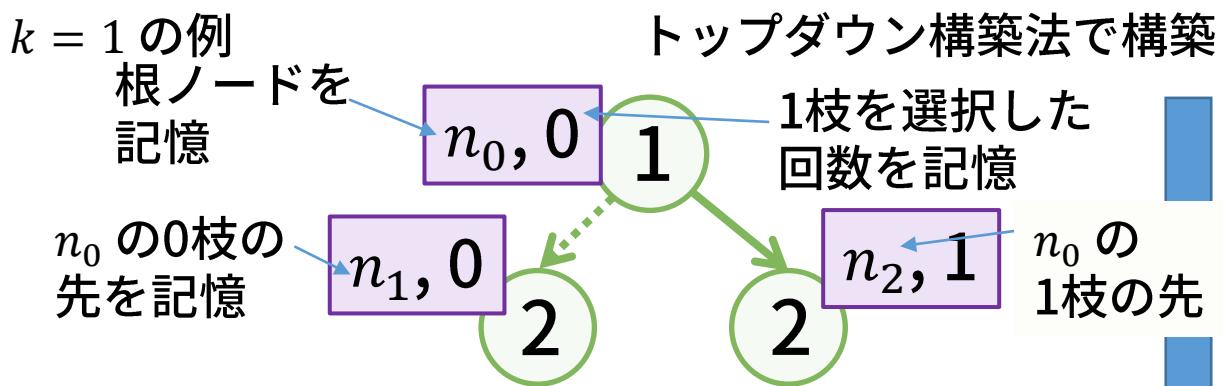
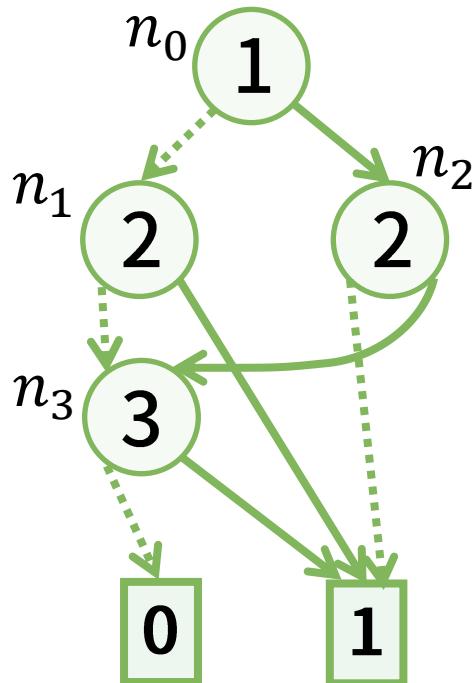
- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



{ {1}, {2}, {3},  
{1, 2, 3} }

# 構築したZDDの活用: 大きさ $k$ の要素の抽出

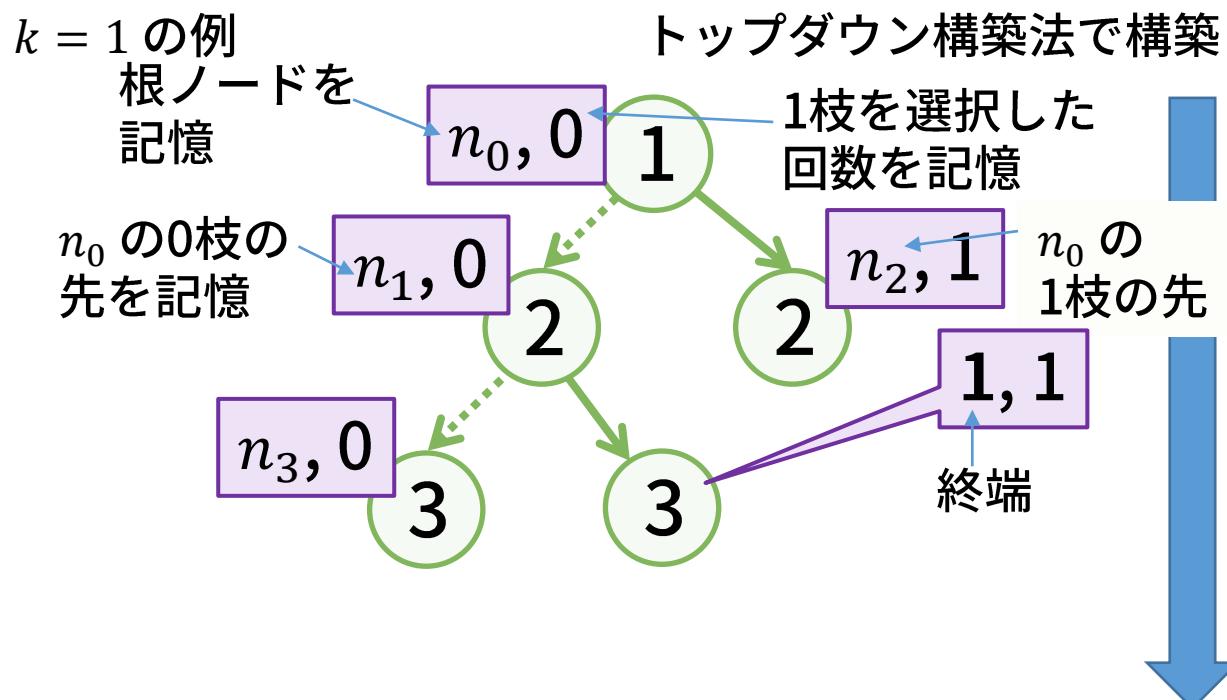
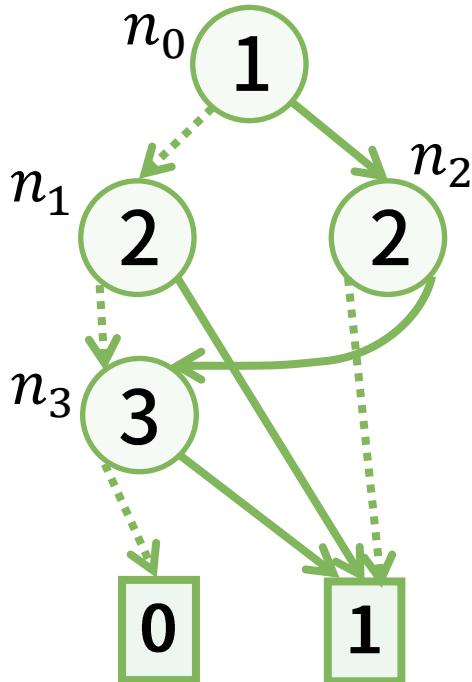
- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



{ {1}, {2}, {3},  
{1, 2, 3} }

## 構築したZDDの活用: 大きさ $k$ の要素の抽出

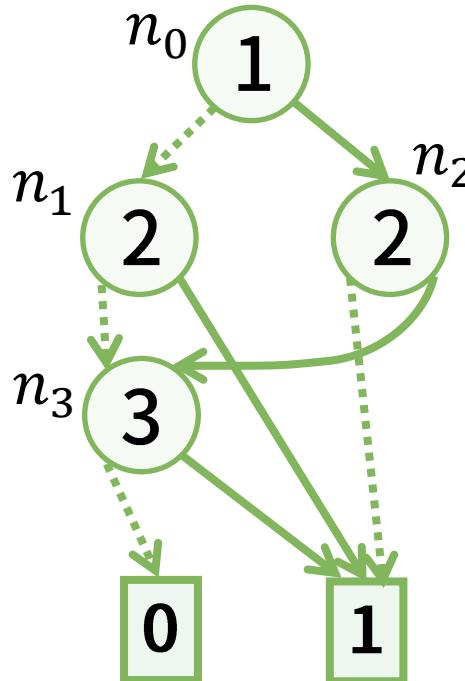
- ・集合族から大きさがちょうど  $k$  の集合を抽出
    - ・方法 2: サブセッティング法 [Iwashita, Minato 2013]



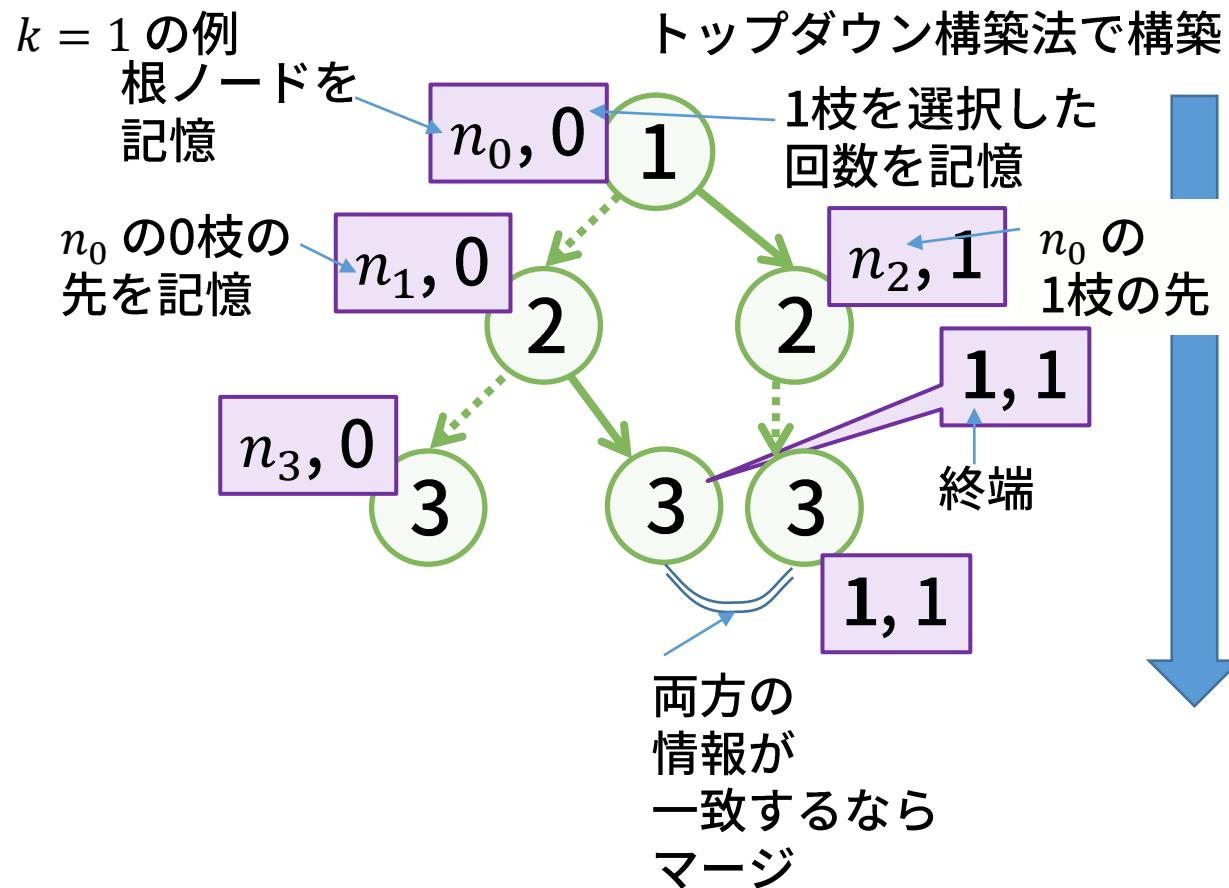
```
{ {1}, {2}, {3},  
{1, 2, 3} }
```

## 構築したZDDの活用: 大きさ $k$ の要素の抽出

- ・集合族から大きさがちょうど  $k$  の集合を抽出
    - ・方法 2: サブセッティング法 [Iwashita, Minato 2013]

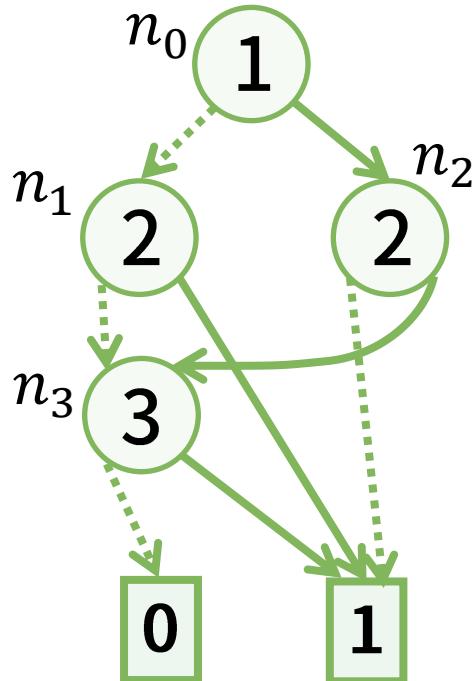


```
{ {1}, {2}, {3},  
{1, 2, 3} }
```

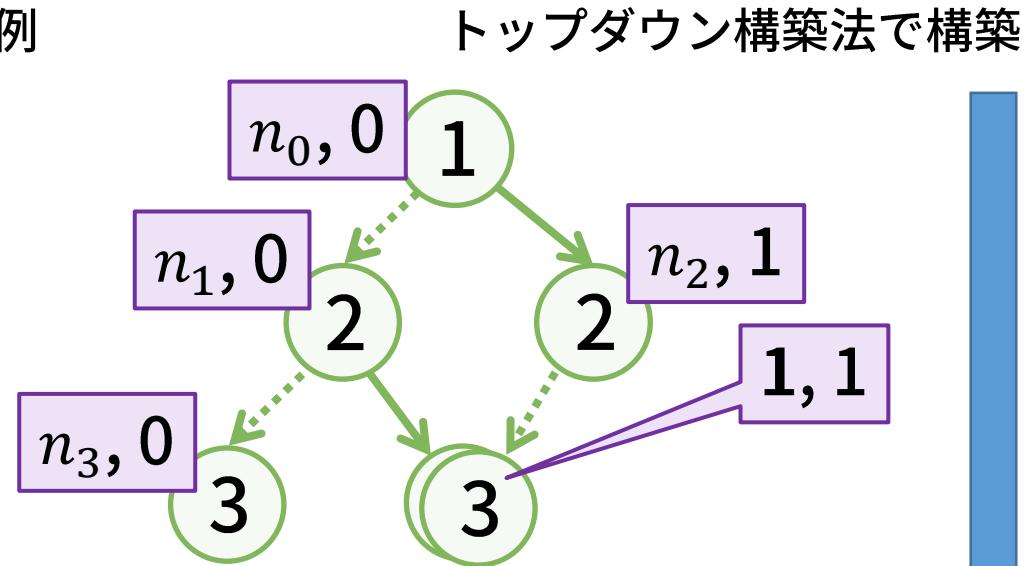


# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



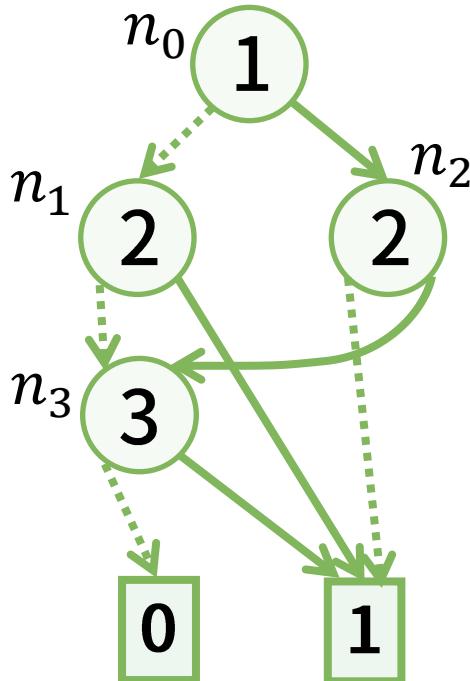
$k = 1$  の例



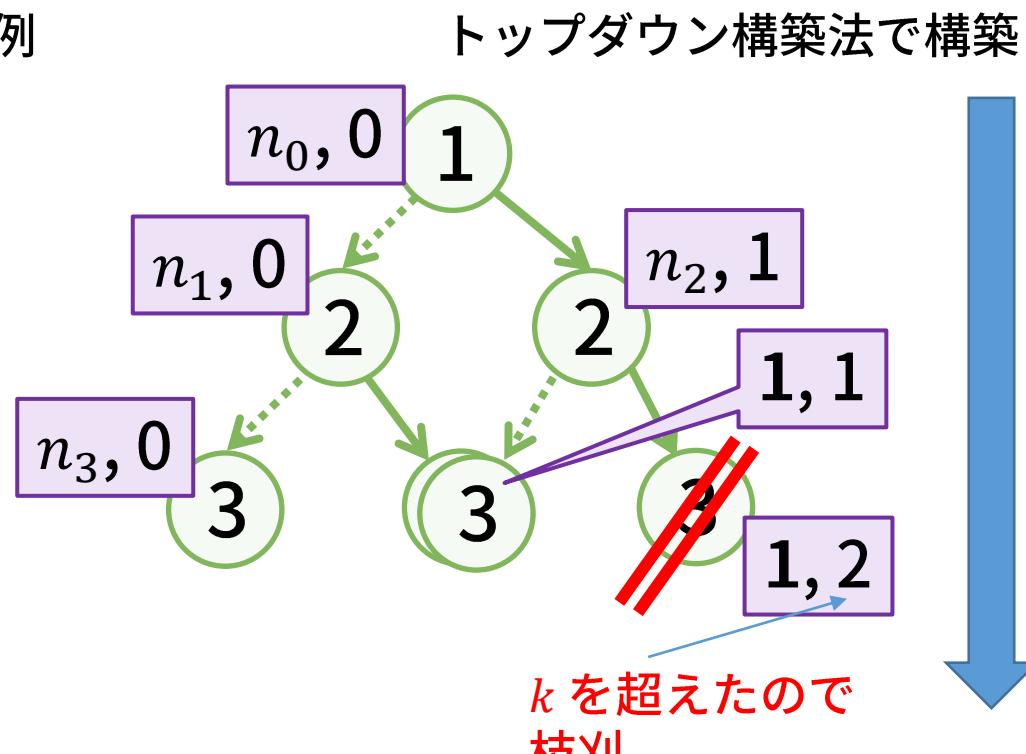
{ {1}, {2}, {3},  
{1, 2, 3} }

# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



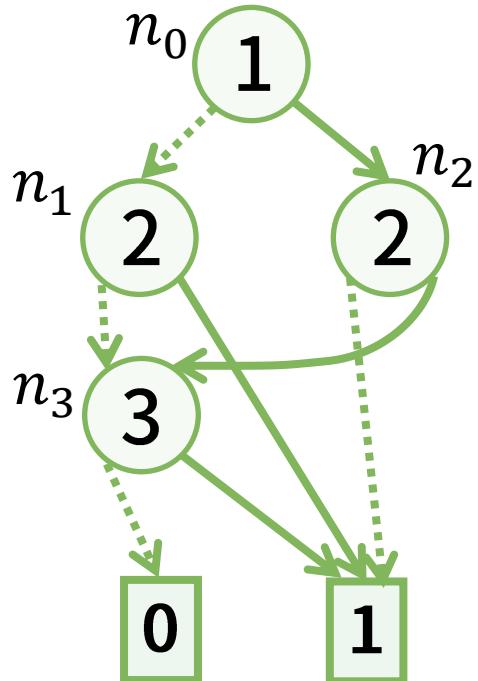
$k = 1$  の例



{ {1}, {2}, {3},  
{1, 2, 3} }

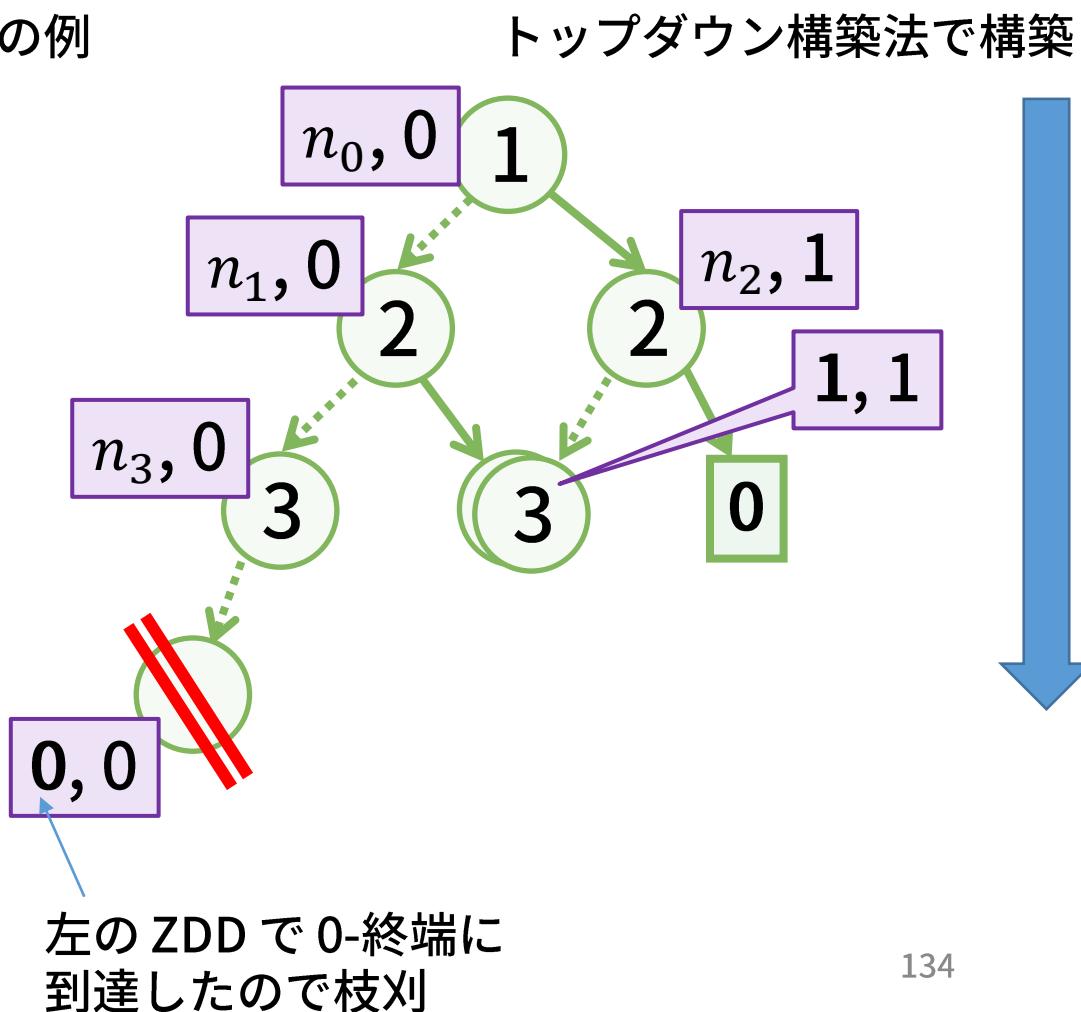
# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



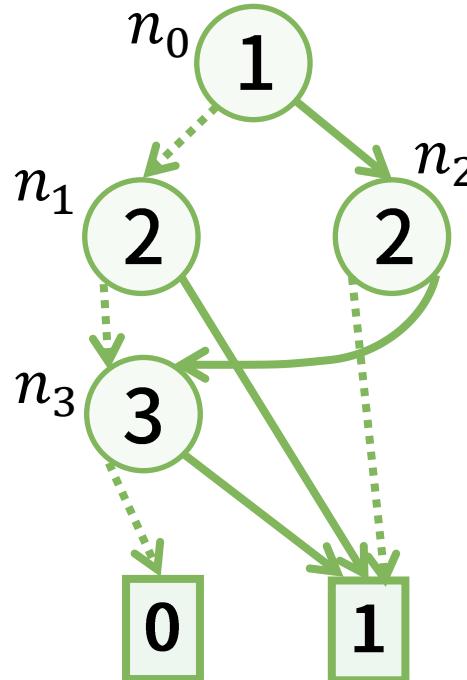
$k = 1$  の例

{ {1}, {2}, {3},  
{1, 2, 3} }



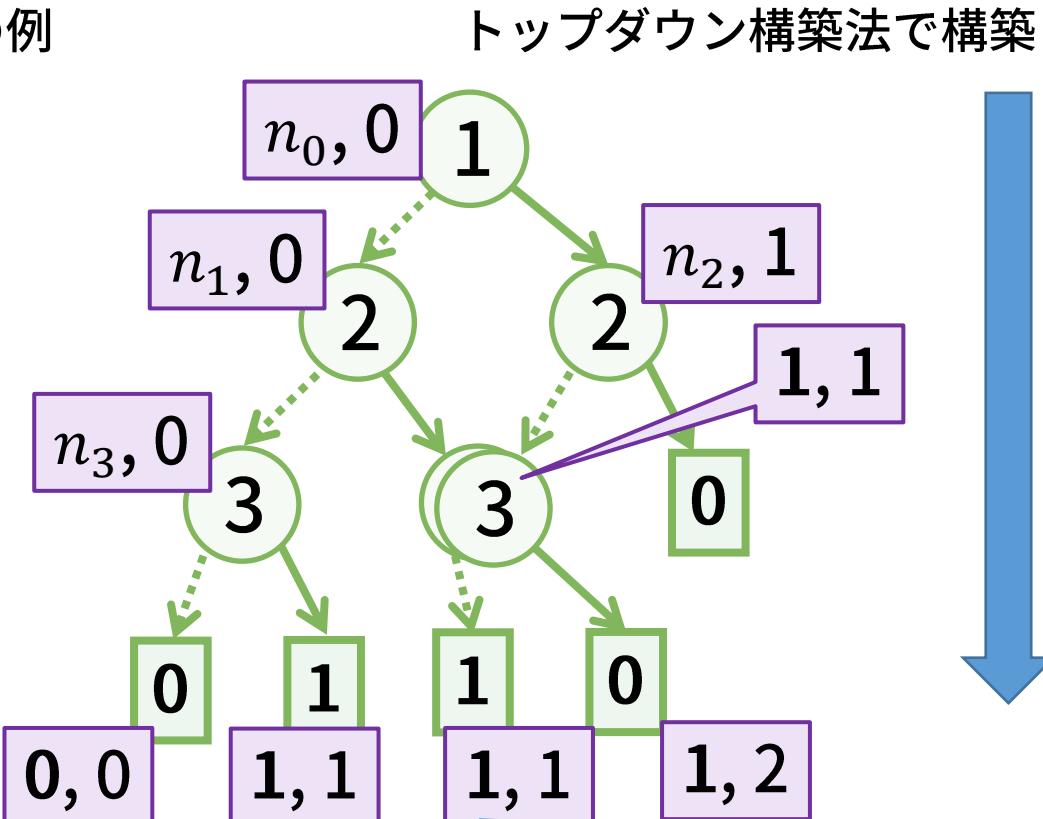
# 構築したZDDの活用: 大きさ $k$ の要素の抽出

- 集合族から大きさがちょうど  $k$  の集合を抽出
  - 方法 2: サブセッティング法 [Iwashita, Minato 2013]



$k = 1$  の例

{ {1}, {2}, {3},  
{1, 2, 3} }

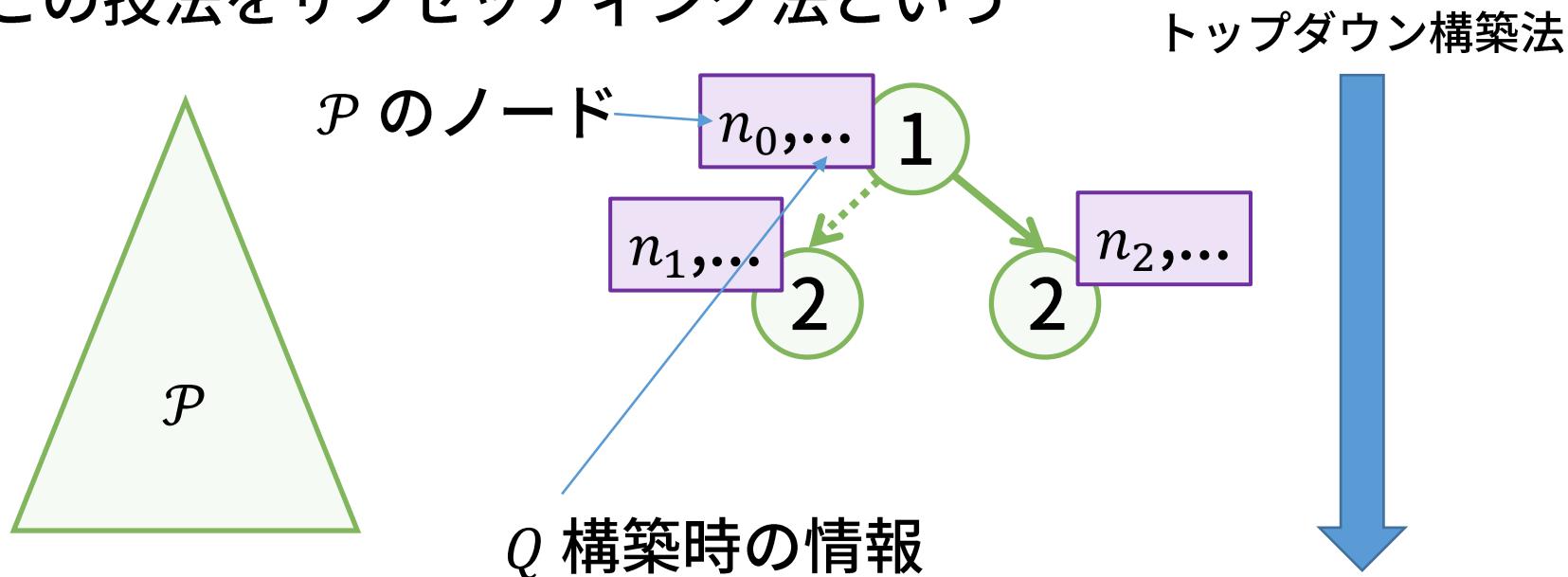


トップダウン構築法で構築

左の ZDD で 1-終端に達し、  
大きさがちょうど  $k$  の場合、1-終端になる

# サブセッティング法 [Iwashita, Minato 2013]

- 一般に、ZDD  $\mathcal{P}$  と、  
トップダウン構築法で構築する ZDD  $\mathcal{Q}$  について、  
 $\mathcal{P}$  のノードと、 $\mathcal{Q}$  構築時の情報を同時に記憶しながら  
トップダウン構築法を走らせることで、  
 $\mathcal{P} \cap \mathcal{Q}$  を表す ZDD を構築できる。  
この技法をサブセッティング法という



# ZDD以外の「DD系」データ構造

- BDD: 論理関数 [Aker 1978], [Bryant 1986]
- SeqDD (Sequential Decision Diagram): 文字列の集合 [Loekito+ 2010]
- $\pi$ DD: 置換の集合 [Minato 2011]
- SDD (Sentential Decision Diagram): 論理関数 [Darwiche 2011]
- ZSDD: 集合族 [Nishino+ 2016]

部分グラフ集合構築で考えるなら  
ZDD が path decomposition に基づくのに対し、  
ZSDD は branch decomposition に基づく

ZSDD の「フロンティア法」も考案されている [Nishino+ 2017]  
[Nakahata+ 2020]

# グラフ最適化のためのライブラリ

- **graphillion** <http://graphillion.org>
  - グラフ集合を扱うためのライブラリ
  - python や C++ で手軽に
  - パス、全域木、マッチング等、様々な対象を列挙可能

Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato,  
"Graphillion: software library designed for very large sets of labeled graphs,"  
International Journal on Software Tools for Technology Transfer, pp. 57-66, Feb. 2016

- **TdZdd** <https://github.com/kunisura/TdZdd>
  - トップダウンでZDDを構築するための汎用ライブラリ
  - 細かな挙動設定が可能
  - graphillion の内部でも使用されている

戸田 貴久, 斎藤 寿樹, 岩下 洋哲, 川原 純, 湊 真一,  
"ZDDと列挙問題 - 最新の技法とプログラミングツール,"  
コンピュータソフトウェア, 2017. (to appear)

# 参考文献

## 論文

Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, Shin-ichi Minato,  
"Frontier-based Search for Enumerating All Constrained Subgraphs  
with Compressed Representation," IEICE Transactions on  
Fundamentals of Electronics, Communications and Computer  
Sciences, vol. E100-A, no. 9, pp. 1773-1784, Sep. 2017.

## 書籍

ERATO 湊離散構造処理系プロジェクト(著), 湊真一(編集),  
"超高速グラフ列挙アルゴリズムー〈フカシギの数え方〉が拓く, 組合せ  
問題への新アプローチー," 森北出版, 2015/4/8.  
<http://www.amazon.co.jp/dp/4627852614>