ZDD (Zero-suppressed binary Decision Diagram)

超高速グラフ列挙アルゴリズム　著者: 湊 真一

理論談話会 #14

6月28日

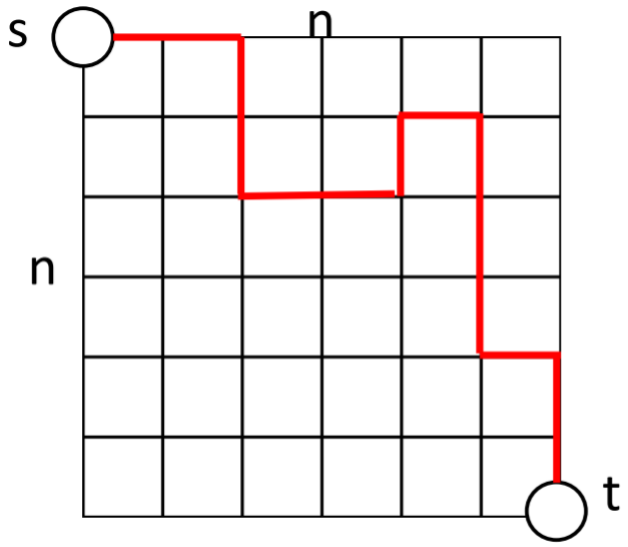
M1 石井 健太

はじめに

経路列挙の難しさ

$n \times n$の格子グラフの対角2頂点を連結する経路列挙問題

（ただし，同じところを2度通らない）



n	経路の数
1	2
2	12
3	184
4	8,512
5	1,262,816
6	575,780,564
7	789,360,053,252
8	3,266,598,486,981,642
9	41,044,208,702,632,496,804
10	1,568,758,030,464,750,013,214,100
11	182,413,291,514,248,049,241,470,885,236

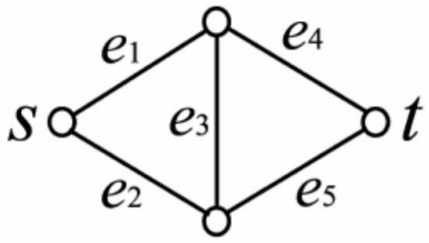
→ 11×11の場合，スパコンでも計算に290億年かかる（2015年当時）

はじめに

数え上げ方

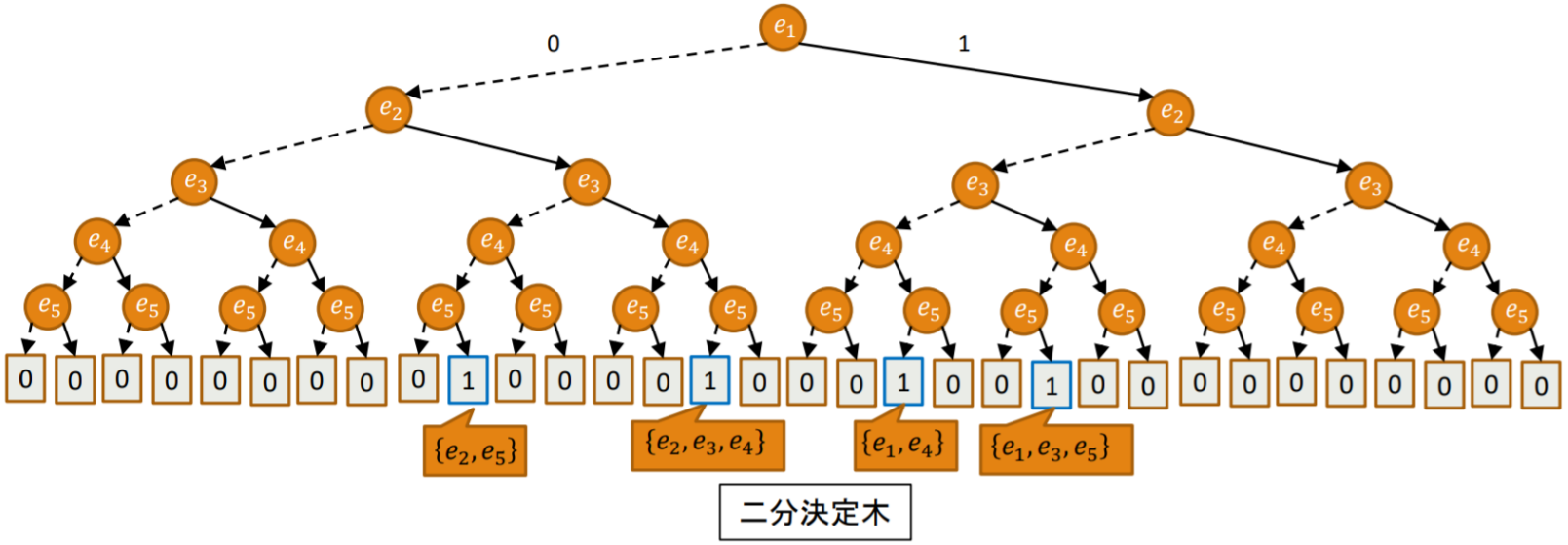
各リンクを経路として選択するかどうかの2通りの決定の連続として
場合分け二分木で考える

例)



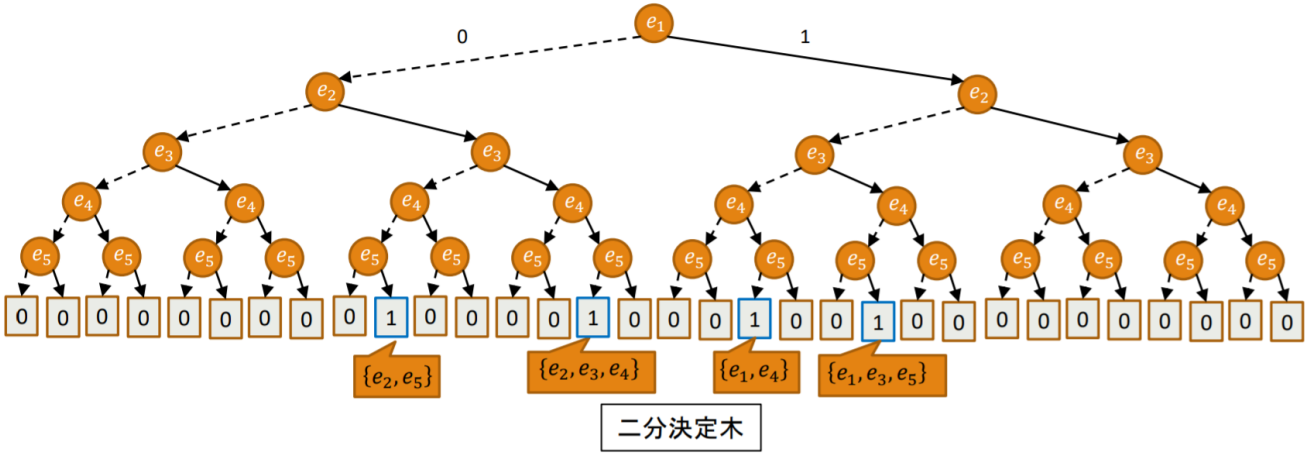
$s \to t$の可能経路集合

$\{e_1, e_4\}, \{e_1, e_3, e_5\}$
$\{e_2, e_5\}, \{e_2, e_3, e_4\}$



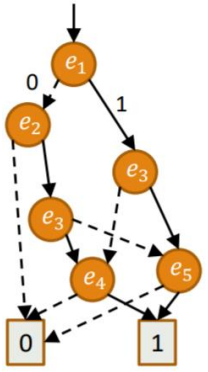
はじめに

ZDD (Zero-suppressed binary Decision Diagram)とは



二分決定木

11 × 11の格子グラフの
対角2頂点を連結する経路列挙問題
では290億年から数秒に短縮が可能



ZDD

省略や節点の共有により
ネットワークを圧縮した表現方法

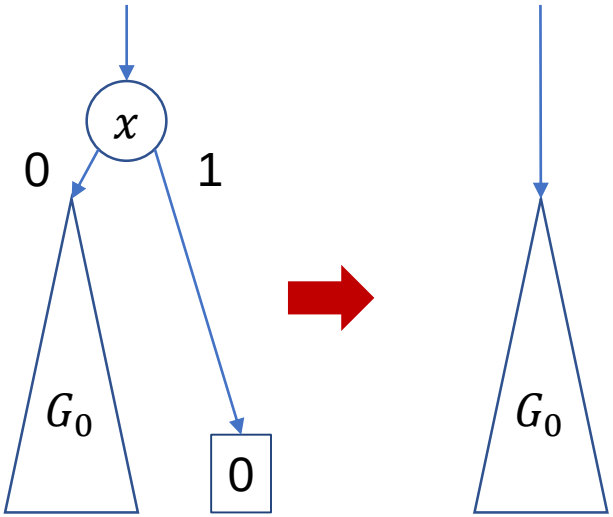
ZDDの作り方

①冗長節点の削除

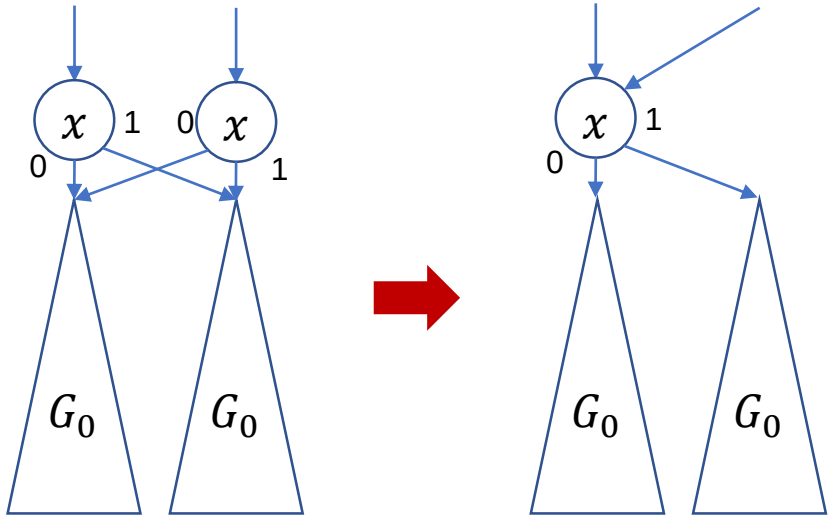
1-枝が0の値を持つ葉を指している場合に，この節点を取り除き，
0-枝の行き先に直結させる

②等価節点の共有

等価な節点（アイテム名が同じで，0-枝同士，1-枝同士の行き先が同じ）を共有する



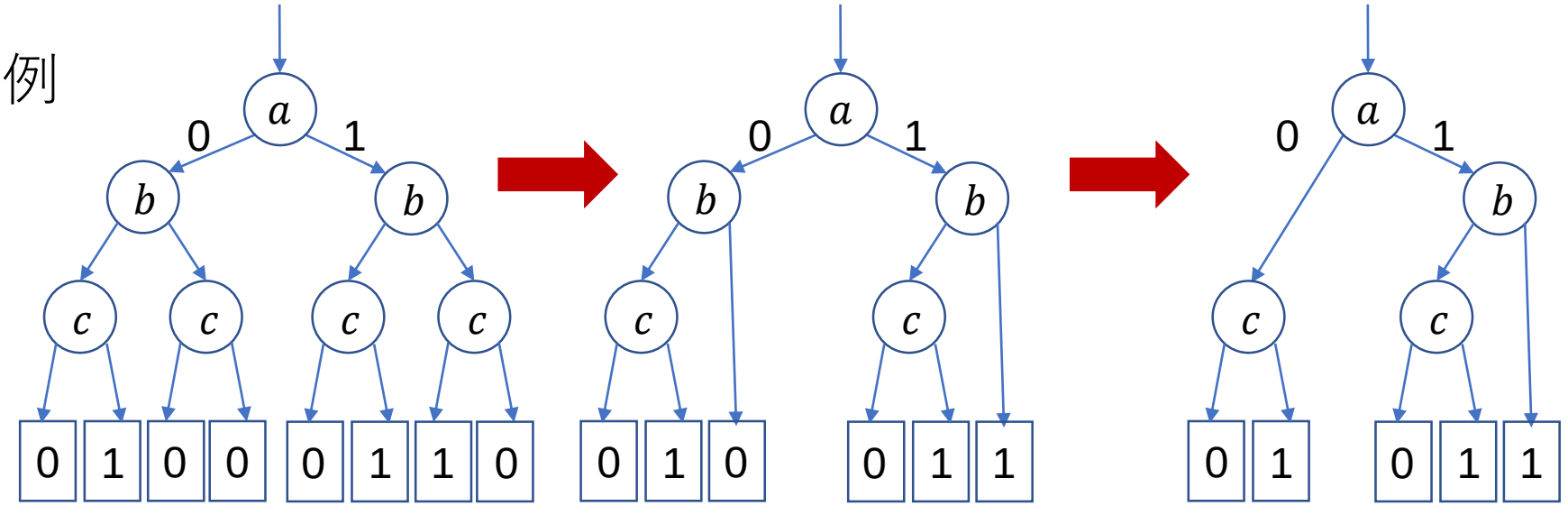
①冗長節点の削除

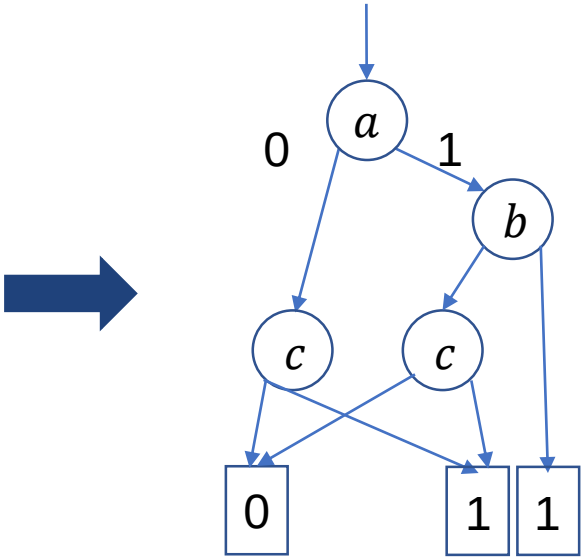


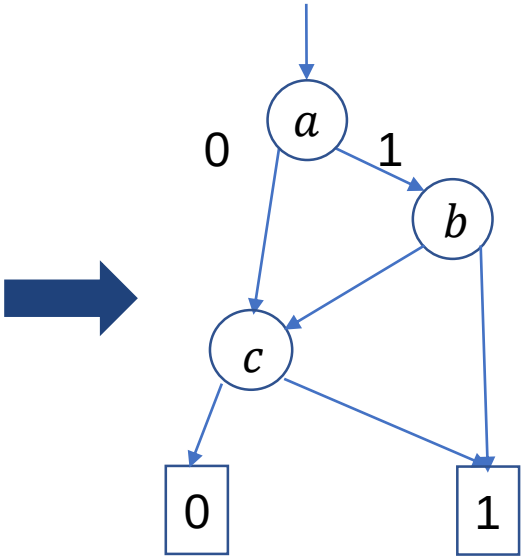
②等価節点の共有

ZDDの作り方

例







冗長節点の削除

等価節点の共有

BDDとZDDの違い

BDD (Binary Decision Diagram)
ZDD (Zero-suppressed binary Decision Diagram)

- 共通点
  - 論理関数のグラフ表現
  - 論理関数のそれぞれの変数について，0, 1を導入した結果を2分岐の枝 (0-枝，1-枝)で場合分けし，最終的に得られる論理関数の値を2値の定数節点 (0-終端, 1-終端) で表現
  - 場合分けする際変数の順序を固定
  - 等価節点の共有

$(a \wedge b) \vee c$

a	b	c	-	$(a \wedge b) \vee c$
0	0	0	→	0
0	0	1	→	1
0	1	0	→	0
0	1	1	→	1
1	0	0	→	0
1	0	1	→	1
1	1	0	→	1
1	1	1	→	1

- 相違点
  - BDD：0-枝，1-枝が同じ節点を指しているときに冗長として取り除く
  - ZDD：上記の節点は取り除かず，1-枝が0-終端を指している場合に取り除く

    = 一度も出現しないアイテムに関する節点が自動的に削除されていく．

    ➡ 組み合わせが疎な場合に効果が大きい

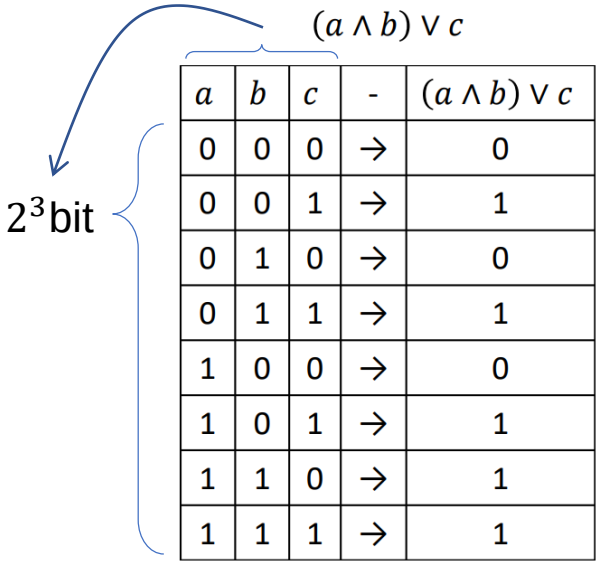
ZDDの特徴整理

1. 圧縮表現が可能
2. ZDD同士の演算でZDDを新たに作成可能
3. 索引化したデータベースとも捉えられる

ZDDの特徴整理

1. 圧縮表現が可能

単純な数え上げだと
特徴の個数Nに対して2^N bitのメモリを使う



a	b	c	-	$(a \wedge b) \vee c$
0	0	0	→	0
0	0	1	→	1
0	1	0	→	0
0	1	1	→	1
1	0	0	→	0
1	0	1	→	1
1	1	0	→	1
1	1	1	→	1

例) 8×8の格子グラフの
対角2頂点を連結する経路列挙問題

単純な二分木

辺の本数 $7 \times 8 \times 2 = 112$

局面数 2^{112} 通り？！ → エクサバイトを超える…

ZDDを用いて表現

ZDDを用いて表現した場合の頂点数：33580個
これを識別するのに16bit
112辺を識別するのに8bit → 計算すると約168キロバイト

ここまで大きなデータ量を
168キロバイトまで圧縮して
表現できる

ZDDの特徴整理

2. ZDD同士の演算でZDDを新たに作成可能（ボトムアップ的方法）

集合演算例)

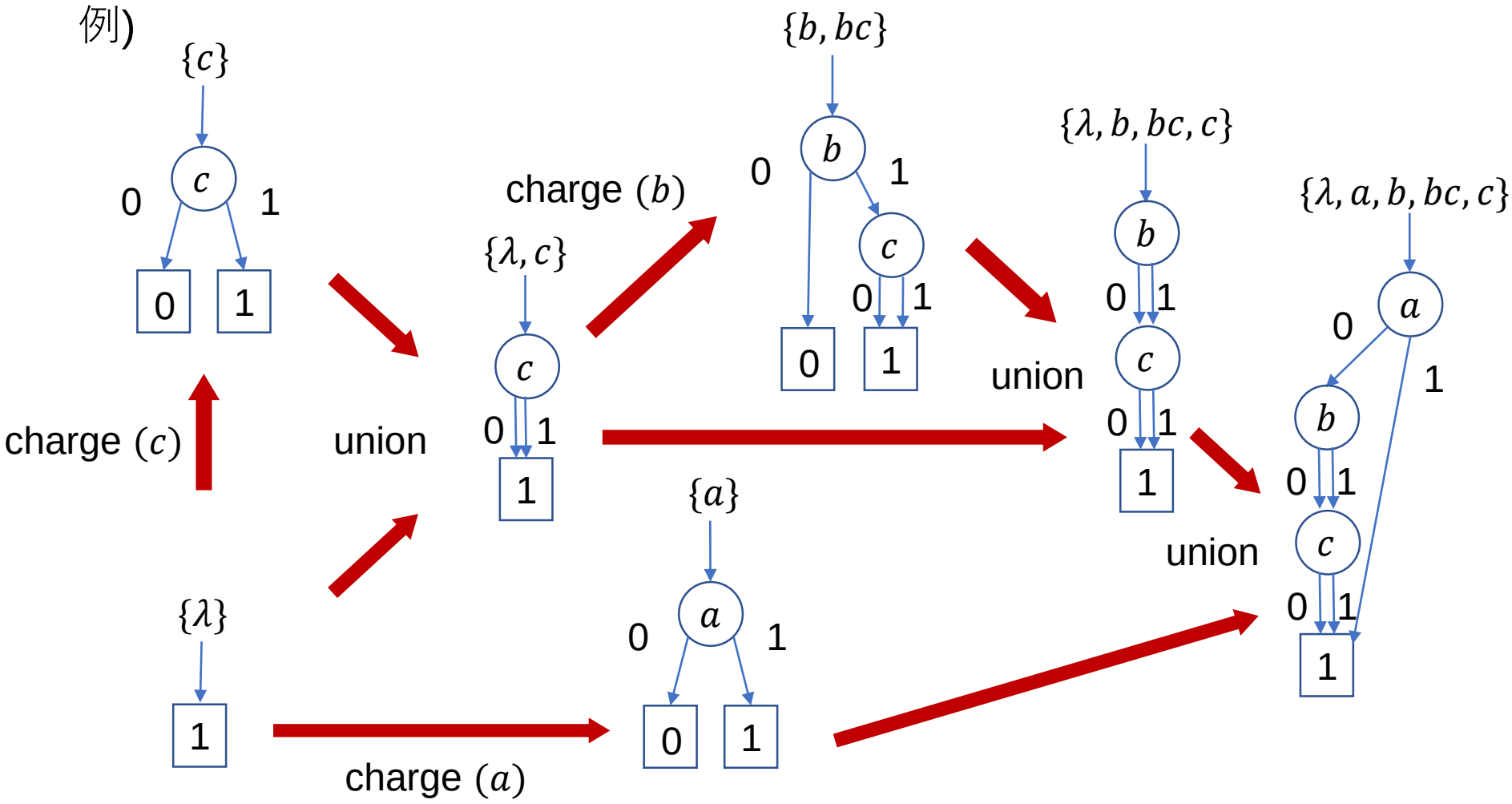
- union: 集合和をとる
- charge: 組み合わせ集合の各要素に新しいアイテムを追加する

集合演算アルゴリズムは，入力として与えられるZDDの節点数,
および出力されるZDDの節点数の総和にほぼ比例する

それぞれのZDDが圧縮されていればいるほど高速に新たなZDDを作成できる

ZDDの特徴整理

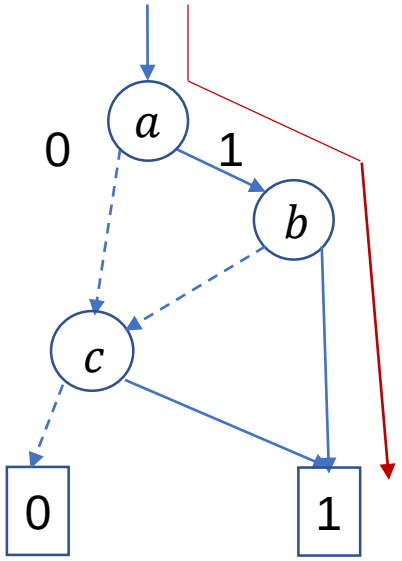
2. ZDD同士の演算でZDDを新たに作成可能（ボトムアップ的方法）



ZDDの特徴整理

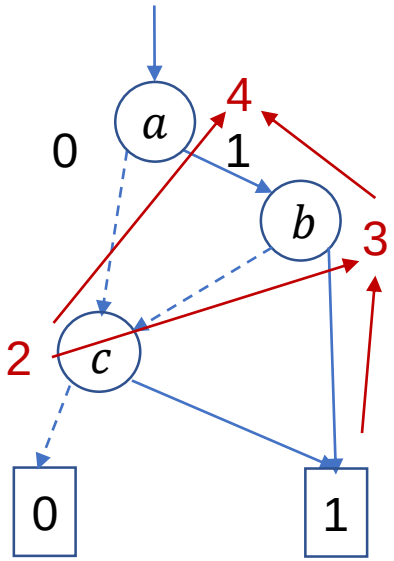
3. 索引化したデータベースとも捉えられる

・ある組合せがデータベースにあるか？



根節点からたどって1-終端節点に
行くか調べれば良い

・各接点以下の経路数が知りたい

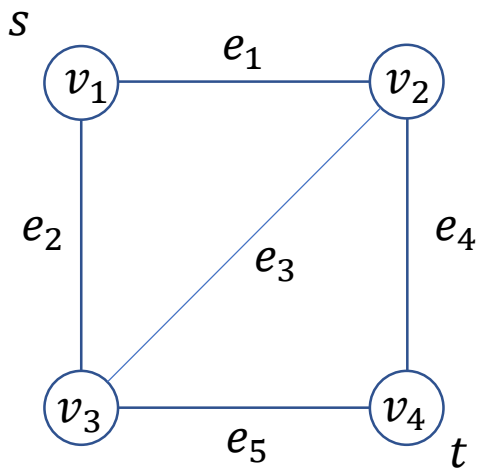


下から経路数を保存していけば,
その節点の0-枝を通る経路数と1-枝を通る
経路数の和で求めていける

ZDDを用いたグラフ列挙アルゴリズム

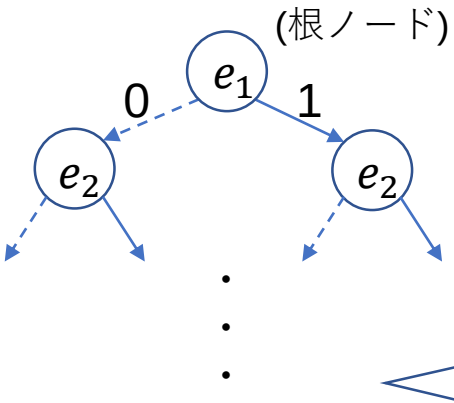
- フロンティア法（トップダウン的方法）

  根ノードから下方向に向かって順に構築して行く方法



集合演算によるボトムアップ的手法では一度全探索を行って作ったZDDを組合せていくことで新たなZDDを構築する

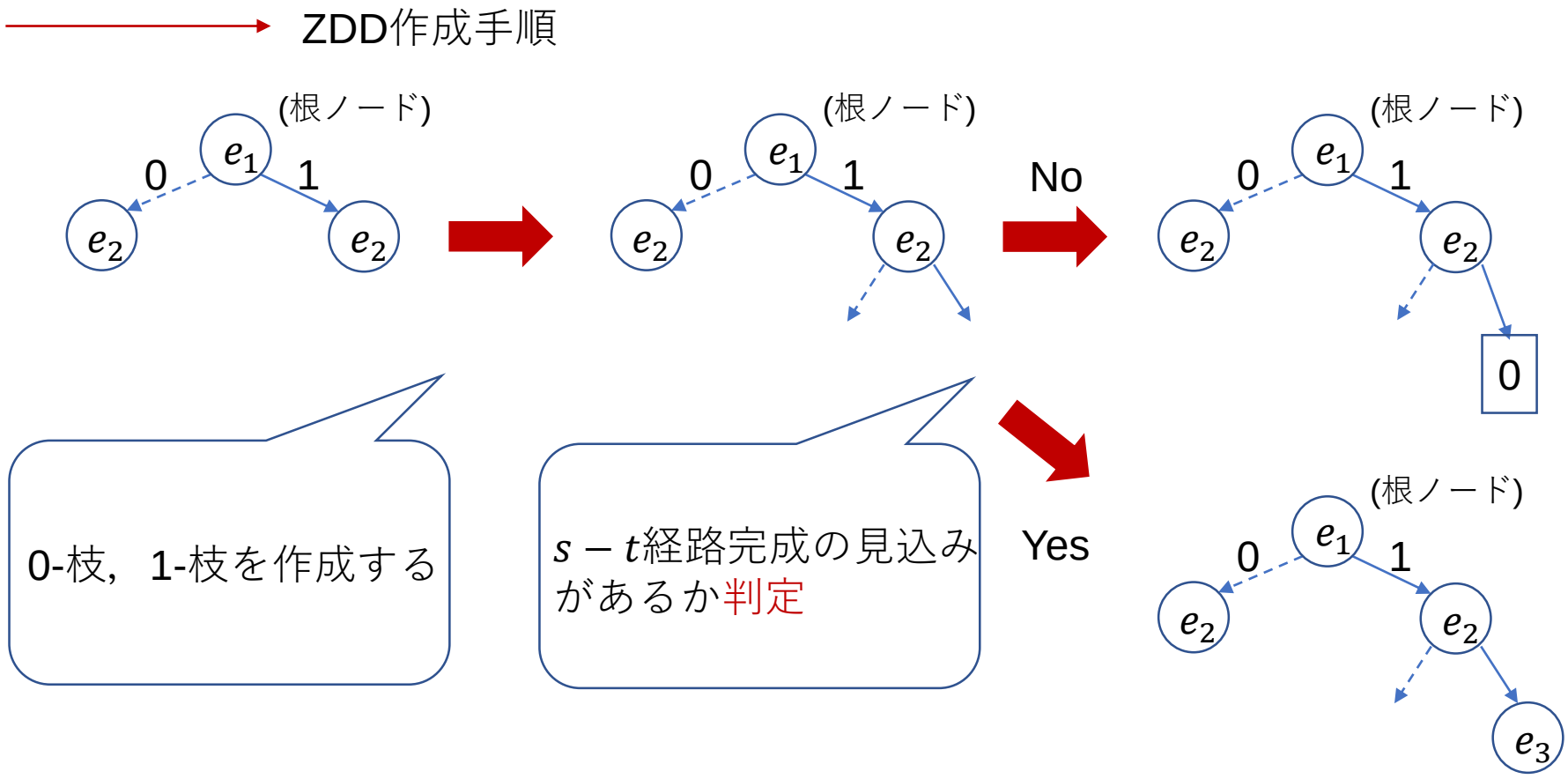
例）　左図のような全体グラフG_1において$s-t$経路を列挙する問題を考える



作りながら枝刈りやノードの共有をしてZDDを構築していきたい

ZDDを用いたグラフ列挙アルゴリズム

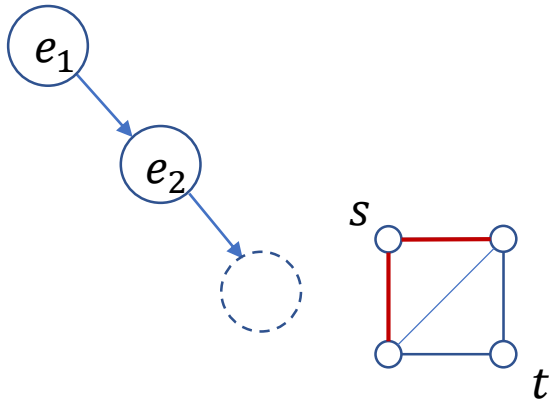
- 枝刈りとは？



ZDDを用いたグラフ列挙アルゴリズム

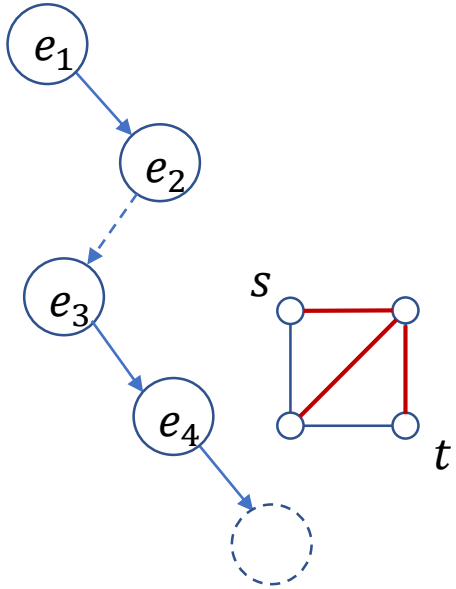
- $s-t$経路完成見込み判定基準

1)



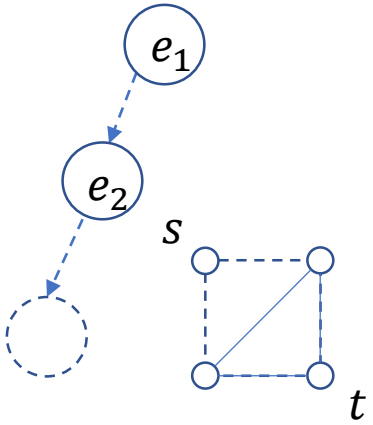
sまたはtに辺が2本以上接続する

2)



s,t以外の頂点に辺が3本以上接続する

3)

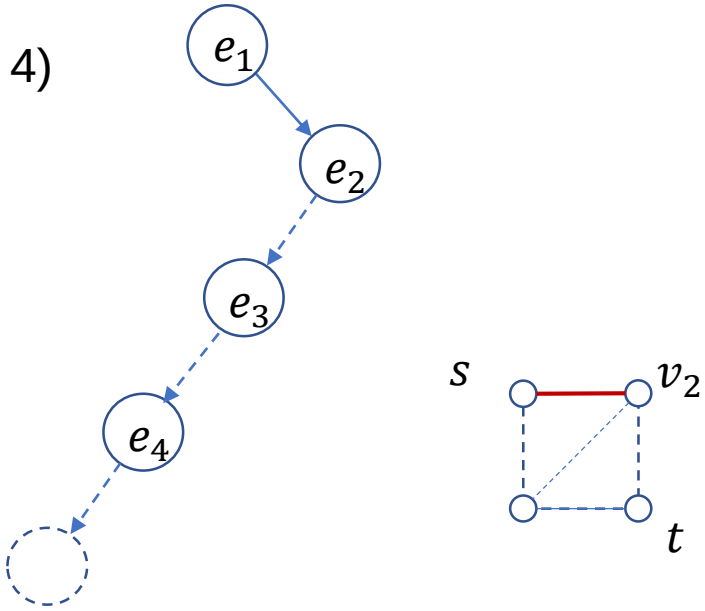


sまたはtの次数が0に確定する
＝ sまたはtに辺が接続しない

ZDDを用いたグラフ列挙アルゴリズム

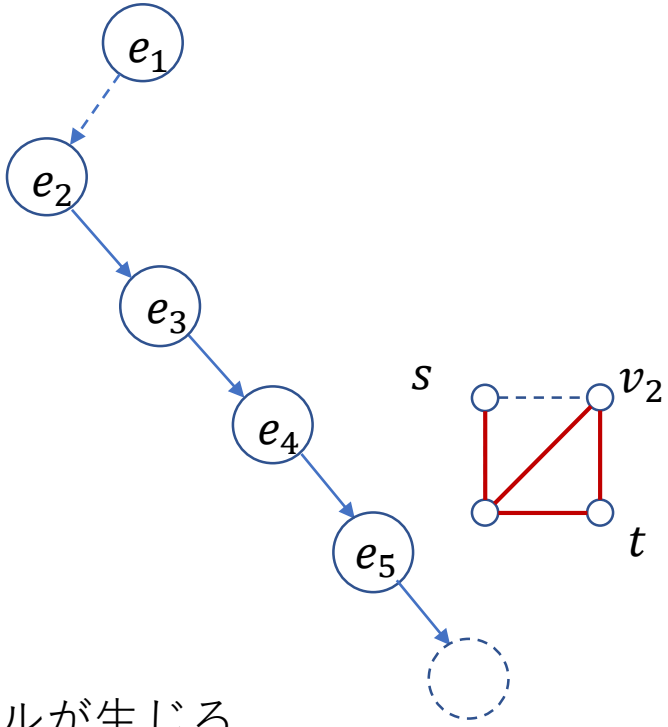
- $s-t$経路完成見込み判定基準

4)



$s-t$経路が途中で分断される
＝ v_2の次数が0に決定される

5)



サイクルが生じる
残りの辺をどう加えても$s-t$
経路の完成は見込めない

ZDDを用いたグラフ列挙アルゴリズム

(1)~(5)を判定するために各ノードに以下の2つの配列を記憶させる

・deg: 部分グラフの各頂点の次数を記憶する

頂点vの次数deg$[v]$, ノードnが持つ配列n.degで表す

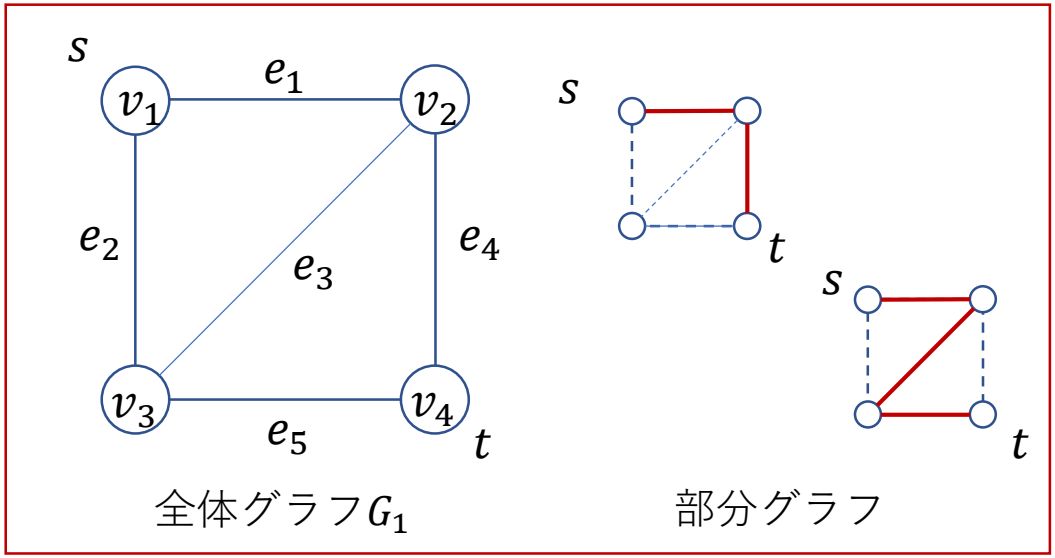
・comp: 各頂点の連結成分の番号を記憶する

頂点vとwが部分グラフの同じ連結成分に含まれる場合

comp$[v]$=comp$[w]$

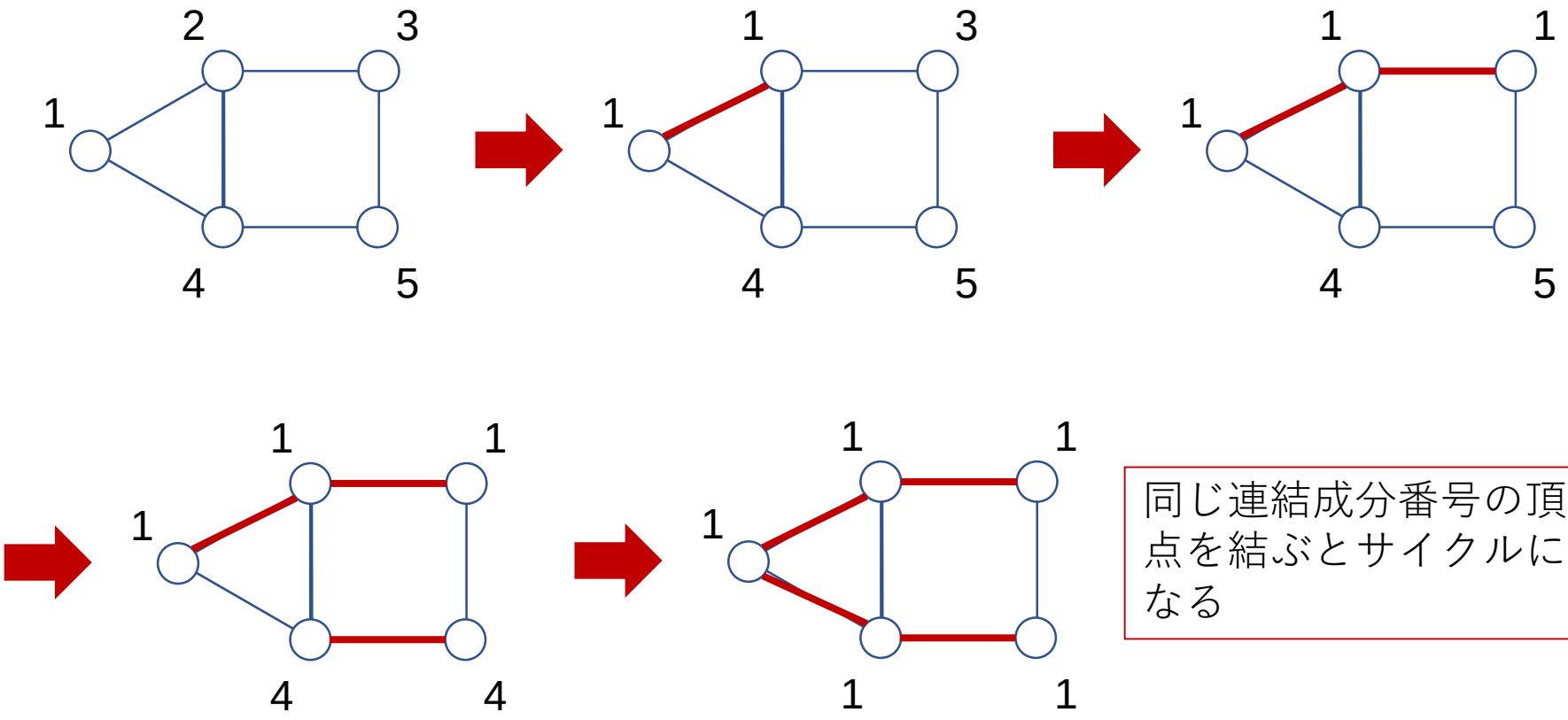
そうでない場合

comp$[v]$≠comp$[w]$



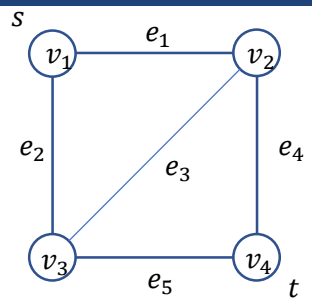
ZDDを用いたグラフ列挙アルゴリズム

comp配列値更新の例）



同じ連結成分番号の頂点を結ぶとサイクルになる

ZDDを用いたグラフ列挙アルゴリズム



枝刈りの例）

deg: 0 0 0 0
comp: 1 2 3 4

e_1

0 0 0 0
1 2 3 4

1 1 0 0
1 1 3 4

e_2 e_2

e_3 e_3

1 0 1 0
1 2 1 4

1 1 0 0
1 1 3 4

e_4 e_4 e_4 e_4

1 0 1 0
1 2 1 4

1 1 2 0
1 1 1 4

1 1 0 0
1 1 3 4

1 2 1 0
1 1 1 4

e_5 e_5 e_5

1 0 1 0
1 2 1 4

1 2 2 0
1 1 1 4

1 2 0 1
1 1 3 1

1 2 1 0
1 1 1 4

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

(3),(4) (1),(2),(5) (1),(4) (3),(4)

(4) (2)

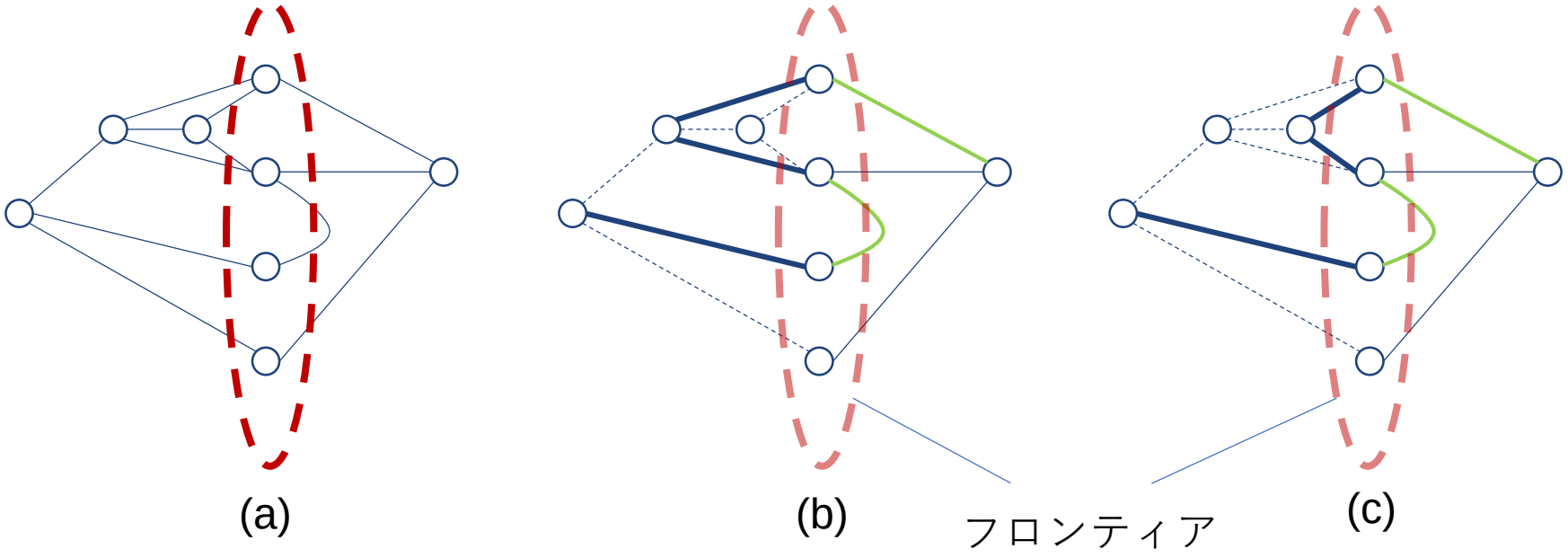
sまたはtの次数が0に確定する (3)

(1) sまたはtに辺が2本以上接続する

s, t以外の頂点に辺が3本以上接続する

ZDDを用いたグラフ列挙アルゴリズム

- フロンティア法におけるノードの共有



楕円より左側まで辺の処理が完了しているとする（例：(b), (c)の楕円右側）

楕円内の全頂点について楕円の左側での接続のされ方が同じであれば，楕円の右側での辺の取捨選択の仕方は同じになる

➡ フロンティア内のみdeg, compを記録しておけばよい

ZDDを用いたグラフ列挙アルゴリズム

- フロンティア法におけるノードの共有

ノードの共有の条件

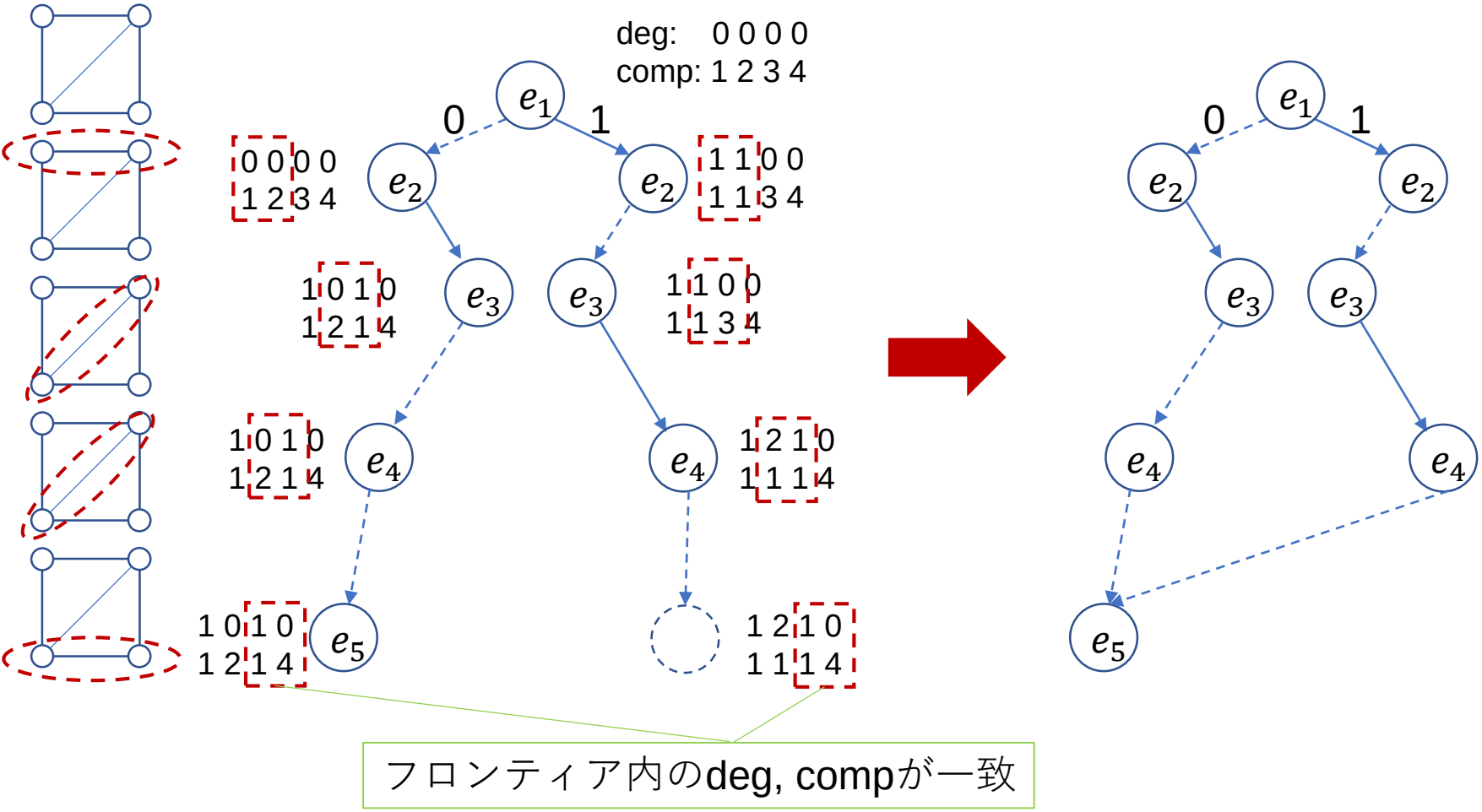
2つのノードn', n''について，フロンティア上のすべての頂点について

$$\begin{cases} n'.\mathrm{deg} = n''.\mathrm{deg} \\ n'.\mathrm{comp} = n''.\mathrm{comp} \end{cases}$$

を満たすとき， n', n''の子葉は完全一致し，共有可能

ZDDを用いたグラフ列挙アルゴリズム

- フロンティア法におけるノードの共有



まとめ

- ZDDは組み合わせ集合を圧縮して表すことができるデータ構造
- ZDDの代表的な作り方にはボトムアップ的手法と
  フロンティア法がある
- 行動モデルの分野では，ネットワーク次第で経路列挙に大きなメモリを費やす．ZDDを用いて記憶しておけば，計算負荷を大幅に軽減できる

Tips

- Pythonライブラリに**Graphillion**というものがあり，これを使うと簡単に**ZDD**を構築，参照できる
- **Ekilion**というサイトは鉄道網の経路探索を**Graphillion**を用いて実装しており，誰でも気軽に大都市近郊区間大回りや最短ルートなど様々な経路を検索できるようになっている