フロンティア法について
～応用範囲絶賛拡大中！～

斎藤 寿樹 (ERATO)

共同研究

井上 武, 岩下 洋哲, 川原 純(ERATO), 岸本 章宏(東工大),
津田 宏治(産総研, ERATO), 堀山貴史(埼玉大),
湊 真一(北大, ERATO), 吉仲 亮(京大)

フロンティア法に関連する講演

- フロンティア法について
  - 担当：斎藤
- ZDDとフロンティア法を用いた電力配電網の制約充足解列挙
  - 担当：井上さん
- ZDDサブセッティング手法とそれによるフロンティア法の高速化
  - 担当：岩下さん
- 3つの直方体を折れる共通の展開図の発見に向けて
  - 担当：堀山さん
- ZDDを用いたリンクパズルアルゴリズム
  - 担当：吉仲さん

BDD/ZDD

- **BDD/ZDD** [Bryant 1986] [Minato 1993]
  - BDD: **論理関数**を何でも表せます
    ZDD: **組合せ集合**の表現が得意
  - xDD への演算体系 : 圧縮したまま演算可能できる
  - 制約条件を個別に xDD でどう表すかを考える
  - すべての制約条件を合わせた xDD を作る
    (組み合せる順番によっては、途中の xDD が爆発)

圧縮表現

フロンティア法でBDD/ZDDを構築しよう！

フロンティア法

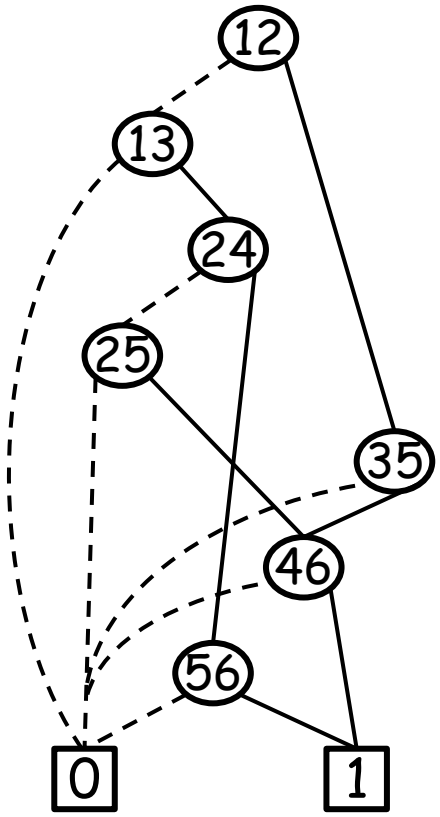
- **BDD/ZDD をトップダウンで直接（ラフに）**作る
  - パーティションの列挙 [Sekineら(1995) , Hardy ら(2007)]
    パスの列挙 [Knuth 2009] を **一般化**した**枠組み**
  - マッチング、全域木、（クリーク、）などなど
    適用範囲が日々拡大中
- むちゃくちゃ速い *!!*

完全マッチングを例に説明

[Minato 1993]

ZDD (Zero-suppressed Binary Decision Diagram)

{ { 12, 35, 46 }, { 13, 24, 56 }, { 13, 25, 46 } }



12
13
24
25
35
46
56

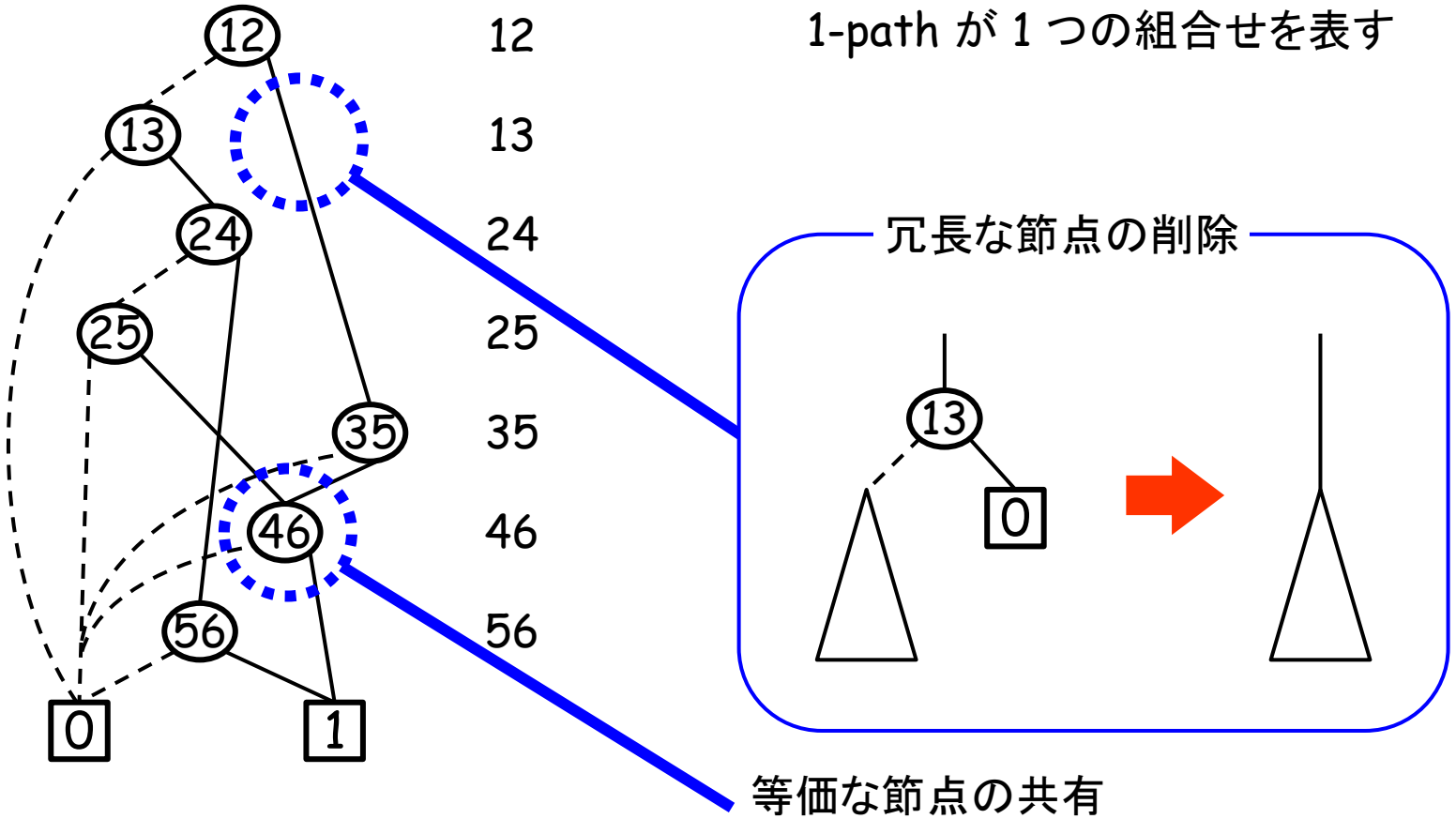
1-path が 1 つの組合せを表す

特長

- 変数の出現順序を定めると、組合せ集合を**一意に表現**
- **コンパクト**に表現
- 演算を**高速**に行える

ZDD (Zero-suppressed Binary Decision Diagram)

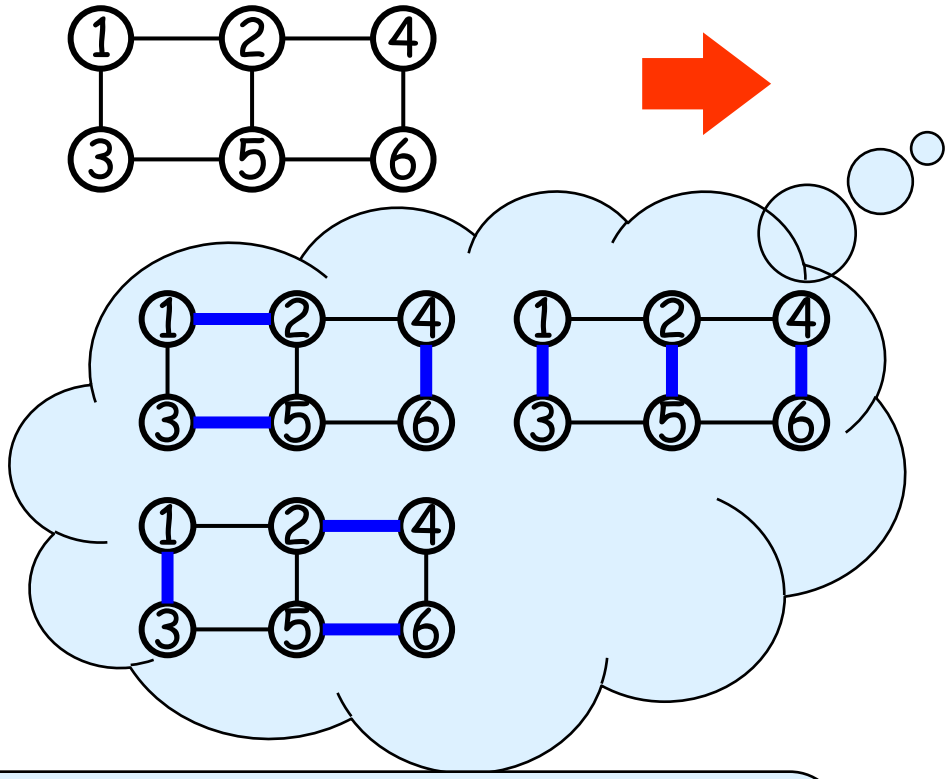
{ { 12, 35, 46 }, { 13, 24, 56 }, { 13, 25, 46 } }

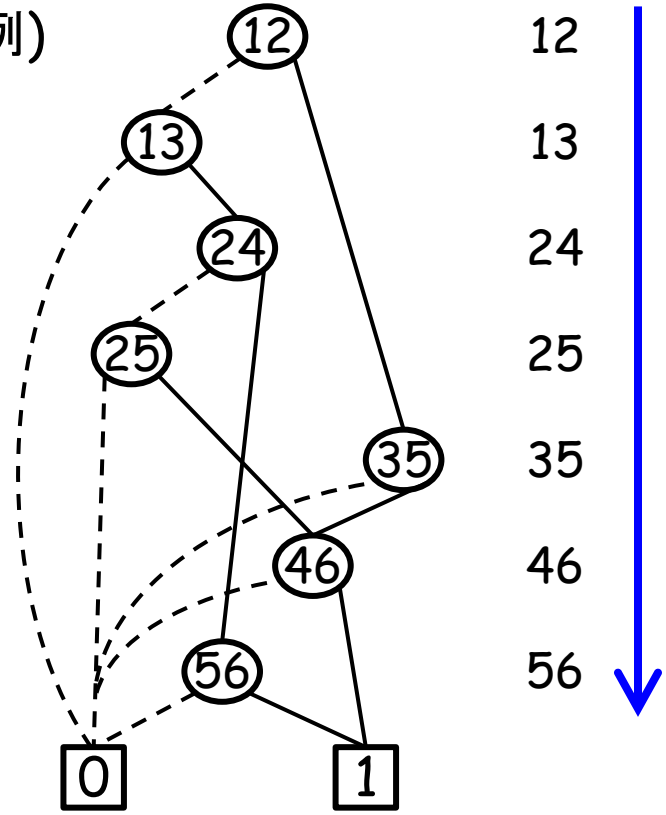


Frontier 法で、やってみよう

辺を順番に調べる

- 完全マッチング (すごく簡単な例)





- 頂点は、2つ以上の頂点とマッチすることはない
- どの頂点ともマッチしない頂点があってはいけない

- **0節点，1節点**に行くのを早めに検知できるといいな
- **節点を共有**できるといいな

Frontier 法で設計すべきこと

完全マッチング

- 格納する情報（mate配列）
- 辺 (u, v) に対する処理
  - mateの更新処理
    - 辺 (u, v) を採用する
    - 辺 (u, v) を採用しない
  - 0/1-node に行くかチェック
    - 採用する
    - 採用しない
  - フロンティアから外れる頂点の処理
  - 節点の共有の処理

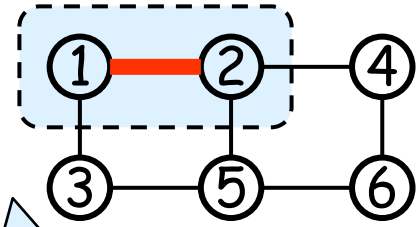
0/1：マッチして
いない/いる

u か v に相手がいる
→ 0-node
すべての頂点がマッチ
→ 1-node

ある頂点に
相手がいない
→ 0-node

まだ相手がいない
頂点の集合が一致
→ 今後の処理が同じ

完全マッチング



フロンティア
(処理の前線)

12

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

mate[ ]
マッチ
しているか

色の付いた
部分しか
使わない

辺 (1, 2) を
採用しない

mate[ ] は
変化しない

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

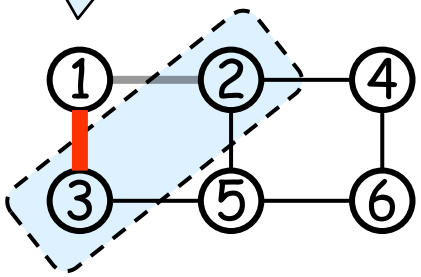
辺 (1, 2) を
採用する

1, 2 が
マッチする

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

各所で何をするかは、
後で順に説明します…

完全マッ



フロンティアから外れる

3 がフロンティアに入った

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

12

13

13

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

辺 (1, 3) を採用しない

フロンティアから 1 が外れた

mate[1] = 0
完全マッチングが作れない

0

辺 (1, 3) を採用する

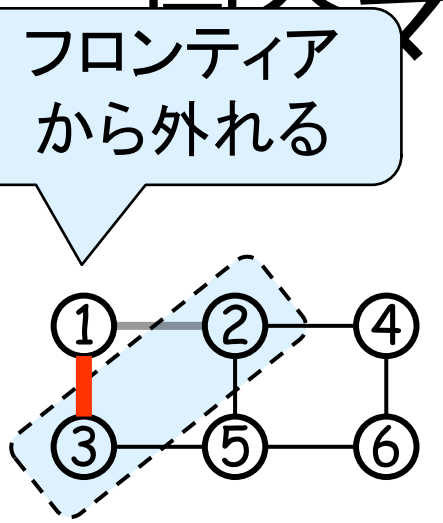
1	2	3	4	5	6
1	0	1	0	0	0

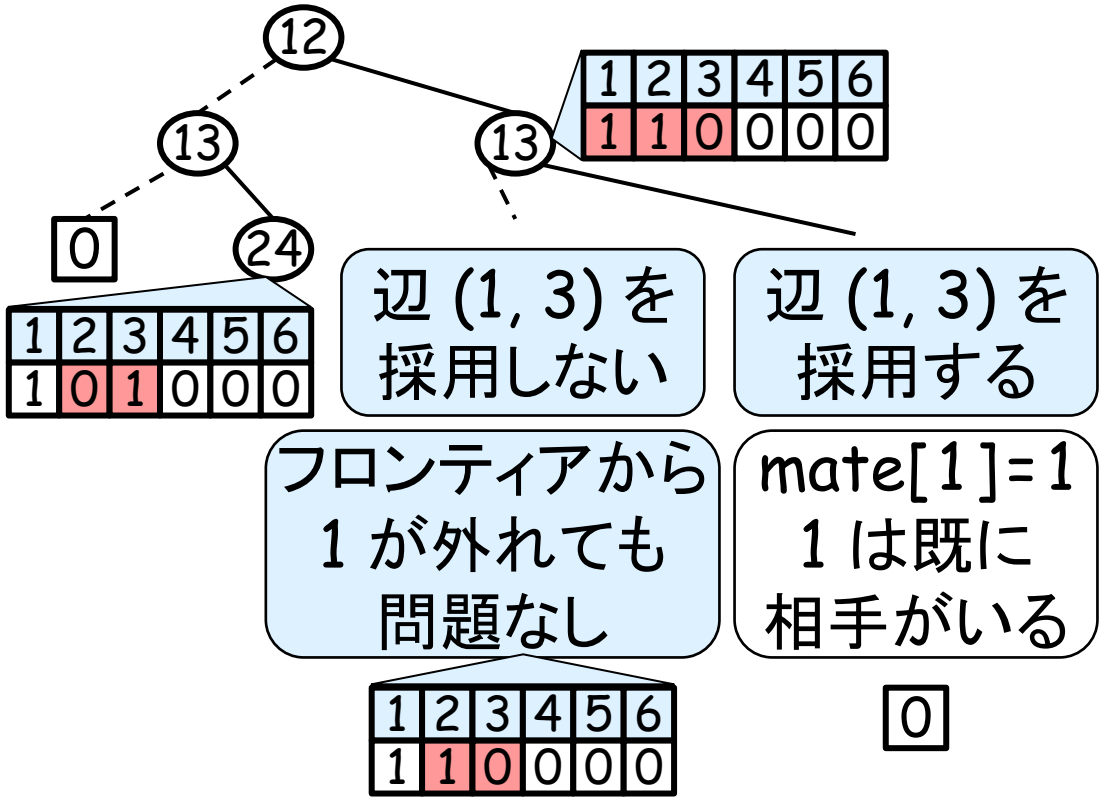
フロンティアから 1 が外れても問題なし

1 については解決なので、1 は外す

- フロンティアから外れる頂点に相手がいなければ、0-node

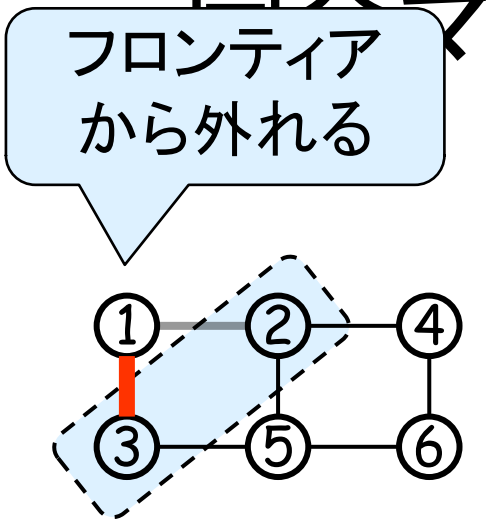
完全マッチング

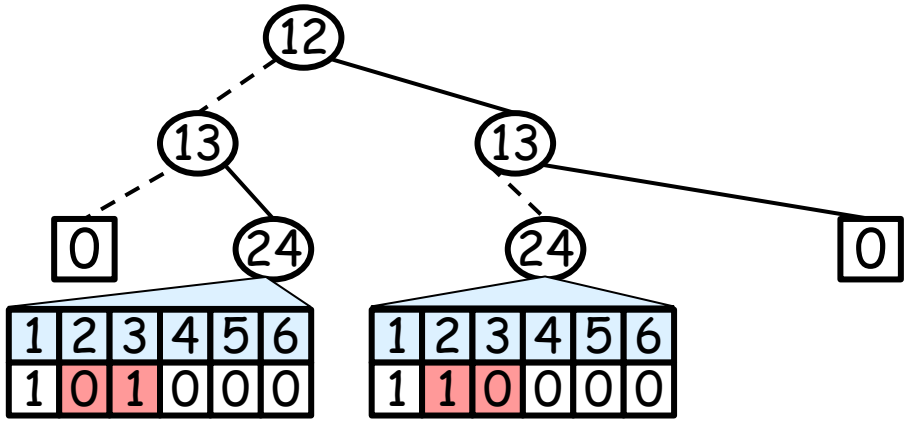




- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node

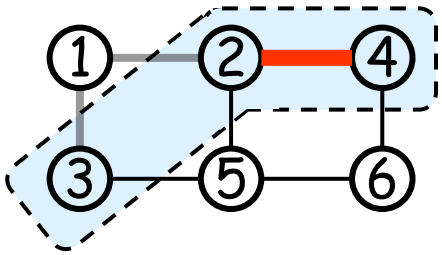
完全マッチング

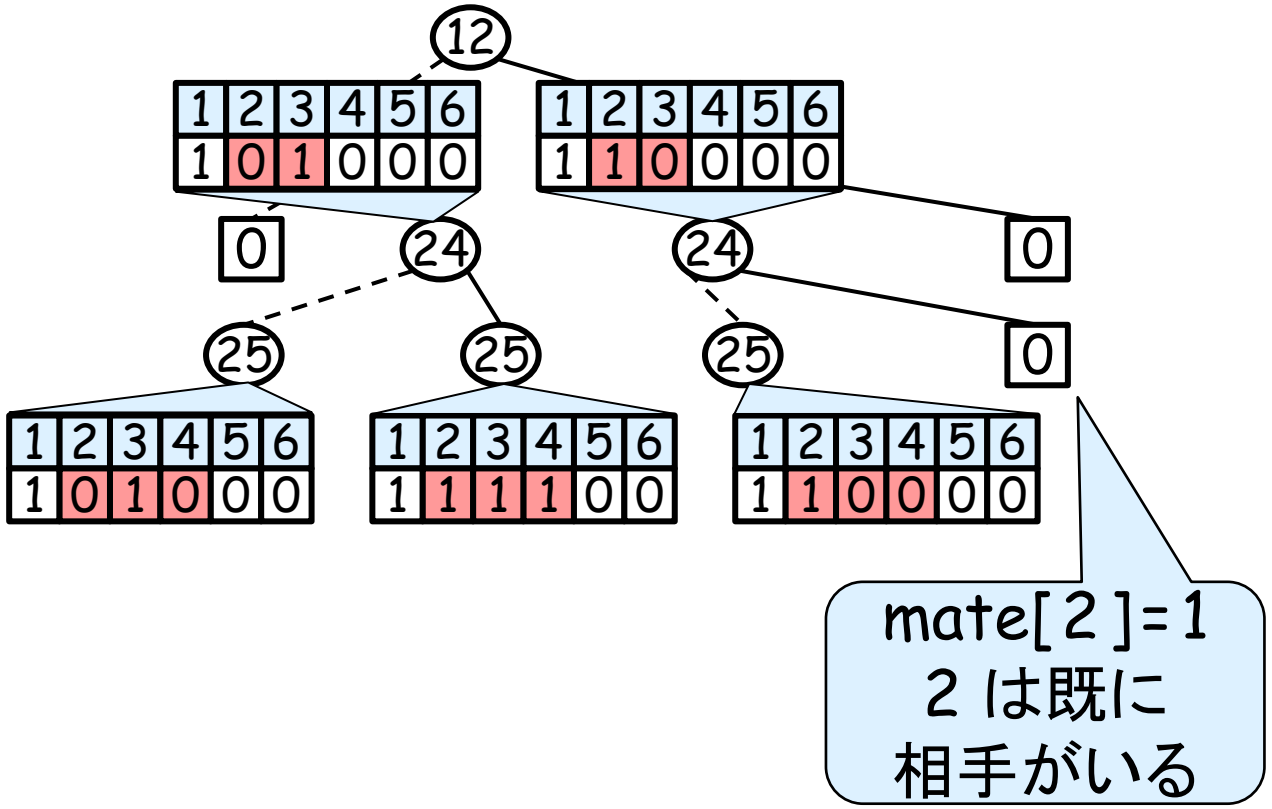




- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node

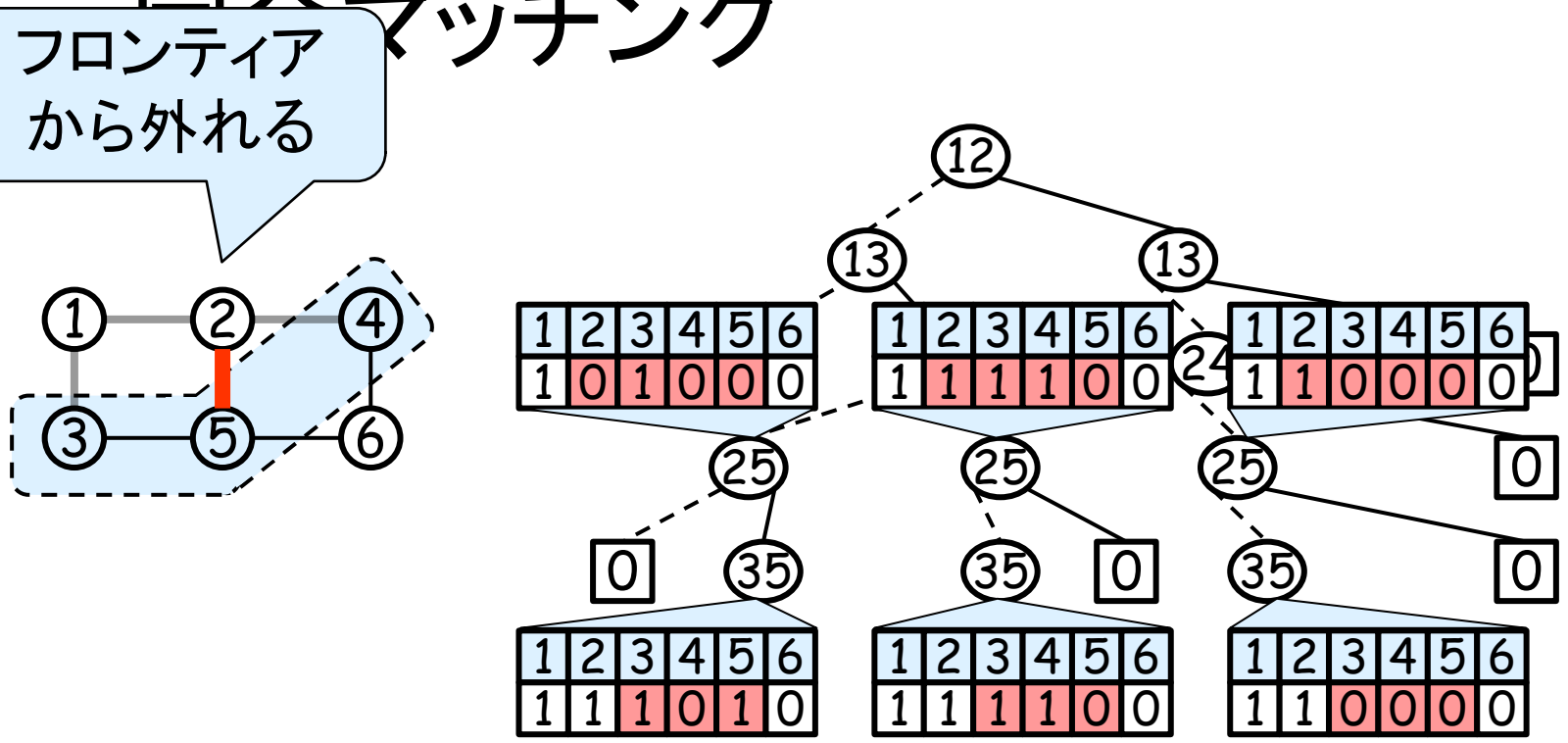
完全マッチング





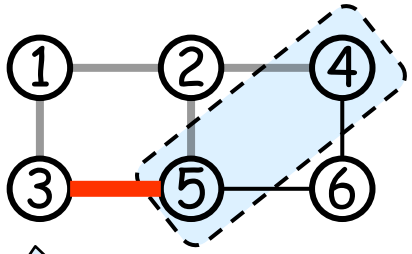
- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node

完全マッチング

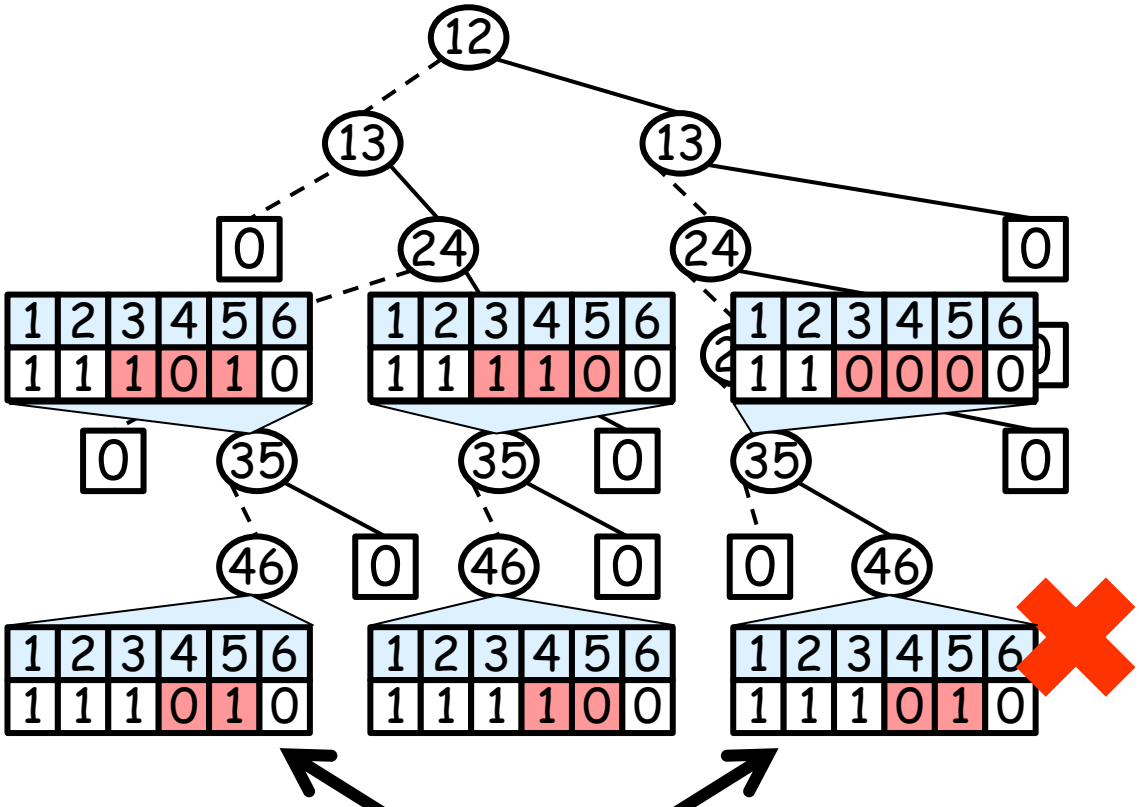


- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node

完全マッチング



フロンティア
から外れる



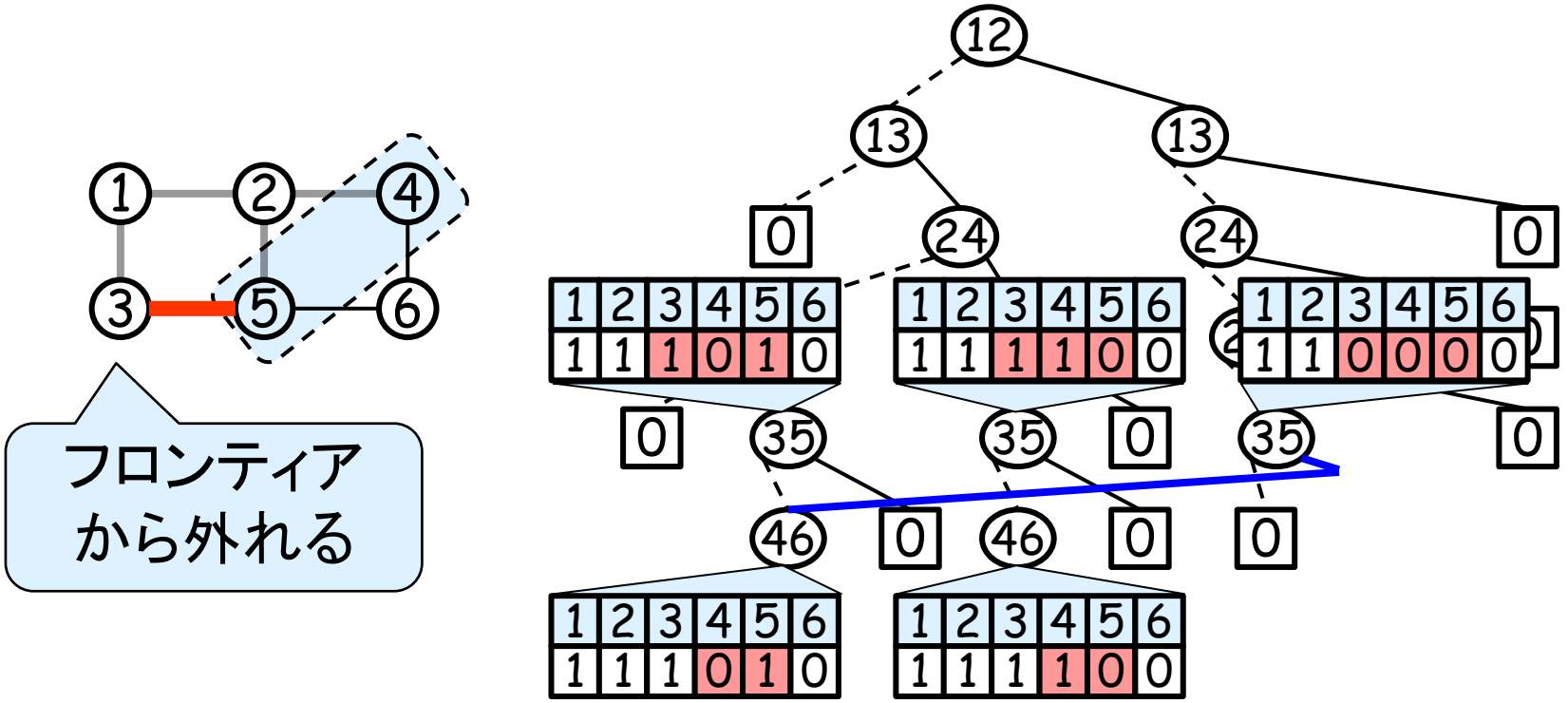
- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node

どちらも
mate[4] = 0

相手がいない人が一致
(過去は関係ない)

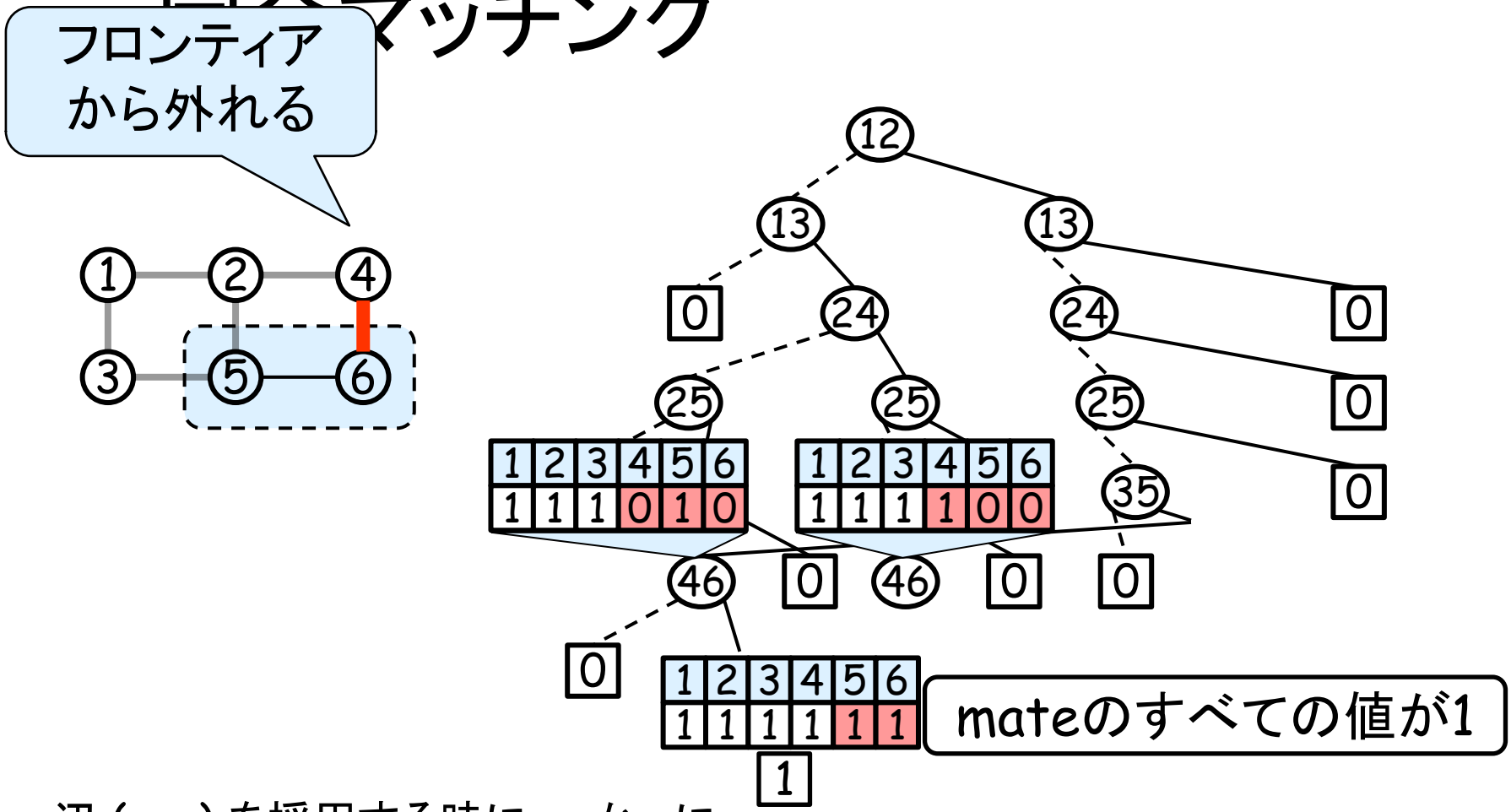
ZDD で同じ節点にまとめる

完全マッチング



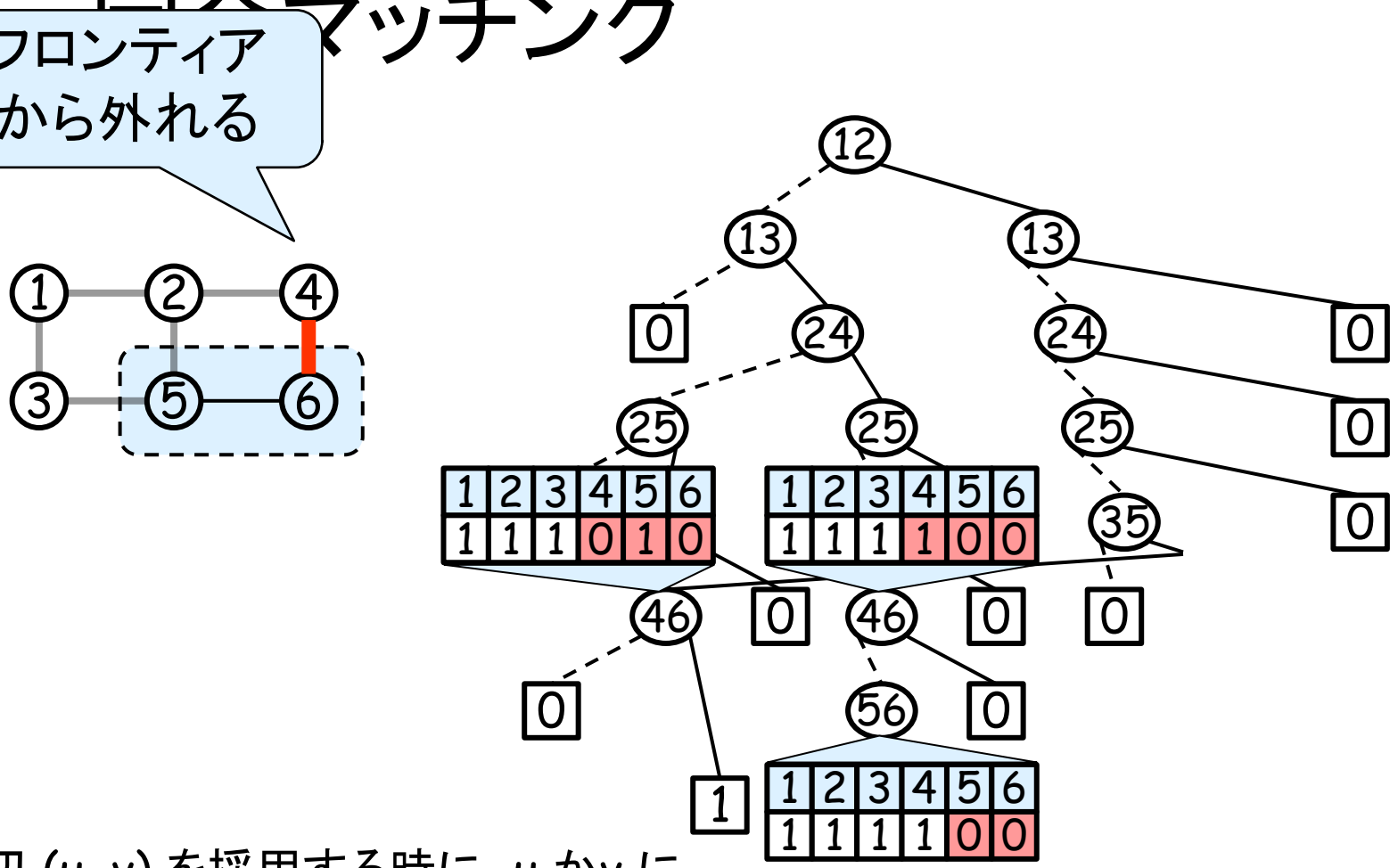
- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

完全マッチング



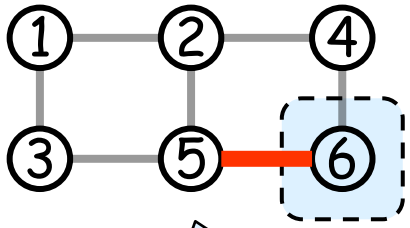
- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

完全マッチング

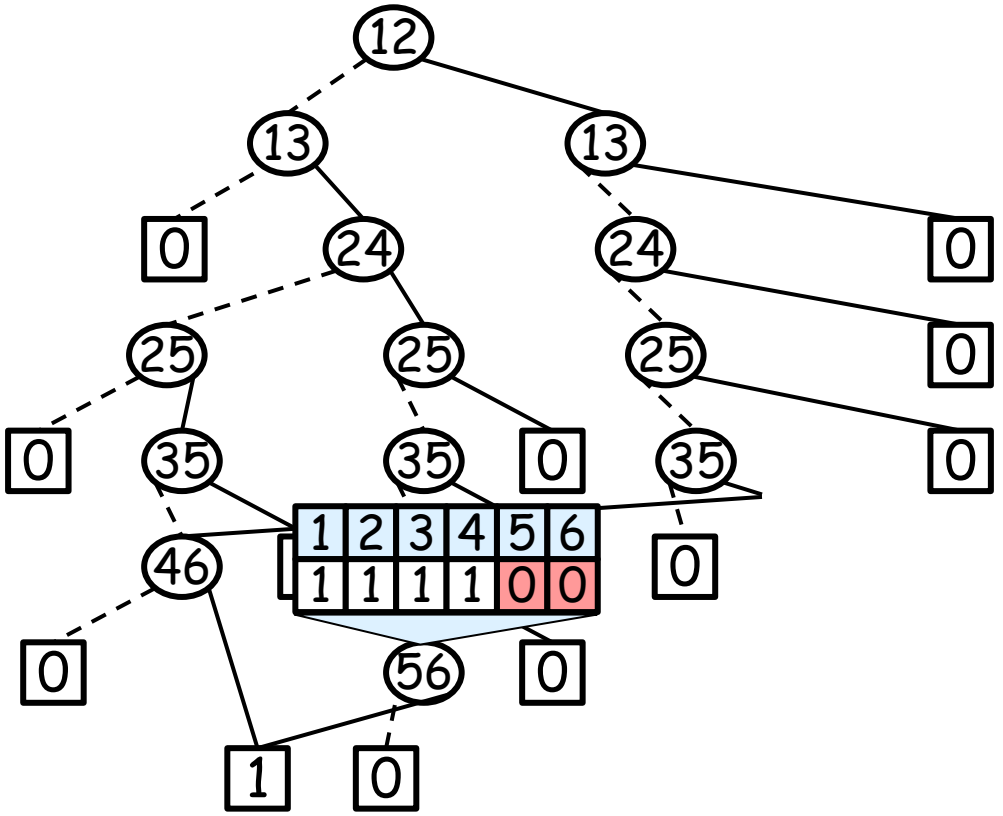


- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- mateのすべての値が1なら、1-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

完全マッチング

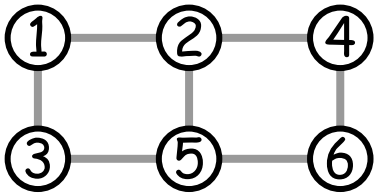


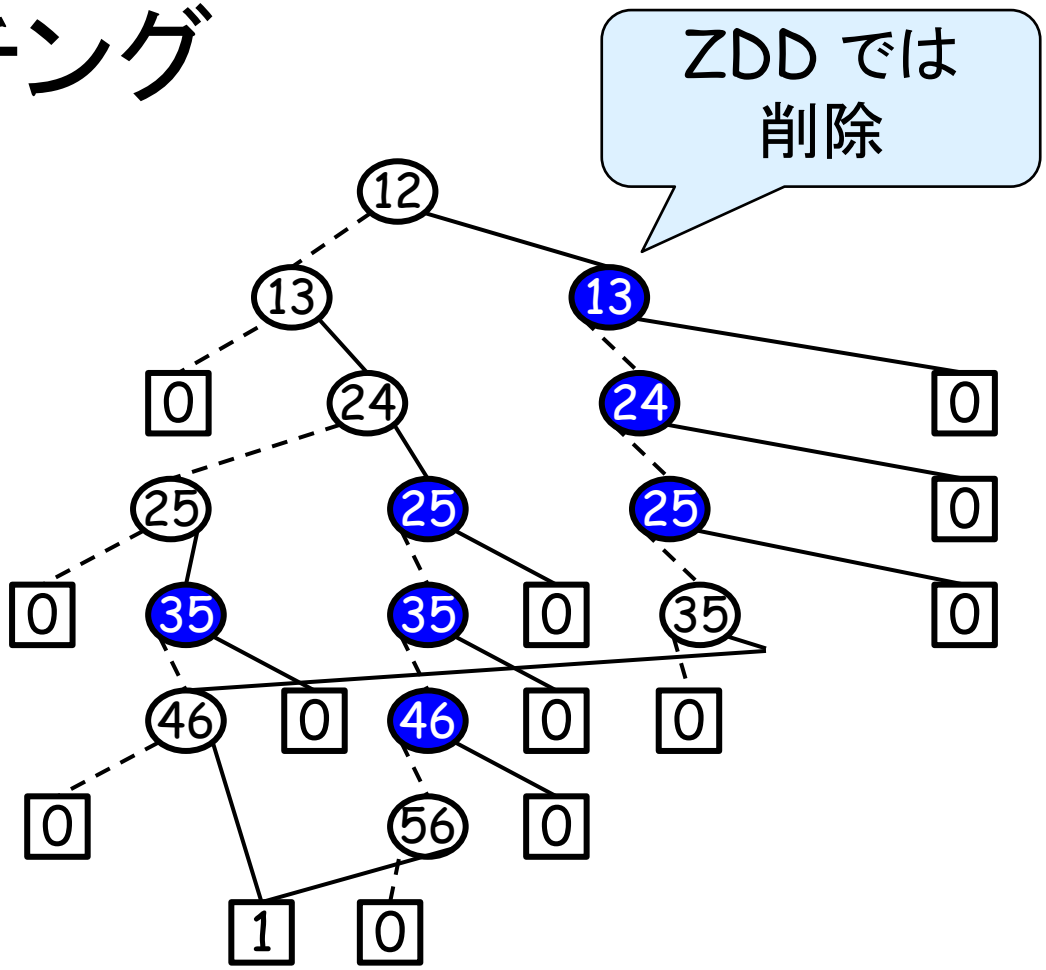
フロンティア
から外れる



- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- mateのすべての値が1なら、1-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

完全マッチング

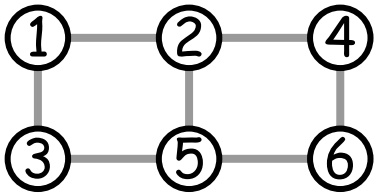


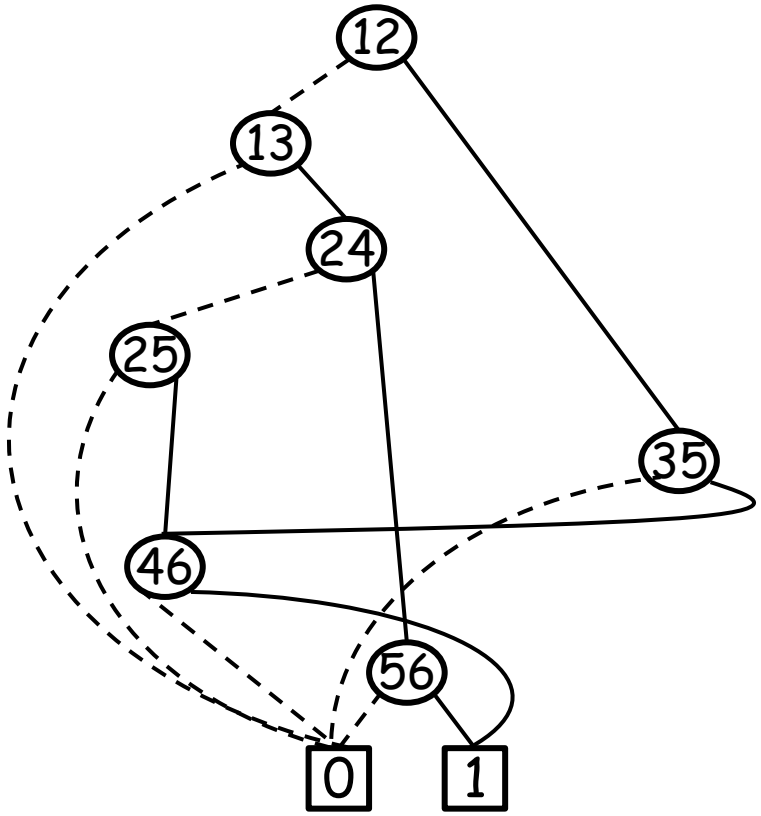


- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- mateのすべての値が1なら、1-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

完全マッチング

計算時間
(入力の辺の数) x (ZDD サイズ)
x (各節点での処理時間)





- 辺 (u, v) を採用する時に、u かv に相手がいれば、0-node
- フロンティアから外れる頂点に相手がいなければ、0-node
- mateのすべての値が1なら、1-node
- 相手がいない頂点が一致するならば、ZDD の節点を共有する

Frontier 法で設計すべきこと

完全マッチング

- 格納する情報（mate配列）
- 辺 (u, v) に対して
  - mateの更新処理
    - 辺 (u, v) を採用する
    - 辺 (u, v) をしない
  - 0/1-node に行くかチェック
    - 採用する
    - 採用しない
  - フロンティアから外れる頂点の処理
  - 節点の共有の処理

0/1：マッチしていない/いる

u か v に相手がいる
→ 0-node
すべての頂点がマッチ
→ 1-node

ある頂点に
相手がいない
→ 0-node

まだ相手がいない
頂点の集合が一致
→ 今後の処理が同じ

Frontier 法で設計すべきこと

s-t パス

- 格納する情報（mate配列）
- 辺 (u, v) に対して
  - mateの更新処理
    - 辺 (u, v) を採用する
    - 辺 (u, v) をしない
  - 0/1-node に行くかチェック
    - 採用する
    - 採用しない
  - フロンティアから外れる頂点の処理
  - 節点の共有の処理

$$mate[i] = \begin{cases} j & \text{頂点} i \text{ と} j \text{ を端点とするパスが存在} \\ i & \text{頂点} i \text{ に接続する辺を未使用} \\ 0 & \text{頂点} i \text{ はパスの内点} \end{cases}$$

余計な辺→ 0-node
s-t パス→ 1-node

パスの作りかけで頂点を放棄
→ 0-node

フロンティア上の状況が同じ
→ 今後の処理が同じ

Frontier 法で設計すべきこと

ハミルトンパス

- 格納する情報（mate配列）
- 辺 (u, v) に対して
  - mateの更新処理
    - 辺 (u, v) を採用する
    - 辺 (u, v) をしない
  - 0/1-node に行くかチェック
    - 採用する
    - 採用しない
  - フロンティアから外れる頂点の処理
  - 節点の共有の処理

$$mate[i] = \begin{cases} j & \text{頂点} i \text{ と} j \text{ を端点とするパスが存在} \\ i & \text{頂点} i \text{ に接続する辺を未使用} \\ 0 & \text{頂点} i \text{ はパスの内点} \end{cases}$$

余計な辺→ 0-node
s-t パス→ 1-node

パスの作りかけで
頂点を放棄/**次数0の頂点が存在**
→ 0-node

フロンティア上の状況が同じ
→ 今後の処理が同じ

Frontier 法で設計すべきこと

全域木

- 格納する情報（mate配列）
- 辺 (u, v) に対して
  - mateの更新処理
    - 辺 (u, v) を採用する
    - 辺 (u, v) をしない
  - 0/1-node に行くかチェック
    - 採用する
    - 採用しない
  - フロンティアから外れる頂点の処理
  - 節点の共有の処理

コンポーネント番号

(u, v)が同じコンポーネント→ 0-node
全域木→ 1-node

次数0の頂点が存在→ 0-node

フロンティア上の状況が同じ
→ 今後の処理が同じ

まとめ

- **フロンティア法**
  - **BDD/ZDD をトップダウンで直接（ラフに）**作る
    - パーティションの列挙 [Hardy ら, 2007]
      パスの列挙 [Knuth 2009] の 一般化
    - マッチング、全域木、（クリーク、）などなど
      適用範囲が日々拡大中
  - むちゃくちゃ速い *!!*

フロンティア上のmateだけで、同値なものとしてまとめることができるもの