DD ライブラリ入門

川原 純

2022/7/29 版

本資料の目的

- SAPPOROBDD や TdZdd など、DD ライブラリの解説
  - TdZdd については作成中
- ZDD の基本的な知識は既知であると仮定します。
- その他の情報
  - 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: DD のアルゴリズムを解説
  - DD ライブラリ入門: 本資料
  - DD の再帰演算のカタログ: DD の再帰演算の一覧

DDライブラリ概要

- SAPPOROBDD（湊 真一先生 作）
  - DD処理系ライブラリ、ボトムアップ構築
- SAPPOROBDD helper（川原 作）
  - SAPPOROBDD の補助ライブラリ
- TdZdd（岩下 洋哲氏）
  - DDトップダウン構築ライブラリ

いずれも C++ 言語のライブラリ
MIT ライセンス（オープンソース）

graphillion は、ZDD を意識することなく部分グラフを圧縮列挙するためのライブラリであり、本資料では使わない。

DDライブラリ導入方法

- https://github.com/junkawahara/dd_package
  に書かれている通りに導入する。
  （本資料では詳しくは省略する）
- コマンドラインで、wget と git が使える環境があれば、比較的簡単に導入できる。
  - wget を使わず、ブラウザで downloader.sh をダウンロードしてもよい。
  - git は必須
- ↑のページに従って導入すると、すぐにコードを書き始めることができる。

SAPPOROBDD 使用上の注意

- SAPPOROBDD は32ビット版と64ビット版が存在する。特別な理由がない限り、64ビット版を使用するのがよい。64ビット版はコンパイラ（g++ など）でのコンパイル時に、-DB_64 オプションを指定し、B_64 マクロを定義する。また、ライブラリは BDD64.a を使用する。
  - dd_package の Makefile にはこの設定が既にされている
- SAPPOROBDD は（現在のバージョンでは）名前空間の中にはない。 SAPPOROBDD helper は sbddh、TdZdd は tdzdd 名前空間の中にある。プログラムの初めの方に以下を記述しておく。

```
using namespace tdzdd;
using namespace sbddh;
```

SAPPOROBDD を早速使ってみる(1)

- SAPPOROBDD では、集合族の各集合の要素を、1以上の整数で表す。これを変数番号と呼ぶ。
  - 集合族の例: { {1, 2}, {2, 3}, {3} }

必ず最初に呼び出す必要がある。

最初に確保するメモリ量

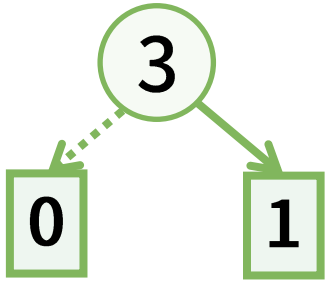
最大確保メモリ量
（十分大きくしておけばよい）

```
BDD_Init(1024, 1000000000);

for (int i = 0; i < 10; ++i) {
    BDD_NewVar();
}
```

使用する変数番号の最大値の回数だけ
BDD_NewVar() を呼び出す
この例では10である。
{ {1, 9, 10}, {3,10} } は OK だが、
{ {1, 10, 11}, {11} } は NG

SAPPOROBDD を早速使ってみる(2)

- シングルトンZDD（要素1つの集合1つからなるZDD）



{{3}}

ZDD は ZBDD 型（クラス）で表される。

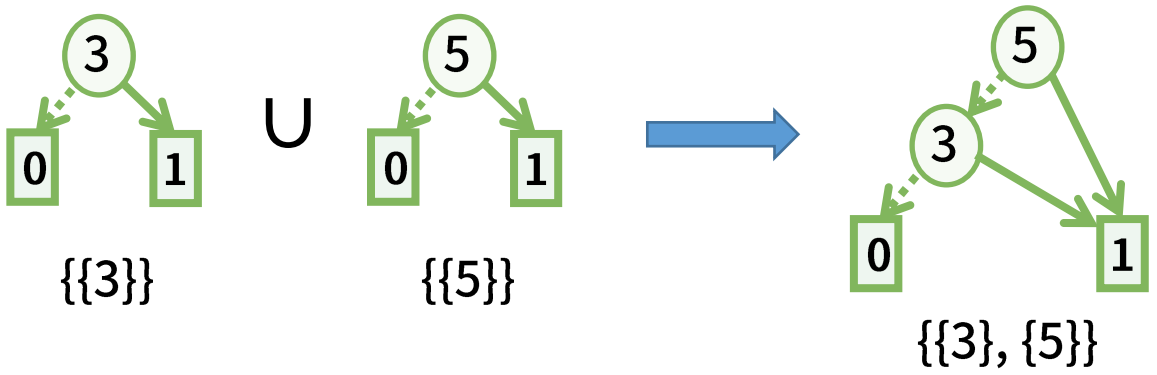
```
ZBDD z = getSingleton(3);

std::cout << ZStr(z) << std::endl;
```

シングルトンZDDを構築する関数（SAPPOROBDD helper の関数）

ZDD z が表す集合族を文字列として返す関数（SAPPOROBDD helper の関数）。
この場合は {3} が出力される
（これは {{3}} の意味である。外側の括弧は出力されない。）

SAPPOROBDD を早速使ってみる(3)

- Union : 2つの集合族の和集合を計算




```
ZBDD za = getSingleton(3);
ZBDD zb = getSingleton(5);

ZBDD zc = za + zb;

std::cout << ZStr(zc) << ":" << zc.Card() << std::endl;
```

union

{5}, {3} が出力される

zc の集合の個数 2 を得る

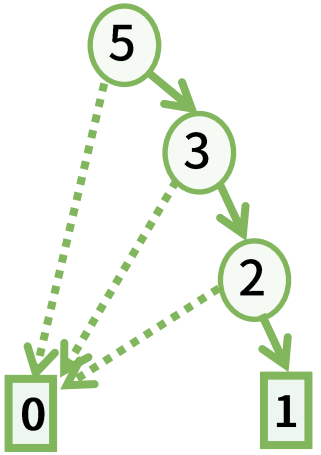
注: 変数番号は、（本資料に掲載されている変数宣言法を用いている限り）終端に近い側が小さくなる

SAPPOROBDD を早速使ってみる(4)

- 任意の集合1つからなる集合族のZDD

```
std::vector<int> v;
v.push_back(2); v.push_back(3); v.push_back(5);
ZBDD za = getSingleSet(v);

std::cout << ZStr(za) << std::endl;
```

"{5, 3, 2}"



{{2, 3, 5}}

vector から、その vector の中身を要素とする集合1つからなる
集合族の ZDD を返す（SAPPOROBDD helper の関数）

SAPPOROBDD を早速使ってみる(5)

- 任意の集合族を表す ZDD

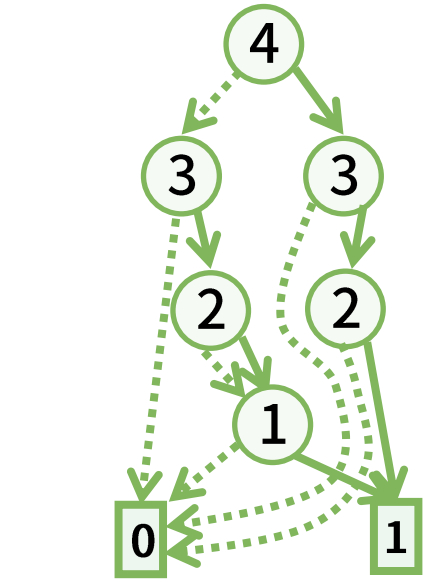
例: 集合族 { {1, 2, 3}, {1, 3}, {2, 3, 4} } を表す ZDD

第1引数は　第2引数以降に
要素数　要素を並べる

```
ZBDD z1 = getSingleSet(3, 1, 2, 3);
ZBDD z2 = getSingleSet(2, 1, 3);
ZBDD z3 = getSingleSet(3, 2, 3, 4);

ZBDD z = z1 + z2 + z3;

std::cout << ZStr(z) << std::endl;
```

union

```
z1 は {{1, 2, 3}}
z2 は {{1, 3}}
z3 は {{2, 3, 4}}
```



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

SAPPOROBDD を早速使ってみる(5)

- 集合族が vector の vector で表されている場合に ZDD を構築する

```
std::vector<int> v1;
v1.push_back(1); v1.push_back(2); v1.push_back(3);
std::vector<int> v2;
v2.push_back(1); v2.push_back(3);
std::vector<int> v3;
v3.push_back(2); v3.push_back(3); v3.push_back(4);
std::vector<std::vector<int> > f;
f.push_back(v1); f.push_back(v2); f.push_back(v3);

ZBDD z(0); // z は空集合を表す ZDD
for (int i = 0; i < (int)f.size(); ++i) {
    z += getSingleSet(f[i]);
}
std::cout << ZStr(z) << std::endl;
```

v1 は {1, 2, 3}

v2 は {1, 3}

v3 は {2, 3, 4}

f は vector の vector

z と右辺の union をとって z に代入

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

ZDD のコピーとZDD 同士の比較演算

- ZBDD クラスのコピーや比較は定数時間で可能

```
ZBDD z = getSingleSet(3, 1, 2, 3);  // {{1, 2, 3}}

// {{1, 2, 3}} と {{1, 2, 3}} の 共通部分、すなわち {{1, 2, 3}}
ZBDD y = z & z;
ZBDD x;
x = z; // ZDD のコピー。定数時間でコピー可能

// 2 つの ZDD の比較。同じ集合族を表していたら true 。
// 定数時間で実行可能。 "true" が出力される
if (x == y) {
    std::cout << "true" << std::endl;
} else {
    std::cout << "false" << std::endl;
}
z += getSingleSet(1, 2); // z は変更されるが、x は変更されない
```

SAPPOROBDD の終端

- 終端を表す ZDD

0 終端　{} 空集合を表す

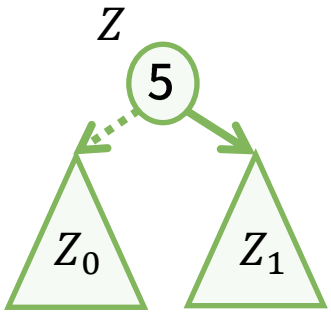
1 終端　{{}} 空集合1つだけからなる集合を表す

```
ZBDD za = ZBDD(0);
ZBDD zb = ZBDD(1);

// 例えば、ZBDD の配列の各要素の和を求める際の初期値に使う
ZBDD z_sum = ZBDD(0);
for (int i = 0; i < 10; ++i) {
    z_sum += z[i];  // z[i] は ZBDD
}

// ZBDD が終端ノードであるかの判定を行う際にも使う
ZBDD f = ...;  // 何らかの方法で f を作成
... // f を計算
if (f == ZBDD(0) || f == ZBDD(1)) { // f が終端ノードであるなら
    ...
}
```

SAPPOROBDD の機能(1)

```
ZBDD za = ...; // 適当に構築
ZBDD zb = ...;

ZBDD zc = za & zb; // za と zb の共通部分集合

ZBDD zd = zc.OffSet(1); // 変数 1 を含まない集合を抽出
// zc.OffSet(1) を呼び出しにより、zc は変化しない。
// 左辺に代入を忘れないこと

ZBDD ze = zc.OnSet(1); // 変数 1 を含む集合を抽出

ZBDD zf = zc.Change(1); // 変数 1 を含まない集合それぞれ
に 1 を追加し、含む集合それぞれから 1 を削除する。

ZBDD zg = za.Restrict(zb); // "Restrict 演算"
```

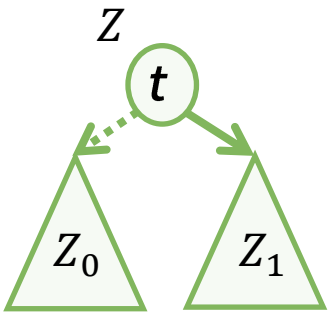
ZDD 構造の扱い方




```
ZBDD z = ...; // 適当に構築

int t = z.Top(); // 根ノードの変数番号を取得

ZBDD z0 = z.OffSet(t); // 変数 t を含まない集合を抽出、すなわち z の0枝側の ZDD を取得

ZBDD z1 = z.OnSet0(t); // 変数 t を含む集合から t を除去した集合を抽出、すなわち z の1枝側の ZDD を取得
（OnSet の後ろの ‘0’ を忘れないこと）
```

SAPPOROBDD での再帰演算（キャッシュ無）

ZDD の再帰構造を利用した関数の定義の例。
ZDD が表現する集合の個数を計算。
Z0 の個数と Z1 の個数の和が Z の個数になるので、
再帰で計算。




```
int countElems(ZBDD z) {
    if (z == ZBDD(0)) { // 0 終端
        return 0;
    } else if (z == ZBDD(1)) { // 1 終端
        return 1; // {{}} の要素の個数は 1
    }
    int t = z.Top(); // 根ノードの変数番号を取得
    ZBDD z0 = z.OffSet(t); // z の0枝側の ZDD を取得
    ZBDD z1 = z.OnSet0(t); // z の1枝側の ZDD を取得
    // 0枝側と1枝側それぞれを再帰呼び出しし、結果を足し算
    int value = countElems(z0) + countElems(z1);
    return value;
}
```

SAPPOROBDD での再帰演算（キャッシュ有）

同じ計算を2回目以降に呼び出すときは、キャッシュを
参照すると高速化できる

```
std::map<bddp, int> cache; // キャッシュ（グローバル変数）

int countElems(ZBDD z) {
    if (z == ZBDD(0)) return 0;
    else if (z == ZBDD(1)) return 1;

    bddp zid = z.GetID(); // ノードIDを取得
    if (cache.count(zid) > 0) { // キャッシュにヒット
        return cache[zid]; // キャッシュされた結果を返す
    }

    int t = z.Top();
    ZBDD z0 = z.OffSet(t);
    ZBDD z1 = z.OnSet0(t);
    int value = countElems(z0) + countElems(z1);
    cache[zid] = value; // キャッシュに計算結果を格納
    return value;
}
```

SAPPOROBDD helper の機能(1)

```
std::vector<int> v;
v.push_back(2); v.push_back(3); v.push_back(5);
ZBDD za = getPowerSet(v); // v の「べき集合」ZDD を構築

ZBDD zb = ... // 適当に構築
if (isMemberZ(zb, v)) { // v（この例だと {2, 3, 5}）が
                        // zb が表す集合族に含まれるなら true を返す。
    ...
}
```

SAPPOROBDD helper の機能(2)

- ZDD の出力

```
3 2 1
3 2
2 1
```
のように出力

```
ZBDD z = ...; // 何らかの方法で構築
// z の表す集合族を出力。第3引数は集合の区切り。
// 第4引数は集合の要素の区切り
printZBDDElements(std::cout, z, "¥n", " ");

// graphviz（グラフ描画ソフトウェア）の dot 形式で出力。
// ZDD の形を視覚的に画像として出力できる。
writeZBDDForGraphviz(std::cout, z);
```

以下のコマンドで画像を生成する

```
dot -Tpng -o zdd.png < dotfile.txt
```

writeZBDDForGraphviz の出力

SAPPOROBDD helper の機能(3)

- 集合族の要素（集合）を巡回するイテレータ

集合族の各要素に処理をしたい場合に使用できる

```
ZBDD f = ...;  // {{1, 2}, {2, 3}} を表す集合族であるとする

// ElementIterator を使用するときは、ElementIteratorHolder を
// 作成する。コンストラクタに f を指定する。
ElementIteratorHolder eih(f);

// ElementIteratorHolder の begin メンバ関数により、イテレータを取得
ElementIterator itor = eih.begin();

// イテレータの使い方は C++ の STL と同じである。
// 以下の while 文で、f が表す集合族の各集合を巡行する。
while (itor != eih.end()) {
    // 1回目の実行で s は {1, 2} となり、
    // 2回目の実行で s は {2, 3} となる。
    std::set<bddvar> s = *itor;
    ++itor;
}
```

TdZdd の使い方

- TdZdd の使い方解説については作成中
- ZDDと列挙問題―最新の技法とプログラミングツールを参照