1 章 章末問題解答

1. 頂点数 n の木は $n-1$ 本の辺を持つことを数学的帰納法で証明する．まず頂点数 1 の木は辺を持たないため，明らかに成り立つ，頂点数 $k(\leq n-1)$ の木が $k-1$ 本の辺を持つと仮定する．頂点数 n の木 T から一つの辺を削除したグラフは，2 つの木 T_1 と T_2 からなる森である．T_1 と T_2 の頂点数をそれぞれ n_1 と n_2 とすると，帰納法の仮定より，T_1 と T_2 の辺数はそれぞれ n_1-1 と n_2-1 である．したがって，元の木 T の辺数は，$(n_1-1)+(n_2-1)+1=(n_1+n_2)-1=n-1$ であるため，$k=n$ のときも成り立つ．したがって，数学的帰納法より題意が成り立つ．
2. 各頂点の次数を順に足しあげていくことを考える．これは，頂点において接続している辺を一つずつ数えてカウントアップしていき，それを終えると次の頂点に移って同様にカウントアップを続けていくことを意味する．その際，一つの辺はその両端点のそれぞれで数えられるため，一つの辺によってカウンタ値は 2 だけアップされる．したがって，このカウントアップを終えたときは，辺の総数の 2 倍のカウンタ値になっている．つまり，次数の和は辺数の 2 倍である．
3. 次数 d_i の頂点を v_i とする．左辺は，頂点 v_1 から v_k までの次数の和を意味する．$S=\{v_1,v_2,...,v_k\}$ の頂点に接続している辺集合は，両端点が S の頂点であるような辺の集合 E_1 と，片方の端点が S の頂点で，他方が S 以外の頂点であるような辺の集合 E_2 に分割できる．したがって，次数の和は，E_1 に属する辺のみを数え上げた値と，E_2 に属する辺のみを数え上げた値の合計になっている．前者は高々 $k(k-1)$ である．なぜなら，S は k 個の頂点集合なので，$|E_1|\leq k(k-1)/2$（k 個の頂点からなる完全グラフの辺数が上限）だからである．S 以外の頂点 $v_j(k+1\leq j\leq n)$ と S との間の辺の数は高々 $\min\{d_j,k\}$ なので，E_2 に属する辺のみに関

する次数の総和は，$\sum_{j=k+1}^{n} \min\{d_j, k\}$ を超えることはない．以上から，左辺は，$k(k-1) + \sum_{j=k+1}^{n} \min\{d_j, k\}$ 以下であることがわかる．

この結果からも，グラフの各頂点の次数は任意の値をとることはできず，とりうる範囲が限定されることがわかる．

4. 略．

5. (a) $O(n^2)$
   (b) $O(n^{\log n})$
   (c) $O(n^2 \cdot 2^n / \log n)$

6. 「大きさ p のナップザックに k 個目までの要素を入れる場合の価格の合計の最大値」に関して最適性原理が成り立つことに着目すればよい．

2章 章末問題解答

1. 辺重みが全て 1 の場合は，経路長は経路に含まれる辺の本数に一致する．幅優先探索によって得られる探索木において，頂点から始点までの唯一の経路の経路長より短い経路があるとすると，その経路に沿って先に訪問されているはずであるので矛盾する．したがって，辺重みが全て 1 の場合は，幅優先探索による探索木は最短路木になっている．
2. 図 1 に最短路木のみ示す．

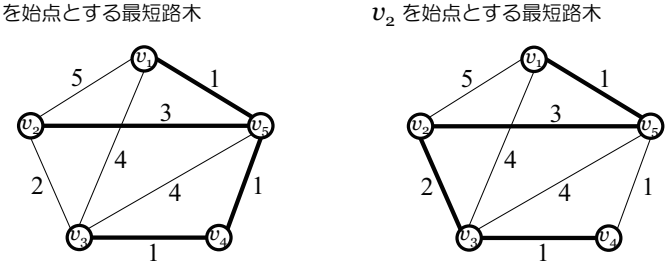
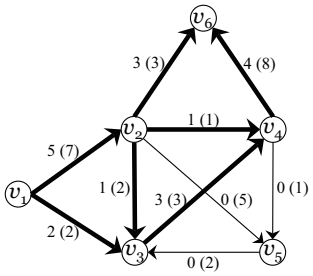


図 1 ネットワーク N_1 の最短路木

3. 略
4. 図 2 に最大フローのみ示す．



※ a (b) は，その辺を流れるフローの大きさが a
辺容量が b であることを意味する

図 2 ネットワーク N_2 における最大フロー

３章 章末問題解答

1. グラフ G_1 における頂点 v, w 間の局所辺連結度は 4，局所点連結度は 4 である．グラフの辺連結度は 3，点連結度も 3 である．
2. グラフ G_2 における頂点 v_1 と領域 W の間の NA 辺連結度は 3，NA 点連結度は 2 である．頂点 v_2 と領域 W の間の NA 辺連結度は 3，NA 点連結度は 3 である．
3. 略
4. (a) 図 3 に MA 順序と辺の順位の一例を挙げる．

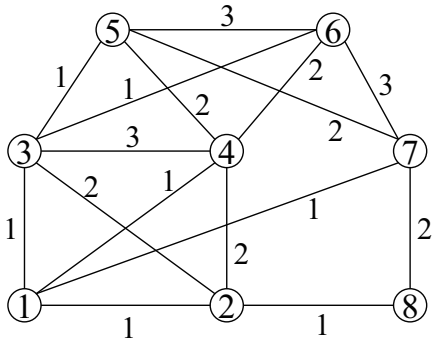
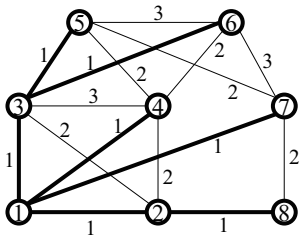


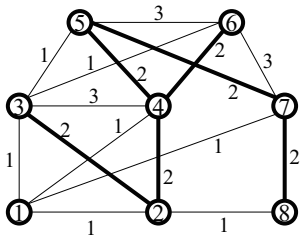
図 3 G_3 の MA 順序と辺の順位

(b) G_3 における 2 辺連結性を保存する全域部分グラフの例を図 4 に挙げる．

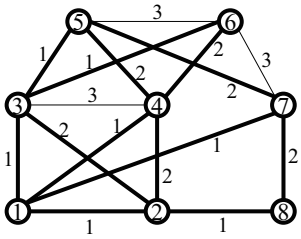
(c) 本文では 3 辺連結化のためのアルゴリズムは紹介していないが，問題例の程度のグラフサイズであれば，３辺連結グラフの定義を理解していれば容易に見つけられるだろう（図 5）．



(a) 部分グラフ F_1



(b) 部分グラフ F_2



(c) 部分グラフ G'

図 4 G_3 における 2 辺連結性を保存する全域部分グラフ

5. 本文では 2 本の辺削除の場合の保護辺決定アルゴリズムについては紹介していないが，頂点保護問題のところで紹介した 2 頂点削除の場合と同様の考え方で扱うことができる．まず，どのような一本の辺削除に対しても条件を満たす．次に，2 本の同時辺削除に対して，条件を満たさなくなるような辺の組を列挙する（辺番号は図 6 を参照）．2 本の辺の組 $\{e_3, e_4\}, \{e_5, e_6\}, \{e_5, e_8\}, \{e_6, e_7\}, \{e_6, e_8\}, \{e_7, e_8\}, \{e_9, e_{10}\}$ のそれぞれは，その 2 辺が削除されるとサーバに対応する頂点を含む連結成分のサイズが 8 未満になってしまって条件を満たさないため，すくなくともいずれか一本の辺は保護されていなければならない．そのような最小本数の保護辺集合を求めることは，辺を頂点とするグラフ（図 7）において最小サイズの頂点被覆を求めることと等価である．最小頂点被覆問題は一般には NP 困難問題ではあるが，この問題例の場合はグラフのサイズが小さいのですぐに最適解がわかり，$\{e_3, e_5, e_6, e_7\}$ である．これが

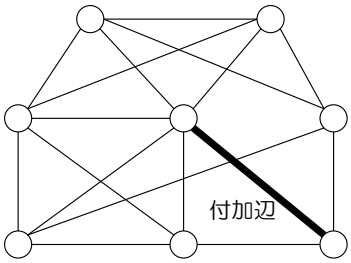


図 5 G_3 を 3 辺連結化する付加辺

最小本数の保護辺集合である．なお，2 本の辺削除の場合に対する多項式時間アルゴリズムも存在するが，ここで述べた考え方とは異なる．関心のある読者は参考文献 52 を参照してほしい．

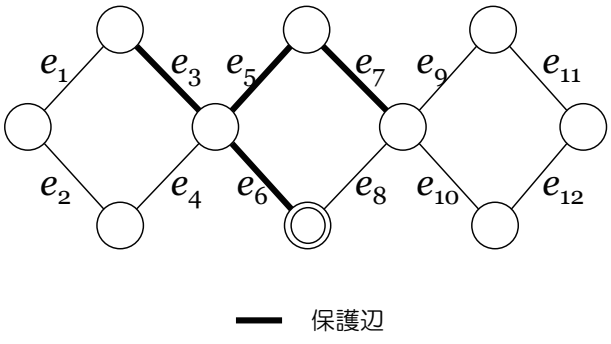


図 6 G_4 における保護辺集合

6. 本文で紹介した頂点保護問題とは少し異なるが，同様の考え方で扱うことができる．一つの頂点削除に対して連結であるような保護頂点集合は，$\{v_7, v_{10}\}$ である．2 個の頂点削除に対して連結であるような保護頂点集合は，頂点被覆問題に帰着して考えればよい．一般には NP 困難問題ではあるが，この問題例の場合はグラフのサイズが小さいのですぐに最適解がわかり，$\{v_2, v_3, v_5\}$ である．あわせて，$\{v_2, v_3, v_5, v_7, v_{10}\}$ が最小個数の保護頂点集合である．

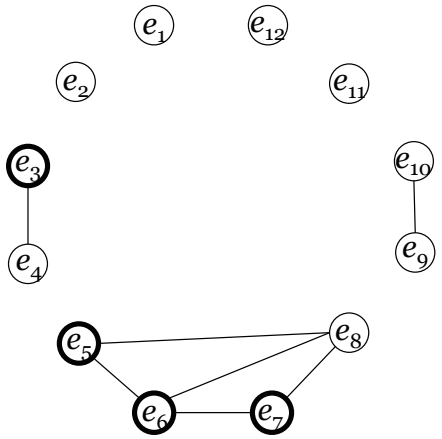


図 7　保護辺決定のための頂点被覆問題

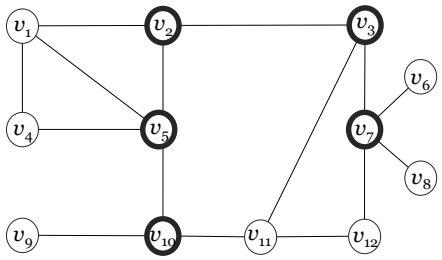


図 8　G_5 における保護頂点集合

4 章 章末問題解答

1. できる．もっとも右の x_3 節点から出る 0 枝を x_5 節点につなぎ直しても，BDD の意味は変わらず，同じ確率を得られる．
2. 本章で紹介したアルゴリズムは，リンク順序によらず適用でき，同じ確率を得られる．ただし，得られる BDD の形状や圧縮度は，リンク順序によって異なる．BDD の節点数を最小にするリンク順序を求める問題は NP 困難であり，経験的に幅優先探索による順序づけが用いられることが多い．
3. リンクを縮退するとき，そのリンクの両端ノードが同じ角括弧に属していると閉路が構成される．よって，その先の探索を終了し，終端節点 $\perp$ につなげばよい．たとえば，図 4.11 において，もっとも左の x_3 節点が表すネットワーク状態は $[v_2v_3]^*$ である．つまり，ノード v_2 と v_3 はすでに連結されている．ここで，リンク $e_3 = (v_2, v_3)$ を稼働に決定すると $(x_3 = 1)$，すでに連結されている v_2 と v_3 にさらにリンクを追加することで，閉路を構成することになる．このように，同じ角括弧に属するノードをさらにつながないように，BDD を構築していけばよい．このようにして得られる閉路を含まないサブグラフは「木の集合」となるため，一般に「森」と呼ばれる．
4. 連結対象ノード数を変更するのは簡単なので，次数の管理について述べる．ノードの次数が変化するのは，そのノードがフロンティアにあるときだけであるため，フロンティアのノードについてのみ次数を管理する．各ノードについて次数の初期値を 0 とし，隣接リンクの状態を稼働に決定したら，次数を $+1$ する．ノードがフロンティアから外れると，そのノードの次数はもう変化しない．よって，フロンティアから外れるときに次数が指定された値になっていなければ，その先の探索を終了し，終

端節点 $\perp$ につなぐ.

5 章 章末問題解答

1. 本文では，ハミング制約を表す BDD の節点数を $(N+1)(m-1)-1$ と述べたが，図 5.6b 左下のふたつの節点は分岐しないために省略でき，節点数は $(N+1)(m-2)$ となる．この $(N+1)(m-2)$ の BDD 節点を以下のルールに従ってつなぐと，ハミング制約を表す BDD になる (節点番号は図 5.6b と異なる)．なお，$a\%b$ は，a を b で割った余りとする．

$$s_i.x'_j \leftarrow \begin{cases} s_{i+N} & 1 \leqq i \leqq (N+1)(m-3) \\ \top & (N+1)(m-3) < i \end{cases} \tag{1}$$

$$s_i.((x'_j+1)\%2) \leftarrow \begin{cases} s_{i+1} & i\%(N+1) \neq 0 \\ \bot & i\%(N+1) = 0 \end{cases} \tag{2}$$

2. まとめてよい．これらの節点をまとめる前の BDD と，まとめた後の BDD は，いずれも同じ論理関数を表す．つまり，すべてのネットワーク状態 $\boldsymbol{x}$ に対し，同じ値 ($\top$ あるいは $\bot$) に対応付けられている．
3. ４つの節点，$s_{c,9} \wedge s_{h,11}$，$s_{c,9} \wedge s_{h,9}$，$s_{c,9} \wedge s_{h,10}$，$\top \wedge s_{h,11}$ が等価になる．
4. $s_{c,9} \wedge s_{h,11}$ と $s_{c,9} \wedge s_{h,9}, s_{c,9} \wedge s_{h,10}, \top \wedge s_{h,11}$ は，いずれも 0 枝が $\bot$，1 枝が前問でまとめた節点を指しており，ひとつにまとめられる．このようにして，下から等価な節点をまとめ直す (規約化する) ことで，BDD をさらに小さく圧縮できる．本書はアルゴリズムではなくネットワークの教科書であるため規約化についての詳細な説明は行わないが，BDD における重要な技術であり，興味のある読者は 5 章のコーヒーブレイクを参照されたい．
5. 可能である．アルゴリズムが論理演算の種類に依存するのは，終端節点を処理する部分だけである．たとえば，論理積と論理積のアルゴリズムを比べると，1–4 行目は演算ごとに定義されるが，他の行はほぼ同じで

Algorithm 1: $s_1 \diamond s_2$: 任意の論理演算 $\diamond$ を実行する APPLY アルゴリズム

Input: $s_1, s_2, \diamond$ /* ふたつの節点と演算子 */
Output: $s_1 \diamond s_2$
1 **if** *(*$s_1 = \top$ **or** $s_1 = \bot$*)* **and** *(*$s_2 = \top$ **or** $s_2 = \bot$*)* **then** /* 終端節点を処理する */
2 　**return** $s_1 \diamond s_2$
3 **else if** $(s_1, s_2) \in S$ **then** /* 等価な節点が S にある */
4 　**return** $S[s_1, s_2]$
5 **else** /* 新たに節点を生成する (図 6.8) */
6 　$s' \leftarrow$ NEWBDDNODE
7 　$s'.0 \leftarrow s_1.0 \diamond s_2.0$ /* 0 枝に対して再帰的に呼び出す */
8 　$s'.1 \leftarrow s_1.1 \diamond s_2.1$ /* 1 枝に対して再帰的に呼び出す */
9 　$S[s_1, s_2] \leftarrow s'$ /* $S[s_1, s_2]$ に保存する */
10 　**return** s'

ある．よって，演算対象が終端節点 ($\top$ あるいは $\bot$) であるときは演算ごとの処理を呼び出すが，それ以外は演算によらない再帰アルゴリズムとして記述できる．ある演算を行う演算子を $\diamond$ と表すとき，$s_1 \diamond s_2$ を行うアルゴリズム APPLY を示す．

6. (略)

6章 章末問題解答

1. IP アドレス $\boldsymbol{x}$ と接頭辞長 l が与えられたとき，l の節点を以下のルールに従って接続する．

$$s_i.x_i \leftarrow \begin{cases} s_{i+1} & 1 < i < l \\ \top & i = l \end{cases} \tag{3}$$

$$s_i.((x'_j + 1)\%2) \leftarrow \bot \tag{4}$$

2. 接尾辞を l ビットとすると，$32 - l$ の節点を以下のルールに従って接続する．

$$s_i.x_i \leftarrow \begin{cases} s_{i+1} & 32 - l < i < 32 \\ \top & i = 32 \end{cases} \tag{5}$$

$$s_i.((x'_j + 1)\%2) \leftarrow \bot \tag{6}$$

3. 直線状であったアドレス部分の BDD は，下位に移動することで幅を持つようになる．これは，上位に移動したポート番号によってアドレスに関する条件が異なるためである．この例ではビットを入れ替えても節点数は変わらないが，一般には変化する．ただし，BDD を再構築することなく節点数を正確に予測することは難しく，ビット順の最適化は重要な課題となっている．

4. BDD ベクトルについて説明する．k 種類の処理を行うとき，処理番号を $l = \lceil \log_2 k \rceil$ ビットにエンコードする．i 番目の処理番号を a_i とすると，$(a_1 = 00\cdots0, a_2 = 00\cdots1, \cdots, a_k)$ のようになる．各ビット $j \in \{1, 2, \cdots, l\}$ について，ビット値を $0 \to \bot, 1 \to \top$ のように読み替えて BDD を構築する．このようにして，処理番号のビットごとに l の BDD を得る．与えられたパケット $\boldsymbol{x}$ に対して l の BDD を辿り，$0 \leftarrow \bot$,

$1 \leftarrow \top$ のように逆に読み替えると，処理番号を得られる．BDD ベクトルが l 回の探索を行うのに対し，Multi-valued decision diagram は一回の探索で処理番号を得られる (詳細は参考文献 60 を参照).

5. アルゴリズム REACH 5 行目で代入を行う前に，書き換えルールに従って BDD を修正する．BDD の修正方法は書き換えルールによるが，多くの場合は指定されたビットを特定の値に変更するだけなので，容易である．

7 章 章末問題解答

1. 略
2. 略