半構造データに対する効率良い近接パターン照合アルゴリズム

和佐　州洋† 　金田　悠作† 　宇野　毅明†† 　有村　博紀†

† 北海道大学大学院情報科学研究科　〒 060–0814 札幌市北区北 14 条西 9 丁目
†† 国立情報学研究所　〒 101–8430 東京都千代田区一橋 2-12
E-mail: †{wasa,y-kaneta,arim}@ist.hokudai.ac.jp, ††uno@nii.ac.jp

あらまし　本稿では，近接パターン照合問題のひとつとして，木に対するグラフモチーフ問題に着目する．そのための重要な副問題である k-部分木列挙問題を考察し，その効率良い列挙アルゴリズムを与える．本論文では，問題の定義とわれわれのアルゴリズムを述べ，アルゴリズムの正当性と時間計算量について考察する．

キーワード　グラフマイニング, 近接パターン照合, 双方向ビットカウンタ, 部分木列挙, 文字列アルゴリズム

1. はじめに

文字列や，木，グラフのような非定型構造データを半構造データと呼ぶ．**近接パターン照合問題**(proximity pattern match problem) [3], [4], [6], [7] は，与えられた半構造データ上で，制約を満たしたアイテムの出現を発見する問題である．これは，検索とマイニングの中間に位置する問題であり，大規模文書検索や生物情報学で研究されている近接文字列照合問題 [6], [7] やグラフモチーフ問題 [3], [4] は近接パターン照合問題の例である．これらの問題では，パターンに対する最もゆるい制約として，連結性制約が用いられる．

グラフモチーフ問題(graph motif problem) [4] は，近接パターン照合問題のひとつである．これは，頂点にラベルを持つグラフ $G = (V(G), E(G))$(テキスト) とラベルの多重集合 M(モチーフ) を入力として受け取り，G の連結部分グラフ S で，そのラベルの多重集合が M に一致するようなものを見つける問題である．

本稿では，木に対するグラフモチーフ問題について着目し，その重要な副問題である **k-部分木列挙問題**(k-subtree enumeration problem) の効率良いアルゴリズムについて考察する．このとき，G は根付き木であり，S は G の (連結) 部分木である．主結果として，頂点数 n の木と正整数 $k \geq 1$ が与えられた時，解 1 個あたりならし定数時間で頂点数 k の部分木を重複なく全て列挙する効率良いアルゴリズムを与える．このような部分木を用いた DB 処理として，Saito と，Morishita [8] などがある．

2. 準　　備

2.1 木の定義

根付き木 $T = (V(T), E(T))$ とは，ノード集合 V と辺集合 $E \in V^2$ から成り立つ．V の要素数は $n = |V|$ で表す．また，根付き木の**サイズ**を $|T| = |V| = n$ と定義する．各辺 $(u, v) \in E$ に対して，u を v の**親ノード**といい，$\mathsf{pnode}(v) = u$ と書く．また，v を u の**子ノード**という．u の子ノードの数を u の**次数**という．根ノード $\mathsf{root}(T)$ 以外は 1 つの親ノードを持つ．子ノードのないノードを**葉**という．根ノードと葉以外のノードを**中間ノード**という．同じ親ノードを持つノードを**兄弟**という．T の葉の集合を $\mathsf{L}(T)$ とする．ノード v に対して，$v[i]$ を v の **i 番目の子供**とする．

兄弟間に大小関係のある根付き木を**順序木**という．大きいノードほど左側にあるとする．以下，順序木を木という．ノード u, v に対して，u から v への長さ $k \geqq 0$ の**パス**とは，ノードの列 $\pi = (v_0 = u, v_1, \cdots, v_k = v)$ で，各 $i = 1, \cdots, k$ に対して，$(v_{i-1}, v_i) \in E$ であるものである．木 T で，任意の 2 ノードの間にパスが存在する時，T は**連結**(connected) であるという．ノード v の**先祖**とは，ノード v と根ノードを結ぶパス上の v を除くノード集合をいう．

ここで，左優先深さ優先探索順 (以下，DFS 順という) を考える． 木 $T = (V(T), E(T))$ の各ノード v の *DFS* **順の番号**を $\mathsf{dfs}(v) \in \{1, \cdots, |V|\}$ とおき，$\mathsf{dfs}(v)$ を v の**番号**という．以下では，文脈から明らかならば，ノードと番号を同一視する．よって，頂点数 n の順序木 T の頂点集合を $V(T) = \{1, \ldots, n\}$ とみなす．

2.2 部分木の定義

木 $T = (V(T), E(T))$ を固定する．$V(T)$ の部分集合 S が誘導する T の連結な**部分木**を $T(S) = (S, E(S))$ とする．ここで，$E(S) = \left\{(u, v) \in E(T) | \{u, v\} \subseteqq S\right\}$ とする．以後，文脈から明らかならば，部分集合 S を部分木ということがある．部分木 S のサイズをノード数 $|S|$ と定義する．サイズ k の部分木を **k-部分木**という．$T(v)$ は，ノード v を根ノードとする T の部分木を表す．ノード u の**子孫**$\mathsf{dec}(u)$ とは，$\mathsf{dec}(u) = \{v \in T(u) | u \neq v\}$ で表されるノード集合をいう．

2.3 列挙アルゴリズム

列挙とは，与えられた条件を満たす解を漏れ無く重複なく重

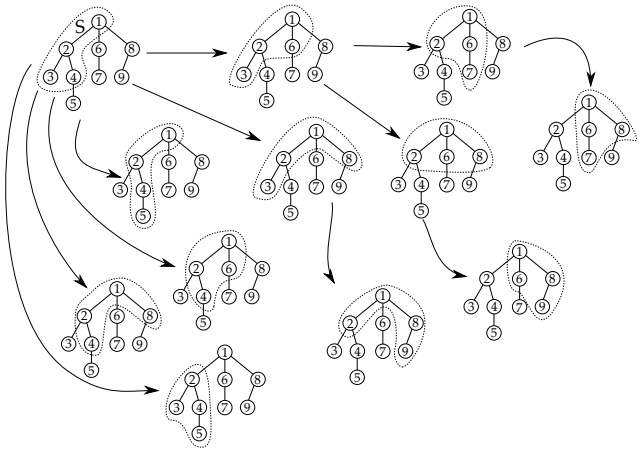

図 1　探索木の例.

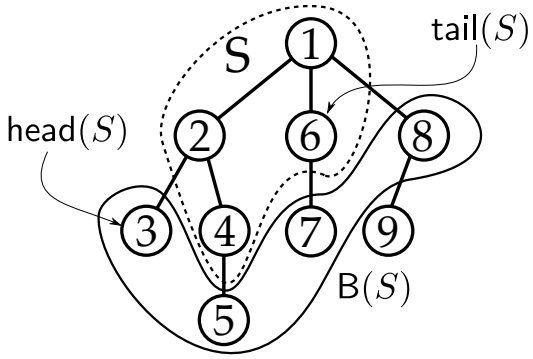

図 2　番号 $1,2,4,6$ をノードとする部分木 S に対する $\mathsf{tail}(S)$, $\mathsf{B}(S)$, $\mathsf{head}(S)$ の例である．$\mathsf{tail}(S)$ は番号 6 を持つノード, $\mathsf{B}(S)$ は番号 $3,5,7,8$ を持つノードの集合, $\mathsf{head}(S)$ は番号 3 を持つノードである.

複することである [1], [9]．列挙アルゴリズムの計算量は，解一個あたりを出力するのに必要な計算時間 (**遅延**) で評価する．本論文では，Ferreira と，Grossi，Rizzi ら [5] が研究したグラフ上の k-部分木列挙問題を，入力グラフ G が木の場合に制限して次の問題を考察する．以下では，k を任意の正整数とする．

問題 1 (k-部分木列挙問題)**.** 入力として与えられた木 $T = (V(T), E(T))$ においてサイズ k の部分木を重複なくすべて列挙する.

計算モデルとしては，通常の命令を持つ RAM [2] を用いる.

3. 部分木同士の親子関係

われわれの列挙アルゴリズムは逆探索手法 [1], [9] に基づく．これは，与えられた解空間上に探索木 (家系木) を定義し (図 1)，根から出発して，バックトラックしながら，全ての解を列挙する手法である．$T = (V(T), E(T))$ をサイズ n の任意の根付き木とする．以下では，与えられた根付き木のノードに対して，DFS 順の番号が振られていると仮定する．ノードとその番号を同一視する．よって，ノード u がノード v より大きいなどという．初めに，出発点となる初期木を定義する．その準備として，連続木を定義する.

定義 1 (連続木)**.** 木 T の任意のノード $r \in T(V)$ が $|T(r)| \geqq k$ を満たすとき，サイズ k の**連続木**は $\mathsf{con}_k(r) = \{r, \ldots, r+k-1\}$ である.

ここで，ノードと番号を同一視していることに注意されたい.

定義 2 (初期木)**.** 木 T の連続木 $\mathsf{con}_k(1)$ を特に**初期木**といい，$\perp(k) = \mathsf{con}_k(1) = \{1, \ldots, k\}$ と書く.

補題 1. 木 T の任意のノード r が $|T(r)| \geqq k$ を満たすとき，$S = \mathsf{con}_k(r)$ はサイズ k の連結した部分木である.

証明. 仮定から，S に含まれるノード全ては $T(r)$ に含まれる．ノードの番号付けが DFS 順であることを考慮すると，これらから S は明らかに連結な部分木である. ■

次に，与えられた部分木 $S \subseteqq V(T)$ の「親」となる部分木 $\mathsf{P}(S)$ を定義する．T 上のサイズ $k \leqq |T|$ の部分木 S に対して $\mathsf{tail}(S)$ と, $\mathsf{head}(S)$, $\mathsf{B}(S)$ を以下のように定義する (図 2).

定義 3 (境界)**.** S に含まれるノードの子ノードで，かつ S に含まれないノードの集合を S の境界といい，$\mathsf{B}(S) = \{v \in V(T) \mid u \in S, (u, v) \in E(T), v \notin S\}$ とする.

定義 4 (先頭)**.** $\mathsf{B}(S)$ の中で最も小さいノードを先頭といい，$\mathsf{head}(S) = \operatorname*{argmin}_{v \in \mathsf{B}(S)} v$ とする.

定義 5 (末尾)**.** S の中で最も大きいノードを末尾といい，$\mathsf{tail}(S) = \operatorname*{argmax}_{v \in S} v$ とする.

補題 2. 木 T のサイズ k の部分木 S における $\mathsf{tail}(S)$ は葉である．ただし，$k \leqq |T|$ とする.

証明. $v = \mathsf{tail}(S)$ が S において中間ノードであるとすると，子ノード u を持つ．このとき，$u > v$ であるので，$\mathsf{tail}(S)$ は S 中で最大という定義に反する．よって，$\mathsf{tail}(S)$ は葉である. ■

定義 6 (親木)**.** $n \geqq k$ を満たす木 T のサイズ k の部分木 S の**親木**$\mathsf{P}(S)$ は，以下で定義される．ただし，$S \neq \perp(k)$ とする.

$$\mathsf{P}(S) = \begin{cases} (S \setminus \{\mathsf{tail}(S)\}) \cup \{\mathsf{head}(S)\}, \text{ if } S \text{ が連続木でない} \\ (S \setminus \{\mathsf{tail}(S)\}) \cup \{\mathsf{pnode}(\mathsf{root}(S))\}, \text{ if } S \text{ が連続木である} \end{cases}$$

定義 7 (子木)**.** 部分木 R に対して，ある手続きを行い出力された部分木を S とする．このとき，$P(S) = R$ となるような S を R の**子木**という．また，このような R と S に対して，R と S に**親子関係**がある，という.

補題 3. 木 T の任意のノード r が $|T(r)| \geqq k$ を満たすとする．このとき，根ノードが r でサイズ k の部分木 S を考える．*(1)*S がサイズ k の連続木ではないならば，その時に限り，*(2)*$\mathsf{head}(S) < \mathsf{tail}(S)$ である.

証明. (1) ⇒ (2): 木 T の連続木でない部分木 $S = \{x_1, x_2, \cdots, x_k\}$ を考える．ただし，$x_1 < x_2 < \cdots < x_k$ とす

る．このとき，$1 \leqq i < k$ に対して，$x_i \in S$ かつ $y = x_i + 1 \notin S$ かつ $y < x_{i+1}$ となる i が存在する．ここで，DFS の番号付けから，y は x_i の先祖の子，または x_i の子となる．また，$x_k = \mathsf{tail}(S)$ である．よって，このような y の中で最小のノードが定義 4 より $\mathsf{head}(S)$ となるので，$\mathsf{head}(S) < \mathsf{tail}(S)$ となる．

(2) ⇒ (1): S が連続木であるとする．このとき，$\mathsf{tail}(S) = r + k - 1$ である．$S = \{r, \cdots, \mathsf{tail}(S)\}$ と連続であり，かつ $\mathsf{head}(S) \notin S$ であるので，$r < \mathsf{head}(S)$ から $\mathsf{tail}(S) < \mathsf{head}(S)$ となる．対偶より，$\mathsf{head}(S) \neq \mathsf{tail}(S)$ から $\mathsf{head}(S) < \mathsf{tail}(S)$ であるとき，S は連続木ではない． ■

また，補題 3 より，自明に系 1 が導かれる．

系 1. 木 T で任意の連続木でない部分木 S に対して，$(\mathsf{tail}(S), \mathsf{head}(S)) \notin E(T)$ である．

補題 4. T のサイズ k の任意の部分木 S に対して，もし，S が初期木でないならば $\mathsf{P}(S)$ は正しく定義される．

証明． $\mathsf{P}(S)$ は S が連続木であるときと，そうでないときに異なる．よって，場合分けをして証明する．

S が連続木でないとき: S において，$\mathsf{head}(S)$ は S のノードの子ノードであり $\mathsf{tail}(S)$ は補題 2 より葉ノードである．また，S は連続木ではないので，系 1 より $\mathsf{head}(S)$ が $\mathsf{tail}(S)$ の子ノードとなることはない．よって，$\mathsf{tail}(S)$ を除き $\mathsf{head}(S)$ を加えた部分木は連結であるので，$\mathsf{P}(S)$ は正しく定義される．

S が連続木であるとき: S において，$\mathsf{tail}(S)$ は補題 2 より葉ノードである．また，S は初期木ではないので，$\mathsf{pnode}(\mathsf{root}(S))$ は必ず存在し，また，$\mathsf{root}(S)$ の親ノードである．よって，$\mathsf{tail}(S)$ を除き，$\mathsf{pnode}(\mathsf{root}(S))$ を加えた部分木は連結であるので，$\mathsf{P}(S)$ は正しく定義される． ■

4. 列挙アルゴリズム

前章では，子木から親木を求める方法について考察した．本章では，親木から子木を求める方法について，子木が連続木であるときとそうでないときに分けて考察する．そのあとで，木 T の k-部分木を定数遅延時間で列挙するアルゴリズムを示し，その正当性と時間計算量について考察する．

4.1 連続木でない子木の生成

木 T の部分木 R から子木 S を求めるには，子木 S からその親木である R を求める逆の手順を踏めば良い．つまり，適当なノード $u \in \mathsf{L}(R)$ を削除し，R に含まれるノードの子ノードでかつ $\mathsf{tail}(R)$ よりも大きく R に含まれないノード v を追加すれば良い．しかし，ノード u, v の候補は一つとは限らない．そこで，u の候補となるノードの集合を $\mathsf{can}(R)$，v の候補となるノードの集合を $\mathsf{LB}(R)$ とし，以下のように定義する (図 3)．

定義 8. 部分木 R の葉の中で，$\mathsf{head}(R)$ よりも小さいノードの集合を $\mathsf{can}(R) = \{u \in \mathsf{L}(R) \mid u < \mathsf{head}(R)\}$ とする．

定義 9. 部分木 R のボーダーの中で，$\mathsf{tail}(R)$ よりも大きいノードの集合を $\mathsf{LB}(R) = \{v \in \mathsf{B}(R) \mid \mathsf{tail}(R) < v\}$ とする．

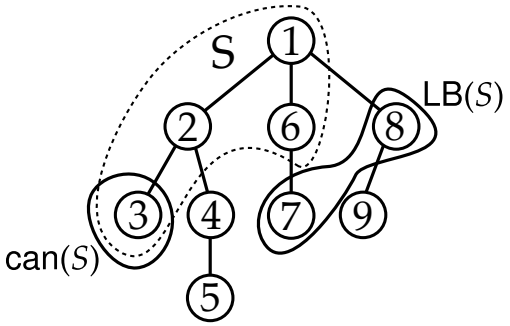

図 3 部分木 S に対する $\mathsf{can}(S)$ と $\mathsf{LB}(S)$ の例．$\mathsf{can}(S)$ は番号 3 を持つノードの集合，$\mathsf{LB}(S)$ は番号 7, 8 を持つノードの集合である．

定義 10. 木 T の部分木 R に対して，$u \in \mathsf{can}(R)$，$v \in \mathsf{LB}(R)$，$(u, v) \notin E(T)$ であるような u, v に対して，

$$\mathsf{C}_1(R) = (R \setminus \{u\}) \cup \{v\}$$

を R の子木とする．

補題 5. 木 T の部分木を R とする．ここで，$\mathsf{can}(R) \neq \emptyset$ かつ $\mathsf{LB}(R) \neq \emptyset$ のとき，$\mathsf{C}_1(R) = (R \setminus \{u\}) \cup \{v\}$ は木 T の連結な部分木である．ただし，$u \in \mathsf{can}(R)$，$v \in \mathsf{LB}(R)$，$(u, v) \notin E(T)$ とする．

証明． $(u, v) \notin E(T)$ であり $\mathsf{can}(R) \subseteqq \mathsf{L}(R)$, $\mathsf{LB}(R) \subseteqq \mathsf{B}(R)$ であるので，$\mathsf{C}_1(R)$ は連結な部分木である． ■

補題 6. 木 T の任意の部分木を R とし，T の連続木でない部分木を S とする．ここで，*(1)*$u \in \mathsf{can}(R)$ かつ $v \in \mathsf{LB}(R)$ かつ $(u, v) \notin E(T)$ であるような u, v に対して，木 T の部分木 S が $S = \mathsf{C}_1(R) = (R \setminus \{u\}) \cup \{v\}$ が R の子木であるならば，その時に限り，*(2)*$\mathsf{head}(S) \in \mathsf{can}(R)$ かつ $\mathsf{tail}(S) \in \mathsf{LB}(R)$ である．

証明． (1) ⇒ (2): 補題 5 より，S は連結な部分木である．対偶を用いて示す．$\mathsf{head}(S) \notin \mathsf{can}(R)$ または $\mathsf{tail}(S) \notin \mathsf{LB}(R)$ であるとする．ここで，$\mathsf{P}(S) = (S \setminus \{\mathsf{tail}(S)\}) \cup \{\mathsf{head}(S)\}$ である．しかし，$R = (S \setminus \{v\}) \cup \{u\}$ であるが，$u \in \mathsf{can}(R)$ かつ $v \in \mathsf{LB}(R)$ であるので，$\mathsf{P}(S)$ と R は一致しない．よって，S は R の子木でない．よって，(1) ⇒ (2) が示された．

(2) ⇒ (1): $\mathsf{head}(S) \in \mathsf{can}(R)$ かつ $\mathsf{tail}(S) \in \mathsf{LB}(R)$ と仮定する．ここで，$u = \mathsf{head}(S)$ かつ $v = \mathsf{tail}(S)$ とおく．このとき，S は連続木ではないので，系 1 から $(v, u) \notin E(T)$ である．よって，S は連結な部分木である．ここで，

$$\begin{aligned} \mathsf{P}(S) &= (S \setminus \{\mathsf{tail}(S)\}) \cup \{\mathsf{head}(S)\} \\ &= ((R \setminus \{\mathsf{head}(S)\}) \cup \{\mathsf{tail}(S)\}) \setminus \{\mathsf{tail}(S)\}) \cup \{\mathsf{head}(S)\} \\ &= R \end{aligned}$$

であるので，S と R は親子関係である．よって，(2) ⇒ (1) が示された． ■

補題 7. T 上の連続木でないサイズ k の S について，$(u, v) \notin E(T)$ である．ただし，$u \in \mathsf{can}(S)$，$v \in \mathsf{LB}(S)$ とする．

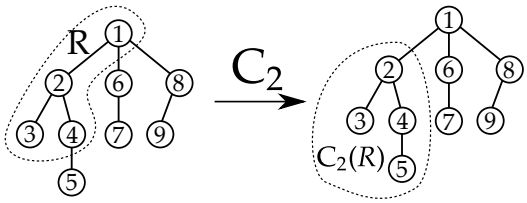

図 4　擬連続木 R から得られる連続木 $\mathsf{C}_2(R)$ の例.

証明.　S は連続木ではないので，補題 3 より $\mathsf{head}(S) < \mathsf{tail}(S)$ である．よって，$\mathsf{can}(S)$ の定義から，$\mathsf{tail}(S)$ と $\mathsf{tail}(S)$ の先祖は $\mathsf{can}(S)$ に含まれない．一方，$v \in \mathsf{LB}(S)$ は定義より $v > \mathsf{tail}(S)$ であるので，v の親ノードは $\mathsf{tail}(S)$ または，$\mathsf{tail}(S)$ の先祖である．これらから，任意の v は任意の u の子ノードでない．■

補題 8. T 上のサイズ k の連続木 S について，$(u,v) \in E(T)$ であるのは，$u = \mathsf{tail}(S)$ のときである．ただし，$u \in \mathsf{can}(S)$，$v \in \mathsf{LB}(S)$ とする．

証明.　任意の $v \in \mathsf{LB}(S)$ に対して，v の親ノード y が S 中で葉であるときとそうでないときに場合分けし，y と u が一致する場合について考察する．$y \notin \mathsf{L}(S)$ のとき，$\mathsf{can}(S)$ の定義から $u \in \mathsf{L}(S)$ であるので，明らかに $(u,v) \notin E(T)$ である．$y \in \mathsf{L}(S)$ のとき，$\mathsf{LB}(S)$ の定義から y は $\mathsf{tail}(S)$ と等しくなくてはいけない．よって，$(u,v) \in E(T)$ であるのは，$y = u$，つまり $u = \mathsf{tail}(S)$ のときである．よって，これらから，v が u の子ノードであるのは，$u = \mathsf{tail}(S)$ のときである．■

4.2　連続木である子木の生成

次に，初期木でない連続木 S をある木の子木として求める方法について考察する．その準備として，擬連続木を導入する．

定義 11 (擬連続木)**.** 木 T の部分木 R に対して，部分木 R の根ノード $\mathsf{root}(R)$ が $|T(v)| \geqq k$ を満たすノード v を子ノードとしてただひとつに持ち，かつ，$\mathsf{tail}(R) - v = k - 2$ であるような R を**擬連続木**という．

定義 12. 木 T のサイズ k の擬連続木であるような部分木 R に対して，

$$\mathsf{C}_2(R) = (R \setminus \{\mathsf{root}(R)\}) \cup \{\min \mathsf{LB}(R)\}$$

を R の子木とする．

擬連続木 R に対して，$\mathsf{C}_2(R)$ とすることで，連続木が得られることを以下で示していく 4.

補題 9. 木 T の任意の部分木 S に対して，$\mathsf{LB}(S) \neq \emptyset$ であるとき，$\mathsf{tail}(S) + 1 = \min \mathsf{LB}(S)$ である．

証明.　$l = \mathsf{tail}(S) + 1$ とする．l は，DFS 番号付けと $\mathsf{tail}(S)$ の定義から $\mathsf{tail}(S)$ の子ノード，もしくは $\mathsf{tail}(S)$ の先祖の子ノードである．また，$\mathsf{tail}(S)$ が S で最大のノードであるので，$l \notin S$ である．よって，$l \in \mathsf{LB}(S)$ である．ここで，ノードの番号付けが整数であること考えると，$\mathsf{tail}(S)$ と l の間にノードは存在しないので，定義 9 から $l = \min \mathsf{LB}(S)$ である．■

Algorithm 1 木 T のサイズ k の部分木 S を列挙

```
1: procedure Main(T, k)
2:     Input: T : 木, k : サイズ;
3:     T に対して各ノードに対し，DFS 順に番号をつける;
4:     S ← 初期木;
5:     can(S)，LB(S) の計算;
6:     EnumSubTreeConstant(S, T, k);

1: procedure EnumSubTreeConstant(R, U, k)
2:     Input: R : 部分木，U : 木, k : サイズ;
3:     print R;
4:     if R が擬連続木である then
5:         R' ← C2(R)
6:         can(R')，LB(R') の再計算;
7:         EnumSubTreeConstant(R', U, k);
8:     for each u ∈ can(R) do
9:         for each {v ∈ LB(R) | (tail(R), v) ∉ E(R)} do
10:            S ← (R \ {u}) ∪ {v} ;
11:            can(S)，LB(S) の再計算;
12:            EnumSubTreeConstant(S, U, k);
```

補題 10. 木 T の擬連続木 S に対して，$|S| \geqq k$ のとき，$\mathsf{C}_2(S)$ は連続木である．

証明.　補題 9 より，$\mathsf{tail}(S)+1 = \min \mathsf{LB}(S)$ であるので，$\mathsf{C}_2(S)$ は定義より連続木である．■

補題 11. 木 T のサイズ k の初期木でない連続木 S には $\mathsf{C}_2(R) = S$ となる擬連続木 R がただひとつ存在する．

証明.　$\mathsf{C}_2(R) = S$ となるような擬連続木 R の根ノードを α とする．定義 11 より，$S \cap R = \{\alpha, \alpha+1, \cdots, \alpha+k-2\}$ である．ここで，$A = (A \cap R) \cup \{\min \mathsf{LB}(A)\}$ である．また，R は初期木でない連続木であるので，α には T 上で親ノードがただひとつ存在する．よって，これらから R に対応する A はただひとつ存在する．■

4.3　アルゴリズム

これらから，入力として与えられた木 $T = (V(T), E(T))$ の k-部分木 $S = (S, E(S))$ をならし定数時間で列挙できるアルゴリズムを Algorithm 1 に挙げる．

Algorithm 1 は与えられた木の初期木の子木を再帰的に出力する．その子木の中に擬連続木が存在すれば，その擬連続木に対応する連続木を求め，その連続木の子木を再帰的に出力することで，全ての k-部分木を列挙するアルゴリズムである．

4.4　正　当　性

本節では，Algorithm1 の正当性について考察する．Main は木 T とサイズ k を受け取り，T 上のサイズ k の部分木をすべて列挙するメインルーチンである．3 行目で，T に DFS 番号付けをし，6 行目から，初期木で EnumSubTreeConstant を呼び出している．初期木は家系木の根となる．

EnumSubTreeConstant の 4-7 行目で連続木を擬連続木か

ら呼び出し，呼び出した連続木に対して EnumSubTreeConstant を再帰的に呼び出す．また，8-12 行目より R の子木を再帰的に出力している．正当性は，4-7 行目の処理で，T に含まれる連続木を再帰的にすべて呼び出すことができることが補題 11 からわかる．また，8-12 行目の処理で，処理全体を通して家系木を構築し，T の部分木をすべて列挙できることが補題 7,8 からわかる．

よって，これらから Algorithm1 は Main を実行することで，木 T とサイズ k を入力とした時，サイズ k の T 中に含まれる部分木をすべて列挙する．

4.5 時間計算量

本節では，Algorithm1 の時間計算量について考察する．ここで，部分木 S の葉集合の保存には双方向連結リスト dl_S を，$\mathsf{LB}(S)$ の保存には双方向連結リストを db_S を用いる．dl_S を求めることで，$\mathsf{head}(S)$ を用いると，$\mathsf{can}(S)$ が定義より得られる．また，$\mathsf{dl}_S(\mathsf{db}_S)$ の i 番目の要素は $\mathsf{dl}_S[i](\mathsf{db}_S[i])$ と書く．ただし，添字 i は 1 から始まるものとする．

補題 12. 木 T の初期木に対して $\mathsf{can}(\perp(k))$ は $O(|T|)$ 時間で得られる．

証明． DFS 順に葉を $\mathsf{dl}_{\perp(k)}$ に追加すると，$O(|T|)$ 時間で初期木の葉の昇順のリスト $\mathsf{dl}_{\perp(k)}$ を構築できる． ■

補題 13. 木 T の初期木に対して $\mathsf{LB}(\perp(k))$ は $O(|T|)$ 時間で得られる．

証明． DFS 順で $\mathsf{tail}(\perp(k))$ に到達したあと，行き掛け順に $\mathsf{LB}(\perp(k))$ の定義を満たすノードをリストに追加していけば，$O(|T|)$ 時間で，$\mathsf{LB}(\perp(k))$ の昇順のリスト $\mathsf{db}_{\perp(k)}$ が得られる． ■

補題 12, 13 から，初期化として Main の 3-5 行目の処理には $O(|T|)$ 時間かかる．

EnumSubTreeConstant の 4 行目の判定には根ノードの子の数が 1 つであること，またその子ノード f と $\mathsf{tail}(R)$ まで連番であること，つまり $\mathsf{tail}(R) - f = k - 2$ であることを確認し，$\mathsf{B}(R)$ に適切なノードが入っているか確認すればよいので，$O(1)$ 時間である．また，5 行目の初期木の生成には定数回の操作で済むので $O(1)$ 時間である．また，8-12 行目の処理で 12 行目が呼び出される間の遅延を $O(1)$ 時間にするには，u が $\mathsf{tail}(R)$ であるときに，u の子であるようなノード v をすべてスキップすればよい．このスキップは u の子ノードで最も大きいノードへのアクセスが $O(1)$ 時間で出来れば，db_R が昇順で並んでいることを考慮すると，12 行目の呼び出しの遅延は $O(1)$ 時間である．6 行目の再計算，11 行目の再計算は以下の補題に関して考察する必要がある．

補題 14. 木 T 上の擬連続木 S に対して，$S' = \mathsf{C}_2(S)$ を求めた後，$\mathsf{can}(S')$ の再計算にかかる時間は $O(1)$ 時間である．

証明． $v = \min \mathsf{LB}(S)$ とする．S' は連続木であるので，補題 3 より，$\mathsf{head}(S') > \mathsf{tail}(S')$ となる．また，$\mathsf{dl}_{S'}$ は dl_S の末尾に v が追加し，もし，$\mathsf{pnode}(v)$ が S で葉であった時は，dl_S から $\mathsf{pnode}(v)$ を除いたリストである．また，$\mathsf{head}(S')$ は $\mathsf{LB}(S')$ の先頭であるので，$\mathsf{LB}(S')$ の操作が $O(1)$ 時間で再計算できれば，$\mathsf{can}(S')$ の再計算は $O(1)$ 時間で実行可能である． ■

補題 15. 木 T の擬連続木 S に対して，$S' = \mathsf{C}_2(S)$ を求めた後，$\mathsf{LB}(S')$ の再計算にかかる時間は $O(1)$ 時間である．

証明． $v = \min \mathsf{LB}(S)$ とする．$\mathsf{LB}(S')$ は $T(\mathsf{root}(S'))$ の子孫ノードであるので，$\mathsf{db}_{S'}$ は db_S から子孫でないノードを削除する．この操作は，db_S が昇順であることを考慮すると，$O(1)$ 時間で可能である．また，S' に新しく追加するノード v を db_S から除いてから，v の子ノードすべてを db_S の先頭に加えれば維持できる．よって，再計算にかかる時間は $O(1)$ 時間である． ■

補題 16. 木 T の部分木 S に対して，$S' = \mathsf{C}_1(S)$ を求めた後，$\mathsf{can}(S')$ の再計算にかかる時間は $O(1)$ 時間である．

証明． まず S の葉集合の管理に関して考察する．ここで，u は S の子木を求めるときに削除した葉ノードであるとする．$\mathsf{dl}_{S'}$ は dl_S から求める場合は，以下の 2 つに場合分けをする．$\mathsf{pnode}(u)$ が u のみを S 中で子ノードに持つ時は，dl_S 中の u を $\mathsf{pnode}(u)$ に置き換える．もし，$\mathsf{pnode}(u)$ が S 中で 2 つ以上の子ノードを持つ時は，u を dl_S 中から削除する．よって，$\mathsf{dl}_{S'}$ は dl_S から求める時，$O(1)$ 時間で可能である．よって，u が $\mathsf{head}(S')$ と等しくなることを考慮すると，$\mathsf{can}(S')$ の再計算には $\mathsf{dl}_{S'}$ と u を持っておけば良いので，再計算にかかる時間は $O(1)$ 時間である． ■

補題 17. 木 T の部分木 S に対して，$S' = \mathsf{C}_1(S)$ を求めた後，$\mathsf{LB}(S')$ の再計算にかかる時間は $O(1)$ 時間である．

証明． $\mathsf{C}_1(S)$ で S に追加したノード v が $\mathsf{db}_S[j] = v$ であるとする．このとき，v が木 T の中で葉かどうかで場合分けして考察する．v が T の中で葉の時，$\mathsf{LB}(S)$ から $\mathsf{LB}(S')$ に再計算される際に追加されるノードはない．v は定義から $\mathsf{tail}(S') = v$ となるので，$\mathsf{LB}(S)$ から $\mathsf{LB}(S')$ に再計算される際，v より小さいノードが $\mathsf{LB}(S)$ から削除される．よって，更新された $\mathsf{db}_{S'}$ は $\mathsf{db}_{S'}[1] = \mathsf{db}_S[j+1], \mathsf{db}_{S'}[2] = \mathsf{db}_S[j+2], \cdots$ となる．v が T の中で中間ノードの時，$\mathsf{LB}(S)$ から $\mathsf{LB}(S')$ に再計算される際に追加されるノードは v の T の子ノード全てである．v は定義から $\mathsf{tail}(S') = v$ となるので，$\mathsf{LB}(S)$ から $\mathsf{LB}(S')$ に再計算される際，v より小さいノードが $\mathsf{LB}(S)$ から削除される．v の次数を d とすると，更新された $\mathsf{db}_{S'}$ は $\mathsf{db}_{S'}[1] = v[1], \mathsf{db}_{S'}[2] = v[2], \cdots, \mathsf{db}_{S'}[d] = v[d], \mathsf{db}_{S'}[d+1] = \mathsf{db}_S[j+1], \cdots$ となる．これらの操作は，リストのポインタの張替えで行えるので，$O(1)$ 時間で実行できる．よって，再計算にかかる時間は $O(1)$ 時間である． ■

よって，EnumSubTreeConstant の 6 行目の再計算は補題 14，15，11 行目の再計算は補題 16，17 から $O(1)$ 時間であることがわかる．今までの議論から以下の定理が得られる．

定理 1. *Algorithm1* は前処理に $O(|T|)$ 時間，解 *1* 個あたりならし定数時間で k-部分木をすべて列挙する．

5. ま と め

本稿では，木に対するグラフモチーフ問題を着目し，そのための重要な手続きである根付き木に対する部分木列挙問題を考察した．さらに与えられた木のサイズ k の部分木を定数時間遅延で列挙するアルゴリズムを与えた．

文　献

[1] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1993.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2 edition, 2001.

[3] R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. In *the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM'11)*, volume 6661 of *Lecture Notes in Computer Science*, pages 388–401. Springer, 2011.

[4] M. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In *Proceedings of 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *Lecture Notes in Computer Science*, pages 340–351. Springer, 2007.

[5] R. Ferreira, R. Grossi, and R. Rizzi. Output-sensitive listing of bounded-size trees in undirected graphs. In *Proceedings of the 19th European conference on Algorithms (ESA'11)*, volume 6942 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2011.

[6] S.-R. Kim, I. Lee, and K. Park. A fast algorithm for the generalized k-keyword proximity problem given keyword offsets. *Inf. Process. Lett.*, 91(3):115–120, 2004.

[7] K. Sadakane and H. Imai. Fast algorithms for k-word proximity search. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E84-A(9):2311–2318, 2001.

[8] T. L. Saito and S. Morishita. Relational-style xml query. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 303–314, New York, NY, USA, 2008. ACM.

[9] A. Shioura, A. Tamura, and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.