1996年度日本オペレーションズ・リサーチ学会
春季研究発表会

有向グラフの根付き木を列挙するアルゴリズム

02003590 東京工業大学 宇野 毅明 UNO Takeaki

$G=(V,A)$ を有向グラフとする. 頂点 r を根とする G の有向根付き木で, すべての頂点を含むものを全域有向根付き木と呼ぶ. 以後, 簡単のため単に根付き木と記する. 今回の発表では G に含まれる全ての全域根付き木を効率良く列挙するアルゴリズムを提案する.

この問題に関する研究は, 1978 年の H.N.Gabow と E.W.Myers の論文や [1] 等で行われている。また、無向グラフに対する問題に関しては [2] で最適なアルゴリズムが提案されている。今回のアルゴリズムはこの論文の技法を使用している。

T_0 を, G に対して深さ優先探索を行い求められた根付き木とする. G の各頂点に探索順に添え字を付け, 枝の添え字をその終点の添え字とする. この T_0 を用いて, 根付き木間に親子関係を定義する. ある根付き木 T_c があるとき, $v^*(T_c)$ を $T_0 \setminus T_c$ に含まれる最小添え字の枝 e の終点とする. また, f を e と終点を共有する T_c の枝とする. このとき, $T_p = T_c \setminus f \cup e$ を T_c の親とする. e と f は一意的に定まるので親も一意的に定まる.

この親子関係をグラフで表してみる. 各頂点に根付き木をあてがい, その親に対応する頂点に枝を張る. T_c の親 T_p は, T_c よりも T_0 の枝を 1 つ多く含むので, このグラフは閉路を含まないことが確認できる. よってこの枝集合はすべての根付き木を張るような全域木を定める. アルゴリズムはこの全域木に対し深さ優先探索を行い, すべての頂点, すなわち対応する根付き木を訪問する. この方法で列挙を行うと, 根付き木の出力を前に出力した木との差分で行うことにより, 出力時間を根付き木 1 個当たり $O(1)$ にすることが可能である.

ある根付き木 T_p が与えられたときに, T_p から $v^*(T_p)$ の添え字より小さい添え字の枝 e を抜き, 枝 f を入れて $T_c = T_p \setminus e \cup f$ を作る. もし T_c が根付き木となれば T_c は T_p の子となる. T_c が根付き木となるとき, またそのときに限り T_p で f の始点が f の終点の子孫にならない. このような枝を T_p の非後退枝と呼ぶ. また, それ以外の T_p に含まれない枝を後退枝と呼ぶ.

後退枝については以下のような補題が成り立つ [1].

補題 1 T_p の子 $T_c = T_p \setminus e \cup f$ の後退枝は, その添え字が e の添え字より小さければ T_p の後退枝である.

証明: T_0 が深さ優先探索によって作られていることより, e より添え字の小さい頂点 v に関しては, v の T_c での子孫で T_p での子孫でないものは存在しない.∎

ここで根付き木を列挙するアルゴリズムの枠組みを示す.

アルゴリズム: ENUM_ROOTED_TREES (T_0)
・T_0 の 全ての子供 T_c について,
　・T_c で非後退枝を列挙 (※).
　・ENUM_ROOTED_TREES(T_c) を再帰呼びだしし, T_c の子孫を列挙.

既存の研究 [1] では, (※) の作業を高速化することが出来ず, 特に, 新しく発生する非後退枝, 即ち T_0 で後退枝であり T_c で非後退枝となるような枝を 1 つみつけるのに $O(|V|)$ の時間を必要とした. 他の部分は $O(\log|V|)$ 程度の時間で実行できるのに, これがネックとなり根付き木 1 個当たりの計算時間が $O(|V|)$ となっていたのである. 今回の改良点はこの作業時間を最小全張木を更新する時間で押さえたことである. 最小全張木の更新は, $O(|A|^{1/2})$ [4], $O(|V|^{1/2}\log(|A|/|V|))$ [3] で行えるという研究結果がある.

根付き木 Tに対して, T と T に対する後退枝の一部を合わせ, 枝を無向化したグラフを $B(T)$ とする. ただし $B(T)$ は, 添え字が $v^*(T)$ よりも小さい後退枝はすべて含むものとする. $B(T_0)$ は T_0 とそのすべての後退枝を合わせたグラフになる. また $B(T)$ の枝の重みを, T の枝は 0, T 以外の枝は, ($|V|+1$ − 添え字) と定義する. $B(T)$ の 最小全張木は T である.

先の補題より, $B(T_p)$ から T_p の子 $T_c = T_p \backslash e \cup f$ の非後退枝をすべて消去すると $B(T_c)$ になることがわかる. 消去する枝は次の性質によって特徴づけられる. e の終点を v, v と f の始点の T_p での最近共通祖先の頂点を w とする.

補題 2 T_p での w と e の始点を結ぶパスから w を除いたものを P と置く. T_p の後退枝で T_c の非後退枝であるものは, その始点が T_p で v の子孫であり, 終点が P の頂点である. また, 逆も成り立つ.

証明: v の子孫は, T_p では P の頂点の子孫であるが T_c では子孫でない. また, T_c と T_p で先祖集合が異なる頂点は v の子孫, 子孫集合が異なる頂点は P の頂点だけである.∎

以下に, T_p の後退枝で T_c の非後退枝であるものを列挙し, $B(T_p)$ から $B(T_c)$ を求めるアルゴリズムを示す.

1: e を $B(T_p)$ から消去する.
2: $B(T_p)$ の最小全張木を求め, 追加された枝を b とする.
3: b は v の子孫を始点とするので, b の終点が w の真の子孫であれば b は T_c で非後退枝になる. b を $B(T_p)$ から消去し, **2:** へもどる.
4: そうでなければ $B(T_c)$ の非後退枝はもう存在しない (**2:** で最小全張木を求めているため). $B(T_p)$ に f を加え, その重みを 0 とする.

定理 1 このアルゴリズムは非後退枝を 1 つあたり最小全張木の更新時間で列挙する.

証明: b は後退枝であるので b の終点は v と r を結ぶパス上にある. 重みの付け方に注意すると, b はそのような枝の中で終点がもっとも v に近いものになる. よって, これが非後退枝とならなければ非後退枝は存在しない.

次に計算時間を示す. w の発見, そのためのデータ構造の更新に $O(\log|V|)$ 時間かかる. また, **2:**, **3:**, **4:**は, 枝を消去・重みの変更をしたときに, 最小全張木を更新するのにかかる時間を必要とする. **2:**, **3:** の繰り返し回数は, (非後退枝になるものの数 +1) 回である.∎

このアルゴリズムにより, 有向根付き木の列挙は 1 つあたり最小全張木の更新時間, 即ち $O(|V|^{1/2}\log(|A|/|V|))$ で行える.

参考文献

[1] H.N.Kapoor and H.Ramesh, "Algorithms for Generating All Spanning Trees of Undirected, Directed and Weighted Graphs, " in *Lecture Notes in Computer Science*, Springer-Verlag, 461-472, 1992.

[2] A. Shioura, A. Tamura and T. Uno, "An Optimal Algorithm for Scanning All Spanning Trees of Undirected Graphs, " SIAM J. Comp., to be appeared.

[3] D.Eppstein, Z. Galil, G.F.Italiano and A.Nissenzweig, "Sparsification - A Technique for Speeding up Dynamic Graph Algorithms, " FOCS 33, 60-69, 1992.

[4] G. N. Frederickson, "Data Structure for On-line Updating of Minimum Spanning Trees, with Applications, " SIAM J. Comp., 14, No 4, 781-798, 1985.