アルゴリズム 8－1
（1989. 6. 23）

推移的閉包を求めるアルゴリズムのならし計算量（II）

平田富夫† 島谷隆司‡ 稲垣康善†

† 名古屋大学工学部 ‡ KDD

グラフGに辺を挿入する、あるいは除去するという操作を施しながら、何らかの質問（query）に答えるダイナミックなデータ構造がいくつか提案されている．筆者らは、先に、有向グラフGに辺の挿入と除去を繰り返し行うときのGの推移的閉包をオンラインで求めるアルゴリズムを提案し、そのならし計算量を解析した．本論文では、このアルゴリズムの改良版を提案する．これにより、Gに閉路が生じる場合も取り扱えるようになった．改良されたアルゴリズムは、Gがn個の頂点を持つとき、m回の辺の挿入とd回の辺の除去をO((d＋1)mn＋(m－d)n)時間で実行する．

Amortized Complexity of a Transitive Closure Algorithm(II)

Tomio HIRATA,† Takashi SHIMATANI‡ and Yasuyoshi INAGAKI†

† Faculty of Engeneering, Nagoya University, ‡ KDD

Given a directed graph G, suppose that edges are added to or deleted from G, and we want to compute its transitive closure in the on-line manner. This paper presents an improved version of the previously proposed algorithm. The result of the paper is as follows: the on-line computation of the transitive closure of G with n nodes, while m eddges are added to and d eddges are deleted from G, can be done in O((d+1)mn+(m-d)n) time. The new algorithm works even if an cycle occures in G, which is forbidden in the original algorithm.

１．まえがき

グラフに辺を挿入する、あるいは除去するという操作を施しながら、何らかの質問(query)に答えるダイナミックなデータ構造が幾つか提案されている.[2-6]筆者らは、先に、有向グラフＧに辺の挿入と除去を繰り返し行うときのＧの推移的閉包をオンラインで求めるアルゴリズムを提案した.[10]本論文では、その改良版を与える.

有向グラフの推移的閉包をオンラインで求める問題に関しては、Ibaraki and Katohらが次の二つの結果を得ている.[4]

(i)　ｎ個の頂点を持つ有向グラフＧに辺の挿入を行いながらその推移的閉包をオンラインで計算する問題に対し、挿入される辺の本数にかかわらず$O(n^3)$時間で実行するアルゴリズムが存在する．また、

(ii)　ｎ個の頂点、ｍ本の辺を持つ有向グラフＧから辺の除去を行いながらその推移的閉包をオンラインで計算する問題に対し、除去される辺の本数にかかわらず$O(n^2(m+n))$時間で実行するアルゴリズムが存在する.

最近、Italianoは工夫したデータ構造を用いることにより次の結果を得た.[5,6] (i)の問題に対し、初期においては辺のないグラフから始めるとき、１回の辺の挿入を$O(n)$のならし時間(amortized time)で実行するアルゴリズムを与えた．また、(ii)の問題に対しては、初期においては閉路のないグラフから始めるとき、任意回数の辺の除去を$O(mn)$時間で実行するアルゴリズムを与えた．なお、Italianoのアルゴリズムでは、Ｇの推移的閉包が求まるだけでなく、任意の２点を結ぶ路があるときにはその路の一つを（複数ある場合はそのうちの一つを）その長さに比例した時間で返すことができる.

筆者らは、先に、ｎ個の頂点を持つ辺のない有向グラフＧに辺の挿入と除去を繰り返し行うとき（ただし、閉路が生じるような辺の挿入は許さない)、Ｇの推移的閉包をオンラインで求めるという問題に対し、ｍ回の挿入とｄ回の除去を$O((d+1)mn+(m-d)n)$時間で実行するアルゴリズムを提案した.[10]本論文では、Ｇに閉路が生じる場合にも対応できるように、このアルゴリズムを改良する.

以下、２章においては本論文で必要となる基本事項を説明する．３章でデータ構造とアルゴリズムについて述べ、４章でそのならし計算量を解析する（これらの章では文献(10)の誤りが訂正されている)．５章で改良箇所を解説する．６章はまとめである.

２．準備

Ｇ＝（Ｖ，Ｅ）を有向グラフとする．Ｖは頂点の集合、Ｅ（$\subset V\times V$）は辺の集合である．辺ｅ＝（ｖ，ｗ）に対し、頂点ｖをｅの始点と呼び、頂点ｗをｅの終点と呼ぶ．頂点v_1から頂点v_nへ至る長さｎ－１の路とは辺の列$(v_1,v_2),(v_2,v_3),\cdots,(v_{n-1},v_n)$である．頂点ｕから頂点ｖへ至る路があるとき、ｖはｕから到達可能である、または、ｖはｕの子孫であるという．頂点ｘの子孫の集合をdesc(x)と記述する.

グラフＧ＝（Ｖ，Ｅ）と同じ頂点集合をもつ有向グラフＧ'で、Ｇにおいてｖからｗへの路があるとき、またそのときに限ってＧ'に辺(v,w)があるようなものをＧの推移的閉包と呼ぶ.

本論文で取り扱う問題を定式化するため、Ｇ上の３つの操作を定義する.

(i) add(i,j)：　Ｇに辺(i,j)を挿入する．ただし閉路を生ずるような辺の挿入はないものとする.

(ii) delete(i,j)：　Ｇから辺(i,j)を除去する.

(iii) searchpath(i,j)：　Ｇにおいて頂点ｉから頂点ｊに至る路の有無を調べ、もしあればその路（複数ある時はそのうちの一つ）を返す.

本論文では、ｎ個の頂点を持つ辺のない有向グラフＧから出発し、Ｇに上記の３つの操作からなる操作列ρを実行するという問題を取り扱う.

３．データ構造

ここでは、先に提案したデータ構造について、その概略を述べる．詳細は文献(10)を参照されたい．そのデータ構造は、Italianoが文献(5)と(6)で提案したデータ構造を基本として、それを改良したものである．その基本的なアイデアは、各頂点$x\in V$に対しdesc(x)を根付き木で表すというものである．この根付き木は頂点集合がdesc(x)で、辺集合はＥの部分集合である．なお、以下では、集合desc(x)とdesc(x)を表す根付き木を区別しないことがある.

次のように定義されるｎ×ｎの正方行列を２次元配列で実現し、操作列ρのもとで保持する.

$$\mathrm{index}(x,v)=\begin{cases}\text{根付き木desc(x)の頂点ｖへのポインタ} & (v\in \mathrm{desc}(x)\text{のとき})\\ \mathrm{null} & (v\notin \mathrm{desc}(x)\text{のとき})\end{cases}$$

根付き木desc(x)の各頂点は子に向かうポインタと親に向かうポインタを持つ．searchpath操作はつぎのように実行される．頂点ｉから頂点ｊへ至る路の有無は

index(i,j)を調べるだけなので、定数時間で判定できる．index(i,j)がnullでないときには、その指し示す頂点から根に向けての探査を行うことによりｉからｊへ至る路を見つける．よって次の補題を得る．

［補題１］　searchpath操作はＯ(ℓ)時間で実行できる．ただしℓは返す路の長さである．

次に、delete(i,j)を実行する手順を説明する．まず、辺(i,j)の現れる根付き木を見つけ、そこから辺(i,j)を除去する．この処理は各ｘ∈Ｖについてindex(x,j)の値を調べ、nullでないときdesc(x)におけるｊの親がｉであるか否かを調べればよいので、Ｏ(ｎ)時間で実行できる．辺(i,j)を根付き木desc(x)から除去したとき、ｊがなおｘから到達可能であるかという問題が残る．つまり、辺(i,j)を通らずにｘからｊへ至る路がＧにあるならば、頂点ｊはdesc(x)に残さなければならないし、そのような路がＧになくなるなら、頂点ｊはdesc(x)から除去しなければならない．これを効率よく実行するためにdesc(x)の各頂点ｗについて次の集合を双方向リストで実現し保持する．

in(x,w)＝｛v∈desc(x)｜(v,w)∈Ｅ　かつ　(v,w)はdesc(x)の枝ではない｝

in(x,w)はＧにおいてｗに入る辺の始点のうち、desc(x)の要素であるが、根付き木desc(x)におけるｗの親ではないものの集合である．以下では、集合in(x,w)とそのリストとを区別しない．また、根付き木desc(x)の各頂点ｗはリストin(x,w)の先頭を指すポインタを持つ．

辺(i,j)をdesc(x)から除去した時、in(x,j)≠φであったとする．このときin(x,j)の要素の一つｋを任意に選び、desc(x)に辺(k,j)を挿入する．このｋを（desc(x)における）ｊのフック（hook）と呼び、ｊはｋにフックするという．in(x,j)＝φであったとすると、Ｇにおいてｊはもはやｘから到達可能ではないので、desc(x)からｊを除去する．さらに、desc(x)におけるｊの全ての子供ｓについて辺(j,s)がdesc(x)から除去されたものとして、上と同様の処理を再帰的に繰り返す．リストin(x,j)の更新を効率よく行うために、次の集合を双方向リストで実現し保持する．

out(x,v)＝｛w∈desc(x)｜(v,w)∈Ｅ　かつ　(v,w)はdesc(x)の辺ではない｝

以下では、集合out(x,v)とそのリストを区別しない．根付き木desc(x)の各頂点vはリストout(x,v)の先頭を指すポインタを持つ．これらの双方向リストの導入により、一つのデータ構造でaddとdeleteの両操作が可能となっている．

アルゴリズムを簡潔に記述するために、次の基本操作を定義する．

(1) vin(x,v)：in(x,v)が空の時はnullを返す．空でないときはその先頭の要素を返す．

(2) vout(x,v)：out(x,v)が空の時はnullを返す．空でないときはその先頭の要素を返す．

(3) insert(x,u,v)：in(x,v)にｕを挿入し、out(x,u)にｖを挿入する．

(4) del(x,u,v)：in(x,v)からｕを除去し、out(x,u)からｖを除去する．

(5) cut(x,v)：根付き木desc(x)から頂点ｖを除去し、index(x,v)をnullにする．

基本操作insert(x,u,v)は、desc(x)を有向グラフとみて辺(u,v)を挿入することに相当する．（挿入された辺(u,v)は根付き木の辺ではない．）　del(x,u,v)は辺(u,v)を削除することに相当する．

さらに次のような３次元配列pinとpoutを用意する．これによって、上記の５つの操作はそれぞれ定数時間で実行できる．

pin(x,i,j)：リストin(x,i)中のjへのポインタ

pout(x,i,j)：リストout(x,i)中のｊへのポインタ

なお、indexを含めこれらの配列の初期化は文献(1)演習問題2.12の方法により、Ｏ(ｍｎ)時間で実行できることに注意されたい．delete(i,j)を実行する手続きを図１に示す．なお、この手続きの正当性はＧに閉路がないことから明らかであろう．

最後に、add(i,j)を実行する手順について説明する．もし、i∉desc(x)であれば、根付き木desc(x)は影響を受けない．i,j∈desc(x)の場合には、desc(x)に新たに加わる頂点はなく、in(x,j)とout(x,i)を更新するだけでよい．　i∈desc(x)かつj∉desc(x)の場合には、根付き木desc(x)に新たにいくつかの頂点が加わる．まず、desc(x)において頂点ｉの子供として頂点ｊを挿入する．さらに、desc(j)における頂点ｊのそれぞれの子供ｗについて、ｗ∉desc(x)であればdesc(x)の頂点ｊの子供として挿入する．これを再帰的に繰り返す．このとき、集合inと集合outの更新も行われる．add(i,j)を実行する手続きを図２に示す．

４．アルゴリズムの解析

ならし計算量解析の一つの基本的な方法は、データ構造の状態にポテンシャルと呼ばれる実数Φを対応づけるものである．初期状態D_0のデータ構造に対し操作

```
procedure delete(i,j);
begin
  for each x∈V do
    if index(x,i)≠null and index(x,j)≠null
      then
      if desc(x) における j の親が i である
        then begin
             desc(x) から辺 (i,j) を取り除く;
             findhook(x,j)
           end
        else del(x,i,j)
end;

procedure findhook(x,j);
begin
  if vin(x,j) = null
    then begin
      for each w∈out(x,j) do del(x,j,w);
      for each child s of j in desc(x) do
        begin
          desc(x) から辺 (j,s) を取り除く;
          findhook(x,s)
        end
      cut(x,j)
      end
    else begin
      u:=vin(x,j);
      desc(x) に辺 (u,j) を加える;
      del(x,u,j)
      end
end;
```

図1　手続きdelete(i,j)

```
procedure add(i,j);
begin
  for each x∈V do
    begin
      if index(x,i)≠null then
        if index(x,j)≠null then insert(x,i,j)
          else meld(x,j,i,j)
    end
end;

procedure meld(x,j,u,v);
begin
  頂点 v を desc(x) の u の子供として挿入する;
  index(x,v) がその v を指すように設定する;
  for each child w of v in desc(j) do
    if index(x,w)≠null then insert(x,v,w)
      else meld(x,j,v,w);
  for each k∈out(j,v) do insert(x,v,k)
end;
```

図2　手続きadd(i,j)

列$op_1, op_2, \cdots, op_k$を施し、状態が順に$D_1, D_2, \cdots, D_k$と変化した状況を考える．各D_i（$1 \leqq i \leqq n$）のポテンシャルをΦ_iと記述する．操作op_iの実コストがt_iのとき、op_iのならしコスト(amortized cost)a_iを次式で定義する．

$$a_i = t_i + \Phi_{i+1} - \Phi_i \qquad (1)$$

するとt_iの総和は次のようになる．

$$\sum_{i=1}^{k} t_i = \sum_{i=1}^{k} a_i + \Phi_0 - \Phi_k \qquad (2)$$

すなわち、実コストの総和はならしコストの総和とポテンンシャルΦの減少量との和に等しい．したがって、(2)式の右辺を計算することによって、操作列の計算時間を求めることができる．

右辺のΣが計算しやすいようにΦを定めるのがならし計算量解析の重要な点である．ここでは、G＝（V, E）に対応するデータ構造Dのポテンシャルを以下のように定める．

［定義１］　Gの各頂点xについて集合VIS(x)を次のように定義する．

$VIS(x) = \{ (u,v) \in E \mid u \in desc(x) \}$

［定義２］　Gの頂点xのポテンシャル$\varphi(x)$を次式で定める．

$\varphi(x) = |desc(x)| - 5|VIS(x)|$

［定義３］　データ構造Dのポテンシャルを次のように定義する．

$$\Phi = \sum_{x \in V} \varphi(x)$$

このとき、次の補題２～補題４が成立する．

［補題２］　１回のadd操作のならしコストはO(n)である．

（証明）　図２の手続きadd(i,j)において、７行目の手続き呼び出しmeld(x,j,i,j)を除いた実行時間がO(n)であるのは明かである．meld(x,j,i,j)が実行されると根付き木desc(x)が更新される．このとき、実行される主な処理は、根付き木desc(j)の探査と根付き木desc(x)への頂点と辺の挿入、それに根付き木desc(x)におけるinとoutの更新である．手続きmeldの３行目の実行によりdesc(x)に木の辺として挿入される辺の本数をh_0、６行目のinsertで挿入される辺の本数をh_1、８行目のinsertで挿入される辺の本数をh_2とする．

根付き木desc(j)の探査で調べられた辺は、desc(x)に木の辺として挿入されるかまたは手続きmeldの６行目でinsertされる．したがって、１本の辺の探査が１

単位コストで実行されるものとすると、desc(j)の探査に要するコストはh_0+h_1である.

desc(x)に木の辺(u,v)を挿入するとき、次の３つのステップが実行される.

(s1) ｖをdesc(x)におけるｕの子供として挿入する.

(s2) ｖからｕへのポインタを用意する.

(s3) index(x,v)がdesc(x)のｖを指すようにする.

これらの各ステップが１単位コストで実行されるものとすると、desc(x)に木の辺を挿入するのに要するコストは$3h_0$である.

insertの実行では双方向リストinとoutにそれぞれ要素を挿入するので、これは４単位コストを要するものとする. また、手続きmeldの８行目のinsertの実行では、insert(x,v,k)の実行前にout(j,v)からkを読み出す操作が実行されるが、これは１単位コストで行われるものとする. すると、６行目と８行目のinsertを実行するのに要するコストは、$4(h_1+h_2)+h_2$となる.

したがって、desc(x)の更新に要する実コストは、

$$(h_0+h_1)+3h_0+(4(h_1+h_2)+h_2)$$
$$=4h_0+5(h_1+h_2) \qquad (3)$$

となる.

次に、desc(x)の更新に伴うポテンシャル$\varphi(x)$の変化を考える. desc(x)に加えられる頂点の個数はh_0であり、|vis(x)|の値は$h_0+h_1+h_2$だけ増加する。よって、$\varphi(x)$は、

$$5(h_0+h_1+h_2)-h_0$$
$$=4h_0+5(h_1+h_2) \qquad (4)$$

だけ減少する. 式(3)と式(4)より、meld(x,j,i,j)の実行に対するならしコストは０となり、add操作１回あたりのならしコストはO(n)となる. □

次に、delete操作のならしコストを考える.

［補題３］ 途中にadd操作を挟まないｄ回の連続したdelete操作のならしコストはO(mn)である.

（証明） まず、一つの根付き木desk(x)について、連続したdelete操作の実コストがO(m)であることを示す. 図１の手続きdelete(i,j)の、９行目の手続き呼び出しfindhook(x,j)を除いた実行時間はO(d)であり、したがてO(m)である. findhook(x,j)の実行中に探査されたdesc(x)の辺はすぐに除去されるので（手続きfindhookの８行目)、ｄ回のdelete操作を実行する間にdesc(x)の木の各辺はたかだか１回しか探査されない. 手続きfindhookの下から４行目で新たに木の辺が付加されるが、その総数は、ｍ本以下である. なぜなら、

$$\left|\bigcup_{v\in desc(x)} in(x,v)\right| \leqq m$$

だからである. さらに、

$$\left|\bigcup_{v\in desc(x)} out(x,v)\right| \leqq m$$

より、この他の主な仕事である基本操作del(x,u,v)の実コストもO(m)である. 更新される根付き木はたかだかｎ個であるから、delete操作の実コストはO(mn)となる.

次に、頂点ｘ($\in V$)のポテンシャル$\varphi(x)$の変化を考える. ｄ回のdelete操作の実行中、|desc(x)|が増加することはなく、しかも|VIS(x)|は高々ｍ減るだけなので、頂点ｘのポテンシャル$\varphi(x)$の増加量は高々O(m)である. 従って、データ構造全体のポテンシャルの増加は高々O(mn)である. よって、実コストと合わせて、ｄ回の連続したdelete操作のならしコストはO(mn)である. □

［補題４］ 辺のないグラフGから始めて、add操作をｍ回、delete操作をｄ回実行したとき、ポテンシャルΦの減少量は高々O((m－d)n)である.

（証明） Φの定義より明らか. □

補題１～補題４より次の定理を得る.

［定理１］ 操作列ρに含まれるadd操作がｍ回、delete操作がｄ回で、これらのｄ回のdelete操作は途中にadd操作を挟まないｑ個の連続した部分に分割されるとする. ｎ個の頂点を持ち、辺のない有向グラフGに操作列ρを施すとき、途中でGに閉路が生じないならば、それぞれのsearch path操作はO(ℓ)時間（ℓは返す路の長さ）で実行され、ｍ個のadd操作とｄ個のdelete操作がO((q＋1)mn＋(m－d)n)時間で実行される. また、その際に使用する記憶領域はO(n^3)である.

（証明） ｍ回のadd操作のならしコストはO(mn)であり、delete操作のならしコストはO(qmn)である. また、ポテンシャルの減少量は高々O((m－d)n)である. ｍ回のadd操作とｄ回のdelete操作を実行するに要する時間は、これらのならしコストとポテンシャルの減少量の和なので、O((q＋1)mn＋(m－d)n)となる. 使用記憶領域は、pinとpoutを３次元配列で実現するためO(n^3)となる. □

［系１］ ｎ個の頂点を持ち辺のない有向グラフGに、前述の３つの操作からなる操作列ρを施すとき、途中でGに閉路が生じないならば、ｍ回のadd操作とｄ回のdelete操作がO((d＋1)mn＋(m－d)n)時間で実行される.

５．アルゴリズムの改良

前章のアルゴリズムは、Ｇに閉路が生じないという制約がついている．本章ではこの制約を除去する．前章のアルゴリズムでこの制約を要請していたのは図１の手続きdelete(i,j)である．Ｇに閉路があると、findhook(x,j)の実行において頂点ｊは自分の子孫にフックしてしまう可能性があるからである．そこで、delete(i,j)を図３のように変更する．図３のdelete(i,j)では、辺(i,j)を根付き木desc(x)から除去した後、手続きsearch(x)により根ｘから深さ優先探索を行い、根付き木desc(x)を再構成している．その後、手続きeliminate(x,j)により、根ｘから到達不可能となった頂点をdesc(x)から削除している．次の補題の証明は補題３と同様なので省略する．

［補題５］　図３の手続きdelete(i,j)を一回実行するときのならしコストはＯ（ｍｎ）である．

補題１～補題３と補題５より次の定理を得る．

［定理２］ｎ個の頂点を持ち辺のない有向グラフＧから始めるとき、ｍ回のadd操作とｄ回のdelete操作がＯ((ｄ＋１)ｍｎ＋(ｍ－ｄ)ｎ)時間で実行できる．

```
procedure delete(i,j);
begin
  for each x∈V do
    if index(x,i)≠null and index(x,j)≠null
    then
      if desc(x)におけるｊの親がｉである
        then begin
          desc(x)から辺(i,j)を取り除く;
          search(x)
          if j is not marked then eliminate(x,j)
        end
        else del(x,i,j)
end;

procedure search(v);
begin
  mark v;
  for each child w of v in desc(x) do
    if w is not marked then search(w);
  for each w∈vout(v) do
    if w is not marked then
       begin
         if w≠j then begin
               desc(x)から辺(p(w),w)を取り除く;
               insert(x,p(w),w);  {p(w)はwの親}
             end
         desc(x)に辺(v,w)を加える;
         del(x,v,w);
         search(w)
       end
end;

procedure eliminate(x,v);
begin
  cut(x,v);
  for each child s of v in desc(x) do
    begin
      desc(x)から辺(v,s)を取り除く;
      if s is not marked then eliminate(x,s)
    end
end;
```

図３　手続きdelete(i,j)（改良版）

６．むすび

有向グラフＧの推移的閉包をオンラインで求めるアルゴリズムの改良を行った．この改良により、途中でＧに閉路が生じる場合も取り扱えるようになった．

文　献

(1) A. V. Aho, J. E. Hopcroft and J. D. Ullman: "The Designand Analysis of Computer Algorithms," Addison-Wesley, Reading, MA(1974).

(2) S. Even and Y. Shiloach: "An on-line edge deletion problem," J. Assoc. Comput. Mach. 28, pp. 1-4(1981).

(3) G. F. Frederickson: "Data structures for on-line updating of minimum spanning trees," Proc. 15th ACM Symp. on Theory of Computing, pp. 252-257(1983).

(4) T. Ibaraki and N. Katoh:"On-line computation of transitive closures for graphs," Inform. Process. Lett., 16, pp. 95-97(1983).

(5) G. F. Italiano: "Amortized efficiency of a path retrieval data structure," Theoret. Comput. Sci., 48, pp. 273-281(1986).

(6) G. F. Italiano: "Finding paths and deleting edges in directed acyclic graphs," Inform. Process. Lett., 28, pp. 5-11(1988).

(7) D. D. Sleator and R. E. Tarjan: "Amortized efficiency of list update and paging rules," Commun. ACM, 28, 2, pp. 202-208(1985).

(8) R. E. Tarjan and J. van Leeuwen: "Worst-case analysis of set union algorithms," J. Assoc. Comput. Mach., 31, pp. 245-281(1984).

(9) R. E. Tarjan: "Amortized time complexity," SIAM J. Alg. Disc. Meth., 6, pp. 306-318(1985).

(10) 島谷、平田、稲垣："推移的閉包を求めるアルゴリズムのならし計算量,"信学技報 comp88-99, pp. 21-27(1989).