1995年度日本オペレーションズ・リサーチ学会
秋季研究発表会

補グラフ入力に対する線形時間グラフ探索アルゴリズム

01009550　NTT通信網研究所　伊藤大雄　ITO Hiro

1．まえがき

アルゴリズムの計算時間は、計算量理論の観点から普通は線形時間が最良であるとされる。グラフ上の問題に関しても入力の線形時間で解ける重要な問題が色々発見されている。また最近、線形時間で解ける問題のクラスが定義される[1]など、線形時間解法の重要性が注目されている。

無向単純グラフG=(V,E)はその補グラフH=(V,B)を用いても表現が可能である。Gを表現するのには$\Theta(n+m)$、Hを表現するのには$\Theta(n+M)$のデータ量が必要十分である（但しn=|V|, m=|E|, M=|B|）。mとMとの間にはm+M=n(n-1)/2の関係がある、即ちmとMとは互いに線形オーダーでは押さえられない。よって、補グラフのデータを入力することによってグラフを表現するならば、これまで線形時間で解けるとされてきた問題が線形時間で解かれるとは限らなくなる。そこで本稿では、単純グラフに対する既存の線形時間アルゴリズムを、補グラフ入力に対してもなお線形時間を保てる様に改良することを試みる。

グラフのアルゴリズムにおける操作としてグラフの探索がある。代表的なものに幅優先探索(BFS; breadth-first-search)と深さ優先探索(DFS; depth-first-search)がある。どちらもグラフ全体の探索がO(n+m)時間ででき、グラフの線形時間アルゴリズムはこのどちらかを用いていることが多い。補グラフ入力に対するBFSとDFSについてはKao and Teng [2]によって一部取り上げられているが、線形時間での探索は達成されていなかった。筆者はこの二つの探索法について、補グラフ入力上で線形時間で行なえることを発見したのでここに報告する。本稿ではより難しい方のDFSについてのみ解説する。これをBFSに焼き直すのは容易である。詳細は文献[3]を参照されたい。なお本稿では無向単純グラフに基づいて説明するが、同様の議論が有向単純グラフにも適用できる。

2．諸定義

無向単純グラフG=(V,E)。Vは節点集合、Eは枝集合。各節点には連番が付いているものとする、すなわち節点集合をV={1,2,…,n}とする。枝はその両端点を用いてe(i,j)の様に表すこともできる。補グラフの枝を補枝と呼び、補枝集合Bを

$$B:=\{b(i,j) \mid e(i,j)\notin E,\ i,j\in V,\ i\neq j\}$$

で定義する。枝e(i,j)の両端点i,j (i<j)は、term1(e(i,j))=i, term2(e(i,j))=jで表す。補枝についても同様である。ある補枝b(i,j)が存在するとき、i,jは互いに補接すると呼ぶ。

3．深さ優先探索

本節では補グラフ入力に対し、深さ優先探索を線形時間で行なう方法を述べる。出力は深さ優先探索木Tである。まず前処理として各節点に隣接する補枝のリストblancを、次の性質を持つように作成する。

dimension blanc0(n), blanc(2,M): 節点に接続する補枝のリスト。但しblanc0(i)は節点iに接続する補枝のうちiでない方の端点の番号が最も小さい補枝を指し、blanc(1,b)はterm1(b)に接続する補枝b'のうちterm1(b)でない方の端点の番号がterm2(b)の次に小さい補枝を指し、blanc(2,b)はterm2(b)に接続する補枝b'のうちterm2(b)でない方の端点の番号がterm1(b)の次に小さい補枝を指す。

各補枝bに対するterm1,term2のデータから上記の性質を持つリストblancを作成するのは、バケットソートを２度適用することでO(n+M)時間で可能である[3]。なお、各節点iに補接するダミーの補枝b(i,0)とb(i,n+1)を加えておく。

次に木Tに選ばれていない節点のリストfreeを番号の小さい順で作成しておく。

dimension free(0;n): 木に選ばれていない節点のリスト。
初期値は free(0)=1, free(i)=i+1とする。

そして節点iがTに選ばれている時に、j≧iでTに選ばれていない最小の節点jを得る関数find(i)を定義しておく。即ち

function find: find(i)=min{j | j≧iかつ節点jはTに選ばれていない}, i=0,1,2,...,n

但し全てのj≧iがTに選ばれている場合はfind(i)=n+1とする。また、ダミーの補枝b(i,0)に対する処理のため、ダミーの節点0（始めからTに選ばれていると解釈する）も定義しておく。

find(i)の実現法及びそれに要する計算時間の考察はアルゴリズムの計算量の解説のところで述べる。

```
procedure COMPL-DFS
dimension tree(0;n): 木に選ばれた節点のリスト
  初期値は: free(0)=free(i)=0
dimension scan(n): scan(i)=jは、節点iの走査を、次は補枝
b(i,j)から始めれば良いことを表す。初期値はscan(i)=0（ダ
ミーの補枝b(i,0)が存在することに注意。）
  begin
1   i0:=free(0)
2   リストfreeからi0を除き、リストtreeにi0を加える。
3   do while i0≠0
4     js:=scan(i0)
5     jf:=find(js)
```

```
6     do while jf≦n
7       if js<jf then
8         j0:=js
9         リストblancにより、i0の、jsの次の補接節点を
          求め新たにjsとする。
10      elseif js=jf then
11        j0:=js
12        リストblancにより、i0の、jsの次の補接節点を
          求め新たにjsとする。
13        jf:=free(jf)
14      elseif js>jf then
15        リストfreeからjfを除き、リストtreeのi0の次にjf
          を加える。
16        scan(i0):=j0
17        goto line 20
18      fi;
19    od;
20    i0:=tree(i0)
21  od;
  end
```

アルゴリズムの解説をする。i0は現在走査中の節点を意味する。i0に補接する（すなわち隣接しない）節点をjsで表し、その時点で木に含まれていない節点をjfで表す。i0を走査する時の手順を解説する。深さ優先探索なのでi0を一気に走査しないので、これまでにどの補接節点まで走査が進んでいてどこから次の走査を実行するべきかを、scan(i0)に記憶しておく。（初期値はscan(i0)=0、すなわちリストの最初のダミーの補接節点を指している。）第4行でそのデータをjsに代入し、第5行でjsがTに選ばれている場合は、Tに選ばれていない、js以上の最小の節点を選んで（find(js)）、改めてそれをjsとおき直す。すなわち、js未満の節点はi0の隣接節点としてはもはや調べる必要が無いことを意味する。よって、js≧jfとなるまでjsをリストblancを用いて更新していく（第7-9行）。js=jfとなったならば、その節点はTに含まれてはいないが、補接しているので、i0のTにおける子とはなり得ず、js, jf共に更新する（第10-13行）。js>jfとなった時、jfがi0に隣接しかつTにまだ含まれていないのでTにおけるi0の子とすることが出来る（第14, 15行）。そこで深さ優先探索の原則に基づき走査をi0から最も新しくTに選ばれた節点jf(=tree(i0))にうつす（第20行）が、その前にscan(i0)に現在調べたところまでを記憶しておく（第16行）。scan(i0):=jsとせずにjsの一つ前の補接節点j0を入れるのは、jfとjsの間の隣接節点が跳ばされないない様にする為である。以上からCOMPL-DFSが深さ優先探索を正しく実行することがわかる。

次に計算時間を見積る。第6-19のdo文中でjs<jfあるいはjs=jfと判定される（すなわち第8-9行あるいは第11-13行が実行される）のは、全アルゴリズムを通じて高々2M+n回である（nは、第16行でscan(i0):=j0としたので節点数回の重なりが生ずることから）。js>jfと判定される（すなわち第15-17行が実行される）のは全体で高々n回である。よって第6-19行の操作はアルゴリズム全体でO(n+M)時間でできる。のこるは第5行のfind(js)であるが、これはUNION-FINDのアルゴリズム[4]を利用する。すなわち、初期状態では節点一つ一つが各々要素数1の集合を構成していて、節点iがTに選ばれると同時に、iを含む集合はi+1を含む集合と併せられる（UNION）とする。集合の代表節点はその集合の最大節点とすると、find(js)はそのままjsを含む集合の代表節点を求める操作（FIND）となる。するとCOMPL-DFSを通じてUNIONとFINDは各々高々n回しか呼び出されないので、Gabow and Tarjan [4]で提案されたUNION-TREEを用いれば、これらの操作は全体でO(n)時間で実行できることがわかる。以上からCOMPL-DFSがO(n+M)時間で実行できることが示された。すなわち、補グラフ入力に対し深さ優先探索は線形時間（O(n+M)時間）で出来る。

4．まとめ

本稿ではDFSが、補グラフ入力に対しても線形時間で出来ることを示した。BFSもDFSを焼き直すことによって簡単に線形時間アルゴリズムが得られる[3]。この二つの探索アルゴリズムは多くのグラフ上のアルゴリズムの基本となっており、これらが線形時間で出来たことによって、様々なグラフ上の線形時間アルゴリズムが補グラフ上でも線形時間で出来る見込みが出てきた。BFSを改良すれば、文献[5]で提案されたk-連結性を保存する枝数O(kn)の全域部分グラフが線形時間で得られる。このことからk≦3に対して、k-点連結成分、あるいはk-枝連結成分の作成が補グラフ入力においても線形時間で行なえることになる。また平面性判定が線形時間で行なえることは自明である。今後の課題としては、グラフ入力上では線形時間で出来るが、補グラフ入力上では線形時間では出来ない様な問題が存在するかという問題が興味が持たれる。

謝辞　貴重な御助言を頂いた、京都大学工学部の茨木俊秀教授ならびに永持仁 助教授に感謝いたします。

参考文献 [1] E. Grandjean, "Linear Time Algorithms and NP-Complete Problems," SIAM J. Comput., Vol. 23, No. 3, pp. 573-597 (1994). [2] Ming-Yang Kao and Shang-Hua Teng, "Simple and Efficient Graph Compression Schemes for Dense and Complement Graphs," Proceedings of ISAAC'94, LNCS #834, pp.451-459 (1994). [3] 伊藤大雄, "補グラフ入力に対する線形時間グラフ探索アルゴリズム" 信学技報, Vol. 95, No. 82, COMP95-12, pp. 9-16 (1995). [4] H. N. Gabow and R. E. Tarjan, "A Linear-Time Algorithm for a Special Case of Disjoint Set Union," J. of Computer and System Sciences, Vol. 30, pp. 209-221 (1985). [5] A. Frank, Nagamochi H. and Ibaraki T., "On Sparse Subgraphs Preserving Connectivity Properties," J. of Graph Theory, Vol. 17, No. 3, pp. 275-281 (1993).