概略設計書

作成者　　築山　修治　　作成日　2012 年 10 月 1 日

関数名： $Lca(\text{int}\quad T, m)$

概要　（どのような入力に対して，どのような出力をするかの概要説明）

＊ 木 T および質問点対の集合 P が与えられたとき，各質問点対 $p = (v,w) \in P$ の最下位共通先祖（すなわち木 T において点 v と w の共通の先祖 a で，a の真の子孫には v と w の共通の先祖が無いような点）を見出す関数である．
＊ 整数の引数 T は，木 T の構造を表す配列 *NodeSpace*[・] へのカーソル（cursor）で，木 T の根を示す．従って，木 T の各点は，点番号を持つと考えることができる．点番号は正の整数とする．
＊ 木 T は配列 *NodeSpace*[・] の中に格納されており，この配列の各要素 *NodeSpace*[*v*] は，点 v の長男へのカーソル *NodeSpace*[・].*lm_child* および次弟へのカーソル *NodeSpace*[・].*r_sibling* の 2 つの欄から成る構造体である．
＊ 整数の引数 m は質問点対の個数であり，各質問点対 $(v_p,w_p) \in P$ には $1 \le p \le m$ なる番号が付けられている．
＊ 質問点対の番号 p は，質問点対の集合を表す 2 次元配列 *QueryPair*[・][2] へのカーソルになっており，第 p 番目の質問点対 (v_p,w_p) の点 v_p および w_p は，それぞれ *QueryPair*[*p*][0] および *QueryPair*[*p*][1] に格納されている．
＊ 第 p 番目の質問点対 $p_p \in P$ の最下位共通先祖（出力）は，配列 *Ancestor*[*p*] に入れる．

入力　（入力するものの形式や制約条件を説明）

＜入力名＞	＜形式＞	＜制約条件等＞
NodeSpace[・]	グローバル変数で，2 つの欄 *lm_child* および *r_sibling* を持つ構造体の配列	木 T を格納するための配列．各要素 *NodeSpace*[*v*] の 2 つの欄 *lm_child* および *r_sibling* には，それぞれ点 *v* の長男および次弟の点番号を蓄えている
T	整数．値呼び	木 T の根の点番号（配列 *NodeSpace*[・] へのカーソル）
QueryPairs[・][2]	グローバル変数で，2 次元の配列．	質問点対の集合を格納するための配列．第 p 番目の質問点対 $(v_p,w_p) \in P$ の 2 点 v_p および w_p は，それぞれ *QueryPairs*[*p*][0] および *QueryPairs*[*p*][1] に蓄えられている
m	整数．値呼び	質問点対の個数．配列 *QueryPairs*[・][・] の *QueryPairs*[1][・] から *QueryPairs*[*m*][・] に質問点対が入っている

出力　（出力するものの形式や制約条件を説明）

＜出力名＞	＜形式＞	＜性質・特徴・条件等＞
Ancestor[・]	整数のグローバル変数	*Ancestor*[*p*] は，*QueryPairs*[*p*][・] に蓄えられた各質問点対の最下位共通先祖（点番号．*NodeSpace*[・] へのカーソル）を示す

処理概要　（処理の概要および主要なデータ）

始めに, 最下位共通先祖を求めるための関数 *LcaDFS*(int *v*) の処理を記述する. この関数は値を返さない再帰的な void 関数で, 点 *v* を根とする木 T の部分木を深さ優先探索する. 整数の引数 *v* は, 木 T の点を示す点番号で, 配列 *NodeSpace*[・] へのカーソルになっている. 関数 *Lca*(*T, m*) は, *LcaDFS*(*T*) として, この関数を呼ぶ. このとき, T は木の根を示すカーソルである.

関数 *LcaDFS*(*T*) を呼ぶ前に, 以下の初期化を終了しておく.

- 木 T の各点 *v* に対して, 未探索の印を付ける.
- 木 T の各点 *v* に対して, *v* を含む質問点対のリスト *P*(*v*) を作成する.
- 木 T の各点 *v* に対して, 集合 *U*(*v*) を記憶するためのデータ構造(MF 木)を作成する.

ここで, 各点 *v* に対する集合 *U*(*v*) = { *v* } ∪ { *d,*…} は, *v* と *v* の探索済みの子孫 *d* からなるが, *v* の探索済みの子孫 *d* でも, *v* の子孫に探索中の点 *u* があり, *d* が *u* の子孫になっているようなもの(子孫 *d*)は含まない.

```
void  LcaDFS(  int  v  )
{
    c を点 v の長男とする;
    while(  点 v の子 c が存在する  ) {
        LcaDFS(  c  );                          /* 点 v の子 c に対して LcaDFS を実行する */
        集合 U(c) と集合 U(v) を Merge し, U(v) とする;     /* U(v) := U(v) ∪ U(c) */
        c を c の次弟にする;
    }
    /* 点 v からその親に戻る際, 以下の操作を実行する */
    if(  点 v を含む質問点対のリスト P(v) は空リストでない  ) {
        for(  各質問点対 (v_p,w_p)∈P(v)  ) {
            配列 QueryPairs[・][・] から, 質問点対 (v_p,w_p) を見出す;
            if(  点 w_p は探索済みである  ) {
                点 w_p を含む集合 U(a) を Find する;
                Ancestor[p] に点 a を入れる;
            } /* end if */
        } /* end for */
    } /* end if */
    点 v に探索済みの印を付ける;
} /*  LcaDFS  */
```

このアルゴリズムによって, 各質問点対 (*v,w*)∈P(*v*) の最下位共通先祖 *a* が正しく見出される理由は, 教科書を参照せよ. この概略設計書では関数 *LcaDFS*(int *v*) の説明しかしていない. 初期化を行う関数 *DFS*1(int *T*) の概略設計書を書いてみよ. その際, プログラム設計の順番が逆だが, 以下の詳細概略設計書を参考にするとよい.

必要な関数　(この関数の処理に必要な関数)

*DFS*1(int *T*) :　関数 *LcaDFS*(*T*) を呼ぶために行う初期化関数. 深さ優先探索を利用する

LcaDFS(int *v*) :　点 *v* を根とする木 T の部分木を深さ優先探索し, 質問点対の最下位共通先祖を求める関数

Merge(int *x, y, z*) :　2 つの MF 木を併合し, 1 つの MF 木にする関数

Find(int *v*) :　点 *v* を含む MF 木を求める関数

詳細設計書

作成者　　築山　修治　　作成日　2012 年 10 月 1 日

関数名：*Lca*(int　*T*, *m*) ,　　*DFS*1(int　*T*)

主要データ構造　（関数における主要なデータ構造）

・引数の整数 *T* および *m* は値呼びで, それぞれ木 T の根および質問点対の個数を示す.

・木 T を表すグローバル変数

```
typedef  struct {
            int          lm_child ;          /*  長男の点番号. 長男が無いときは 0  */
            int          r_sibling ;         /*  次弟の点番号. 次弟が無いときは 0  */
        } NDCELL ;
NDCELL              NodeSpace[NS_length] ;   /*  NS_length は定数  */
```

・質問点対を表すグローバル変数

```
int                 QueryPairs[Q_length][2] ;   /*  Q_length は定数  */
                        /*  第 p 番目(1 ≤ p ≤ m)の質問点対 (vp,wp) の点 vp および wp は, それぞれ
                                    QueryPairs[p][0] および QueryPairs[p][0] に蓄えられている  */
```

・質問点対に対する最下位共通先祖を表すグローバル変数

```
int                 Ancestor[Q_length] ;        /*  Q_length は定数  */
```

・MF 木の点を表す構造体

```
typedef  struct {
            int          set_name ;          /*  根の場合, 集合 U(x) の点番号 x  */
            int          size ;              /*  根の場合, MF 木に含まれる点の個数  */
            MFNODE↑      parent ;            /*  親へのポインタ  */
        } MFNODE ;
```

・質問点対のリスト *P*(・) を実現する連結リストのセル（構造体）

```
typedef  struct {
            int          pair ;              /*  質問点対番号(1 ≤ p ≤ m)  */
            PCELL↑       next ;              /*  次の PCELL へのポインタ  */
        } PCELL ;
```

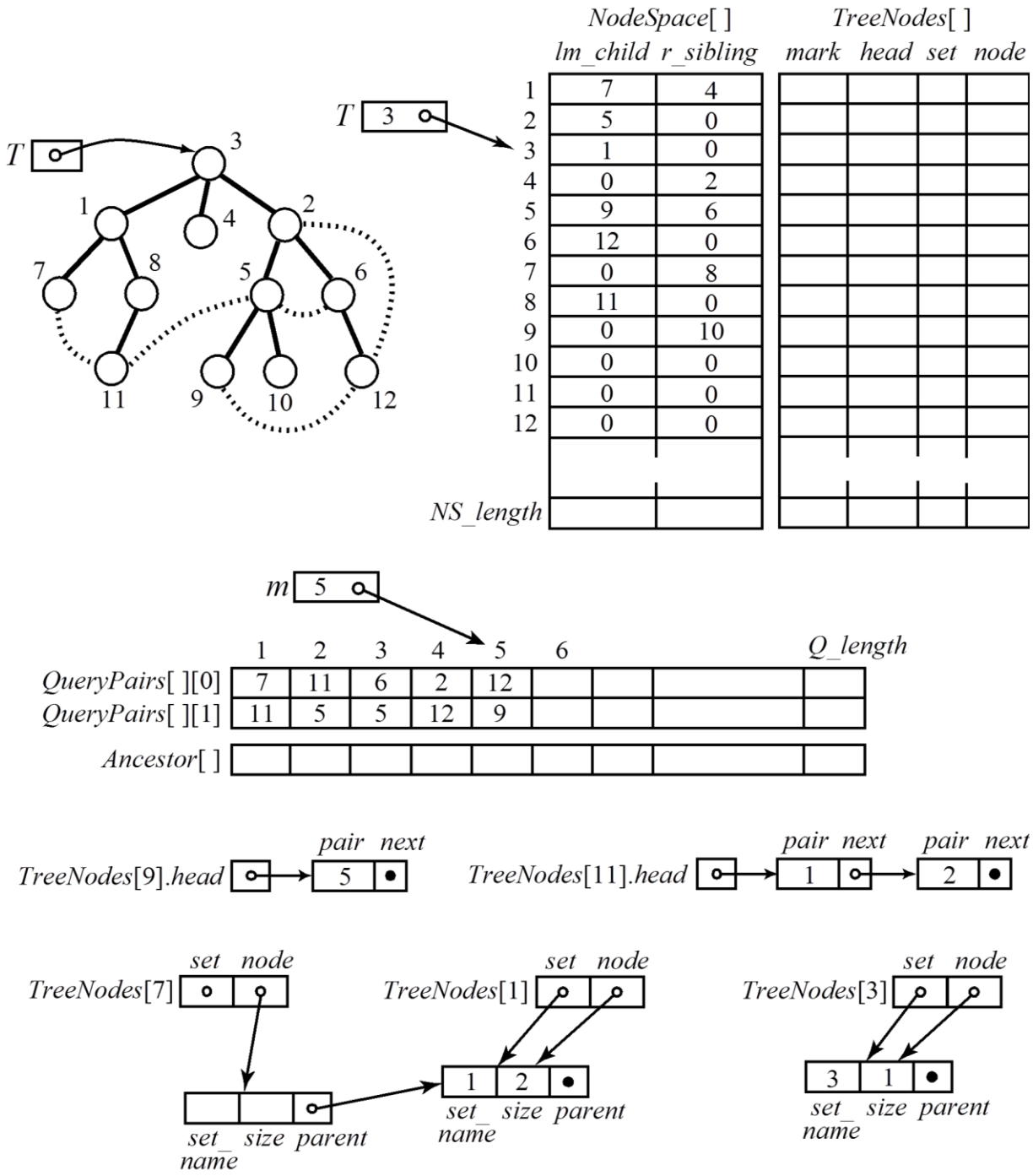
・木 T の各点に付随させておくべきデータとそれを記憶する配列(グローバル変数とする)

```
typedef  struct {
            int          mark ;              /*  探索済みか否かを示す. 0 ときは未探索  */
            PCELL↑       head ;              /*  質問点対のリスト P(・) の先頭のセルへのポインタ  */
            MFNODE↑      set ;               /*  MF 木の根(集合 U(・) に対応)へのポインタ  */
            MFNODE↑      node ;              /*  MF 木の点(木 T の点に対応)へのポインタ  */
        } TNODE ;
TNODE               TreeNodes[NS_length] ;   /*  NS_length は定数  */
```

データ構造の概略を以下の図に示す.

下の木 T に対する配列 *NodeSpace*[・], *TreeNodes*[・], *QueryPairs*[・][2], および *Ancestor*[・] の概要を示す. 木 T の図において, 点線は質問点対を示している.

また, 質問点対の集合 P(9) および P(11) を覚えている連結リスト, ならびに MF 木の一部も示す.



上の MF 木は, U(1) = { 1, 7 }, U(3) = { 3 } に関するものである.

処理の詳細　（アルゴリズムの詳細を説明）

```
void  Lca(  int  T,  int  m  )
{
    int             p,  i,  v ;          /*  p は質問点対の番号, i は繰り返し用の変数, v は点番号  */
    PCELL↑          ptrPcell ;           /*  点 v を含む質問点対のリスト P(v) のセルへのポインタ  */

    DFS1( T ) ;                          /*  配列 TreeNodes[・] の各要素の欄および MF 木を初期化する  */

    /*  木 T の各点 v に対して, v を含む質問点対のリスト P(v) を作成する  */
    for (  1 ≤ p ≤ m なる各 p  ) {       /*  各質問点対 (v_p,w_p) に対して, 以下を実行する  */
        for (  0 ≤ i ≤ 1 なる各 i  ) {   /*  質問点対 (・,・) の各点 v に対して, 以下を実行する  */
            v  :=  QueryPairs[p][i] ;
            ptrPcell  :=  new( PCELL ) ;                  /*  新たな PCELL 型のセルを作る  */
            (↑ptrPcell).pair  :=  p ;                     /*  そのセルにデータを入れる  */
            if ( TreeNodes[v].head = NULL )               /*  リスト P(v) は未だ空リストである  */
                (↑ptrPcell).next  :=  NULL ;              /*  新たに作ったセルを最後尾のセルにする  */
            else
                (↑ptrPcell).next  :=  TreeNodes[v].head  ;  /*  新たに作ったセルのリンクを付ける  */
            TreeNodes[v].head  :=  ptrPcell ;             /*新たに作ったセルを P(v) の先頭のセルにする  */
        }  /* end for  */
    }  /* end for  */

    LcaDFS( T ) ;                        /*  木 T を深さ優先探索し, 各質問点対の最下位共通先祖を求める  */
} /*  Lca  */


/*  配列 TreeNodes[・] の欄および MF 木を初期化する関数  */
void  DFS1(  int  v  )
{
    int             c ;                  /*  c は点番号で, 点 v の子を示す  */
    MFNODE↑         ptrMFnode;           /*  MF 木の点を表すセルへのポインタ  */

    TreeNodes[v].mark  :=  0 ;           /*  点 v を未探索とする  */
    TreeNodes[v].head  :=  NULL ;        /*  リスト P(v) を空リストとする  */
    ptrMFnode  :=  new( MFNODE ) ;       /*  点 v に対応する MF 木の点を作る  */
    TreeNodes[v].node  :=  ptrMFnode ;
    TreeNodes[v].set  :=  ptrMFnode ;
    (↑ptrMFnode).set_name  :=  v ;       /*  集合 U(v) の名前 set_name を v にする  */
    (↑ptrMFnode).size  :=  1 ;           /*  集合 U(v) の点の個数を 1 にする  */
    (↑ptrMFnode).parent  :=  NULL ;      /*  MF 木の点(根)の親へのポインタを NULL にする  */

    c  :=  NodeSpace[v].lm_child ;
    while(  c > 0  ) {                   /*  点 v の各子 c に対して関数 DFS1 を実行する  */
        DFS1(  c  ) ;
        c  :=  NodeSpace[c].r_sibling ;
    }
} /*  DFS1  */
```

以上

詳細設計書

作成者　　築山　修治　　作成日　2012 年 10 月 1 日

関数名： *LcaDFS*(int　*v*)

主要データ構造　（関数における主要なデータ構造）

・引数の整数 *v* は, 木 T の点 *v* を示し, 値呼びである.
・他のデータ構造は，関数 *Lca*(*T*, *m*) と同じである.

処理の詳細　（アルゴリズムの詳細を説明）

```
void  LcaDFS(  int  v  )
{
    int             c ;                     /*  c は点番号で, 点 v の子を示す  */
    int             w ;                     /*  w は点番号で, (v,w) は質問点対である  */
    int             p ;                     /*  p は質問点対番号  */
    PCELL↑          ptrPcell ;              /*  点 v を含む質問点対のリスト P(v) のセルへのポインタ  */

    c  :=  NodeSpace[v].lm_child ;
    while(  c > 0  ) {
        LcaDFS(  c  ) ;                         /*  点 v の子 c に対して LcaDFS を実行する  */
        Merge(  v, c, c  ) ;                    /*  U(v)  :=  U(v) ∪ U(c)  */
        c  :=  NodeSpace[c].r_sibling ;
    }

    ptrPcell  :=  TreeNodes[v].head ;
    while(  ptrPcell ≠ NULL  ) {                /*  点 v を含む質問点対 (v, w) が存在する  */
        p  :=  (↑ptrPcell).pair ;
        if(  QueryPairs[p][0] = v )     w  :=  QueryPairs[p][1] ;
        else                            w  :=  QueryPairs[p][0] ;
        if(  TreeNodes[w].mark ≠ 0  )   Ancestor[p]  :=  Find( w ) ;     /*  最下位共通先祖を求める  */
        ptrPcell  :=  (↑ptrPcell).next ;
    }
    TreeNodes[v].mark  :=  1 ;              /*  点 v を探索済みにする  */
} /*  LcaDFS  */
```

以上

詳細設計書

作成者　　築山　修治　　作成日　2010 年 10 月 5 日

関数名： *Merge*(int　*x, y, z*)

主要データ構造　（関数における主要なデータ構造）

・引数の整数 *x, y,* および *z* は, それぞれ木 T の点に対応し, 値呼びである.

・他のデータ構造は, 関数 *Lca*(*T, m*) と同じである.

処理の詳細　（アルゴリズムの詳細を説明）

/* 互いに素な集合 *U*(*x*) および *U*(*y*) が与えられたとき, 集合 *U*(*z*) を *U*(*z*) := *U*(*x*) ∪ *U*(*y*) とする関数 */
/* 各集合 *U*(・) は MF 木で与えられている */

```
void  Merge(  int  x, y, z  )
{
    MFNODE↑        Large,  Small ;

    /* 点の個数の大きい方の MF 木の根を指すポインタを Large に, 小さい方を Small にする */
    if (  (↑TreeNodes[x].set).size  ≥  (↑TreeNodes[y].set).size  ) {
        Large  :=  TreeNodes[x].set ;
        Small  :=  TreeNodes[y].set ;
    } else {
        Large  :=  TreeNodes[y].set ;
        Small  :=  TreeNodes[x].set ;
    }

    /* Large が指す MF 木の根を, 集合 U(z) に対応する MF 木の根にする */
    (↑Large).size  :=  (↑Large).size + (↑Small).size ;
    (↑Small).parent  :=  Large ;
    (↑Large).set_name  :=  z ;
    TreeNodes[z].set  :=  Large ;
} /*  Merge  */
```

以上

詳細設計書

作成者　　築山　修治　　作成日　2010 年 10 月 5 日

関数名： *Find*(int　*x*)

主要データ構造　（関数における主要なデータ構造）

・引数の整数 *x* は，木 T の点 *x* を示し，値呼びである．

・MF 木の点へのポインタを入れるスタック　*S*

```
MFNODE↑      S_Ele[MF_height]      /*  MF_height =⌈log2(NS_length)⌉ は定数  */
int          S_top ;               /*  MF 木の点へのポインタを入れる配列  */
                                   /*  スタックの先頭を指すカーソル．空のときは 0  */
```

・他のデータ構造は，関数 *Lca*(*T*, *m*) と同じである．

処理の詳細　（アルゴリズムの詳細を説明）

```
/*  互いに素な集合の集合（族：family）{ U(・) } が与えられたとき，x を含む集合 U(a) を見出し，a を返す関数  */
/*  各集合 U(・) は MF 木で与えられており，a は MF 木の set_name 欄に入っている  */

int  Find( int x )
{
    int              S_top ;                    /*  S_top はスタック S の先頭を示す  */
    MFNODE↑          S_Ele[MF_height] ;         /*  スタック S に入る要素を蓄えるための配列  */
    MFNODE↑          ptrMFnode ;                /*  MF 木の点へのポインタ  */

    ptrMFnode := TreeNodes[x].node ;
    S_top := 0 ;
    while( (↑ptrMFnode).parent ≠ NULL ) {       /*  訪問した点をスタック S に入れながら，MF 木の根を探す  */
        S_top := S_top + 1 ;
        S_Ele[S_top] := ptrMFnode ;             /*  スタック S に ptrMFnode を Push する  */
        ptrMFnode := (↑ptrMFnode).parent ;
    }                                           /*  この時点で，ptrMFnode は MF 木の根を指す  */
    while( S_top ≠ 0 ) {                        /*  道の圧縮を行う  */
        (↑S_Ele[S_top]).parent := ptrMFnode ;
        S_top := S_top – 1 ;                    /*  スタック S から先頭の要素を Pop する  */
    }

    return (↑ptrMFnode).set_name ;              /*  MF 木の根に書かれている集合名を返す  */
} /*  Find  */
```

以上