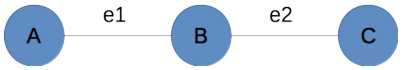
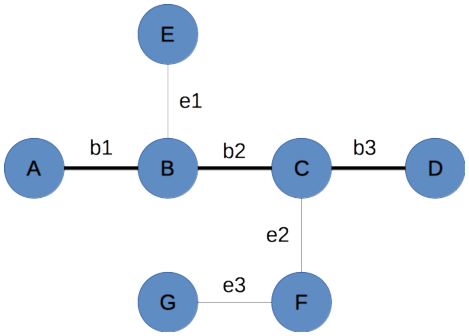
ACM ICPC2014 国内予選　問題 E 橋の撤去

【考察】

- 島の数が n の時に橋の数は n-1 であり全ての島と島の間は到達可能であるので、島を節点、橋を枝にしたグラフは木となる。
- degree が 2 以上の島は必ず訪問する必要がある。例えば、下図において、島 B を訪問すること無く橋 e1 と e2 を撤去するためには、島 A と島 C を訪問する必要がある。これは、島 A と島 C の間にパスが存在することを意味する。このパスと e1, e2 で loop になるので、グラフが木であることに矛盾する。



- degree が 1 の島は、その島を訪問することなく、その島に架かっている橋を撤去することができる。例えば上図において、島 B を訪問した時に橋 e1 と e2 を撤去すれば良い。したがって、degree が 1 の島に架かっている橋のコストの和が、そのような橋の撤去にかかるコストとなる。
- 次に、元のグラフから上記の島と橋（degree が 1 の島と、その島に架かっている橋）を取り除いたグラフを構築する。
- このグラフの枝（橋）を撤去する作業が、島 A から始まって島 D で終了したとする（下図で太線の経路）。このとき、例えば橋 e1 を撤去する為には、まず島 B から島 E へ渡り（degree 1 の島は既に除いているので、島 E を訪問する必要がある）、島 B に戻ってから橋 e1 を撤去することになるので、これらにかかるコストは e1 のコストの 3 倍になる。同様に、島 C を訪問した際に、橋 e2 と e3 を撤去する為には、まず島 F にわたり、次に島 G にわたり、そこから島 F に戻って橋 e3 を撤去し、その後島 C に戻って橋 e2 を撤去することになる。これに必要なコストは、e2 と e3 のコストの和の 3 倍となる。一方、橋 b1, b2, b3 の撤去は、島 A から島 B に渡って橋 b1 を撤去し、次に島 C に渡ってから橋 b2 を撤去し、最後に島 D に渡ってから橋 b3 を撤去すれば良いので、このコストは、b1, b2, b3 のコストの和の 2 倍となる。この両者をあわせると、$3\times(e1+e2+e3)+2\times(b1+b2+b3) = 3\times(e1+e2+e3+b1+b2+b3)-(b1+b2+b3)$ となる。即ち、全ての枝（橋）のコストの和（撤去順によらずコンスタント）の 3 倍から、島 A から島 D への経路上の橋のコストの和を引いたものになる。



したがって、総コストを最小にする為には、各島を始点として最遠島までの距

離（コスト）を求め、それらの最大値を求めれば良い（即ち最遠頂点間距離）[1]。

- ある島を始点とした時に、その島から最も遠い島までの距離は、木の深さ優先探索を行えば求められる。
- 以上をまとめると
  1. degree が 1 の島に架かっている橋の総コスト C1 を求める。
  2. 元の木（グラフ）から、degree が 1 の島とその島に架かっている橋を取り除いた木を構築する。
  3. この木の枝のコストの総和を C3 とする。
  4. この木の直径（最遠頂点間距離）を C2 とする。
  5. 全体の最小コストはＣ１＋３×C3－C2 となる。

【プログラム例１】

```
// ACM-ICPC2014 Domestic Problem E 橋の撤去
//   http://icpc.iisf.or.jp/past-icpc/domestic2014/#section_E
//   Filename =    pe1.c
//   Compile:      cc pe1.c
//   Execution:    ./a.out < E0 > E0.out
//   Check:        diff E0.ans E0.out

#include <stdio.h>
#include <stdlib.h>

#define MAXN 800            // 島の数の最大値
#define INF 2000000000

typedef struct {    // 橋のデータの構造体
  int n1, n2;               // 橋が架かっている島の番号
  int len;                  // 橋の長さ
  int rm;                   // 撤去済み
} edge_t;
edge_t e[MAXN-1];

typedef struct {    // 島のデータの構造体
  int ne;                   // 橋の数（degree）
  //  int re;                 // degree 1 の島の橋撤去後の橋の数
  int eid[MAXN-1];          // この島に架かっている橋の ID
} node_t;
node_t nd[MAXN];

int n;              // 島の数（3 <= n <= 800）
int c1;             // degree が 1 の島に架かっている橋のコストの総和
int c2;             // degree が 1 以外の島に架かっている橋のコストの総和
int maxd;           // 島間の距離の最大値
```

[1] この問題は「木の直径」を求める問題であり、より効率の良いアルゴリズムが知られている（http://www.prefield.com/algorithm/graph/tree_diameter.html）。まず、適当な島から最も遠い位置の島 u を求める。次に、島 u から最も遠い位置の島 v を求めると、u, v 間の距離が木の直径となる。

```
// 島間の距離の最大値を求める
// pn=直前の島、cn=現在の島、d=現在の島までの距離
void find_maxd(int pn, int cn, int d)
{
  int i, edge;
  if(d > maxd) maxd = d;     // 現時点の距離が最大値を超えていれば更新
  for(i = 0; i < nd[cn].ne; i++){          // この島に架かっている各橋に対して
    edge = nd[cn].eid[i];      // 橋の ID
    if(e[edge].rm) continue; // 撤去された橋は無視
    if(e[edge].n1 == pn || e[edge].n2 == pn) continue;       // 直前の島へは帰らない
    // この橋を渡り、島間距離の最大値を再帰的に更新
    if(e[edge].n1 != cn)
      find_maxd(cn, e[edge].n1, d+e[edge].len);
    else
      find_maxd(cn, e[edge].n2, d+e[edge].len);
  }
}



int main()
{
  int i,j,k;
  while(1){
    scanf("%d", &n);                    // 島の数 n の入力
    if(n==0) break;                     // n が 0 なら終了
    for(i=0; i<n; i++)                  // 島のデータの初期化
      nd[i].ne = 0;                     // 島に架かっている橋の数を 0 に初期化
    c1 = c2 = maxd = 0;                 // コストの総和や島間距離の最大値を 0 に初期化
    for(i=0; i<n-1; i++){       // 橋のデータを読み込む
      int i1, i2;               // 橋が架かっている島の ID (0 ～ n-1)
      scanf("%d", &i1);
      i1--;                             // この橋が架かっている島の ID
      e[i].n1 = i1;
      e[i].n2 = i2 = i+1;       // この橋が架かっているもう一つの島の ID
      nd[i1].eid[nd[i1].ne++] = i;      // 2 つの島に橋 ID をセット
      nd[i2].eid[nd[i2].ne++] = i;
    }
    for(i=0; i<n-1; i++){       // 各橋の長さを読み込む
      scanf("%d", &e[i].len);
      e[i].rm = 0;
    }

    // degree が 1 の橋はその橋を渡ること無く撤去できる
    for(i=0; i<n-1; i++){       // 各橋 i に対して
      // この橋が結ぶ島のどちらかの degree が 1 なら
      if(nd[e[i].n1].ne == 1 || nd[e[i].n2].ne == 1){
       e[i].rm = 1;                     // その橋を撤去
       c1 += e[i].len;                  // 撤去費用を c1 に加算
      }
      else
       c2 += e[i].len;                  // そうでなければ撤去費用を c2 に加算
```

```
    }

    //   2 島間の最大距離を求める
    for(i=0; i<n; i++){                    // 各島をスタート地点として
      if(nd[i].ne == 1) continue;          // degree が 1 の島はスキップ
      find_maxd(-1, i, 0);    // スタート地点からの距離を再帰的に求めながら
                                           // その最大値を更新
    }

    //      printf("%d¥t%d %d %d¥n", c1+3*c2-maxd, c1, c2, maxd);
    printf("%d¥n", c1+3*c2-maxd);
  }
}
```

【プログラム例２】（木の直径を求めるのに２頁脚注のアルゴリズムを使用）

```
// ACM-ICPC2014 Domestic Problem E 橋の撤去
//   http://icpc.iisf.or.jp/past-icpc/domestic2014/#section_E
//   Filename =     pe2.c
//   Compile:       cc pe2.c
//   Execution:     ./a.out < E0 > E0.out
//   Check:         diff E0.ans E0.out
//   木の直径：   ある頂点を始点とした時の最遠頂点 u を求め、
//                次に u を始点とする最遠頂点 v を求めれば、u,v が最遠頂点対となる
//                  O(E) = O(N)

#include <stdio.h>
#include <stdlib.h>

#define MAXN 800            // 島の数の最大値
#define INF 2000000000

typedef struct {   // 橋のデータの構造体
  int n1, n2;                // 橋が架かっている島の番号
  int len;                   // 橋の長さ
  int rm;                    // 撤去済み
} edge_t;
edge_t e[MAXN-1];

typedef struct {   // 島のデータの構造体
  int ne;                    // 橋の数（degree）
  //  int re;                 // degree 1 の島の橋撤去後の橋の数
  int eid[MAXN-1];           // この島に架かっている橋の ID
} node_t;
node_t nd[MAXN];

int n;              // 島の数（3 <= n <= 800）
int c1;             // degree が 1 の島に架かっている橋のコストの総和
int c2;             // degree が 1 以外の島に架かっている橋のコストの総和
int maxd;           // 最遠島までの距離
int maxid;          // 最遠島の ID
```

```
// 島間の距離の最大値を求める
// pn=直前の島、cn=現在の島、d=現在の島までの距離
void find_maxd(int pn, int cn, int d)
{
  int i, edge;
  if(d > maxd){                  // 現時点の距離が最大値を超えていれば更新
    maxd = d; maxid = cn;
  }
  for(i = 0; i < nd[cn].ne; i++){          // この島に架かっている各橋に対して
    edge = nd[cn].eid[i];       // 橋の ID
    if(e[edge].rm) continue; // 撤去された橋は無視
    if(e[edge].n1 == pn || e[edge].n2 == pn) continue;       // 直前の島へは帰らない
    // この橋を渡り、島間距離の最大値を再帰的に更新
    if(e[edge].n1 != cn)
      find_maxd(cn, e[edge].n1, d+e[edge].len);
    else
      find_maxd(cn, e[edge].n2, d+e[edge].len);
  }
}

int main()
{
  int i,j,k;
  while(1){
    scanf("%d", &n);                    // 島の数 n の入力
    if(n==0) break;                     // n が 0 なら終了
    for(i=0; i<n; i++)                  // 島のデータの初期化
      nd[i].ne = 0;                     // 島に架かっている橋の数を 0 に初期化
    c1 = c2 = maxd = 0;                 // コストの総和や島間距離の最大値を 0 に初期化
    for(i=0; i<n-1; i++){       // 橋のデータを読み込む
      int i1, i2;               // 橋が架かっている島の ID (0 ～ n-1)
      scanf("%d", &i1);
      i1--;                             // この橋が架かっている島の ID
      e[i].n1 = i1;
      e[i].n2 = i2 = i+1;       // この橋が架かっているもう一つの島の ID
      nd[i1].eid[nd[i1].ne++] = i;      // 2 つの島に橋 ID をセット
      nd[i2].eid[nd[i2].ne++] = i;
    }
    for(i=0; i<n-1; i++){       // 各橋の長さを読み込む
      scanf("%d", &e[i].len);
      e[i].rm = 0;
    }

    // degree が 1 の橋はその橋を渡ること無く撤去できる
    for(i=0; i<n-1; i++){       // 各橋 i に対して
      // この橋が結ぶ島のどちらかの degree が 1 なら
      if(nd[e[i].n1].ne == 1 || nd[e[i].n2].ne == 1){
       e[i].rm = 1;                     // その橋を撤去
       c1 += e[i].len;                  // 撤去費用を c1 に加算
      }
      else
       c2 += e[i].len;                  // そうでなければ撤去費用を c2 に加算
```

```
    }

    //   2 島間の最大距離を求める

    for(i=0; i<n; i++)
      if(nd[i].ne > 1) break; // degree が 1 の島はスキップ
    maxid=i; maxd=0;
    find_maxd(-1, i, 0);        // まず島 i からの最遠島を求める
    //      printf("from=%d to=%d dist=%d¥n", i, maxid, maxd);
    maxd = 0;
    i = maxid;
    find_maxd(-1, i, 0);        // その島からの最遠島までの距離を求める
    //      printf("from=%d to=%d dist=%d¥n", i, maxid, maxd);
    //      printf("%d¥t%d %d %d¥n", c1+3*c2-maxd, c1, c2, maxd);
    printf("%d¥n", c1+3*c2-maxd);
  }
}
```