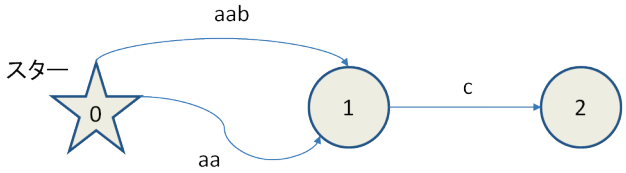
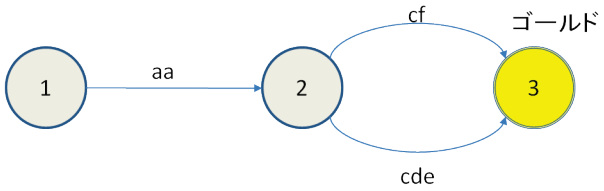
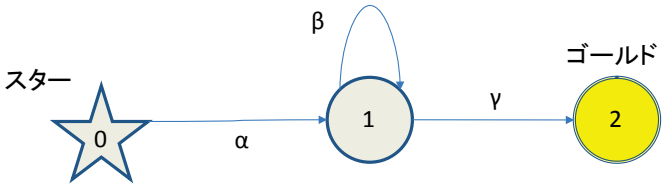
[Problem E]　最強の呪文



例えば、上図のような場合を考えると、節点 0（スター）から節点 1 に至るパスの最強の呪文は「aa」であるが、節点 0 から節点 2 に至るパスの最強の呪文は「aabc」であり、節点 0 と節点 1 の間のパスとして最強の「aa」は用いられていない。したがって、スターから各節点への最強の呪文を求めていく方法は旨く機能しないと考えられる。



一方、上図において、節点 2 から節点 3（ゴールド）に至るパスの最強の呪文は「cde」であり、節点 1 から節点 2 を経由して節点 3 に至るパスの最強呪文は「aacde」であり、節点 2 と節点 3 の間のパスとして最強の「cde」が用いられている。したがって、各節点からゴールドに至るパスの最強呪文をパスを構成する枝数の順に求めていけば、スターからゴールドにいたるパスの最強呪文を求めることができる（存在する場合）。

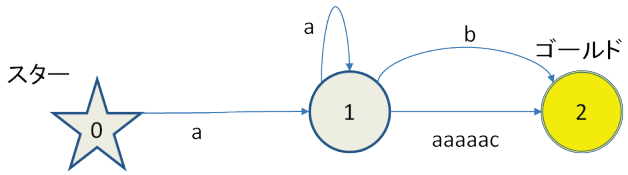


最強の呪文が存在する場合、その呪文にはループは含まれない。もし上図のようなループを含んでいるとすると、「α β γ」の方が「α γ」より強い事を意味し、その場合、「α β β γ」は「α β γ」より強くなる。したがってループを回る回数を増やすことによりいくらでも強い呪文を作ることができ、最強の呪文は存在しなくなる。

これらの考察より、節点数を n とするときに、枝の本数を 1 から n − 1 まで順次増やしながら、指定の枝の本数以下のパスを対象として、各節点からゴールドにいたるパスの中での最強の呪文を求めていく。これにより最強の呪文が存在する場合は、ループ無しの最強呪文が「スター」節点のところに求まっている。プログラム例では、この操作を 1 サイクルと呼んでいる。

最強の呪文が存在するかどうかについては、もう少し検討が必要である。各枝のラベルの

文字数は１～６なので、同じ本数の枝を持つパスで生成される呪文の長さは最大で６倍の差がある。



例えば、上図のようなケースを考える。１サイクル終了した時点では、スター節点のところに最強呪文として「aaaaaac」が求まる。しかしながら、サイクルを繰り返していくと、やがてスター節点のところに「aaaaaac」より強い呪文として「aaaaaab」が現れるので、最強呪文は存在しないと判定できる。枝のラベルの文字数は最大で６倍の差があるので、おそらく合計６サイクルぐらい計算を行えば、最強呪文が存在しない場合には、１サイクル終了時点での最強呪文よりも強いものが現れると思われる（本当？）。

プログラム例

```
// ACM-ICPC 2010 Japan Online Contest Problem E
// http://icpc2010.honiden.nii.ac.jp/domestic-contest/problems#section_E
//      ファイル名: pe.c
//      コンパイル方法: cc pe.c
//      実行方法: ./a.out < E0 > E0.result  など
//      チェック方法: diff E0.ans E0.result  など
//
// アルゴリズム概略
// 各節点に対して、その節点から枝 k 本以内でゴールドに至る最強呪文を
// k = 1, ... , n-1  の順に求める。この操作を１サイクルと名付ける。
// 最強呪文が存在する場合、それはループを含まないので、
// スター節点の所には、その時点で最強呪文が求まっている。
// これが最強呪文であるためには、ループによりこれより強い最強呪文を
// 任意個作りだすことができないことを確認する必要がある。
// スター節点の所に求まっている最強呪文の長さは高々6(n-1)であり、
// より強い最強呪文を作り出すループが存在する場合、そのループに含まれる
// 文字列長は最短の場合１である。従って、合計で６サイクル分(?)計算を行えば、
// １サイクル目に求められたスター節点の最強呪文よりも強い呪文が求まるはずである。
// 既に１サイクル目の計算は終わっているので、あと５サイクル分の計算を行ってみて、
// より強い呪文が出てくれば、最強呪文は存在しない。
// 一方、より強い呪文が出てこなければ、１サイクル目に求まっていたものが
// 最強呪文となる。

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

#define MAX_N    40                 // 節点数の最大値
#define MAX_A    400                // 枝数の最大値
#define MAX_LABEL_LEN 6             // 枝のラベルの最大文字列長

int n;      // 節点数  2 <= n <= 40
int a;      // 枝数     0 <= a <= 400
```

```
int s;      // スター節点     0 <= s < 40
int g;      // ゴールド節点  0 <= g < 40
// s != g

struct edge_type {
   int src;             // 枝の開始節点
   int dst;             // 枝の終了節点
   char label[MAX_LABEL_LEN+1];             // 枝のラベル
};
typedef struct edge_type edge_t;

struct node_type {
   char *css;         // 現時点でこの節点からゴールドに至る最強の呪文
   char *new;         // より強力な新しい呪文
};
typedef struct node_type node_t;

edge_t edge[MAX_A];         // 枝の情報を格納する配列
node_t node[MAX_N];         // 節点の情報を格納する配列

// 1 サイクル = 枝を n-1 回たどる
void cycle(int c)
{
   int i,j,k,flag;
   for(i=0; i < (n-1)*c; i++){              // c サイクル分、枝をたどる
      flag = 0;
      for(j=0; j<a; j++){                   // 各枝に対し以下の操作を行う
         int sn, dn;
         char *el, *nl;
         int len1, len2;
         sn = edge[j].src;                  // 枝の始点
         dn = edge[j].dst;                  // 枝の終点
         el = edge[j].label;                // 枝のラベル
         nl = NULL;
         // 終点の節点の暫定最強呪文が求まっていない場合は何もしない
         if(node[dn].css == NULL) continue;
         len1 = strlen(el);                 // 枝のラベルの文字列の長さ
         len2 = strlen(node[dn].css);    // 終点節点の暫定最強呪文の長さ
         nl = (char *)malloc(len1+len2+1); // 新しい呪文を格納する領域を確保
         strcpy(nl, el);                    // 新しい呪文は、 枝のラベル＋
         strcat(nl,node[dn].css);           //                  終点節点の暫定最強呪文
         if(node[sn].new == NULL){    // 始点節点に対し新しい呪文が未登録で
            if(node[sn].css == NULL){   // 始点節点に対し暫定最強呪文が未登録なら
               node[sn].new = nl;          // 今作成した新しい呪文を  new  に登録し
               flag = 1;                   // 変化があったことを示す  flag  をセット
            }
            else {      // 始点節点に対し、新呪文は未登録、暫定最強呪文は登録済み
               // 新しい呪文が暫定最強呪文よりも強ければ、
               if(strcmp(nl, node[sn].css) < 0){
                  node[sn].new = nl;   // それを新呪文として登録し
                  flag = 1;                // 変化があったことを示す  flag  をセット
               }
            }
         }
         else {         // 始点節点に対し、新呪文が登録済
```

```
            // 新しい呪文が既登録の新呪文よりも強ければ
            if(strcmp(nl, node[sn].new) < 0){
              free(node[sn].new);       // 古い新呪文の領域を解放し
              node[sn].new = nl;        // 新たに見つかった呪文を新呪文として登録
              flag = 1;                 // 変化があったことを示す flag をセット
            }
          }
        }
        if(flag){                       // より強力な新呪文が見つかっていれば
          for(j=0; j<n; j++){
            if(node[j].new){            // 新呪文が見つかった各節点に対して
              if(node[j].css) free(node[j].css);  //暫定最強呪文が登録されていれば
              // その領域を解放する
              node[j].css = node[j].new;  // 登録されていた新呪文を暫定最強呪文とし
              node[j].new = NULL;       // 新呪文は未登録とする
            }
          }
        }
        else
          break;  //  より強力な新呪文は見つからなかったので終了
      }
    }

    void solve()
    {
      char *save;
      cycle(1);             // サイクルを 1 回まわすと、ループ無しの最強呪文があれば見つかる
      if(node[s].css == NULL){
        printf("NO¥n");           // 見つかっていなければ最強呪文は存在しない
        return;
      }
      // 現時点での暫定最強呪文を記憶する領域を確保し保存しておく
      save = (char *)malloc(strlen(node[s].css)+1);
      strcpy(save, node[s].css);

      cycle(5);    // あと 5 サイクルまわす
      if(strcmp(node[s].css, save) < 0) // ループでより強力な呪文を構成できる場合は
        printf("NO¥n");    // 最強呪文は存在しない
      else   // より強力な呪文を構成できなかった場合は
        printf("%s¥n",save);  // 先ほど保存しておいたものが最強呪文である
      free(save); // 最強呪文を保存していた領域を解放
    }

    int main()
    {
      int i,x,y;
      char labelstr[MAX_LABEL_LEN+1];       // 枝のラベル文字列を読み込むための配列
      while(1){
        scanf("%d %d %d %d", &n, &a, &s, &g);           // n, a, s, g を入力
        if(n==0 && a==0 && s==0 && g==0) break;        // 全て 0 なら終了
        for(i=0; i<n; i++){
          // 各節点に登録される呪文データを未登録状態に初期化
          node[i].css = node[i].new = NULL;
        }
        for(i=0; i<a; i++){
```

```
        // 各枝の開始点、終了点、ラベルを入力
        scanf("%d %d %s", &x, &y, labelstr);
        edge[i].src = x;
        edge[i].dst = y;
        strcpy(edge[i].label, labelstr);
      }
      // ゴールド節点の最強呪文として空文字列を登録
      node[g].css =   (char *)malloc(1);
      *node[g].css =   '¥0';
      solve();          // 答えを求める
      for(i=0; i<n; i++){
        // 各節点に登録した呪文データの格納領域を解放
        if(node[i].css) free(node[i].css);
        if(node[i].new) free(node[i].new);
      }
    }
}
```