```
(global-font-lock-mode t)
(column-number-mode 1)
(show-paren-mode 1)

(defun huk () (interactive)
  (defun vit (&optional arg) (interactive)
    (insert "{")
    (c-indent-command)
    (newline)
    (insert "}")
    (c-indent-command)
    (previous-line 1)
    (end-of-line)
    (newline-and-indent)
    )

  (defun kompiliraj (&optional arg) (interactive)
    (save-buffer arg)
;;  (compile (concat "g++ -o \"" exe-name "\" \"" buffer-file-name "\" -O2 -s -Wall -lm")
)
;;  (compile (concat "compile " buffer-file-name))
    )

  (setq c-tab-always-indent t)
  (setq indent-tabs-mode nil)
  (setq c-basic-offset 3)
  (c-toggle-hungry-state 1)
  (define-key c-mode-base-map "\M-2" 'kompiliraj)
  (define-key c-mode-base-map "\M-p" 'previous-error)
  (define-key c-mode-base-map "\M-n" 'next-error)
  (define-key c-mode-base-map "\M-j" 'vit)
  (define-key c-mode-base-map (kbd "<C-tab>") 'c-indent-command)
  )

(add-hook 'c-mode-common-hook 'huk)

(global-set-key (kbd "C-c c") 'comment-region)
(global-set-key (kbd "C-c u")
                (function (lambda () (interactive)
                            (uncomment-region (mark) (point))
                            (indent-region (mark) (point) nil))))
```

```
/*********************************************
 *                arhitekt.cpp               *
 *********************************************/

#include <algorithm>
#include <cassert>
#include <cstdio>
#include <vector>
using namespace std;

#define MAX    200001
#define OFFSET (MAX/2+1)
#define BLOK   3500

typedef pair<int, int> Point;

struct Loga {
   int* loga;
   Loga() {
      loga = new int[MAX+1];
      fill( loga, loga+MAX+1, 0 );
   }

   void add(int x) {
      while (x <= MAX) {
         loga[x] += 1;
         x += x&-x;
      }
   }

   int query(int x) {
      int ret = 0;
      while (x > 0) {
         ret += loga[x];
         x -= x&-x;
      }
      return ret;
   }
};

Loga loge[MAX/BLOK+1];
vector<Point> tocke;
int xstart[MAX+1], xend[MAX+1];

int vertical_query( int x, int y2 ) {
   if (y2 < 1) return 0;

   vector<Point>::iterator
      it1 = tocke.begin() + xstart[x],
      it2 = upper_bound( it1, tocke.begin() + xend[x], Point(x, y2) );
   return it2-it1;
}

int full_query( int x1, int x2, int y ) {
   if (x1 > x2) return 0;

   int pocetni_blok = x1 / BLOK, zavrsni_blok = x2 / BLOK;

   int ret = 0;
   if (pocetni_blok == zavrsni_blok) {
      for ( int x=x1; x<=x2; ++x )
         ret += vertical_query( x, y );
   } else {
      for ( int x=x1; x<(pocetni_blok+1)*BLOK; ++x )
         ret += vertical_query( x, y );

      for ( int blok=pocetni_blok+1; blok<zavrsni_blok; ++blok)
         ret += loge[blok].query(y);

      for ( int x=zavrsni_blok*BLOK; x<=x2; ++x )
         ret += vertical_query( x, y );
   }
   return ret;
```

```
}

enum { LIJEVO, DESNO, GORE, DOLJE };

int main() {
   int npts;
   scanf( "%d", &npts );
   for (int i=0; i<npts; ++i) {
      int x, y;
      scanf( "%d%d", &x, &y );
      x += OFFSET; y += OFFSET;
      tocke.push_back( Point(x, y) );
      loge[x/BLOK].add(y);
   }
   sort( tocke.begin(), tocke.end() );

   xstart[0] = 0; xend[0] = 0;
   int i = 0;
   for (int x=1; x<=MAX; ++x) {
      xstart[x] = i;
      while (i<(int)tocke.size() && tocke[i].first == x) ++i;
      xend[x] = i;
   }

   int Q;
   scanf( "%d", &Q );
   for (int q=0; q<Q; ++q) {
      int n;
      static int qx[30], qy[30], tip[30];
      scanf( "%d", &n );
      for (int i=0; i<n; ++i) {
         scanf( "%d%d", &qx[i], &qy[i] );
         qx[i] += OFFSET; qy[i] += OFFSET;
      }
      qx[n] = qx[0]; qy[n] = qy[0];

      int ret = 0;
      for (int i=0; i<n; ++i) {
         if (qx[i] == qx[i+1] ) {
            tip[i] = qy[i+1] > qy[i] ? GORE : DOLJE;
         } else if (qx[i+1] > qx[i]) {
            tip[i] = DESNO;
            ret -= full_query( qx[i]+1, qx[i+1]-1, qy[i]-1 );
         } else {
            tip[i] = LIJEVO;
            ret += full_query( qx[i+1]+1, qx[i]-1, qy[i] );
         }
      }
      tip[n] = tip[0];

      for (int i=0; i<n; ++i) {
         int a = tip[i], b = tip[i+1];
         if (a == GORE && b == LIJEVO) {
            ret += vertical_query( qx[i+1], qy[i+1] );
         } else if (a == GORE && b == DESNO) {

         } else if (a == DOLJE && b == LIJEVO) {

         } else if (a == DOLJE && b == DESNO) {
            ret -= vertical_query( qx[i+1], qy[i+1]-1 );
         } else if (a == LIJEVO && b == GORE) {

         } else if (a == LIJEVO && b == DOLJE) {
            ret += vertical_query( qx[i+1], qy[i+1] );
         } else if (a == DESNO && b == GORE) {
            ret -= vertical_query( qx[i+1], qy[i+1]-1 );
         } else if (a == DESNO && b == DOLJE) {

         }
      }

      printf( "%d\n", ret );
   }
```

```
   return 0;
}

/*********************************************
 *                 bigint.cc                 *
 *********************************************/

#include <iostream>
#include <string>
#include <vector>
using namespace std;

class bigint {
    static const int BASE = 10000;
    int           sign;
    vector< int > limbs;

    bigint &trim()
    { while ( limbs.size() > 0 && limbs.back() == 0 ) limbs.pop_back(); if ( limbs.size()
 == 0 ) sign = 1; return *this; }

    int operator[]( const size_t index ) const
    { return index >= limbs.size() ? 0 : limbs[index]; }

    // ternarni predikat za usporedivanje
    static int cmp( const bigint &a, const bigint &b )
    {
        if ( a.sign != b.sign ) return a.sign;
        if ( a.limbs.size() < b.limbs.size() ) return -a.sign;
        if ( a.limbs.size() > b.limbs.size() ) return a.sign;

        for ( int i=(int)a.limbs.size()-1; i>=0; --i ) {
            if ( a.limbs[i] < b.limbs[i] ) return -a.sign;
            if ( a.limbs[i] > b.limbs[i] ) return a.sign;
        }

        return 0;
    }

    // oduzimanje kad je ( a.sign == b.sign == 1 ) && a >= b
    static bigint absolute_subtract( const bigint &a, const bigint &b )
    {
        int borrow = 0;
        bigint ret;

        for ( int i=0; i<(int)a.limbs.size(); ++i ) {
            ret.limbs.push_back( a.limbs[i] - b[i] - borrow );
            if ( ret.limbs[i] < 0 ) {
                ret.limbs[i] += BASE;
                borrow = 1;
            }
            else borrow = 0;
        }
        return ret.trim();
    }

    // zbrajanje kad je ( a.sign == b.sign == 1 )
    static bigint absolute_add( const bigint &a, const bigint &b )
    {
        int maxi = max( a.limbs.size(), b.limbs.size() );
        int carry = 0;
        bigint ret;

        for ( int i=0; i<maxi; ++i ) {
            ret.limbs.push_back( a[i] + b[i] + carry );
            if ( ret.limbs[i] >= BASE ) {
                ret.limbs[i] -= BASE;
                carry = 1;
            }
            else carry = 0;
        }
        ret.limbs.push_back( carry );
        return ret.trim();
    }
```

```
    // pomocne operacije za ++ i --
    bigint &absolute_inc( void )
    {
        int i;
        for ( i=0; i<(int)limbs.size() && limbs[i] == BASE-1; ++i )
            limbs[i] = 0;
        if ( i == (int)limbs.size() ) limbs.push_back( 0 );
        ++limbs[i];
        return *this;
    }
    bigint &absolute_dec( void )
    {
        int i;
        for ( i=0; i<(int)limbs.size() && limbs[i] == 0; ++i )
            limbs[i] = BASE-1;
        if ( i == (int)limbs.size() ) limbs.push_back( 1 );
        --limbs[i];
        return trim();
    }

    bigint &multiply_and_add( const bigint &b, int s, int offset )
    {
        int carry = 0;
        int i;

        for ( i=0; i<(int)b.limbs.size(); ++i ) {
            while ( (int)limbs.size() <= offset+i ) limbs.push_back( 0 );
            limbs[ offset+i ] += b.limbs[i] * s + carry;
            carry = limbs[ offset+i ] / BASE;
            limbs[ offset+i ] %= BASE;
        }
        for ( ; carry > 0; ++i ) {
            while ( (int)limbs.size() <= offset+i ) limbs.push_back( 0 );
            limbs[ offset+i ] += carry;
            carry = limbs[ offset+i ] / BASE;
            limbs[ offset+i ] %= BASE;
        }
        return trim();
    }

    static void binary_search_div( const bigint &a, const bigint &b, bigint &quo, bigint
&rem )
    {
        rem.limbs.clear();
        quo.limbs.resize( a.limbs.size() );

        for ( int i=(int)a.limbs.size()-1; i>=0; --i ) {
            rem.limbs.insert( rem.limbs.begin(), a[i] ); // rem = rem * BASE + a[i]

            int lo = 0, hi = BASE-1;
            while ( lo < hi ) {
                int mid = ( lo + hi ) / 2 + 1;
                bigint key = b * mid;
                if ( key.sign == -1 ) key.negate();
                int r = cmp( rem, key );
                if      ( r >= 0 ) { lo = mid; }
                else if ( r <  0 ) { hi = mid-1; }
            }

            quo.limbs[i] = lo;
            rem = rem - b * lo;
            if ( rem.limbs.size() > 0 && rem.sign == -1 ) rem = rem + b;
        }

        quo.sign = a.sign * b.sign;
        quo.trim();
    }

public:
    bigint() { sign = 1; }                          // default
    bigint( const bigint &drugi ) { *this = drugi; }   // copy

    // string konstruktor
    bigint( const string &str )
    {
        int rank = 1;
        int limb = 0;

        sign = 1;
        for ( int i=(int)str.size()-1; i>=0; --i ) {
            if ( str[i] == '-' ) { sign=-1; break; }
            limb += ( str[i] - '0' ) * rank;
            if ( ( rank *= 10 ) == BASE ) {
                limbs.push_back( limb );
                limb = 0; rank = 1;
            }
        }
        if ( limb != 0 ) limbs.push_back( limb );
        trim();
    }

    // int konstruktor
    bigint( int x )
    {
        sign = x < 0 ? -1 : 1;
        while ( x != 0 ) {
            limbs.push_back( x % BASE * sign );
            x /= BASE;
        }
    }

    // mijenja predznak
    bigint &negate()
    {
        sign = -sign;
        return *this;
    }

    friend inline bool operator<( const bigint &a, const bigint &b )
    { return cmp( a, b ) < 0; }
    friend inline bool operator>( const bigint &a, const bigint &b )
    { return cmp( a, b ) > 0; }
    friend inline bool operator<=( const bigint &a, const bigint &b )
    { return cmp( a, b ) <= 0; }
    friend inline bool operator>=( const bigint &a, const bigint &b )
    { return cmp( a, b ) >= 0; }
    friend inline bool operator==( const bigint &a, const bigint &b )
    { return cmp( a, b ) == 0; }
    friend inline bool operator!=( const bigint &a, const bigint &b )
    { return cmp( a, b ) != 0; }

    // zbrajanje bigintova
    friend bigint operator+( const bigint &a, const bigint &b )
    {
        if ( a.sign == b.sign ) {
            bigint ret = absolute_add( a, b );
            ret.sign = a.sign;
            return ret;
        }
        else {
            if ( a.sign == -1 ) {
                bigint nega = -a;
                if ( nega > b ) return absolute_subtract( nega, b ).negate();
                else            return absolute_subtract( b, nega );
            }
            else {
                bigint negb = -b;
                if ( negb > a ) return absolute_subtract( negb, a ).negate();
                else            return absolute_subtract( a, negb );
            }
        }
    }

    // unarni minus i oduzimanje
    friend bigint operator-( const bigint &b )
    {
        bigint ret( b );
```

```
        ret.negate();
        return ret;
    }
    friend bigint operator-( const bigint &a, const bigint &b )
    { return a + (-b); }

    // mnozenje
    friend bigint operator*( const bigint &a, const bigint &b )
    {
        bigint ret;
        for ( int i=0; i<(int)b.limbs.size(); ++i )
            ret.multiply_and_add( a, b[i], i );
        ret.sign = a.sign * b.sign;
        return ret.trim();
    }

    // dijeljenje sa intom
    friend bigint operator/( const bigint &a, int s )
    {
        if ( s >= BASE ) return a / bigint( s );

        int rem = 0;
        bigint ret;
        ret.limbs.resize( a.limbs.size() );
        ret.sign = a.sign;
        if ( s < 0 ) { ret.sign = -ret.sign; s = -s; }
        for ( int i=(int)a.limbs.size()-1; i>=0; --i ) {
            rem = rem * BASE + a.limbs[i];
            ret.limbs[i] = rem / s;
            rem %= s;
        }
        return ret.trim();
    }

    friend int operator%( const bigint &a, int s )
    {
        if ( s >= BASE ) ( a % bigint( s ) ).toInt();

        int rem = 0;
        int sign = a.sign;
        if ( s < 0 ) { s = -s; }
        for ( int i=(int)a.limbs.size()-1; i>=0; --i ) {
            rem = ( rem * BASE + a.limbs[i] ) % s;
        }
        return rem * sign;
    }

    // dijeljenje bigintova
    friend bigint operator/( const bigint &a, const bigint &b )
    {
        bigint quo, rem;
        binary_search_div( a, b, quo, rem );
        return quo;
    }

    friend bigint operator%( const bigint &a, const bigint &b )
    {
        bigint quo, rem;
        binary_search_div( a, b, quo, rem );
        return rem;
    }

    // inkrement i dekrement
    friend bigint &operator++( bigint &b )
    {
        if ( b.limbs.size() == 0 ) { b.sign = 1; return b.absolute_inc(); }
        if ( b.sign == 1 ) return b.absolute_inc();
        else               return b.absolute_dec();
    }
    friend bigint operator++( bigint &b, int )
    { bigint ret( b ); ++b; return ret; }
    friend bigint &operator--( bigint &b )
    {
        if ( b.limbs.size() == 0 ) { b.sign = -1; return b.absolute_inc(); }
        if ( b.sign == -1 ) return b.absolute_inc();
        else                return b.absolute_dec();
    }
    friend bigint operator--( bigint &b, int )
    { bigint ret( b ); --b; return ret; }

    friend ostream& operator<<( ostream &os, const bigint &a )
    {
        if ( a.limbs.size() == 0 ) os << '0';
        else {
            if ( a.sign == -1 ) os << '-';
            for ( int i=(int)a.limbs.size()-1; i>=0; --i ) {
                if ( i < (int)a.limbs.size()-1 )
                    for ( int pad=bigint::BASE/10; pad>a.limbs[i] && pad>1; pad/=10 )
                        os << '0';
                os << a.limbs[i];
            }
        }
        return os;
    }

    int toInt() const
    {
        int ret = 0;
        for ( int i=(int)limbs.size()-1; i>=0; --i )
            ret = ret * BASE + limbs[i] * sign;
        return ret;
    }

    friend bigint abs( const bigint &a ) {
        bigint ret( a );
        if ( ret.sign == -1 ) ret.sign = 1;
        return ret;
    }
};

/*********************************************
 *                 bojanje.cpp               *
 *********************************************/

// bojanje edgeova u bipartitnom

#include <algorithm>
#include <cstdio>

using namespace std;

int n;
int c[100][100];
char g[100][101];

int maxdeg;
int deg[2][100];
int bio[2][100];
int how[2][100];
int spojen[2][100];
int best;

int a[100];

void dfs( int side, int i, int dad ) {
  if( bio[side][i] ) return;
  bio[side][i] = 1;
  how[side][i] = dad;
  if( side == 0 )
     for( int j = 0; j < n; ++j ) {
        if( g[i][j] != '*' ) continue;
        if( spojen[0][i] == j ) continue;
        dfs( 1, j, i );
     }
  else
     if( spojen[1][i] == -1 ) {
        if( best == -1 ) best = i;
        if( deg[1][i] > deg[1][best] ) best = i;
```

```
    } else dfs( 0, spojen[1][i], i );
}

bool cmp( const int &A, const int &B ) { return deg[0][A] > deg[0][B]; }
void match( int color ) {
  for( int i = 0; i < n; ++i ) a[i] = i;
  sort( a, a+n, cmp );

  for( int i = 0; i < n; ++i ) spojen[0][i] = spojen[1][i] = -1;

  for( int i = 0; i < n; ++i ) {
     best = -1;
     for( int j = 0; j < n; ++j ) bio[0][j] = bio[1][j] = 0;

     dfs( 0, a[i], -1 );
     if( best == -1 ) continue;
     deg[0][a[i]]--;
     deg[1][best]--;
     int y = best;
     for( ;; ) {
        int x = how[1][y];
        spojen[0][x] = y;
        spojen[1][y] = x;
        if( x == a[i] ) break;
        y = how[0][x];
     }
  }
  for( int i = 0; i < n; ++i ) {
     if( spojen[0][i] == -1 ) continue;
     c[i][spojen[0][i]] = color;
     g[i][spojen[0][i]] = '.';
  }
}

int main( void ) {
  int T;
  scanf( "%d", &T );
  for( int tt = 0; tt < T; ++tt ) {
     scanf( "%d", &n );
     for( int i = 0; i < n; ++i ) scanf( "%s", g[i] );

     for( int i = 0; i < n; ++i ) deg[0][i] = deg[1][i] = 0;

     maxdeg = 0;
     for( int i = 0; i < n; ++i )
        for( int j = 0; j < n; ++j ) {
           maxdeg >?= deg[0][i] += g[i][j] == '*';
           maxdeg >?= deg[1][j] += g[i][j] == '*';
           c[i][j] = 0;
        }

     for( int i = 1; i <= maxdeg; ++i ) match( i );

     printf( "%d\n", maxdeg );
     for( int i = 0; i < n; ++i ) {
        for( int j = 0; j < n; ++j ) {
           if( j ) printf( " " );
           printf( "%d", c[i][j] );
        }
        printf( "\n" );
     }
  }
  return 0;
}


/*********************************************
 *               dfs_digraph.cpp             *
 *********************************************/

// DFS na directed grafu:
//    Detekcija je li DAG
//    Detekcija je li forest
//    Pronalazak strongly connected components-a
```

```
//    Topolosko sortiranje

#include <cstdio>
#include <stack>
#include <vector>

using namespace std;

#define MAX 100

vector<int> adj[MAX];

int Time;
int Discover[MAX];
int Finish[MAX];
enum { White, Gray, Black } Color[MAX];

int LowLink[MAX];   // SCC
int Components;     // SCC
int Component[MAX]; // SCC
stack<int> S;       // SCC
int InStack[MAX];   // SCC

stack<int> T;       // Topological Sort

void dfs_visit( int x ) {
   Color[x] = Gray;
   Discover[x] = ++Time;

   LowLink[x] = Discover[x]; // SCC
   S.push( x );              // SCC
   InStack[x] = 1;           // SCC

   for( vector<int>::iterator it = adj[x].begin(); it != adj[x].end(); ++it ) {
      switch( Color[*it] ) {
         case White:
            // tree edge
            dfs_visit( *it );
            LowLink[x] <?= LowLink[*it]; // SCC
            break;
         case Gray:
            // back edge
            // throw "Cycle detected (Graph is not DAG)";
            break;
         case Black:
            // forward or cross edge
            // throw "Graph is not a tree/forest";
            break;
      }
      if( InStack[*it] )               // SCC
         LowLink[x] <?= Discover[*it]; // SCC
   }

   if( LowLink[x] == Discover[x] )                                                    // S
CC
      for( ++Components; !S.empty() && Discover[S.top()] >= Discover[x]; S.pop() ) { // S
CC
         Component[S.top()] = Components;                                            // S
CC
         InStack[S.top()] = 0;                                                       // S
CC
      }

   Color[x] = Black;
   Finish[x] = ++Time;

   T.push( x ); // Topological Sort
}

void dfs( int n ) {
   Components = 0; // SCC
   Time = 0;
   for( int i = 0; i < n; ++i ) Color[i] = White;
   for( int i = 0; i < n; ++i ) InStack[i] = 0; // SCC
```

```
   for( int i = 0; i < n; ++i )
      if( Color[i] == White )
         dfs_visit( i );
}

int main( void ) {
   int n, m;
   scanf( "%d%d", &n, &m );
   for( int i = 0; i < m; ++i ) {
      int a, b;
      scanf( "%d%d", &a, &b );
      adj[a].push_back( b );
   }
   dfs( n );
   for( int i = 0; i < n; ++i ) printf( "%d ", Component[i] );
   printf( "\n" );
   return 0;
}

/*********************************************
 *                 diofant.cpp               *
 *********************************************/

#include <cmath>
#include <cstdio>
#include <utility>

using namespace std;

int euclid( int a, int b ) {
   if( b == 0 ) return a;
   return euclid( b, a%b );
}

pair<int,int> ext_euclid( int a, int b ) {
   if( b == 0 ) return make_pair( a, 0 );
   pair<int,int> r = ext_euclid( b, a%b );
   return make_pair( r.second, r.first - r.second*(a/b) );
}

pair<int,int> diofant( int a, int b, int c ) {
   int g = euclid( a, b );
   if( g == 0 || c%g != 0 ) throw 1;
   a /= g; b /= g; c /= g;
   pair<int,int> r = ext_euclid( a, b );
   return make_pair( c*r.first, c*r.second );
}

int main( void ) {
   pair<int,int> ret;
   int a, b, c;
   scanf( "%d%d%d", &a, &b, &c );
   try {
      ret = diofant( a, b, c );
      printf( "%d*%d + %d*%d = %d\n", ret.first, a, ret.second, b, c );
   } catch( int tmp ) {
      printf( "Nema rjesenja\n" );
   }
   return 0;
}

/*********************************************
 *                  dvaput.c                 *
 *********************************************/

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

#define MAXLEN  200000
#define MAXHASH 200003

typedef struct {
   int hash, key, next;
} node;

int nnodes = 0;
node nodes[MAXLEN];
int tablica[MAXHASH];

int trazi2( const char *str, int len )
{
   unsigned hash, sub;
   int i;

   if ( len == 0 ) return 1;

   while ( nnodes > 0 ) {
      tablica[ nodes[--nnodes].hash ] = -1;
   }

   hash = 0; sub = 1;
   for ( i=0; i<len; ++i ) {
      sub = (26*sub) % MAXHASH;
      hash = (26*hash + str[i]) % MAXHASH;
   }

   sub = MAXHASH - sub;
   for ( i=len-1; str[i]!='\0'; ++i ) {
      int p;
      for ( p=tablica[hash]; p!=-1; p=nodes[p].next ) {
         int isti = 1, j;
         for ( j=0; j<len; ++j )
            if ( str[ nodes[p].key+j ] != str[ i-len+1+j ] ) {
               isti = 0;
               break;
            }
         if ( isti )
            return 1;
      }

      nodes[nnodes].hash = hash;
      nodes[nnodes].key  = i-len+1;
      nodes[nnodes].next = tablica[hash];
      tablica[hash] = nnodes++;

      hash = (26*hash + sub*str[i-len+1] + str[i+1]) % MAXHASH;
   }

   return 0;
}

int main()
{
   int L, i, lo, hi;
   static char str[MAXLEN+1];
   scanf( "%d%s", &L, str );

   for ( i=0; i<MAXHASH; ++i )
      tablica[i] = -1;

   lo = 0; hi = L-1;
   while ( lo<hi ) {
      int mid = (lo+hi+1) / 2;

      if ( trazi2(str, mid) ) lo = mid;
      else hi = mid-1;
   }

   printf( "%d\n", lo );
   return 0;
}

/*********************************************
 *              flow_na_sparse.cpp           *
 *********************************************/

// flow na sparse grafu
```

```
#include <algorithm>
#include <cstdio>
#include <set>
#include <vector>

using namespace std;

typedef long long llint;

const llint inf = 1000000000000000000LL;

struct edge {
   int u, v;
   llint cost, capacity;
   vector<edge>::iterator reverse;

   edge( int U, int V, llint Cost, llint Capacity = 0 ) {
      u = U;
      v = V;
      cost = Cost;
      capacity = Capacity;
   }
};
bool operator < ( const edge &A, const edge &B ) {
   if( A.u != B.u ) return A.u < B.u;
   if( A.v != B.v ) return A.v < B.v;
   return A.cost < B.cost;
}
bool operator == ( const edge &A, const edge &B ) {
   return A.u == B.u && A.v == B.v && A.cost == B.cost;
}

struct sparse_graph {
   int n;
   vector<edge> E;
   vector<vector<edge>::iterator> V;

   void init( int N ) {
      n = N;
      E.clear();
      V.clear();
   }

   void add_edge( int u, int v, llint cost, llint capacity ) {
      E.push_back( edge( u, v, cost, capacity ) );
      E.push_back( edge( v, u, -cost, 0 ) );
   }

   void done_input() {
      sort( E.begin(), E.end() );

      for( vector<edge>::iterator it = E.begin(); it != E.end(); ++it )
         it->reverse = lower_bound( E.begin(), E.end(), edge( it->v, it->u, -it->cost ) )
;

      V.resize( n+1 );
      V[0] = E.begin();
      for( int u = 1; u <= n; ++u )
         for( V[u] = V[u-1]; V[u] != E.end() && V[u]->u < u; ++V[u] );
   }

   inline vector<edge>::iterator & begin( int u ) { return V[u]; }
   inline vector<edge>::iterator & end( int u ) { return V[u+1]; }
} G;

llint pi[101];
llint dist[101];
llint ff[101];
vector<edge>::iterator how[101];

struct cmp {
   bool operator () ( int a, int b ) {
      if( dist[a] != dist[b] ) return dist[a] < dist[b];
      return a < b;
   }
};

llint dijkstra( int source, int target ) {
   for( int i = 0; i < G.n; ++i ) dist[i] = inf;
   for( int i = 0; i < G.n; ++i ) ff[i] = 0;

   set<int, cmp> PQ;
   dist[source] = 0;
   ff[source] = inf;
   PQ.insert( source );

   while( !PQ.empty() ) {
      int u = *PQ.begin();
      if( u == target ) break;
      PQ.erase( PQ.begin() );

      for( vector<edge>::iterator it = G.begin(u); it != G.end(u); ++it ) {
         llint f = ff[u] <? it->capacity;
         if( f == 0 ) continue;

         llint weight = pi[it->u] + it->cost - pi[it->v];
         if( dist[it->v] <= dist[it->u] + weight ) continue;

         if( dist[it->v] != inf ) PQ.erase( it->v );
         dist[it->v] = dist[it->u] + weight;
         ff[it->v] = f;
         how[it->v] = it;
         PQ.insert( it->v );
      }
   }

   for( int i = 0; i < G.n; ++i ) pi[i] = (pi[i]+dist[i]) <? inf;

   return ff[target];
}


int main( void ) {
   int n, m;
   while( scanf( "%d%d", &n, &m ) == 2 ) {
      int source = 0;
      int sink = n;

      G.init( n+1 );

      for( int i = 0; i < m; ++i ) {
         int u, v, cost;
         scanf( "%d%d%d", &u, &v, &cost ); --u; --v;
         G.add_edge( u, v, cost, 1 );
         G.add_edge( v, u, cost, 1 );
      }
      int D, K;
      scanf( "%d%d", &D, &K );
      for( vector<edge>::iterator it = G.E.begin(); it != G.E.end(); ++it )
         if( it->capacity == 1 ) it->capacity = K;

      G.add_edge( n-1, sink, 0, D );

      G.done_input();

      llint ret = 0, flow = 0;

      for( int i = 0; i < G.n; ++i ) pi[i] = 0;

      for( llint f; (f = dijkstra( source, sink )); flow += f ) {
         for( int x = sink; x != source; x = how[x]->u ) {
            how[x]->capacity -= f;
            how[x]->reverse->capacity += f;
            ret += f * how[x]->cost;
         }
      }
```

```
      if( flow == D ) printf( "%lld\n", ret );
      else printf( "Impossible.\n" );
   }

   return 0;
}

/*********************************************
 *                  frac.cc                  *
 *********************************************/

template<typename frac_t>
class frac {
   // helper functions
   static string itostr( const long long &x ) { static char buf[30]; sprintf( buf, "%lld"
, x ); return string( buf ); }
   static frac_t gcd( frac_t a, frac_t b ) { frac_t t; a = abs(a); b = abs(b); while (b>0
) { t=a%b; a=b; b=t; } return a; }

   frac &reduce() {
      frac_t g = frac::gcd( num, denom );
      num = num / g; denom = denom / g;
      if ( denom < 0 ) { num = -num; denom = -denom; }
      return *this;
   }

   inline static frac __frac( const frac_t &a, const frac_t &b ) { frac ret; ret.num = a;
 ret.denom = b; return ret; }

public:
   frac_t num, denom;
   frac() : num(), denom() { }
   frac( const frac_t &a ) : num(a), denom(1) { reduce(); }
   frac( const frac_t &a, const frac_t &b ) : num(a), denom(b) { reduce(); }
   frac( const frac &drugi ) : num(drugi.num), denom(drugi.denom) { reduce(); }

   friend inline frac operator+( const frac &a, const frac &b ) {
      frac_t g = frac::gcd( a.denom, b.denom );
      return __frac( b.denom / g * a.num + a.denom / g * b.num,
                     a.denom / g * b.denom ).reduce();
   }
   friend inline frac operator-( const frac &a ) { return frac( -a.num, a.denom ); }
   friend inline frac operator-( const frac &a, const frac &b ) { return a + (-b); }
   friend inline frac operator*( const frac &a, const frac &b ) {
      frac_t g1 = frac::gcd( a.num, b.denom ), g2 = frac::gcd( a.denom, b.num );
      return __frac( a.num / g1 * b.num / g2,
                     a.denom / g2 * b.denom / g1 );
   }
   friend inline frac operator/( const frac &a, const frac &b ) { return a * b.reciprocal
(); }

   friend inline bool operator==( const frac &a, const frac &b ) { return a.num == b.num
&& a.denom == b.denom; }
   friend inline bool operator!=( const frac &a, const frac &b ) { return a.num != b.num
|| a.denom != b.denom; }
   friend inline bool operator<( const frac &a, const frac &b ) { return ( a - b ).num <
0; }
   friend inline bool operator>( const frac &a, const frac &b ) { return ( a - b ).num >
0; }
   friend inline bool operator<=( const frac &a, const frac &b ) { return ( a - b ).num <
= 0; }
   friend inline bool operator>=( const frac &a, const frac &b ) { return ( a - b ).num >
= 0; }

   frac reciprocal() const { return __frac( denom, num ); }
   string toString( bool cut = false, const string &delim = "/" ) const {
      if ( denom == 1 && cut ) return itostr( num );
      else return itostr( num ) + delim + itostr( denom );
   }
   double toDouble() const { return double(num) / denom; }
};

/*********************************************
 *               geometrija.cpp              *
 *********************************************/

#include <algorithm>
#include <cmath>
#include <iostream>
#include <vector>

using namespace std;

#define EPS 1e-9
#define eq(a,b) ( ((a)-EPS <= (b)) && ((b) <= (a)+EPS) )
#define lt(a,b) ( (a)+EPS < (b) )
#define gt(a,b) ( (a)-EPS > (b) )
#define lte(a,b) ( (a)-EPS <= (b) )
#define gte(a,b) ( (a)+EPS >= (b) )
#define sq(a) ((a)*(a))

struct point {
   double x, y;
   point() { x = y = 0; }
   point( const point &drugi ) { *this = drugi; }
   point( double X, double Y ) { x = X; y = Y; }
};

struct segment {
   point a, b;
   segment() {}
   segment( const segment &drugi ) { *this = drugi; }
   segment( const point &A, const point &B ) { a = A; b = B; }
   segment( double X1, double Y1, double X2, double Y2 )
      { a = point( X1, Y1 ); b = point( X2, Y2 ); }
};

struct line { // !! Ax+By=C, a ne Ax+By+C=0 !!
   double A, B, C;
   line() { A = B = C = 0; }
   line( const line &drugi ) { *this = drugi; }
   line( double a, double b, double c ) { A = a; B = b; C = c; }
   line( const segment &S ) {
      A = S.b.y - S.a.y;
      B = S.a.x - S.b.x;
      C = A*S.a.x + B*S.a.y;
   }
   line( const point &A, const point &B ) { *this = line( segment( A, B ) ); }
   line( double fi, const point &T ) {
      A = sin( fi );
      B = -cos( fi );
      C = A*T.x + B*T.y;
   }
};

/*****************************************************/
int mysgn( double val ) {
   if( lt( val, 0 ) ) return -1;
   if( gt( val, 0 ) ) return 1;
   return 0;
}
double myhypot( double A, double B )
{  return sqrt( A*A + B*B );  }

pair<double,double> kvadratna( double A, double B, double C ) {
   if( eq(sq(B)-4*A*C, 0) ) return make_pair(-B/(2*A),-B/(2*A));
   return make_pair( (-B+sqrt(sq(B)-4*A*C))/(2*A),
                     (-B-sqrt(sq(B)-4*A*C))/(2*A) );
}
int strana( const line &L, const point &T )
{  return mysgn( L.A*T.x + L.B*T.y - L.C );  }

int ccw( const point &A, const point &B, const point &C )
{  return mysgn( (B.x-A.x)*(C.y-A.y) - (C.x-A.x)*(B.y-A.y) );  }

line okomica( const line &L, const point &T ) {
   line ret;
   ret.A = L.B; ret.B = -L.A;
```

```
   ret.C = ret.A*T.x + ret.B*T.y;
   return ret;
}
point midpoint( const point &A, const point &B )
{  return point( (A.x+B.x)/2, (A.y+B.y)/2 );  }

/*****************************************************/
bool intersects( const line &L, const point &P )
{  return eq( L.A*P.x + L.B*P.y - L.C, 0 ); }

bool intersects( const segment &S, const point &P ) {
   return intersects( line(S), P ) &&
      eq( fabs(S.b.x-P.x)+fabs(S.a.x-P.x), fabs(S.b.x-S.a.x) ) &&
      eq( fabs(S.b.y-P.y)+fabs(S.a.y-P.y), fabs(S.b.y-S.a.y) );
}
bool intersects( const line &L1, const line &L2 )
{  return !eq( L1.A*L2.B, L2.A*L1.B );  }

bool intersects( const line &L, const segment &S )
{  return strana( L, S.a ) * strana( L, S.b ) <= 0;  }

bool intersects( const segment &S1, const segment &S2 ) {
   if( intersects( line(S1), S2.a ) && intersects( line(S1), S2.b ) )
      return intersects( S1, S2.a ) || intersects( S1, S2.b ) ||
             intersects( S2, S1.a ) || intersects( S2, S1.b );
   return intersects( line(S1), S2 ) && intersects( line(S2), S1 );
}
point intersect( const line &L1, const line &L2 ) {
   return point( (L1.C*L2.B-L2.C*L1.B) / (L1.A*L2.B-L2.A*L1.B),
                 (L1.A*L2.C-L2.A*L1.C) / (L1.A*L2.B-L2.A*L1.B) );
}
point intersect( const line &L, const segment &S )
{  return intersect( L, line(S) ); }

point intersect( const segment &S1, const segment &S2 )
{  return intersect( line(S1), line(S2) ); }

/*****************************************************/
double dist( const point &A, const point &B )
{  return myhypot( A.x-B.x, A.y-B.y );  }

double dist( const line &L, const point &T )
{  return fabs(L.A*T.x + L.B*T.y - L.C) / myhypot( L.A, L.B ); }

double dist( const segment &S, const point &T ) {
   double a = dist( S.b, T ), b = dist( S.a, T ), c = dist( S.a, S.b );
   if( gte( sq(b), sq(a)+sq(c) ) ) return a;
   if( gte( sq(a), sq(b)+sq(c) ) ) return b;
   return dist( line( S ), T );
}
double dist( const segment &S1, const segment &S2 ) {
   if( intersects( S1, S2 ) ) return 0;
   return dist(S1,S2.a) <? dist(S1,S2.b) <?
          dist(S2,S1.a) <? dist(S2,S1.b);
}
double dist( const line &L1, const line &L2 ) {
   if( intersects( L1, L2 ) ) return 0;
   return dist( intersect( L1, okomica(L2,point()) ),
                intersect( L2, okomica(L1,point()) ) );
}
double dist( const line &L, const segment &S ) {
   if( intersects( L, S ) ) return 0;
   return dist( L, S.a ) <? dist( L, S.b );
}
/*****************************************************/
struct circle {
   point S;
   double r;
   circle() { S.x = S.y = r = 0; }
   circle( const circle &drugi ) { *this = drugi; }
   circle( double X, double Y, double R ) { S.x = X; S.y = Y; r = R; }
   circle( const point &T, double R ) { S = T; r = R; };
   circle( const point &A, const point &B, const point &C ) {
      S = intersect( okomica( line(A,B), midpoint(A,B) ),
                     okomica( line(B,C), midpoint(B,C) ) );
      r = dist( S, A );
   }
};

bool intersects( const circle &C, const line &L )
{  return lte( dist( L, C.S ), C.r );  }

bool intersects( const circle &C, const segment &S )
{  return lte( dist( S, C.S ), C.r );  }

bool intersects( const circle &C1, const circle &C2 ) {
   double d = dist( C1.S, C2.S );
   return !(lt(d, fabs(C1.r-C2.r)) || gt(d, C1.r + C2.r) ||
            eq(d, 0) && eq(C1.r, C2.r));
}
pair<point,point> intersect( circle C, line L ) {
   pair<double,double> X, Y;
   bool swp = eq( L.B, 0 );
   if( swp ) swap( L.A, L.B ), swap( C.S.x, C.S.y );
   X = kvadratna( sq(L.A)/sq(L.B) + 1,
       2*(L.A*C.S.y/L.B - L.A*L.C/sq(L.B) - C.S.x),
       sq(C.S.x) + sq(C.S.y) - sq(C.r) + sq(L.C)/sq(L.B) - 2*L.C*C.S.y/L.B );
   Y.first  = (L.C - L.A*X.first)/L.B;
   Y.second = (L.C - L.A*X.second)/L.B;
   if( swp ) swap( X, Y );
   return make_pair( point( X.first, Y.first ), point( X.second, Y.second ) );
}
pair<point,point> intersect( const circle &C1, const circle &C2 ) {
   double d = dist( C1.S, C2.S );
   double a = (sq(C1.r) - sq(C2.r) + sq(d)) / (2*d);
   point T( C1.S.x+a*(C2.S.x-C1.S.x)/d,
            C1.S.y+a*(C2.S.y-C1.S.y)/d );
   return intersect( C1, okomica( line(C1.S,C2.S), T ) );
}
pair<point,point> tangent_points( const circle &C, const point &P )
{  return intersect( C, circle( midpoint(P,C.S), dist(P,C.S)/2 ) );  }

pair<line,line> tangents( const circle &C, const point &P ) {
   pair<point,point> Q = tangent_points( C, P );
   return make_pair( line(P, Q.first), line(P, Q.second) );
}

/*************************************************
 *                  hullovi.cpp                  *
 *************************************************/

struct point {
   int x, y;
   point( int X=0, int Y=0 ) { x=X; y=Y; }
};

int dist( const point &a, const point &b ) { // overflow hazard!
   return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}

int ccw( const point &a, const point &b, const point &c ) {
   int d = (b.y-a.y)*(c.x-a.x) - (b.x-a.x)*(c.y-a.y);
   if( d < 0 ) return -1;
   if( d > 0 ) return 1;
   return 0;
}

struct hull {
   struct cmp {
      point o;
      cmp( point O ) { o = O; }
      bool operator () ( const point &a, const point &b ) {
         int t = ccw( o, a, b );
         if( t != 0 ) return t == -1;
         return dist( o, a ) < dist( o, b );
      }
   };
```

```
   vector<point> a;
   hull() {
      for( int i = 0; i < n; ++i ) {
         point p;
         scanf( "%d%d", &p.x, &p.y );
         a.push_back( p );

         if( a[i].y < a[0].y || a[i].y == a[0].y && a[i].x < a[0].x ) swap( a[i], a[0] );
      }
      sort( a.begin()+1, a.end(), cmp( a[0] ) );

      /**** za najmanje tocaka ****/
      if( n >= 3 ) {
         { // makni kolinearne
            int m = 1;
            for( int i = 1; i+1 < (int)a.size(); ++i )
               if( ccw( a[0], a[i], a[i+1] ) != 0 )
                  a[m++] = a[i];
            a[m++] = a[a.size()-1];
            a.resize( m );
         }

         { // graham scan
            int m = 2;
            for( int i = 2; i < (int)a.size(); a[m++] = a[i++] )
               while( ccw( a[m-2], a[m-1], a[i] ) != -1 ) --m;
            a.resize( m );
         }
      }
      a.push_back( a[0] );

      /**** za najvise tocaka ****/
      if( a.size() >= 3 ) {
         { // obrni kraj
            int i = a.size()-2;
            while( i > 0 && ccw( a[0], a[i], a[i+1] ) == 0 ) --i;
            if( i > 0 ) reverse( a.begin()+i+1, a.end() );
         }

         { // graham scan
            int m = 2;
            for( int i = 2; i < (int)a.size(); ++i ) {
               if( a[i].x == a[m-1].x && a[i].y == a[m-1].y ) continue;
               while( ccw( a[m-2], a[m-1], a[i] ) == 1 ) --m;
               a[m++] = a[i];
            }
            a.resize( m );
         }
         a.push_back( a[0] );
      }
   }
   int size() { return a.size()-1; }
   const point & operator () ( int i ) { return a[i]; }
};

/*********************************************
 *              javaprimjeri.java            *
 *********************************************/

import java.io.*;
import java.math.*;

public class Main {
   void solve() {
      /* ... */
   }

   public static void main(String[] args) throws IOException {
      Main solution = new Main();

      /* ucitavanje liniju po liniju: */
      BufferedReader in = new BufferedReader( new InputStreamReader(System.in) );
      String[] tok;
      tok = in.readLine().split(" ");
```

```
      /* ucitavanje cijele fajle */
      BufferedReader in = new BufferedReader( new InputStreamReader(System.in) );
      StringBuilder sb = new StringBuilder();
      char[] buf = new char[4096];
      int nread = 0;
      while ( (nread = in.read(buf, 0, buf.length)) > 0 ) {
         sb.append( buf, 0, nread );
      }

      /* komprimiranje whitespacea */
      s = s.replaceAll( "[ \t\r\n]+", " " );
   }
}

/*********************************************
 *                  kmp.cpp                  *
 *********************************************/

#include <cstdio>
#include <cstring>
#define MAXT 1000
#define MAXP 1000

int pi[MAXP+1];
char T[MAXT+1]; int n;
char P[MAXP+1]; int m;

void compute_prefix_function() {
   pi[1] = 0;
   int k = 0;
   for( int q = 2; q <= m; ++q ) {
      while( k > 0 && P[k] != P[q-1] ) k = pi[k];
      if( P[k] == P[q-1] ) k++;
      pi[q] = k;
   }
}

void KMP_matcher() {
   int q = 0;
   for( int i = 1; i <= n; ++i ) {
      while( q > 0 && P[q] != T[i-1] ) q = pi[q];
      if( P[q] == T[i-1] ) q++;
      if( q == m ) {
         // PATTERN JE NADJEN SA SHIFTOM i-m
         q = pi[q];
      }
   }
}

int main( void ) {
   scanf( "%s", T ); n = strlen( T );
   scanf( "%s", P ); m = strlen( P );
   compute_prefix_function();
   KMP_matcher();
   return 0;
}

/*********************************************
 *           konstantna_struktura.cc         *
 *********************************************/

#include <climits>
#include <cstdio>
using namespace std;

#define MAXN         100005
#define CEIL_LG_MAXN 17

static int key[MAXN];

int dub[MAXN], poc[MAXN];
int konst[CEIL_LG_MAXN+1][1<<CEIL_LG_MAXN];
```

```
void konst_stvori( int n ) // stvara strukturu sa key[0..n-1]
{
   int i;
   int nelem;
   __asm__ __volatile__ ( "bsrl %1,%0" : "=r"(nelem) : "rm"(n) );
   nelem = 1 << nelem;
   if ( n > nelem ) nelem <<= 1;

   for ( i=0; i<n; ++i )     konst[0][i] = i;
   for ( i=n; i<nelem; ++i ) konst[0][i] = n-1;

   for ( int level=1; ( 1 << (level-1) ) < nelem; ++level ) {
      int w = ( 1 << (level-1) );
      for ( int i=0; i+w+w<=nelem; ++i )
         konst[level][i] =
            key[ konst[level-1][i] ] < key[ konst[level-1][i+w] ] ?
            konst[level-1][i] : konst[level-1][i+w];
   }
}

int query( int a, int b ) // vraca minimum izmedu a i b, ukljucivo
{
   ++b;
   int z;
   __asm__ __volatile__ ( "bsrl %1,%0" : "=r"(z) : "rm"(b-a) );
   return
      key[ konst[z][a] ] < key[ konst[z][b-(1<<z)] ] ?
      konst[z][a] : konst[z][b-(1<<z)]; // vraca index minimuma
}

/*********************************************
 *                 matrix2.cpp                *
 *********************************************/

// kartezijevo stablo u ulozi queuea s daj_mi_min metodom

#include <cstdio>

int next[101];
int matrix[1000][1000]; // ulazna matrica
int sumAB[1000][1000]; // suma matrix[i-A+1 .. i][j-B+1 .. j]
int sumCD[1000][1000]; // suma matrix[i-C+1 .. i][j-D+1 .. j]
int mini[1000][1000]; // min sumCD[i-A-C .. i][j-B-D .. j]
int temp[1000][1000]; // temp
int n, m, A, B, C, D;

namespace cartesian {
   int root;
   int value[1000];
   int parent[1000];
   int left[1000];
   int right[1000];

   inline void push_front( int i ) {
      left[i] = right[i] = -1;
      if( root == -1 ) parent[i] = root; else parent[i] = i-1;

      while( parent[i] != -1 && value[i] < value[parent[i]] ) {
         left[i] = parent[i];
         parent[i] = parent[parent[i]];
      }
      if( left[i] != -1 ) parent[left[i]] = i;
      if( parent[i] != -1 ) right[parent[i]] = i;
      else root = i;
   }

   inline void pop_back( int i ) {
      if( parent[i] == -1 ) {
         root = right[i];
         if( right[i] != -1 ) parent[right[i]] = -1;
      } else {
         left[parent[i]] = right[i];
         if( right[i] != -1 ) parent[right[i]] = parent[i];
      }
   }
}
using namespace cartesian;

void calc_sum( int source[1000][1000], int target[1000][1000], int X, int Y ) {
   for( int c = 0; c < m; ++c ) {
      int running = 0;
      for( int r = 0; r < n; ++r ) {
         running += source[r][c];
         if( r-X >= 0 ) running -= source[r-X][c];
         temp[r][c] = running;
      }
   }

   for( int r = 0; r < n; ++r ) {
      int running = 0;
      for( int c = 0; c < m; ++c ) {
         running += temp[r][c];
         if( c-Y >= 0 ) running -= temp[r][c-Y];
         target[r][c] = running;
      }
   }
}
void calc_min( int source[1000][1000], int target[1000][1000], int X, int Y ) {
   for( int c = 0; c < m; ++c ) {
      root = -1;
      for( int r = 0; r < n; ++r ) {
         value[r] = source[r][c];
         push_front( r );
         if( r-X >= 0 ) pop_back( r-X );
         temp[r][c] = value[root];
      }
   }

   for( int r = 0; r < n; ++r ) {
      root = -1;
      for( int c = 0; c < m; ++c ) {
         value[c] = temp[r][c];
         push_front( c );
         if( c-Y >= 0 ) pop_back( c-Y );
         target[r][c] = value[root];
      }
   }
}

int main( void ) {
   for( int i = 0; i <= 100; ++i ) next[i] = (i*71+17)%100+1;

   int T;
   scanf( "%d", &T );
   for( int tt = 0; tt < T; ++tt ) {
      scanf( "%d%d%d%d%d%d", &n, &m, &A, &B, &C, &D );
      for( int i = 0; i < n; ++i ) {
         scanf( "%d", &matrix[i][0] );
         int last = matrix[i][0]%100;
         for( int j = 1; j < m; ++j )
            last = matrix[i][j] = next[last];
      }

      calc_sum( matrix, sumAB, A, B );
      calc_sum( matrix, sumCD, C, D );
      calc_min( sumCD, mini, A-C-1, B-D-1 );

      int ret = 0;
      for( int r = A-1; r < n; ++r )
         for( int c = B-1; c < m; ++c )
            ret >?= sumAB[r][c] - mini[r-1][c-1];

      printf( "%d\n", ret );
   }

   return 0;
}
```

```cpp
/*********************************************
 *              mincostflow.cpp              *
 *********************************************/

#include <cstdio>

using namespace std;

const double eps = 1e-8;

int n, a, b;
double inf;
int c[42][42];
double g[42][42];
double dist[42][43];
int how[42][43];

int bellman_ford() {
  for( int i = 0; i < n; ++i )
     for( int j = 0; j <= n; ++j )
        dist[i][j] = inf + 1;

  dist[0][0] = 0;

  for( int k = 1; k <= n; ++k ) {
     for( int i = 0; i < n; ++i )
        for( int j = 0; j < n; ++j )
           if( c[i][j] && dist[i][k-1] <= inf && dist[i][k-1] + g[i][j] < dist[j][k] ) {
              dist[j][k] = dist[i][k-1] + g[i][j];
              how[j][k] = i;
           }
     for( int a = 0; a < n; ++a ) {
        int i, j, kk = k;
        for( i = a; kk > 0; i = how[i][kk--] )
           if( kk != k && i == a ) break;
        if( i != a ) continue;
        if( dist[a][k] - dist[a][kk] >= -eps ) continue;
        for( j = a; k > kk; --k ) {
           i = how[j][k];
           c[i][j]--;
           c[j][i]++;
           j = i;
        }
        return 1;
     }
  }
  return 0;
}

int main( void ) {
  while( scanf( "%d%d", &a, &b ) == 2 ) {
     if( a == 0 && b == 0 ) break;

     inf = 0;
     n = a+b+2;
     for( int i = 0; i < n; ++i )
        for( int j = 0; j < n; ++j ) {
           c[i][j] = 0;
           g[i][j] = 0;
        }

     for( int i = 1; i <= a; ++i ) c[i+b][a+b+1] = 1;
     for( int j = 1; j <= b; ++j ) c[0][j] = 1;
     for( int i = 1; i <= a; ++i )
        for( int j = 1; j <= b; ++j ) {
           scanf( "%lf", &g[j][i+b] );
           inf += g[j][i+b];
           g[i+b][j] = -g[j][i+b];
           c[j][i+b] = 1;
        }
     inf *= n;

     g[a+b+1][0] = -inf; g[0][a+b+1] = inf;
     c[a+b+1][0] = 1000000000;

     while( bellman_ford() );

     double ret = 0;
     for( int i = 1; i <= a; ++i )
        for( int j = 1; j <= b; ++j )
           if( c[i+b][j] )
              ret -= g[i+b][j];

     printf( "%.2lf\n", ret/a + eps );
  }
  return 0;
}

/*********************************************
 *       mincostmaxflow s vise bridova.cc    *
 *********************************************/

// mislim da svi bridovi moraju kostati 1

#define MAXV 105

typedef int flow_t;
int V;

flow_t cap[MAXV][MAXV], flow[MAXV][MAXV], dist[MAXV];
int via[MAXV];
bool vis[MAXV];

struct Edge {
   int a, b, w, cap, reverse;
   Edge(int a, int b, int w, int cap, int reverse) : a(a), b(b), w(w), cap(cap), reverse(
reverse) { }
};

const int INF = 100000000;

void bellmanford(const vector<Edge> &graf, int source, int sink) {
   fill(dist, dist+V, INF);
   dist[source] = 0;

   for ( int iter=0; iter<V-1; ++iter ) {
      for ( int i=0; i<(int)graf.size(); ++i ) {
         const Edge &e = graf[i];
         if ( e.cap <= 0 ) continue;
         const int a = e.a, b = e.b, w = e.w;
         if ( dist[a]+w < dist[b] ) {
            dist[b] = dist[a]+w;
            via[b] = i;
         }
      }
   }
}

int minimumcostflow(vector<Edge> graf, int source, int sink, int minflow) {
   int weight = 0, ret = 0;
   fill(via, via+V, -1);
   while (true) {
      bellmanford(graf, source, sink);
      if ( dist[sink] == INF )
         break;

      int aug = INT_MAX;
      fill(vis, vis+V, 0);
      for ( int p=sink; p!=source; ) {
         if ( vis[p] ) {
            aug = 0;
            break;
         }
         vis[p] = true;
         aug = min(aug, graf[via[p]].cap);
         p = graf[via[p]].a;
      }
      if ( aug == 0 ) break;
```

```
      for ( int p=sink; p!=source; ) {
         int dad = graf[via[p]].a;
         graf[via[p]].cap -= aug;
         graf[graf[via[p]].reverse].cap += aug;
         flow[dad][p] += aug;
         flow[p][dad] -= aug;
         p = dad;
      }
      ret += aug;
      weight += aug * dist[sink];
   }
   return ret == minflow ? weight : -1;
}

struct ChessMatchup {
   void add_edge( vector<Edge> &graf, int a, int b, int w ) {
      Edge e1( a, b,  w, 1, graf.size()+1 );
      Edge e2( b, a, -w, 0, graf.size() );
      graf.push_back(e1);
      graf.push_back(e2);
      ++cap[a][b];
   }

   int maximumScore( vector <int> us, vector <int> them ) { // caret here
      int n = (int)us.size();
      V = 2*n + 2;
      int src = V-2, sink = V-1;

      memset( cap, 0, sizeof cap );
      memset( flow, 0, sizeof flow );

      vector<Edge> graf;
      for ( int i=0; i<n; ++i ) {
         add_edge(graf, src, i, 0);
         for ( int j=0; j<n; ++j )
            add_edge(graf, i, n+j, us[i] > them[j] ? -2 : us[i] == them[j] ? -1 : 0);
         add_edge(graf, n+i, sink, 0);
      }

      return -minimumcostflow(graf, src, sink, n);
   }
};

/*********************************************
 *             ortogonal_range.cpp           *
 *********************************************/

// orthogonal-range stablo u zadatku sa stablima sa SPOJA

#include <cstdio>
#include <cstdlib>

using namespace std;

#define MAX 100000
#define CEIL_LOG_MAX 17

const int inf = 2000000000;

struct node {
   int target;
   node *next;
} memorija[2*MAX], *adj[MAX], *alokator = memorija;

int n;
int label[MAX+1];

int discovery[MAX], finish[MAX], traversal_time = 0;
int niz[MAX];

void dfs( int x, int dad ) {
   discovery[x] = traversal_time++;
   for( node *it = adj[x]; it; it = it->next ) {
      if( it->target == dad ) continue;

      dfs( it->target, x );
   }
   finish[x] = traversal_time;
}

namespace ttree {
   struct cvor_u_listi {
      int value, index;
      cvor_u_listi *left, *right;
   } cvorovi[MAX*(CEIL_LOG_MAX+1)], *alloc = cvorovi;

   struct cvor_u_stablu {
      int lo, hi;
      cvor_u_listi *lista;
   } tree[2<<CEIL_LOG_MAX];

   void generiraj( int x, int lo, int hi ) {
      int size = hi-lo;
      tree[x].lo = lo;
      tree[x].hi = hi;
      if( hi-lo == 1 ) {
         tree[x].lista = alloc;
         alloc->value = niz[lo]; alloc->left = alloc->right = NULL; alloc->index = 0; ++a
lloc;
         alloc->value = n; alloc->left = alloc->right = NULL; alloc->index = 1; ++alloc;
      } else {
         int mid = (lo+hi)/2;
         generiraj( x*2, lo, mid );
         generiraj( x*2+1, mid, hi );

         tree[x].lista = alloc;
         cvor_u_listi *L = tree[x*2].lista;
         cvor_u_listi *R = tree[x*2+1].lista;

         for( int i = 0; i < size; ++i ) {
            alloc->left = L; alloc->right = R;
            alloc->index = i;
            if( label[L->value] < label[R->value] ) alloc->value = L++->value;
            else alloc->value = R++->value;
            ++alloc;
         }
         alloc->value = n; alloc->left = L; alloc->right = R; alloc->index = size; ++allo
c;
      }
   }

   struct queue_stuff {
      int x;
      cvor_u_listi *y;
   } Q[4*(CEIL_LOG_MAX+1)], *qhead, *qtail;

   int daj_mi_ktog_iz_intervala( int k, int a, int b ) {

      int root = 1;
      while( tree[root].hi - tree[root].lo > 1 ) {
         if( tree[root*2].hi <= a ) root = root*2+1;
         else if( tree[root*2+1].lo >= b ) root = root*2;
         else break;
      }

      cvor_u_listi *lo = tree[root].lista + k, *hi = tree[root].lista + n;

      while( lo != hi ) {
         cvor_u_listi *mid = lo + ((hi-lo)/2);

         int ret = 0;

         qhead = qtail = Q;
         qtail->x = root; qtail->y = mid; ++qtail;
         for( ; qhead != qtail; ++qhead ) {
            if( tree[qhead->x].hi <= a || tree[qhead->x].lo >= b ) continue;
            if( tree[qhead->x].lo >= a && tree[qhead->x].hi <= b ) {
```

```
               ret += qhead->y->index;
               continue;
            }

            qtail->x = qhead->x*2; qtail->y = qhead->y->left; ++qtail;
            qtail->x = qhead->x*2+1; qtail->y = qhead->y->right; ++qtail;
         }

         if( ret >= k ) hi = mid;
         else lo = mid + 1;
      }

      return (lo-1)->value;
   }
};

int main( void ) {
   scanf( "%d", &n );
   for( int i = 0; i < n; ++i ) {
      scanf( "%d", &label[i] );
      adj[i] = NULL;
   }
   label[n] = inf;

   for( int i = 1; i < n; ++i ) {
      int a, b;
      scanf( "%d%d", &a, &b ); --a; --b;

      alokator->target = b; alokator->next = adj[a]; adj[a] = alokator++;
      alokator->target = a; alokator->next = adj[b]; adj[b] = alokator++;
   }

   dfs( 0, -1 );

   for( int i = 0; i < n; ++i ) niz[discovery[i]] = i;

   ttree::generiraj( 1, 0, n );

   int m;
   scanf( "%d", &m );
   for( int i = 0; i < m; ++i ) {
      int x, k;
      scanf( "%d%d", &x, &k ); --x;

      printf( "%d\n", ttree::daj_mi_ktog_iz_intervala( k, discovery[x], finish[x] )+1 );
   }

   return 0;
}

/*********************************************
 *                permutacije.cc             *
 *********************************************/

#include <vector>
using namespace std;

long long fakt[] = { 1 };

// permutacija -> indeks

long long rank( const vector<int> &v )
{
   int n = (int)v.size();
   long long ret = 0;
   for ( int i=0; i<n; ++i ) {
      int manjih=0;
      for ( int j=i+1; j<n; ++j )
         manjih += v[j] < v[i];
      ret += manjih * fakt[n-i-1];
   }
   return ret;
}
```

```
// indeks -> permutacija

template<typename T>
vector<T> unrank( vector<T> v, long long rank )
{
   int n = (int)v.size();
   for ( int i=0; i<n; ++i ) {
      int p = rank/fakt[n-i-1];
      rank -= p*fakt[n-i-1];
      rotate( v.begin()+i, v.begin()+i+p, v.begin()+i+p+1 );
   }
   return v;
}

void generiraj_fakt()
{
   long long x = 1;
   printf( "long long fakt[] = { 1LL" );
   for ( int i=1; i<21; ++i )
      printf( ",%lldLL", x*=i );
   printf( " };\n" );
}

int main()
{
   generiraj_fakt();
}

/*********************************************
 *                primality.cpp              *
 *********************************************/

#include <cstdio>

typedef long long llint;

// racuna (a*b)%mod... radi za mod do 10^16
llint mymult( llint a, llint b, llint mod ) {
   static llint p[32];
   p[0] = 1 % mod;
   for( int i = 1; i < 32; ++i ) p[i] = (p[i-1] << 4) % mod;

   llint ret = 0;
   for( int i = 0; i < 16; ++i )
      for( int j = 0; j < 16; ++j )
         ret = (ret + ((a>>(i*4))&0xF) * ((b>>(j*4))&0xF) * p[i+j]) % mod;

   return ret;
}

// racuna (a^n)%mod... radi za mod do 10^16
llint mypow( llint a, llint n, llint mod ) {
   if( n == 0 ) return 1%mod;
   if( n % 2 ) return mymult( a, mypow( a, n-1, mod ), mod );
   llint x = mypow( a, n>>1, mod );
   return mymult( x, x, mod );
}

// provjerava dal je n strong probable prime to base a... radi za n do 10^16
int strong_probable_prime( llint n, llint a ) {
   if( n <= 1 ) return 0;
   if( n == a ) return 1;

   int s = 0;
   while( (n-1) % (1LL<<(s+1)) == 0 ) ++s;
   llint d = (n-1) / (1LL<<s);

   if( mypow( a, d, n ) == 1 ) return 1;
   for( int r = 0; r < s; ++r )
      if( mypow( a, d*(1LL<<r), n ) == n-1 ) return 1;

   return 0;
}
```

```
// provjerava je li n prost... radi tocno za n do 10^16
int is_prime( llint n ) {
   if( n == 1 ) return 0;
   if( n != 2 && n % 2 == 0 ) return 0;
   if( n == 46856248255981LL ) return 0;
   if( strong_probable_prime( n, 2 ) == 0 ) return 0;
   if( strong_probable_prime( n, 3 ) == 0 ) return 0;
   if( strong_probable_prime( n, 7 ) == 0 ) return 0;
   if( strong_probable_prime( n, 61 ) == 0 ) return 0;
   if( strong_probable_prime( n, 24251 ) == 0 ) return 0;
   return 1;
}

// Ako zatreba, Miller's Test kaze:
// If the extended Riemann hypothesis is true, then if n is an a-SPRP for
// all integers a with 1 < a < 2(log n)^2, then n is prime.

int main( void ) {
   long long n = 382370000000000LL;
   while( !is_prime( n ) ) ++n;
   printf( "%lld\n", n );

   return 0;
}

/*********************************************
 *               radix_sort.cc               *
 *********************************************/

void counting_sort(const vector<unsigned> &A, vector<unsigned> &B, int key) {
   int n = A.size();
   vector<unsigned> cnt(256, 0);

   for (int i=0; i<n; ++i)
      ++cnt[(A[i] >> key) & 0xFF];

   for (int i=1; i<256; ++i)
      cnt[i] += cnt[i-1];

   for (int i=n-1; i>=0; --i)
      B[--cnt[(A[i] >> key) & 0xFF]] = A[i];
}

void radix_sort(vector<unsigned> &A) {
   vector<unsigned> B( A.size() );
   for (int key=0; key<32; key+=8) {
      counting_sort(A, B, key);
      A.swap(B);
   }
}

/*********************************************
 *                rbtree1.cpp                *
 *********************************************/

// rbtree, zadatak neki sa stablima sa spoja

#include <cstdio>
#include <cstdlib>

using namespace std;

#define MAX 100000
#define MAXQ 10000

const int inf = 2000000000;

struct node {
   node *left, *right, *parent;
   bool left_child, red;

   int value;
   int size;
   int index;
   inline void link_left( node *y ) { left = y; y->parent = this; y->left_child = 1; }
   inline void link_right( node *y ) { right = y; y->parent = this; y->left_child = 0; }
   inline void update_statistics() { size = 1 + left->size + right->size; }
} memory[4*MAX], *z;
node *allocator = memory;

struct rbtree {
   node *root;

   rbtree() {
      root = allocator++;
      root->left = root->right = root->parent = z;
      root->size = 1;
      root->value = -inf;
      root->red = 0;
   }

   void rotate( node *x, node *p ) {
      bool x_left = x->left_child;

      if( p->left_child ) p->parent->link_left( x );
      else p->parent->link_right( x );

      if( x_left ) {
         p->link_left( x->right );
         x->link_right( p );
      } else {
         p->link_right( x->left );
         x->link_left( p );
      }
      p->update_statistics();
      x->update_statistics();
   }

   void split( node *x ) {
      x->red = 1; x->left->red = x->right->red = 0;
      if( x->parent->red ) {
         node *p = x->parent, *g = p->parent;
         if( x->left_child != p->left_child ) { rotate( x, p ); p = x; }
         rotate( p, g );
         p->red = 0;
         g->red = 1;
      }
      root->left->red = root->right->red = 0;
   }

   void validate( node *x ) {
      while( x != z ) {
         if( x->left->red && x->right->red ) split( x );
         x->update_statistics();
         x = x->parent;
      }
   }

   node * element_at( int k ) {
      node *x = root;

      while( x != z ) {
         if( x->left->size > k ) {
            x = x->left;
         } else {
            k -= x->left->size;
            if( k-- == 0 ) break;
            x = x->right;
         }
      }
      return x;
   }

   node * create( int index, int val ) {
      node *x = allocator++;
      x->left = x->right = z;
      x->size = 1;
```

```
      x->value = val; x->index = index;
      return x;
   }

   void insert( node *t ) {
      t->left = t->right = z;

      node *x = root;
      for( ;; ) {
         if( t->value < x->value ) {
            if( x->left == z ) { x->link_left( t ); break; }
            else x = x->left;
         } else {
            if( x->right == z ) { x->link_right( t ); break; }
            else x = x->right;
         }
      }
      split( t );
      validate( t );
   }
};

int k[MAXQ], answer[MAXQ];

struct list_node {
   int val;
   list_node *next;
};
list_node memorija[2*MAX+MAXQ], *alokator = memorija;
list_node *adj[MAX], *queries[MAX];

int n, m;
int label[MAX+1];

void merge( node *x, rbtree RB ) {
   if( x == z ) return;
   merge( x->left, RB );
   merge( x->right, RB );
   RB.insert( x );
}

rbtree dfs( int x, int dad ) {
   rbtree JA;
   JA.insert( JA.create( x, label[x] ) );
   for( list_node *it = adj[x]; it; it = it->next ) {
      if( it->val == dad ) continue;

      rbtree ON = dfs( it->val, x );

      if( JA.root->size < ON.root->size ) {
         merge( JA.root->right, ON );
         JA.root = ON.root;
      } else {
         merge( ON.root->right, JA );
      }
   }

   for( list_node *it = queries[x]; it; it = it->next )
      answer[it->val] = JA.element_at( k[it->val] )->index+1;

   return JA;
}

int main( void ) {
   z = allocator++;
   z->left = z->right = z->parent = z;
   z->size = 0;
   z->value = inf;
   z->red = 0;

   scanf( "%d", &n );
   for( int i = 0; i < n; ++i ) {
      scanf( "%d", &label[i] );
      adj[i] = NULL;
      queries[i] = NULL;
   }
   label[n] = inf;

   for( int i = 1; i < n; ++i ) {
      int a, b;
      scanf( "%d%d", &a, &b ); --a; --b;

      alokator->val = b; alokator->next = adj[a]; adj[a] = alokator++;
      alokator->val = a; alokator->next = adj[b]; adj[b] = alokator++;
   }

   int m;
   scanf( "%d", &m );
   for( int i = 0; i < m; ++i ) {
      int x;
      scanf( "%d%d", &x, &k[i] ); --x;
      alokator->val = i; alokator->next = queries[x]; queries[x] = alokator++;
   }

   dfs( 0, -1 );

   for( int i = 0; i < m; ++i ) printf( "%d\n", answer[i] );

   return 0;
}

/*********************************************
 *                 rbtree2.cpp                *
 *********************************************/

// rbtree zadatak ZMASINA

#include <cstdio>

const int inf = 1000000000;

struct node {
   node *left, *right, *parent;
   bool left_child, red;

   int value;            // data
   int size; bool alive; // order
   int min;              // statistic

   inline void link_left( node *y ) { left = y; y->parent = this; y->left_child = 1; }
   inline void link_right( node *y ) { right = y; y->parent = this; y->left_child = 0; }

   inline void update_statistics() {
      size = alive + left->size + right->size;
      min = left->min <? right->min <? (alive ? value : inf);
   }
} memory[200002];
node *allocator = memory;

struct rbtree {
   node *root, *z;

   rbtree() {
      z = allocator++;
      z->left = z->right = z->parent = z;
      z->size = z->alive = 0;
      z->min = z->value = inf;
      z->red = 0;

      root = allocator++;
      root->left = root->right = root->parent = z;
      root->size = root->alive = 1;
      root->min = root->value = inf;
      root->red = 0;
   }

   void rotate( node *x, node *p ) {
      bool x_left = x->left_child;
```

```
   if( p->left_child ) p->parent->link_left( x );
   else p->parent->link_right( x );

   if( x_left ) {
      p->link_left( x->right );
      x->link_right( p );
   } else {
      p->link_right( x->left );
      x->link_left( p );
   }
   p->update_statistics();
   x->update_statistics();
}

void split( node *x ) {
   x->red = 1; x->left->red = x->right->red = 0;
   if( x->parent->red ) {
      node *p = x->parent, *g = p->parent;
      if( x->left_child != p->left_child ) { rotate( x, p ); p = x; }
      rotate( p, g );
      p->red = 0;
      g->red = 1;
   }
   root->left->red = root->right->red = 0;
}

void validate( node *x ) {
   while( x != z ) {
      if( x->left->red && x->right->red ) split( x );
      x->update_statistics();
      x = x->parent;
   }
}

node * element_at( int k ) {
   node *x = root;

   while( x != z ) {
      if( x->left->size > k ) {
         x = x->left;
      } else {
         k -= x->left->size;
         if( x->alive && k == 0 ) break;
         if( x->alive ) --k;
         x = x->right;
      }
   }
   return x;
}

node * create( int val ) {
   node *x = allocator++;
   x->left = x->right = z;
   x->size = x->alive = 1;
   x->min = x->value = val;
   return x;
}

void erase( node *x ) {
   x->alive = 0;
   validate( x );
}

node * insert_after( node *x, int val ) {
   node *t = create( val );
   if( !x->right->size ) {
      x->link_right( t );
   } else {
      x = x->right;
      while( x->left->size ) x = x->left;
      x->link_left( t );
   }
   split( t );
   validate( t );
   return t;
}

node * insert_before( node *x, int val ) {
   node *t = create( val );
   if( !x->left->size ) {
      x->link_left( t );
   } else {
      x = x->left;
      while( x->right->size ) x = x->right;
      x->link_right( t );
   }
   split( t );
   validate( t );
   return t;
}

int a, b;
int min( node *x, int lo, int hi ) {
   if( lo > hi ) return inf;
   if( b < lo || a > hi ) return inf;
   if( lo >= a && hi <= b ) return x->min;
   int ret = inf;
   if( x->alive ) {
      int mid = lo + x->left->size;
      if( mid >= a && mid <= b ) ret <?= x->value;
   }
   ret <?= min( x->left, lo, hi - x->alive - x->right->size );
   ret <?= min( x->right, lo + x->alive + x->left->size, hi );
   return ret;
}
int min( int a, int b ) {
   this->a = a; this->b = b;
   return min( root, 0, root->size-1 );
}
};

int main( void ) {
   int n;
   scanf( "%d", &n );

   rbtree niz;

   for( int i = 0; i < 100000; ++i )
      niz.insert_after( niz.element_at(i), inf );

   for( int i = 0; i < n; ++i ) {
      int op, x, val;
      scanf( "%d %d", &op, &x );
      if( op == 1 ) {
         scanf( "%d", &val );
         niz.insert_before( niz.element_at(x), val );
      } else {
         printf( "%d\n", niz.min( 1, x ) );
      }
   }

   return 0;
}

/*********************************************
 *                  roof.cpp                 *
 *********************************************/

#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <set>
#include <queue>
#include <vector>

using namespace std;
```

```
const int inf = 1000000000;

int n;
int minx, miny, maxx, maxy;
struct point {
   int x, y;
   int index;
} p[1000];

struct rect {
   int x1, x2;
   int y1, y2;
   rect( int X1, int X2, int Y1, int Y2 ) { x1=X1; x2=X2; y1=Y1; y2=Y2; }
};
vector<rect> A, B;

namespace rectangularizacija {
   struct v_segment {
      int x, y1, y2;
      v_segment( int X, int Y1, int Y2 ) { x=X; y1=Y1; y2=Y2; }
   };
   vector<v_segment> E;
   bool operator < ( const v_segment &A, const v_segment &B ) { return A.y1 < B.y1; }

   bool cmp_x( const v_segment &A, const v_segment &B ) { return A.x < B.x; }

   void init() {
      B.clear();
      E.clear();
      for( int i = 0; i < n; ++i ) {
         int j = (i+1==n) ? 0 : i+1;
         if( p[i].x == p[j].x )
            E.push_back( v_segment( p[i].x, p[i].y <? p[j].y, p[i].y >? p[j].y ) );
      }
   }

   void sweep() {
      sort( E.begin(), E.end(), cmp_x );

      set<v_segment> S;
      S.insert( v_segment( -inf, -inf, -inf ) );
      S.insert( v_segment( inf, inf, inf ) );

      queue<v_segment> add;
      for( vector<v_segment>::iterator it = E.begin(); it != E.end(); ++it ) {
         set<v_segment>::iterator a = --S.upper_bound( *it ), b;

         for( ;; ) {
            v_segment v = *a;
            b = a; ++b;

            int puno_lo = it->y1 >? v.y1;
            int puno_hi = it->y2 <? v.y2;

            if( puno_lo < puno_hi ) {
               B.push_back( rect( v.x, it->x, puno_lo, puno_hi ) );
               S.erase( a );
               if( v.y1 != puno_lo ) add.push( v_segment( v.x, v.y1, puno_lo ) );
               if( v.y2 != puno_hi ) add.push( v_segment( v.x, puno_hi, v.y2 ) );
            }

            int prazno_lo = it->y1 >? v.y2;
            int prazno_hi = it->y2 <? b->y1;

            if( prazno_lo < prazno_hi ) add.push( v_segment( it->x, prazno_lo, prazno_hi
) );

            if( prazno_hi == it->y2 ) break;

            a = b;
         }

         for( ; !add.empty(); add.pop() ) S.insert( add.front() );
      }
   }
};

namespace pokrivac {
   namespace interval_tree {
      struct node {
         int propagiraj_pokriveno, propagiraj_upit;
         int pokriveno, greska;
         char redoslijed;
      } tree[1<<15];
      int mx;

      void update_pokriveno_then_upit( int i, int lo, int hi, int y1, int y2, int x_pokri
veno, int x_upit );
      void update_upit_then_pokriveno( int i, int lo, int hi, int y1, int y2, int x_upit,
 int x_pokriveno );

      inline void update( int i ) {
         tree[i].pokriveno = tree[2*i].pokriveno <? tree[2*i+1].pokriveno;
         tree[i].greska >?= tree[2*i].greska >? tree[2*i+1].greska;
      }

      inline void propagiraj( int i, int lo, int hi, int mid ) {
         if( tree[i].redoslijed == 0 ) {
            update_pokriveno_then_upit( 2*i, lo, mid, lo, mid, tree[i].propagiraj_pokrive
no, tree[i].propagiraj_upit );
            update_pokriveno_then_upit( 2*i+1, mid+1, hi, mid+1, hi, tree[i].propagiraj_p
okriveno, tree[i].propagiraj_upit );
         } else {
            update_upit_then_pokriveno( 2*i, lo, mid, lo, mid, tree[i].propagiraj_upit, t
ree[i].propagiraj_pokriveno );
            update_upit_then_pokriveno( 2*i+1, mid+1, hi, mid+1, hi, tree[i].propagiraj_u
pit, tree[i].propagiraj_pokriveno );
         }
      }

      void init( int i, int lo, int hi ) {
         tree[i].propagiraj_upit = tree[i].propagiraj_pokriveno = -inf;
         tree[i].pokriveno = tree[i].greska = -inf;
         tree[i].redoslijed = 0;
         if( lo < hi ) {
            int mid = (lo+hi)/2;
            init( 2*i, lo, mid );
            init( 2*i+1, mid+1, hi );
         }
      }

      void update_pokriveno_then_upit( int i, int lo, int hi, int y1, int y2, int x_pokri
veno, int x_upit ) {
         if( y2 < lo || y1 > hi ) return;

         if( lo >= y1 && hi <= y2 ) {
            if( x_pokriveno > tree[i].pokriveno ) {
               tree[i].pokriveno = x_pokriveno;
               tree[i].propagiraj_pokriveno = x_pokriveno;
               tree[i].redoslijed = 1;
            }
            if( x_upit > tree[i].pokriveno ) {
               tree[i].greska >?= x_upit;
               tree[i].propagiraj_upit = x_upit;
               tree[i].redoslijed = 0;
            }
         } else {
            int mid = (lo+hi)/2;
            propagiraj( i, lo, hi, mid );
            update_pokriveno_then_upit( 2*i, lo, mid, y1, y2, x_pokriveno, x_upit );
            update_pokriveno_then_upit( 2*i+1, mid+1, hi, y1, y2, x_pokriveno, x_upit );
            update( i );
         }
      }

      void update_upit_then_pokriveno( int i, int lo, int hi, int y1, int y2, int x_upit,
 int x_pokriveno ) {
         if( y2 < lo || y1 > hi ) return;
```

```
        if( lo >= y1 && hi <= y2 ) {
           if( x_upit > tree[i].pokriveno ) {
              tree[i].greska >?= x_upit;
              tree[i].propagiraj_upit = x_upit;
              tree[i].redoslijed = 0;
           }
           if( x_pokriveno > tree[i].pokriveno ) {
              tree[i].pokriveno = x_pokriveno;
              tree[i].propagiraj_pokriveno = x_pokriveno;
              tree[i].redoslijed = 1;
           }
        } else {
           int mid = (lo+hi)/2;
           propagiraj( i, lo, hi, mid );
           update_upit_then_pokriveno( 2*i, lo, mid, y1, y2, x_upit, x_pokriveno );
           update_upit_then_pokriveno( 2*i+1, mid+1, hi, y1, y2, x_upit, x_pokriveno );
           update( i );
        }
     }

     int query_greska( int i, int lo, int hi, int y1, int y2, int x_upit ) {
        if( y2 < lo || y1 > hi ) return -inf;

        if( lo >= y1 && hi <= y2 ) {
           if( x_upit > tree[i].pokriveno ) {
              tree[i].greska >?= x_upit;
              tree[i].propagiraj_upit = x_upit;
              tree[i].redoslijed = 0;
           }
           return tree[i].greska;
        } else {
           int mid = (lo+hi)/2;
           propagiraj( i, lo, hi, mid );
           int ret = query_greska( 2*i, lo, mid, y1, y2, x_upit ) >? query_greska( 2*i+1
, mid+1, hi, y1, y2, x_upit );
           update( i );
           return ret;
        }
     }
  };

  struct event {
     int x, type, index;
     int y1, y2;
     event( int X, int T, int I, int Y1, int Y2 ) { x=X; type=T; index=I; y1=Y1; y2=Y2;
}
  };
  bool operator < ( const event &A, const event &B ) { return A.x < B.x; }

  vector<event> E;

  void sazmi() {
     static int mapa[200001];

     vector<int> yyy;
     for( vector<event>::iterator it = E.begin(); it != E.end(); ++it ) {
        yyy.push_back( it->y1 );
        yyy.push_back( it->y2 );
     }

     sort( yyy.begin(), yyy.end() );
     yyy.erase( unique( yyy.begin(), yyy.end() ), yyy.end() );

     interval_tree::mx = yyy.size();
     for( int i = 0; i < interval_tree::mx; ++i ) mapa[yyy[i]] = i+1;

     for( vector<event>::iterator it = E.begin(); it != E.end(); ++it ) {
        it->y1 = mapa[it->y1];
        it->y2 = mapa[it->y2];
     }
  }

  void init() {
     E.clear();
     for( int i = 0; i < A.size(); ++i ) E.push_back( event( A[i].x1, 0, i, A[i].y1, A[i
].y2 ) );
     for( int i = 0; i < B.size(); ++i ) E.push_back( event( B[i].x2, 1, i, B[i].y1, B[i
].y2 ) );
     sazmi();
     sort( E.begin(), E.end() );
     interval_tree::init( 1, 1, interval_tree::mx );
  }

  int sweep() {
     for( vector<event>::iterator it = E.begin(); it != E.end(); ++it ) {
        if( it->type == 0 ) {
           interval_tree::update_upit_then_pokriveno( 1, 1, interval_tree::mx, it->y1, i
t->y2-1, A[it->index].x1, A[it->index].x2 );
        } else {
           if( interval_tree::query_greska( 1, 1, interval_tree::mx, it->y1, it->y2-1, B
[it->index].x2 ) > B[it->index].x1 ) return 0;
        }
     }
     return 1;
  }
};

int main( void ) {
  int T;
  scanf( "%d", &T );
  for( int tt = 0; tt < T; ++tt ) {
     scanf( "%d", &n );

     minx = inf; miny = inf; maxx = -inf; maxy = -inf;
     for( int i = 0; i < n; ++i ) {
        scanf( "%d%d", &p[i].x, &p[i].y ); p[i].x *= 2; p[i].y *= 2;
        p[i].index = i;
        minx <?= p[i].x; maxx >?= p[i].x;
        miny <?= p[i].y; maxy >?= p[i].y;
     }

     rectangularizacija::init();
     rectangularizacija::sweep();

     int lo = 1, hi = ((maxx-minx) <? (maxy-miny))/2;

     while( lo < hi ) {
        int mid = (lo+hi)/2;

        A.clear();
        for( int i = 0; i < n; ++i ) {
           int j = (i+1==n) ? 0 : i+1;
           A.push_back( rect( (p[i].x <? p[j].x) - mid, (p[i].x >? p[j].x) + mid,
                              (p[i].y <? p[j].y) - mid >? miny, (p[i].y >? p[j].y) + mid
 <? maxy ) );
        }

        pokrivac::init();
        if( pokrivac::sweep() ) hi = mid; else lo = mid+1;
     }

     printf( "%.1lf\n", lo/2.0 );
  }
  return 0;
}

/*********************************************
 *                 slalom.cpp                *
 *********************************************/

// zadatak slalom u O(n)

#include <cmath>
#include <cstdio>
#include <queue>

using namespace std;
```

```
struct point { int x, y; };

point A[10002];
int strana[10002];
double dp[10002];

int orient( point a, point b, point c ) {
   int t = (b.x-a.x)*(c.y-a.y) - (c.x-a.x)*(b.y-a.y);
   if( t < 0 ) return -1;
   if( t > 0 ) return 1;
   return 0;
}

double dist( point a, point b ) {
   return sqrt( (double)(a.x-b.x)*(a.x-b.x) + (double)(a.y-b.y)*(a.y-b.y) );
}

int main( void ) {
   int n;
   scanf( "%d", &n );
   scanf( "%d%d", &A[0].x, &A[0].y );
   for( int i = 0; i < n; ++i ) {
      scanf( "%d%d%d", &A[2*i+1].x, &A[2*i+2].x, &A[2*i+1].y );
      A[2*i+2].y = A[2*i+1].y;
      strana[2*i+1] = 1;
      strana[2*i+2] = 2;
   }
   scanf( "%d%d", &A[2*n+1].x, &A[2*n+1].y ); strana[2*n+1] = 1;

   n = 2*n+2;

   int apex = 0;
   deque<int> funnel;
   funnel.push_back( 0 );
   funnel.push_front( 1 );
   funnel.push_back( 2 );
   dp[0] = 0.0;
   dp[1] = dist( A[0], A[1] );
   dp[2] = dist( A[0], A[2] );

   int cw = orient( A[1], A[0], A[2] );
   int ccw = -cw;

   for( int i = 3; i < n; ++i ) {
      if( strana[i] == 1 ) {
         int prva = funnel.front(); funnel.pop_front();
         while( prva != apex ) {
            if( funnel.empty() || orient( A[funnel.front()], A[prva], A[i] ) == cw ) brea
k;
            prva = funnel.front(); funnel.pop_front();
         }
         while( prva == apex ) {
            if( funnel.empty() || orient( A[prva], A[funnel.front()], A[i] ) == cw ) brea
k;
            apex = prva = funnel.front(); funnel.pop_front();
         }
         funnel.push_front( prva );
         dp[i] = dp[prva] + dist( A[prva], A[i] );
         funnel.push_front( i );
      } else {
         int prva = funnel.back(); funnel.pop_back();
         while( prva != apex ) {
            if( funnel.empty() || orient( A[funnel.back()], A[prva], A[i] ) == ccw ) brea
k;
            prva = funnel.back(); funnel.pop_back();
         }
         while( prva == apex ) {
            if( funnel.empty() || orient( A[prva], A[funnel.back()], A[i] ) == ccw ) brea
k;
            apex = prva = funnel.back(); funnel.pop_back();
         }
         funnel.push_back( prva );
         dp[i] = dp[prva] + dist( A[prva], A[i] );
         funnel.push_back( i );
      }
   }

   printf( "%.4lf\n", dp[n-1] );

   return 0;
}

/*******************************************
 *            stable_marriage.cpp          *
 *******************************************/

// stable_marriage

#include <algorithm>
#include <cstdio>
#include <map>
#include <queue>
#include <string>
#include <vector>

using namespace std;

int n, m;
int k;
int A[100][100];
int B[100][100];
int perm[100][100];
vector<string> namesA; map<string,int> numA;
vector<string> namesB; map<string,int> numB;

int tipA[100];
int tipB[100];

int p[100];
int veza[100];

char buff[16];

struct cmp {
  int i;
  cmp( int I ) { i=I; }

  bool operator () ( const int &x, const int &y ) {
     return A[tipA[i]][tipB[x]] > A[tipA[i]][tipB[y]];
  }
};

int main( void ) {
  for( int tt = 1; ; ++tt ) {
     scanf( "%d", &n );
     if( n == 0 ) break;
     for( int i = 0; i < n; ++i ) {
        scanf( "%s", buff ); string s( buff );
        namesA.push_back( s ); numA[s] = i;
     }
     scanf( "%d", &m );
     if( m == 0 ) break;
     for( int i = 0; i < m; ++i ) {
        scanf( "%s", buff ); string s( buff );
        namesB.push_back( s ); numB[s] = i;
     }

     for( int i = 0; i < n; ++i )
        for( int j = 0; j < m; ++j )
           scanf( "%d", &A[i][j] );

     for( int i = 0; i < m; ++i )
        for( int j = 0; j < n; ++j )
           scanf( "%d", &B[i][j] );

     for( int mm = 1; scanf( "%d", &k ) == 1; ++mm ) {
        if( k == 0 ) break;
```

```
        for( int i = 0; i < k; ++i ) {
           scanf( "%s", buff ); string s( buff );
           string aa = s.substr( 0, 2 );
           string bb = s.substr( 2, 2 );
           tipA[i] = numA[aa];
           tipB[i] = numB[bb];
        }

        for( int i = 0; i < k; ++i ) {
           for( int j = 0; j < k; ++j )
              perm[i][j] = j;
           sort( perm[i], perm[i] + k, cmp(i) );
        }

        queue<int> Q;
        for( int i = 0; i < k; ++i ) {
           veza[i] = -1;
           p[i] = 0;
           Q.push( i );
        }

        while( !Q.empty() ) {
           int x = Q.front(); Q.pop();
           for( ;; ) {
              int y = perm[tipA[x]][p[x]++];
              if( veza[y] == -1 ) {
                 veza[y] = x;
                 break;
              } else if( B[tipB[y]][tipA[x]] > B[tipB[y]][tipA[veza[y]]] ) {
                 Q.push( veza[y] );
                 veza[y] = x;
                 break;
              }
           }
        }

        printf( "Scenario %d, Mixture %d:\n", tt, mm );
        for( int i = 0; i < k; ++i ) {
           if( i ) printf( " " );
           printf( "%s%s", namesA[tipA[veza[i]]].c_str(), namesB[tipB[i]].c_str() );
        }
        printf( "\n\n" );
     }
  }


  return 0;
}

/*********************************************
 *               stoerwagner.cpp             *
 *********************************************/

#include <algorithm>
#include <cstdio>
#include <vector>

using namespace std;

#define MAX 150
const int inf = 1000000000;

int n, m;
vector<int> alive;
int g[MAX][MAX];
int w[MAX];
int bio[MAX];

// O(|V|^2) je slozenost... moze se ubrzat sa setom na O( (|V|+|E|)log|V| )
int mincutphase() {
   for( vector<int>::iterator it = alive.begin(); it != alive.end(); ++it ) {
      bio[*it] = 0;
      w[*it] = 0;
   }
   w[alive[0]] = inf;

   int s = -1, t = -1;
   for( int i = 0; i < alive.size(); ++i ) {
      int m = -1;
      for( vector<int>::iterator it = alive.begin(); it != alive.end(); ++it )
         if( !bio[*it] && (m == -1 || w[*it] > w[m]) ) m = *it;

      bio[m] = 1;
      if( i + 2 == alive.size() ) s = m;
      if( i + 1 == alive.size() ) t = m;

      for( vector<int>::iterator it = alive.begin(); it != alive.end(); ++it )
         if( !bio[*it] )
            w[*it] += g[*it][m];
   }

   alive.erase( find( alive.begin(), alive.end(), t ) );
   int ret = 0;
   for( vector<int>::iterator it = alive.begin(); it != alive.end(); ++it ) {
      ret += g[*it][t];
      if( *it == s ) continue;
      g[*it][s] += g[*it][t];
      g[s][*it] += g[t][*it];
   }
   return ret;
}

// O( |V|^3 ) je slozenost
int mincut() {
   if( n <= 1 ) return 0;
   alive.clear();
   for( int i = 0; i < n; ++i ) alive.push_back( i );

   int ret = inf;
   while( alive.size() > 1 ) ret <?= mincutphase();
   return ret;
}

int main( void ) {
   int T;
   scanf( "%d", &T );
   for( int tt = 1; tt <= T; ++tt ) {
      scanf( "%d%d", &n, &m );

      for( int i = 0; i < n; ++i )
         for( int j = 0; j < n; ++j )
            g[i][j] = 0;

      for( int i = 0; i < m; ++i ) {
         int a, b, c;
         scanf( "%d%d%d", &a, &b, &c ); --a; --b;
         g[a][b] += c;
         g[b][a] += c;
      }

      printf( "Case #%d: %d\n", tt, mincut() );
   }
   return 0;
}

/*********************************************
 *                  sudoku.cpp               *
 *********************************************/

// sudoku s dancing linksima

#include <cstdio>
#include <cstring>

using namespace std;

struct node {
```

```
   int L, R, U, D;
   int C, size;
   int row, col, num;

   node() {}
   node( int x ) { L = R = U = D = C = x; size = 0; }
} table[20000];
char a[16][512];

// Dancing links implementacija algoritma X
void cover_column( int c ) {
   table[table[c].R].L = table[c].L;
   table[table[c].L].R = table[c].R;

   for( int i = table[c].D; i != c; i = table[i].D )
      for( int j = table[i].R; j != i; j = table[j].R ) {
         table[table[j].D].U = table[j].U;
         table[table[j].U].D = table[j].D;
         table[table[j].C].size --= 1;
      }
}
void uncover_column( int c ) {
   for( int i = table[c].U; i != c; i = table[i].U )
      for( int j = table[i].L; j != i; j = table[j].L ) {
         table[table[j].C].size += 1;
         table[table[j].D].U = j;
         table[table[j].U].D = j;
      }

   table[table[c].R].L = c;
   table[table[c].L].R = c;
}
int rec() {
   if( table[0].R == 0 ) return 1;

   int c = table[0].R;
   for( int i = table[c].R; i != 0; i = table[i].R )
      if( table[i].size < table[c].size ) c = i;

   if( table[c].size == 0 ) return 0;

   cover_column( c );

   for( int r = table[c].D; r != c; r = table[r].D ) {
      a[table[r].row][table[r].col] = table[r].num + 'A';

      for( int j = table[r].R; j != r; j = table[j].R ) cover_column( table[j].C );
      if( rec() ) return 1;
      for( int j = table[r].L; j != r; j = table[j].L ) uncover_column( table[j].C );
   }

   uncover_column( c );

   return 0;
}
// Kraj dancinga linksa

int n;
int rowcol[16][16];
int rownum[16][16];
int colnum[16][16];
int blonum[16][16];

// inicijalizacija tablice - svedemo na exact set problem
void insert_column( int c ) {
   table[c].R = 0;
   table[c].L = table[0].L;
   table[table[c].R].L = c;
   table[table[c].L].R = c;
}
void insert_cell_in_column( int x, int c ) {
   table[x].C = c;
   table[c].size += 1;
   table[x].D = c;
   table[x].U = table[c].U;
   table[table[x].D].U = x;
   table[table[x].U].D = x;
}
void init_table() {
   n = 0;
   table[n] = node(n), n++;
   for( int k = 0; k < 4; ++k )
      for( int i = 0; i < 16; ++i )
         for( int j = 0; j < 16; ++j ) {
            table[n] = node(n);
            insert_column( n++ );
         }

   int c[4], x[4];
   for( int row = 0; row < 16; ++row )
      for( int col = 0; col < 16; ++col )
         for( int num = 0; num < 16; ++num ) {
            int blo = (row/4)*4 + col/4;

            c[0] = 1 + 0*16*16 + row*16 + col;
            c[1] = 1 + 1*16*16 + row*16 + num;
            c[2] = 1 + 2*16*16 + col*16 + num;
            c[3] = 1 + 3*16*16 + blo*16 + num;

            for( int i = 0; i < 4; ++i ) {
               table[n] = node(n);
               x[i] = n++;
            }

            for( int i = 0; i < 4; ++i ) {
               table[x[i]].row = row;
               table[x[i]].col = col;
               table[x[i]].num = num;

               table[x[i]].L = x[(i+3)%4];
               table[x[i]].R = x[(i+1)%4];
               insert_cell_in_column( x[i], c[i] );
            }
         }


   for( int row = 0; row < 16; ++row )
      for( int col = 0; col < 16; ++col )
         if( rowcol[row][col] )
            cover_column( 1 + 0*16*16 + row*16 + col );

   for( int row = 0; row < 16; ++row )
      for( int num = 0; num < 16; ++num )
         if( rownum[row][num] )
            cover_column( 1 + 1*16*16 + row*16 + num );

   for( int col = 0; col < 16; ++col )
      for( int num = 0; num < 16; ++num )
         if( colnum[col][num] )
            cover_column( 1 + 2*16*16 + col*16 + num );

   for( int blo = 0; blo < 16; ++blo )
      for( int num = 0; num < 16; ++num )
         if( blonum[blo][num] )
            cover_column( 1 + 3*16*16 + blo*16 + num );

}
// kraj inicijalizacije

void input() {
   memset( rowcol, 0, sizeof rowcol );
   memset( rownum, 0, sizeof rownum );
   memset( colnum, 0, sizeof colnum );
   memset( blonum, 0, sizeof blonum );
   for( int row = 0; row < 16; ++row ) {
      scanf( "%s", a[row] );
```

```
      for( int col = 0; col < 16; ++col ) {
         if( a[row][col] == '-' ) continue;

         int num = a[row][col] - 'A';
         int blo = (row/4)*4 + col/4;

         rowcol[row][col] = 1;
         rownum[row][num] = 1;
         colnum[col][num] = 1;
         blonum[blo][num] = 1;
      }
   }
}

int main( void ) {
   int T;
   scanf( "%d", &T );
   for( int tt = 0; tt < T; ++tt ) {
      if( tt ) printf( "\n" );

      input();

      init_table();

      if( rec() )
         for( int row = 0; row < 16; ++row ) {
            a[row][16] = 0;
            printf( "%s\n", a[row] );
         }
   }
   return 0;
}

/*********************************************
 *               suffix_array.cpp            *
 *********************************************/

// Given a string, we need to find the total number of its distinct substrings.

#include <cstdio>
#include <cstring>

using namespace std;

int rjesenje;

int n;
int tree[1<<17];
char S[1<<16];

int Pos[1<<16];
int PosInv[1<<16];
int Count[1<<16];
bool StartH[1<<16];
bool Start2H[1<<16];

void tree_init( int i, int lo, int hi ) {
   tree[i] = n+1;
   if( hi - lo == 1 ) return;

   int mid = (lo+hi)>>1;

   tree_init( i<<1, lo, mid );
   tree_init( i<<1|1, mid, hi );
}


void tree_update( int i, int lo, int hi, int target, int value ) {
   tree[i] <?= value;
   if( hi - lo == 1 ) return;

   int mid = (lo+hi)>>1;

   if( target <= mid ) tree_update( i<<1, lo, mid, target, value );
   else tree_update( i<<1|1, mid, hi, target, value );
}

int tree_query( int i, int lo, int hi, int a, int b ) {
   if( lo >= b || hi <= a ) return n+1;
   if( lo >= a && hi <= b ) return tree[i];

   int mid = (lo+hi)>>1;

   return tree_query( i<<1, lo, mid, a, b ) <? tree_query( i<<1|1, mid, hi, a, b );
}

int Bucket[128];
int Link[1<<16];
void bucket_sort() {
   for( int i = 0; i < 128; ++i ) Bucket[i] = -1;
   for( int i = 0; i < n; ++i ) {
      int x = S[i];
      Link[i] = Bucket[x];
      Bucket[x] = i;
   }
   int k = 0;
   for( int i = 0; i < 128; ++i )
      for( int j = Bucket[i]; j != -1; j = Link[j] )
         Pos[k++] = j;
   Pos[n] = n;
}

void init_suffix_array() {
   bucket_sort();
   for( int i = 0; i <= n; ++i ) PosInv[Pos[i]] = i;

   tree_init( 1, 0, n-1 );

   Start2H[0] = StartH[0] = true;
   for( int i = 1; i < n; ++i )
      if( S[Pos[i]] == S[Pos[i-1]] ) {
         Start2H[i] = StartH[i] = false;
      } else {
         Start2H[i] = StartH[i] = true;
         tree_update( 1, 0, n-1, i, 0 );
      }
   Start2H[n] = StartH[n] = true;

   for( int H = 1; H < n; H *= 2 ) {

      for( int i = 0; i < n; ++i ) {
         if( StartH[i] ) Count[i] = 0;
         else PosInv[Pos[i]] = PosInv[Pos[i-1]];
      }

      int x = n - H;
      PosInv[x] += Count[PosInv[x]]++;
      Start2H[PosInv[x]] = true;

      for( int i = 0, j; i < n; i = j ) {

         for( j = i+1; !StartH[j]; ++j );

         for( int k = i; k < j; ++k ) {
            int x = Pos[k] - H;
            if( x < 0 ) continue;
            PosInv[x] += Count[PosInv[x]]++;
            Start2H[PosInv[x]] = true;
         }

         for( int k = i; k < j; ++k ) {
            int x = Pos[k] - H;
            if( x < 0 || !Start2H[PosInv[x]] ) continue;
            for( int y = PosInv[x] + 1; !StartH[y] && Start2H[y]; ++y ) Start2H[y] = fals
e;
         }
      }
```

```
      for( int i = 0; i < n; ++i ) Pos[PosInv[i]] = i;

      for( int i = 0; i < n; ++i )
         if( Start2H[i] && !StartH[i] ) {
            StartH[i] = true;
            int a = PosInv[Pos[i-1] + H];
            int b = PosInv[Pos[i] + H];
            int h = H;
            if( a != n ) h += tree_query( 1, 0, n-1, a <? b, a >? b );
            tree_update( 1, 0, n-1, i, h );
            rjesenje -= h;
         }
   }
}


int main( void ) {
   int T;
   scanf( "%d", &T );

   for( int tt = 0; tt < T; ++tt ) {
      scanf( "%s", S );
      n = strlen( S );

      if( n == 1 ) { printf( "1\n" ); continue; }

      rjesenje = (long long)n*(n+1)/2;

      init_suffix_array();

      printf( "%d\n", rjesenje );
   }
   return 0;
}

/*********************************************
 *             tournament_tree.cc            *
 *********************************************/

#include <algorithm>
using namespace std;

#define MAXN 100005

int key[MAXN+1];
int tree_pointer;
int tree[2*MAXN+1], tree_left[2*MAXN+1], tree_rajt[2*MAXN+1];

int _tt_stvori( int a, int b )
{
   int node = tree_pointer++;
   if ( a == b ) {
      tree[node] = a;
   }
   else {
      tree_left[node] = _tt_stvori( a, (a+b)/2 );
      tree_rajt[node] = _tt_stvori( (a+b)/2+1, b );
      // vraca index minimuma
      tree[node] = ( key[ tree[tree_left[node]] ] < key[ tree[tree_rajt[node]] ] ? tree[t
ree_left[node]] : tree[tree_rajt[node]] );
   }
   return node;
}

int _tt_query( int node, int a, int b, int qa, int qb )
{
   if ( qa == a && qb == b ) return tree[node];

   int mid = (a+b)/2;
   if ( qa <= mid && qb > mid ) {
      int l = _tt_query( tree_left[node], a    , mid, qa   , mid );
      int r = _tt_query( tree_rajt[node], mid+1, b  , mid+1, qb  );
      return key[l] < key[r] ? l : r;
   }
   else if ( qa <= mid )
      return _tt_query( tree_left[node], a, mid, qa, qb );
   else
      return _tt_query( tree_rajt[node], mid+1, b, qa, qb );
}

void tt_stvori( int n )
{
   tree_pointer = 0;
   _tt_stvori( 1, n );
}

int tt_query( int n, int a, int b )
{
   return _tt_query( 0, 1, n, a, b );
}

/*********************************************
 *                 ugarska.cpp               *
 *********************************************/

#include <cmath>
#include <cstdio>
#include <queue>

using namespace std;

#define MAX_R 20 // mora biti >= MAX_C
#define MAX_C 20
#define VELIKO 1000000000 // svi costovi moraju biti manji

bool zero( int x ) { return x == 0; }
bool zero( double x ) { return fabs(x) < 1e-12; }

template <typename tip>
struct hungarian {
   int n, m;
   tip costs[MAX_R][MAX_C]; // pocetne vrijednosti NE OSTAJU ocuvane
   bool ret[MAX_R][MAX_C]; // na kraju, jedinice su matching
   int stars;
   int star_r[MAX_R], star_c[MAX_C];
   int prime_r[MAX_R], prime_c[MAX_C];
   int cover_r[MAX_R], cover_c[MAX_C];

   void matching() {
      for( ; n < m; ++n )
         for( int c = 0; c < m; ++c )
            costs[n][c] = 0;
      for( int r = 0; r < n; ++r ) { star_r[r] = -1; cover_r[r] = 0; }
      for( int c = 0; c < m; ++c ) { star_c[c] = -1; cover_c[c] = 0; }
      stars = 0;
      step1();
   }

   void step1() {
      for( int r = 0; r < n; ++r ) {
         tip mini = VELIKO;
         for( int c = 0; c < m; ++c ) mini <?= costs[r][c];
         for( int c = 0; c < m; ++c ) costs[r][c] -= mini;
      }
      step2();
   }
   void step2() {
      for( int r = 0; r < n; ++r ) {
         for( int c = 0; c < m; ++c ) {
            if( star_c[c] != -1 ) continue;
            if( !zero( costs[r][c] ) ) continue;
            star_r[r] = c;
            star_c[c] = r;
            ++stars;
            break;
         }
      }
      step3();
```

```
   }
   void step3() {
      if( stars == m ) {
         for( int r = 0; r < n; ++r )
            for( int c = 0; c < m; ++c )
               ret[r][c] = (star_r[r] == c);
         return; // zavrsetak algoritma
      }
      for( int r = 0; r < n; ++r ) cover_r[r] = 0;
      for( int c = 0; c < m; ++c ) cover_c[c] = star_c[c] != -1;

      step4();
   }
   void step4() {
      queue<int> Q;
      for( int c = 0; c < m; ++c ) if( !cover_c[c] ) Q.push( c );

      for( ; !Q.empty(); Q.pop() ) {
         int c = Q.front();
         for( int r = 0; r < n; ++r ) {
            if( cover_r[r] ) continue;
            if( !zero( costs[r][c] ) ) continue;
            if( star_r[r] != -1 ) {
               cover_c[star_r[r]] = 0;
               cover_r[r] = 1;
               prime_r[r] = c;
               prime_c[c] = r;
               Q.push( star_r[r] );
            } else {
               step5( r, c );
               return;
            }
         }
      }
      tip mini = VELIKO;
      for( int r = 0; r < n; ++r )
         if( !cover_r[r] )
            for( int c = 0; c < m; ++c )
               if( !cover_c[c] )
                  mini <?= costs[r][c];
      step6( mini );
   }
   void step5( int r, int c ) {
      while( star_c[c] != -1 ) {
         int tmp_r = star_c[c];
         star_r[r] = c;
         star_c[c] = r;
         c = prime_r[tmp_r];
         r = tmp_r;
      }
      star_r[r] = c;
      star_c[c] = r;
      stars++;
      step3();
   }
   void step6( tip mini ) {
      for( int r = 0; r < n; ++r )
         for( int c = 0; c < m; ++c )
            if( cover_r[r] && cover_c[c] ) costs[r][c] += mini;
            else if( !cover_r[r] && !cover_c[c] ) costs[r][c] -= mini;
      step4();
   }
};

int C[20][20];
int main( void ) {
   hungarian<int> H;
   scanf( "%d%d", &H.n, &H.m );
   for( int r = 0; r < H.n; ++r )
      for( int c = 0; c < H.m; ++c )
         scanf( "%d", &H.costs[r][c] );

   H.matching();
   int cost = 0;
   for( int r = 0; r < H.n; ++r ) {
      for( int c = 0; c < H.m; ++c ) {
         printf( "%d", H.ret[r][c] );
         if( H.ret[r][c] ) cost += C[r][c];
      }
      printf( "\n" );
   }
   printf( "Cost: %d\n", cost );

   return 0;
}

/*********************************************
 *                union-find.cc              *
 *********************************************/

int dad[MAXN], rank[MAXN];
// int kids[MAXN]; // korisno za trazenje najvece komponente

int union_find( int a, int b, bool spoji=true )
{
   int topa, topb;
   int newtop;

   for ( topa=a; topa!=dad[topa]; topa=dad[topa] ) ;
   for ( topb=b; topb!=dad[topb]; topb=dad[topb] ) ;
   dad[a] = topa; dad[b] = topb;

   if ( topa != topb && spoji ) {
      if ( rank[topa] > rank[topb] ) {
         // kids[topa] += kids[topb];
         dad[topb] = newtop = topa;
      }
      else {
         // kids[topb] += kids[topa];
         dad[topa] = newtop = topb;
         if ( rank[topa] == rank[topb] ) rank[topb]++;
      }

      int x;
      for ( ; a!=topa; ) x=dad[a], dad[a] = newtop, a=x;
      for ( ; b!=topb; ) x=dad[b], dad[b] = newtop, b=x;

      return SPOJIO;
   }
   else {
      int x;
      for ( ; a!=topa; ) x=dad[a], dad[a] = topa, a=x;
      for ( ; b!=topb; ) x=dad[b], dad[b] = topb, b=x;

      return spoji || topa == topb ? VEC_SPOJENI : NISU_SPOJENI ;
   }
}

void union_find_init()
{
   // postavi dad[i] na i
   // postavi rank[i] na 0
   // postavi kids[i] na 1
}
```