Contents

1 Non-code things

1.1 Hash

Hash: 9538616d87aa2d06c37c129736430a98

```
tr -d '[:space:]' | md5sum | cut -d ' ' -f 1
```

1.2 Makefile

Hash: 1be30703415446aaf3a1260294222d71

```
CXX = g++
CXXFLAGS = -Wall -Wextra -pedantic -std=c++11 -O2
    ↪ -Wshadow -Wformat=2 -Wfloat-equal
    ↪ -Wconversion -Wlogical-op -Wshift-overflow=2
    ↪ -Wduplicated-cond -Wcast-qual -Wcast-align
DEBUGFLAGS = -D_GLIBCXX_DEBUG
    ↪ -D_GLIBCXX_DEBUG_PEDANTIC -fsanitize=address
    ↪ -fsanitize=undefined
    ↪ -fno-sanitize-recover=all -fstack-protector
    ↪ -D_FORTIFY_SOURCE=2

CXXFLAGS += $(DEBUGFLAGS)

TARGET := $(notdir $(CURDIR))
EXECUTE := ./$(TARGET)

CASES := $(sort $(basename $(wildcard *.in)))
TESTS := $(sort $(basename $(wildcard *.out)))

all: $(TARGET)

clean:
  -rm -rf $(TARGET) *.res

%: %.cpp
  $(LINK.cpp) $< $(LOADLIBES) $(LDLIBS) -o $@

run: $(TARGET)
  time $(EXECUTE)

%.res: $(TARGET) %.in
  time $(EXECUTE) < $*.in > $*.res

%.out: %

test_%: %.res %.out
  diff $*.res $*.out

runs: $(patsubst %,%.res,$(CASES))
test: $(patsubst %,test_%,$(TESTS))

.PHONY: all clean run test test_% runs

.PRECIOUS: %.res
```

1.3 vimrc

Hash: 8f870abf0ba8837fb91734ae9a941ba8

```
set nocp ai bs=2 cul hls ic is lbr ls=2 mouse=a nu
    ↪ ru sc scs smd so=3 sw=4 ts=4
filetype plugin indent on
syntax on

map gA m'ggVG"+y''
```

1.4 nanorc

Hash: 4364dc56fff2b10d5aacd6dc61625802

```
set tabsize 4
set const
set autoindent
```

2 Geometry

2.1 Point

Hash: a1ef04616fa78cdafb4e4425490521b7

```
/**
 * Author: Ulf Lundstrom
 * Date: 2009-02-26
 * License: CC0
 * Source: My head with inspiration from tinyKACTL
 * Description: Class to handle points in the plane.
 *  T can be e.g. double or long long. (Avoid int.)
 * Status: Works fine, used a lot
 */
#pragma once

template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) <
      ↪ tie(p.x,p.y); }
  bool operator==(P p) const { return
      ↪ tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return
      ↪ (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return
      ↪ sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes
      ↪ dist()=1
  P perp() const { return P(-y, x); } // rotates +90
      ↪ degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around
      ↪ the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
};
```

3 Data Structure

Hash: d01d504b98495ea3340d6d2f6a6c6ffd

```
////////////////////////////
//
// LCT
//
////////////////////////////
struct T {
  bool rr;
  T *son[2], *pf, *fa;
} f1[N], *ff = f1, *f[N], *null;

void downdate(T *x) {
  if (x -> rr) {
    x -> son[0] -> rr = !x -> son[0] -> rr;
    x -> son[1] -> rr = !x -> son[1] -> rr;
    swap(x -> son[0], x -> son[1]);
    x -> rr = false;
  }
  // add stuff
}
```

```
void update(T *x) {
  // add stuff
}

void rotate(T *x, bool t) {
  T *y = x -> fa, *z = y -> fa;
  if (z != null)  z -> son[z -> son[1] == y] = x;
  x -> fa = z;
  y -> son[t] = x -> son[!t];
  x -> son[!t] -> fa = y;
  x -> son[!t] = y;
  y -> fa = x;
  update(y);
}

void xiao(T *x) {
  if (x -> fa != null)  xiao(x -> fa), x -> pf = x
      ↪ -> fa -> pf;
  downdate(x);
}

void splay(T *x) {
  xiao(x);
  T *y, *z;
  while (x -> fa != null) {
    y = x -> fa; z = y -> fa;
    bool t1 = (y -> son[1] == x), t2 = (z -> son[1]
        ↪ == y);
    if (z != null) {
      if (t1 == t2) rotate(y, t2), rotate(x, t1);
      else  rotate(x, t1), rotate(x, t2);
    }else rotate(x, t1);
  }
  update(x);
}

void access(T *x) {
  splay(x);
  x -> son[1] -> pf = x;
  x -> son[1] -> fa = null;
  x -> son[1] = null;
  update(x);
  while (x -> pf != null) {
    splay(x -> pf);
    x -> pf -> son[1] -> pf = x -> pf;
    x -> pf -> son[1] -> fa = null;
    x -> pf -> son[1] = x;
    x -> fa = x -> pf;
    splay(x);
  }
  x -> rr = true;
}

bool Cut(T *x, T *y) {
  access(x);
  access(y);
  downdate(y);
  downdate(x);
  if (y -> son[1] != x || x -> son[0] != null)
    return false;
  y -> son[1] = null;
  x -> fa = x -> pf = null;
  update(x);
  update(y);
  return true;
}

bool Connected(T *x, T *y) {
  access(x);
  access(y);
  return x == y || x -> fa != null;
}

bool Link(T *x, T *y) {
  if (Connected(x, y))
    return false;
  access(x);
  access(y);
  x -> pf = y;
  return true;
}


int main() {
  read(n); read(m); read(q);
  null = new T; null -> son[0] = null -> son[1] =
      ↪ null -> fa = null -> pf = null;
  for (int i = 1; i <= n; i++) {
    f[i] = ++ff;
    f[i] -> son[0] = f[i] -> son[1] = f[i] -> fa =
        ↪ f[i] -> pf = null;
    f[i] -> rr = false;
  }
  // init null and f[i]
}
```

4 String

Hash: f566cda2e4994762e460a8553dc79ff4

```
////////////////////////////
//
//SAM
//
////////////////////////////
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
int
    ↪ n,i,init,L,len,ll,q,h,ch,p,last[1700000],n1[1700000],du[1700000],s[1700000],fa[800001],l[1700000],son[1700000
char S[8000001],k;
long long ans,sum[1600001];
void ins(int p,int ss,int k)
{
  int np=++len,q,nq;
  l[np]=l[p]+1;
  s[np]=1;
  while (p&&!son[p][k]) son[p][k]=np,p=par[p];
  if (!p) par[np]=1;
  else {
    q=son[p][k];
    if (l[p]+1==l[q]) par[np]=q;
    else {
      nq=++len;
      l[nq]=l[p]+1;
      s[nq]=0;
      memset(son[nq], son[q], sizeof son[q]);
      par[nq]=par[q];
      par[q]=nq;
      par[np]=nq;
      while (p&&son[p][k]==q) son[p][k]=nq,p=par[p];
    }
  }
  last[ss]=np;
}
int main()
{
  read(n);
  last[1]=init=len=1;
  for (i=2;i<=n;i++)
  {
    read(fa[i]);
    for (k=getchar();k<=32;k=getchar());
    ins(last[fa[i]],i,k-'a');
  }
}
```

5 Math

Hash: 82f0550573a34efcbcbaa2487bc74766

```
////////////////////////////
//SIMPLEX
//WARNING: segfaults on empty (size 0)
//max cx st Ax<=b, x>=0
//do 2 phases; 1st check feasibility;
//2nd check boundedness & ans
////////////////////////////
vector<double> simplex(vector<vector<double> > A,
    ↪ vector<double> b, vector<double> c) {
  int n = (int) A.size(), m = (int) A[0].size()+1, r
      ↪ = n, s = m-1;
  vector<vector<double> > D = vector<vector<double>
      ↪ > (n+2, vector<double>(m+1));
  vector<int> ix = vector<int> (n+m);
  for (int i=0; i<n+m; i++) ix[i] = i;
  for (int i=0; i<n; i++) {
    for (int j=0; j<m-1; j++)D[i][j]=-A[i][j];
    D[i][m-1] = 1;
    D[i][m] = b[i];
    if (D[r][m] > D[i][m]) r = i;
  }
  for (int j=0; j<m-1; j++) D[n][j]=c[j];
  D[n+1][m-1] = -1; int z = 0;
  for (double d;;) {
    if (r < n) {
      swap(ix[s], ix[r+m]);
      D[r][s] = 1.0/D[r][s];
      for (int j=0; j<=m; j++) if (j!=s) D[r][j] *=
          ↪ -D[r][s];
      for(int i=0; i<=n+1; i++)if(i!=r) {
        for (int j=0; j<=m; j++) if(j!=s) D[i][j] +=
            ↪ D[r][j] * D[i][s];
        D[i][s] *= D[r][s];
      }
    }
    r = -1; s = -1;
    for (int j=0; j <m; j++) if (s<0 || ix[s]>ix[j])
        ↪ {
```

```
      if (D[n+1][j]>eps || D[n+1][j]>-eps &&
          ↪ D[n][j]>eps) s = j;
    }
    if (s < 0) break;
    for (int i=0; i<n; i++) if(D[i][s]<-eps) {
      if (r < 0 || (d =
          ↪ D[r][m]/D[r][s]-D[i][m]/D[i][s]) < -eps
        || d < eps && ix[r+m] > ix[i+m]) r=i;
    }
    if (r < 0) return vector<double>(); // unbounded
  }
  if (D[n+1][m] < -eps) return vector<double>(); //
      ↪ infeasible
  vector<double> x(m-1);
  for (int i = m; i < n+m; i ++) if (ix[i] < m-1)
      ↪ x[ix[i]] = D[i-m][m];
  printf("%.2lf\n", D[n][m]);
  return x; // ans: D[n][m]
}
```

6 Graph

Hash: 56383bfdb29dfc826d0462e99b723479

```
//// Max clique N<64. Bit trick for speed
/**
* WishingBone's ACM/ICPC Routine Library
* maximum clique solver
*/
// clique solver calculates both size and
    ↪ consitution of maximum clique
// uses bit operation to accelerate searching
// graph size limit is 63, the graph should be
    ↪ undirected
// can optimize to calculate on each component, and
    ↪ sort on vertex degrees
// can be used to solve maximum independent set
class clique {
  public:
  static const long long ONE = 1;
  static const long long MASK = (1 << 21) - 1;
  char* bits;
  int n, size, cmax[63];
  long long mask[63], cons;
  // initiate lookup table
  clique() {
    bits = new char[1 << 21];
    bits[0] = 0;
    for (int i = 1; i < (1<<21); ++i)
      bits[i] = bits[i >> 1] + (i & 1);
  }
  ~clique() {
    delete bits;
  }
  // search routine
  bool search(int step,int siz,LL mor,LL con);
  // solve maximum clique and return size
  int sizeClique(vector<vector<int> >& mat);
  // solve maximum clique and return set
  vector<int>getClq(vector<vector<int> >&mat);
};
// step is node id, size is current sol., more is
    ↪ available mask, cons is constitution mask
bool clique::search(int step, int size,
                    LL more, LL cons) {
  if (step >= n) {
    // a new solution reached
    this->size = size;
    this->cons = cons;
    return true;
  }
  long long now = ONE << step;
  if ((now & more) > 0) {
    long long next = more & mask[step];
    if (size + bits[next & MASK] +
        bits[(next >> 21) & MASK] +
        bits[next >> 42] >= this->size
     && size + cmax[step] > this->size) {
      // the current node is in the clique
      if (search(step+1,size+1,next,cons|now))
        return true;
    }
  }
  long long next = more & ~now;
  if (size + bits[next & MASK] +
      bits[(next >> 21) & MASK] +
      bits[next >> 42] > this->size) {
    // the current node is not in the clique
    if (search(step + 1, size, next, cons))
      return true;
  }
  return false;
}
// solve maximum clique and return size
int clique::sizeClique(vector<vector<int> >& mat) {
  n = mat.size();
  // generate mask vectors
  for (int i = 0; i < n; ++i) {
    mask[i] = 0;
    for (int j = 0; j < n; ++j)
      if (mat[i][j] > 0) mask[i] |= ONE << j;
  }
  size = 0;
  for (int i = n - 1; i >= 0; --i) {
    search(i + 1, 1, mask[i], ONE << i);
    cmax[i] = size;
  }
  return size;
}
// calls sizeClique and restore cons
vector<int> clique::getClq(
    vector<vector<int> >& mat) {
  sizeClique(mat);
  vector<int> ret;
  for (int i = 0; i < n; ++i)
    if ((cons&(ONE<<i)) > 0) ret.push_back(i);
  return ret;
}
```

7 Java/Python

7.1 Java IO

7.2 Java BigInteger

7.3 Python IO