ACM Template

UESTC_Duiming

Last build at October 26, 2017

Contents

1 Datastructure

1.1 Fenwick

```
/* Fenwick Tree (Binary Indexed Tree), by Abreto <m@abreto.net>. */

#define MAXN 100001
#define LOWBIT(x)   ((x)&(-(x)))

int N;
int fen[MAXN];

void update(int i, int dx) {
  while(i <= N) {
    fen[i] += dx;
    i += LOWBIT(i);
  }
}

int sum(int i) {
  int s = 0;
  while(i > 0) {
    s += fen[i];
    i -= LOWBIT(i);
  }
  return s;
}
```

1.2 BST in pb_ds

```
/* Red-Black tree via pb_ds. */
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

int main() {
  ordered_set<int>  s;
  s.insert(1);
  s.insert(3);
  cout << s.order_of_key(2) << endl; // the number of elements in the s less than 2
  cout << *s.find_by_order(0) << endl; // print the 0-th smallest number in s(0-based)
}
```

1.3 Segment Tree

```
/* Segment tree (Interval tree, range tree), by Abreto <m@abreto.net>. */

template <int STMAX = 1000000>
struct segment_tree {
  struct node_t {
    static inline node_t merge(node_t n1, node_t n2) {
      node_t ans;
      ans.l = n1.l;
      ans.r = n2.r;
      /* merge n1 and n2 to ans. */
```

```
    return ans;
  }

  /* Data field */
  int l,r;
} nodes[(STMAX+1)<<2];

struct lazy_t {
  int marked; /* Optional */
  /* lazy mark. */

  lazy_t(void) {
    clear();
  }
  void clear(void) {
    marked=0;
  }
} marks[(STMAX+1)<<2];

inline void maintain_leaf(int o, int idx) {
  nodes[o].l = nodes[o].r = idx;
  /* Operations to single elements ... */
}
inline void maintain(int o) {
  nodes[o] = node_t::merge(nodes[o<<1], nodes[o<<1|1]);
}

/* Usage: build(1,1,n); */
void build(int o, int l, int r) { /* [l,r] */
  if( r <= l ) {
    maintain_leaf(o, l);
  } else {
    int mid = l+r>>1;
    build(o<<1, l, mid);
    build(o<<1|1, mid+1, r);
    maintain(o);
  }
}

/* Modify all elements in [l,r] */
void mark(lazy_t act, int o) {
  /* do something .. */
  marks[o].marked = 1;
}

/* Pass cached updates. */
void pushdown(int o) {
  if( marks[o].marked ) {
    mark(marks[o], o<<1);
    mark(marks[o], o<<1|1);
    marks[o].clear();
  }
}

/* Do act on all elements in [L,R] */
void upd(int L, int R, lazy_t act, int o, int l, int r) {
  if( L <= l && r <= R ) {
    mark(act, o);
  } else if (L <= R) {
    int mid = (l+r)>>1;
    pushdown(o);
    if( L <= mid ) upd(L, R, act, o<<1, l, mid);
    if( R > mid ) upd(L, R, act, o<<1|1, mid+1, r);
    maintain(o);
```

```
    }
  }

  node_t qry(int L, int R, int o, int l, int r) {
    if(L <= l && r <= R)
      return nodes[o];
    else if (L <= R) {
      int mid = (l+r)>>1;
      pushdown(o);
      if(R <= mid) return qry(L,R,o<<1,l,mid);
      if(L > mid) return qry(L,R,o<<1|1,mid+1,r);
      return node_t::merge(qry(L,R,o<<1,l,mid),qry(L,R,o<<1|1,mid+1,r));
    }
  }

  int N;

  segment_tree(void):N(STMAX) {}
  segment_tree(int n):N(n) {}
  void build(int n) {
    N = n;
    build(1,1,N);
  }
  void update(int L, int R, lazy_t act) {
    upd(L,R,act,1,1,N);
  }
  node_t query(int L, int R) {
    return qry(L,R,1,1,N);
  }
};
```

2 Dynamic Programming

2.1 LIS $O(n \log n)$

```

int top = 0;
for( int i=1; i<=n; i++ ) {
  if( ap[i] > dp[top] ) { // 如果大于 ”模拟栈” 的栈顶元素直接 入栈 长度加 1
    top++;
    dp[top] = ap[i];
    continue;
  }
  int m = ap[i];
  // lower_bound 前闭后开 返回不小于 m 的最小值的位置
  pos = lower_bound(dp,dp+top,m)-dp; // 注意减去dp
  if( dp[pos] > ap[i])
    dp[pos] = ap[i];
}
```

2.2 LCS $O(n \log n)$

```

/**
总的来说，就是把LCS转化成LIS，然后用LIS的NlogN算法来求解。
实现如下：（引用）
假设有两个序列 s1[ 1~6 ] = { a, b, c, a, d, c }, s2[ 1~7 ] = { c, a, b, e, d, a, b }
    。
记录s1中每个元素在s2中出现的位置, 再将位置按降序排列, 则上面的例子可表示为：
loc(a) = { 6, 2 }, loc( b ) = { 7, 3 }, loc( c ) = { 1 }, loc( d ) = { 5 }。 // 倒着
    扫一遍s2即可把位置扔进vector
```

```
将s1中每个元素的位置按s1中元素的顺序排列成一个序列s3 = { 6, 2, 7, 3, 1, 6, 2, 5, 1 }
    。
在对s3求LIS得到的值即为求LCS的答案。
*/
```

2.3 Sparse-Table

```
/* RMQ with Sparse Table, by Abreto <m@abreto.net>. */

int min(int a, int b) {
  return (a<b)?a:b;
}

#define MAXN    100001
#define MAXLOG  32

int N;
int A[MAXN];    /* indexed from 0. */
int st[MAXN][MAXLOG];

void st_init() {
  int i = 0, j = 0, t = 0;
  for(i = 0; i < N; ++i) st[i][0] = A[i];
  for(j = 1; (t=(1<<j)) <= N; ++j)
    for(i = 0; (i+t-1) < N; ++i)
      st[i][j] = min(st[i][j-1], st[i+(t>>1)][j-1]);
  /* st(i,j) = min(st(i,j-1), st(i+2^(j-1),j-1)). */
}

int st_query(int l, int r) {
  int k = 0;
  while((1<<(k+1)) <= (r-l+1)) k++;
  return min(st[l][k], st[r-(1<<k)+1][k]);
}
```

2.4 Improved by quadrilateral inequality

```
/*
 * 四边形不等式
 *
 * 如果 dp(i,j) 满足 dp(i,j)<=dp(i,j+1)<=dp(i+1,j+1)
 * 那么决策 s(i,j) 满足 s(i,j)<=s(i,j+1)<=s(i+1,j+1)
 * 可以变形为:
 *      s(i-1,j) <= s(i,j) <= s(i,j+1)  // i增j减
 *  或
 *      s(i,j-1) <= s(i,j) <= s(i+1,j)  // 区间长度L增
 */
#include <bits/stdc++.h>

using namespace std;

#define MAXN    1024
#define inf     (0x3fffffff)

int n, m;
int v[MAXN];
int s[MAXN];
int w[MAXN][MAXN];
int dp[MAXN][MAXN];
int c[MAXN][MAXN];

```

```
int wa(void) {
  int i, j, k;
  for(i = 1; i <= n; ++i) {
    scanf("%d", v+i);
    s[i] = v[i] + s[i-1];
  }
  for(i = 1; i <= n; ++i) {
    w[i][i] = 0;
    for(j = i+1; j <= n; ++j)
      w[i][j] = w[i][j-1] + v[j] * (s[j-1] - s[i-1]);
  }
  /* doing dp */
  for(i = 1; i <= n; ++i) {
    dp[i][0] = w[1][i];
    c[i][0] = 1;
    c[i][i] = i-1;
    for(j = i-1; j > 0; j--) {
      dp[i][j] = inf;
      for(k = c[i-1][j]; k <= c[i][j+1]; ++k)
        if(dp[k][j-1]+w[k+1][i] <= dp[i][j]) {
          dp[i][j] = dp[k][j-1] + w[k+1][i];
          c[i][j] = k;
        }
    }
  }
  /* dp done */
  return dp[n][m];
}

int main(void) {
  while(EOF != scanf("%d%d", &n, &m) && n && m) {
    printf("%d\n", wa());
  }
  return 0;
}
```

2.5 Improved by Slope

```
/* type 1: */
/* bzoj 1010 */
#include <bits/stdc++.h>

using namespace std;
typedef long double ll;
#define MAXN    50050
#define eps     (1e-8)

int N;
ll L;
ll S[MAXN];
ll f[MAXN];
ll dp[MAXN];

inline ll k(int j) {
  return (-2.0) * (f[j] + L);
}
inline ll b(int j) {
  return dp[j] + f[j]*f[j] + 2ll*f[j]*L;
}
inline ll g(int j, int i) {
  return k(j) * f[i] + b(j);
}

```

```
/* check if l1 & l3 <= l2 */
inline int check(int l1, int l2, int l3) {
  /*ll left = b(l3)*k(l1)+b(l1)*k(l2)+b(l2)*k(l3);
  ll right = b(l1)*k(l3)+b(l3)*k(l2)+b(l2)*k(l1);*/
  ll left = b(l3)*k(l1)-b(l1)*k(l3);
  ll right = k(l2)*(b(l3)-b(l1))+b(l2)*(k(l1)-k(l3));
  return (left <= right);
}

int Q[MAXN], ql, qr;

int main(void) {
  int i;
  scanf(”%d%Lf”, &N, &L);
  L += 1.0;
  for(i = 1; i <= N; ++i) {
    scanf(”%Lf”, S+i);
    S[i] += S[i-1];
    f[i] = S[i] + (double)i;
  }
  Q[qr++] = 0;
  for(i = 1; i <= N; ++i) {
    /* <!-- STARED */
    for(; ql+1 < qr && g(Q[ql],i) >= g(Q[ql+1],i); ql++);
    dp[i] = g(Q[ql], i) + f[i]*f[i] + L*L; //printf(”%d: %lld,%lld\n”, i, dp[i], dp[i]-f[i]*f[i]);
    for(; ql+1 < qr && check(Q[qr-2], Q[qr-1], i); qr--);
    Q[qr++] = i;
    /* --> */
  }
  printf(”%lld\n”, (long long int)round(dp[N]));
  return 0;
}
```

3 Geometry

3.1 2D

3.1.1 Point

```
/* 2D Point Class, by Abreto<m@abreto.net> */
#include <cmath>

using namespace std;

#define EPS (1e-8)
bool fe(double a, double b) {
  return ((a-b>=-EPS)&&(a-b<=EPS));
}
bool fl(double a, double b) {
  return (a-b<-EPS);
}
bool fle(double a, double b) {
  return (a-b<=EPS);
}

template <typename T>
struct point {
  T x,y;
  point(void):x(T()),y(T()) {}
  point(T xx, T yy):x(xx),y(yy) {}
  T& operator[](int i) {
```

```
    if(0==i)return x;
    else return y;
  }
  inline point operator-(void) const {
    return point(-x,-y);
  }
  inline point operator+(const point& b) const {
    return point(x+b.x,y+b.y);
  }
  inline point operator-(const point& b) const {
    return point(x-b.x,y-b.y);
  }
  inline T operator*(const point& b) const {
    return ((x)*(b.x))+((y)*(b.y)); /* inner product */
  }
  inline T operator^(const point& b) const {
    return ((x)*(b.y))-((b.x)*(y)); /* outter product */
  }
  inline point& operator+=(const point& b) {
    point tmp=(*this)+b;
    (*this)=tmp;
    return (*this);
  }
  inline point& operator-=(const point& b) {
    point tmp=(*this)-b;
    (*this)=tmp;
    return (*this);
  }
  inline bool operator==(const point& b) const {
    return (x==b.x)&&(y==b.y);
  }
  inline bool operator!=(const point& b) const {
    return !((*this)==b);
  }
};

#define vec point
```

3.1.2 Convex hull

```
/* 2D Convex Hull, by Abreto <m@abreto.net>. */
#include "2d_base.hh"
#include <cmath>
#include <algorithm>

using namespace std;

point O;

bool comp_angle(point_t a, point_t b) {
  double t = (a-O).X(b-O);
  if(fe(t,0.0)) return fl((b-O).mag2(),(a-O).mag2());
  else return fl(0.0,t);
}

void convex_hull_graham(vp& convex, vp src) {
  int i = 0, top = 0;
  O = src[0];
  for(auto pt : src)
    if( pt.x < O.x || (pt.x == O.x && pt.y < O.y))
      O = pt;
  sort(src.begin(), src.end(), comp_angle);
  convex.push_back(src[0]);
```

```
  convex.push_back(src[1]);
  top = 1;
  for(i = 2; i < src.size(); ++i) {
    while(top>1 && fle((convex[top]-convex[top-1]).X(src[i]-convex[top]),0.0)) {
      convex.pop_back();
      --top;
    }
    convex.push_back(src[i]);
    ++top;
  }
}
```

3.1.3 Intersect Area

```
#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

//#define inf 1000000000000
#define M 8
#define LL long long
#define eps 1e-12
#define PI acos(-1.0)
using namespace std;
struct node {
  double x,y;
  node() {}
  node(double xx,double yy) {
    x=xx;
    y=yy;
  }
  node operator -(node s) {
    return node(x-s.x,y-s.y);
  }
  node operator +(node s) {
    return node(x+s.x,y+s.y);
  }
  double operator *(node s) {
    return x*s.x+y*s.y;
  }
  double operator ^(node s) {
    return x*s.y-y*s.x;
  }
};
double max(double a,double b) {
  return a>b?a:b;
}
double min(double a,double b) {
  return a<b?a:b;
}
double len(node a) {
  return sqrt(a*a);
}
double dis(node a,node b) { //两点之间的距离
  return len(b-a);
}
double cross(node a,node b,node c) { //叉乘
  return (b-a)^(c-a);
}
double dot(node a,node b,node c) { //点成
  return (b-a)*(c-a);
```

```
}
int judge(node a,node b,node c) { //判断c是否在ab线段上（前提是c在直线ab上）
  if(c.x>=min(a.x,b.x)
      &&c.x<=max(a.x,b.x)
      &&c.y>=min(a.y,b.y)
      &&c.y<=max(a.y,b.y))
    return 1;
  return 0;
}
double area(node b,node c,double r) {
  node a(0.0,0.0);
  if(dis(b,c)<eps)
    return 0.0;
  double h=fabs(cross(a,b,c))/dis(b,c);
  if(dis(a,b)>r-eps&&dis(a,c)>r-eps) { //两个端点都在圆的外面则分为两种情况
    double angle=acos(dot(a,b,c)/dis(a,b)/dis(a,c));
    if(h>r-eps) {
      return 0.5*r*r*angle;
    } else if(dot(b,a,c)>0&&dot(c,a,b)>0) {
      double angle1=2*acos(h/r);
      return 0.5*r*r*fabs(angle-angle1)+0.5*r*r*sin(angle1);
    } else {
      return 0.5*r*r*angle;
    }
  } else if(dis(a,b)<r+eps&&dis(a,c)<r+eps) { //两个端点都在圆内的情况
    return 0.5*fabs(cross(a,b,c));
  } else { //一个端点在圆上一个端点在圆内的情况
    if(dis(a,b)>dis(a,c)) { //默认b在圆内
      swap(b,c);
    }
    if(fabs(dis(a,b))<eps) { //ab距离为0直接返回0
      return 0.0;
    }
    if(dot(b,a,c)<eps) {
      double angle1=acos(h/dis(a,b));
      double angle2=acos(h/r)-angle1;
      double angle3=acos(h/dis(a,c))-acos(h/r);
      return 0.5*dis(a,b)*r*sin(angle2)+0.5*r*r*angle3;

    } else {
      double angle1=acos(h/dis(a,b));
      double angle2=acos(h/r);
      double angle3=acos(h/dis(a,c))-angle2;
      return 0.5*r*dis(a,b)*sin(angle1+angle2)+0.5*r*r*angle3;
    }
  }
}

node A, B, C;
int R;

bool compar(node &p1, node &p2) {
  return (p1^p2)>eps;
}

double f(double x, double y) {
  node O(x,y);
  node p[8];
  p[0] = A-O;
  p[1] = B-O;
  p[2] = C-O;
  sort(p, p+3, compar);
  p[3] = p[0];
  O=node(0,0);
```

```
  double sum=0;
  /* <!-- 求面积交部分 */
  for(int i=0; i<3; i++) { /* 按顺或逆时针顺序最后取绝对值就好 */
    int j=i+1;
    double s=area(p[i],p[j],(double)R);
    if(cross(O,p[i],p[j])>0)
      sum+=s;
    else
      sum-=s;
  }
  if(sum < -eps) sum = -sum;
  /* --> */
  return sum;
}

double trifind(double x, double y1, double y2) {
  double l = y1, r = y2;
  while(r-l>eps) {
    double mid = (l+r)/2.0;
    double mmid = (mid+r)/2.0;
    if( f(x,mmid) > f(x,mid)+eps )
      l = mid;
    else
      r = mmid;
  }
  return f(x,l);
}

double findmin(double x1, double x2, double y1, double y2) {
  double l = x1, r = x2;
  while(r-l>eps) {
    double mid = (l+r)/2.0;
    double mmid = (mid+r)/2.0;
    if( trifind(mmid,y1,y2) > trifind(mid,y1,y2)+eps )
      l = mid;
    else
      r = mmid;
  }
  return trifind(l,y1,y2);
}

double ans(int a, int b, int c, int r) {
  A = node(0,0);
  B = node((double)c,0);
  R = r;
  double da = a, db = b, dc = c;
  double cosa = (db*db+dc*dc-da*da)/(2.0*db*dc);
  double alpha = acos(cosa);
  C = node(db*cosa, db*sin(alpha));
  return findmin(0.0, c, 0.0, db*sin(alpha));
}

int main(void) {
  int a = 0, b = 0, c = 0, r = 0;
  while(EOF != scanf("%d%d%d%d",&a,&b,&c,&r) && (a||b||c||r))
    printf("%.8lf\n", ans(a,b,c,r));
  return 0;
}
```

3.1.4 Universe

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct Point {
  double x, y;
  Point(double x = 0, double y = 0) : x(x), y(y) {}
};

typedef Point Vector;

Vector operator + (Vector A, Vector B) {
  return Vector(A.x + B.x, A.y + B.y);
}
Vector operator - (Vector A, Vector B) {
  return Vector(A.x - B.x, A.y - B.y);
}
Vector operator * (Vector A, double p) {
  return Vector(A.x*p, A.x*p);
}
Vector operator / (Vector A, double p) {
  return Vector(A.x/p, A.x/p);
}

bool operator < (const Point& a, const Point b) {
  return a.x < b.x || (a.x == b.x && a.y < b.y);
}

const double EPS = 1e-10;

int dcmp(double x) {
  if(fabs(x) < EPS) return 0;
  else return x < 0 ? -1 : 1;
}

bool operator == (const Point& a, const Point& b) {
  return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
}

//向量a的极角
double Angle(const Vector& v) {
  return atan2(v.y, v.x);//\share\CodeBlocks\templates\wizard\console\cpp
}

//向量点积
double Dot(Vector A, Vector B) {
  return A.x*B.x + A.y*B.y;
}

//向量长度\share\CodeBlocks\templates\wizard\console\cpp
double Length(Vector A) {
  return sqrt(Dot(A, A));
}

//向量夹角
double Angle(Vector A, Vector B) {
  return acos(Dot(A, B) / Length(A) / Length(B));
}

//向量叉积
double Cross(Vector A, Vector B) {
  return A.x*B.y - A.y*B.x;
}

//三角形有向面积的二倍
double Area2(Point A, Point B, Point C) {
  return Cross(B-A, C-A);
```

```
}

//向量逆时针旋转rad度(弧度)
Vector Rotate(Vector A, double rad) {
  return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

//计算向量A的单位法向量。左转90°，把长度归一。调用前确保A不是零向量。
Vector Normal(Vector A) {
  double L = Length(A);
  return Vector(-A.y/L, A.x/L);
}

/*************************************************************************
使用复数类实现点及向量的简单操作

#include <complex>
typedef complex<double> Point;
typedef Point Vector;

double Dot(Vector A, Vector B) { return real(conj(A)*B)}
double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }

*************************************************************************/

/*****************************************************************************
* 用直线上的一点p0和方向向量v表示一条指向。直线上的所有点P满足P = P0+t*v;
* 如果知道直线上的两个点则方向向量为B-A, 所以参数方程为A+(B-A)*t;
* 当t 无限制时， 该参数方程表示直线。
* 当t > 0时， 该参数方程表示射线。
* 当 0 < t < 1时， 该参数方程表示线段。
*****************************************************************************/

//直线交点,须确保两直线有唯一交点。
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
  Vector u = P - Q;
  double t = Cross(w, u)/Cross(v, w);
  return P+v*t;
}

//点到直线距离
double DistanceToLine(Point P, Point A, Point B) {
  Vector v1 = B - A, v2 = P - A;
  return fabs(Cross(v1, v2) / Length(v1)); //不取绝对值，得到的是有向距离
}

//点到线段的距离
double DistanceToSegmentS(Point P, Point A, Point B) {
  if(A == B) return Length(P-A);
  Vector v1 = B-A, v2 = P-A, v3 = P-B;
  if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
  else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
  else return fabs(Cross(v1, v2)) / Length(v1);
}

//点在直线上的投影
Point GetLineProjection(Point P, Point A, Point B) {
  Vector v = B - A;
  return A+v*(Dot(v, P-A)/Dot(v, v));
}

//线段相交判定，交点不在一条线段的端点
bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2) {
```

```
  double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
  double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
  return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
}

//判断点是否在点段上，不包含端点
bool OnSegment(Point P, Point a1, Point a2) {
  return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P, a2-P)) < 0));
}

//计算凸多边形面积
double ConvexPolygonArea(Point *p, int n) {
  double area = 0;
  for(int i = 1; i < n-1; i++)
    area += Cross(p[i] - p[0], p[i+1] - p[0]);
  return area/2;
}

//计算多边形的有向面积
double PolygonArea(Point *p, int n) {
  double area = 0;
  for(int i = 1; i < n-1; i++)
    area += Cross(p[i] - p[0], p[i+1] - p[0]);
  return area/2;
}

/**********************************************************************
* Morley定理：三角形每个内角的三等分线，相交成的三角形是等边三角形。
* 欧拉定理：设平面图的定点数，边数和面数分别为V,E,F。则V+F-E = 2;
**********************************************************************/

struct Circle {
  Point c;
  double r;

  Circle(Point c, double r) : c(c), r(r) {}
  //通过圆心角确定圆上坐标
  Point point(double a) {
    return Point(c.x + cos(a)*r, c.y + sin(a)*r);
  }
};

struct Line {
  Point p;
  Vector v;
  double ang;
  Line() {}
  Line(Point p, Vector v) : p(p), v(v) {}
  bool operator < (const Line& L) const {
    return ang < L.ang;
  }
};

//直线和圆的交点，返回交点个数，结果存在sol中。
//该代码没有清空sol。
int getLineCircleIntersecion(Line L, Circle C, double& t1, double& t2, vector<Point>&
    sol) {
  double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
  double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
  double delta = f*f - 4*e*g;
  if(dcmp(delta) < 0) return 0; //相离
  if(dcmp(delta) == 0) {        //相切
    t1 = t2 = -f / (2*e);
    sol.push_back(C.point(t1));
```

```
    return 1;
  }
  //相交
  t1 = (-f - sqrt(delta)) / (2*e);
  sol.push_back(C.point(t1));
  t2 = (-f + sqrt(delta)) / (2*e);
  sol.push_back(C.point(t2));
  return 2;
}

//两圆相交
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
  double d = Length(C1.c - C2.c);
  if(dcmp(d) == 0) {
    if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完全重合
    return 0;                                //同心圆，半径不一样
  }
  if(dcmp(C1.r + C2.r - d) < 0) return 0;
  if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;

  double a = Angle(C2.c - C1.c);              //向量C1C2的极角
  double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
  //C1C2到C1P1的角
  Point p1 = C1.point(a-da), p2 = C1.point(a+da);
  sol.push_back(p1);
  if(p1 == p2) return 1;
  sol.push_back(p2);
  return 2;
}

const double PI = acos(-1);
//过定点做圆的切线
//过点p做圆C的切线，返回切线个数。v[i]表示第i条切线
int getTangents(Point p, Circle C, Vector* v) {
  Vector u = C.c - p;
  double dist = Length(u);
  if(dist < C.r) return 0;
  else if(dcmp(dist - C.r) == 0) {
    v[0] = Rotate(u, PI/2);
    return 1;
  } else {
    double ang = asin(C.r / dist);
    v[0] = Rotate(u, -ang);
    v[1] = Rotate(u, +ang);
    return 2;
  }
}

//两圆的公切线
//返回切线的个数，-1表示有无数条公切线。
//a[i], b[i] 表示第i条切线在圆A，圆B上的切点
int getTangents(Circle A, Circle B, Point *a, Point *b) {
  int cnt = 0;
  if(A.r < B.r) {
    swap(A, B);
    swap(a, b);
  }
  int d2 = (A.c.x - B.c.x)*(A.c.x - B.c.x) + (A.c.y - B.c.y)*(A.c.y - B.c.y);
  int rdiff = A.r - B.r;
  int rsum = A.r + B.r;
  if(d2 < rdiff*rdiff) return 0;   //内含
  double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
  if(d2 == 0 && A.r == B.r) return -1;   //无限多条切线
  if(d2 == rdiff*rdiff) {          //内切一条切线
```

```
    a[cnt] = A.point(base);
    b[cnt] = B.point(base);
    cnt++;
    return 1;
  }
  //有外共切线
  double ang = acos((A.r-B.r) / sqrt(d2));
  a[cnt] = A.point(base+ang);
  b[cnt] = B.point(base+ang);
  cnt++;
  a[cnt] = A.point(base-ang);
  b[cnt] = B.point(base-ang);
  cnt++;
  if(d2 == rsum*rsum) {  //一条公切线
    a[cnt] = A.point(base);
    b[cnt] = B.point(PI+base);
    cnt++;
  } else if(d2 > rsum*rsum) {   //两条公切线
    double ang = acos((A.r + B.r) / sqrt(d2));
    a[cnt] = A.point(base+ang);
    b[cnt] = B.point(PI+base+ang);
    cnt++;
    a[cnt] = A.point(base-ang);
    b[cnt] = B.point(PI+base-ang);
    cnt++;
  }
  return cnt;
}

typedef vector<Point> Polygon;

//点在多边形内的判定
int isPointInPolygon(Point p, Polygon poly) {
  int wn = 0;
  int n = poly.size();
  for(int i = 0; i < n; i++) {
    if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1; //在边界上
    int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));
    int d1 = dcmp(poly[i].y - p.y);
    int d2 = dcmp(poly[(i+1)%n].y - p.y);
    if(k > 0 && d1 <= 0 && d2 > 0) wn++;
    if(k < 0 && d2 <= 0 && d1 > 0) wn++;
  }
  if(wn != 0) return 1;       //内部
  return 0;                   //外部
}

//凸包
/***************************************************************
* 输入点数组p， 个数为p， 输出点数组ch。 返回凸包顶点数
* 不希望凸包的边上有输入点，把两个<= 改成 <
* 高精度要求时建议用dcmp比较
* 输入点不能有重复点。函数执行完以后输入点的顺序被破坏
****************************************************************/
int ConvexHull(Point *p, int n, Point* ch) {
  sort(p, p+n);      //先比较x坐标，再比较y坐标
  int m = 0;
  for(int i = 0; i < n; i++) {
    while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];
  }
  int k = m;
  for(int i = n-2; i >= 0; i++) {
    while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
```

```
    ch[m++] = p[i];
  }
  if(n > 1) m--;
  return m;
}

//用有向直线A->B切割多边形poly， 返回“左侧”。 如果退化，可能会返回一个单点或者线段
//复杂度O(n2);
Polygon CutPolygon(Polygon poly, Point A, Point B) {
  Polygon newpoly;
  int n = poly.size();
  for(int i = 0; i < n; i++) {
    Point C = poly[i];
    Point D = poly[(i+1)%n];
    if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
    if(dcmp(Cross(B-A, C-D)) != 0) {
      Point ip = GetLineIntersection(A, B-A, C, D-C);
      if(OnSegment(ip, C, D)) newpoly.push_back(ip);
    }
  }
  return newpoly;
}

//半平面交

//点p再有向直线L的左边。（线上不算）
bool Onleft(Line L, Point p) {
  return Cross(L.v, p-L.p) > 0;
}

//两直线交点，假定交点唯一存在
Point GetIntersection(Line a, Line b) {
  Vector u = a.p - b.p;
  double t = Cross(b.v, u) / Cross(a.v, b.v);
  return a.p+a.v*t;
}

int HalfplaneIntersection(Line* L, int n, Point* poly) {
  sort(L, L+n);                //按极角排序

  int first, last;             //双端队列的第一个元素和最后一个元素
  Point *p = new Point[n];     //p[i]为q[i]和q[i+1]的交点
  Line *q = new Line[n];       //双端队列
  q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
  for(int i = 0; i < n; i++) {
    while(first < last && !Onleft(L[i], p[last-1])) last--;
    while(first < last && !Onleft(L[i], p[first])) first++;
    q[++last] = L[i];
    if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
      last--;
      if(Onleft(q[last], L[i].p)) q[last] = L[i];
    }
    if(first < last) p[last-1] = GetIntersection(q[last-1], q[last]);
  }
  while(first < last && !Onleft(q[first], p[last-1])) last--;
  //删除无用平面
  if(last-first <= 1) return 0;   //空集
  p[last] = GetIntersection(q[last], q[first]);

  //从deque复制到输出中
  int m = 0;
  for(int i = first; i <= last; i++) poly[m++] = p[i];
  return m;
}
```

4 Graph

4.1 Tree

4.1.1 Universe

```

/* find root(重心) */

void findroot(int u, int fa) {
  int i;
  size[u] = 1;
  f[u] = 0;
  for (i = last[u]; i; i = e[i][2]) {
    if (!vis[e[i][0]] && e[i][0] != fa) {
      findroot(e[i][0], u);
      size[u] += size[e[i][0]];
      if (f[u] < size[e[i][0]])
        f[u] = size[e[i][0]];
    }
  }
  if (f[u] < ALL - size[u])
    f[u] = ALL - size[u];
  if (f[u] < f[root]) root = u;
}

/* --- da --- */

int dep[MAXN+1];
int ancestor[MAXN+1][MAXLGN];
int minw[MAXN+1][MAXLGN];

void dfs(int u, int fa) {
  ancestor[u][0] = fa;
  dep[u] = dep[fa] + 1;
  for(int e = u[front]; e; e = E[e].n) {
    int v = E[e].v, w = E[e].w;
    if(v != fa) {
      minw[v][0] = w;
      dfs(v, u);
    }
  }
}

void init_system(void) {
  int i = 0, w = 0;
  int t = 0;
  dep[0] = -1;
  dfs(1,0);
  for(w = 1; (t=(1<<w)) < N; ++w)
    for(i = 1; i <= N; ++i) if( dep[i] >= t ) {
        ancestor[i][w] = ancestor[ancestor[i][w-1]][w-1];
        minw[i][w] = min(minw[i][w-1], minw[ancestor[i][w-1]][w-1]);
      }
}

int query(int a, int b) {
  if(dep[a] < dep[b]) return query(b,a);
  else {  /* now dep[s] > dep[t] */
    int i = 0;
    int maxbit = MAXLGN-1;
    int ret = INF;
    //while((1<<maxbit) <= dep[a]) maxbit++;
    /* first up a to same dep with b. */
```

```
    for(i = maxbit; i >= 0; i--)
      if(dep[a] - (1<<i) >= dep[b]) {
        ret = min(ret, minw[a][i]);
        a = ancestor[a][i];
      }
    if(a == b) return ret;
    for(i = maxbit; i >= 0; i--)
      if(dep[a] - (1<<i) >= 0 && ancestor[a][i] != ancestor[b][i]) {
        ret = min(ret, min(minw[a][i], minw[b][i]));
        a = ancestor[a][i];
        b = ancestor[b][i];
      }
    ret = min(ret, min(minw[a][0], minw[b][0]));
    return ret;
  }
}
```

4.1.2 Point Divide and Conquer

Version 1

```
/* Tree::Point divide and conquer, by Abreto<m@abreto.net>. */
#include <bits/stdc++.h>

using namespace std;
typedef long long int ll;

#define MAXN    (100001)
#define MAXV    (MAXN+1)
#define MAXE    (MAXN<<1)
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv,edge *nn):v(vv),n(nn) {}
};
int nE;
edge E[MAXE];
edge *front[MAXV];
int label[MAXV];    /* 0 for '(', 1 for ')' */
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}

int n;
ll ans;

char del[MAXV];
namespace findroot {
int ALL;
int nfind;
int vis[MAXV];
int size[MAXV];
int f[MAXV];
int root;
void __find(int u, int fa) {
  vis[u] = nfind;
  size[u] = 1;
  f[u] = 0;
  for(edge *e=u[front]; e; e = e->n) {
    int v = e->v;
```

```
    if((!del[v]) && (vis[v] != nfind) && (v != fa)) {
      __find(v, u);
      size[u] += size[v];
      if(f[u] < size[v])  f[u] = size[v];
    }
  }
  if(f[u] < ALL - size[u])  f[u] = ALL - size[u];
  if(f[u] < f[root])  root = u;
}
int find(int u, int all) {
  ++nfind;
  ALL = all;
  f[root = 0] = MAXV;
  __find(u,0);
  return root;
}
}

namespace workspaces {
int maxdep;
int dep[MAXV];
ll cntin[MAXV], cntout[MAXV];
int in[2][MAXV];    /* 0 for '(', 1 for ')' */
int out[2][MAXV];
void getdeep(int u, int fa) {
  dep[u] = dep[fa] + 1;
  if(dep[u] > maxdep) maxdep = dep[u];
  for(edge *e = u[front]; e; e = e->n)
    if((!del[e->v]) && (fa != e->v))
      getdeep(e->v, u);
}
void dfs(int u, int fa) {
  {
    /* out from root */
    out[0][u] = out[0][fa];
    out[1][u] = out[1][fa];
    if(0 == label[u]) { /* meet '(' */
      out[0][u]++;
    } else {    /* meet ')' */
      if(out[0][u]) out[0][u]--;
      else out[1][u]++;
    }
    if(out[0][u] == 0)
      cntout[out[1][u]]++;
  }
  {
    /* in to root */
    in[0][u] = in[0][fa];
    in[1][u] = in[1][fa];
    if(0 == label[u]) { /* meet '(' */
      if(in[1][u]) in[1][u]--;
      else in[0][u]++;
    } else {    /* meet ')' */
      in[1][u]++;
    }
    if(0 == in[1][u])
      cntin[in[0][u]]++;
  }
  /* do something */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if((!del[v]) && (v != fa)) {
      dfs(v, u);
    }
```

```
  }
}
inline void init_maxdep(void) {
  maxdep = 0;
}
inline void update_maxdep(int u) {
  dep[u] = 1;
  if(dep[u] > maxdep) maxdep = dep[u];
  for(edge *e = u[front]; e; e = e->n)
    if((!del[e->v]))
      getdeep(e->v, u);
}
inline void clear(void) {
  for(int i = 0; i <= maxdep+1; ++i)
    cntin[i] = cntout[i] = 0;
}
inline void work(int u) {
  in[0][u] = in[1][u] = out[0][u] = out[1][u] = 0;
  in[label[u]][u] = out[label[u]][u] = 1;
  if(out[0][u] == 0) cntout[out[1][u]]++;
  if(0 == in[1][u]) cntin[in[0][u]]++;
  /* update in and out if neccessary */
  for(edge *e = u[front]; e; e = e->n)
    if(!(del[e->v]))
      dfs(e->v, u);
}
};

ll count(int u, int p) {
  ll ret = 0;
  workspace::init_maxdep();
  workspace::update_maxdep(u);
  workspace::clear();
  if(-1 == p) {
    for(edge *e = u[front]; e; e = e->n)
      if((!(del[e->v])))
        workspace::work(e->v);
    p = label[u];
    /* single end */
    if(0 == p) ret = workspace::cntout[1];
    else ret = workspace::cntin[1];
  } else {
    workspace::work(u);
  }
  if(0 == p) { /* p is '(' */
    for(int i = 0; i < workspace::maxdep; ++i)  /* concatenation */
      ret += workspace::cntin[i] * workspace::cntout[i+1];
  } else {    /* p is ')' */
    for(int i = 0; i < workspace::maxdep; ++i)  /* concatenation */
      ret += workspace::cntin[i+1] * workspace::cntout[i];
  }
  return ret;
}

void handle(int u) {
  del[u] = 1; /* delete current root. */
  ans += count(u, -1);
  /* do something */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if(!del[v]) {
      ans -= count(v, label[u]);
      /* do something */
      int r = findroot::find(v, findroot::size[v]);
```

```
      handle(r);
    }
  }
}

void proc(void) {
  int r = findroot::find(1,n);
  handle(r);
}

char ls[MAXV+1];
int main(void) {
  int i = 0;
  scanf("%d", &n);
  scanf("%s", ls);
  for(i = 0; i < n; ++i)
    label[i+1] = ls[i] - '(';
  for(i = 1; i < n; ++i) {
    int ai, bi;
    scanf("%d␣%d", &ai, &bi);
    add_edge(ai, bi);
    add_edge(bi, ai);
  }
  proc();
  printf("%lld\n", ans);
  return 0;
}
```

Version 2

```
/* 2016 ACM/ICPC Asia Regional Dalian. Problem , by Abreto<m@abreto.net>. */
#include <bits/stdc++.h>

using namespace std;
typedef long long int ll;

/* offset in [1,k] */
#define GET(i,offset)   (((i)>>((offset)-1))&1)
#define SET(i,offset)   ((i)|(1<<((offset)-1)))
#define REV(i,offset)   ((i)^(1<<((offset)-1)))

#define MAXN    (50005)
#define MAXV    (MAXN+1)
#define MAXE    (MAXN<<1)
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv,edge *nn):v(vv),n(nn) {}
};
int nE;
edge E[MAXE];
edge *front[MAXV];
int label[MAXV];    /* each kind */
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}

int n, k;
ll ans;
int all_kind;

int ndel;
```

```
int del[MAXV];
namespace findroot {
int ALL;
ll nfind;
ll vis[MAXV];
int size[MAXV];
int f[MAXV];
int root;
void __find(int u, int fa) {
  vis[u] = nfind;
  size[u] = 1;
  f[u] = 0;
  for(edge *e=u[front]; e; e = e->n) {
    int v = e->v;
    if((del[v] != ndel) && (vis[v] != nfind) && (v != fa)) {
      __find(v, u);
      size[u] += size[v];
      if(f[u] < size[v])  f[u] = size[v];
    }
  }
  if(f[u] < ALL - size[u])  f[u] = ALL - size[u];
  if(f[u] < f[root])  root = u;
}
int find(int u, int all) {
  ++nfind;
  ALL = all;
  f[root = 0] = MAXV;
  __find(u,0);
  return root;
}
}

namespace workspace {
ll cnt[1024];
int dp[MAXV];
void dfs(int u, int fa) {
  dp[u] = dp[fa] | label[u];
  cnt[dp[u]] ++;
  /* dig into children */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if((del[v] != ndel) && (v != fa)) {
      dfs(v, u);
    }
  }
}
inline void clear(void) {
  for(int i = 1; i <= all_kind; ++i)
    cnt[i] = 0;
}
inline void work(int u) {
  dp[u] = label[u];
  cnt[dp[u]] ++;
  for(edge *e = u[front]; e; e = e->n)
    if((del[e->v] != ndel))
      dfs(e->v, u);
}
inline void show(void) {
  for(int i = 0; i <= all_kind; ++i)
    printf("cnt[%d] = %lld\n", i, cnt[i]);
  for(int i = 1; i <= n; ++i)
    printf("dp[%d] = %d\n", i, dp[i]);
}
};
```

```


ll count(int u, int p) {
  ll ret = 0;
  workspace::clear();
  //printf("%d,%d  :\n", u, p);
  if(-1 == p) {
    for(edge *e = u[front]; e; e = e->n)
      if(((del[e->v]) != ndel))
        workspace::work(e->v);
    p = label[u];
    /* single end */
    for(int i = 1; i <= all_kind; i++)
      if(all_kind == (i|p))
        ret += (workspace::cnt[i]<<1);
  } else {
    workspace::work(u);
  }
  //workspace::show();
  for(int i = 1; i <= all_kind; ++i)
    if( workspace::cnt[i] > 0 )
      for(int j = 1; j <= all_kind; ++j)
        if(all_kind == (i|p|j))
          ret += workspace::cnt[i] * workspace::cnt[j];
  //printf("%lld\n", ret);
  return ret;
}

void handle(int u) {
  //printf("proccessing %d\n", u);
  del[u] = ndel; /* delete current root. */
  ans += count(u, -1);
  /* do something */
  for(edge *e = u[front]; e; e = e->n) {
    int v = e->v;
    if(del[v] != ndel) {
      ans -= count(v, label[u]);
      /* do something */
      int r = findroot::find(v, findroot::size[v]);
      handle(r);
    }
  }
}

void proc(void) {
  int r = findroot::find(1,n);
  handle(r);
}

void clear(void) {
  int i;
  ans = 0;
  nE = 0;
  for(i = 0; i <= n; ++i) {
    front[i] = NULL;
  }
  //findroot::nfind = 0;
  ndel++;
}

void mozhu(void) {
  int i = 0;
  int li;
  for(i = 1; i <= n; ++i) {
```

```
    scanf("%d", &li);
    label[i] = 1<<(li-1);
  }
  for(i = 1; i < n; ++i) {
    int ai, bi;
    scanf("%d %d", &ai, &bi);
    add_edge(ai, bi);
    add_edge(bi, ai);
  }
  all_kind = (1<<k)-1;
  proc();
  if(1 == k) ans += n;
  printf("%lld\n", ans);
}

int main(void) {
  while( EOF != scanf("%d%d", &n, &k) ) {
    clear();
    mozhu();
  }
  return 0;
}
```

4.2 2-SAT

```
#include <bits/stdc++.h>

using namespace std;

namespace two_sat {
const int maxn = 100000;
const int maxm = 1000000;
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv, edge *nn):v(vv),n(nn) {}
};
typedef edge *ep;
int n;
int nE;
edge E[maxm];
ep front[maxn];
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}
/* (x = xval or y = yval), indexed from 0 */
void add_clause(int x, int xv, int y, int yv) {
  x = x*2 + xv;
  y = y*2 + yv;
  add_edge(x^1, y);
  add_edge(y^1, x);
}

char mark[maxn<<1];
int S[maxn<<1], c;
void init(int N) {
  n = N;
  for(int i = 0; i < n*2; ++i) {
    i[front] = NULL;
    i[mark] = 0;
```

```
  }
  nE = 0;
}

int dfs(int x) {
  if(mark[x^1]) return 0;
  if(mark[x]) return 1;
  mark[x] = 1;
  S[c++] = x;
  for(ep e = x[front]; e; e = e->n)
    if(!dfs(e->v)) return 0;
  return 1;
}

int solve(void) {
  for(int i = 0; i < n*2; i += 2)
    if(!mark[i] && !mark[i+1]) {
      c = 0;
      if(!dfs(i)) {
        while(c > 0) mark[S[--c]] = 0;
        if(!dfs(i+1)) return 0;
      }
    }
  return 1;
}
}
```

4.3 Cut Edge and Point

```
/***
Finding cut edges
The code below works properly because the lemma above (first lemma):
h[root] = 0
par[v] = -1
dfs (v):
        d[v] = h[v]
        color[v] = gray
        for u in adj[v]:
                if color[u] == white
                        then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
                        if d[u] > h[v]
                                then the edge v-u is a cut edge
                else if u != par[v])
                        then d[v] = min(d[v], h[u])
        color[v] = black
In this code, h[v] =  height of vertex v in the DFS tree and d[v] = min(h[w] where
    there is at least vertex u in subtree of v in the DFS tree where there is an edge
    between u and w).

Finding cut vertices
The code below works properly because the lemma above (first lemma):
h[root] = 0
par[v] = -1
dfs (v):
        d[v] = h[v]
        color[v] = gray
        for u in adj[v]:
                if color[u] == white
                        then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
                        if d[u] >= h[v] and (v != root or number_of_children(v) > 1)
                                then the edge v is a cut vertex
                else if u != par[v])
                        then d[v] = min(d[v], h[u])
```

```
        color[v] = black
In this code, h[v] =  height of vertex v in the DFS tree and d[v] = min(h[w] where
    there is at least vertex u in subtree of v in the DFS tree where there is an edge
    between u and w).
***/
```

4.4 Euler Path

```
/* Euler path, by Abreto<m@abreto.net>. */
#define MAXV    (1024)
#define MAXE    (MAXV*MAXV)

typedef struct {
  int id;
  int nxt;
  int del;
} egde_t;
int front[MAXV];
egde_t edg[MAXE];
int d[MAXV];
int ind[MAXV], outd[MAXV];
int nedges;
void add_edge(int u, int v) {
  int newedge = ++nedges;
  edg[newedge].id = v;
  edg[newedge].nxt = u[front];
  edg[newedge].del = 0;
  u[front] = newedge;
  outd[u]++;
  ind[v]++;
  d[u]++;
  d[v]++;
}
void del_edge(int u, int v) {
  int e = 0;
  for(e=u[front]; e; e=edg[e].nxt)
    if(edg[e].id==v) {
      edg[e].del = 1;
      outd[u]--;
      ind[v]--;
      d[u]--;
      d[v]--;
      return;
    }
}

int path[MAXV];
int l;

void add2path(int u) {
  path[l++] = u;
}

/* Directed graph */
void euler(int x) {
  if(outd[x]) {
    int e = 0;
    for(e=x[front]; e; e=edg[e].nxt)
      if(!edg[e].del) {
        int v = edg[e].id;
        del_edge(x,v);
        euler(v);
      }
```

```
  }
  add2path(x);
}

/* Undirected graph */
void euler(int x) {
  if(d[x]) {
    int e = 0;
    for(e=x[front]; e; e=edg[e].nxt)
      if(!edg[e].del) {
        int v = edg[e].id;
        del_edge(x,v);
        del_edge(v,x);
        euler(v);
      }
  }
  add2path(x);
}
```

4.5 Shortest Path

4.5.1 Dijkstra

```
/* Shortest Path Dijstra, by Abreto<m@abreto.net>. */
#include <cstdio>
#include <set>
#include <utility>

using namespace std;
typedef set< pair<int,int> > spii;

#define MAXN    512
#define MAXV    (MAXN*MAXN)

struct egde_t {
  int id;
  int nxt;
};
int front[MAXV];
egde_t edg[MAXV<<3];
int nedges;
void add_edge(int u, int v) {
  int newedge = ++nedges;
  edg[newedge].id = v;
  edg[newedge].nxt = u[front];
  u[front] = newedge;
}

int d[MAXV];
int vis[MAXN];
int solid[MAXV];

int dijstra(int s, int t) {
  int v = s[front];
  spii q;
  q.insert(make_pair(0, s));
  while(!q.empty()) {
    auto it = q.begin();
    int u = it->second;
    int v = u[front];
    q.erase(it);
    solid[u] = 1;
    if(u == t) break;
```

```
    while(v) {
      int w = edg[v].id;
      if(!solid[w]) {
        if( (0==d[w]) || (d[u] + 1 < d[w]) ) {
          q.erase(make_pair(d[w],w));
          d[w] = d[u] + 1;
          q.insert(make_pair(d[w],w));
        }
      }
      v = edg[v].nxt;
    }
  }
  return d[t];
}
```

4.5.2 Shortest Path Fast Algorithm

```
/* Shortest Path Fast Algorithm, by Abreto<m@abreto.net>. */
#include <cstdio>
#include <cstring>
#include <queue>
#include <utility>

using namespace std;

#define MAXN    128

struct edge {
  int v;
  int w;
  int n;
};
edge edg[MAXN<<1];
int nedg;
int indegree[MAXN];
int front[MAXN];
int find_edge(int u, int v) {
  int e = u[front];
  while(e) {
    if(edg[e].v == v) return e;
    e = edg[e].n;
  }
  return 0;
}
void add_edge(int u, int v, int w) {
  int e = find_edge(u,v);
  if(0==e) {
    int newnode = ++nedg;
    edg[newnode].v = v;
    edg[newnode].w = w;
    edg[newnode].n = u[front];
    u[front] = newnode;
    indegree[v]++;
  } else {
    edg[e].w = (w < edg[e].w)?w:(edg[e].w);
  }
}

int n;

char inq[MAXN];
int vis[MAXN];
int d[MAXN];
```

```
int spfa(int s) { /* return 1 if fuhuan exists. */
  queue<int> q;
  memset(inq, 0, sizeof(inq));
  memset(d, -1, sizeof(d));
  memset(vis, 0, sizeof(vis));
  d[s] = 0;
  inq[s] = 1;
  q.push(s);
  while(!q.empty()) {
    int u = q.front();
    q.pop();
    printf("proc %d..\n", u);
    inq[u] = 0;
    if(vis[u]++ > n)
      return 1;
    for(int e = front[u]; e; e = edg[e].n) {
      int v = edg[e].v, w = edg[e].w;
      if( -1==d[v] || d[u] + w < d[v] ) {
        d[v] = d[u] + w;
        if(!inq[v]) {
          inq[v] = 1;
          q.push(v);
        }
      }
    }
  }
  return 0;
}
```

4.6 Maxflow

```
/* Max Flow Problem, by Abreto<m@abreto.net> */

#include <bits/stdc++.h>
using namespace std;

#define MAXV    (100000)
#define MAXE    (1000000)
struct edge {
  static int N;
  int v, w;
  edge *n;
  edge(void):v(0),w(0),n(NULL) {}
  edge(int vv, int ww, edge *nn):v(vv),w(ww),n(nn) {}
};
int nE;
edge E[MAXE];
edge *front[MAXV];
void add_edge(int u, int v, int w) {
  int ne = ++nE;
  E[ne] = edge(v, w, u[front]);
  u[front] = &(E[ne]);
}
edge *find_edge(int u, int v) {
  for(edge *e = u[front]; e != NULL; e = e->n)
    if(e->v == v)
      return e;
  return NULL;
}
void grant_e(int u, int v, int w) {
  edge *e = find_edge(u, v);
  if(NULL == e) add_edge(u,v,w);
  else e->w += w;
```

```
}

int vis[MAXV];
int path[MAXV];
int dfs(int u, int t) {
  vis[u] = 1;
  if(u == t) return 1;
  for(edge *e = u[front]; e != NULL; e = e->n) {
    int v = e->v;
    if(!vis[v] && e->w && dfs(v,t)) {
      path[u] = v;
      return 1;
    }
  }
  return 0;
}
int find_path(int s, int t) {
  memset(vis, 0, sizeof(vis));
  return dfs(s,t);
}
int max_flow(int s, int t) {
  int flow = 0;
  while(find_path(s,t)) {
    int i = 0;
    int minf = find_edge(s,path[s])->w;
    for(i = path[s]; i != t; i = path[i])
      minf = min(minf, find_edge(i,path[i])->w);
    for(i = s; i != t; i = path[i]) {
      grant_e(i, path[i], -minf);
      grant_e(path[i], i, minf);
    }
    flow += minf;
  }
  return flow;
}

/* Dinic */
#define N 1000
#define INF 100000000

struct Edge {
  int from,to,cap,flow;
  Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f) {}
};

struct Dinic {
  int n,m,s,t;//结点数，边数（包括反向弧），源点编号，汇点编号
  vector<Edge>edges;//边表，dges[e]和dges[e^1]互为反向弧
  vector<int>G[N];//邻接表，G[i][j]表示结点i的第j条边在e数组中的编号
  bool vis[N]; //BFS的使用
  int d[N]; //从起点到i的距离
  int cur[N]; //当前弧下标

  void addedge(int from,int to,int cap) {
    edges.push_back(Edge(from,to,cap,0));
    edges.push_back(Edge(to,from,0,0));
    int  m=edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }

  bool bfs() {
    memset(vis,0,sizeof(vis));
    queue<int>Q;
```

```
    Q.push(s);
    d[s]=0;
    vis[s]=1;
    while(!Q.empty()) {
      int x=Q.front();
      Q.pop();
      for(int i=0; i<G[x].size(); i++) {
        Edge&e=edges[G[x][i]];
        if(!vis[e.to]&&e.cap>e.flow) { //只考虑残量网络中的弧
          vis[e.to]=1;
          d[e.to]=d[x]+1;
          Q.push(e.to);
        }
      }

    }
    return vis[t];
  }

  int dfs(int x,int a) { //x表示当前结点，a表示目前为止的最小残量
    if(x==t||a==0)return a;//a等于0时及时退出，此时相当于断路了
    int flow=0,f;
    for(int&i=cur[x]; i<G[x].size(); i++) { //从上次考虑的弧开始，注意要使用引用，同时修改cur[x]
      Edge&e=edges[G[x][i]];//e是一条边
      if(d[x]+1==d[e.to]&&(f=dfs(e.to,min(a,e.cap-e.flow)))>0) {
        e.flow+=f;
        edges[G[x][i]^1].flow-=f;
        flow+=f;
        a-=f;
        if(!a)break;//a等于0及时退出，当a!=0,说明当前节点还存在另一个曾广路分支。

      }
    }
    return flow;
  }

  int Maxflow(int s,int t) { //主过程
    this->s=s,this->t=t;
    int flow=0;
    while(bfs()) { //不停地用bfs构造分层网络，然后用dfs沿着阻塞流增广
      memset(cur,0,sizeof(cur));
      flow+=dfs(s,INF);
    }
    return flow;
  }
};

/* ISAP */
struct Edge {
  int from,to,cap,flow;
};
const int maxn=650;
const int INF=0x3f3f3f3f;
struct ISAP {
  int n,m,s,t;//结点数，边数（包括反向弧），源点编号，汇点编号
  vector<Edge>edges;
  vector<int>G[maxn];
  bool vis[maxn];
  int d[maxn];
  int cur[maxn];
  int p[maxn];
  int num[maxn];
  void AddEdge(int from,int to,int cap) {
```

```
    edges.push_back((Edge) {
      from,to,cap,0
    });
    edges.push_back((Edge) {
      to,from,0,0
    });
    m=edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }
  bool RevBFS() {
    memset(vis,0,sizeof(vis));
    queue<int>Q;
    Q.push(t);
    d[t]=0;
    vis[t]=1;
    while(!Q.empty()) {
      int x=Q.front();
      Q.pop();
      for(int i=0; i<G[x].size(); i++) {
        Edge &e =edges[G[x][i]^1];
        if(!vis[e.from]&&e.cap>e.flow) {
          vis[e.from]=1;
          d[e.from]=d[x]+1;
          Q.push(e.from);
        }
      }
    }
    return vis[s];
  }
  int Augment() {
    int x=t, a=INF;
    while(x!=s) {
      Edge &e = edges[p[x]];
      a= min(a,e.cap-e.flow);
      x=edges[p[x]].from;
    }
    x=t;
    while(x!=s) {
      edges[p[x]].flow+=a;
      edges[p[x]^1].flow-=a;
      x=edges[p[x]].from;
    }
    return a;
  }
  int Maxflow(int s,int t,int n) {
    this->s=s,this->t=t,this->n=n;
    int flow=0;
    RevBFS();
    memset(num,0,sizeof(num));
    for(int i=0; i<n; i++) {
      num[d[i]]++;
    }
    int x=s;
    memset(cur,0,sizeof(cur));
    while(d[s]<n) {
      if(x==t) {
        flow+=Augment();
        x=s;
      }
      int ok=0;
      for(int i=cur[x]; i<G[x].size(); i++) {
        Edge &e =edges[G[x][i]];
        if(e.cap>e.flow&&d[x]==d[e.to]+1) {
```

```
          ok=1;
          p[e.to]=G[x][i];
          cur[x]=i;
          x=e.to;
          break;
        }
      }
      if(!ok) {
        int m=n-1;
        for(int i=0; i<G[x].size(); i++) {
          Edge &e =edges[G[x][i]];
          if(e.cap>e.flow)
            m=min(m,d[e.to]);
        }
        if(--num[d[x]]==0)
          break;
        num[d[x]=m+1]++;
        cur[x]=0;
        if(x!=s)
          x=edges[p[x]].from;
      }
    }
    return flow;
  }
};
int main() {
  int n,m,a,b,c,res;
  while(scanf("%d%d",&m,&n)!=EOF) {
    ISAP tmp;
    for(int i=0; i<m; i++) {
      scanf("%d%d%d",&a,&b,&c);
      tmp.AddEdge(a,b,c);
    }
    res=tmp.Maxflow(1,n,n);
    printf("%d\n",res);
  }
  return 0;
}
```

4.7 Strongly Connected Component

```
/* Kosaraju */
#define MAXN    10010
#define MAXM    100010
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv, edge *nn):v(vv),n(nn) {}
};
int nE;
edge E[MAXM<<1];
edge *ori[MAXN];
edge *inv[MAXN];
void add_edge(edge *front[], int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}
void connect(int u, int v) {
  add_edge(ori, u, v);
  add_edge(inv, v, u);
}
```

```

int vis[MAXN];
int vst[MAXN];
void first_dfs(int u, int &sig) {
  vis[u] = 1;
  for(edge *e = u[ori]; e; e = e->n)
    if(!vis[e->v])
      first_dfs(e->v, sig);
  vst[++sig] = u;
}
int mark[MAXN];
void second_dfs(int u, int sig) {
  vis[u] = 1;
  mark[u] = sig;
  for(edge *e = u[inv]; e; e = e->n)
    if(!vis[e->v])
      second_dfs(e->v, sig);
}

int N, M;

int kosaraju(void) {
  int i;
  int sig = 0;
  for(i = 0; i <= N; ++i) vis[i] = 0;
  for(i = 1; i <= N; ++i) {
    if(!vis[i])
      first_dfs(i, sig);
  }
  sig = 1;
  for(i = 0; i <= N; ++i) vis[i] = 0;
  for(i = N; i > 0; --i) {
    if(!vis[vst[i]])
      second_dfs(vst[i], sig++);
  }
  for(i = 1; i <= N; ++i)
    if(mark[i] != 1)
      return 0;
  return 1;
}


void clear(void) {
  nE = 0;
  for(int i = 0; i <= N; ++i) {
    ori[i] = inv[i] = NULL;
  }
}

/* Tarjan */
#define MAXN    10010
#define MAXM    100010
struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv, edge *nn):v(vv),n(nn) {}
};
typedef edge *ep;
int nE;
edge E[MAXM];
edge *front[MAXN];
void add_edge(int u, int v) {
  int ne = ++nE;
```

```
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
}

int mark[MAXN];
int dfn[MAXN], low[MAXN];
int stk[MAXN];
int stk_top;

void tardfs(int u, int stamp, int &scc) {
  mark[u] = 1;
  dfn[u] = low[u] = stamp;
  stk[stk_top++] = u;
  for(ep e = u[front]; e; e = e->n) {
    if(0 == mark[e->v]) tardfs(e->v, ++stamp, scc);
    if(1 == mark[e->v]) low[u] = min(low[u], low[e->v]);
  }
  if(dfn[u] == low[u]) {
    ++scc;
    do {
      low[stk[stk_top-1]] = scc;
      mark[stk[stk_top-1]] = 2;
    } while(stk[(stk_top--)-1] != u);
  }
}

int tarjan(int n) {
  int scc = 0, lay = 1;
  for(int i = 1; i <= n; ++i)
    if(0 == mark[i])
      tardfs(i, lay, scc);
  return scc;
}

int N, M;

void clear(void) {
  nE = 0;
  for(int i = 0; i <= N; ++i) {
    i[front] = NULL;
    mark[i] = low[i] = 0;
  }
  stk_top = 0;
}

/* Garbow */
#define MAXN    10010
#define MAXM    100010

struct edge {
  int v;
  edge *n;
  edge(void):v(0),n(NULL) {}
  edge(int vv, edge *nn):v(vv),n(nn) {}
};
typedef edge *ep;

int nE;
edge E[MAXM];
edge *front[MAXN];
void add_edge(int u, int v) {
  int ne = ++nE;
  E[ne] = edge(v, u[front]);
  u[front] = &(E[ne]);
```

```
}

int stk1[MAXN], stk1t;
int stk2[MAXN], stk2t;
int low[MAXN], belg[MAXN];

void garbowdfs(int u, int lay, int &scc) {
  stk1[++stk1t] = u;
  stk2[++stk2t] = u;
  low[u] = ++lay;
  for(ep e=u[front]; e; e = e->n) {
    if(!low[e->v]) garbowdfs(e->v, lay, scc);
    else if (0 == belg[e->v])
      while(low[stk2[stk2t]] > low[e->v])
        --stk2t;
  }
  if(stk2[stk2t] == u) {
    stk2t--;
    scc++;
    do {
      belg[stk1[stk1t]] = scc;
    } while(stk1[stk1t--] != u);
  }
}

int grabow(int n) {
  int i;
  int scc = 0, lay = 0;
  for(i = 0; i <= n; ++i) {
    belg[i] = low[i] = 0;
  }
  for(i = 1; i <= n; ++i)
    if(0 == low[i])
      garbowdfs(i, lay, scc);
  return scc;
}

int N, M;

void clear(void) {
  nE = 0;
  for(int i = 0; i <= N; ++i) {
    front[i] = NULL;
  }
}
```

5 Math

5.1 Euler Function

```
/* Euler function phi(x), by Abreto<m@abreto.net>. */

#define MAXX    3000000

int phi[MAXX];
void get_euler(void) {
  int i = 0, j = 0;
  phi[1] = 1;
  for(i = 2; i < MAXX; ++i)
    if(!phi[i])
      for(j = i; j < MAXX; j += i) {
        if(!phi[j]) phi[j] = j;
```

```
      phi[j] = phi[j]/i * (i-1);
    }
}
```

5.2 Chinese Remainder Theorem

$$x \equiv a_i \pmod{m_i}$$

```
/* Chinese Remainder Theorem, by Abreto<m@abreto.net>. */
#include "euler.c"

#define MAXN    64

typedef long long int ll;

ll quickpow(ll a, ll b, ll mod) {
  ll ret = 1, base = a;
  while(b > 0) {
    if(b & 1) ret = (ret * base) % mod;
    base = (base * base) % mod;
    b >>= 1;
  }
  return ret;
}

ll N;
/* x = a[i] (mod m[i]) */
ll a[MAXN], m[MAXN]; /* a and m is indexed from 0. */
ll x, M;
/* x: only solution (mod M) */

void naive_crt(void) {
  int i = 0;
  ll Mi[MAXN], nMi[MAXN];
  ll t[MAXN];

  M = 1;
  for(i = 0; i < N; ++i)
    M *= m[i];
  for(i = 0; i < N; ++i)
    Mi[i] = M / m[i];
  get_euler();
  for(i = 0; i < N; ++i)
    nMi[i] = quickpow(Mi[i], phi[a[i]]-1, a[i]);
  for(i = 0; i < N; ++i)
    t[i] = ((a[i] * Mi[i]) % M) * nMi[i] % M;
  for(i = 0; i < N; ++i)
    x = (x + t[i]) % M;
}
```

5.3 FFT

```
#include <cmath>
using namespace std;
namespace fft {
#define eps (1e-9)
template < typename T = double >
struct dbl {
  T x;
  dbl(void):x(0.0) {}
```

```
  template <typename U>
  dbl(U a):x((T)a) {}
  inline char sgn(void) {
    return ((x>=-eps)&&(x<=eps))?(0):((x>eps)?(1):(-1));
  }
  inline T tabs(void) {
    return ((x>=-eps)&&(x<=eps))?(0.0):((x>eps)?(x):(-x));
  }
  inline dbl abs(void) {
    return dbl(tabs());
  }
  template <typename U> inline dbl &operator=(const U b) {
    x=(T)b;
    return (*this);
  }
  inline T *operator&(void) {
    return &x;
  }
  inline dbl operator-(void) const {
    return dbl(-x);
  }
  inline dbl operator+(const dbl &b) const {
    return dbl(x+b.x);
  }
  inline dbl operator-(const dbl &b) const {
    return dbl(x-b.x);
  }
  inline dbl operator*(const dbl &b) const {
    return dbl(x*b.x);
  }
  inline dbl operator/(const dbl &b) const {
    return dbl(x/b.x);
  }
  template <typename U> inline dbl operator^(const U &b) const {
    T ret=1.0,base=x;
    while(b) {
      if(b&1)ret*=base;
      base*=base;
      b>>=1;
    }
    return dbl(ret);
  }
  inline dbl operator+=(const dbl &b) {
    return dbl(x+=b.x);
  }
  inline dbl operator-=(const dbl &b) {
    return dbl(x-=b.x);
  }
  inline dbl operator*=(const dbl &b) {
    return dbl(x*=b.x);
  }
  inline dbl operator/=(const dbl &b) {
    return dbl(x/=b.x);
  }
  template <typename U> inline dbl operator^=(const U &b) {
    dbl tmp=(*this)^b;
    *this=tmp;
    return tmp;
  }
  inline bool operator==(const dbl &b) const {
    return (0 == ((*this)-b).sgn());
  }
  inline bool operator!=(const dbl &b) const {
    return (0 != ((*this)-b).sgn());
```

```
  }
  inline bool operator<(const dbl &b) const {
    return (-1 == ((*this)-b).sgn());
  }
  inline bool operator<=(const dbl &b) const {
    return (((*this)==b) || ((*this)<b));
  }
  inline bool operator>(const dbl &b) const {
    return (b < (*this));
  }
  inline bool operator>=(const dbl &b) const {
    return (((*this)==b) || ((*this)>b));
  }
  template <typename U> inline operator U() const {
    return (U)x;
  }
  inline char operator[](unsigned n) {
    if(n >= 0) {
      long long int ret=x;
      while(n--) {
        ret/=10;
      }
      return (ret%10);
    } else {
      T ret=x;
      n=-n;
      while(n--)ret*=10.0;
      return ((long long int)ret)%10;
    }
  }
};
template <typename T>
struct Complex {
  T x,y;  /* x + iy */
  Complex(void):x(T()),y(T()) {}
  Complex(T xx):x(xx) {}
  Complex(T xx,T yy):x(xx),y(yy) {}
  inline Complex operator-(void) const {
    return Complex(-x,-y);
  }
  inline Complex operator+(const Complex& b) const {
    return Complex(x+b.x,y+b.y);
  }
  inline Complex operator-(const Complex& b) const {
    return Complex(x-b.x,y-b.y);
  }
  inline Complex operator*(const Complex& b) const {
    return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
  }
  inline Complex operator/(const Complex& b) const {
    T bo=b.x*b.x+b.y*b.y;
    return Complex((x*b.x+y*b.y)/bo,(y*b.x-x*b.y)/bo);
  }
  inline Complex& operator+=(const Complex& b) {
    Complex tmp=(*this)+b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator-=(const Complex& b) {
    Complex tmp=(*this)-b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator*=(const Complex& b) {
```

```
    Complex tmp=(*this)*b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator/=(const Complex& b) {
    Complex tmp=(*this)/b;
    (*this)=tmp;
    return (*this);
  }
  inline friend Complex operator+(const T& a, const Complex& b) {
    return Complex(a)+b;
  }
  inline friend Complex operator-(const T& a, const Complex& b) {
    return Complex(a)-b;
  }
  inline friend Complex operator*(const T& a, const Complex& b) {
    return Complex(a)*b;
  }
  inline friend Complex operator/(const T& a, const Complex& b) {
    return Complex(a)/b;
  }
};
typedef dbl<> Double;
typedef Complex<Double> ComplexD;
typedef long long int ll;
const int maxn = 2000000; /* !! */
const Double pi(acos(-1.0));

void build(ComplexD _P[], ComplexD P[], int n, int m, int curr, int &cnt) {
  if(m == n) {
    _P[curr] = P[cnt++];
  } else {
    build(_P, P, n, m*2, curr, cnt);
    build(_P, P, n, m*2, curr+m, cnt);
  }
}

void FFT(ComplexD P[], int n, int oper) { /* n should be 2^k. */
  static ComplexD _P[maxn];
  int cnt = 0;
  build(_P, P, n, 1, 0, cnt);
  copy(_P, _P+n, P);
  for(int d = 0; (1<<d)<n; ++d) {
    int m = 1<<d;
    int m2 = m*2;
    Double p0 = pi / m * oper;
    ComplexD unit_p0(cos(p0.x), sin(p0.x));
    for(int i = 0; i < n; i += m2) {
      ComplexD unit(1,0);
      for(int j = 0; j < m; ++j) {
        ComplexD &P1 = P[i+j+m], &P2 = P[i+j];
        ComplexD t = unit * P1;
        P1 = P2 - t;
        P2 = P2 + t;
        unit *= unit_p0;
      }
    }
  }
  if(-1 == oper) {
    for(int i = 0; i < n; ++i)
      P[i] /= Double(n);
  }
}
}
```

5.4 Number Theory Inverse

```
#include <bits/stdc++.h>
using namespace std;

const int n=10000000;      /*  */
const long long mod=1e9+7;  /*  prime required. */

long long fact[n],fiv[n],inv[n];

int main() {
  fact[0]=fact[1]=1;
  fiv[0]=fiv[1]=1;
  inv[1]=1;
  for (int i=2; i<n; i++) {
    fact[i]=fact[i-1]*i%mod;
    inv[i]=(mod-mod/i)*inv[mod%i]%mod;
    fiv[i]=inv[i]*fiv[i-1]%mod;
  }
  for (int i=1; i<n; i++) {
    if (fact[i]*fiv[i]%mod!=1)  printf("fact wrong: %d\n",i);
    if (inv[i]*i%mod!=1)        printf("intv wrong: %d\n",i);
  }
  cout<<"complete"<<endl;
  return 0;
}
```

5.5 Linear Programming

```
/* 线性规划 */
#include<bits/stdc++.h>

using namespace std;
const int Maxn=110,Maxm=59;
class Simplex {
  /*
     功能：
     接受有n个约束，m个基本变量的方程组a[0~n][0~m]
     a[0][]存放需要最大化的目标函数，a[][0]存放常数
     Base[]存放基本变量的id,初始为1~m
     Rest[]存放松弛变量的id,初始为m+1~m+n
     返回此线性规划的最小值ans
     要求方案的话，Base[]中的变量值为0,Rest[]中的变量值为相应行的[0]
     如果solve
     返回1,说明运行正常ans是它的最大值
     返回0,说明无可行解
     返回-1,说明解没有最大值
     测试：
     m=2,n=3
     double a[4][3]={
     {0,1,3},
     {8,-1,1},
     {-3,1,1},
     {2,1,-4}
     };
     solve=1,ans=64/3;
     注意ac不了可能是eps的问题
   */
public:
  static const double Inf;
  static const double eps;
  int n,m;
```

```
double a[Maxn][Maxm];
int Base[Maxm],Rest[Maxn];
double val[Maxm];
double ans;
void pt() {
  for(int i=0; i<=n; i++) {
    for(int j=0; j<=m; j++)printf("%.2f␣",a[i][j]);
    puts("");
  }
}
void pivot(int x,int y) { //将第x个非基本变量和第y个基本变量调换
  swap(Rest[x],Base[y]);
  double tmp=-1./a[x][y];
  a[x][y]=-1.;
  for(int j=0; j<=m; j++)a[x][j]*=tmp;
  for(int i=0; i<=n; i++) {
    if(i==x||fabs(a[i][y])<eps)continue;
    tmp=a[i][y];
    a[i][y]=0;
    for(int j=0; j<=m; j++)a[i][j]+=tmp*a[x][j];
  }
}
bool opt() {
  while(1) {
    int csi=0;
    for(int i=1; i<=m; i++)if(a[0][i]>eps&&(!csi||Base[i]<Base[csi]))csi=i;
    if(!csi)break;
    int csj=0;
    double cur;
    for(int j=1; j<=n; j++) {
      if(a[j][csi]>-eps)continue;
      double tmp=-a[j][0]/a[j][csi];
      if(!csj||tmp+eps<cur||(fabs(tmp-cur)<eps&&Rest[j]<Rest[csj]))csj=j,cur=tmp;
    }
    if(!csj)return 0;
    pivot(csj,csi);
  }
  ans=a[0][0];
  return 1;
}
bool init() {
  ans=0;
  for(int i=1; i<=m; i++)Base[i]=i;
  for(int i=1; i<=n; i++)Rest[i]=m+i;
  int cs=1;
  for(int i=2; i<=n; i++)if(a[i][0]<a[cs][0])cs=i;
  if(a[cs][0]>=-eps)return 1;
  static double tmp[Maxm];
  for(int i=0; i<=m; i++)tmp[i]=a[0][i],a[0][i]=0;
  for(int i=1; i<=n; i++)a[i][m+1]=1.;
  a[0][m+1]=-1.;
  Base[m+1]=m+n+1;
  pivot(cs,++m);
  opt();
  m--;
  if(a[0][0]<-eps)return 0;
  cs=-1;
  for(int i=1; i<=n; i++) {
    if(Rest[i]>m+n) {
      cs=i;
      break;
    }
  }
  if(cs>=1) {
```

```
      int nxt=-1;
      m++;
      for(int i=1; i<=m; i++)if(a[cs][i]>eps||a[cs][i]<-eps) {
          nxt=i;
          break;
        }
      pivot(cs,nxt);
      m--;
    }
    for(int i=1; i<=m; i++) {
      if(Base[i]>m+n) {
        swap(Base[i],Base[m+1]);
        for(int j=0; j<=n; j++)a[j][i]=a[j][m+1];
        break;
      }
    }
    for(int i=1; i<=m; i++)a[0][i]=0;
    a[0][0]=tmp[0];
    for(int i=1; i<=m; i++)if(Base[i]<=m)a[0][i]=tmp[Base[i]];
    for(int i=1; i<=n; i++) {
      if(Rest[i]<=m) {
        for(int j=0; j<=m; j++)a[0][j]+=tmp[Rest[i]]*a[i][j];
      }
    }
    return 1;
  }
  void getval() {
    for(int i=1; i<=m; i++)val[i]=0;
    for(int i=1; i<=n; i++)if(Rest[i]<=m)val[Rest[i]]=a[i][0];
    //for(int i=1;i<=m;i++)printf("%.2f ",val[i]);puts("");
  }
  int solve() {
    if(!init())return 0;
    if(!opt())return -1;
    getval();
    return 1;
  }
} solver;
const double Simplex:: Inf=1e80;
const double Simplex:: eps=1e-8;
int main() {
  int m,n,type;
  scanf("%d%d%d",&m,&n,&type);
  solver.a[0][0]=0;
  for(int i=1; i<=m; i++)scanf("%lf",&solver.a[0][i]);
  for(int i=1; i<=n; i++) {
    for(int j=1; j<=m+1; j++) {
      if(j==m+1)scanf("%lf",&solver.a[i][0]);
      else {
        scanf("%lf",&solver.a[i][j]);
        solver.a[i][j]=-solver.a[i][j];
      }
    }
  }
  solver.m=m,solver.n=n;
  int rep=solver.solve();
  if(rep==0)puts("Infeasible");
  else if(rep==-1)puts("Unbounded");
  else {
    printf("%.12f\n",solver.ans);
    if(type==1) {
      for(int i=1; i<=m; i++)printf("%.12f%c",solver.val[i],i==m?'\n':' ');
    }
  }
```

```
}
```

6 String

6.1 Hash

```
/* Common hash for any substrings. */

typedef unsigned long long int llu;
#define MAXN 1000000
int n;
char s[MAXN];
llu H[MAXN], xP[MAXN], P = 99991ll;
void init(void) {
  int i = 0;
  xP[0] = 1ll;
  for(i = 1; i < MAXN; ++i) xP[i] = xP[i-1] * P;
  H[n] = 0;
  for(i = n-1; i >= 0; --i) H[i] = H[i+1]*P + s[i];
}
#define HASH(i,l)   (H[i] - H[i+l]*xP[l])
```

6.2 KMP

```
/* KMP, by Abreto<m@abreto.net>. */
#include <string.h>

/* !!NEED IMPROVING!! */

#define MAXW    10002
#define MAXT    1000002
char W[MAXW];   /* pattern */
char T[MAXT];
int pi[MAXW];

void compute(void) {
  int i = 0, k = 0;
  int m = strlen(W+1);
  pi[1] = 0;
  for(i = 2; i <= m; i++) {
    while(k && W[k+1]!=W[i])
      k = pi[k];
    if(W[k+1] == W[i])
      k++;
    pi[i] = k;
  }
}
int kmp(void) {
  int i = 0, q = 0;
  int ret = 0;
  int n = strlen(T+1), m = strlen(W+1);
  compute();
  for(i = 1; i <= n; ++i) {
    while(q && W[q+1]!=T[i])
      q = pi[q];
    if( W[q+1]==T[i])
      q++;
    if(q == m) {
      ret++;
      q = pi[q];
    }
```

```
  }
  return ret;
}
```

6.3 Suffix Array

```
/* Suffix Array, copied. */

#define MAXN    (200010)
namespace mzry_sa {
int wx[MAXN],wy[MAXN],*x,*y,wss[MAXN],wv[MAXN];

bool dacmp(int *r,int n,int a,int b,int l) {
  return a+l<n && b+l<n && r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int str[],int sa[],int rank[],int height[],int n,int m) {
  int *s = str;
  int *x=wx,*y=wy,*t,p;
  int i,j;
  for(i=0; i<m; i++)wss[i]=0;
  for(i=0; i<n; i++)wss[x[i]=s[i]]++;
  for(i=1; i<m; i++)wss[i]+=wss[i-1];
  for(i=n-1; i>=0; i--)sa[--wss[x[i]]]=i;
  for(j=1,p=1; p<n && j<n; j*=2,m=p) {
    for(i=n-j,p=0; i<n; i++)y[p++]=i;
    for(i=0; i<n; i++)if(sa[i]-j>=0)y[p++]=sa[i]-j;
    for(i=0; i<n; i++)wv[i]=x[y[i]];
    for(i=0; i<m; i++)wss[i]=0;
    for(i=0; i<n; i++)wss[wv[i]]++;
    for(i=1; i<m; i++)wss[i]+=wss[i-1];
    for(i=n-1; i>=0; i--)sa[--wss[wv[i]]]=y[i];
    for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)
      x[sa[i]]=dacmp(y,n,sa[i-1],sa[i],j)?p-1:p++;
  }
  for(int i=0; i<n; i++) rank[sa[i]]=i;
  for(int i=0,j=0,k=0; i<n; height[rank[i++]]=k)
    if(rank[i]>0)
      for(k?k--:0,j=sa[rank[i]-1];
          i+k < n && j+k < n && str[i+k]==str[j+k];
          k++);
}
}

/*
Suffix array O(n lg^2 n)
LCP table O(n)
*/
#include <cstdio>
#include <algorithm>
#include <cstring>

using namespace std;

#define REP(i, n) for (int i = 0; i < (int)(n); ++i)

namespace SuffixArray {
const int MAXN = 1 << 21;
char * S;
int N, gap;
int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];

bool sufCmp(int i, int j) {
  if (pos[i] != pos[j])
```

```
    return pos[i] < pos[j];
  i += gap;
  j += gap;
  return (i < N && j < N) ? pos[i] < pos[j] : i > j;
}

void buildSA() {
  N = strlen(S);
  REP(i, N) sa[i] = i, pos[i] = S[i];
  for (gap = 1;; gap <<= 1) {
    sort(sa, sa + N, sufCmp);
    REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i + 1]);
    REP(i, N) pos[sa[i]] = tmp[i];
    if (tmp[N - 1] == N - 1) break;
  }
}

void buildLCP() {
  for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1) {
      for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
        ++k;
      lcp[pos[i]] = k;
      if (k)--k;
    }
}
} // end namespace SuffixArray

namespace HashSuffixArray {
const int
MAXN = 1 << 21;

typedef unsigned long long hash;

const hash BASE = 137;

int N;
char * S;
int sa[MAXN];
hash h[MAXN], hPow[MAXN];

#define getHash(lo, size) (h[lo] - h[(lo) + (size)] * hPow[size])

inline bool sufCmp(int i, int j) {
  int lo = 1, hi = min(N - i, N - j);
  while (lo <= hi) {
    int mid = (lo + hi) >> 1;
    if (getHash(i, mid) == getHash(j, mid))
      lo = mid + 1;
    else
      hi = mid - 1;
  }
  return S[i + hi] < S[j + hi];
}

void buildSA() {
  N = strlen(S);
  hPow[0] = 1;
  for (int i = 1; i <= N; ++i)
    hPow[i] = hPow[i - 1] * BASE;
  h[N] = 0;
  for (int i = N - 1; i >= 0; --i)
    h[i] = h[i + 1] * BASE + S[i], sa[i] = i;

  stable_sort(sa, sa + N, sufCmp);
```

```
}

} // end namespace HashSuffixArray

namespace lrj_sa {
const int MAXN = 1000;
char s[MAXN]; /* 原始字符数组（最后一个字符应必须是0，而前面的字符必须非0） */
int sa[MAXN], t[MAXN], t2[MAXN], c[MAXN], n;  /* n seems to be the length of s. */
/* every charactor is in [0,m-1] */
void build_sa(int m) {
  int i, *x = t, *y = t2;
  for(i = 0; i < m; ++i) c[i] = 0;
  for(i = 0; i < n; i++) c[x[i]=s[i]]++;
  for(i = 1; i < m; ++i) c[i] += c[i-1];
  for(i = n-1; i >= 0; --i) sa[--c[x[i]]] = i;
  for(int k = 1; k <= n; k <<= 1) {
    int p = 0;
    for(i = n-k; i < n; ++i) y[p++] = i;
    for(i = 0; i < n; ++i) if(sa[i] >= k) y[p++] = sa[i]-k;
    for(i = 0; i < m; i++) c[i] = 0;
    for(i = 0; i < n; i++) c[x[y[i]]]++;
    for(i = 0; i < m; ++i) c[i]+=c[i-1];
    for(i = n-1; i >= 0; --i) sa[--c[x[y[i]]]] = y[i];
    swap(x,y);
    p = 1;
    x[sa[0]] = 0;
    for(i = 1; i < n; ++i)
      x[sa[i]] = y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k] ? p-1:p++;
    if(p >= n) break;
    m = p;
  }
}
int rank[MAXN], height[MAXN];
void get_height(void) {
  int i,j,k = 0;
  for(i = 0; i < n; ++i) rank[sa[i]] = i;
  for(i = 0; i < n; ++i) {
    if(k) k--;
    j = sa[rank[i]-1];
    while(s[i+k]==s[j+k]) k++;
    height[rank[i]] = k;
  }
}
} // end namespace lrj_sa
```

7 Tool

7.1 IO plug-in

```
/* I/O Plug-in, by Abreto <m@abreto.net>. */
#include <stdio.h>

#if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
#define INT64 "%I64d"
#else
#define INT64 "%lld"
#endif

#if ( _WIN32 || __WIN32__ || _WIN64 || __WIN64__ )
#define UNS64 "%I64u"
#else
#define UNS64 "%llu"
```

```
#endif

#define ISDIGIT(x) ((x>='0')&&(x<='9'))
int readn(int *n) {
  int c=0;
  *n=0;
  for(; !ISDIGIT(c); c=getchar());
  for(; ISDIGIT(c); c=getchar())*n=(*n)*10+c-'0';
  return (*n);
}
void putn(int n) {
  int ns[16]= {0,n%10},nd=1;
  while(n/=10)ns[++nd]=n%10;
  while(nd)putchar(ns[nd--]+'0');
}
```

7.2 Matrix (including quickpow)

```

typedef long long int ll;
template <class T, int maxn, ll mod>
struct mat {
  int N;
  T a[maxn][maxn];    /* 1-based. */
  mat(void):N(0) {}
  mat(int n, int v = 0) {
    N = n;
    for(int i = 1; i <= N; ++i)
      for(int j = 1; j <= N; ++j)
        a[i][j] = 0;
    for(int i = 1; i <= N; ++i)
      a[i][i] = v;
  }
  mat operator-(void) const {
    mat ret(N);
    for(int i = 1; i <= N; ++i)
      for(int j = 1; j <= N; ++j)
        ret.a[i][j] = (mod-a[i][j])%mod;
    return ret;
  }
  mat operator+(const mat &b) const {
    mat ret(N);
    for(int i = 1; i <= N; ++i)
      for(int j = 1; j <= N; ++j)
        ret.a[i][j] = (a[i][j] + b.a[i][j])%mod;
    return ret;
  }
  mat operator-(const mat &b) const {
    mat ret(N);
    for(int i = 1; i <= N; ++i)
      for(int j = 1; j <= N; ++j)
        ret.a[i][j] = (a[i][j] - b.a[i][j] + mod) % mod;
    return ret;
  }
  mat operator*(const mat &b) const {
    mat ret(N);
    for(int i = 1; i <= N; ++i)
      for(int j = 1; j <= N; ++j)
        for(int k = 1; k <= N; ++k) {
          /* T t = (a[i][k] * b.a[k][j]) % mod; // delete %mod if get TLE. */
          ret.a[i][j] = (ret.a[i][j] + a[i][k] * b.a[k][j]) % mod; /* seprate this
              line if get WA. */
        }
```

```
    return ret;
  }
  mat operator^(long long int p) const {
    mat ret(N,1);
    mat base = (*this);
    while(p) {
      if(p & 1) ret = ret * base;
      base = base * base;
      p >>= 1;
    }
    return ret;
  }
};
```

7.3 Double Class

```
/* Double Class, by Abreto<m@abreto.net>. */

#define eps (1e-9)
template < typename T = double >
struct dbl {
  T x;
  dbl(void):x(0.0) {}
  template <typename U>
  dbl(U a):x((T)a) {}
  inline char sgn(void) {
    return ((x>=-eps)&&(x<=eps))?(0):((x>eps)?(1):(-1));
  }
  inline T tabs(void) {
    return ((x>=-eps)&&(x<=eps))?(0.0):((x>eps)?(x):(-x));
  }
  inline dbl abs(void) {
    return dbl(tabs());
  }
  template <typename U> inline dbl &operator=(const U b) {
    x=(T)b;
    return (*this);
  }
  inline T *operator&(void) {
    return &x;
  }
  inline dbl operator-(void) const {
    return dbl(-x);
  }
  inline dbl operator+(const dbl &b) const {
    return dbl(x+b.x);
  }
  inline dbl operator-(const dbl &b) const {
    return dbl(x-b.x);
  }
  inline dbl operator*(const dbl &b) const {
    return dbl(x*b.x);
  }
  inline dbl operator/(const dbl &b) const {
    return dbl(x/b.x);
  }
  template <typename U> inline dbl operator^(const U &b) const {
    T ret=1.0,base=x;
    while(b) {
      if(b&1)ret*=base;
      base*=base;
      b>>=1;
    }
```

```
    return dbl(ret);
  }
  inline dbl operator+=(const dbl &b) {
    return dbl(x+=b.x);
  }
  inline dbl operator-=(const dbl &b) {
    return dbl(x-=b.x);
  }
  inline dbl operator*=(const dbl &b) {
    return dbl(x*=b.x);
  }
  inline dbl operator/=(const dbl &b) {
    return dbl(x/=b.x);
  }
  template <typename U> inline dbl operator^=(const U &b) {
    dbl tmp=(*this)^b;
    *this=tmp;
    return tmp;
  }
  inline bool operator==(const dbl &b) const {
    return (0 == ((*this)-b).sgn());
  }
  inline bool operator!=(const dbl &b) const {
    return (0 != ((*this)-b).sgn());
  }
  inline bool operator<(const dbl &b) const {
    return (-1 == ((*this)-b).sgn());
  }
  inline bool operator<=(const dbl &b) const {
    return (((*this)==b) || ((*this)<b));
  }
  inline bool operator>(const dbl &b) const {
    return (b < (*this));
  }
  inline bool operator>=(const dbl &b) const {
    return (((*this)==b) || ((*this)>b));
  }
  template <typename U> inline operator U() const {
    return (U)x;
  }
  inline char operator[](unsigned n) {
    if(n >= 0) {
      long long int ret=x;
      while(n--) {
        ret/=10;
      }
      return (ret%10);
    } else {
      T ret=x;
      n=-n;
      while(n--)ret*=10.0;
      return ((long long int)ret)%10;
    }
  }
};

typedef dbl<> Double;
```

7.4 Complex Class

```
/* Complex Class, by Abreto<m@abreto.net>. */

template <typename T>
```

```
struct Complex {
  T x,y;  /* x + iy */
  Complex(void):x(T()),y(T()) {}
  Complex(T xx):x(xx) {}
  Complex(T xx,T yy):x(xx),y(yy) {}
  inline Complex operator-(void) const {
    return Complex(-x,-y);
  }
  inline Complex operator+(const Complex& b) const {
    return Complex(x+b.x,y+b.y);
  }
  inline Complex operator-(const Complex& b) const {
    return Complex(x-b.x,y-b.y);
  }
  inline Complex operator*(const Complex& b) const {
    return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
  }
  inline Complex operator/(const Complex& b) const {
    T bo=b.x*b.x+b.y*b.y;
    return Complex((x*b.x+y*b.y)/bo,(y*b.x-x*b.y)/bo);
  }
  inline Complex& operator+=(const Complex& b) {
    Complex tmp=(*this)+b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator-=(const Complex& b) {
    Complex tmp=(*this)-b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator*=(const Complex& b) {
    Complex tmp=(*this)*b;
    (*this)=tmp;
    return (*this);
  }
  inline Complex& operator/=(const Complex& b) {
    Complex tmp=(*this)/b;
    (*this)=tmp;
    return (*this);
  }
  inline friend Complex operator+(const T& a, const Complex& b) {
    return Complex(a)+b;
  }
  inline friend Complex operator-(const T& a, const Complex& b) {
    return Complex(a)-b;
  }
  inline friend Complex operator*(const T& a, const Complex& b) {
    return Complex(a)*b;
  }
  inline friend Complex operator/(const T& a, const Complex& b) {
    return Complex(a)/b;
  }
};
```

8 Appendix

9 Graph Algorithms

Welcome to the new episode of PrinceOfPersia presents: Fun with algorithms ;)
You can find all the definitions here in the book "Introduction to graph theory", Douglas.B West.

Important graph algorithms :

9.1 DFS

The most useful graph algorithms are search algorithms. DFS (Depth First Search) is one of them. While running DFS, we assign colors to the vertices (initially white). Algorithm itself is really simple :

```
dfs (v):
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u)
    color[v] = black
```

Black color here is not used, but you can use it sometimes.
Time complexity : $O(n + m)$.

9.1.1 DFS tree

DFS tree is a rooted tree that is built like this :

```
let T be a new tree
dfs (v):
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u) and par[u] = v (in T)

    color[v] = black
```

Lemma: There is no cross edges, it means if there is an edge between v and u, then v = par[u] or u = par[v].

9.1.2 Starting time, finishing time

Starting time of a vertex is the time we enter it (the order we enter it) and its finishing time is the time we leave it. Calculating these are easy :

```
TIME = 0
dfs (v):
    st[v] = TIME ++
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then dfs(u)
    color[v] = black
    ft[v] = TIME // or we can use TIME ++
```

It is useable in specially data structure problems (convert the tree into an array).
Lemma: If we run dfs(root) in a rooted tree, then v is an ancestor of u if and only if stv stu ftu ftv
.
So, given arrays st and ft we can rebuild the tree.

9.1.3 Finding cut edges

The code below works properly because the lemma above (first lemma):

```
h[root] = 0
par[v] = -1
dfs (v):
    d[v] = h[v]
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
            if d[u] > h[v]
                then the edge v-u is a cut edge
        else if u != par[v])
            then d[v] = min(d[v], h[u])
    color[v] = black
```

In this code, h[v] = height of vertex v in the DFS tree and d[v] = min(h[w] where there is at least vertex u in subtree of v in the DFS tree where there is an edge between u and w).

9.1.4 Finding cut vertices

The code below works properly because the lemma above (first lemma):

```
h[root] = 0
par[v] = -1
dfs (v):
    d[v] = h[v]
    color[v] = gray
    for u in adj[v]:
        if color[u] == white
            then par[u] = v and dfs(u) and d[v] = min(d[v], d[u])
            if d[u] >= h[v] and (v != root or number_of_children(v) > 1)
                then the edge v is a cut vertex
        else if u != par[v])
            then d[v] = min(d[v], h[u])
    color[v] = black
```

In this code, h[v] = height of vertex v in the DFS tree and d[v] = min(h[w] where there is at least vertex u in subtree of v in the DFS tree where there is an edge between u and w).

9.1.5 Finding Eulerian tours

It is quite like DFS, with a little change :

```
vector E
dfs (v):
    color[v] = gray
    for u in adj[v]:
        erase the edge v-u and dfs(u)
    color[v] = black
    push v at the end of e
```

e is the answer.

9.2 BFS

BFS is another search algorithm (Breadth First Search). It is usually used to calculate the distances from a vertex v to all other vertices in unweighted graphs.
Code :

```
BFS(v):
    for each vertex i
        do d[i] = inf
    d[v] = 0
    queue q
    q.push(v)
    while q is not empty
        u = q.front()
        q.pop()
        for each w in adj[u]
            if d[w] == inf
                then d[w] = d[u] + 1, q.push(w)
```

Distance of vertex u from v is d[u].
Time complexity : $O(n + m)$.

9.2.1 BFS tree

BFS tree is a rooted tree that is built like this :

```
let T be a new tree
    BFS(v):
        for each vertex i
            do d[i] = inf
        d[v] = 0
        queue q
        q.push(v)
        while q is not empty
            u = q.front()
            q.pop()
            for each w in adj[u]
                if d[w] == inf
                    then d[w] = d[u] + 1, q.push(w) and par[w] = u (in T)
```

9.3 SCC

The most useful and fast-coding algorithm for finding SCCs is Kosaraju.
In this algorithm, first of all we run DFS on the graph and sort the vertices in decreasing of their finishing time (we can use a stack).
Then, we start from the vertex with the greatest finishing time, and for each vertex v that is not yet in any SCC, do : for each u that v is reachable by u and u is not yet in any SCC, put it in the SCC of vertex v. The code is quite simple.

9.4 Shortest path

Shortest path algorithms are algorithms to find some shortest paths in directed or undirected graphs.

9.4.1 Dijkstra

This algorithm is a single source shortest path (from one source to any other vertices). Pay attention that you can't have edges with negative weight.
Pseudo code :

```
dijkstra(v) :
    d[i] = inf for each vertex i
    d[v] = 0
    s = new empty set
    while s.size() < n
        x = inf
        u = -1
        for each i in V-s //V is the set of vertices
            if x >= d[i]
                then x = d[i], u = i
        insert u into s
        // The process from now is called Relaxing
        for each i in adj[u]
            d[i] = min(d[i], d[u] + w(u,i))
```

There are two different implementations for this. Both are useful (C++11).
One) $O(n^2)$

```
int mark[MAXN];
void dijkstra(int v){
fill(d,d + n, inf);
fill(mark, mark + n, false);
d[v] = 0;
int u;
while(true){
int x = inf;
u = -1;
for(int i = 0;i < n;i ++)
if(!mark[i] and x >= d[i])
x = d[i], u = i;
if(u == -1) break;
mark[u] = true;
for(auto p : adj[u]) //adj[v][i] = pair(vertex, weight)
if(d[p.first] > d[u] + p.second)
d[p.first] = d[u] + p.second;
}
}
```

Two)
1) Using std :: set :

```
void dijkstra(int v){
fill(d,d + n, inf);
d[v] = 0;
int u;
set<pair<int,int> > s;
s.insert({d[v], v});
while(!s.empty()){
u = s.begin() -> second;
```

```
s.erase(s.begin());
for(auto p : adj[u]) //adj[v][i] = pair(vertex, weight)
if(d[p.first] > d[u] + p.second){
s.erase({d[p.first], p.first});
d[p.first] = d[u] + p.second;
s.insert({d[p.first], p.first});
}
}
}
```

2) Using std :: priority_queue (better):

```
bool mark[MAXN];
void dijkstra(int v){
fill(d,d + n, inf);
fill(mark, mark + n, false);
d[v] = 0;
int u;
priority_queue<pair<int,int>,vector<pair<int,int> >, less<pair<int,int> > > pq;
pq.push({d[v], v});
while(!pq.empty()){
u = pq.top().second;
pq.pop();
if(mark[u])
continue;
mark[u] = true;
for(auto p : adj[u]) //adj[v][i] = pair(vertex, weight)
if(d[p.first] > d[u] + p.second){
d[p.first] = d[u] + p.second;
pq.push({d[p.first], p.first});
}
}
}
```

9.4.2 Floyd-Warshall

Floyd-Warshal algorithm is an all-pairs shortest path algorithm using dynamic programming. It is too simple and undrestandable :

```
Floyd-Warshal()
d[v][u] = inf for each pair (v,u)
d[v][v] = 0 for each vertex v
for k = 1 to n
for i = 1 to n
for j = 1 to n
        d[i][j] = min(d[i][j], d[i][k] + d[k][j])
```

Time complexity : $O(n^3)$.

9.4.3 Bellman-Ford

Bellman-Ford is an algorithm for single source shortest path where edges can be negative (but if there is a cycle with negative weight, then this problem will be NP).
The main idea is to relax all the edges exactly n - 1 times (read relaxation above in dijkstra). You can prove this algorithm using induction.

If in the n - th step, we relax an edge, then we have a negative cycle (this is if and only if).
Code :

```
Bellman-Ford(int v)
d[i] = inf for each vertex i
d[v] = 0
for step = 1 to n
for all edges like e
i = e.first // first end
j = e.second // second end
w = e.weight
if d[j] > d[i] + w
if step == n
then return "Negative cycle found"
                d[j] = d[i] + w
```

Time complexity : $O(nm)$.

9.4.4 SPFA

SPFA (Shortest Path Faster Algorithm) is a fast and simple algorithm (single source) that its complexity is not calculated yet. But if m = O(n2) it's better to use the first implementation of Dijkstra.
The origin of this algorithm is unknown. It's said that at first Chinese coders used it in programming contests.
Its code looks like the combination of Dijkstra and BFS :

```
SPFA(v):
d[i] = inf for each vertex i
d[v] = 0
queue q
q.push(v)
while q is not empty
u = q.front()
q.pop()
for each i in adj[u]
if d[i] > d[u] + w(u,i)
then d[i] = d[u] + w(u,i)
if i is not in q
                    then q.push(i)
```

Time complexity : Unknown!.

9.5 MST

MST = Minimum Spanning Tree :) (if you don't know what it is, google it).
Best MST algorithms :

9.5.1 Kruskal

In this algorithm, first we sort the edges in ascending order of their weight in an array of edges.
Then in order of the sorted array, we add ech edge if and only if after adding it there won't be any cycle (check it using DSU).
Code :

```
Kruskal()
solve all edges in ascending order of their weight in an array e
ans = 0
for i = 1 to m
v = e.first
u = e.second
w = e.weight
if merge(v,u) // there will be no cycle
        then ans += w
```

Time complexity : $O(m \log m)$.

9.5.2 Prim

In this approach, we act like Dijkstra. We have a set of vertices S, in each step we add the nearest vertex to S, in S (distance of v from $S = \min_{u \in S}(weight(u, v))$ where weight(i, j) is the weight of the edge from i to j) .
So, pseudo code will be like this:

```
Prim()
S = new empty set
for i = 1 to n
d[i] = inf
while S.size() < n
x = inf
v = -1
for each i in V - S // V is the set of vertices
if x >= d[v]
then x = d[v], v = i
d[v] = 0
S.insert(v)
for each u in adj[v]
        do d[u] = min(d[u], w(v,u))
```

C++ code: One) $O(n^2)$

```
bool mark[MAXN];
void prim(){
fill(d, d + n, inf);
fill(mark, mark + n, false);
int x,v;
while(true){
x = inf;
v = -1;
for(int i = 0;i < n;i ++)
if(!mark[i] and x >= d[i])
x = d[i], v = i;
if(v == -1)
break;
d[v] = 0;
mark[v] = true;
for(auto p : adj[v]){ //adj[v][i] = pair(vertex, weight)
int u = p.first, w = p.second;
d[u] = min(d[u], w);
}
```

```
}
}
```

Two) $O(m \log n)$

```
void prim(){
fill(d, d + n, inf);
set<pair<int,int> > s;
for(int i = 0;i < n;i ++)
s.insert({d[i],i});
int v;
while(!s.empty()){
v = s.begin() -> second;
s.erase(s.begin());
for(auto p : adj[v]){
int u = p.first, w = p.second;
if(d[u] > w){
s.erase({d[u], u});
d[u] = w;
s.insert({d[u], u});
}
}
}
}
```

As Dijkstra you can use std :: priority_queue instead of std :: set.

9.6 Maximum Flow

I only wanna put the source code here (EdmondsKarp):

```
algorithm EdmondsKarp
    input:
        C[1..n, 1..n] (Capacity matrix)
        E[1..n, 1..?] (Neighbour lists)
        s             (Source)
        t             (Sink)
    output:
        f             (Value of maximum flow)
        F             (A matrix giving a legal flow with the maximum value)
    f := 0 (Initial flow is zero)
    F := array(1..n, 1..n) (Residual capacity from u to v is C[u,v] - F[u,v])
    forever
        m, P := BreadthFirstSearch(C, E, s, t, F)
        if m = 0
            break
        f := f + m
        (Backtrack search, and write flow)
        v := t
        while v   s
            u := P[v]
            F[u,v] := F[u,v] + m
            F[v,u] := F[v,u] - m
            v := u
    return (f, F)
```

```
algorithm BreadthFirstSearch
    input:
        C, E, s, t, F
    output:
        M[t]          (Capacity of path found)
        P             (Parent table)
    P := array(1..n)
    for u in 1..n
        P[u] := -1
    P[s] := -2 (make sure source is not rediscovered)
    M := array(1..n) (Capacity of found path to node)
    M[s] := ∞
    Q := queue()
    Q.offer(s)
    while Q.size() > 0
        u := Q.poll()
        for v in E[u]
            (If there is available capacity, and v is not seen before in search)
            if C[u,v] - F[u,v] > 0 and P[v] = -1
                P[v] := u
                M[v] := min(M[u], C[u,v] - F[u,v])
                if v   t
                    Q.offer(v)
                else
                    return M[t], P
    return 0, P
```

EdmondsKarp pseudo code using Adjacency nodes:

```
algorithm EdmondsKarp
    input:
        graph (Graph with list of Adjacency nodes with capacities,flow,reverse and destinations)
        s             (Source)
        t             (Sink)
    output:
        flow             (Value of maximum flow)
    flow := 0 (Initial flow to zero)
    q := array(1..n) (Initialize q to graph length)
    while true
        qt := 0             (Variable to iterate over all the corresponding edges for a source)
        q[qt++] := s    (initialize source array)
        pred := array(q.length)    (Initialize predecessor List with the graph length)
        for qh=0;qh < qt && pred[t] == null
            cur := q[qh]
            for (graph[cur]) (Iterate over list of Edges)
                 Edge[] e :=  graph[cur]  (Each edge should be associated with Capacity)
                 if pred[e.t] == null && e.cap > e.f
                    pred[e.t] := e
                    q[qt++] : = e.t
        if pred[t] == null
            break
```

```
        int df := MAX VALUE (Initialize to max integer value)
        for u = t; u != s; u = pred[u].s
            df := min(df, pred[u].cap - pred[u].f)
        for u = t; u != s; u = pred[u].s
            pred[u].f  := pred[u].f + df
            pEdge := array(PredEdge)
            pEdge := graph[pred[u].t]
            pEdge[pred[u].rev].f := pEdge[pred[u].rev].f - df;
        flow := flow + df
    return flow
```

9.6.1 Dinic's algorithm

Here is Dinic's algorithm as you wanted.
Input: A network G = ((V, E), c, s, t).
Output: A max s - t flow.
1.set f(e) = 0 for each e in E
2.Construct G_L from G_f of G. if dist(t) == inf, then stop and output f
3.Find a blocking flow fp in G_L
4.Augment flow f by fp and go back to step 2.
Time complexity : $O(mm\log n)$.
Theorem: Maximum flow = minimum cut.

9.6.2 Maximum Matching in bipartite graphs

Maximum matching in bipartite graphs is solvable also by maximum flow like below :
Add two vertices S, T to the graph, every edge from X to Y (graph parts) has capacity 1, add an edge from S with capacity 1 to every vertex in X, add an edge from every vertex in Y with capacity 1 to T.
Finally, answer = maximum matching from S to T .
But it can be done really easier using DFS.
As, you know, a bipartite matching is the maximum matching if and only if there is no augmenting path (read Introduction to graph theory).
The code below finds a augmenting path:

```
bool dfs(int v){// v is in X, it reaturns true if and only if there is an augmenting path starting from v
if(mark[v])
return false;
mark[v] = true;
for(auto &u : adj[v])
if(match[u] == -1 or dfs(match[u])) // match[i] = the vertex i is matched with in the current matching, initia
return match[v] = u, match[u] = v, true;
return false;
}
```

An easy way to solve the problem is:

```
for(int i = 0;i < n;i ++)if(match[i] == -1){
memset(mark, false, sizeof mark);
dfs(i);
}
```

But there is a faster way:

```
while(true){
memset(mark, false, sizeof mark);
bool fnd = false;
for(int i = 0;i < n;i ++) if(match[i] == -1 && !mark[i])
fnd |= dfs(i);
if(!fnd)
break;
}
```

In both cases, time complexity = $O(nm)$.

9.7 Trees

Trees are the most important graphs.
In the last lectures we talked about segment trees on trees and heavy-light decomposition.

9.7.1 Partial sum on trees

We can also use partial sum on trees.
Example: Having a rooted tree, each vertex has a value (initially 0), each query gives you numbers v and u (v is an ancestor of u) and asks you to increase the value of all vertices in the path from u to v by 1.
So, we have an array p, and for each query, we increase p[u] by 1 and decrease p[par[v]] by 1. The we run this (like a normal partial sum):

```
void dfs(int v){
for(auto u : adj[v])
if(u - par[v])
dfs(u), p[v] += p[u];
}
```

9.7.2 DSU on trees

We can use DSU on a rooted tree (not tree DSUs, DSUs like vectors).
For example, in each node, we have a vector, all nodes in its subtree (this can be used only for offline queries, because we may have to delete it for memory usage).
Here again we use DSU technique, we will have a vector V for every node. When we want to have V[v] we should merge the vectors of its children. I mean if its children are $u1, u2, ..., uk$ where V[u1].size() V[u2].size() ... V[uk].size(), we will put all elements from V[ui] for every 1 i < k, in V[k] and then, V[v] = V[uk].
Using this trick, time complexity will be .
C++ example (it's a little complicated) :

```
typedef vector<int> vi;
vi *V[MAXN];
void dfs(int v, int par = -1){
int mx = 0, chl = -1;
for(auto u : adj[v])if(par - u){
dfs(u,v);
if(mx < V[u]->size()){
mx = V[u]->size();
chl = u;
}
}
for(auto u : adj[v])if(par - u and chl - u){
```

```
for(auto a : *V[u])
V[chl]->push_back(a);
delete V[u];
}
if(chl + 1)
V[v] = V[chl];
else{
V[v] = new vi;
V[v]->push_back(v);
}
}
```

9.7.3 LCA

LCA of two vertices in a rooted tree, is their lowest common ancestor.
There are so many algorithms for this, I will discuss the important ones.
Each algorithm has complexities $< O(f(n)), O(g(n)) >$, it means that this algorithm's preprocess is O(f(n)) and answering a query is O(g(n)) .
In all algorithms, h[v] = height of vertex v. One) Brute force $< O(n), O(n) >$
The simplest approach. We go up enough to achieve the goal.
Preproccess :

```
void dfs(int v,int p = -1){
if(par + 1)
h[v] = h[p] + 1;
par[v] = p;
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}
```

Query :

```
int LCA(int v,int u){
if(v == u)
return v;
if(h[v] < h[u])
swap(v,u);
return LCA(par[v], u);
}
```

Two) SQRT decomposition
I talked about SQRT decomposition in the first lecture.
Here, we will cut the tree into $\sqrt{H}$ (H = height of the tree), starting from 0, k - th of them contains all vertices with h in interval $[k\sqrt{H}, (k+1)\sqrt{H}]$.
Also, for each vertex v in k - th piece, we store r[v] that is, its lowest ancestor in the piece number k - 1.
Preprocess:

```
void dfs(int v,int p = -1){
if(par + 1)
h[v] = h[p] + 1;
par[v] = p;
if(h[v] % SQRT == 0)
r[v] = p;
else
```

```
r[v] = r[p];
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}
```

Query:

```
int LCA(int v,int u){
if(v == u)
return v;
if(h[v] < h[u])
swap(v,u);
if(h[v] == h[u])
return (r[v] == r[u] ? LCA(par[v], par[u]) : LCA(r[v], r[u]));
if(h[v] - h[u] < SQRT)
return LCA(par[v], u);
return LCA(r[v], u);
}
```

Three) Sparse table <$O(n \log n)$, $O(1)$>
Let's introduce you an order of tree vertices, haas and I named it Euler order. It is like DFS order, but every time we enter a vertex, we write it's number down (even when we come from a child to this node in DFS).
Code for calculate this :

```
vector<int> euler;
void dfs(int v,int p = -1){
euler.push_back(v);
for(auto u : adj[v]) if(p - u)
dfs(u,v), euler.push_back(v);
}
```

If we have a vector<pair<int,int> > instead of this and push h[v], v in the vector, and the first time h[v], v is appeared is s[v] and s[v] < s[u] then LCA(v, u) = $(min_{i\,=\,s[v]}^{s[u]} euler[i])$.second.
For this propose we can use RMQ problem, and the best algorithm for that, is to use Sparse table.
Four) Something like Sparse table :) <$O(n \log n)$,$O(\log n)$>
This is the most useful and simple (among fast algorithms) algorithm.
For each vector v and number i, we store its 2^i-th ancestor. This can be done in $O(n \log n)$. Then, for each query, we find the lowest ancestors of them which are in the same height, but different (read the source code for understanding).
Preprocess:

```
int par[MAXN][MAXLOG]; // initially all -1
void dfs(int v,int p = -1){
par[v][0] = p;
if(p + 1)
h[v] = h[p] + 1;
for(int i = 1;i < MAXLOG;i ++)
if(par[v][i-1] + 1)
par[v][i] = par[par[v][i-1]][i-1];
for(auto u : adj[v]) if(p - u)
dfs(u,v);
}
```

Query:

```
int LCA(int v,int u){
if(h[v] < h[u])
swap(v,u);
for(int i = MAXLOG - 1;i >= 0;i --)
if(par[v][i] + 1 and h[par[v][i]] >= h[u])
v = par[v][i];
// now h[v] = h[u]
if(v == u)
return v;
for(int i = MAXLOG - 1;i >= 0;i --)
if(par[v][i] - par[u][i])
v = par[v][i], u = par[u][i];
return par[v][0];
}
```

Five) Advance RMQ $< \mathrm{O}(\mathrm{n}), \mathrm{O}(1) >$

In the third approach, we said that LCA can be solved by RMQ.

When you look at the vector euler you see that for each i that 1 i < euler.size(), |euler[i].first - euler[i + 1].firs

So, we can convert the euler from its size(we consider its size is n + 1) into a binary sequence of length n (if euler[i].first - euler[i + 1].first = 1 we put 1 otherwise 0).

So, we have to solve the problem on a binary sequence A .

To solve this restricted version of the problem we need to partition A into blocks of size . Let A'[i] be the minimum value for the i - th block in A and B[i] be the position of this minimum value in A. Both A and B are long. Now, we preprocess A' using the Sparse Table algorithm described in lecture 1. This will take time and space. After this preprocessing we can make queries that span over several blocks in O(1). It remains now to show how the in-block queries can be made. Note that the length of a block is , which is quite small. Also, note that A is a binary array. The total number of binary arrays of size l is . So, for each binary block of size l we need to lock up in a table P the value for RMQ between every pair of indices. This can be trivially computed in time and space. To index table P, preprocess the type of each block in A and store it in array . The block type is a binary number obtained by replacing - 1 with 0 and + 1 with 1 (as described above).

Now, to answer RMQA(i, j) we have two cases:

i and j are in the same block, so we use the value computed in P and T

i and j are in different blocks, so we compute three values: the minimum from i to the end of i's block using P and T, the minimum of all blocks between i's and j's block using precomputed queries on A' and the minimum from the beginning of j's block to j, again using T and P; finally return the position where the overall minimum is using the three values you just computed.

Six) Tarjan's algorithm O(na(n)) (a(n) is the inverse ackermann function)

Tarjan's algorithm is offline; that is, unlike other lowest common ancestor algorithms, it requires that all pairs of nodes for which the lowest common ancestor is desired must be specified in advance. The simplest version of the algorithm uses the union-find data structure, which unlike other lowest common ancestor data structures can take more than constant time per operation when the number of pairs of nodes is similar in magnitude to the number of nodes. A later refinement by Gabow & Tarjan (1983) speeds the algorithm up to linear time.

The pseudocode below determines the lowest common ancestor of each pair in P, given the root r of a tree in which the children of node n are in the set n.children. For this offline algorithm, the set P must be specified in advance. It uses the MakeSet, Find, and Union functions of a disjoint-set forest. MakeSet(u) removes u to a singleton set, Find(u) returns the standard representative of the set containing u, and Union(u, v) merges the set containing u with the set containing v. TarjanOLCA(r) is first called on the root r.

```
function TarjanOLCA(u)
    MakeSet(u);
    u.ancestor := u;
```

```
    for each v in u.children do
        TarjanOLCA(v);
        Union(u,v);
        Find(u).ancestor := u;
    u.colour := black;
    for each v such that {u,v} in P do
        if v.colour == black
            print "Tarjan's Lowest Common Ancestor of " + u +
                  " and " + v + " is " + Find(v).ancestor + ".";
```

Each node is initially white, and is colored black after it and all its children have been visited. The lowest common ancestor of the pair u, v is available as Find(v).ancestor immediately (and only immediately) after u is colored black, provided v is already black. Otherwise, it will be available later as Find(u).ancestor, immediately after v is colored black.

```
function MakeSet(x)
    x.parent := x
    x.rank   := 0

function Union(x, y)
    xRoot := Find(x)
    yRoot := Find(y)
    if xRoot.rank > yRoot.rank
        yRoot.parent := xRoot
    else if xRoot.rank < yRoot.rank
        xRoot.parent := yRoot
    else if xRoot != yRoot
        yRoot.parent := xRoot
        xRoot.rank := xRoot.rank + 1

function Find(x)
    if x.parent == x
       return x
    else
       x.parent := Find(x.parent)
       return x.parent
```