Template v2.0

poore

September 23, 2016

目录

1 Math

1.1 CRT

```
// 中国剩余定理
// x ≡ a[i] (mod m[i])
// m[i] is coprime
LL CRT(LL *a, LL *m, int n)
{
  LL M = 1, Mi, x0, y0, d, ret = 0;
  for(int i = 0; i < n; i++)
    M *= m[i];
  for(int i = 0; i < n; i++)
  {
    Mi = M/m[i];
    EXT_GCD(Mi, m[i], d, x0, y0);
    LL tmp = slow_mul(Mi, x0, M);
    tmp = slow_mul(tmp, a[i], M);
    ret = (ret + tmp) % M;
    //ret = (ret+Mi*x0*a[i]) % M;
  }
  if(ret < 0)
    ret += M;
  return ret;
}
```

1.2 Gauss

```
/*
 * a 保存系数矩阵和每个方程的值
 * v 储存方程组的解
 */
void gauss(int n, double a[SZ][SZ], double v[SZ]) {
  int k=1;
  for (int i=1;i<=n;i++) {
    int p=0;
    for (int j=k;j<=n;j++) if (fabs(a[j][i])>eps) {
      p=j;
      break;
    }
    if (!p) continue;
    for (int l=1;l<=n+1;l++) swap(a[p][l],a[k][l]);
    for (int j=k+1;j<=n;j++) {
      double rate=a[j][i]/a[k][i];
      for (int l=1;l<=n+1;l++)
        a[j][l]-=a[k][l]*rate;
    }
    k++;
  }
  for (int i=n;i;i--) {
    v[i]=a[i][n+1];
    for (int j=i+1;j<=n;j++)
      v[i]-=v[j]*a[i][j];
    v[i]/=a[i][i];
  }
}
```

1.3 Mo's

```
/*
 * offlien Algorithm
 */
int rtn = sqrt(n);
bool operator < (const Query &a, const Query &b) {
  if (a.lp == b.lp) return a.r < b.r;
  else return a.lp < b.lp;
}

void update(int &l, int &r, int L, int R) {
  while(r < R) {
    r++;
    bita.add(a[r], 1);
    bitb.add(b[r], 1);
  }
  while(r > R) {
    bita.add(a[r], -1);
    bitb.add(b[r], -1);
    r--;
  }
  while(l < L) {
    bita.add(a[l], -1);
    bitb.add(b[l], -1);
    l++;
  }
  while(l > L) {
    l--;
    bita.add(a[l], 1);
    bitb.add(b[l], 1);
  }
}
```

1.4 ext_gcd1

```
// 扩展欧几里得
// a , b 任意
void EXT_GCD(LL a, LL b, LL &d, LL &x, LL &y)
{
  if(!b) {d = a, x = 1, y = 0;}
  else {EXT_GCD(b, a % b, d, y, x), y -= x * (a / b);}
}
```

1.5 ext_gcd2

```
// 扩展欧几里得
// a >= 0, b > 0
LL ext_gcd(LL a, LL b, LL& x, LL& y)
{
  LL x1=0LL, y1=1LL, x0=1LL, y0=0LL;
  LL r = (a%b + b) % b;
  LL q = (a-r) / b;
  x = 0LL,y = 1LL;
  while(r)
  {
    x=x0-q*x1;y=y0-q*y1;
    x0=x1;y0=y1;
    x1=x;y1=y;
    a=b;b=r;
    r=a%b;
    q=(a-r)/b;
  }
  return b;
}
```

1.6 gcd

```
// 非递归
LL gcd(LL a,LL b)
{
  if (a < b) swap(a, b);
  LL Rem;
  while(b > 0)
  {
    Rem = a % b;
    a = b;
    b = Rem;
  }
  return a;
}
```

1.7 gcd_re

```
// 递归求gcd
LL GCD(LL a, LL b)
{
  if(a < b) swap(a, b);
  LL r = a % b;
  if(r == 0) return b;
  return GCD(b, r);
}
```

1.8 josephes

```
#include <iostream>
using namespace std;
//編號從0開始，也就是說如果編號從1開始結果要加1
int josephus(int n, int k) { //非遞回版本
  int s = 0;
  for (int i = 2; i <= n; i++)
    s = (s + k) % i;
  return s;
}
int josephus_recursion(int n, int k) { //遞回版本
  return n > 1 ? (josephus_recursion(n - 1, k) + k) % n : 0;
}
int main() {
  for (int i = 1; i <= 100; i++)
    cout << i << ' ' << josephus(i, 5) << endl;
  return 0;
}
```

1.9 linear_mod_equation

```
// 线性方程求解
// ax = b (mod n)
vector<LL> linear_mod_equation(LL a, LL b, LL n)
{
  LL x, y, d;
  vector<LL> sol;
  sol.clear();
  EXT_GCD(a, n, d, x, y);
  if( b%d ) d = 0;
  else
  {
    sol.push_back(x * (b/d) % n);
    for (int i = 1; i < d; i++)
      sol.push_back((sol[i-1] + n/d + n) % n);
  }
  return sol;
}
```

1.10 matrix

```
//poj 3233
//cal (A + A^2 + A^3 + ... + A^K) % Mod
#define maxn 30
typedef long long LL;
struct Matrix{
  LL m[maxn][maxn];
  Matrix(){memset(m, 0, sizeof(m));}
};
typedef Matrix matrix;
LL Mod;
int n;
matrix operator* (matrix A, matrix B) {
  matrix C;
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++) {
      C.m[i][j] = 0LL;
      for(int k = 0; k < n; k++)
        C.m[i][j] += A.m[i][k]*B.m[k][j];
      C.m[i][j] %= Mod;
    }
  return C;
}
matrix operator+ (matrix A, matrix B) {
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      A.m[i][j] = (A.m[i][j] + B.m[i][j]) % Mod;
  return A;
}
matrix operator% (matrix A, LL m) {
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      A.m[i][j] %= m;
  return A;
}
matrix matrix_pow(int k, matrix M) {
  if(k == 1) return M;
  matrix ans;
  memset(ans.m, 0, sizeof(ans.m));
  for(int i = 0; i < n; i++)
    ans.m[i][i] = 1LL;
  while(k) {
    if(k&1) {
      ans = ans * M;
      k--;
    }
    else {
      k /= 2;
      M = M * M;
    }
  }
  return ans;
}
matrix sum(matrix ma, int k) {
  matrix ret;
  if(k == 1) return ma;
  if(k&1) {
    matrix tmp = sum(ma, k/2) % Mod, tmp1 = matrix_pow(k
    /2+1, ma) % Mod;
    ret = (tmp + tmp1 + tmp * tmp1) % Mod;
  }
  else {
    matrix tmp = sum(ma, k/2) % Mod, tmp1 = matrix_pow(k
    /2, ma) % Mod;
    ret = (tmp + tmp * tmp1) % Mod;
  }
  return ret;
}
int main() {
  int k;
  matrix A;
  scanf("%d%d%lld", &n, &k, &Mod);
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      scanf("%lld", &A.m[i][j]);
  A = sum(A, k);
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
    {
      printf("%lld%c", A.m[i][j],(j == n-1)? '\n':' ');
    }
  return 0;
}
```

1.11 matrix_pow

```
struct matrix {
  ll m[SZ][SZ];
  matrix() {
    memset(m, 0, sizeof(m));
  }
  matrix(int x) {
    memset(m, 0, sizeof(m));
    for (int i = 0; i < SZ; i++) m[i][i] = x;
  }
  void clear() {
    memset(m, 0, sizeof(m));
  }
  friend matrix operator *(matrix a, matrix b) {
    matrix c;
    for (int k = 0; k < SZ; k++)
      for (int i = 0; i < SZ; i++) if (a.m[i][k])
        for (int j = 0; j < SZ; j++)
          (c.m[i][j]+=a.m[i][k]*b.m[k][j]%oo)%=oo;
    return c;
  }
  friend matrix operator ^(matrix e, ll k) {
    matrix tmp = matrix(1);
    while(k) {
      if (k&1) tmp = tmp*e;
      k>>=1;
      e=e*e;
    }
    return tmp;
  }
  void show() {
    printf("===========\n");
    for (int i = 0; i < SZ; i++)
      for (int j = 0; j < SZ; j++)
        printf("%I64d%c", m[i][j], j == SZ-1? '\n':' ');
    printf("===========\n");
  }
};
```

1.12 matrix_rotate

```
struct Matrix {
  double m[3][3];
  Matrix(){
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        m[i][j]=0.0;
  }
  Matrix(double a) {
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
      m[i][j]=0.0;
    for (int i = 0; i < 3; i++)
      m[i][i] = a;
  }
  Matrix operator * (Matrix M) {
    Matrix ret;
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        for (int k = 0; k < 3; k++)
          ret.m[i][j] += m[i][k]*M.m[k][j];
    return ret;
  }
  void init(double x, double y, double r) {
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        m[i][j]=0.0;
    m[0][0]=cos(r);
    m[0][1]=sin(r);
    m[1][0]=-sin(r);
    m[1][1]=cos(r);
    m[2][0]=x*(1-cos(r))+y*sin(r);
    m[2][1]=y*(1-cos(r))-x*sin(r);
    m[2][2]=1.0;
  }
}t;
int main() {
  int T;
  scanf("%d", &T);
  while (T--) {
    scanf("%d", &n);
    Matrix ans = Matrix(1);
    double x, y, r;
    for (int i = 1; i <= n; i++) {
      scanf("%lf%lf%lf", &x,&y,&r);
      t.init(x,y,r);
      ans = ans*t;
    }
    double theta = atan2(ans.m[0][1], ans.m[0][0]);
    if (dcmp(theta) < 0) theta += pi*2.0;
    double a = 1-ans.m[0][0];
    double b = ans.m[0][1];
    double A = ans.m[2][0];
    double B = ans.m[2][1];
    y = (b*A+a*B) / (a*a+b*b);
    x = (a*A-b*B) / (a*a+b*b);
    printf("%.8f %.8f %.8f\n", x, y, theta);
  }
  return 0;
}
```

1.13 mega_mod

```
// 解 n 个一元线性同于方程组
// x ≡ r (mod a)
// 求x
LL mega_mod(int n)
{
  LL a1, a2, r1, r2, d, c, x, y, x0,s;
  bool flag = true;
  scanf("%lld%lld", &a1, &r1);
  for(int i = 1; i < n; i++)
  {
    scanf("%lld%lld", &a2, &r2);
    if(!flag) continue;
    c = r2 - r1;
    EXT_GCD(a1, a2, d, x, y);
    if(c%d!=0)
    {
      flag = false;
      continue;
    }
    x0 = x*c/d;
    s = a2/d;
    x0 = (x0%s+s)%s;
    r1=r1+x0*a1;
    a1=a1*a2/d;
  }
  if(flag) return r1;
  else return -1LL;
}
```

1.14 mersenneprime

```
// 判断Mp = 2^p-1 是否为梅森素数
bool lucas_lehmer(int p)
{
  if(p == 2) return true;
  LL m = (1LL<<p)-1LL, tmp = 4LL;
  for(int i = 0; i < p-2; i++)
  {
    tmp = (slow_mul(tmp, tmp, m) - 2 + m) % m;
  }
  if(tmp == 0LL) return true;
  return false;
}
```

1.15 miller_rabin

```
LL witness(LL a,LL b,LL c)
{
  if(b==0)return 1;
  LL x,y,t=0;
  while((b&1)==0)
    b>>=1,t++;
  y=x=pow_mod(a,b,c);
  // 二次探测
  while(t--)
  {
    y=slow_mul(x,x,c);
    if(y==1 && x!=1 && x!=c-1)
      return false;
    x=y;
  }
  return y==1;
}
bool miller_rabin(LL n)
// ..质数为true, 非质数为false..
{
  if(n==2)return true;
  if(n<2 || (n&1)==0)return false;
  for(int i=0;i<3;i++)
    if(witness(rand()%(n-2)+2,n-1,n)!=1)
      return false;
  return true;
}
```

1.16 phi

```
// 欧拉函数预处理
int phi[maxn];
void getpthi(int n)
{
  memset(phi, 0, sizeof(phi));
  phi[1] = 1;
  for(int i = 2; i <= n; i++)if(!phi[i])
  {
    for(int j = i; j <= n; j+=i)
    {
      if(!phi[j])
        phi[j] = j;
      phi[j] = phi[j]/i*(i-1);
    }
  }
}
```

1.17 pollard_rho

```
// ..随机返回一个 n 的约数..
LL ans = INF;
LL pollard_rho(LL n,LL c)
{
  if(n%2==0)return 2;
  LL i=1,k=2,x=rand()%n,y=x,d;
  while(1){
    i++;
    x=(slow_mul(x,x,n)+c)%n;
    d=gcd(y-x,n);
    if(d==n)return n;
    if(d!=n && d>1)return d;
    if(i==k) y=x,k<<=1;
  }
}
void calc(LL n,LL c=240)
// 寻找最小的约数..
```

```
{
  if(n==1)return;
  if(miller_rabin(n)){
    ans=min(ans,n);
    return;
  }
  LL k=n;
  while(k==n)k=pollard_rho(n,c--);
  calc(k,c),calc(n/k,c);
}
```

1.18 prime_seive

```
// 筛法求质数
// prime[] 储存质数。1-based index;
const int maxn = 100020;
bool isprime[maxn];
LL prime[maxn];
int doprime(LL N)
{
  int nprime = 0;
  memset(isprime, true, sizeof(isprime));
  isprime[1] = false;
  for(LL i = 2; i <= N; i++)
  {
    if(isprime[i])
    {
      prime[++nprime] = i;
      for(LL j = i*i; j <= N; j+=i)
        isprime[j] = false;
    }
  }
  return nprime;
}
```

1.19 quick_pow

```
// 快速幂
// (a^b) % p
LL pow_mod(LL a, LL b, LL p)
{
  LL ret = 1;
  while(b) {
    if(b & 1) ret = (ret*a)%p;
    a = (a*a)%p;
    b >>= 1;
  }
  return ret%p;
}
```

1.20 reverse_extgcd

```
// 用扩展欧几里得求逆元
// 要求 a, c 互质
// 如果没有逆元返回 -1
LL inv(LL a, LL c)
{
  LL d, x, y;
  EXT_GCD(a, c, d, x, y);
  return d == 1 ? (x + c) % c : -1;
}
```

1.21 reverse_rer

```
// 递归求逆元
// p, x 互质
LL inv(LL x, LL m)
{
  if (x == 1) return x;
  return inv(m % x, m)*(m - m / x) % m;
}
```

1.22 slow_mul

```
// 慢速乘
// a * b % p;
LL slow_mul(LL a, LL b, LL p)
{
  if (b < 0 && a >= 0) swap(a, b);
  else if (a < 0 && b < 0) {a = -a; b = -b;}
  LL ret = 0;
  while(b) {
    if(b & 1) ret = (ret + a) % p;
    a = (a + a) % p;
    b >>= 1;
  }
  return ret % p;
}
```

1.23 split_int

```
// 把整数 n 拆分成几个数相加的形式， 问有多少种拆分方法
int dp[maxn];
void splitint()
{
  memset(dp, 0, sizeof(dp));
  dp[0]=1;
  for(int i = 1; i <= maxn; i++)
  {
    for(int j = 1, r = 1; i - (3*j*j-j)/2 >= 0; j++, r *=-1)
    {
      dp[i] += dp[i-(3*j*j-j)/2]*r;
      dp[i] %= MOD;
      dp[i] = (dp[i]+MOD)%MOD;
      if(i-(3*j*j+j)/2 >= 0)
      {
        dp[i] += dp[i-(3*j*j+j)/2] *r;
        dp[i] %= MOD;
        dp[i] = (dp[i] + MOD)%MOD;
      }
    }
  }
}
```

1.24 stirling

```
// Stirling N的阶乘的长度
const double PI=3.1415926;
int main()
{
  int t,n,a;
  while(scanf("%d",&n)!=EOF)
  {
    a=(int)((0.5*log(2*PI*n)+n*log(n)-n)/log(10));
    printf("%d\n",a+1);
  }
  return 0;
}
```

1.25 zzz

```
/*
Something Tasteless

1.素数个数估算
  设PI(x) 为小于 x 的素数的个数
  当 x 足够大时，PI(x) = x/lnx;
2.n! 的素因子分解中的素数 p 的次数 为
  [n/p] + [n/(p^2)] + [n/(p^3)] + ... +


*/
```

2 strings

2.1 Acauto

```
struct ACAuto {
  deque<int>q;
  struct Trie{
    int fail, next[26], cnt;
  }trie[maxn];
  int tot;
  void insert(char *s) {
    int cur = 1;
    for(int i = 0; s[i]; i++) {
      if(!trie[cur].next[s[i]-'a'])
        trie[cur].next[s[i]-'a'] = ++tot;
      cur = trie[cur].next[s[i]-'a'];
    }
    trie[cur].cnt++;
  }
  void build_acauto() {
    q.clear();
    q.push_back(1);
    trie[1].fail = 1;
    while(!q.empty()) {
      int cur = q.front();
      q.pop_front();
      for(int i = 0;i<26;i++)if(trie[cur].next[i]) {
        int next = trie[cur].next[i];
        if(cur == 1) {
          trie[next].fail = 1;
        }
        else {
          int tmp = trie[cur].fail;
          while(tmp != 1 && trie[tmp].next[i] == 0) tmp = trie[tmp].fail;
          if(trie[tmp].next[i]) {
            trie[next].fail = trie[tmp].next[i];
          }
          else {
            trie[next].fail = 1;
          }
        }
        q.push_back(next);
      }
    }
  }
  int query(char *s) {
    int ans = 0, cur = 1, tmp;
    for(int i = 0; s[i]; i++) {
      if(trie[cur].next[s[i]–'a']) cur = trie[cur].next[s[i]–'a'];
      else {
        while(cur != 1 && trie[cur].next[s[i]–'a'] == 0) cur = trie[cur].fail;
        if(trie[cur].next[s[i]–'a']) cur = trie[cur].next[s[i]–'a'];
      }
      tmp = cur;
      while(tmp != 1 && trie[tmp].cnt != –1) {
        ans += trie[tmp].cnt;
        trie[tmp].cnt = –1;
        tmp = trie[tmp].fail;
      }
    }
    return ans;
  }
  void clear() {
    memset(trie, 0, sizeof(trie));
    tot = 1;
  }
}acat;
int main() {
  // printf("%d\n", sizeof(trie) / 1024);
  acat.insert(str);
  acat.build_acauto();
  acat.query();
}
```

2.2 Hash

```
/*
 * getH(s, l, H)
 * 求从位置s开始长度为l的字符串的hash值
 * O(n) 预处理, O(1) 查询
 */
const ull xxx = 131;
ull H[maxn], xl[maxn];
void prepre() {
  xl[0] = 1;
  for (int i = 1; i < maxn; i++)
    xl[i] = xxx * xl[i–1];
}
void pre(char *str, ull *H) {
  memset(H, 0, sizeof(H));
  int len = strlen(str);
  H[len] = 0;
  for (int i = len–1; i >= 0; i––) {
    H[i] = H[i+1]*xxx + str[i];
  }
}
ull getH(int s, int l, ull *H) {
  return H[s] – H[s+l] * xl[l];
}

/*
 *  BKDR Hash Function
 *  也可以将返回类型设为ull, 免去取模运算(&)
 */
unsigned int BKDRHash(char *str) {
  unsigned int seed = 131; // 31 131 1313 13131 131313 etc..
  unsigned int hash = 0;
  while (*str) {
    hash = hash * seed + (*str++);
  }
  return (hash & 0x7FFFFFFF);
}
```

2.3 KMP

```
const int maxn = 1000020;
char src[maxn],substring[maxn];
int nxt[maxn];
void get_nxt(char* substring) {
  int substring_len = strlen(substring);
  memset(nxt, 0, sizeof(nxt));
  nxt[0] = –1;
  int j = –1;
  for(int i = 1; i < substring_len; i++)
  {
    while(j > –1 && substring[i] != substring[j + 1])
      j = nxt[j];
    if(substring[j+1] == substring[i])
      j = j + 1;
    nxt[i] = j;
  }
}
//process src & substring to get the position
int kmp(char* src, char* substring) {
  int j = –1; int ans = 0;
  int substring_len = strlen(substring);
  int src_len = strlen(src);
  for(int i = 0; i < src_len; i++) {
    while(j > –1 && src[i] != substring[j + 1])
      j = nxt[j];
    if(src[i] == substring[j + 1])
      j++;
    if(j == substring_len –1) {
      ans ++;
          printf("From position %d to position %d\n", i + 2 – substring_len, i+1);
      j = nxt[j];
    }
  }
  return ans;
}
```

2.4 SuffixArray

```
/*
 * 后缀数组
 * dc3 的时间是 da的 3/4
 * da/dc3(int *r, int *sa, int n, int m);
 * n 为字符串的长度+1, m 为 r中值的范围
 *
 * calheight(int *r, int *sa, int n)
 * n 为字符串的长度
 *
 * r 数组为处理的串的int值
 * sa[i] 表示排名第i的后缀的起始下标
 * heigh[i] 表示排名第i的后缀于排名第 i–1 的后缀的最长公共前缀
 * Rank[i] 表示以i开始的后缀的排名
 *
 */
const int maxn = 1000020;
const int oo = 0x3f3f3f3f;
int tta[maxn],ttb[maxn],wv[maxn],tts[maxn];
int Rank[maxn],height[maxn];
int cmp(int *r,int a,int b,int l)
{
  return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m)
```

```
{
  int i,j,p,*x=tta,*y=ttb,*t;
  for(i=0;i<m;i++) tts[i]=0;
  for(i=0;i<n;i++) tts[x[i]=r[i]]++;
  for(i=1;i<m;i++) tts[i]+=tts[i-1];
  for(i=n-1;i>=0;i--) sa[--tts[x[i]]]=i;
  for(j=1,p=1;p<n;j*=2,m=p)
  {
    for(p=0,i=n-j;i<n;i++) y[p++]=i;
    for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
    for(i=0;i<n;i++) wv[i]=x[y[i]];
    for(i=0;i<m;i++) tts[i]=0;
    for(i=0;i<n;i++) tts[wv[i]]++;
    for(i=1;i<m;i++) tts[i]+=tts[i-1];
    for(i=n-1;i>=0;i--) sa[--tts[wv[i]]]=y[i];
    for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
    x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
  }
  return;
}

// dc3 算法
#define N 1000005
#define MOD 1000000007
#define F(x) ((x)/3+((x)%3==1?0:tb))
#define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
int wsf[N],wa[N],wb[N],wv[N],sa[N],rank[N],height[N],f[N];
int s[N],a[N];
char str[N],str1[N],str2[N];
//sa:字典序中排第i位的起始位置在str中第sa[i]
//rank:就是str第i个位置的后缀是在字典序排第几
//height：字典序排i和i-1的后缀的最长公共前缀
int c0(int *r,int a,int b)
{
  return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];
}
int c12(int k,int *r,int a,int b)
{
  if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
  else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];
}
void sort(int *r,int *a,int *b,int n,int m)
{
  int i;
  for(i=0; i<n; i++) wv[i]=r[a[i]];
  for(i=0; i<m; i++) wsf[i]=0;
  for(i=0; i<n; i++) wsf[wv[i]]++;
  for(i=1; i<m; i++) wsf[i]+=wsf[i-1];
  for(i=n-1; i>=0; i--) b[--wsf[wv[i]]]=a[i];
  return;
}
void dc3(int *r,int *sa,int n,int m)
{
  int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
  r[n]=r[n+1]=0;
  for(i=0; i<n; i++) if(i%3!=0) wa[tbc++]=i;
  sort(r+2,wa,wb,tbc,m);
  sort(r+1,wb,wa,tbc,m);
  sort(r,wa,wb,tbc,m);
  for(p=1,rn[F(wb[0])]=0,i=1; i<tbc; i++)
    rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
  if(p<tbc) dc3(rn,san,tbc,p);
  else for(i=0; i<tbc; i++) san[rn[i]]=i;
  for(i=0; i<tbc; i++) if(san[i]<tb) wb[ta++]=san[i]*3;
  if(n%3==1) wb[ta++]=n-1;
  sort(r,wb,wa,ta,m);
  for(i=0; i<tbc; i++) wv[wb[i]=G(san[i])]=i;
  for(i=0,j=0,p=0; i<ta && j<tbc; p++)
      sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
  for(; i<ta; p++) sa[p]=wa[i++];
  for(; j<tbc; p++) sa[p]=wb[j++];
  return;
}

void calheight(int *r,int *sa,int n)
{
  int i,j,k=0;
  for(i=1;i<=n;i++) Rank[sa[i]]=i;
  for(i=0;i<n;height[Rank[i++]]=k)
  for(k?k--:0,j=sa[Rank[i]-1];r[i+k]==r[j+k];k++);
  return;
}

// 查询[s, e](排名)区间中最小的height值
// 即s-1 与 e 的最长公共前缀的长度
int query(int s, int e)
{
  int k = log2(e - s + 1);
  return min(low[s][k], low[e - (1 << k) + 1][k]);
}

// 查询s1, s2开始的后缀的最长公共前缀的长度
// 注意s1 != s2
int callen(int s1, int s2)
{
  int l = Rank[s1], r = Rank[s2];
  if (l > r) swap(l, r);
  return query(l+1, r);
}
void rmqinit()
{
  int mxbit = 20;
  for (int i = 0; i <= n; i++)
    low[i][0] = height[i];
  for (int j = 1; j <= mxbit; j++)
    for (int i = 1; i <= n; i++) if (i + (1<<j) - 1 <= n)
      low[i][j] = min(low[i][j-1], low[i+(1<<(j-1))][j-1])
      ;
}

int main()
{
  s[len] = 0;
  da(s, sa, len+1, 256);
  calheight(s, sa, len);
  return 0;
}
```

2.5 extKMP

```
/*
 * getExtNext(char *s, char *t, int *ex)
 * ex 保存s[i...n] 和 t[0...m] 的最长公共前缀的长度
 * next[i] = 保存 t[0...m] 和 t[i...m] 的最长公共前缀的长
 度
 */
#define next Nnext
int next[maxn], ex[maxn];
void getExtNext(const char *t, int *next)
{
  int lt = strlen(t);
  next[0] = lt;
  for (int i = 1, j = -1, a, p; i < lt; i++, j--)
    if (j < 0 || i + next[i-a] >= p)
    {
      if (j < 0) j = 0, p = i;
      while(p < lt && t[j] == t[p]) j++, p++;
      next[i] = j, a = i;
    }
    else next[i] = next[i-a];
}
void getExtend(const char *s, const char *t, int *extend)
{
  int ls = strlen(s), lt = strlen(t);
  getExtNext(t, next);
  // a : 最长匹配的下标
  // p : s中最长匹配的位置+1
  for (int i = 0, j = -1, a, p; i < ls; i++, j--)
    if (j < 0 || i + next[i-a] >= p)
    {
      if (j < 0) j = 0, p = i;
      while(p < ls && j < lt && s[p] == t[j]) j++, p++;
      extend[i] = j, a = i;
    }
    else
      extend[i] = next[i-a];
}
```

2.6 manacher

```
/*
 *  求最长回文字串
 *  O(n);
 *  Ma 为增加了分隔符之后的字符串
 *  Mp[i] 表示以i为中心的回文串的半径(包括自身)
 *  mx, 最长长度，id下标
 */
char Ma[maxn*2];
int Mp[maxn*2];
void manacher(char *s, int *Mp) {
  int l = 0, len = strlen(s);
  Ma[l++] = '$';
  Ma[l++] = '#';
  for(int i = 0; i < len; i++) {
    Ma[l++] = s[i];
    Ma[l++] = '#';
  }
  Ma[l] = 0;
  int mx = 0, id = 0;
  for(int i = 0; i < l; i++) {
    Mp[i] = mx>i? min(Mp[2*id-i],mx-i):1;
    while(Ma[i+Mp[i]] == Ma[i-Mp[i]]) Mp[i]++;
    if(i+Mp[i]>mx) {
      mx = i + Mp[i];
      id = i;
    }
  }
}
```

2.7 minSTR

```
/*
 * minstr/maxstr(char *s)
 * 返回一个pair<int,int>
 * first 表示最小/最大表示的起始下表
 * second 表示出现的次数
 */
typedef pair<int,int> pii;
pii minstr(char *s) {
  int l = strlen(s);
  for (int i = 0; i < l; i++) s[i+l] = s[i];
  s[2*l] = 0;
  int i = 0, j = 1;
  while(i < l && j < l) {
    int k = 0;
    while(s[i+k] == s[j+k] && k < l) k++;
    if (k == l)
      return pii(min(i, j), l/(abs(i-j)));
    else if(s[i+k] > s[j+k]) i = max(i+k+1, j+1);
    else j = max(j+k+1, i+1);
  }
  return pii(min(i, j), 1);
}
pii maxstr(char *s) {
  int l = strlen(s);
  for (int i = 0; i < l; i++) s[i+l] = s[i];
  s[2*l] = 0;
  int i = 0, j = 1;
  while(i < l && j < l) {
    int k = 0;
    while(s[i+k] == s[j+k] && k < l) k++;
    if (k == l)
      return pii(min(i, j), l/(abs(i-j)));
    else if(s[i+k] < s[j+k]) i = max(i+k+1, j+1);
    else j = max(j+k+1, i+1);
  }
  return pii(min(i, j), 1);
}
```

3 CG

3.1 4_points_1_plane

```
struct  Point3 {
  double x, y, z;
  Point3  operator - ( Point3 & p ) {
    Point3  ans;
    ans.x = this->x - p.x;
    ans.y = this->y - p.y;
    ans.z = this->z - p.z;
    return ans;
  }
};
Point3 operator * ( const Point3 & a, const Point3 & b ) {
  Point3  ans;
  ans.x = a.y * b.z - a.z * b.y;
  ans.y = a.z * b.x - a.x * b.z;
  ans.z = a.x * b.y - a.y * b.x;
  return ans;
}
double  dot( const Point3 & a, const Point3 & b ) {
  return a.x * b.x + a.y * b.y + a.z * b.z;
}
int main() {
  Point3  p[4];
  int T;
  cin >> T;
  while(T--)
  {
    for( int i = 0; i < 4; ++i ) scanf( "%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z );
    puts( dot( p[3] - p[0], (p[2] - p[0])*(p[1] - p[0])) == 0.0 ? "Yes" : "No" );
  }
  return 0;
}
```

3.2 CG

```
const double PI = acos(-1.0);
const double MAXN = 1000000.0;
struct Point{
  double x,y;
  Point(double x=0,double y=0):x(x),y(y){}
};
typedef Point Vec;
//向量+向量 = 向量，点+向量 = 点
Vec operator +(Vec A,Vec B){return Vec(A.x+B.x,A.y+B.y);}
//点-点 = 向量
Vec operator -(Point A,Point B){return Vec(A.x-B.x,A.y-B.y);}
//向量*数 = 向量
Vec operator *(Vec A,double p){return Vec(A.x*p,A.y*p);}
//向量/数 = 向量
Vec operator /(Vec A,double p){return Vec(A.x/p,A.y/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) == 0;
}
double ang(Vec v){return atan2(v.y,v.x);}
/*========直线基本定义========*/
struct Line{
  Point P;
  Vec v;
  double ang;
  Line(){}
  Line(Point P,Vec v):P(P),v(v){ang  = atan2(v.y,v.x);}
  bool operator < (const Line& L) const {
    return ang < L.ang;
  }
  Point point(double a){
    return P+v*a;
  }
};
/*========圆的基本定义========*/
struct Circle{
  Point c;
  double r;
  Circle(Point c,double r):c(c),r(r){}
  Point point(double a){
    return Point(c.x+cos(a)*r,c.y + sin(a)*r);
  }
};
/*==========以上为基本定义============*/


double Dot(Vec A,Vec B){return A.x*B.x+A.y*B.y;}
double Length(Vec A){ return sqrt(Dot(A,A));}
//只能就算较小的那个角！
double Angle(Vec A,Vec B){return acos(Dot(A,B)/Length(A)/Length(B));}
/*==========用点积算向量长度和两个向量夹角============*/

```

```cpp

double Cross(Vec A,Vec B){return A.x*B.y - A.y*B.x;}
//ABC的三角形有向面积的两倍
double Area2(Point A,Point B,Point C){return Cross(B-A,C-A);}
//rad是弧度 逆时针旋转 //注意：是绕原点旋转，否则要加上坐标
Vec Rotate(Vec A,double rad){
  return Vec(A.x*cos(rad)-A.y*sin(rad),A.x*sin(rad)+A.y*cos(rad));
}
//逆时针旋转90°的单位法向量
Vec Normal(Vec A){
  double L = Length(A);
  return Vec(-A.y/L,A.x/L);
}
/*=====以上为叉积的基本运算=====*/


//P+tv和Q+tw两条直线的交点,确保有唯一交点
Point GetlineIntersection(Line a,Line b){
  Point P = a.P,Q = b.P;
  Vec v = a.v,w = b.v;
  Vec u = P-Q;
  double t = Cross(w,u)/Cross(v,w);
  return a.point(t);
}
//点到直线的距离
double DistanceToLine(Point P,Line a){
  Point A = a.P,B = a.P + a.v;
  Vec v1 = B-A,v2 = P-A;
  return fabs(Cross(v1,v2)/Length(v1));//不取绝对值那么得到的是有向距离
}
//点到线段的距离
double DistanceToSegment(Point P,Point A,Point B){
  if(A == B)return Length(P-A);
  Vec 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,Line a){
  Point A = a.P,B = a.P + a.v;
  Vec v = B - A;
  return A + v*(Dot(v,P-A) / Dot(v,v));
}
//判断两条线段是否相交 此处必须为规范相交
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;
}
//如果允许端点相交，则用以下代码，判断一个点是否在一条线段上
bool OnSegment(Point p,Point a1,Point a2){   //不对，不允许端点相交
  if (p == a1 || p == a2) return 1;
  return dcmp(Cross(a1-p,a2-p)) == 0 && dcmp(Dot(a1-p,a2-p))<0;
}
/*=========以上为点和直线，直线和直线关系的内容========*/
//判断点和多边形位置关系
int isPointInPolygon(Point p, const vector<Point>& poly){
  int w = 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)w++;
    if(k < 0 && d2 <= 0 && d1 > 0)w--;
  }
  if(w != 0)return 1;
  return 0;
}
//多边形有向面积
double PolygonArea(vector<Point> p){
  int n = p.size();
  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;
}
//多边形周长
double PolygonZhouc(vector<Point> p){
  int n = p.size();
  if (!n) return 0.0;
  double ans = 0;
  for(int i=0;i<n-1;i++)
    ans+= Length(p[i+1]-p[i]);
  ans+=Length(p[0]-p[n-1]);
  return ans;
}
double isint(double x){
  return fabs(x - (int)(x+0.5))<EPS;
}
/*========多边形面积等内容=========*/

bool OnLeft(Line L,Point P){
  return Cross(L.v,P - L.P)>=0;          //如果线上的点不算就改成>
}
int HalfplaneIntersection(Line* L,int n,Point* poly){ //L数组是从0开始
  sort(L,L+n);
  int first,last;
  Point *p = new Point[n];
  Line *q = new Line[n];
  q[first = last = 0] = L[0];
  for(int i=1;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] = GetlineIntersection(q[last-1],q[last]);
  }
  while(first<last && !OnLeft(q[first],p[last-1]))last --;
  if(last-first<=1)return 0;
  p[last] = GetlineIntersection(q[last],q[first]);
  int m = 0;
  for(int i=first;i<=last;i++)poly[m++] = p[i];
  return m;
}
/*=========半平面交所需函数及主过程=========*/

vector<Point> ConvexHull(vector<Point> p){
  sort(p.begin(),p.end());
  //删除重复点
  p.erase(unique(p.begin(),p.end()),p.end());
  int n = p.size();
  vector<Point> ch(n+1);
  int m = 0;
  for(int i=0;i<n;i++){
    while(m>1 && dcmp(Cross(ch[m-1]-ch[m-2],p[i]-ch[m-2])) <= 0) m--;          //若需要把边线上的点也算上，把等号去掉
    ch[m++] = p[i];
  }
  int k = m;
  for(int i = n-2;i>=0;i--){
    while(m > k && dcmp(Cross(ch[m-1] - ch[m-2] , p[i]-ch[m-2] )) <= 0)m--;
    ch[m++] = p[i];
  }
  if(n > 1)m--;
  ch.resize(m);
  return ch;
}
/*============以上为凸包==========*/
double RotatingCalipers(const vector<Point>& p){
  int n = p.size();
  double ans = 0;
  Point v;
  int cur = 1;
  for(int i=0;i<n;i++){
    v = p[i]-p[(i+1)%n];
    while(dcmp(Cross(v,p[(cur+1)%n]-p[cur]))<0)
      cur = (cur+1)%n;
    ans = max(ans,max(Length(p[i]-p[cur]),Length(p[(i+1)%n]-p[(cur+1)%n])));
  }
  return ans;
}//求凸包上两点间最远距离
/*=========以上为旋转卡壳=========*/
double earthdis(Point a,Point b){
  double x1=PI*a.x/180.0;
  double y1=PI*a.y/180.0;
  double x2=PI*b.x/180.0;
  double y2=PI*b.y/180.0;
  return acos(cos(x1-x2)*cos(y1)*cos(y2)+sin(y1)*sin(y2));
}
/*=======给出经纬度，算出球体上两点之间的角度=====*/


//判断圆和直线交点,方程法
int getLineCircleIntersection(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-C.r*C.r;
  double delta = f*f - 4*e*g;             //判别式
  if(dcmp(delta) < 0)return 0;              //相离
  if(dcmp(delta) == 0){                   //相切
    t1 = t2 = -f/(2*e);sol.push_back(L.point(t1));
    return 1;
  }
  //相交
  t1 = (-f - sqrt(delta)) / (2*e); sol.push_back(L.point(t1));
  t2 = (-f + sqrt(delta)) / (2*e); sol.push_back(L.point(t2));
  return 2;
}
//圆和直线交点,几何法
int getLineCircleIntersection(Line L, Circle C,vector<Point>& sol){
  double d = DistanceToLine(C.c, L);
  if(dcmp(d - C.r)>0)return 0;  //相离
  Point ans = GetlineIntersection(L, Line(C.c,Normal(L.v)));
  if(dcmp(d - C.r) == 0){     //相切
    sol.push_back(ans);
    return 1;
  }
  //相交
  double len = sqrt(C.r*C.r-d*d);
  Vec v = L.v / Length(L.v);
  sol.push_back(ans + v * len),sol.push_back(ans - v * len);
  return 2;
}
//判断两圆相交
int getCircleCircleIntersection(Circle C1, Circle C2,vector<Point>& sol){
  double d = Length(C1.c-C2.c);
  if( dcmp(C1.r - C2.r)>0)
    swap(C1,C2);
  if(dcmp(d)==0){
    if(dcmp(C1.r-C2.r)==0)return -1;//重合
    return 0;
  }
  if(dcmp(C1.r + C2.r -d) < 0) return 0;//外离
  if(dcmp(fabs(C1.r-C2.r)-d)>0)return 0;//内含
  if(dcmp(C1.r + C2.r - d) == 0 || dcmp(fabs(C1.r - C2.r)-d) == 0){
    Vec p = C1.c-C2.c;
    sol.push_back(C2.c + p / Length(p) * C2.r);
    return 1;
  }//外切或内切
  double a = ang(C2.c - C1.c);
  double da = acos((C1.r * C1.r+d*d-C2.r*C2.r)/(2*C1.r*d));
  Point p1 = C1.point(a-da),p2 = C1.point(a+da);
  sol.push_back(p1);
  sol.push_back(p2);
  return 2;//相交
}
//过点p到圆C的切线
int getTangents(Point p,Circle C,vector<Line>& L){
  Vec u = C.c - p , temp;
  double dist = Length(u);
  if(dist < C.r) return 0;
  else if(dcmp(dist - C.r) == 0){
    temp = Normal(u);
    L.push_back(Line(p,temp));
    return 1;
  }else {
    double ang = asin(C.r/dist);
    temp = Rotate(u,-ang),L.push_back(Line(p,temp));
    temp = Rotate(u,+ang),L.push_back(Line(p,temp));
    return 2;
  }
}
//两圆公切线,返回切线条数，-1表示无穷多条
//注意,当两圆内切或者外切的时候,切点相同,均为p.
//sol里存的是切线，p为a上切点,p+v为b上切点
int getTangents(Circle A,Circle B,vector<Line>& sol){
  int cnt = 0;
  Point a,b;
  if(dcmp(A.r - B.r) < 0)swap(A,B);
  double 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);
  double rdiff = A.r - B.r;
  double rsum = A.r + B.r;
  if(dcmp(d2 - rdiff * rdiff) < 0) return 0;//内含

  double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
  if(dcmp(d2)==0 && A.r == B.r)return -1;//重合,切线无限多
  if(dcmp(d2-rdiff * rdiff)==0){//内切，一条切线
    a = A.point(base),b = B.point(base);
    sol.push_back(Line(a,Normal(A.c-B.c)));
    return 1;
  }
  //有外切线
  double ang = acos((A.r-B.r) / sqrt(d2));
  a = A.point(base+ang),b = B.point(base+ang),sol.push_back(Line(a,b-a)),cnt++;
  a = A.point(base-ang),b = B.point(base-ang),sol.push_back(Line(a,b-a)),cnt++;
  if(dcmp(d2 - rsum * rsum)==0){
    a = A.point(base),b = B.point(PI + base),sol.push_back(Line(a,Normal(A.r-B.r))),cnt++;
  }else if(dcmp(d2 - rsum * rsum)>0){
    double ang2 = acos((A.r + B.r) / sqrt(d2));
    a = A.point(base+ang2),b = B.point(PI+base+ang2),sol.push_back(Line(a,b-a)),cnt++;
    a = A.point(base-ang2),b = B.point(PI+base-ang2),sol.push_back(Line(a,b-a)),cnt++;
  }
  return cnt;

}
/*=========以上为圆的常用函数及计算=========*/

//double RotatingCalipers(const vector<Point>& p){
int main(){
    int n;
    vector <Point> p;
    double tx,ty;
    while(~scanf("%d",&n)){
        p.clear();
        for(int i=1;i<=n;i++){
            scanf("%lf%lf",&tx,&ty);
            p.push_back(Point(tx,ty));
        }
        p=ConvexHull(p);
        double goal=RotatingCalipers(p);
        goal*=goal;
        printf("%.f\n",goal);
    }
  return 0;
}
```

3.3 CGLines

```
struct Line {
  // Ax + By = C
  double A, B, C;
  Line(double _A, double _B, double _C):A(_A),B(_B),C(_C){}
  Line():A(0.0),B(0.0),C(0.0){}
};
```

```

Line getLineFromPoints(Point p1, Point p2) {
  double A = p2.y-p1.y;
  double B = p1.x-p2.x;
  double C = A*p1.x + B*p1.y;
  return Line(A,B,C);
}

// int = 0 无交点
// int = 1 一个交点
// int = 2 两直线重合
pair<Point, int> intersectPoint(Line l1, Line l2) {
  double det = l1.A*l2.B - l2.A*l1.B;
  if (dcmp(det) == 0) {
    // 两直线平行 重合
  if (dcmp(l1.A*l2.C - l2.A*l1.C) == 0 &&
        dcmp(l1.B*l2.C - l2.B*l1.C) == 0)
  return make_pair(Point(), 2);
    else
    return make_pair(Point(), 0);
  }
  else {
    double x = (l2.B*l1.C - l1.B*l2.C) / det;
    double y = (l1.A*l2.C - l2.A*l1.C) / det;
    return make_pair(Point(x,y), 1);
  }
}
```

3.4 CGkuangbin

3.5 CGpoints

```
struct Point {
  double x, y;
  Point(double _x, double _y):x(_x),y(_y){}
  Point():x(0.0),y(0.0){}
};
typedef Point Vec;

void showPoint(Point A) {
  printf("(%.6f, %.6f)\n", A.x, A.y);
}
const Vec operator + (Vec A, Vec B) {
  return Vec(A.x+B.x, A.y+B.y);
}
const Vec operator - (Vec A, Vec B) {
  return Vec(A.x-B.x, A.y-B.y);
}
const double operator * (Vec A, Vec B) {
  return A.x*B.x + A.y*B.y;
}
// A*B = |A|*|B|*sin(theta)
// theta为 A,B 向量的夹角
// 如果 A 在 B 的顺时针方向180度内，则theta取正值
// 向量叉乘
// 返回值为正时，表示 A 在 B 的右侧180度内
// 返回值的绝对值等于以A,B向量为邻边的平行四边型的面积
const double operator ^ (Vec A, Vec B) {
  return A.x*B.y - A.y*B.x;
}

double Lenth(Vec &v) {
  return sqrt(v*v);
}


// 将点 A 绕 p 逆时针旋转 theta 角度(弧度制)
// 1.平移坐标
// 2.旋转
// 3.平移坐标
Vec rotate(point A, point p， double theta) {
  A = A-p;
  point ret = point(A.x*cos(theta)-A.y*sin(theta), A.x*sin(theta)+A.y*cos(theta)) + p;
  return ret;
}


// 计算C到AB的距离
// isSeg = 1 : AB为线段
// isSeg = 0 : AB为直线
double linePointDist(Point A, Point B, Point C, bool isSeg) {
  double dist = ((B-A)^(C-A)) / sqrt((B-A)*(B-A));
  if (isSeg) {
    double dot1 = (C-B)*(B-A);
    if (dcmp(dot1) > 0) return sqrt((B-C)*(B-C));
    double dot2 = (C-A)*(A-B);
    if (dcmp(dot2) > 0) return sqrt((A-C)*(A-C));
  }
  return fabs(dist);
}

// 判断线段的两个端点是否在直线的两侧
bool lineCrossSeg(Point p1, Point p2, Point ls, Point le)
{
  Vec ver = ls-le, v1 = p1-ls, v2 = p2-ls;
  return dcmp((ver^v1)*(ver^v2)) <= 0;
}

// 判断点是否在线段上
// 叉积为 0 表示共线
bool pointOnSeg(point s1, point s2, point p, bool includeEnd)
{
  if ((s1 == p || s2 == p) && !includeEnd) return false;
  s2 = s2-s1;
  p = p-s1;
  return dcmp(s2^p)==0 && dcmp(s2*p)>=0;

  double minx = min(s1.x, s2.x);
  double maxx = max(s1.x, s2.x);
  double miny = min(s1.y, s2.y);
  double maxy = max(s1.y, s2.y);

  if ((s1 == p || s2 == p) && !includeEnd) return false;
  if (p.x-minx>=0
      && maxx-p.x>=0
      && p.y-miny>=0
      && maxy-p.y>=0
      && ((s2-s1)^(p-s1)) == 0)
    return true;
  else
    return false;
}

// 判断两条线段是否相交
// 两次跨立试验
typedef pair<Point, Point> seg;
bool segCrossSeg(seg a, seg b, bool includeEnd)
{
  Vec ver = b.X-b.Y, v1 = a.X-b.X, v2 = a.Y-b.X;
  int tmp1 = dcmp((ver^v1)*(ver^v2));
  ver = a.X-a.Y, v1 = b.X-a.X, v2 = b.Y-a.X;
  int tmp2 = dcmp((ver^v1)*(ver^v2));
  if (includeEnd)
    return tmp1 <= 0 && tmp2 <= 0;
  else
    return tmp1 < 0 && tmp2 < 0;
}

double areaPoly(vector<Point> &P) {
  double area = 0.0;
  for (int i = 1, n = P.size(); i+1 < n; i++) {
    area += (P[i+1]-P[0])^(P[i]-P[0]);
  }
  return area / 2.0;
}
```

3.6 areaCircle2

```
double areaCircle2(double x1, double y1, double r1, double x2, double y2, double r2) {
  double d = dist(x1, y1, x2, y2);
  if (r1+r2 < d-eps) return 0.0;
  if (fabs(r1-r2) > d-eps) {
    double tmp = min(r1, r2);
    return pi*tmp*tmp;
  }
  double ang1 = acos((r1*r1+d*d-r2*r2)/(2.0*r1*d));
  double ang2 = acos((r2*r2+d*d-r1*r1)/(2.0*r2*d));
  double ret = ang1*r1*r1+ang2*r2*r2-d*r1*sin(ang1);
  return ret;
}
```

3.7 closePair

```
bool cmp(Point a, Point b) {
  if (a.x == b.x) return a.y < b.y;
  return a.x < b.x;
}
bool cmpy(int a, int b) {
  return p[a].y < p[b].y;
}

double closePair(int l, int r) {
  if (l == r) return inf;
  if (l+1 == r) return Lenth((p[l]-p[r]));
  int mid =(l+r)/2;
  double tdis = min(closePair(l, mid), closePair(mid+1, r));
  int cnt = 0;
  for (int i = l; i <= r; i++) if (fabs(p[i].x-p[i+1].x) < tdis)
    t[cnt++] = i;
  sort(t, t + cnt, cmpy);
  for (int i = 0; i < cnt; i++) {
    for (int j = i+1; j < cnt && dcmp(p[t[j]].y-p[t[i]].y-tdis) < 0; j++) {
      double tmp = Lenth((p[t[i]]-p[t[j]]));
      tdis = min(tdis, tmp);
    }
  }
  return tdis;
}
```

3.8 convexHull

```
int n, s[maxn];
int top;
bool cmp(point a, point b) {
  int tmp = (a-p[0])^(b-p[0]);
  int dis1 = (a-p[0])*(a-p[0]);
  int dis2 = (b-p[0])*(b-p[0]);
  if (tmp > 0 || (tmp == 0 && dis1 > dis2)) return true;
  return false;
}

int graham() {
  for (int i = 0; i < n; i++) {
    if (p[i].y < p[0].y || (p[i].y == p[0].y && p[i].x < p[0].x))
      swap(p[i], p[0]);
  }
  sort(p+1, p+n, cmp);
  s[0] = 0;
  s[1] = 1;
  top = 1;
  for (int i = 2; i < n; i++) {
    // 注意是否包含边上的点
    // while(top && ((p[i-1]-p[0])^(p[i]-p[0])) <= 0) top--;
    while(top && ((p[s[top]]-p[s[top-1]])^(p[i]-p[s[top-1]])) < 0) top--;
    s[++top] = i;
  }
  top++;
  return top;
}

bool cmpxy(point a, point b) {
  if (a.x == b.x) return a.y < b.y;
  return a.x < b.x;
}
// 包含边上的点就将 <= 改为 <
int convexHull(Point *p, int n) {
  sort(p, p + n, cmpxy);
  int top = 0;
  for (int i = 0; i < n; i++) {
    while(top>1 && ((p[s[top-1]]-p[s[top-2]])^(p[i]-p[s[top-2]])) <= 0) top--;
    s[top++] = i;
  }
  int k = top;
  for (int i=n-2;i>=0;i--) {
    while(top>k && ((p[s[top-1]]-p[s[top-2]])^(p[i]-p[s[top-2]])) <= 0) top--;
    s[top++] = i;
  }
  if (n > 1) top--;
  return top;
}
```

3.9 halfPlane

```
struct Line {
  Point p, v;
  double ang;
  Line(){}
  Line(Point p, Vec v):p(p),v(v) {
    ang = atan2(v.y, v.x);
  }
  bool operator < (const Line &L) const {
    return ang < L.ang;
  }
};
// 包含边上的点将 > 改为 >=
bool onLeft(Line L, Point p) {
  return (L.v^(p-L.p)) > 0;
}
Point lineIntersection(Line a, Line b) {
  Point u = a.p-b.p;
  double t = (b.v^u)/(a.v^b.v);
  return a.p+a.v*t;
}
int halfPlaneIntersection(Line *L, int n, Point *poly) {
  sort(L, L+n);
  int first, last;
  Point *p = new Point[n];
  Line *q = new Line[n];
  q[first=last=0] = L[0];
  for (int i = 1; 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(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] = lineIntersection(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; // empty
  p[last] = lineIntersection(q[last], q[first]);
  int m = 0;
  for (int i = first; i <= last; i++) poly[m++] = p[i];
  return m;
}
```

3.10 minCircleCover

```
void minCircleCover(int n, Point &c, double &r) {
  random_shuffle(p, p+n); c = p[0]; r = 0;
  for (int i = 1; i < n; i++) if (Lenth(p[i]-c) > r + eps) {
      c = p[i]; r = 0;
      for (int j = 0; j < i; j++) if (Lenth(p[j]-c) > r + eps) {
          c.x = (p[i].x+p[j].x)/2.0;
          c.y = (p[i].y+p[j].y)/2.0;
          r = Lenth(p[j]-c);
          for (int k = 0; k < j; k++) if (Lenth(p[k]-c) > r + eps) {
              c = outerCircle(p[i].x,p[i].y,p[j].x,p[j].y,p[k].x,p[k].y);
              r = Lenth(p[i]-c);
          }
      }
  }
}
```

3.11 rotatingCaliper

```
// s 为保存有序的凸包点的下标的栈
// 此处的凸包点为顺时针的顺序
```

```
int rotatingCaliper(Point *p, int top) {
  int q = 1;
  int ans = 0;
  s[top] = 0;
  for (int i = 0; i < top; i++) {
    while( ((p[s[i+1]]-p[s[i]])^(p[s[q+1]]-p[s[i]])) >
           ((p[s[i+1]]-p[s[i]])^(p[s[q]]  -p[s[i]])) )
      q = (q+1)%top;
    ans = max(ans, (p[s[i]]-p[s[q]])*(p[s[i]]-p[s[q]]));
    // 处理两条边平行的情况
    ans = max(ans, (p[s[i+1]]-p[s[q+1]])*(p[s[i+1]]-p[s[q+1]]));
  }
  return ans;
}
```

4 DataStructure

4.1 Ltree

```
/*
 * 左倾树
 * 多个优先队列，合并logn
 *
 */

int lc[maxn], rc[maxn];
int v[maxn], size[maxn], d[maxn];
int tot;
int merge(int x, int y) {
  if (x==0||y==0) return x+y;
  if (v[x] < v[y]) swap(x, y);
  rc[x] = merge(rc[x], y);
  size[x] = size[lc[x]]+size[rc[x]]+1;
  if (d[rc[x]]>d[lc[x]]) swap(lc[x],rc[x]);
  d[x] = d[rc[x]]+1;
  return x;
}
int top(int x) {
  return v[x];
}
int sz(int x) {
  return size[x];
}
void pop(int &x) {
  x = merge(lc[x], rc[x]);
}
int newHeap(int x) {
  tot++;
  v[tot] = x;
  size[tot] = 1;
  lc[tot]=rc[tot]=d[tot]=0;
  return tot;
}
ll a[maxn];
int rt[maxn];
int L[maxn], R[maxn];
int n;
int main() {
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) {
    scanf("%d", &a[i]);
    a[i]-=i;
  }
  tot = 0;
  int cnt = 0;
  for (int i = 1; i <= n; i++) {
    cnt++;
    rt[cnt] = newHeap(a[i]);
    L[cnt] = R[cnt] = i;
    while(cnt>1 && top(rt[cnt]) < top(rt[cnt-1])) {
      rt[cnt-1] = merge(rt[cnt], rt[cnt-1]);
      R[cnt-1] = R[cnt];
      while(sz(rt[cnt-1]) > (R[cnt-1]-L[cnt-1]+2)/2)
        pop(rt[cnt-1]);
      cnt--;
    }
  }
  ll ans = 0;
  for (int i = 1; i <= cnt; i++) {
    for (int j = L[i]; j <= R[i]; j++)
      ans += abs(top(rt[i]) - a[j]);
  }
  printf("%I64d\n", ans);
  return 0;
}
```

4.2 ST_v2

```
const int maxn = 1e5+20;
int low[maxn][30];
void init(int *d) {
  for (int i = 1; i <= n; i++)
    low[i] = d[i];
  for (int j = 1; j <= 20; j++)
    for (int i = 1; i <= n; i++) if (i+(1<<j)-1 <= n)
      low[i][j] = min(low[i][j-1], low[i+(1<<(j-1))][j-1]);
}
int query(int s, int e) {
  int k = log2(e-s+1);
  return min(low[s][k], low[e-(1<<k)+1][k]);
}
```

4.3 Trie_v2

```
struct Trie{
  int next[26], cnt;
  void clear() {
    memset(next,0,sizeof(next));
  }
}trie[maxn];
int tot;
void insert(char *s)
{
  int cur = 1;
  for(int i = 0; s[i]; i++)
  {
    if(!trie[cur].next[s[i]-'a']) {
      trie[cur].next[s[i]-'a'] = ++tot;
      trie[tot].clear();
    }
    cur = trie[cur].next[s[i]-'a'];
  }
  trie[cur].cnt++;
}
int query(char *s) {
  int cur = 1;
  for (int i=0;s[i];i++) {
    if (trie[cur].next[s[i]-'a'])
      cur = trie[cur].next[s[i]-'a'];
    else
      return 0;
  }
  return trie[cur].cnt;
}
```

4.4 bit

```
struct bit {
  int s[maxn];
  int num;
  void add(int x, int z) {
    for (int i=x;i<=num;i+=(i&-i)) s[i]+=z;
  }
  int ask(int x) {
    int tmp=0;
    for (int i=x;i;i-=(i&-i)) tmp+=s[i];
    return tmp;
  }
  void clear(int n) {
    num=n;
    memset(s,0,sizeof(s));
  }
};
```

4.5 segtree_v2

```
#define lson l, m, rt<<1
#define rson m+1, r, rt<<1|1
typedef long long ll;
const int maxn = 1e5+20;
using namespace std;
```

```
ll segsum[maxn<<2], lazy[maxn<<2];
void pushup(int rt) {
  segsum[rt] = segsum[rt<<1] + segsum[rt<<1|1];
}
void build(int l, int r, int rt) {
  if(l == r) {
    scanf("%lld",&segsum[rt]);
    return;
  }
  int m = (l+r)>>1;
  build(lson); build(rson);
  pushup(rt);
}
void pushdown(int rt, int m) {
  lazy[rt<<1] += lazy[rt];
  lazy[rt<<1|1] += lazy[rt];
  segsum[rt<<1] += lazy[rt]*(m-(m>>1));
  segsum[rt<<1|1] += lazy[rt]*(m>>1);
  lazy[rt] = 0;
}
void update(int L, int R, int c, int l, int r, int rt) {
  if(L <= l && r <= R) {
    lazy[rt] += c;
    segsum[rt] += (r - l + 1) * c;
    return;
  }
  if(lazy[rt] != 0)
    pushdown(rt, r - l + 1);
  int m = (l + r) >> 1;
  if(L <= m) update(L, R, c, lson);
  if(R > m) update(L, R, c, rson);
  pushup(rt);
}
ll querysum(int L, int R, int l, int r, int rt) {
  if(L <= l && R >= r)
    return segsum[rt];
  if(lazy[rt] != 0)
    pushdown(rt, r - l + 1);
  int m = (l+r)>>1;
  ll tmp = 0;
  if(L <= m) tmp += querysum(L, R, lson);
  if(R > m) tmp += querysum(L, R, rson);
  return tmp;
}
```

5 Graph

5.1 Dijstra

```
/*
 *  Dijstra shortest path and minimal cost
 */
#define maxn 1020
#define INF 0x7f7f7f7f
using namespace std;
typedef pair<int, int> PII;
struct Edge
//l 为边的长度， c为费用
{
  int u, v, l, c;
  Edge(){}
  Edge(int u,int v, int l, int c):u(u),v(v),l(l),c(c){}
};
struct Node
//Node 用于 priority_queue 的记录
//v: node id
//l: length from start
//c: mincost
{
  int v, l, c;
  Node(){}
  Node(int v, int l, int c):v(v),l(l),c(c){}
  bool operator < (const Node &a) const
  //priority_queue 的优先级和 < 相反
  {
    if(l == a.l) return c > a.c;
    return l > a.l;
  }
};
vector<Edge>G[maxn];
priority_queue<Node>pq;
int dist[maxn],cost[maxn],vis[maxn],tot;
void add_edge(int u, int v, int l, int c)
{
  G[u].push_back(Edge(u, v, l, c));
}
PII dijstra(int s, int d)
//start s, dest d
{
  memset(dist, INF, sizeof(dist));
  memset(cost, INF, sizeof(cost));
  memset(vis, 0, sizeof(vis));
  while(!pq.empty()) pq.pop();
  pq.push(Node(s, 0, 0));
  while(!pq.empty())
  {
    const Node nd = pq.top();
    pq.pop();
    if(vis[nd.v]) continue;
    vis[nd.v] = true;
    dist[nd.v] = nd.l;
    cost[nd.v] = nd.c;
    if(nd.v == d) return make_pair(dist[d], cost[d]);
    for(int i = 0, len = G[nd.v].size(); i < len; i++)
    {
      Edge& e = G[nd.v][i];
      if(!vis[e.v])
      {
        pq.push(Node(e.v, nd.l + e.l, nd.c+e.c));
      }
    }
  }
  //dist[d]: shortest distance
  //cost[d]: mincost
  return make_pair(dist[d], cost[d]);
}
```

5.2 Dinic

```
#define ll long long
#define maxn 320
using namespace std;
int G[maxn][maxn], layer[maxn];
int m, n;
bool vis[maxn];
bool countLayer()
{
  queue<int>q;
  memset(layer, 0xff, sizeof(layer));
  layer[1] = 0;q.push(1);

  while(!q.empty())
  {
    int v = q.front();q.pop();
    for(int j = 1; j <= n; j++)
      if(G[v][j] > 0 && layer[j] == -1)
      {
        layer[j] = layer[v] + 1;
        if(j == n) return true;
        else q.push(j);
      }
  }
  return false;
}
int Dinic()
{
  int i;
  int maxflow = 0;
  deque<int> q;
  while(countLayer())
  {
    q.push_back(1);
    memset(vis, 0, sizeof(vis));
    vis[1] = true;
    while(!q.empty())
    {
      int nd = q.back();
      if(nd == n)
      {
        int minc = 1000000000;
        int minstart;
        for(i = 1; i < q.size();i++)
        {
          int vs = q[i-1];
          int ve = q[i];
          if(G[vs][ve] > 0 && minc > G[vs][ve])
          {
            minc = G[vs][ve];
            minstart = vs;
          }
        }
        maxflow += minc;
        for(i = 1; i < q.size(); i++)
        {
          int vs = q[i-1];
          int ve = q[i];
          G[vs][ve] -= minc;
          G[ve][vs] += minc;
        }
        while(!q.empty() && q.back() != minstart)
        {
          vis[q.back()] = false;
          q.pop_back();
        }
      }else{
        for(i = 1; i <= n; i++)
          if(G[nd][i] > 0 && layer[i] == layer[nd] + 1 && !vis[i])
          {
            vis[i] = true;
            q.push_back(i);
            break;
          }
        if(i > n) q.pop_back();
      }
    }

  }

  return maxflow;
}

int main()
{
  while(scanf("%d%d", &m, &n) != EOF)
  {
    int s, e, c;
    memset(G, 0, sizeof(G));
    for(int i = 0; i < m; i++)
    {
      scanf("%d%d%d",&s,&e,&c);
      G[s][e] += c;
    }
    printf("%d\n", Dinic());
  }
  return 0;
}
```

5.3 MaxFlowMinCost

```
const int maxn=120;
const int oo = 0x3f3f3f3f;
struct Edge
{
  int u, v, cap, flow, cost;
  Edge(int u, int v, int cap, int f, int cost):u(u), v(v), cap(cap), flow(f), cost(cost) {}
};
struct MCMF
{
  int n, m, s, t;
  vector<Edge> edge;
  vector<int> G[maxn];
  int inq[maxn], d[maxn], p[maxn], a[maxn];
  void init(int n)
  {
    this->n=n;
    for(int i=0; i<=n; i++)G[i].clear();
    edge.clear();
  }
  void AddEdge(int u, int v, int cap, int cost)
  {
    edge.push_back(Edge(u, v, cap, 0, cost));
    edge.push_back(Edge(v, u, 0, 0, -cost));
    m=edge.size();
    G[u].push_back(m-2);
    G[v].push_back(m-1);
  }
  bool SPFA(int s, int t, int& flow, int& cost)
  {
    memset(d, 0x3f, sizeof d);
    memset(inq, 0, sizeof inq);
    d[s]=0, inq[s]=1, p[s]=0, a[s]=oo;

    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
      int u=q.front();
      q.pop();
      inq[u]=0;
      for(int i=0; i<G[u].size(); i++)
      {
        Edge& e=edge[G[u][i]];
        if(e.cap>e.flow && d[e.v]>d[u]+e.cost)
        {
          d[e.v]=d[u]+e.cost;
          p[e.v]=G[u][i];
          a[e.v]=min(a[u], e.cap-e.flow);
          if(!inq[e.v])
          {
            q.push(e.v);
            inq[e.v]=true;
          }
        }
      }
    }
    if(d[t]==oo)return false;
    flow+=a[t];
    cost+=d[t]*a[t];
    int u=t;
    while(u!=s)
    {
      edge[p[u]].flow+=a[t];
      edge[p[u]^1].flow-=a[t];
      u=edge[p[u]].u;
    }
    return true;
  }
  int Mincost(int s, int t, int& cost)
  {
    int flow=0;
    while(SPFA(s, t, flow, cost))
      ;
    return flow;
  }
} net;

int ord[55][55], sto[55][55];
int main()
{
  int n, m, k;
  while(~scanf("%d%d%d", &n, &m, &k) && n+m+k)
  {
    for(int i=1; i<=n; i++)
      for(int j=1; j<=k; j++)
        scanf("%d", &ord[i][j]);
    for(int i=1; i<=m; i++)
      for(int j=1; j<=k; j++)
        scanf("%d", &sto[i][j]);
    int S=0, T=n+m+2;
    int cost=0;
    for(int p=1; p<=k; p++)
    {
      int sum=0;
      net.init(n+m+10);
      for(int i=1; i<=n; i++)
      {
        net.AddEdge(i, T, ord[i][p], 0);
        sum+=ord[i][p];
      }
      for(int i=1; i<=m; i++)
        net.AddEdge(S, i+n, sto[i][p], 0);
      for(int i=1; i<=n; i++)
        for(int j=1; j<=m; j++)
        {
          int x;
          scanf("%d", &x);
          net.AddEdge(n+j, i, oo, x);
        }
      if(~cost && net.Mincost(S, T, cost)<sum)
```

```
        cost=-1;
    }
    printf("%d\n", cost);
  }
  return 0;
}
```

5.4 floyed

```
const int maxn=110;
const int INF=10000000;
int  dist[maxn][maxn], G[maxn][maxn];
int  n, m, num, minc;
void floyd()
{
  minc=INF;
  // 求最小环
  for(int k=1; k<=n; k++)
  {
    for(int i=1; i<k; i++)
      for(int j=i+1; j<k; j++)
      {
        int  ans=dist[i][j]+G[i][k]+G[k][j];
        if(ans<minc)  //找到最优解
        {
          minc=ans;
        }
      }
    for(int i=1; i<=n; i++)
      for(int j=1; j<=n; j++)
      {
        if(dist[i][j]>dist[i][k]+dist[k][j])
        {
          dist[i][j]=dist[i][k]+dist[k][j];
        }
      }
  }
}
```

5.5 hungary

```
#define __maxNodes 4020
using namespace std;
struct Edge
{
  int from,to,weight;
  Edge(int f, int t, int w):from(f), to(t), weight(w) {}
};
vector<Edge> G[__maxNodes]; /* G[i] 存储顶点 i 出发的边的编号 */
int matching[__maxNodes]; /* 存储求解结果 */
int check[__maxNodes];
int n, m, sum;
/*DFS*/
bool dfs(int u)
{
  for (int i = 0; i < G[u].size(); i++) {
    int v = G[u][i].to;
    if (!check[v]) {     // 要求不在交替路中
      check[v] = true; // 放入交替路
      if (matching[v] == -1 || dfs(matching[v])) {
        // 如果是未盖点，说明交替路为增广路，则交换路径，并返回成功
        matching[v] = u;
        matching[u] = v;
        return true;
      }
    }
  }
  return false; // 不存在增广路，返回失败
}
int hungarian()
{
  int ans = 0;
  memset(matching, -1, sizeof(matching));
  for (int u=1; u <= n; ++u) {
    if (matching[u] == -1) {
      memset(check, 0, sizeof(check));
      if (dfs(u))
        ++ans;
    }
  }
  return ans;
}
```

5.6 kruskal

```
struct Edge
{
  int u, v, c;
  Edge(){}
  Edge(int u, int v, int c):u(u),v(v),c(c){}
  bool operator < (const Edge &e) const {
    return c < e.c;
  }
};
vector<Edge> ve;
int n, m;
int R[10020];
// dsj 1
int root(int x)
{
 while(R[x] != x)
   x = R[x] = R[R[x]];
 return R[x];
}
// dsj 2
int root(int x)
{
  if(R[x] == -1) return x;
  if(R[x] != -1) R[x] = root(R[x]);
  return R[x];
}
int main()
{
  scanf("%d%d", &n, &m);
  int u, v, c;
  for (int i = 1; i <= m; i++)
  {
    scanf("%d%d%d", &u,&n,&c);
    ve.push_back(Edge(u,n,c));
  }
  // for (int i = 1; i <= n; i++)
    // R[i] = i;
  memset(R, -1, sizeof(R));
  sort(ve.begin(), ve.end());
  int ans = 0;
  int Ru, Rv;
  for (int i = 0, len = ve.size(); i < len; i++)
  {
    Edge &now = ve[i];
    Ru = root(now.u);
    Rv = root(now.v);
    if(Ru != Rv)
    {
      ans += now.c;
      R[Ru] = Rv;
    }
  }
  printf("%d\n", ans);
  return 0;
}
```

5.7 prim

```
struct Edge
{
  int u, v, c;
  Edge(){}
  Edge(int u, int v, int c):u(u),v(v),c(c){}
};
vector<Edge> G[10020];
void addedge(int u, int v, int c)
{
  G[u].push_back(Edge(u,v,c));
  G[v].push_back(Edge(v,u,c));
}
int n, m;
int vis[10020];
int dist[10020];
int prim()
{
  int ans = 0;
  memset(vis, 0, sizeof(vis));
  memset(dist, 0x3f, sizeof(dist));
  vis[1] = 1;
  int minid, minc;
  int now = 1;
  for (int t = 1; t < n; t++)
  {
    for (int i = 0, len = G[now].size(); i < len; i++)
    {
      int to = G[now][i].v, c = G[now][i].c;
      if(vis[to] == 1) continue;
      if(dist[to] > c)
        dist[to] = c;
    }
    minid = -1;
    minc = 0x3f3f3f3f;
    for (int i = 1; i <= n; i++) if((!vis[i]) && dist[i] < minc)
    {
      minid = i;
      minc = dist[i];
    }
    ans += minc;
    vis[minid] = 1;
    now = minid;
  }
  return ans;
}

int main()
{
  scanf("%d%d", &n, &m);
  int u,v,c;
  for (int i = 0; i < m; i++)
  {
    scanf("%d%d%d", &u,&v,&c);
    addedge(u,v,c);
  }
  printf("%d\n" ,prim());
  return 0;
}
```

5.8 tarjan

```
#define maxn 100020
using namespace std;
vector<int>v[maxn];
bool instack[maxn];
int dfn[maxn], low[maxn];
int n, m;
int depth, strongcnt;
int belong[maxn], strongsize[maxn];
int stk[maxn], top;
int inde[maxn], outde[maxn];

void clear(){
  memset(dfn, 0, sizeof(dfn));
  memset(low, 0, sizeof(low));
  memset(instack, 0, sizeof(instack));
  memset(belong, 0, sizeof(belong));
  memset(strongsize, 0, sizeof(strongsize));
  memset(inde, 0, sizeof(inde));
  memset(outde, 0, sizeof(outde));
  depth = 0;
  strongcnt = 0;
  for (int i = 1; i <= n; ++i)
  {
    v[i].clear();
  }
}

void tarjan(int u){
  dfn[u] = low[u] = ++depth;
  instack[u] = true;
  stk[++top] = u;
  int to;
  for (int i = 0; i < v[u].size(); ++i)
  {
    to = v[u][i];
    if(!dfn[to]){
      tarjan(to);
      low[u] = min(low[to], low[u]);
    }else if (instack[to])
    {
      //dfn[i] not low[i];
      low[u] = min(low[u], dfn[to]);
    }

  }
  if (dfn[u] == low[u])
  {
    strongcnt++;
    do
    {
      to = stk[top--];
      instack[to] = false;
      belong[to] = strongcnt;
      strongsize[strongcnt]++;
    } while (to != u);
  }
}
int main()
{
  for (int i = 1; i <= n; ++i)
  {
    if (!dfn[i])
    {
      tarjan(i);
    }
  }
  int to;
  for(int i = 1; i <= n; i++)
  {
    for (int j = 0; j < v[i].size(); ++j)
    {
      to = v[i][j];
      if(belong[i] != belong[to]){
        outde[belong[i]]++;
        inde[belong[to]]++;
      }
    }
  }
  return 0;
}
```

6 ToolsTricks

6.1 ConvexHullTrick

```
/*
 *  The ConvexHull Trick
 *  CF 244 div2 E
 *  斜率需要按升序排序
 *  求对应 x 的最大的 y 值
 *  nlogn
 *
 */
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
struct Line
{
  ll a, b;
  ll cal(ll x) { return a*x+b; }
  Line(ll a, ll b):a(a),b(b){}
  Line(){}
};
struct ConvexHull
{
  int size;
  Line *hull;
  ConvexHull(int maxsize)
  {
    hull = new Line[++maxsize], size = 0;
  }
  bool isbad(ll cur, ll pre, ll ne)
  {
    Line c = hull[cur], p = hull[pre], n = hull[ne];
    return (c.b - n.b) * (c.a - p.a) <= (p.b - c.b) * (n.a - c.a);
  }
  void addLine(ll a, ll b)
  {
    hull[size++] = Line(a,b);
    while (size > 2 && isbad(size - 2, size - 3, size - 1))
```

```
      hull[size-2] = hull[size-1], size--;
  }
  ll query(ll x)
  {
    int l = -1, r = size - 1;
    while (r - l > 1)
    {
      int m = (l + r) / 2;
      if (hull[m].cal(x) <= hull[m+1].cal(x))
        l = m;
      else
        r = m;
    }
    return hull[r].cal(x);
  }
};

int const maxn = (int)2e5+1;
int n, a[maxn];
ll sum[maxn], ans, dans;

int main()
{
  scanf("%d", &n);
  ConvexHull ch = ConvexHull(n);
  sum[0] = 0;
  for (int i = 1; i <= n; i++)
  {
    scanf("%d", &a[i]);
    sum[i] = sum[i-1] + a[i];
    ans += (ll)a[i] * i;
  }
  ch.size = 0;
  for (int r = 2; r <= n; r++)
  {
    ch.addLine(r-1, -sum[r-2]);
    dans = max(dans, ch.query(a[r]) + sum[r-1]-(ll)a[r]*r);
  }
  ch.size = 0;
  for (int l = n - 1; l >= 1; l--)
  {
    ch.addLine(-(l+1), -sum[l+1]);
    dans = max(dans, ch.query(-a[l]) + sum[l] - (ll)a[l]*l);
  }
  printf("%I64d\n", ans + dans);
  return 0;
}
```

6.2 cantor

```
/*
 *  康拓展开
 *  元素个数 len
 *  元素0-9
 *  0-based count
 *  last edit : 2015/9/25
 */
int fact[10] = {1,1,2,6,24,120,720,5040,40320,362880};
int cantor(int* a,int len)
{
  int ret = 0;
  for(int i = 0; i < len; i++)
  {
    int tmp = 0;
    for(int j = i+1; j < len; j++)if(a[i] > a[j]) tmp++;
    ret += tmp * fact[len-i-1];
  }
  return ret;
}

void cantorrev(int* a,int d, int len)
{
  int vis[10] = {0}, tmp, tt;
  for(int i = 0; i < len; i++)
  {
    tmp = d / fact[len-i-1];
    d %= fact[len-i-1];
    //the min
    tt = 0;
    while(tmp || vis[tt])
    {
      if(vis[tt] == 0)
        tmp--;
      tt++;
    }
    vis[tt] = 1;
    a[i] = tt;
  }
}
```

6.3 merge_sort_reverse

```
int arr[1000200], tarr[1000200];
int cnt;
void merge(int low, int mid, int high)
{
  int i, j, k;
  for (i = low, j = mid + 1, k = 0; i <= mid && j <= high;)
  {
    if(arr[i] < arr[j])
      tarr[k++] = arr[i++];
    else
    {
      tarr[k++] = arr[j++];
      cnt += mid - i + 1;
    }
  }
  while(i <= mid) tarr[k++] = arr[i++];
  while(j <= high) tarr[k++] = arr[j++];

  for (k = 0; low <= high; low++, k++)
    arr[low] = tarr[k];
}
void mergesort(int low, int high)
{
  if(low == high) return;
  int mid = (low + high) / 2;
  mergesort(low, mid);
  mergesort(mid + 1, high);
  merge(low, mid, high);
}
int main()
{
  int n;
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);
  cnt = 0;
  mergesort(0, n-1);
  printf("%d\n", cnt);
  return 0;
}
```

6.4 stl

```
/*
 *  bool next_permutation(a, a + n);
 *  true:   has next permutation and
 *        change a[n] into the next permutation
 *  false:  the a[n] now is the last permutation
 */

//sample code:
sort(a, a + len);
do {
  for (int i = 0; i < len; i++)
    cout << a[i];
  cout << endl;
} while(next_permutation(a, a + len));
```

7 Game

7.1 Wythoff

```
//Wythoff Game
//A first
//B second
//当 n 过大时需要用高精度处理，和精确的黄金比例数
int main()
{
  int T;
  scanf("%d", &T);
  while(T––)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    if(a > b) swap(a, b);

    int k = b – a;
    if(a == (int)((k)*(1+sqrt(5.0))/2.0)) cout << "B" << endl;
    else cout << "A" << endl;
  }
  return 0;
}
```

8 BigInt

8.1 Java

```
/*
//将r进制保存的 string 转化为10进制的数
Integer.parseInt(String, radian);

//将10进制的数转化为对应进制的字符串
Integer.toBinaryString(n);
Integer.toOctalString(n);
Integer.toHexString(n);
*/

import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.StringTokenizer;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.InputStream;

/**
 * Built using CHelper plug–in
 * Actual solution is at the top
 */
public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        PrintWriter out = new PrintWriter(outputStream);
        TaskA solver = new TaskA();
        solver.solve(1, in, out);
        out.close();
    }

    static class TaskA {
        public void solve(int testNumber, InputReader in, PrintWriter out) {
            int n = in.nextInt();
            int l = in.nextInt();
            int v1 = in.nextInt();
            int v2 = in.nextInt();
            int k = in.nextInt();
            double gain = 1.0 / v1 – 1.0 / v2;
            double together = 1.0 / (v1 + v2);
            double left = 0;
            double right = l / (double) v1;
            while (right – left > 1e–7 * Math.max(1.0, right)) {
                double middle = (left + right) / 2;
                double needBus = (l / (double) v1 – middle) / gain;
                double time = 0;
                for (int first = 0; first < n; first += k) {
                    if (first + k < n)
                        time += 2 * needBus * together;
                    else
                        time += needBus / v2;
                }
                if (time <= middle) {
                    right = middle;
                } else {
                    left = middle;
                }
            }
            out.println((left + right) / 2);
        }

    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }

    }
}
```