ACM-ICPC Reference

Universidad de Oriente

Conquer & Divide

Index

MATHS

Geometric Series:

$$\sum_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1, \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$$

$$\sum_{i=0}^{n} ic^i = \frac{nc^2-(n+1)c^{n+1}+c}{(c-1)^2}, c \neq 1, \sum_{i=0}^{\infty} ic^i = \frac{c}{(1-c)^2}, |c| < 1$$

Fibonnaci Formulas:

$$F_{i+1}F_{i-1} - F_i^2 = (-1)^i \qquad F_{n+k} = F_k F_{n+1} + F_{k-1}F_n$$

$$\sum_{i=0}^{n} F_i = F_{n+2} - 1 \qquad F_n^2 - F_{n+1}F_{n-1} = (-1)^n$$

$$\gcd(F_m, F_n) = F_{\gcd(m.n)}$$

Catalan Numbers:

$$C[n] = FOR(k = 0, n-1)\ C[k] * C[n-1-k]$$

$$C[n] = Comb(2*n, n) / (n+1)$$

$$C[n] = C[n-1] * (4*n-2) / (n+1)$$

NUMBER THEORY

```
// Modular Multiplication of big numbers
inline ll mulmod(ll a, ll b, ll m) {
        ll x = 0, y = a % m;
        while (b > 0) {
                if (b % 2 == 1)  x = (x + y) % m;
                y = (y * 2) % m;
                b /= 2;
        }
        return x;
}

// Miller-Rabin is prime? (probability test)
bool suspect(ll a, int s, ll d, ll n) {
    ll x = powMod(a, d, n);
    if (x == 1) return true;
        for (int r = 0; r < s; r++) {
          if (x == n - 1)    return true;
          x = mulmod(x, x, n);
    }
    return false;
}

// {2,7,61,0}      is for n < 4759123141 (= 2^32)
unsigned test[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 0 }; //n < 1e16

bool miller_rabin(ll n) {
    if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
    ll d = n - 1; int s = 0;
    while (d % 2 == 0) ++s, d /= 2;
    for (int i = 0; test[i] < n && test[i] != 0; i++)
        if (!suspect(test[i], s, d, n))
            return false;
    return true;
}

// Pollard Rho - Randomized Factorization Algorithm O(sqrt(s(n))) expected
#define func(x) (mulmod(x, x+B, n)+ A )

ll pollard_rho(ll n) {
  if( n == 1 ) return 1;
  if( miller_rabin( n ) )  return n;
  ll d = n;
  while( d == n ){
    ll A = 1 + rand()%(n-1), B = 1 + rand()%(n-1);
    ll x = 2, y = 2;
    d = -1;
    while( d == 1 || d == -1 ){
      x = func(x), y = func(func(y));
      d = __gcd( x-y, n );
    }
  }
  return abs(d);
}

// Algoritmo Shanka-Tonelli,  devuelve x (mod p) tal que x^2 = a (mod p)
long long solve_quadratic( long long a, int p ){
    if( a == 0 ) return 0;
    if( p == 2 ) return a;
    if( powMod(a,(p-1)/2, p) != 1 ) return -1;

    int phi = p-1, n = 0, k = 0, q = 0;
    while( phi%2 == 0 ) phi/=2, n ++;
    k = phi;
    for( int j = 2; j < p; j ++ )
        if( powMod( j, (p-1)/2, p ) == p-1 ){
          q = j; break;
        }
    long long t = powMod( a, (k+1)/2, p );
    long long r = powMod( a, k, p );
    while( r != 1 ){
```

```
        int i = 0, v = 1;
        while( powMod( r, v, p ) != 1 ) v *= 2, i ++;
        long long e = powMod( 2, n-i-1, p );
        long long u = powMod( q, k*e, p );
        t = (t*u)%p;
        r = (r*u*u)%p;
    }
    return t;
}

//GCD extendido - devuelve x,y tal que ax+by = gcd(a,b)
par egcd (int a, int b){
    if (b == 0) return make_pair(1,0);
    else {
        par RES = egcd (b , a%b) ;
        return par(RES.second, RES.first-RES.second*(a/b));
    }
}

int inv(int n ,int m){  //Inverso Modular
      ii EGCD = egcd(n, m) ;
      return ((EGCD.first % m)+m)% m;
}

//Teorema Chino de los Restos
int crt (int x[], int m[], int k){
    int i, tmp, MOD, RES;

    MOD = 1;
    for (i=0; i <k ; i++) MOD *= m[i];

    RES = 0;
    for (i =0; i <k ; i++){
        tmp = MOD/m[i];
        tmp *= inv(tmp, m[i]);
        RES += (tmp*x[i]) % MOD;
    }
    return RES % MOD;
}

//Inverso de Factoriales
   fact[0] = 1;
   for(int i=1;i<MN;i++) fact[i] = (fact[i-1]*(ll)i)%mod;
   ifact[MN-1] = POW(fact[MN-1], mod - 2);
   for(int i=MN-2;i>=0;i--) ifact[i] = (ifact[i+1]*(ll)(i+1))%mod;
```

```
//FFT
typedef complex<double> base;
void fft (vector<base> & a, bool invert) {
    int n = (int) a.size();
    for (int i=1, j=0; i<n; ++i) {
        int bit = n >> 1;
            for (; j>=bit; bit>>=1)
                j -= bit;
            j += bit;
            if (i < j)
                swap (a[i], a[j]);
    }
    for (int len=2; len<=n; len<<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        base wlen (cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            base w (1);
            for (int j=0; j<len/2; ++j) {
                base u = a[i+j],  v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
    } } }
    if (invert)
        for (int i=0; i<n; ++i)
            a[i] /= n;
}
void multiply (const vector<int> & a, const vector<int> & b,
                    vector<int> & res) {

    vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
    size_t n = 1;
    while (n < max (a.size(), b.size()))  n <<= 1;
    n <<= 1;
    fa.resize (n),  fb.resize (n);

    fft (fa, false),  fft (fb, false);
    for (size_t i=0; i<n; ++i)  fa[i] *= fb[i];
    fft (fa, true);

    res.resize (n);
    for (size_t i=0; i<n; ++i)
            res[i] = int (fa[i].real() + 0.5);
}
```

GEOMETRY

```
inline double dot(PT p, PT q){return p.x*q.x + p.y+q.y;}
inline double dist2(PT p, PT q){ return dot(p-q,p-q); }
inline double cross(PT p, PT q){
   return p.x*q.y - p.y*q.x;
}
inline PT rotateCCW90(PT p) { return PT(-p.y,p.x); }
inline PT rotateCW90(PT p) { return PT(p.y,-p.x); }
inline PT rotateCCW(PT p, double t){
   return PT(p.x*cos(t) - p.y*sin(t), p.x*sin(t) + p.y*cos(t)); }

inline bool linesParallel(PT a, PT b, PT c, PT d){
   return fabs(cross(a-b,c-d)) < EPS;
}
inline bool linesCollinear(PT a, PT b, PT c, PT d){
   return linesParallel(a,b,c,d)
      && fabs(cross(a-b,a-c)) < EPS
      && fabs(cross(c-d,c-a)) < EPS;
}
inline bool segmentsIntersects(PT a, PT b, PT c, PT d){
   if (linesCollinear(a,b,c,d)){
      if (dist2(a,c) < EPS || dist2(a,d) < EPS ||
          dist2(b,c) < EPS || dist2(b,d) < EPS) return true;
      if (dot(c-a, c-b) > 0 && dot(d-a,d-b) > 0
           && dot(c-b,d-b) > 0)
          return false;
      return true;
   }
   if (cross(d-a,b-a)*cross(c-a,b-a) > 0) return false;
   if (cross(a-c,d-c)*cross(b-c,d-c) > 0) return false;
}

PT computeLineIntersection(PT a, PT b, PT c, PT d) {
   b=b-a; d=c-d; c=c-a;
   assert(dot(b, b) > EPS && dot(d, d) > EPS);
   return a + b*cross(c, d)/cross(b, d);
}

PT computeCircleCenter(PT a, PT b, PT c) {
   b=(a+b)/2, c=(a+c)/2;
   return computeLineIntersection(b, b+rotateCW90(a-b), c,
                                     c+rotateCW90(a-c));
}
```

```
double area(vector<PT> &P) {
   double result = 0.0, x1, y1, x2, y2;
   for (int i = 0; i < (int)P.size() - 1; i++) {
      x1 = P[i].x; x2 = P[(i + 1)].x; // assume that the first vertex
      y1 = P[i].y; y2 = P[(i + 1)].y; // is equal to the last vertex
      result += (x1 * y2 - x2 * y1);
   }
   return fabs(result) / 2.0;
}

PT pivot;
bool collinear(PT p, PT q, PT r){
   return fabs(cross(r-q,p-q)) < EPS;
}
bool angleCmp(PT a, PT b) {
   if (collinear(pivot, a, b))
      return dist2(pivot, a) < dist2(pivot, b); // determine closer
   double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
   double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
   return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
}
```

STRINGS

```
//Z - Algorithm
void Z_algorithm( ){
  int L = 0, R = 0, k;
  for (int i = 1; i < n; i++){
    if( i <= R && z[i-L] < R-i+1 )
      z[i] = z[i-L];
    else{
      L = i, R = max( R, i );
      while( R < n && s[R-L] == s[R] )
        R ++;
      z[i] = R - L;
      R --;
} } }

//Manacher
int rad[ 2 * MAXLEN ], n;
char s[MAXLEN];
void manacher( ){ /// i%2!=0 par, i%2==0 impar
  int i, j, k;
  for ( i = 0, j = 0; i < 2 * n - 1; i += k ) {
```

```
   while ( i - j >= 0 && i + j + 1 < 2 * n &&
           s[ ( i - j ) / 2 ] == s[ ( i + j + 1 ) / 2 ] )
             j++;
           rad[i] = j;
   for ( k = 1; k <= rad[i] && rad[ i - k ] != rad[i] - k; k++ )
     rad[ i + k ] = min( rad[ i - k ], rad[i] - k );
           j = max( j - k, 0 );
} }

//Suffix-Array (N log(N))
#define MN 200005
int N, in[305], prox[MN], sa[MN], k, cant[MN], pos[MN], lcp[MN], may, s1;
char A[MN];
bool b1[MN], b2[MN];
void LCP(  ){
 for( int p = 0, i = 0; i < N; i ++ )
   if( pos[i] != N - 1 ){
     for(int j = sa[pos[i]+1]; i + p <= N && j + p <= N && A[i+p] == A[j+p]; p++);
     lcp[pos[i]] = p;
     if( p ) p --;
  }
}
inline void upper( int x ){
  int p = pos[x];
  pos[x] = p + cant[p];
  cant[p] ++;
  b2[pos[x]] = true;
}
void Suffix_Array( ){
 fill( in, in + 300, -1 );
 for( int i = 0; i < N; i ++ )
   prox[i] = in[(int)A[i]], in[(int)A[i]] = i;
 for( int i = 'a'; i <= 'z'; i ++ )
   for( int j = in[i]; j != -1; j = prox[j] ){
     sa[k] = j;
     if( j == in[i] )
       b1[k] = true;
     k ++;
   }
 int p;
 for( int H = 1; H < N; H *= 2 ){
  fill( b2, b2 + N + 1, false );
  for( int i = 0; i < N; i = k ){
   for( k = i+1; k < N && !b1[k]; k++);
   cant[i] = 0;
   for( int j = i; j < k; j ++ )
    pos[sa[j]] = i;
  }
  upper( N - H );
  for( int i = 0; i < N; i = k ){
   for( k = i+1; k < N && !b1[k]; k ++ );

   for( int j = i; j < k; j ++ )
    if( sa[j] - H >= 0 )
     upper( sa[j] - H );

   for( int j = i; j < k; j ++ )
    if( sa[j] - H >= 0 && b2[pos[sa[j] - H]] ){
     for( p = pos[sa[j] - H] + 1; p < N && !b1[p] && b2[p]; p ++ )
       b2[p] = false;
    }
  }
  for( int i = 0; i < N; i ++ ){
   sa[pos[i]] = i;
   b1[i] = ( b1[i] || b2[i] );
  }
 }
 LCP(  );
}

//Aho-Corasick
const int alph = 26;
struct tree {
 int parent, slink;
 bool band;
 int hij[30];
 tree( int p ){
  parent = p, slink = 0, band = false;
  fill( hij, hij + 30, -1 );
 }
};
vector<tree> trie;
void addWord( string s1 ){
 int root = 0;
 for( int i = 0; i < (int)s1.length(); i ++ ){
   if( trie[root].hij[s1[i] - 'a'] == -1 ){
     trie[root].hij[s1[i] - 'a'] = trie.size();
     trie.push_back( tree( root ) );
   }
   root = trie[root].hij[s1[i] - 'a'];
```

```
 }
 trie[root].band = true;
}

queue<int> Q;
void buildSuffixLinks(  ){
 int nod, nextC;
 Q.push( 0 );
 Q.push( 0 );

 while( !Q.empty() ){
  nod = Q.front(), Q.pop();
  nextC = Q.front(), Q.pop();

  for( int i = 0; i <= alph; i ++ )
   if( trie[nod].hij[i] != -1 ){
    Q.push( trie[nod].hij[i] );
    Q.push( i );
   }

  if( nod == 0 || trie[0].hij[nextC] == nod )
   continue;

  int& link = trie[nod].slink;
  link = trie[trie[nod].parent].slink;
  while( link != 0 && trie[link].hij[nextC] == -1 )
    link = trie[link].slink;

  link = trie[link].hij[nextC];

  if( link == -1 )
    link ++;

  if( trie[link].band )
    trie[nod].band = true;
 }
}

int go( int nod, char c ){
 if( nod == 0 )
   return trie[0].hij[c - 'a'];

 if( trie[nod].hij[c - 'a'] != -1 )
  return trie[nod].hij[c - 'a'];

 int link = trie[nod].slink;
 while( link != 0 && trie[link].hij[c-'a'] == -1 )
  link = trie[link].slink;

 return trie[link].hij[c-'a'];
}

long long Dp[10005][1005], MOD = 1e9+7;
int automata[10005][30], N, M;
void Aho_Corasick( ){
  string tmp;
  cin >> N >> M;

  trie.clear();
  trie = vector<tree> ( 1, tree(0) );
  for( int i = 1; i <= M; i ++ ){
    cin>>tmp;
    addWord( tmp );
  }

  buildSuffixLinks( );

  for( int j = 0; j < (int)trie.size(); j ++ )
    for( int h = 'a'; h <= 'z'; h ++ )
      automata[j][h-'a'] = go( j, h );
}

// Decomposition of Lyndon  s = w1w2w3..wk, w1 >= w2 >=...>= wk.
void lyndon( ){
    string s; cin >> s;
    int n = (int)s.length(), i = 0;
    while( i < n ){
       int j = i+1, k = i;
       while( j < n && s[k] <= s[j] ){
          if( s[k] < s[j] ) k = i;
          else ++k;
          ++j;
       }
       while( i <= k ){
          cout << s.substr( i, j-k )<<endl; /// lyndon descomp
          i += j-k;
}  }  }
//Menor Rotación Lexicográfica (O (N))
int lexRot(string str){
    int n = str.size(), ini=0, fim=1, rot=0;
```

```
    str += str;
    while(fim < n && rot+ini+1 < n)
        if (str[ini+rot] == str[ini+fim]) ini++;
        else if (str[ini+rot] < str[ini+fim]) fim += ini+1, ini = 0;
        else rot = max(rot+ini+1, fim), fim = rot+1, ini = 0;
    return rot;
}
```

GRAFOS (FLUJOS)

```
int pos, Index[10005];///index = -1
struct edges{
 int nod, newn, cap, cost, next;
 bool band;
 edges( int a = 0, int b = 0, int c = 0, int d = 0, int e = 0 ){
   nod = a, newn = b, cap = c, cost = d, next = e;
 }
 int nextn ( int a ){
  if( nod == a )
   return newn;
  return nod;
 }
} G[100005];

///Params: nod, newn, cap, cost
void insertar( int a, int b, int c, int d = 0 ){
  G[pos] = edges( a, b, c, d, Index[a] );
  Index[a] = pos ++;
  G[pos] = edges( b, a, 0, -d, Index[b] );
  Index[b] = pos ++;
}

// Dinic
int lv[2005], Id[2005];
bool Bfs( int limt ){
 while( !Q.empty() )
  Q.pop();

 fill( lv, lv + 2001, 0);
 lv[0] = 1;
 Q.push( 0 );

 int nod, newn;
 while( !Q.empty() ) {
  nod = Q.front();
  Q.pop();

 for( int i = Index[nod]; i != -1; i = G[i].next ){
  newn = G[i].newn;

  if( lv[newn] != 0 || G[i].cap < limt )
   continue;

  lv[newn] = lv[nod] + 1;
  Q.push( newn );

  if( newn == fin )
   return true;
 }
 }
 return false;
}
bool Dfs( int nod, int limt ){
 if( nod == fin )
  return true;

 int newn;
 for( ; Id[nod] != -1; Id[nod] = G[Id[nod]].next ){
  newn = G[Id[nod]].newn;

  if( lv[nod] + 1 == lv[newn] && G[Id[nod]].cap >= limt && Dfs( newn, limt ) ){
   G[Id[nod]].cap -= limt;
   G[Id[nod]^1].cap += limt;
   return true;
  }
 }
 return false;
}
int flow;
void Dinic( ){
 flow = 0;
 for( int limt = 4; limt > 0;  ){
  if( !Bfs( limt ) ){
   limt >>= 1;
   continue;
  }
  for( int i = 0; i <= fin; i ++ )
   Id[i] = Index[i];
  while( limt > 0 && Dfs( 0, limt ) )
   flow += limt;
```

```
 }
}
//Edomnd Karp
void Edmond_Karp( ){
 int nod, newn, flow[10005], P[10005];
 bool band;
 for( ; ; ){
  fill( flow, flow + 2 + 2*N, 0 );
  fill( P, P + 2 + 2*N, -1 );
  P[0] = 0, flow[0] = 1;
  band = false;
  while( !Q.empty() ) Q.pop();
  Q.push( 0 );

  while( !band && !Q.empty() ){
   nod = Q.front();
   Q.pop();
   for( int i = Index[nod]; i != -1; i = G[i].next ){
    newn = G[i].newn;
    if( P[newn] != -1 || !G[i].cap  )
     continue;
    flow[newn] = min( G[i].cap, flow[nod] );
    P[newn] = i;
    Q.push( newn );
    if( newn == fin ){
     band = true;
     break;
  }}}
  if( !flow[fin] )
    break;
  sol += flow[fin];
  for( int i = fin; i != 0; i = G[P[i]].nod ){
    G[P[i]].cap -= flow[fin];
    G[P[i]^1].cap += flow[fin];
}}}

//StoerWagner
int G[MAXN][MAXN], w[MAXN], N;
bool A[MAXN], merged[MAXN];
int StoerWagner(int n){
    int best = 1e8;
    for(int i=1;i<n;++i) merged[i] = 0;
    merged[0] = 1;
    for(int phase=1;phase<n;++phase){
        A[0] = 1;
        for(int i=1;i<n;++i){
            if(merged[i]) continue;
            A[i] = 0;
            w[i] = G[0][i];
        }
        int prev = 0,next;
        for(int i=n-1-phase;i>=0;--i){
            next = -1;
            for(int j=1;j<n;++j)
                if(!A[j] && (next==-1 || w[j]>w[next]))
                    next = j;
            A[next] = true;
            if(i>0){
                prev = next;
                for(int j=1;j<n;++j) if(!A[j])
                        w[j] += G[next][j];
            }
        }
        if(best>w[next]) best = w[next];
        for(int i=0;i<n;++i){
            G[i][prev] += G[next][i];
            G[prev][i] += G[next][i];
        }
        merged[next] = true;
    }
    return best;
}

//Max Flow Min Cost
priority_queue<par, vector<par>, greater<par> >Qp;
par Max_Flow_Min_Cost( ){
 int FlowF = 0, CostF = 0, F[1005], parent[1005], nod, newn, newc, flow,
dist[1005], cost;

 for( ; ; ){
  fill( F + 1, F + 1 + Fin, 0 );
  fill( dist + 1, dist + 1 + Fin, 1 << 30 );
  F[In] = 1 << 30, dist[In] = 0;
  Qp.push( par( 0, In ) );
  while( !Qp.empty() ){
   nod = Qp.top().second, cost = Qp.top().first;
   Qp.pop();
   flow = F[nod];
   for( int i = Index[nod]; i != -1; i = G[i].next ){
    newn = G[i].newn;
```

```
    newc = cost + G[i].cost + Phi[nod] - Phi[newn];

    if( G[i].cap > 0 && dist[newn] > newc ){
     dist[newn] = newc;
     F[newn] = min( flow, G[i].cap );
     parent[newn] = i;
     Qp.push( par( newc, newn ) );
   }}}

   if( F[Fin] <= 0 )
     break;
   CostF += (( dist[Fin] + Phi[Fin] ) * F[Fin] );
   FlowF += F[Fin];
   for( int i = 1; i <= N; i ++ )
    if( F[i] )
     Phi[i] += dist[i];
   nod = Fin;
   while( nod != In ){
    G[parent[nod]].cap -= F[Fin];
    G[parent[nod]^1].cap += F[Fin];
    nod = G[parent[nod]].nod;
   }
  }
  return par( CostF, FlowF );
}

//Edmond – MaxMatching en grafo general
namespace MaxMatching{
    const int MAXV = 1e3 + 10, MAXE = 1e3 + 10;
    int V, edges;
    int match[MAXV];
    int que[MAXV], head, tail;
    int start, finish;
    int father[MAXV], base[MAXV];
    bool inpath[MAXV], inblossom[MAXV], inqueue[MAXV];
    int ady[2*MAXE], next[2*MAXE], last[MAXV];
    void initialize(int __nodes){
        V = __nodes;
        edges = 0;
        memset(last, -1, sizeof(int)*(V + 1));
    }
    void addEdge(int u, int v){
          ady[edges] = v;
          next[edges] = last[u]; last[u] = edges++;
          ady[edges] = u;
          next[edges] = last[v]; last[v] = edges++;
    }
    inline void push(int u){
          que[tail++] = u;
          inqueue[u] = true;
    }
    int findCommonAncestor(int u, int v){
          memset(inpath, 0, sizeof(inpath));
          while (true){
              u = base[u];
              inpath[u] = true;
              if (u == start) break;
              u = father[ match[u] ];
          }
          while (true){
              v = base[v];
              if (inpath[v]) break;
              v = father[ match[v] ];
          }
          return v;
    }
    void resetTrace(int u, int newbase){
          while (base[u] != newbase){
              int v = match[u];
                    inblossom[ base[u] ] = true;
                    inblossom[ base[v] ] = true;
                    u = father[ v ];
                    if (base[u] != newbase) father[u] = v;
          }
    }

    void blossomContract(int u, int v){
          int newbase = findCommonAncestor(u, v);
          memset(inblossom, false, sizeof(inblossom));

          resetTrace(u, newbase);
          resetTrace(v, newbase);

          if (base[u] != newbase) father[u]= v;
          if (base[v] != newbase) father[v]= u;
```

```
        for (int i = 1; i <= V; i++)
            if (inblossom[ base[i] ]){
                base[i] = newbase;
                if (!inqueue[i])
                    push(i);
            }
    }
    void find_augmenting_path(){
        memset(inqueue, false, sizeof(inqueue));
        memset(father, 0, sizeof(father));
        for (int i = 1; i <= V; i++) base[i] = i;

        head = 0, tail = 0;
        push(start);
        finish = 0;

        while (head < tail){
            int u = que[head++];
            for(int i = last[u]; i != -1; i = next[i]){
                int v = ady[i];
                if ((base[u] != base[v]) && (match[u] != v)) {
                    if ((v == start)||((match[v] > 0) && (father[match[v]] > 0))){
                        blossomContract(u, v);
                        continue;
                    }
                    if (father[v] == 0){
                        father[v] = u;
                        if (match[v] > 0) push(match[v]);
                        else{
                            finish = v;
                            return;
    }   }   }   }   }

    void augment_path(){
        int u = finish;
        while (u > 0){
            int v = father[u];
            int w = match[v];
            match[v] = u;
            match[u] = v;
            u = w;
        }
    }
```

```
    int edmonds(){
        memset(match, 0, sizeof(match));
        for (int i = 1; i <= V; i++) if (!match[ i ]){
            start = i;
            find_augmenting_path();
            if (finish > 0)     augment_path();
        }
        int ans = 0;
        for(int i=1; i<=V; i++)if(match[i] > 0) ans++;
        return ans/2;
    }
};

//Hungarian
int N,A[MAXN+1][MAXN+1],p,q, oo = 1 <<30;
int fx[MAXN+1],fy[MAXN+1],x[MAXN+1],y[MAXN+1];
int hungarian()
{
    memset(fx,0,sizeof(fx));
    memset(fy,0,sizeof(fy));
    memset(x,-1,sizeof(x));
    memset(y,-1,sizeof(y));
    for(int i = 0; i < N; ++i)
        for(int j = 0; j < N; ++j) fx[i] = max(fx[i],A[i][j]);
    for(int i = 0; i < N; )
    {
        vector<int> t(N,-1), s(N+1,i);
        for(p = q = 0; p <= q && x[i]<0; ++p)
            for(int k = s[p], j = 0; j < N && x[i]<0; ++j)
                if (fx[k]+fy[j]==A[k][j] && t[j]<0)
                {
                    s[++q]=y[j];
                    t[j]=k;
                    if(s[q]<0)
                        for(p=j; p>=0; j=p)
                            y[j]=k=t[j], p=x[k], x[k]=j;
                }
        if (x[i]<0)
        {
            int d = oo;
            for(int k = 0; k < q+1; ++k)
                for(int j = 0; j < N; ++j)
                    if(t[j]<0) d=min(d,fx[s[k]]+fy[j]-A[s[k]][j]);
            for(int j = 0; j < N; ++j) fy[j]+=(t[j]<0?0:d);
            for(int k = 0; k < q+1; ++k) fx[s[k]]-=d;
```

```
        }
        else ++i;
    }
    int ret = 0;
    for(int i = 0; i < N; ++i) ret += A[i][x[i]];
    return ret;
}

// Hopcroft – Karp O(M*sqrt(N))
const int MAXV = 1001;
const int MAXV1 = 2*MAXV;
int N,M;
vector<int> ady[MAXV];
int D[MAXV1],Mx[MAXV], My[MAXV];
bool BFS(){
    int u, v, i, e;
    queue<int> cola;
    bool f = 0;
    for (i = 0; i < N+M; i++) D[i] = 0;
    for (i = 0; i < N; i++)
        if (Mx[i] == -1) cola.push(i);
    while (!cola.empty()){
        u = cola.front(); cola.pop();
        for (e = ady[u].size()-1; e >= 0; e--) {
            v = ady[u][e];
            if (D[v + N]) continue;
            D[v + N] = D[u] + 1;
            if (My[v] != -1){
                D[My[v]] = D[v + N] + 1;
                cola.push(My[v]);
            }
            else f = 1;
        }
    }
    return f;
}
int DFS(int u){
    for (int v, e = ady[u].size()-1; e >=0; e--){
        v = ady[u][e];
        if (D[v+N] != D[u]+1) continue;
        D[v+N] = 0;
        if (My[v] == -1 || DFS(My[v])){
            Mx[u] = v;  My[v] = u;  return 1;
        }
    }
    return 0;
}
int Hopcroft_Karp(){
    int i, flow = 0;
    for (i = max(N,M); i >=0; i--) Mx[i] = My[i] = -1;
    while (BFS())
        for (i = 0; i < N; i++)
            if (Mx[i] == -1 && DFS(i))
                ++flow;
     return flow;
}
```

GRAFOS (OTROS)

```
/// Heavy Light Descomposition
int N, M;
vector<int> V[MN];
vector<int> G[MN];
vector<bool> L[MN];
/// cant- la cantidad de nodos
/// pos- la pos. donde aparece
/// nn- el nod en el cual aparece
/// pd- el link con el padre full superior
/// G-Dp
/// L-lazy
int cant[MN], pos[MN], nn[MN], pd[MN];
void Dfs( int nod, int pad ){
    int t = V[nod].size(), newn;
    if( t == 1 && nod != 1 ){
      pos[nod] = 0;
      nn[nod] = nod;
      cant[nod] = 1;
      pd[nod] = pad;
      return;
    }

    int mej = nod;
    for( int i = 0; i < t; i ++ ){
        newn = V[nod][i];
        if( newn == pad )
            continue;

        Dfs( newn, nod );
        if( cant[mej] < cant[nn[newn]] )
            mej = nn[newn];
```

```
    }

    pos[nod] = cant[mej];
    cant[mej] ++;
    nn[nod] = mej;
    pd[mej] = pad;
}
typedef pair<int, int> par;
typedef pair<int, par> tri;
typedef vector<tri> vt;
typedef vector<par> vp;
/// me da el recorrido desde a hasta b en vector<tri>
/// f posicion s.f in, s.f fin
vt rec( int a, int b ){
  vp A1, B1;
  A1.clear(), B1.clear();
  for( int i = a; i != -1; i = pd[nn[i]] )
       A1.push_back( par( nn[i], pos[i] ) );
  for( int i = b; i != -1; i = pd[nn[i]] )
       B1.push_back( par( nn[i], pos[i] ) );

  vt C1;
  C1.clear();
  reverse( A1.begin(), A1.end() );
  reverse( B1.begin(), B1.end() );
  int t = 0;
  while( t < (int)A1.size() && t < (int)B1.size() && A1[t] == B1[t] )
    t ++;
  if( t >= (int)A1.size() || t >= (int)B1.size() || ( t < (int)B1.size()
       && t < (int)A1.size() && A1[t].first != B1[t].first ) )
            t --;
  if( (t <(int) A1.size() && t < (int)B1.size()) && A1[t].first == B1[t].first ){
     C1.push_back( tri( A1[t].first, par( min( A1[t].second, B1[t].second ),
                              max( A1[t].second, B1[t].second ) ) ) );
     t ++;
  }
  for( int i = t; i <(int) A1.size(); i ++ )
    C1.push_back( tri( A1[i].first, par( A1[i].second, cant[A1[i].first] - 1 )));
  for( int i = t; i < (int)B1.size(); i ++ )
    C1.push_back( tri( B1[i].first, par( B1[i].second, cant[B1[i].first] - 1 )));

  return C1;
}
```

```
void havy_light( ){
  Dfs( 1, -1 ); // root

  for( int i = 1; i <= N; i ++ )/// rellenar con 4*cant
     if( cant[i] ){
       G[i] = vector<int> ( cant[i]*4, 0 );
       L[i] = vector<bool> ( cant[i]*4, false );
       G[i][1] = cant[i], L[i][1] = true;
      }
}

//Puentes y Puntos de Articulación
void bridges_PtoArt ( int nod ){
 int newn, num;
 vector<int>::iterator it;
 Td[nod] = low[nod] = ++ k;

 for(it = V[nod].begin(); it != V[nod].end(); it ++){
  num = *it;
  newn = G[num].nextn( nod );

  if( G[num].band )
    continue;

  G[num].band = true;

  if( Td[newn] ){
    low[nod] = min( low[nod], Td[newn] );
    continue;
  }

  bridges_PtoArt( newn );
  low[nod] = min( low[nod], low[newn] );

  if(Td[nod] < low[newn])
     puente.push(par( nod, newn ));

  if((Td[nod] == 1 && Td[newn] > 2)||(Td[nod] != 1 && Td[nod] <= low[newn]))
     Punto_art[nod] = true;
 }
}

//Componentes Biconexas
void BCC ( int nod ){
 Td[nod] = Low[nod] = ++ k;
```

```
 int newn, id;
 vector<int>::iterator it;
 for( it = V[nod].begin(); it != V[nod].end(); it ++ ){
  id = *it;
  newn = G[id].nextn( nod );

  if( !mark[id] ){
   P.push( id );
   mark[id] = true;
  }
  if( Td[newn] ){
   Low[nod] = min( Low[nod], Td[newn] );
   continue;
  }
  BCC( newn );
  Low[nod] = min( Low[newn], Low[nod] );
  if( Td[nod] <= Low[newn] ){
   num ++;
   while( !CB[id] ){
    CB[P.top()] = num;
    P.pop();
}}}}

//Componentes Fuertemente Conexas
void Tarjan_SCC( int nod ){
 int newn;
 vector<int>::iterator it;
 Td[nod] = low[nod] = ++ k;
 P.push( nod );

 for(it = V[nod].begin(); it != V[nod].end(); it ++){
  newn = *it;

  if( Td[newn] ){
   if( !mark[newn] )
    low[nod] = min( low[nod], Td[newn] );
   continue;
  }

  Tarjan_SCC( newn );
  low[nod] = min( low[nod], low[newn] );
 }

 if( low[nod] == Td[nod] ){
  sol ++;
  printf("SCC %d: ", sol);
  while( !mark[nod] )
  {
   printf("%d ", (int)P.top());
   mark[(int)P.top()] = true;
   P.pop();
  }

  printf("\n");
 }
}
```

DATA STRUCTURES

```
//Segment Tree Persistente
const int N = 100000 + 100, LOGN = 20;
const int TOT = 4*N + N*LOGN;

int sum[TOT], L[TOT], R[TOT];
int sz = 1;
int newNode(int s = 0){
    sum[sz] = s;
    return sz++;
}
int build(int b, int e){
   if(b == e) return newNode();

   int mid =(b + e) >> 1;
   int cur = newNode();
   L[cur] = build(b, mid);
   R[cur] = build(mid+1, e);

   return cur;
}

int update(int node, int b, int e, int p){
   if(b == e) return newNode(sum[node] + 1);

   int mid = (b + e) >> 1;
   int cur = newNode();

   if(p <= mid){
      L[cur] = update(L[node], b, mid, p);
      R[cur] = R[node];
   }
```

```
    else{
        R[cur] = update(R[node], mid+1, e, p);
        L[cur] = L[node];
    }
    sum[cur] = sum[L[cur]] + sum[R[cur]];

    return cur;
}

int query(int node1, int node2, int b, int e, int k){
    if(b == e) return b;

    int s = sum[L[node2]] - sum[L[node1]];
    int mid = (b + e) >> 1;

    if(s >= k) return query(L[node1], L[node2], b, mid, k);
    else return query(R[node1], R[node2], mid+1, e, k-s);
}
int root[N];

int main()
{
    int n, m;
    cin >> n >> m;

    root[0] = build(1, n);
    vector<int> v(n), tmp(n);

    for(int i = 0; i < n; ++i){
        cin >> v[i];   tmp[i] = v[i];
    }

    sort(tmp.begin(), tmp.end());
    tmp.resize(unique(tmp.begin(), tmp.end()) - tmp.begin());

    for(int i = 0; i < n; ++i)
        root[i+1] = update(root[i], 1, n, lower_bound(tmp.begin(),
                                tmp.end(), v[i]) - tmp.begin() + 1);

    while(m--){
        int i, j, k;    cin >> i >> j >> k;
        cout << tmp[query(root[i-1], root[j], 1, n, k)-1] << endl;
    }
}
```

```
//Suma de intervalos con BIT

void updater( int x, int v ){
 int tmp = x-1;
 for( ; x <= N; x += (x&-x) ){
    Dp[1][x] += v, Dp[2][x] += v*tmp;
 }
}
int sum( int p, int x ){
 int s = 0;
 for( ; x >= 1; x -= (x&-x) )
    s += Dp[p][x];
 return s;
}
int sumsum( int a ){
  return sum( 1, a )*a - sum( 2, a );
}
void updater_interv( int a, int b, int v ){
 updater( a, v ), updater( b+1, -v );
}

//Splay Trees
struct splay_tree{

const int inf = 1e9;
struct nodo {
    int size, cant[30];
    nodo *l, *r, *p;
    bool inv;
    int laz, let;
    nodo(nodo *f=0, nodo *i = 0, nodo *d = 0){
        l = i, p = f, r = d, size = 1, let = 0, laz = -1, inv = false;
        for(int i=0; i<30; i++) cant[i]=0;
    }
} *root;
splay_tree(){ root = NULL; }
inline void zig(nodo *x) {
    nodo *y = x->p, *z = y->p;
    y->l = x->r;
    if( x->r )
        x->r->p = y;
    x->p = z;
    if( z )
        if (z->l == y)z->l = x; else z->r = x;
    y->p = x, x->r = y;
```

```
    updata(y);
}
inline void zag(nodo *x) {
    nodo *y = x->p, *z = y->p;
    y->r = x->l;
    if( x->l )
        x->l->p = y;
    x->p = z;
    if( z )
        if (z->l == y) z->l = x; else z->r = x;
    y->p = x, x->l = y;
    updata(y);
}
inline void splay(nodo *x) {
    for (; x->p ;) {
        nodo *y = x->p, *z = y->p;
        if (!z) {
            if (y->l == x) zig(x); else zag(x);
        } else {
            if (z->l == y){
                if (y->l == x)    zig(y), zig(x);
                else zag(x), zig(x);
            }
            else if (y->r == x) zag(y), zag(x);
            else zig(x), zag(x);
        }
    }
    root = x, updata(root);
}
void find(int x) {
    if(!root)return;
    nodo *p = root;
    for( ; ; ) {
        lazy( p );
        int izq = (p->l)?p->l->size:0;
        if (x == izq + 1) break;
        if (x > izq + 1){
            x -= izq + 1;
            if ( p->r ) p = p->r; else break;
        }
        else
            if ( p->l ) p = p->l; else break;
    }
    splay(p);
}
```

```
inline void insertpos( int a, int b ){
nodo *nn = new nodo( 0, 0, 0 );
    nn->let = b;
    find( a );
    if( !root ){ root = nn, updata(root); return; }
    nodo *p = root;
    root = root->r;
    if( root )
        root->p = 0;
    p->r = nn, nn->p = p;
    find( -inf );
    nn->r = root;
    if( root )
        root->p = nn;
    root = p;
    updata(nn), updata(root);
}

inline void insert(int a) {
    nodo *p = root, *f=0;
    while(p){ f=p; p = p->r; }
    p = new nodo(f, 0, 0);
    p->let = a;
    if( f )
        f->r = p;
    splay(p);
}
inline splay_tree split(int x){
    if(!root) return splay_tree();
    splay_tree L = splay_tree();
    find(x);
    if( root->l )
        root->l->p=0;
    L.root = root->l, root->l=0;
    updata(root);
    return L;
}

inline void join(splay_tree L){
    if( !L.root ) return;

    if(!root) root = L.root;
    else {
        find(-inf);
```

```
        root->l = L.root, root->l->p = root;
        updata(root);
    }
    L.root = NULL;
}

void print(nodo *r){
    if(r == NULL)return;
    lazy(r);
    print(r->l);
    printf("%c ", r->let);
    print(r->r);
}
void erase(int x) {
    find(x);
    if(!root)return;

    if (!root->l) {
        nodo *tmp = root;
        root = root->r;
        if(root)
            root->p = 0;
        delete tmp;
    } else {
        nodo *t = root->r, *tmp = root;
        root = root->l;
        if(root)root->p = 0;
        find(x);
        if(root)root->r = t;
        if( t ) t->p = root;
        updata(root);
        delete tmp;
    }
}
void clear( nodo*x ){
    if( x ) return;
    clear( x->l );
    clear( x->r );
    delete x;
}
inline void updata(nodo *x) {
    x->size = ((x->l)?x->l->size:0) + ((x->r)?x->r->size:0) + 1;
    for(int i = 0; i < 30; i++)
        x->cant[i] = ((x->l)?x->l->cant[i]:0)+((x->r)?x->r->cant[i]:0)+(x->let == i);
}
inline void lazy(nodo *p){
    if(!p)return;
    if(p->inv){
        swap(p->r, p->l);
        if( p->r ) p->r->inv = !p->r->inv;
        if( p->l ) p->l->inv = !p->l->inv;
        p->inv=0;
    }
    if(p->laz!=-1){
        updlazy(p->l, p->laz);
        updlazy(p->r, p->laz);
        p->laz = -1;
    }
}
inline void updlazy(nodo *p, int laz){
    if( !p ) return;
    p->laz = laz;
    for(int i=0; i<30; i++)
        if(i==p->laz) p->cant[i] = p->size;
        else p->cant[i] = 0;
    p->let = laz;
}
void solve(char opt, int a, int b, int c = 0 ){
    splay_tree t1 = split( a );
    splay_tree t = split( b - a + 2 );

    if(opt=='S') t.updlazy(t.root, c);
    else if( opt == 'R' ) t.root->inv = ( !t.root->inv );
    else printf("%d\n", t.root->cant[c]);

    join(t);
    join(t1);
}
} ST; //Fin de Struct Splay Tree
```

DYNAMIC PROGRAMMING

```
//Convex Hull - Trick
const int MAX = 1e3;
typedef long long i64;
int num, last;
i64 M[MAX], B[MAX];

bool bad( int l1, int l2, int l3 ){
```

```
 return ( B[l3] - B[l1] )*( M[l1] - M[l2] ) < ( B[l2] - B[l1] )*( M[l1] - M[l3] );
}

void add( i64 m, i64 b ){
  M[num] = m, B[num++] = b;
  while( num >= 3 && bad( num-3, num-2, num-1 ) ){
    M[num-2] = M[num-1];
    B[num-2] = B[num-1];
    num --;
  }
}

i64 query( int x ){
 if( last > num )
    return num-1;
 while( last < num-1 && M[last+1]*x + B[last+1] < M[last]*x + B[last] )
    last ++;
 return M[last]*x + B[last];
}

int N, K;

int main()
{
    pair<int, int> a[50005];
    pair<int, int> rect[50005];

    scanf("%d", &K);

    for( int i = 0; i < K; i ++ )
        scanf("%d%d", &a[i].first, &a[i].second);

    sort( a, a + K );

    for( int i = 0; i < K; i ++ ){
        while( N > 0 && rect[N-1].second <= a[i].second )
            N --;

        rect[N++] = a[i];
    }

    i64 cost;
    num = last = 0;
    add( rect[0].second, 0 );
    for( int i = 0; i < N; i ++ ){
        cost = query( rect[i].first );
        if( i+1 < N )
            add( rect[i+1].second, cost );
    }

    printf("%I64d\n", cost);

    return 0;
}
```