

# White Book

Joao Carreira

July 10, 2009

## Contents

<b>1</b>	<b>Strings</b>	<b>2</b>	<b>4</b>	<b>Geometric Algorithms</b>	<b>10</b>
1.1	KMP	2	4.1	Dot Product	10
1.2	Range Minimum Query	2	4.2	Cross Product	10
1.3	Nth Permutation	2	4.3	Point on segment	10
<b>2</b>	<b>Dynamic Programming</b>	<b>2</b>	4.4	Intersection of segments	10
2.1	LCS (Longest Common Subsequence)	2	4.5	Point position relative to a (line) segment	10
2.2	LIS (Longest Increasing Subsequence)	3	4.6	Point distance to (line) segment	10
2.3	MCM (Matrix Chain Multiplication)	3	4.7	Polygon's Area	11
2.4	Knapsack	3	4.8	Convex Hull	11
2.5	Counting Change	4	4.9	Closest pair of points	11
2.6	Coin Changing	4	4.10	Test if point is inside a polygon	12
2.7	Biggest Sum	4	4.11	Intersection of rectangles	12
2.8	Edit Distance	4	4.12	Circle from 3 points	12
2.9	Integer Partitions	5	<b>5</b>	<b>Numerical</b>	<b>12</b>
2.10	Box Stacking	5	5.1	Choose	12
2.11	Building Bridges	5	5.2	Module	13
2.12	Partition Problem	5	5.3	LCM / GCD	13
2.13	Balanced Partition	5	5.4	Base conversion	13
<b>3</b>	<b>Graphs</b>	<b>6</b>	5.5	Horner's Rule	13
3.1	Find an Eulerian Path	6	5.6	Big Mod	13
3.2	Check if there is an Hamiltonian Path	7	5.7	Matrix Multiplication	13
3.3	Breadth First Search	7	5.8	Ternary Search	14
3.4	DFS/TopSort	7	5.9	Long Arithmetic	14
3.5	Prim's Algorithm	7	5.10	Infix para Postfix	14
3.6	Dijkstra	8	5.11	Calculate Postfix expression	15
3.7	Kth Shortest Paths	8	5.12	Postfix to Infix	15
3.8	Floyd-Warshall	9	5.13	Catalan Numbers	15
3.9	Detecting Bridges	9	5.14	Fibonnaci	15
3.10	Finding a Loop in a Linked List	9			
3.11	Tree diameter	9			
3.12	State equivalence	9			
3.13	Union Find	9			

# 1 Strings

## 1.1 KMP

```
void init_mp(char *str, int len, int next[]){
    next[0]=-1;
    for(int i=0,j=-1;i<len;i++,j++,next[i]=j){
        while(j>=0 && str[i]!=str[j])
            j=next[j];
    }
}
```

```
void init_mp(char *pat, int next[]){
    int i = 0, j = next[0]=-1;
    while (pat[i]) {
        while (j > -1 && pat[i] != pat[j])
            j = next[j];
        i++; j++;
        if (pat[i] == pat[j])
            next[i] = next[j];
        else next[i] = j;
    }
}
```

## 1.2 Range Minimum Query

```
int main() {
    int N,Q,i,j,k;

    scanf("%d %d",&N,&Q);

    for (i=0;i<N;i++)
        scanf("%d",&n[i]);

    for (i=0;i<N;i++)
        m[i][0]=M[i][0]=n[i];

    for (i=1;(1<<i)<=N;i++) {
        for (j=0;j+(1<<i)-1<N;j++) {
            m[j][i]=min(m[j][i-1],m[j+(1<<(i-1))][i-1]);
            M[j][i]=max(M[j][i-1],M[j+(1<<(i-1))][i-1]);
        }
    }

    for (k=0;k<Q;k++) {
        scanf("%d %d",&i,&j);
        i--;j--;
        int t,p;
        t=(int)(log(j-i+1)/log(2));
        p=1<<t;
```

```
        printf("%d\n",max(M[i][t],M[j-p+1][t])
            - min(m[i][t], m[j - p + 1][t]));
    }
    return 0;
}
```

$$M[i][j] = \max(M[i][j-1], M[i+2^{j-1}][j-1])$$
$$RMQ_A(i, j) = \max(M[i][k], M[j-2^k+1][k])$$

## 1.3 Nth Permutation

```
/*
 * determine the kth permutation of s
 * k = 0,1,2...
 */
string permutation(long long int k, string s){
    int n = s.size()-1, factorial=1;
    s = "a" + s; // hack
    for (int j = 2;j <= n - 1; j++)
        factorial *= j;
    for (int j = 1;j <= n - 1; j++) {
        int tempj = (k / factorial) % (n + 1 - j);
        int temps = s[j + tempj];
        for( int i = j + tempj; i >= j + 1; i--)
            s[i] = s[i - 1];
        s[j] = temps;
        factorial = factorial / (n - j);
    }
    return s.substr(1); //hack again
}
```

# 2 Dynamic Programming

## 2.1 LCS (Longest Common Subsequence)

```
int L[MAX][MAX] = {{0}};
int LCS(char A[], char B[]) {
    for (int i = m;i >= 0; i--) { /* m = strlen(A) */
        for (int j = n;j >= 0; j--) { /* n = strlen(B) */
            if (!A[i] || !B[j])
                L[i][j] = 0;
            else if (A[i] == B[j])
                L[i][j] = 1 + L[i+1][j+1];
            else L[i][j] = max(L[i+1][j], L[i][j+1]);
        }
    }
    return L[0][0];
}
```

```

int LCSString(int L[MAX][MAX]) {
    int i, j;
    i = j = 0;
    while (i < m && j < n) {
        if (A[i] == B[j]) {
            poe A[i] no fim da str-solucao
            i++; j++;
        }
        if (L[i+1][j] >= L[i][j+1]) i++;
        else j++;
    }
}

```

## 2.2 LIS (Longest Increasing Subsequence)

```

int pred[MAX_SIZE], lasti;
int LIS(int C[], int n) {
    int s[MAX_SIZE], max=INT_MIN;
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; i++) {
            if (C[i] > C[j] && s[i] <= s[j]) {
                pred[i] = j;
                if ((s[i] = s[j] + 1) > max)
                    lasti = i;
                max = s[i];
            }
        }
    }
    return max;
}

```

```

void PrintLIS() {
    int i, j, aux[MAX_SIZE];
    for (j = max-1, i = lasti; j >= 0; j--) {
        aux[j] = C[i];
        i = pred[i];
    }

    for (j = 0; j < max; j++)
        printf("%d\n", aux[j]);
}

```

## 2.3 MCM (Matrix Chain Multiplication)

```

int Calc(int i, int j) {
    int res = INT_MAX;
    for (k = i; k < j; k++) {
        tmp = m[i][k] + m[k+1][j] +
            Line[i] * Col[k] * Col[j];
    }
}

```

```

    if (tmp < res) {
        res = tmp;
        s[i][j] = k;
    }
}
return res;
}

void MCM() {
    int i, j, n = 3;
    for (i = 0; i < n; i++)
        m[i][i] = 0;

    for (i = n-1; i >= 0; i--)
        for (j = i + 1; j <= n; j++)
            m[i][j] = Calc(i, j);
}

//PrintMCM(0,N-1);
void PrintMCM(int i, int j) {
    if (i == j) printf("%d", i);
    else {
        putchar('(');
        PrintMCM(i, s[i][j]);
        putchar('*');
        PrintMCM(s[i][j] + 1, j);
        putchar(')');
    }
}

```

## 2.4 Knapsack

```

int n[WSIZE][ISIZE] = {{0}}
/*
 * put one zero in weight and value;
 * ex: weight={>0<,3,4,5} & value={>0<,3,4,5,6};
 */
int knapsack(int items, int W,
    int value[], int weight[]){
    for (int i = 1; i <= items; i++) {
        for (int j = 0; j <= W; j++) {
            if (weight[i] <= j) {
                if (value[i] + n[i-1][j-weight[i]]
                    > n[i-1][j])
                    n[i][j] = value[i] +
                        n[i-1][j-weight[i]];
                else n[i][j]=n[i-1][j];
            } else n[i][j]=n[i-1][j];
        }
    }
}

```

```

    return n[items][W];
}

void print_sequence(int items, int W, int weight[]) {
    int i = items, k = W;
    while (i > 0 && k > 0) {
        if (n[i][k] != n[i-1][k]) {
            printf("item %d is in\n", i);
            k = k-weight[i-1];
        }
        i--;
    }
}

```

## 2.5 Counting Change

```

int coins[] = {50,25,10,5,1};
int CoinChange(int n) {
    table[0] = 1;
    for (i = 0; i < 5; i++) {
        c = coins[i];
        for (j = c; j <= n; j++)
            table[j] += table[j - c];
    }
    return table[n];
}

```

## 2.6 Coin Changing

```

int n[10000], i, N, coins[]={50,25,10,5,1}, k;
int main() {
    scanf("%d", &N);
    for (int i = 0; i <= N; i++) n[i] = INT_MAX;
    n[0] = 0;
    for (int i = 0; i < 5; i++)
        for (k = 0; k <= N - coins[i]; k++)
            n[k + coins[i]] =
                min(n[k] + 1, n[k + coins[i]]);
    printf("%d\n", n[N]);
    return 0;
}

```

## 2.7 Biggest Sum

```

#define SIZE 20000
int n[SIZE];

int main() {
    int k, s, b;
    int xl, xr, best, prevx;
}

```

```

cin>>k; // size of input
for (int i = 1; i <= k; i++) {
    xr = xl = 0;

    cin >> s;
    for (int j = 0; j < s - 1; j++)
        cin >> n[j];

    prevx = xl = xr = 0;
    best = b = n[0];
    for (int j = 1; j < s - 1; j++) {
        if (b < 0)
            prevx = j;
        b = n[j] + max(0, b);
        if (b > best ||
            (b == best && j - prevx > xr - xl)) {
            xl = prevx;
            xr = j;
            best = b;
        }
    }
    if (best > 0)
        // best is the solution
    }
    return 0;
}

```

## 2.8 Edit Distance

1. Delete a character
2. Insert a new character
3. Replace a letter

```

int DE(char *str1, char *str2) {
    int n[SIZE][SIZE];
    int i, j, value;

    for (i = 0; i <= str1_len; i++) n[i][0] = i;
    for (j = 0; j <= str2_len; j++) n[0][j] = j;

    for (i = 1; i <= str1_len; i++) {
        for (j = 1; j <= str2_len; j++) {
            value = (str1[i - 1] != str2[j - 1]);

            n[i][j] = min(n[i - 1][j - 1] + value,
                n[i - 1][j] + 1,
                n[i][j - 1] + 1);
        }
    }
}

```

```

    }
    return n[str1_len][str2_len];
}

T(i,j) = \min(C_d + T(i-1,j),
             T(i, j-1) + C_i,
             T(i-1, j-1) + (A[i]==B[j] ? 0 : C_r))

```

## 2.9 Integer Partitions

$P(n)$  represents the number of possible partitions of a natural number  $n$ .  $P(4) = 5, 4, 3 + 1, 2 + 2, 2 + 1 + 1, 1 + 1 + 1 + 1$

$P(0) = 1$

$P(n) = 0, n < 0$

$P(n) = p(1, n)$

$p(k, n) = p(k + 1, n) + p(k, n - k)$

$p(k, n) = 0$  if  $k > n$

$p(k, n) = 1$  if  $k = n$

## 2.10 Box Stacking

A set of boxes is given.  $Box_i = h_i, w_i, d_i$ .

We can only stack box  $i$  on box  $j$  if  $w_i < w_j$  and  $d_i < d_j$ .

To consider all the orientations of the boxes, replace each box with 3 boxes such that  $w_i \leq d_i$  and  $box_1[0] = h_i, box_2[0] = w_i, box_3[0] = d_i$ .

Then, sort the boxes by decreasing  $area(w_i * d_i)$ .

$H(j)$  = tallest stack of boxes with box  $j$  on top.

$H(j) = \max_{i < j \& w_i > w_j \& d_i > d_j} (H(i)) + h_j$

Check  $H(j)$  for all values of  $j$ .

## 2.11 Building Bridges

Maximize number of non-crossing bridges. Ex:

bridge1: 2, 5, 1,  $n, \dots, 4, 3$

bridge2: 1, 2, 3, 4,  $\dots, n$

Let  $X(i)$  be the index of the corresponding city on northern bank.  $X(1) = 3, X(2) = 1, \dots$

Find longest increasing subsequence of  $X(1), \dots, X(n)$ .

## 2.12 Partition Problem

**Input:** A given arrangement  $S$  of non-negative numbers  $s_1, \dots, s_n$  and an integer  $k$ .

**Output:** Partition  $S$  into  $k$  ranges, so as to minimize the maximum sum over all the ranges.

```

int M[1000][100], D[1000][100];
void partition_i(vector<int> &v, int k) {

```

```

    int p[1000], i, n = v.size();
    v.insert(v.begin(), 0);
    p[0] = 0;
    for(i = 1; i < v.size(); i++)
        p[i] = p[i - 1] + v[i];

    for (i = 1; i <= n; i++)
        M[i][1] = p[i];
    for (i = 1; i <= k; i++)
        M[1][i] = v[1];
    for (i = 2; i <= n; i++) {
        for (int j = 2; j <= k; j++) {
            M[i][j] = INT_MAX << 1 - 1;
            int s = 0;
            for (int x = 1; x <= i - 1; x++) {
                s = max(M[x][j-1], p[i] - p[x]);
                if (M[i][j] > s) {
                    M[i][j] = s;
                    D[i][j] = x;
                }
            }
        }
    }
    printf("%d\n", M[n][k]);
}

//n = number of elements of the initial set
void reconstruct_partition(
    const vector<int> &S, int n, int k) {
    if (k == 1) {
        for (int i = 1; i <= n; i++)
            printf("%d ", S[i]);
        putchar('\n');
    } else {
        reconstruct_partition(S, D[n][k], k - 1);
        for (int i = D[n][k] + 1; i <= n; i++)
            printf("%d ", S[i]);
        putchar('\n');
    }
}

```

## 2.13 Balanced Partition

```

enum {DONT_GET, GET};
char **sol, **P;

// return 1 if there is a subset of v0...vi with sum j
// 0 otherwise
int calcP(int i, int j, const vi &v) {
    if (i < 0 || j < 0) return 0;

```

```

if (P[i][j] != -1) return P[i][j];

if (j == 0) { // trivial case
    sol[i][j] = DONT_GET; return P[i][j] = 1;
}
if (v[i] == j) {
    sol[i][j] = GET;
    return P[i][j] = 1;
}

int res1 = calcP(i - 1, j, v);
int res2 = calcP(i - 1, j - v[i], v);
if (res1 >= res2)
    P[i][j] = res1, sol[i][j] = DONT_GET;
else P[i][j] = res2, sol[i][j] = GET;
return P[i][j];
}

```

```

// v is the vector of values
// k is the maximum value in v
// sum is the sum of all elements in v
void balanced_partition(vi &v, int k, int sum) {
    P = new char*[v.size()];
    sol = new char*[v.size()];
    for (int i = 0; i < v.size(); i++) {
        P[i] = new char[k * v.size() + 1];
        sol[i] = new char[k * v.size() + 1];
        for (int j = 0; j < k * v.size() + 1; j++)
            P[i][j] = -1, sol[i][j] = DONT_GET;
    }
    for (int i = 0; i < v.size(); i++)
        for (int j = 0; j < v.size() * k + 1; j++)
            calcP(i, j, v);
    //calcP(v.size() - 1, sum/2, v);

    int S = sum / 2;
    if (sum & 1 || !P[v.size() - 1][S])
        cout << "ERROR" << endl;
    else cout << "SUCCESS" << endl;
}

```

```

void free_mem(vi& v) {
    for (int i = 0; i < v.size(); i++) {
        delete P[i]; delete sol[i];
    }
    delete[] P;
    delete[] sol;
}

```

```

// get_solution(v.size() - 1, accumulate(v.begin(), v.end(), 0), v);
// v1, v2, v);
void get_solution(int i, int j,
                  vi &S1, vi &S2, vi &v) {
    if (j < 0 || i < 0) return;
    if (sol[i][j] == GET) {
        S1.push_back(v[i]);
        return get_solution(i - 1, j - v[i], S1, S2, v);
    } else {
        S2.push_back(v[i]);
        return get_solution(i - 1, j, S1, S2, v);
    }
}

```

## 3 Graphs

### 3.1 Find an Eulerian Path

```

stack<int> s;
vector<list<int>> >adj;

void remove_edge(int u, int v) {
    for (list<int>::iterator it=adj[u].begin();
         it != adj[u].end(); it++) {
        if (*it == v) {
            it = adj[u].erase(it);
            return;
        }
    }
}

int path(int v) {
    int w;
    for (;adj[v].size();v = w) {
        s.push(v);
        list<int>::iterator it = adj[v].begin();
        w = *it;;
        remove_edge(v,w);
        remove_edge(w,v);
        edges--;
    }
    return v;
}

//u - source, v-destiny
int eulerian_path(int u, int v) {
    printf("%d\n", v);
    while (path(u) == u && !s.empty()) {
        printf("%d", u = s.top());
    }
}

```

```

    s.pop();
}
return edges == 0;
}

```

### 3.2 Check if there is an Hamiltonian Path

$O(n^2 2^n)$

```

//initially:
// u - dest
// seen = 1 << u
bool memo[20][1 << 20];
void hpath( int u, int seen ) {
    if ( memo[u][seen] ) return;
    memo[u][seen] = true;

    if ( u == t ) { /* check that seen == (1<<n)-
1 (seen every vertex) */ }
    else {
        for ( int v = 0; v < n; v++ )
            if ( !( seen & ( 1 << v ) ) && graph[u][v] )
                hpath( v, seen | ( 1 << v ) );
    }
}

```

### 3.3 Breadth First Search

```

bool adj[N][N];
int colour[N], d[N], p[N];
void bfs() {
    queue<int> q;
    int i, source = 0;

    for ( i = 0; i < N; i++){
        d[i] = INF;
        p[i] = -1;
        colour[i] = WHITE;
    }

    d[source] = 0;
    colour[source] = GRAY;
    q.push(source);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 0; v < N; v++){
            if (colour[v] == WHITE && adj[u][v]) {
                colour[v] = GRAY;
                d[v] = d[u]+1;
            }
        }
    }
}

```

```

        p[v] = u;
        q.push(v);
    }
}
colour[u] = BLACK;
}
}
}

```

### 3.4 DFS/TopSort

$O(V + E)$

```

void dfs(int u) {
    colour[u] = GRAY;
    for (int v = 0; v < N; v++) {
        if (colour[v] == WHITE && adj[u][v]) {
            p[v] = u;
            dfs(v);
        }
    }
    colour[u] = BLACK;
    //put node in front of a list if topsort
}

```

#### Maximum Spanning Tree:

Negate all the edge weights and determine the minimum spanning tree.

#### Minimum Product Spanning Tree:

Replace all the edge weights with their logarithm

### 3.5 Prim's Algorithm

```

double Prim(int start,int nvert) {
    bool in[N];
    double dist[N];
    int p[N], i, v;

    for(i = 0; i < nvert; i++){
        in[i] = false;
        dist[i] = INT_MAX;
        p[i] = -1;
    }

    dist[start] = 0;
    v = start;
    while (!in[v]) {
        in[v] = true;
        for(i = 0; i < nvert; i++){
            if (adj[v][i] && !in[i]){
                if (dist[i] > adj[v][i]){
                    dist[i] = adj[v][i];
                }
            }
        }
    }
}

```

```

        p[i] = v;
    }
}

double d = FLT_MAX;
for (i = 0; i < nvert; i++) {
    if (!in[i] && d > dist[i]) {
        v = i;
        d = dist[i];
    }
}
double res = 0;
for(i = 0; i < nvert; i++)
    res += dist[i];
return res;
}

```

### 3.6 Dijkstra

```

int dijkstra(int source, int dest,
             int nvert, int d[], int p[]) {
    bool in[N];
    int i, u, v;

    for (i = 0; i < nvert; i++) {
        in[i] = false;
        d[i] = INF;
        p[i] = -1;
    }
    d[source] = 0;
    u = source;
    while (!in[u]){
        in[u] = true;
        for (v = 0; v < nvert; v++) {
            if(adj[u][v] && d[v] > d[u]+adj[u][v]) {
                p[v]=u;
                d[v]=d[u]+adj[u][v];
            }
        }

        int dist=INF;
        for(i=0;i<nvert;i++) {
            if(!in[i] && d[i]<dist) {
                u=i;
                dist=d[i];
            }
        }
    }
}

```

```

    return d[dest];
}

```

### 3.7 Kth Shortest Paths

$O(Km)$

```

/*
 * u - source node
 * p - predecessor vector
 * h - vector of transformation
 * v - result vector
 */
#define vvi vector<vector<int> >
void path(int u, const vector<int> &p,
          const vector<int> &h, vector<int> &v) {
    if (u != -1) {
        path(p[u], p, h, v);
        v.push_back(h[u]);
    }
}

vvi dijkstra(int source, int dest, int K) {
    vector<int> count(SIZE), d(10000),
                p(10000), h(10000), X;
    vvi res;

    for (int i = 0; i < N; i++)
        p[i] = -1;
    int elm = 1;
    h[elm] = source;
    d[elm] = 0;
    X.push_back(elm);

    while (count[dest] < K && !X.empty()) {
        int ind = 0;
        for (unsigned int i = 1; i < X.size(); i++) {
            if (d[X[i]] < d[X[ind]])
                ind = i;
        }
        int k = X[ind];
        X.erase(X.begin() + ind);
        int i = h[k];

        count[i]++;
        if (i == dest) {
            vector<int> v;
            path(k, p, h, v);
            res.push_back(v);
        }
    }
}

```

```

if (count[i] <= K) {
    for (int j = 0; j < SIZE; j++) {
        if (adj[i][j]) {
            elm++;
            d[elm] = d[k] + adj[i][j];
            p[elm] = k;
            h[elm] = j;
            X.push_back(elm);
        }
    }
}
return res;
}

```

### 3.8 Floyd-Warshall

$O(n^3)$ ;

```

void floyd(int adj[NVERT][NVERT]){
    int i, j, k, through_k;
    for (k = 1; k <= NVERT; k++){
        for (i = 1; i <= NVERT; i++){
            for (j = 1; j <= NVERT; j++){
                through_k = adj[i][k] + adj[k][j];
                if (through_k < adj[i][j])
                    adj[i][j] = through_k;
            }
        }
    }
}

```

### 3.9 Detecting Bridges

```

int dfs(int u, int p) {
    colour[u] = 1;
    dfsNum[u] = num++;
    int leastAncestor = num;
    for (int v = 0; v < N; v++) {
        if (M[u][v] && v!=p) {
            if (colour[v] == 0) {
                int rec = dfs(v,u);
                if (rec > dfsNum[u])
                    cout<<"Bridge: "<<u<<" "<<v<<endl;
                leastAncestor = min(leastAncestor,rec);
            }
            else{
                leastAncestor = min(leastAncestor,
                    dfsNum[v]);
            }
        }
    }
}

```

```

}
}
colour[u] = 2;
return leastAncestor;
}
}

```

### 3.10 Finding a Loop in a Linked List

$O(n)$

```

// Best solution
function boolean hasLoop(Node startNode){
    Node slowNode, fastNode1, fastNode2;
    slowNode = fastNode1 = fastNode2 = startNode;
    while (slowNode && fastNode1 = fastNode2.next()
        && fastNode2 = fastNode1.next()){
        if (slowNode == fastNode1 ||
            slowNode == fastNode2)
            return true;
        slowNode = slowNode.next();
    }
    return false;
}

```

### 3.11 Tree diameter

Pick a root and start a DFS from it which returns both the diameter of the subtree and its maximum height. The diameter is the maximum of (left diameter, right diameter, left height + right height).

### 3.12 State equivalence

- Initially all states are assumed to be equivalent (equiv[i][j]=true for all i,j)
- Check each pair: if one of them is 'win' and the other 'loose' then equiv[i][j] = false
- Check each pair: if the transition from a given state with any possible value goes to a not equivalent state then these cannot be equivalent.
- Do this while equiv[][] is modified. The 'non equivalences' will be propagated.

### 3.13 Union Find

```

int Rank[SIZE];
int P[SIZE];

void create_set(int x) {
    P[x] = x;
    Rank[x] = 0;
}

```

```

}

void merge_sets(int x, int y) {
    int px = find_set(x);
    int py = find_set(y);
    if(Rank[px] > Rank[py])
        P[py] = px;
    else P[px] = py;

    if(Rank[px] == Rank[py])
        Rank[py]++;
}

int find_set(int x){
    if(x != P[px])
        P[x] = find_set(P[x]);
    return P[x];
}

void connected_components(){
    for each vertex i
        do create_set(i);

    for each edge (u,v)
        if(find_set(u) != find_set(v))
            merge_sets(u,v);
}

bool same_components(int u,int v){
    if(find_set(u) == find_set(v))
        return true;
    else return false;
}

```

## 4 Geometric Algorithms

### 4.1 Dot Product

```

int dot(int[] A, int[] B, int[] C) {
    int AB[2], BC[2];
    AB[0] = B[0]-A[0];
    AB[1] = B[1]-A[1];
    BC[0] = C[0]-B[0];
    BC[1] = C[1]-B[1];
    int dot = AB[0] * BC[0] + AB[1] * BC[1];
    return dot;
}

```

### 4.2 Cross Product

```

int cross(int[] A, int[] B, int[] C) {
    int AB[2], AC[2];
    AB[0] = B[0]-A[0];
    AB[1] = B[1]-A[1];
    AC[0] = C[0]-A[0];
    AC[1] = C[1]-A[1];
    int cross = AB[0] * AC[1] - AB[1] * AC[0];
    return cross;
}

```

### 4.3 Point on segment

A point is on a segment if its distance to the segment is 0. **Given two different points  $(x_1, y_1)$  and  $(x_2, y_2)$  the values of  $A, B,$  and  $C$  for  $Ax + By + C = 0$  are given by**

$$A = y_2 - y_1$$

$$B = x_1 - x_2$$

$$C = A * x_1 + B * y_1$$

### 4.4 Intersection of segments

```

double det = A1*B2 - A2*B1
if (det == 0) {
    //Lines are parallel
} else {
    double x = -(A1*C2 - A2*C1) / det
    double y = -(B1*C2 - B2*C1) / det
}

```

### 4.5 Point position relative to a (line) segment

```

//Input: three points P0, P1, and P2
//Ret: >0 for P2 left of the line through P0 and P1
//      = 0 for P2 on the line
//      < 0 for P2 right of the line
int isLeft( Point P0, Point P1, Point P2 ) {
    return ( (P1.x - P0.x) * (P2.y - P0.y)
            - (P2.x - P0.x) * (P1.y - P0.y) );
}

```

### 4.6 Point distance to (line) segment

If the line is in the form  $Ax + By + C = 0$ :

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

```

//Compute the dist. from AB to C
//if isSegment=true, AB is a seg., not a line.
double linePointDist(int[] A, int[] B,
                    int[] C, boolean isSegment) {
    double dist = cross(A,B,C) / distance(A,B);
    if (isSegment) {
        int dot1 = dot(A,B,C);
        if (dot1 > 0) return distance(B,C);
        int dot2 = dot(B,A,C);
        if (dot2 > 0) return distance(A,C);
    }
    return abs(dist);
}

```

#### 4.7 Polygon's Area

```

int area = 0;
/*int N = lengthof(p);*/

for (int i = 1; i + 1 < N; i++) {
    int x1 = p[i][0] - p[0][0];
    int y1 = p[i][1] - p[0][1];
    int x2 = p[i+1][0] - p[0][0];
    int y2 = p[i+1][1] - p[0][1];
    int cross = x1*y2 - x2*y1;
    area += cross;
}
return fabs(area/2.0);

```

#### 4.8 Convex Hull

```

// It returns the list of points in the convex hull
// in a counter-clockwise order
// Note: The last and the first points in
// the list are the same
#include <vector>
vector<point> ConvexHull(vector<point> P) {
    int n = P.size(), k = 0;
    vector<point> H(2*n);

    // Sort points lexicographically
    sort(P.begin(), P.end());

    // Build lower hull
    for (int i = 0; i < n; i++) {
        while (k >= 2 &&
              cross(H[k-2], H[k-1], P[i]) <= 0)
            k--;
        H[k++] = P[i];
    }
}

```

```

// Build upper hull
for (int i = n-2, t = k+1; i >= 0; i--) {
    while (k >= t &&
          cross(H[k-2], H[k-1], P[i]) <= 0)
        k--;
    H[k++] = P[i];
}

H.resize(k);
return H;
}

```

#### 4.9 Closest pair of points

```

double delta_m(vp &ql, vp &qr, double delta) {
    int i, j=0;
    double dm=delta;
    for (i=0; i<(int)ql.size(); i++) {
        point p=ql[i];

        while (j<(int)qr.size() && qr[j].y < p.y-delta)
            j++;

        int k=j;
        while (k<(int)qr.size()
              && qr[k].y<=p.y+delta) {
            dm=min(dm, dist(p,qr[k]));
            k++;
        }
    }
    return dm;
}

vp select_candidates(vp &p, int l, int r,
                   double delta, double midx) {
    vp n;
    for (int i=l; i<=r; i++) {
        if (abs(p[i].x-midx)<=delta)
            n.push_back(p[i]);
    }
    return n;
}

double closest_pair(vp &p, int l, int r) {
    if (r-l+1<2) return INF;
    int mid=(l+r)/2;
    double midx=p[mid].x;
    double dl=closest_pair(p, l, mid);
}

```

```

double dr=closest_pair(p,mid+1,r);
double delta=min(dl,dr);

vp ql,qr;
ql=select_candidates(p,l,mid,delta,midx);
qr=select_candidates(p,mid+1,r,delta,midx);

double dm=delta_m(ql,qr,delta);

vp res;
merge(p.begin()+l,p.begin()+mid+1,
      p.begin()+mid+1,p.begin()+r+1,
      back_inserter(res),cmp);
copy(res.begin(),res.end(),p.begin()+l);
return min(dm,min(dr,dm));
}

```

#### 4.10 Test if point is inside a polygon

```

//Input: P = ponto
//      V[] = vector de n+1 pontos, com V[n]=V[0]
//Return: wn = 0 only if P is outside V[]
int wn_PnPoly( Point P, Point* V, int n ) {
    int wn = 0; // the winding number counter

    // loop through all edges of the polygon
    for (int i=0; i<n; i++) {
        if (V[i].y <= P.y) {
            if (V[i+1].y > P.y)
                if (isLeft( V[i], V[i+1], P) > 0)
                    ++wn;
        }
        else {
            if (V[i+1].y <= P.y)
                if (isLeft( V[i], V[i+1], P) < 0)
                    --wn;
        }
    }
    return wn;
}

```

#### 4.11 Intersection of rectangles

```

x1 = y2 = INT_MIN;
x2 = y1 = INT_MAX;

for (i=0;i<NRECT;i++) { //for all rectangles
    cin>>ax>>ay>>bx>>by;
    if (ax>x1)x1=ax;
    if (ay<y1)y1=ay;

```

```

    if (bx<x2)x2=bx;
    if (by>y2)y2=by;
}

```

#### 4.12 Circle from 3 points

```

int main() {
    double ax,ay,bx,by,cx,cy,xres,yres;
    double xmid,ymid,A1,B1,C1,A2,C2,B2,dist;

    while (scanf("%lf %lf %lf %lf %lf %lf",
                &ax,&ay,&bx,&by,&cx,&cy)==6) {
        A1 = by - ay;
        B1 = ax - bx;
        xmid=min(ax,bx)+(max(ax,bx)-min(ax,bx))/2.0;
        ymid=min(ay,by)+(max(ay,by)-min(ay,by))/2.0;
        C1 = -B1 * xmid + A1 * ymid;

        B2 = bx - cx;
        A2 = cy - by;
        xmid=min(bx,cx)+(max(bx,cx)-min(bx,cx))/2.0;
        ymid=min(by,cy)+(max(by,cy)-min(by,cy))/2.0;
        C2 = -B2 * xmid + A2 * ymid;
        //intersection of segments
        intersection(A1,B1,C1,A2,B2,C2,&xres,&yres);
        dist = sqrt(pow(xres-bx,2)+pow(yres-by,2));
        printf("(x %s %.3lf)^2 + (y %s %.3lf)^2 =
        %.3lf^2\n",xres<0.0?"":"-",abs(xres),
        yres<0.0?"":"-",abs(yres),dist);
    }

    return 0;
}

```

## 5 Numerical

### 5.1 Choose

```

 $\binom{n}{k}$ 

long long memo[SIZE][SIZE]; //initialized to -1
long long binom(int n,int k){
    if(memo[n][k]!=-1)return memo[n][k];
    if(n<k)return 0;
    if(n==k)return 1;
    if(k==0)return 1;
    return memo[n][k]=binom(n-1,k)+binom(n-1,k-1);
}

```

## 5.2 Module

```
int mod(int a, int n) {
    return (a%n + n)%n;
}
```

## 5.3 LCM / GCD

```
int gcd(int a,int b){
    if(!b)
        return a;
    else return gcd(b,a%b);
}
```

```
struct triple{
    int gcd,x,y;
    int triple(int g=0,int a=0,int b=0):
        gcd(g),x(a),y(b){}
};
```

```
triple ExtendedEuclid(int a,int b){
    if(!b)
        return triple(a,1,0);

    triple t=ExtendedEuclid(b,a%b);
    return triple(t.gcd,t.y,t.x-(a/b)*t.y);
}
```

```
int LCM(int a,int b){
    return a*b/gcd(a,b);
}
```

## 5.4 Base conversion

```
void base(char *res, int num, int base){
    char tmp[100];
    int i, j;
    for (i = 0; num; i++) {
        tmp[i]="0123456789ABCDEFGHIJKLM"[num%base];
        num /= base;
    }
    tmp[i] = 0;
    for (i--, j = 0; i >= 0; i--, j++)
        res[j] = tmp[i];
    res[j] = 0;
}
```

## 5.5 Horner's Rule

$$P(x) = \sum_{k=0}^n a_k x^k = a_0 + x(a_1 + x(a_2 + \dots + (a_{n-1} + x1_n)))$$

```
double Horner(double coef[],int degree,int x) {
    double res = 0;
    int i;

    for (i = degree; i >= 0; i--)
        res = coef[i] + x * res;
    return res;
}
```

## 5.6 Big Mod

$(B^P) \% M$

```
long int bigmod(long long int B,
    long long int P, long long int M) {
    if (P == 0)
        return 1;
    else if(P & 1) {
        long long int tmp =
            bigmod(B,(P - 1) >> 1, M) % M;
        tmp = (tmp * tmp * B) % M;
        return tmp;
    }

    else{
        long long int tmp = bigmod(B, P >> 1, M) % M;
        return (tmp * tmp) % M;
    }
}
```

## 5.7 Matrix Multiplication

$$C_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

```
void matrix_mul(int A[N][P],int B[P][M]) {
    int C[N][M],i,j,k;
    for (i=0;i<N;i++) {
        for (j=0;j<P;j++) {
            C[i][j]=0;
            for (k=0;k<P;k++)
                C[i][j]+=A[i][k]*B[k][j];
        }
    }
}
```

## 5.8 Ternary Search

Find the min or max of a function that is either strictly increasing and then strictly decreasing or vice versa.

```
function ternarySearch(f, left, right, absolutePrecision)
//left and right are the current bounds; the maximum is between them
  if (right-left < absolutePrecision) return (left+right)/2
  leftThird := (left*2+right)/3
  rightThird := (left+right*2)/3
  if (f(leftThird) < f(rightThird))
    return ternarySearch(f, leftThird, right, absolutePrecision)
  else
    return ternarySearch(f, left, rightThird, absolutePrecision)
end
```

## 5.9 Long Arithmetic

Take care of leading zeroes.

**Addition:**

```
/* make sure num1 and num2 are
   filled with '\0' after digits!! */
void add(char *num1,char *num2,char *res){
  int i,carry=0;
  reverse(num1,num1+strlen(num1));
  reverse(num2,num2+strlen(num2));

  for(i=0;num1[i] || num2[i];i++){
    res[i]=num1[i]+num2[i]-'0'+carry;
    if(!num1[i] || !num2[i])res[i]+='0';
    if(res[i]>'9'){
      carry=1;
      res[i]-=10;
    }else carry=0;
  }
  if(carry)res[i]='1';
  reverse(res,res+strlen(res));
}
```

**Multiplication**

```
void mul(char *num1,char *num2,char *str){
  int i,j,res[2*SIZE]={0},carry=0;

  reverse(num1,num1+strlen(num1));
  reverse(num2,num2+strlen(num2));
  for(i=0;num1[i];i++)
    for(j=0;num2[j];j++)
      res[i+j]+=(num1[i]-'0')*(num2[j]-'0');
  for(i=2*SIZE-1;i>=0 && !res[i];i--);
}
```

```
if(i<0)strcpy(str,"0");return;
for(j=0;i>=0;i--,j++){
  str[j]=res[i]+carry;
  carry=str[j]/10;
  str[j]%=10;
  str[j+1]+='0';
  if(carry)str[j]=carry+'0';
}
```

## 5.10 Infix para Postfix

```
#define oper(a) ((a) == '+' || (a) == '-' \
|| (a) == '*' || (a) == '/')
```

```
// true if either: !!
// b is left associative
// and its precedence is <= than a
//
```

```
// b is right associative
// and its prec is < than a
bool be_prec(char a, char b) {
  int p[300];
  p['+'] = p['-'] = 1;
  p['*'] = p['/'] = 2;
  return p[a] >= p[b];
}
```

```
string shunting_yard(string exp) {
  int i = 0;
  string res;
  stack<char> s; //operators (1 char!)

  while (i < exp.size()) {
    // if it is a function token push it onto the stack

    // If it is a func arg separator (e.g., a comma):
    // Until the topmost elem of the stack is '('
    // pop the elem from the stack and
    // append it to res. If no '(' -> error
    // do not pop '('

```

```
if (isdigit(exp[i]) || exp[i] == 'x') { //number. add it
  for (; i < exp.size() &&
    (isdigit(exp[i]) || exp[i] == 'x'); i++)
    res.push_back(exp[i]);
  res.push_back(' ');
  i--; //there's a i++ down there
}
```

```

else if (exp[i] == '(') s.push('(');

else if (exp[i] == ')') {
    while (!s.empty() && s.top() != '(') {
res += (s.top() + string(" "));
s.pop();
    }
    if (s.top() != '(') ;//error
    else s.pop();
}
else if (oper(exp[i])) { //operator
    while (!s.empty() && oper(s.top()) &&
be_prec(s.top(), exp[i])) {
res += (s.top() + string(" "));
s.pop();
    }
    s.push(exp[i]);
}
i++;
}
while (!s.empty()) {
if (s.top() == '(' || s.top() == ')') ;//error
res += (s.top() + string(" "));
s.pop();
}
if (*(res.end() - 1) == ' ') res.erase(res.end() - 1);
return res;
}

```

## 5.11 Calculate Postfix expression

```

// exp is in postfix
double calc(string exp) {
    stack<double> s;
    istringstream iss(exp);
    string op;

    while (iss >> op) {
        // ATTENTION TO THIS
        if (op.size() == 1 && oper(op[0])) {
            if (s.size() < 2) ;//error
            double a = s.top(); s.pop();
            double b = s.top(); s.pop();
            switch (op[0]) {
                case '+': s.push(b + a); break;
                case '-': s.push(b - a); break;
                case '*': s.push(b * a); break;
                case '/': s.push(b / a); break;
            }
        }
    }
}

```

```

}
} else {
    istringstream iss2(op);
    double tmp;
    iss2 >> tmp;
    s.push(tmp);
}
}
return s.top();
}
}

```

## 5.12 Postfix to Infix

```

/*
 * Pass a stack with the expression to rpn2infix.
 * Ex: (bottom) 3 4 5 * + (top)
 */
string rpn2infix(stack<string> &s) {
    string x = s.top();
    s.pop();
    if (isdigit(x[0])) return x;
    else return string("(") +
        rpn2infix(s) + x + rpn2infix(s) + string(")");
}

```

## 5.13 Catalan Numbers

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

- $C_n$  counts the number of expressions containing  $n$  pairs of parentheses which are correctly matched
- $C_n$  is the number of different ways a convex polygon with  $n+2$  sides can be cut into triangles by connecting vertices with straight lines.

## 5.14 Fibonacci

```

long fib(long n){
    long matrix[2][2]={{1,1},{1,0}};
    long res[2][2]={{1,1},{1,0}};
    while(n){
        if(n&1) /* se n e impar*/
            matrix_mul(matrix,res,res);
        matrix_mul(matrix,matrix,matrix);
        n/=2;
    }
    return res[1][1];
}

```

## Additional Material

- $(1 + 2 + 3 + \dots + n)^2 = \left(\frac{n*(n+1)}{2}\right)^2 = \frac{n^2*(n+1)^2}{4}$
- $1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n*(n+1)*(2n+1)}{6}$
- $1 + 3 + 5 + 7 + \dots + (2n - 1) = n^2$
- $1^2 + 3^2 + 5^2 + \dots + (2n - 1)^2 = \frac{n(2n-1)(2n+1)}{3}$
- A number  $n$  is divisible by 11 if the difference between the sum of the odd positioned digits and the sum of the remaining digits is a multiple. e.g.  $65637 \rightarrow (6 + 6 + 7) - (5 + 3) = 11$  so it is multiple of 11

- $x$  is a power of two iff

$$(x \ \& \ (x-1)) == 0$$

- In a quadratic

$$Ax^2 + Bx + C = 0$$

the sum of the roots is  $-\frac{B}{A}$  and the product of the roots is the constant term  $C$ .

- The division of 2 complex numbers  $w$  and  $z$  is  $\frac{z}{w} = \frac{z \cdot w^{-1}}{w \cdot w^{-1}}$  where  $z^{-1}$  is the complex conjugate of  $z$ .  
 $(a + bi)^{-1} = (a - bi)$

$$(x + y)^n = \sum_{i=0}^n \binom{n}{i} x^{n-i} y^i$$

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \dots + \binom{n}{n} = 2^n$$

$$\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \dots + (-1)^n \binom{n}{n} = 0$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

For positive integers  $n, t$ , the coefficient of  $x_1^{n_1} x_2^{n_2} x_3^{n_3} \dots x_t^{n_t}$  in the expansion of  $(x_1 + x_2 + x_3 + \dots + x_t)^n$  is

$$\binom{n}{n_1} \binom{n-n_1}{n_2} \dots \binom{n-n_1-n_2-n_3-\dots-n_{t-1}}{n_t} = \frac{n!}{n_1! n_2! n_3! \dots n_t!}$$

where each  $n_i$  is an integer with  $0 \leq n_i \leq n$ , for all  $1 \leq i \leq t$ , and  $n_1 + n_2 + n_3 \dots n_t = n$ .

### Set Partitions

$B_n$  is the number of different partitions of a set with  $n$  elements.  $B_0 = 1, B_1 = 1, B_2 = 2, B_3 = 5, B_4 = 15$

$$B_{n+1} = \sum_{k=0}^n \binom{n}{k} B_k$$

$S(n, k)$  is the number of ways to partition a set of  $n$  elements in  $k$  nonempty subsets.  $S(n, k) = S(n-1, k-1) + kS(n-1, k)$