good luck guys ❤

1.5 Tricks in cmath

```
// when the number is too large. use powl instead of pow.
// will provide you more accuracy.
powl(a, b)
(int)round(p, (1.0/n)) // nth root of p
```

2.3 Greatest common divisor — GCD

```
int gcd(int a, int b)
{
  if (b==0) return a;
  else return gcd(b, a%b);
}
```

2.4 Least common multiple — LCM

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

2.5 If prime number

```
bool prime(int n)
{
  if (n<2) return false;
  if (n<=3) return true;
  if (!(n%2) || !(n%3)) return false;
  for (int i=5;i*i<=n;i+=6)
    if (!(n%i) || !(n%(i+2))) return false;

  return true;
}
```

2.7 Leap year

```
bool isLeap(int n)
{
  if (n%100==0)
    if (n%400==0) return true;
    else return false;

  if (n%4==0) return true;
  else return false;
}
```

2.6 Prime factorization

```
// smallest prime factor of a number.
function factor(int n)
{
  int a;
  if (n%2==0)
    return 2;
  for (a=3;a<=sqrt(n);a++++)
  {
    if (n%a==0)
    return a;
  }
  return n;
}

// complete factorization
int r;
while (n>1)
{
    r = factor(n);
    printf("%d", r);
    n /= r;
}
```

2.8 Binary exponiential

```
int binpow (int a, int n)
{
  int res = 1;
  while (n)
    if (n & 1)
    {
      res *= a;
      --n;
    }
    else
    {
      a *= a;
      n >>= 1;
    }
  return res;
}
```

2.9 $a^b \bmod p$

```
long powmod(long base, long exp, long modulus) {
  base %= modulus;
  long result = 1;
  while (exp > 0) {
    if (exp & 1) result = (result * base) % modulus;
    base = (base * base) % modulus;
    exp >>= 1;
  }
  return result;
}
```

2.10 Factorial mod

```
//n! mod p
int factmod (int n, int p) {
  long long res = 1;
  while (n > 1) {
    res = (res * powmod (p-1, n/p, p)) % p;
    for (int i=2; i<=n%p; ++i)
      res=(res*i) %p;
    n /= p;
  }
  return int (res % p);
}
```

2.11 Generate combinations

```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m)
{
  if (n<m) return ;
  int a[50]={0};
  int k=0;

  for (int i=1;i<=m;i++) a[i]=i;
  while (true)
  {
    for (int i=1;i<=m;i++)
      cout << a[i] << " ";
    cout << endl;

    k=m;
    while ((k>0) && (n-a[k]==m-k)) k--;
    if (k==0) break;
    a[k]++;
    for (int i=k+1;i<=m;i++)
        a[i]=a[i-1]+1;
   }
  }
```

2.19 Split plane

n lines can split a plane in $\frac{(n+1)n}{2} + 1$ sub-regions.

2.12 10-ary to m-ary

```
char a[16]={'0','1','2','3','4','5','6','7','8','9',
    'A','B','C','D','E','F'};

string tenToM(int n, int m)
{
  int temp=n;
  string result="";
  while (temp!=0)
  {
    result=a[temp%m]+result;
    temp/=m;
  }

  return result;
}
```

2.13 m-ary to 10-ary

```
string num="0123456789ABCDE";

int mToTen(string n, int m)
{
  int multi=1;
  int result=0;

  for (int i=n.size()-1;i>=0;i--)
  {
    result+=num.find(n[i])*multi;
    multi*=m;
  }

  return result;
}
```

2.14 Binomial coefficient

```
#define MAXN 100 // largest n or m
long binomial_coefficient(n,m) // compute n choose m
int n,m;
{
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = 1;
    for (j=0; j<=n; j++) bc[j][j] = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
    return bc[n][m];
}
```

2.15 Catalan numbers

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{n}{k} \tag{1}$$

The first terms of this sequence are 2, 5, 14, 42, 132, 429, 1430 when $C_0 = 1$. This is the number of ways to build a balanced formula from n sets of left and right parentheses. It is also the number of triangulations of a convex polygon, the number of rooted binary tress on $n+1$ leaves and the number of paths across a lattice which do not rise above the main diagonal.

2.16 Eulerian numbers

$$\left\langle {n \atop k} \right\rangle = k \left\langle {n-1 \atop k} \right\rangle + (n-k+1) \left\langle {n-1 \atop k-1} \right\rangle \tag{2}$$

```
// This is the number of permutations of length n with exactly k ascending sequences or runs.
// Basis: k=0 has value 1
#define MAXN 100 // largest n or k
long eularian(n,k)
int n,m;
{
    int i,j;
    long e[MAXN][MAXN];
    for (i=0; i<=n; i++) e[i][0] = 1;
    for (j=0; j<=n; j++) e[0][j] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           e[i][j] = k*e[i-1][j] + (i-j+1)*e[i-1][j-1];
    return e[n][k];
}
```

2.18 Euler's totient function

```
// the positive integers less than or equal to n that are relatively prime to n.
int phi (int n)
{
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if(n %i==0)
    {
      while(n %i==0)
        n /= i;
      result -= result / i;
    }
  if (n > 1)
    result -= result / n;
  return result;
}
```

4.3 Longest common subsequence (LCS)

```
int dp[1001][1001];

int lcs(const string &s, const string &t)
{
  int m = s.size(), n = t.size();
  if (m == 0 || n == 0) return 0;
  for (int i=0; i<=m; ++i)
    dp[i][0] = 0;
  for (int j=1; j<=n; ++j)
    dp[0][j] = 0;
  for (int i=0; i<m; ++i)
    for (int j=0; j<n; ++j)
      if (s[i] == t[j])
            dp[i+1][j+1] = dp[i][j]+1;
          else
            dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1]);
  return dp[m][n];
}
```

3.2 KMP Algorithm

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)
{
  t = VI(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length())
  {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  VI t;

  buildTable(w, t);
  while(m+i < s.length())
  {
    if(w[i] == s[m+i])
    {
      i++;
      if(i == w.length()) return m;
    }
    else
    {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }
  return s.length();
}
```

4.1 0/1 Knapsack problems

```cpp
#include<iostream>

using namespace std;

int f[1000]={0};
int n=0, m=0;

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

  for (int i=1;i<=n;i++)
  {
    int price=0, value=0;
    cin >> price >> value;

    for (int j=m;j>=price;j--)
      if (f[j-price]+value>f[j])
        f[j]=f[j-price]+value;
  }

  cout << f[m] << endl;

  return 0;
}
```

4.2 Complete Knapsack problems

```cpp
#include<iostream>

using namespace std;

int f[1000]={0};
int n=0, m=0;

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

  for (int i=1;i<=n;i++)
  {

    int price=0, value=0;
    cin >> price >> value;

    for (int j=price; j<=m; j++)
      if (f[j-price]+value>f[j])
        f[j]=f[j-price]+value;
  }

  cout << f[m] << endl;

  return 0;
}
```

4.4 Longest increasing common sequence (LICS)

```cpp
#include<iostream>

using namespace std;

int a[100]={0};
int b[100]={0};
int f[100]={0};
int n=0, m=0;

int main(void)
{
  cin >> n;
  for (int i=1;i<=n;i++) cin >> a[i];
  cin >> m;
  for (int i=1;i<=m;i++) cin >> b[i];

  for (int i=1;i<=n;i++)
  {
    int k=0;
    for (int j=1;j<=m;j++)
    {

      if (a[i]>b[j] && f[j]>k) k=f[j];
      else if (a[i]==b[j] && k+1>f[j]) f[j]=k+1;
    }
  }

  int ans=0;
  for (int i=1;i<=m;i++)
    if (f[i]>ans) ans=f[i];

  cout << ans << endl;

  return 0;
}
```

4.5 Longest Increasing Subsequence (LIS)

```cpp
#include<iostream>

using namespace std;

int n=0;
int a[100]={0}, f[100]={0}, x[100]={0};

int main(void)
{
  cin >> n;
  for (int i=1;i<=n;i++)
  {
    cin >> a[i];
    x[i]=INT_MAX;
  }

  f[0]=0;

  int ans=0;
  for(int i=1;i<=n;i++)
  {
    int l=0, r=i;

    while (l+1<r)
    {
      int m=(l+r)/2;
      if (x[m]<a[i]) l=m; else r=m;
      // change to x[m]<=a[i] for non-decreasing case
    }

    f[i]=l+1;
    x[l+1]=a[i];
    if (f[i]>ans) ans=f[i];
  }

  cout << ans << endl;

  return 0;
}
```

4.6 Maximum submatrix

```
// URAL 1146 Maximum Sum
#include<iostream>

using namespace std;

int a[150][150]={0};
int c[200]={0};

int maxarray(int n)
{
   int b=0, sum=-100000000;
   for (int i=1;i<=n;i++)
   {
      if (b>0) b+=c[i];
      else b=c[i];
      if (b>sum) sum=b;
   }

   return sum;
}

int maxmatrix(int n)
{
   int sum=-100000000, max=0;

   for (int i=1;i<=n;i++)
   {
      for (int j=1;j<=n;j++)
         c[j]=0;

      for (int j=i;j<=n;j++)
      {
         for (int k=1;k<=n;k++)
            c[k]+=a[j][k];
         max=maxarray(n);
         if (max>sum) sum=max;
      }
   }

   return sum;
}
```

4.8 Partitions of sets

Number of ways to partition $n+1$ items into k sets.

$$\left\{ {n \atop k} \right\} = k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\} \tag{3}$$

where

$$\left\{ {n \atop 1} \right\} = \left\{ {n \atop n} \right\} = 1 \tag{4}$$

4.7 Partitions of integers

```
#define MAXN 100 // largest n or m
long int_coefficient(n,k) // compute f(n,k)

int n,m;
{
    int i,j;
    long f[[MAXN][MAXN];
    f [1][1] = 1;
    for (i=0;i<=n;i++) f[i][0] = 0;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
           if (i-j <= 0)
               f[i][j] = f[i][k-1];
           else
               f[i][j] = f[i-j][k]+f[i][k-1];
    return f[n][k];
}
```

5.1 Tree traversal

```
int L[100]={0};
int R[100]={0};

void DLR(int m)
{
  cout << m << " ";
  if (L[m]!=0) DLR(L[m]);
  if (R[m]!=0) DLR(R[m]);
}

void LDR(int m)
{
  if (L[m]!=0) LDR(L[m]);
  cout << m << " ";
  if (R[m]!=0) LDR(R[m]);
}

void LRD(int m)
{
  if (L[m]!=0) LRD(L[m]);
  if (R[m]!=0) LRD(R[m]);
  cout << m << " ";
}
```

5.2 Depth and width of tree

```
int l[100]={0};
int r[100]={0};
stack<int> mystack;
int n=0;
int w=0;
int d=0;

int depth(int n)
{
  if (l[n]==0 && r[n]==0)
    return 1;

  int depthl=depth(l[n]);
  int depthr=depth(r[n]);
  int dep=depthl>depthr ? depthl:depthr;
  return dep+1;
}

void width(int n)
{
  if (n<=d)
  {
    int t=0,x;
    stack<int> tmpstack;
    while (!mystack.empty())
    {
      x=mystack.top();
      mystack.pop();
      if (x!=0)
      {
        t++;
        tmpstack.push(l[x]);
        tmpstack.push(r[x]);
      }
    }

    w=w>t?w:t;
    mystack=tmpstack;
    width(n+1);
  }
}
```

6 Graph Theory

6.1 Graph representation

```
// The most common way to define graph is to use adjacency matrix
// example:
//     (1) (2) (3) (4) (5)
// (1)  2   0   5   0   0
// (2)  4   2   0   0   1
// (3)  3   0   0   1   4
// (4)  6   9   0   0   0
// (5)  1   1   1   1   5
// it's always a square matrix.
// suppose a graph has n nodes, if given exactly adjacency matrix
for (int i=1;i<=n;i++)
  for (int j=1;i<=n;j++)
  {
    cin << a[i][j] << endl;
  }
// Usually will go like this representation in data
// start_node end_node weight
// suppose m lines
for (int i=1;i<=m;i++)
{
  int x=0, y=0, t=0;
  cin >> x >> y >> t;
  a[x][y]=t;
  // if undirected graph
  a[y][x]=t;
}
// another variant: on the ith line, has data as
// end_node weight
// when you read data, you can assign matrix as
a[i][x]=t;
// if undirected graph
a[x][i]=t;

// Initialization of graph !!!IMPORTANT
// Depends on usage, normally initialize as 0 for all elements in matrix.
// so that 0 means no connection, non-0 means connection
// (for problem without weight, use weight as 1)
// If weights are important in this context (especially searching for path)
```

```
// Initialize graph as infinity for all elements in matrix.

// Another way to store graph is Adjacency list
// No space advantage if using array (unknown maximum number for in-degree).
// Big space advantage if using dynamic data structure (like list, vector).
// each row represent a node and its connectivity.
// we don't need it so much due to it's search efficiency.
// let's define a node as
struct Node{
  int id; // node id
  int w;  // weight
};
// suppose n nodes and m lines of inputs as
// start_node end_node weight
// assume using <vector> in this example
// g is a vector, and each element of g is also a vector of Node
for (int i=1;i<=m;i++)
{
  int x=0, y=0, t=0;
  cin >> x >> y >> t;
  Node temp; temp.id=y; temp.w=t;
  g[x].push_back(temp);
  // if undirected
  temp.id=x;
  g[y].push_back(temp);
}
// Note that you don't need this node structure if graph has only connectivity information.

/**** Special Structure ****/

// Special structure here is usually not a typical graph, like city-blocks, triangles
// They are represented in 2-d array and shows weights on nodes instead of edges.
// Note that in this case travel through edge has no cost, but visit node has cost.

// Triangles: Read data like this
// 1
// 1 2
// 4 2 7
// 7 3 1 5
// 6 2 9 4 6
for (int i=1;i<=n;i++)
  for (int j=i;j<=n;j++)
    cin >> a[i][j];

// Simple city-blocks: it's just like first form of adjacency matrix, but this time
// represents weights on nodes, may not be square matrix.
// 1 2 4 5 6
// 2 4 5 1 3
// 4 5 2 3 6
for (int i=1;i<=n;i++)
  for (int j=1;<=m;j++)
    cin >> a[i][j];


// More complex data structures: typical city-block structure may has some constraints on
// questions, but it has no boundaries. However, some questions requires to form a maze.
// In these cases, data structures can be very flexible, it totally depends on how the question
// presents the data. A usual way is to record it's adjacent blocks information:
struct Block{
  bool l[4]; // if has 8 neighbors then use bool l[8];
```

```
            // label them as your favor, e.x.
            //   1    1 2 3
            // 4 x 2  8 x 4
            //   3    7 6 5
            // true if there is path, false if there is boundary
  // other informations (optional)
  int weight;
  int component_id;
  // etc.
};

// Note that usually we use array from index 1 instead of 0 because sometimes
// you need index 0 as your boundary, and start from index 1 will give you
// advantage on locating nodes or positions
```

6.2 Flood fill algorithm

```
//component(i) denotes the
//component that node i is in
void flood_fill(new_component)
  do
    num_visited = 0
    for all nodes i
      if component(i) = -2
      num_visited = num_visited + 1
      component(i) = new_component

    for all neighbors j of node i
      if component(j) = nil
        component(j) = -2
  until num_visited = 0

void find_components()
  num_components = 0
  for all nodes i
    component(node i) = nil
  for all nodes i
    if component(node i) is nil
      num_components = num_components + 1
      component(i) = -2
      flood_fill(component num_components)
```

6.3 SPFA — shortest path

```
int q[3001]={0}; // queue for node
int d[1001]={0}; // record shortest path from start to ith node
bool f[1001]={0};
int a[1001][1001]={0}; // adjacency list
int w[1001][1001]={0}; // adjacency matrix

int main(void)
{
  int n=0, m=0;
  cin >> n >> m;

  for (int i=1;i<=m;i++)
  {
```

```
    int x=0, y=0, z=0;
    cin >> x >> y >> z; // node x to node y has weight z
    a[x][0]++;
    a[x][a[x][0]]=y;
    w[x][y]=z;
    /*
    // for undirected graph
    a[x][0]++;
    a[y][a[y][0]]=x;
    w[y][x]=z;
    */
  }

  int s=0, e=0;
  cin >> s >> e; // s: start, e: end
  SPFA(s);
  cout << d[e] << endl;

  return 0;
}

void SPFA(int v0)
{
  int t,h,u,v;
  for (int i=0;i<1001;i++) d[i]=INT_MAX;
  for (int i=0;i<1001;i++) f[i]=false;
  d[v0]=0;
  h=0; t=1; q[1]=v0; f[v0]=true;

  while (h!=t)
  {
    h++;
    if (h>3000) h=1;
    u=q[h];
    for (int j=1; j<=a[u][0];j++)
    {
      v=a[u][j];
      if (d[u]+w[u][v]<d[v]) // change to > if calculating longest path
      {
        d[v]=d[u]+w[u][v];
        if (!f[v])
        {
          t++;
          if (t>3000) t=1;
          q[t]=v;
          f[v]=true;
        }
      }
    }
    f[u]=false;
  }
}
```

6.4 Floyd-Warshall algorithm – shortest path of all pairs

```
// map[i][j]=infinity at start
void floyd()
{
```

```
  for (int k=1; k<=n; k++)
    for (int i=1; i<=n; i++)
      for (int j=1; j<=n; j++)
        if (i!=j && j!=k && i!=k)
          if (map[i][k]+map[k][j]<map[i][j])
            map[i][j]=map[i][k]+map[k][j];
}
```

6.5 Prim — minimum spanning tree

```
int d[1001]={0};
bool v[1001]={0};
int a[1001][1001]={0};

int main(void)
{
  int n=0;
  cin >> n;
  for (int i=1;i<=n;i++)
  {
    int x=0, y=0, z=0;
    cin >> x >> y >> z;
    a[x][y]=z;
  }
  for (int i=1;i<=n;i++)
    for (int j=1;j<=n;j++)
      if (a[i][j]==0) a[i][j]=INT_MAX;

  cout << prim(1,n) << endl;
}
int prim(int u, int n)
{
  int mst=0,k;

  for (int i=0;i<d.length;i++) d[i]=INT_MAX;
  for (int i=0;i<v.length;i++) v[i]=false;

  d[u]=0;
  int i=u;

  while (i!=0)
  {
    v[i]=true;k=0;
    mst+=d[i];
    for (int j=1;j<=n;j++)
      if (!v[j])
      {
        if (a[i][j]<d[j]) d[j]=a[i][j];
        if (d[j]<d[k]) k=j;
      }
    i=k;
  }
  return mst;
}
```

6.6 Eulerian circuit

```cpp
// USACO Fence
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0, c=0;

void dfs(int k)
{
  for (int i=1;i<=n;i++)
    if (g[k][i])
    {
      g[k][i]=false;
      g[i][k]=false;
      dfs(i);
    }
  m++;
  ans[m]=k;
}

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

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    g[x][y]=true;
    g[y][x]=true;
    f[x]++;
    f[y]++;
  }

  m=0;
  int k1=0;
  for (int i=1;i<=n;i++)
  {
    if (f[i]%2==1) k1++;
    if (k1>2)
    {
      cout << "error" << endl;
      return 0;
    }
    if (f[i]%2 && c==0) c=i;
  }

  if (c==0) c=1;
  dfs(x);

  for (int i=m;i>=1;i--) cout << ans[i] << endl;
  return 0;
}
```

6.7 Topological sort

```cpp
// Find any solution of topological sort.
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

void dfs(int k)
{
  int i=0;
  v[k]=true;
  for (int i=1;i<=n;i++)
    if (g[k][i] && !v[i]) dfs(i);

  m++;
  ans[m]=k;
}

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

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    cin >> x >> y;
    g[y][x]=true;
  }

  m=0;
  for (int i=1;i<=n;i++)
    if (!v[i]) dfs(i);

  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```

```cpp
// Find the order of topological sort is dictionary minimum
#include<iostream>

using namespace std;

int f[100]={0}, ans[100]={0};
bool g[100][100]={0}, v[100]={0};
int n=0, m=0;

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

  for (int i=1;i<=m;i++)
  {
    int x=0, y=0;
    cin >> x >> y;
    g[x][y]=true;
    f[y]++;
  }
```

```
  for (int i=1;i<=n;i++)
  {
    for (int j=1;j<=n;j++)
    {
      if (f[j]==0 && !v[j]) break;

      if (f[j]!=0)
      {
        cout << "error" << endl;
        return 0;
      }

      ans[i]=j;
      v[j]=true;
      for (int k=1;k<=n;k++)
        if (g[j][k]) f[k]--;
    }
  }
  for (int i=1;i<=n;i++) cout << ans[i] << endl;
  return 0;
}
```

2.1.3 Products

$$\prod_{k=1}^{\infty}\left(1-\left(\frac{x}{\pi k}\right)^2\right)=\prod_{k=1}^{\infty}\cos\left(\frac{x}{2^k}\right)=\frac{\sin(x)}{x};\qquad \prod_{k=1}^{\infty}\left(1-\frac{4x^2}{(2k-1)^2}\right)=\cos(\pi x)$$

$$\prod_{k=0}^{\infty}\left(1+x^{2^k}\right)=\frac{1}{1-x};\qquad \prod_{k=1}^{\infty}\left(1+\frac{(-1)^{k+1}}{2k-1}\right)=\sqrt{2}$$

2.1.4 Sums

$$\sum_{k=1}^{\infty}\frac{1}{k^2}=\frac{1}{6}\pi^2;\qquad \sum_{k=1}^{\infty}\frac{1}{k^4}=\frac{1}{90}\pi^4$$

$$\sum_{k=1}^{\infty}\frac{1}{(2k-1)^2}=\frac{1}{8}\pi^2;\qquad \sum_{k=1}^{\infty}\frac{1}{(2k-1)^4}=\frac{1}{96}\pi^4$$

$$\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}=\frac{1}{4}\pi;\qquad \sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}=\ln(2)$$

$$\sum_{k=1}^{n}k=\frac{1}{2}n(n+1);\qquad \sum_{k=1}^{n}k^2=\frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{k=1}^{n}k^3=\left(\frac{1}{2}n(n+1)\right)^2;\qquad \sum_{k=1}^{n}\left(\frac{1}{k^p}-\frac{1}{(k+1)^p}\right)=1-\frac{1}{(n+1)^p},\quad p\in\mathbb{R}$$

$$\sum_{k=0}^{n}x^k=\frac{1-x^{n+1}}{1-x};\qquad \sum_{k=0}^{n}\binom{m+k}{m}=\binom{m+n+1}{m+1}$$

$$\sum_{k=0}^{n}\binom{n}{k}a^kb^{n-k}=(a+b)^n$$

Series

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}.$$

In general:

$$\sum_{i=1}^{n} i^m = \frac{1}{m+1}\left[(n+1)^{m+1} - 1 - \sum_{i=1}^{n}\left((i+1)^{m+1} - i^{m+1} - (m+1)i^m\right)\right]$$

$$\sum_{i=1}^{n-1} i^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k} B_k n^{m+1-k}.$$

Geometric series:

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

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

Harmonic series:

$$H_n = \sum_{i=1}^{n} \frac{1}{i}, \qquad \sum_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}.$$

$$\sum_{i=1}^{n} H_i = (n+1)H_n - n, \quad \sum_{i=1}^{n}\binom{i}{m} H_i = \binom{n+1}{m+1}\left(H_{n+1} - \frac{1}{m+1}\right).$$

1. $\binom{n}{k} = \frac{n!}{(n-k)!k!}$, **2.** $\sum_{k=0}^{n}\binom{n}{k} = 2^n$, **3.** $\binom{n}{k} = \binom{n}{n-k}$,

4. $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$, **5.** $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$,

6. $\binom{n}{m}\binom{m}{k} = \binom{n}{k}\binom{n-k}{m-k}$, **7.** $\sum_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$,

8. $\sum_{k=0}^{n}\binom{k}{m} = \binom{n+1}{m+1}$, **9.** $\sum_{k=0}^{n}\binom{r}{k}\binom{s}{n-k} = \binom{r+s}{n}$,

10. $\binom{n}{k} = (-1)^k\binom{k-n-1}{k}$,

```
// Distance de Levenshtein // Edit distance
#include<iostream>
#include<vector>
#include<limits>
#include<algorithm>
#include<cstdio>

// Insertion (1), Deletion (1), Substitution (1)
int edit_distance(const std::string &s, const std::string &t)
{
  auto memo=std::vector<std::vector<int>>(
       s.size()+1, std::vector<int>(t.size()+1, 0));

  for(auto i=0; i<s.size()+1; i++)
  {
    for(auto j=0; j<t.size()+1; j++)
    {
      if(i == 0)
```

```
        memo[i][j] = j;
      else if(j == 0)
        memo[i][j] = i;
      else
      {
        auto deletion = 1 + memo[i-1][j];
        auto insertion = 1 + memo[i][j-1];
        auto substitution = memo[i-1][j-1] + ( (s[i-1]==t[j-1]) ? 0 : 1 );
        memo[i][j] = std::min(deletion, std::min(insertion, substitution));
      }
    }
  }
  return memo[s.size()][t.size()];
} // edit_distance()

int main()
{
  std::string s, t;

  std::cin >> s;
  std::cin >> t;

  printf("%d\n", edit_distance(s, t));

  return 0;
} // main()
```

```
// D◆placement du cavalier sur un ◆chiquier :
// Visiter toutes les cases en passant par chaque case une seule fois.
#include <iostream>
#include <iomanip>
#include <cstdint>
using namespace std;

const int N = 8; // dimension de l'◆chiquier
uint64_t appels = 0; // pour compter le nombre d'essais

// D◆finition d'un type 'carre_t'
struct carre_t
{
  int rang;
  bool visitee;
};

// Un ◆chiquier est un tableau NxN d'objets de type 'carre'
carre_t echiquier[N][N];

// D◆finition d'un type 'dep_t' (pour d◆placement)
struct dep_t { int x, y; };

// Initialisation d'un tableau avec les 8 d◆placements
// possibles du cavalier sur un ◆chiquier
dep_t deplacement[8] =
  { { 1,-2}, { 2,-1}, { 2, 1}, { 1, 2},
    {-1, 2}, {-2,-1}, {-1,-2}, {-2, 1} };

// D◆claration d'une pile de taille NxN (g◆r◆e comme un tableau)
// servant ◆ empiler les coordonn◆es des cases visit◆es
dep_t pile[N*N];
```

```
int sommet = 0; // pile vide au d�part

bool parcours(int, int, int);

int main()
{
  bool p = parcours(0, 0, N*N);
  cout << "sommet = " << sommet << endl << endl;

  for(int l = 0; l < N; l++)
  {
    cout << "-----------------------------------------\n|";
    for(int c = 0; c < N; c++)
    {
      for(int r = 0; r < sommet; r++)
        if((pile[r].x == l) && (pile[r].y == c))
          cout << setw(3) << r+1 << " |";
    }
    cout << endl;
  }
  cout << "-----------------------------------------\n\n";
  cout << appels << " essais.\n";

  return 0;
} // main()

// Fonction de backtracking Cavalier:
//   (i,j) : coordonn�es d'une case
//   k : rang de r�cursivit� (NxN pour le 1er appel, 1 pour le dernier)
bool parcours(int i, int j, int k)
{
  appels++;

  if(i < 0 || i >= N || j < 0 || j >= N) // cas hors bornes
    return false;

  // sinon (le return permet d'�conomiser un else)
  if(echiquier[i][j].visitee) // case d�j� visit�e par le cavalier
    return false;

  // sinon (le return permet d'�conomiser un else)
  if(k == 1) // dernier niveau (rang) de profondeur : c'est gagn� !
  {
    pile[sommet].x = i;
    pile[sommet].y = j;
    sommet++;
    return true;
  }

  // sinon (le return permet d'�conomiser un else)
  echiquier[i][j].rang = k;
  echiquier[i][j].visitee = true;
  pile[sommet].x = i;
  pile[sommet].y = j;
  sommet++;

  for(int p = 0; p < 8; p++)
  {
    // APPEL R�CURSIF au niveau k-1
    // et en essayant les cases atteignables de la position courante
    if(parcours(i+deplacement[p].x, j+deplacement[p].y, k-1) == true)
```

```
      return true;
  }
  // sinon (le return permet d'économiser un else)
  // on a essayé les 8 positions possibles sans succès :
  // IL FAUT RECULER : on est au COEUR DU BACKTRACKING !
  echiquier[i][j].visitee = false;
  --sommet;     // on dépile la position courante qui n'a mené à rien
  return false; // et on renvoie faux
} // parcours()
```

Algorithme à JF:

```
def numberToBase(n, b,num):
       if n == 0:
       return [0]
       digits = []
       while len(digits) != num-1:
       digits.append(int(n % b))
       n //= b
       return digits[::-1]

num = int(input())
arr = []
for i in range(num):
       arr.append(input())
for j in range(3**(num-1)):
       a = numberToBase(j,3,num)
       while(len(a) < num-1):
       a.append(0)
```

N-Reines

```
// Tester si une reine peut Ãªtre placÃ©e en (row,col).
// Fonction appelÃ©e lorsque "col" reines ont dÃ©jÃ 
// Ã©tÃ© placÃ©es dans les colonnes 0 Ã  col-1.
bool isSafe(vector<vector<int>> &board, int row, int col)
{
  int i, j;

  // VÃ©rifier la ligne du cÃ´tÃ© gauche.
  for(i = 0; i < col; i++)
    if(board[row][i] == 1)
      return false;

  // VÃ©rifier la diagonale supÃ©rieure du cÃ´tÃ© gauche.
  for(i = row, j = col; i >= 0 && j >= 0; i--, j--)
    if(board[i][j] == 1)
      return false;

  // VÃ©rifier la diagonale infÃ©rieure du cÃ´tÃ© gauche.
  int N = board.size();
  for(i = row, j = col; j >= 0 && i < N; i++, j--)
    if(board[i][j] == 1)
      return false;

  return true;
} // isSafe()

// Fonction rÃ©cursive pour rÃ©soudre le problÃ¨me.
bool solveNQUtil(vector<vector<int>> &board, int col)
{
  int N = board.size();

  // Si toutes les reines sont placÃ©es, retourner true.
  if(col >= N)
    return true;

  // La colonne col Ã©tant donnÃ©e, on essaye de placer
  // la reine sur chaque ligne une par une.
  for(int i = 0; i < N; i++)
  {
    // VÃ©rifier si la reine peut Ãªtre placÃ©e en (i,col).
    if(isSafe(board, i, col))
    {
      // Placer la reine en (i,col).
      board[i][col] = 1;

      // Appel rÃ©cursif pour placer les autres reines.
      if(solveNQUtil(board, col + 1))
        return true;

      // Si le placement de la reine en (i,col) ne mÃ¨ne
      // Ã  aucune une solution, on retire la reine.
      board[i][col] = 0; // BACKTRACK
    }
  }

  // Si la reine ne peut Ãªtre placÃ©e sur aucune ligne
  // de la colonne col, retourner false.
  return false;
} // solveNQUtil()
```