Dijkstra O(nlogn)

```
vector<vector<pair<int,int>>> adj; //v, weight
int dist[N], par[N];

int Dijkstra(int src, int snk) {
    memset(dist, 63, sizeof(dist[0]) * adj.size());
    par[src] = -1;
    dist[src] = 0;
    priority_queue<pair<int,int>> q; // cost, u
    q.emplace(0, src);
    while(q.size()){
        int d,u,v,w;
        tie(d,u) = q.top();
        q.pop();
        d = -d;
        if(d != dist[u]) continue;
        if(u == snk) return d;
        for(auto&p : adj[u]){
            tie(v,w) = p;
            if(dist[v] > d + w){
                dist[v] = d + w;
                par[v] = u;
                q.emplace(-dist[v],v);
            }
        }
    }
    return -1;
}
```

Dijkstra O(n^2)

```
int mat[N][N], n;
int dist[N], par[N],notVis[N],sz;

int sp(int src, int snk) {
    memset(dist, 63, sizeof(dist[0]) * n);
    par[src] = -1;
    dist[src] = 0;
    //iota(notVis, notVis + n, 0);
    for(int i=0;i<n;++i)notVis[i]=i;
    sz=n;
    int nxt=src,best=0x3f3f3f3f;
    while(~nxt){
        int u=notVis[nxt];
        if(u==snk)return dist[u];
        notVis[nxt]=notVis[--sz];
        nxt=-1,best=0x3f3f3f3f;
        for(int i=0;i<sz;++i){
            int v=notVis[i];
            int d=dist[u]+mat[u][v];
            if(d<dist[v]){
                dist[v]=d;
                par[v]=u;
            }
            if(dist[v]<best){
                best=dist[v];
                nxt=i;
            }
        }
    }
    return -1;
}

int main() {
    int m;
    memset(mat, 63, sizeof(mat));
    cin >> m >> n;
    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
        --u, --v;
        mat[v][u] = mat[u][v] = min(mat[u][v], w);
    }
    cout<<sp(0,n-1)<<endl;
    return 0;
}
```

Bellman

```
vector<vector<pair<int, int>>> adj;
int dist[N], par[N];

bool sp(int src) {
    memset(dist, 63, sizeof(dist[0]) * adj.size());
    par[src] = -1;
    dist[src] = 0;
    int t = adj.size(); //n
    queue<int> q; //nodes
    q.push(src);
    vector<bool> inq(adj.size()); // exist
    inq[src] = 1;
    while (t-- && q.size()) {
        int s = q.size();
        while (s--) {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (auto &p: adj[u]) {
                int v, w;
                tie(v, w) = p;
                if (dist[u] + w < dist[v]) {
                    if(!t) return 0; // negative cycle
                    dist[v] = dist[u] + w;
                    par[v] = u;
                    if (!inq[v]) {
                        q.push(v);
                        inq[v] = 1;
                    }
                }
            }
        }
    }
    return 1;
}

int main() {
    int m, t, n;
    cin >> t;
    while (t--) {
        cin >> n >> m;
        adj.clear();
        adj.resize(++n);
        while (m--) {
            int u, v, w;
            cin >> u >> v >> w;
            adj[u].emplace_back(v, w);
        }
        for (int i = 0; i < n - 1; i++)
            adj[n - 1].emplace_back(i, 0);
        cout << (sp(n - 1) ? "not " : "") << "possible" << endl;
    }
    return 0;
}
```

Floyd

```
int mat[N][N], n;


void floydWarshal() {
    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                mat[i][j] = min(mat[i][j], mat[i][k] + mat[k][j]);
}



        floydWarshal();
        int sum = 0, cnt = 0;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                if (i != j)
                    sum += mat[i][j], cnt++;
        printf("Case %d: average length between pages = %.3lf clicks\n",++tc,sum*1.0/cnt);
```

Knuth (optimal binary search tree)

```
const int N = 250 + 5;
int mem[N][N], roots[N][N], n;
int frq[N];

int sum(int s, int e) {
    int ret = frq[e];
    if (s)ret -= frq[s - 1];
    return ret;
}

int solve() {
    for (int len = 0; len <= n; ++len) {
        for (int s = 0; s + len <= n; ++s) {
            int e = s + len - 1;
            int &ret = mem[s][e];
            int &root = roots[s][len];
            if (s >= e) {
                ret = 0;
                root = s;
                continue;
            }
            ret = 1e9;
            for (int r = roots[s][len-1]; r <= roots[s+1][len-1]; ++r) {
                int cur = mem[s][r - 1] + mem[r + 1][e];
                cur += sum(s, r - 1) + sum(r + 1, e);
                if (cur < ret) {
                    ret = cur;
                    root = r;
                }
            }
```

```
        }
    }
    return mem[0][n - 1];
}

int main() {
    while (~scanf("%d", &n)) {
        for (int i = 0; i < n; ++i) {
            scanf("%d", frq + i);
            if (i) frq[i] += frq[i - 1];
        }
        printf("%d\n", solve());
    }
    return 0;
}
```

Input

The input will contain several instances, one per line.

Each line will start with a number $1 \leq n \leq 250$, indicating the size of S. Following n, in the same line, there will be n non-negative integers representing the query frequencies of the elements of S: $f(e_1), f(e_2), \ldots, f(e_n)$, $0 \leq f(e_i) \leq 100$. Input is terminated by end of file.

Output

For each instance of the input, you must print a line in the output with the total cost of the Optimal Binary Search Tree.

Sample Input

```
1 5
3 10 10 10
3 5 10 20
```

Sample Output

```
0
20
20
```

Divide and Conquer

```
const int N = 1e5 + 5;
int n, w[N], K;
long long mem[2][N], sumw[N], sumwi[N];

void row(int k, int s, int e, int l, int r) {
    if (s > e) return;
    int i = (s + e) / 2;
    auto &res = mem[k & 1][i];
    res = 1e18;
    int p = 0;
    for (int j = max(i, l); j <= r; ++j) {
        auto cost = sumwi[j] - sumwi[i - 1] - (sumw[j] - sumw[i - 1]) * i;
        cost += mem[(k - 1) & 1][j + 1];
        if (res > cost) {
            res = cost;
            p = j;
        }
    }
    row(k, s, i - 1, l, p);
    row(k, i + 1, e, p, r);
}


int main() {
    scanf("%d %d", &n, &K);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &w[i]);
        sumw[i] = sumw[i - 1] + w[i];
        sumwi[i] = sumwi[i - 1] + w[i] * i;
    }
    fill(mem[0] + 1, mem[0] + n + 1, 1e18);
    mem[0][n + 1] = 0;
    for (int k = 1; k <= K; ++k) {
        mem[k & 1][n + 1] = 0;
        row(k, 1, n, 1, n);
    }

    printf("%lld\n", mem[K & 1][1]);
}
```

Example

Input	Output
5 2 1 2 3 4 5	13

It would be best to form the 2 piles in points 1 and 4. Leaves 1, 2 and 3 are taken to coordinate 1, and 4 and 5 are taken to coordinate 4. The total cost is:

$1 * 0 + 2 * 1 + 3 * 2 + 4 * 0 + 5 * 1 = 13$

Convex Hull Trick

```
const int N = 1e5 + 5;
int d[N], b[N], s[N], p[N], idx[N], n, days, tc;
long long mem[N];

void print(int a[]) {
   for (int i = 1; i <= n; i++) {
      cout << a[i] << " \n"[i == n];
   }
}

long long sub(pair<const long long, long long> &pair, long long x) {
   return pair.first * x + pair.second;
}



pair<long long, long long>
intersect(pair<const long long, long long> &l1, pair<const long long, long long> &l2) {
//   m1 * x + c1 = m2 * x + c2
//   m1 * x - m2 * x = c2 - c1
//   x * (m1 - m2) = c2 - c1
//   x = (c2 - c1) / (m1 - m2)
   long long m1 = l1.first, c1 = l1.second, m2 = l2.first, c2 = l2.second;
   return {c2 - c1, m1 - m2};
}

bool notOK(pair<const long long, long long> &bf, pair<const long long, long long> &md,
         pair<const long long, long long> &af) {
   auto x1 = intersect(bf, md), x2 = intersect(md, af);
   return x1.first * x2.second >= x2.first * x1.second;
}

long long solve() {
   //vector<pair<long long, long long>> v;
   map<long long, long long> lines;
   lines.emplace(0, 0);
   for (int i = 0; i <= n + 1; ++i) {
      auto &ret = mem[i];
      int idi = idx[i];
      ret = 0;
//       for (int j = 0; j < i; ++j) {
//          long long m = mem[j];
//          int idj = idx[j];
//          if (b[idi] > m)continue;
```

```
//          if (b[idj] > m)continue;
//          //long long profit = m - b[idj] + s[idj] + (d[idi] - d[idj] - 1) * p[idj];
//          //long long profit = m - b[idj] + s[idj] + d[idi] * p[idj] - d[idj] * p[idj] - p[idj];
//          //long long profit = m - b[idj] + s[idj]  - d[idj] * p[idj] - p[idj] + d[idi] * p[idj];
//          long long C = m - b[idj] + s[idj] - d[idj] * p[idj] - p[idj];
//          long long M = p[idj];
//          long long X = d[idi];
//          long long profit = C + M * X;
//          ret = max(ret, profit);
//       }
//       for (auto &[M, C]: v) {
//          long long X = d[idi];
//          long long profit = C + M * X;
//          ret = max(ret, profit);
//       }
      long long X = d[idi];
      while (lines.size() > 1) {
         auto snd = lines.begin(), fst = snd++;
         if (sub(*snd, X) < sub(*fst, X))break;
         lines.erase(fst);
      }
      ret = sub(*lines.begin(), X);
      if (b[idi] > ret)continue;
      long long m = ret;
      long long C = m - b[idi] + s[idi] - d[idi] * 1ll * p[idi] - p[idi];
      long long M = p[idi];
      auto it = lines.emplace(M, C).first;
      it->second = max(it->second, C);
      auto bf = it, af = it;
      if (it != lines.begin() && --lines.end() != it && notOK(*--bf, *it, *++af)) {
         lines.erase(it);
         continue;
      }
      while (1) {
         auto bf = it;
         if (it == lines.begin())break;
         --bf;
         auto bb = bf;
         if (bf == lines.begin())break;
         --bb;
         if (notOK(*bb, *bf, *it))lines.erase(bf);
         else break;

      }
      while (1) {
         auto af = it;
         if (it == --lines.end())break;
         ++af;
```

```
        auto aa = af;
        if (af == --lines.end())break;
        ++aa;
        if (notOK(*it, *af, *aa))lines.erase(af);
        else break;

      }
    }
    return mem[n + 1];
}

int main() {
    while (scanf("%d%d%d", &n, &s[0], &days), n || s[0] || days) {
      for (int i = 1; i <= n; ++i) {
        scanf("%d%d%d%d", &d[i], &b[i], &s[i], &p[i]);
        idx[i] = i;
      }
      sort(idx + 1, idx + n + 1, [](int i, int j) {
        return d[i] < d[j];
      });
      idx[n + 1] = n + 1;
      d[n + 1] = days + 1;
      printf("Case %d: %lld\n", ++tc, solve());
    }
}
```

Sample Input
Num of machine, dollars, days
6 10 20
Day buy sell profit
6 12 1 3
1 9 1 2
3 2 1 2
8 20 5 4
4 11 7 4
2 10 9 1
0 0 0

Sample Output
Case 1: 44

DSU (Disjoint Set)

```
struct DSU {
    int par[N], sz[N], n, comp;

    void init(int n) {
        comp = this->n = n;
        memset(par, -1, n * sizeof par[0]);
        fill(sz, sz + n, 1);
    }

    int find(int u){
        if(par[u]==-1)return u;
        return par[u]=find(par[u]);
    }
    bool join(int u,int v){
        u=find(u),v=find(v);
        if(u==v)return false;
        if(sz[u]>sz[v])swap(u,v);
        par[u]=v;
        sz[v]+=sz[u];
        comp--;
        return true;
    }
} dsu;
```

Sample 1

Input	copy	
4 4 1 2 2 3 1 3 3 4 3 2 4 3		1 2 3

```
int n,m,q;
int f[N],t[N],ans[N],qur[N];
bool deleted[N];
int main() {
    scanf("%d%d",&n,&m);
    dsu.init(n);
    for(int i=0;i<m;i++){
        scanf("%d%d",&f[i],&t[i]);
        f[i]--,t[i]--;
    }
    scanf("%d",&q);
    for(int i=0;i<q;i++){
        scanf("%d",&qur[i]);
        qur[i]--;
        deleted[qur[i]]=1;
    }
    for(int i=0;i<m;i++){
        if(!deleted[i])dsu.join(f[i],t[i]);
    }
    for(int i=q-1;i>=0;i--){
        ans[i]=dsu.comp;
        dsu.join(f[qur[i]],t[qur[i]]);
    }
    for(int i=0;i<q;i++)printf("%d%c",ans[i],"
\n"[i+1==q]);
}
```

MST - Kruskal

```
int n, m;
int f[N], t[N], cst[N], idx[N];

long long kruskal() {
    dsu.init(n);
    long long ret = 0;
    sort(idx, idx + m, [&](int i, int j) {
        return cst[i] < cst[j];
    });
    for (int i = 0; i < m; i++) {
        int e = idx[i];
        if (dsu.join(f[e], t[e]))
            ret += cst[e];
    }
    return ret;

}

int main() {
    scanf("%d%d", &n, &m);

    for (int i = 0; i < m; i++) {
        scanf("%d%d%d", &f[i], &t[i], &cst[i]);
        idx[i] = i;
        f[i]--, t[i]--;
    }
    long long ans = kruskal();
    if (dsu.comp > 1) puts("IMPOSSIBLE");
    else
        printf("%lld\n", ans);
    return 0;
}
```

Constraints

- $1 \le n \le 10^5$
- $1 \le m \le 2 \cdot 10^5$
- $1 \le a, b \le n$
- $1 \le c \le 10^9$

Example

Input	Output
5 6 1 2 3 2 3 5 2 4 2 3 4 8 5 1 7 5 4 4	14

There are n cities and m roads between them. Unfortunately, the condition of the roads is so poor that they cannot be used. Your task is to repair some of the roads so that there will be a decent route between any two cities.

For each road, you know its reparation cost, and you should find a solution where the total cost is as small as possible.

Input

The first input line has two integers n and m: the number of cities and roads. The cities are numbered $1, 2, \ldots, n$.

Then, there are m lines describing the roads. Each line has three integers a, b and c: there is a road between cities a and b, and its reparation cost is c. All roads are two-way roads.

Every road is between two different cities, and there is at most one road between two cities.

Output

Print one integer: the minimum total reparation cost. However, if there are no solutions, print "IMPOSSIBLE".

MST – Prim

```
long long prim() {
    int src = 0;
    memset(dist, 63, sizeof(dist[0]) * adj.size());
    par[src] = -1;
    dist[src] = 0;
    priority_queue<pair<int, int>> q;
    vector<bool> vis(n);
    q.emplace(0, src);
    long long ans = 0,cnt=n;

    while (q.size()) {
        int d, u, v, w;
        tie(d, u) = q.top();
        q.pop();
        d = -d;
        if (vis[u]) continue;
        vis[u] = true;
        cnt--;
        ans += d;
        for (auto &p: adj[u]) {
            tie(v, w) = p;
            if(vis[v])continue;
            if (dist[v] > w) {
                dist[v] = w;
                par[v] = u;
                q.emplace(-dist[v], v);
            }
        }
    }
    return cnt ? -1 : ans;
}
```

MST – Prim -> for dense graph

```
long long prim() {
    int src = 0;
    memset(dist, 63, sizeof(dist[0]) * n);
    par[src] = -1;
    dist[src] = 0;

    for (int i = 0; i < n; ++i)notVis[i] = i;
    sz = n;
    int nxt = src, best = 0x3f3f3f3f;
    long long cst = 0;
    while (~nxt) {
        int u = notVis[nxt];
        notVis[nxt] = notVis[--sz];
        cst += dist[u];
        nxt = -1, best = 0x3f3f3f3f;
        for (int i = 0; i < sz; ++i) {
            int v = notVis[i];
            int d = mat[u][v];
            if (d < dist[v]) {
                dist[v] = d;
                par[v] = u;
            }
            if (dist[v] < best) {
                best = dist[v];
                nxt = i;
            }
        }
    }
    return sz ? -1 : cst;
}
```

Farmer John has been elected mayor of his town! One of his campaign promises was to bring internet connectivity to all farms in the area. He needs your help, of course.
Farmer John ordered a high speed connection for his farm and is going to share his connectivity with the other farmers. To minimize cost, he wants to lay the minimum amount of optical fiber to connect his farm to all the other farms.
Given a list of how much fiber it takes to connect each pair of farms, you must find the minimum amount of fiber needed to connect them all together. Each farm must connect to some other farm such that a packet can flow from any one farm to any other farm.
The distance between any two farms will not exceed 100,000.

Input

The input includes several cases. For each case, the first line contains the number of farms, N (3 <= N <= 100). The following lines contain the N x N conectivity matrix, where each element shows the distance from on farm to another. Logically, they are N lines of N space-separated integers. Physically, they are limited in length to 80 characters, so some lines continue onto others. Of course, the diagonal will be 0, since the distance from farm i to itself is not interesting for this problem.

Output

For each case, output a single integer length that is the sum of the minimum length of fiber required to connect the entire set of farms.

Sample

Input	Output
4 0 4 9 21 4 0 8 17 9 8 0 16 21 17 16 0	28

Meet in the middle

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

using namespace std;
const int N = 35;
int arr[N],a,b,n;

long long sum[1<<(N/2)];
int main() {
    scanf("%d%d%d",&n,&a,&b);
    for(int i=0;i<n;i++)
        scanf("%d",arr+i);
    int x=n/2,y=n-x;
    for(int i=0;i<(1<<x);i++)
        for(int j=0;j<x;j++)
            if((i>>j)&1)
                sum[i]+=arr[j];
    sort(sum,sum+(1<<x));
    long long ans=0;
    for(int i=0;i<(1<<y);i++){
        long long s=0;
        for(int j=0;j<y;j++)
            if((i>>j)&1)
                s+=arr[j+x];
        int l=lower_bound(sum,sum+(1<<x),a-s)-sum;
        int r=upper_bound(sum,sum+(1<<x),b-s)-sum;
        ans+=r-l;
    }
    printf("%lld\n",ans);
    return 0;
}
```

☆ Subset Sums

SPOJ - SUBSUMS

Given a sequence of N ($1 \le N \le 34$) numbers S_1, ..., S_N ($-20{,}000{,}000 \le S_i \le 20{,}000{,}000$), determine how many subsets of S (including the empty one) have a sum between A and B ($-500{,}000{,}000 \le A \le B \le 500{,}000{,}000$), inclusive.

Input

The first line of standard input contains the three integers N, A, and B. The following N lines contain S_1 through S_N, in order.

Output

Print a single integer to standard output representing the number of subsets satisfying the above property. Note that the answer may overflow a 32-bit integer.

Example

Input	Output
3 -1 2 1 -2 3	5

The following 5 subsets have a sum between -1 and 2:

- 0 = 0 (the empty subset)
- 1 = 1
- 1 + (−2) = −1
- −2 + 3 = 1
- 1 + (−2) + 3 = 2

Ternary Search

You are given a sequence of n integers $a_1, a_2, ..., a_n$.

Determine a real number x such that the *weakness* of the sequence $a_1 - x, a_2 - x, ..., a_n - x$ is as small as possible.

The *weakness* of a sequence is defined as the maximum value of the *poorness* over all segments (contiguous subsequences) of a sequence.

The *poorness* of a segment is defined as the absolute value of sum of the elements of segment.

Input

The first line contains one integer n ($1 \le n \le 200\,000$), the length of a sequence.

The second line contains n integers $a_1, a_2, ..., a_n$ ($|a_i| \le 10\,000$).

Output

Output a real number denoting the minimum possible *weakness* of $a_1 - x, a_2 - x, ..., a_n - x$. Your answer will be considered correct if its relative or absolute error doesn't exceed 10^{-6}.

Sample 1

Input	Output
3 1 2 3	1.000000000000000

Sample 2

Input	Output
4 1 2 3 4	2.000000000000000

Sample 3

Input	Output
10 1 10 2 9 3 8 4 7 5 6	4.500000000000000

Note

For the first case, the optimal value of x is 2 so the sequence becomes $-1, 0, 1$ and the max poorness occurs at the segment "-1" or segment "1". The poorness value (answer) equals to 1 in this case.

For the second sample the optimal value of x is 2.5 so the sequence becomes $-1.5, -0.5, 0.5, 1.5$ and the max poorness occurs on segment "-1.5 -0.5" or "0.5 1.5". The poorness value (answer) equals to 2 in this case.

```
const int N = 2e5 + 5;
int arr[N], n;

double weakness(double x) {
    double sumpos = 0, sumneg = 0, ans = 0;
    for (int i = 0; i < n; i++) {
        sumpos += arr[i] - x;
        sumneg += arr[i] - x;
        if (sumpos < 0)sumpos = 0;
        if (sumneg > 0)sumneg = 0;
        ans = max({ans, sumpos, -sumneg});
    }
    return ans;
}

double ts(double s, double e) {
    for (double sz = e - s; sz > 1e-15; sz *= 2 / 3.0) {
        double a = s + sz / 3, b = a + sz / 3;
        if (weakness(a) > weakness(b))
            s = a;
    }
    return s;
}

int main() {
    scanf("%d", &n);
    int mx = 0;
    for (int i = 0; i < n; i++)scanf("%d", arr + i), mx = max(mx, abs(arr[i]));
    printf("%.9f\n", weakness(ts(-mx, mx)));
    return 0;
}
```

Kth lexicographical

Consider an ordered set S of strings of N (1 <= N <= 31) bits. Bits, of course, are either 0 or 1.

This set of strings is interesting because it is ordered and contains all possible strings of length N that have L (1 <= L <= N) or fewer bits that are '1'.

Your task is to read a number I (1 <= I <= sizeof(S)) from the input and print the Ith element of the ordered set for N bits with no more than L bits that are '1'.

PROGRAM NAME: kimbits

INPUT FORMAT

A single line with three space separated integers: N, L, and I.

SAMPLE INPUT (file kimbits.in)

```
5 3 19
```

Copy

OUTPUT FORMAT

A single line containing the integer that represents the Ith element from the order set, as described.

SAMPLE OUTPUT (file kimbits.out)

```
10011
```

```
long long mem[33][33];

long long solve(int n, int l) {
    if (n < l) return 1ll << n; //same
    if (!l) return 1; //only possible answer is 0
    auto &ret = mem[n][l];
    if (~ret) return ret;
                //1... (so l-1)        0... (so same l)
    return ret = solve(n - 1, l - 1) + solve(n - 1, l);
}
int main() {
    int n, l;
    long long k;
    scanf("%d%d%lld", &n, &l,&k);
    memset(mem, -1, sizeof mem);
    while (n--){
        long long cnt = solve(n, l);
        if (k > cnt) {
            printf("1");
            k -= cnt;
            l--;
        } else printf("0");
    }
    puts("");
    return 0;
}
```

عندي كل الارقام بحسبها الاول شمالها 0
ثم شمالها 1 ف لو عندي الk اكبر من
الcnt يبقي انا في جزء ال1

مديني جراف وبيسال متوسط عدد الedges اللي
محتاجاها عشان يبقو component واحد

Eulerian Tour (idk, not sure ☹)

There are two serious problems in the Kingdom of Lipshire: the roads and the fools who build them. Once upon a time, the King of Lipshire has decided to improve the road system because some roads became completely impassable — it was easier to travel cross-country instead of using those roads.

By King's decree, new roads are to be built in Lipshire. Of course, the new road system must interconnect all towns, i. e. there must be a path connecting any two towns of Lipshire.

The road administration of Lipshire has resources to build exactly one road per year. Unfortunately, the fools who build these roads are completely out of control. So, regardless of the orders given, the fools randomly select two different towns a and b and build a road between them, even when those towns are already connected by a road. All possible choices are equiprobable. The road is build in such a manner that the only points where a traveler can leave it are the towns connected by this road. The only good thing is that all roads are bidirectional.

The King knows about the problem, but he cannot do anything about it. The only thing King needs to know is the expected number of years to wait before the road system of Lipshire becomes interconnected. He asked you to provide this information.

Input

The input will contain several test cases, each of them as described below. Consecutive test cases are separated by a single blank line.

The first line of the input contains two integers n and m ($2 \le n \le 30, 0 \le m \le 1000$) — the number of towns in Lipshire, and the number of roads which are still good. The following m lines describe roads, one per line. Each road is described with two endpoints — two integer numbers u_i and v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$). There can be multiple roads between two towns, but the road from a town to itself is not allowed.

Output

For each test case, output the expected number of years to wait for the interconnected road system. If the system is already interconnected, output zero as an answer. Output the number, on a line by itself, with at least six precise digits after the decimal point.

Sample Input

```
2 1
1 2

4 2
1 2
3 4
```

Sample Output

```
0.0
1.5
```

في كام طريقة اجيب المجموع n بمجموعة ارقام
(في كام رقم sorted المجموع بتاعهم n) sorted

```
int par[31], sz[31], n;

void init(int n) { //similar to DSU
    for (int i = 0; i < n; i++) {
        par[i] = -1;
        sz[i] = 1;
    }
}

int find(int x) { //similar to DSU
    if (par[x] == -1) return x;
    return par[x] = find(par[x]);
}
```

```
void join(int x, int y) { //similar to DSU
    x = find(x);
    y = find(y);
    if (x == y) return;
    if (sz[x] < sz[y]) swap(x, y);
    par[y] = x;
    sz[x] += sz[y];
}

map<vector<int>, double> mem;

double solve(vector<int> v) { //اللي هتحل المسألة
    if (v.size() == 1) return 0; // معناه ان الجراف كلو كومبوننت واحد فمش محتاجة اضيف حاجة
    auto &ret = mem.emplace(v, -1).first->second; //الفيرست تجيب البير والسكند تجيب العنصر نفسو
    if (ret != -1) return ret; //لو اتضاف فعلا -1 تبقي هي مش موجودة قبل كدا
    double q = 0, p = 0, cnt = n * (n - 1) / 2.0;
    for (int i = 0; i < v.size(); i++) {
        p += v[i] * (v[i] - 1) / 2.0; //احتمالية توصيل نفس العنصرداخل نفس الكومبوننت
        for (int j = i + 1; j < v.size(); j++) {
            vector<int> u = v;
            u[i] += v[j];
            u.erase(u.begin() + j);
            sort(u.begin(), u.end());
            q += v[i] * v[j] / cnt * (solve(u) + 1);
        }
    }
    p /= cnt;
    //ret=p*(ret+1)+q; //مينفعش اكتبه كدا فهحللها
    //ret-p*ret=p+q
    return ret = (p + q) / (1 - p);
}

int main() {
    int m;
    while (~scanf("%d%d", &n, &m)) {
        //mem.clear();
        init(n);
        while (m--) {
            int a, b;
            scanf("%d%d", &a, &b);
            join(--a, --b);
        }
        vector<int> v;
        for (int i = 0; i < n; i++)
            if (find(i) == i) v.push_back(sz[i]);
        sort(v.begin(), v.end());
        printf("%.9f\n", solve(v));
    }
    return 0;
}
```

احتمالية اني اوصل i بـ j علي عدد الedges اللي ممكن اختارها وعملت +1 بسبب الedge اللي انا زودته

عندي شوية جزر في البحر وفي كباري بتوصل بين الجزر دي ولو عديت علي كوبري في طريقي لازم اهده (مينفعش اعدي علي كوبري مرتين)

السؤال : هل ممكن اختار جزيرة ابدأ منها بحيث اهد كل الكباري؟ (ينفع اصلا ولا لا؟ ولو ينفع ابدا منين واخلص فين؟)

Eulerian Tour

Farmer John owns a large number of fences that must be repaired annually. He traverses the fences by riding a horse along each and every one of them (and nowhere else) and fixing the broken parts.

Farmer John is as lazy as the next farmer and hates to ride the same fence twice. Your program must read in a description of a network of fences and tell Farmer John a path to traverse each fence length exactly once, if possible. Farmer J can, if he wishes, start and finish at any fence intersection.

Every fence connects two fence intersections, which are numbered inclusively from 1 through 500 (though some farms have far fewer than 500 intersections). Any number of fences (>=1) can meet at a fence intersection. It is always possible to ride from any fence to any other fence (i.e., all fences are "connected").

Your program must output the path of intersections that, if interpreted as a base 500 number, would have the smallest magnitude.

There will always be at least one solution for each set of input data supplied to your program for testing.

PROGRAM NAME: fence

INPUT FORMAT

Line 1:	The number of fences, F (1 <= F <= 1024)
Line 2..F+1:	A pair of integers (1 <= i,j <= 500) that tell which pair of intersections this fence connects.

SAMPLE INPUT (file fence.in)

```
9
1 2
2 3
3 4
4 2
4 5
2 5
5 6
5 7
4 6
```

الحل للundirected : بحسب الdegree عند كل node

ولو عدد الnodes اللي الdegree بتاعتهم odd كان 0 او 2 (مع شرط ان الجراف connected) -> يبقي في حل

غير كدا يبقي مفيش حل!

لو كان odd عددهم 2 يبقي هبدأ من عند واحدة منهم والتانية اخلص عندها ولو كان 0 يبقي اي node هبدأ عندها اقدر اخلص عندها ومش هتفرق هبدأ عن مين

OUTPUT FORMAT

The output consists of F+1 lines, each containing a single integer. Print the number of the starting intersection on the first line, the next intersection's number on the next line, and so on, until the final intersection on the last line. There might be many possible answers to any given input set, but only one is ordered correctly.

SAMPLE OUTPUT (file fence.out)

```
1
2
3
4
2
5
4
6
5
7
```

الحل للdirected : بحسب الin degree و out عند كل node ثم احسب out-in

(مع شرط ان الجراف connected) لو كان الفرق لكل الnodes ب0 يبقي اقدر ابدا من اي حتة وانتهي عند نفس الnode اللي بدات منها

ولو كلو 0 لاكن في واحدة 1 وواحدة -1 يبقي لازم ابدأ من عند ال1 وانتهي عند ال-1

```cpp
int vis[1030]; // edges عاملاها لل
bool oddDeg[501]; //بتشوف هل الdegree هي odd ولا لا
      //v, id of edge
vector<pair<int, int>> adj[501]; // 500 عدد النودز دايما
vector<int> res;

void tour(int u) {
    while(adj[u].size()){
        auto [v, idx] = adj[u].back();
        adj[u].pop_back(); //كاني مسحته من الجراف
        if(vis[idx]) continue;
        vis[idx] = 1;
        tour(v);
    }
    res.push_back(u); //بحط وانا راجعة فالحل هيطلع بالعكس
}

int main() {
    int m, mn = 500;
    scanf("%d", &m);
    while (m--) {
        int u, v;
        scanf("%d%d", &u, &v);
        adj[u].push_back({v, m});
        adj[v].push_back({u, m});
        oddDeg[u] = !oddDeg[u];
        oddDeg[v] = !oddDeg[v];
        mn = min(mn, min(u, v));
    }
    for (int i = 1; i <= 500; i++)
        sort(adj[i].rbegin(), adj[i].rend()); //عملته بالعكس (فوق في الفانكشن شغالة من ورا لقدام)

    int cnt = count(oddDeg + 1, oddDeg + 501, true); //first, last, value
//    if( cnt > 2) {
//        puts("NO");
//        return 0;
//    }
    if (cnt)
        mn = find(oddDeg + 1, oddDeg + 501, true) - oddDeg;

    tour(mn);
    for(int i=res.size()-1; i>=0; i--)
        printf("%d\n", res[i]);
    return 0;
}
```

Note

المسألة دي undirected وفارض دايما ان في حل

ولو في اكتر من حل اطبع اللي بيطلع من الصغير للكبير

Note

في حالة الdirected مش هحتاج الvis ولا الpair اللي في adj وبدل الoddDeg هعمل واحدة in وواحدة out وفي الاخر احسب الفرق بينهم واتاكد ان ياما كلو 0 ياما في 1 و-1 مرة واحدة بس زي ما قولنا فوق

عملت dfs عادي لاكن الvis علي الedges مش الnodes

في المسألة دي فارضة ان دايما في حل لاكن الدكتور ضاف حتة لو مفيش حل نعمل ايه

في حالة الodd بدور علي واحدة منهم ابدأ من عندها