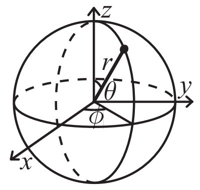
2.4.3 Spherical coordinates



$$x = r\sin\theta\cos\phi \qquad r = \sqrt{x^2+y^2+z^2}$$
$$y = r\sin\theta\sin\phi \qquad \theta = \operatorname{acos}(z/\sqrt{x^2+y^2+z^2})$$
$$z = r\cos\theta \qquad \phi = \operatorname{atan2}(y,x)$$

2.5 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$
$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$
$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$
$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\operatorname{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sums

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1}-c^a}{c-1}, c \neq 1$$

$$1+2+3+\cdots+n = \frac{n(n+1)}{2}$$
$$1^2+2^2+3^2+\cdots+n^2 = \frac{n(2n+1)(n+1)}{6}$$
$$1^3+2^3+3^3+\cdots+n^3 = \frac{n^2(n+1)^2}{4}$$
$$1^4+2^4+3^4+\cdots+n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, \ (-\infty < x < \infty)$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, \ (-1 < x \leq 1)$$
$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, \ (-1 \leq x \leq 1)$$
$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, \ (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, \ (-\infty < x < \infty)$$

2.8 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x-\mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX+bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX+bY) = a^2V(X) + b^2V(Y).$$

2.8.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n,p)$, $n = 1,2,\ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \ \sigma^2 = np(1-p)$$

$\text{Bin}(n,p)$ is approximately $\text{Po}(np)$ for small p.

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each wich yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1}, \ k = 1,2,\ldots$$

$$\mu = \frac{1}{p}, \ \sigma^2 = \frac{1-p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0,1,2,\ldots$$

$$\mu = \lambda, \ \sigma^2 = \lambda$$

2.8.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \ \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \ \sigma^2 = \frac{1}{\lambda^2}$$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef unsigned int uint;
typedef long double ld;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<int> vi;
typedef vector<ll> vll;
typedef vector<pii> vpii;
typedef vector<pll> vpll;
typedef vector<vi> vvi;
typedef vector<vll> vvll;

#define fst first
#define snd second
#define mp make_pair
#define pb push_back
#define all(x) x.begin(),x.end()
#define fill(a) memset(a,0,sizeof(a))
#define uni(c)
c.resize(distance(c.begin(),unique(all(c)))

#define rep(i,n) for(ll i=0,_n=(n); i<_n; i++)
#define rep1(i,n) for(ll i=1, _n=(n); i<=_n;
i++)
#define repA(i, a, n) for(ll i=a,_n=(n);
i<=_n; i++)
#define repD(i, a, n) for(ll i=a,_n=(n);
i>=_n; --i)
#define cout(d) cout << fixed <<
setprecision(d)

const ld eps=1e-12, pi=acosl(-1);
const ll inf=1e16, MOD=1e9+7;
const int N=2e6+10;
```

SIEVE

```
int np(0), prime[N], isprime[N];
void sieve(int N){
  memset(isprime, 1, sizeof isprime);
  for(int i=2; i<N; i++){
    if(isprime[i]){
      prime[++np] = i;
        for(int j=2*i; j<N; j+=i){
          isprime[j] = false; }}}}
```

BIG MULTIPLY

```
inline ll mul(ll a, ll b, ll m){
  ll q = ( (ld)a * (ld)b )/ (ld)m;
  ll r = a*b - q*m;
  return (r<0 ? r+m : r); }
```

LENGTH OF INT

```
int high(ll n){
  return 63 - __builtin_ctzll(n);
}
```

MILLER-RABIN

```
bool ispmiller(ll p){
  if(p<2) return false;
  if(p==2) return true;
  if(p%2==0) return false;
  ll s=p-1; s>>=__builtin_ctzll(s);
  rep(i,30){
    ll val=pwr(myrand(p-1)+1, s, p);
    ll temp=s;
    while(temp!=p-1 and 2<=val and
val<=p-2){
      val=mul(val,val,p);
      temp<<=1;}
    if(val!=p-1 && temp%2==0) return
false; }
  return true; }
```

DSU(INT) and DSU

```
struct dsu{
  vi par,sz;
  dsu(int n): par(n), sz(n) {iota(all(par),
0);}
  int root(int a){ return (par[a]==a) ? a:
par[a]=root(par[a]); }
  void merge(int a,int b){
    a=root(a), b=root(b);
    if(a==b) return;
    if(sz[a]<sz[b]) swap(a,b);
    sz[a]+=sz[b];
    par[b]=a;}};
template <class T> struct dsu1{
  map<T,T> par,sz;
  T root(T a){
    if(par.find(a)==par.end()||par[a]==a)
return par[a]=a;
    return par[a]=root(par[a]);
    }
  void merge(T a, T b){
    a=root(a); b=root(b);
    if(a==b) return;
    par[b]=a;}};
```

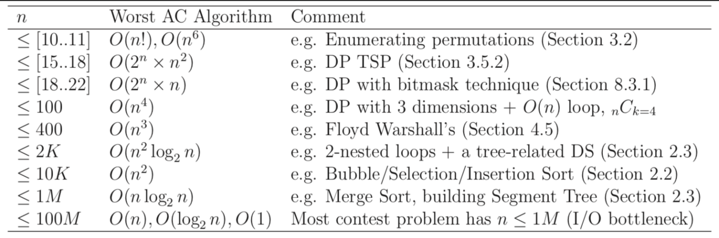
FAST IO

```
template<class T> void in(T&p) {
 p=0;char neg=0,ch=0;
 while(ch<0x30 or ch>0x39){
  if(ch=='-')neg=1; ch=getchar();}
  while(0x30<=ch and ch<=0x39){
   p=(p<<1)+(p<<3)+(ch&15);
   ch=getchar();}
   if(neg)p=-p; }

template<class T> void out(T n) {
 char d[20],i=0;
 if(n<0){putchar('-'); n=-n;}
 do{d[i++]=(n%10)|0x30; n/=10;} while(n);
 while(i) putchar(d[--i]);
 putchar('\n'); }
```

POLLARD-RHO

```
ll pollardrho(ll n){
  if(n==1) return 1;
  if(n%2==0) return 2;
  ll c=myrand(n-1) + 1;
  ll x=myrand(n-2) + 2, y=x;
  ll d=1;
  while(d==1){
    x=mul(x,x,n)+c; if(x>=n) x-=n;
    y=mul(y,y,n)+c; if(y>=n) y-=n;
    y=mul(y,y,n)+c; if(y>=n) y-=n;
    d=__gcd(abs(x-y),n);
    if(d==n) return ispmiller(n)? n:
pollardrho(n);}
  return d;}
```



n	Worst AC Algorithm	Comment
$\leq [10..11]$	$O(n!), O(n^6)$	e.g. Enumerating permutations (Section 3.2)
$\leq [15..18]$	$O(2^n \times n^2)$	e.g. DP TSP (Section 3.5.2)
$\leq [18..22]$	$O(2^n \times n)$	e.g. DP with bitmask technique (Section 8.3.1)
≤ 100	$O(n^4)$	e.g. DP with 3 dimensions + $O(n)$ loop, ${}_nC_{k=4}$
≤ 400	$O(n^3)$	e.g. Floyd Warshall's (Section 4.5)
$\leq 2K$	$O(n^2 \log_2 n)$	e.g. 2-nested loops + a tree-related DS (Section 2.3)
$\leq 10K$	$O(n^2)$	e.g. Bubble/Selection/Insertion Sort (Section 2.2)
$\leq 1M$	$O(n \log_2 n)$	e.g. Merge Sort, building Segment Tree (Section 2.3)
$\leq 100M$	$O(n), O(\log_2 n), O(1)$	Most contest problem has $n \leq 1M$ (I/O bottleneck)

Table 1.4: Rule of thumb time complexities for the 'Worst AC Algorithm' for various single-test-case input sizes n, assuming that your CPU can compute $100M$ items in 3s.

SEGTREE(RMQ)

```
class SegTree{
    private: vi st, A;
    int n;
    int left(int p){  return p<<1; }
    int right(int p){  return (p<<1)+1; }

    void build(int p, int L, int R){
        if(L==R) st[p]=L;
        else{
            build(left(p),L,(L+R)/2);
            build(right(p),(L+R)/2+1,R);
            int p1=st[left(p)],p2=st[right(p)];
            st[p]= (A[p1]<=A[p2])? p1: p2;
        }
    }

    int rmq(int p, int L, int R, int i, int j){
        if(i > R || j< L) return -1;
        if(L>=i && R<=j) return st[p];
        int p1=rmq(left(p),L,(L+R)/2,i,j);
        int p2=rmq(right(p),(L+R)/2+1,R,i,j);
        if (p1 == -1) return p2;
        if (p2 == -1) return p1;
        return (A[p1] <= A[p2]) ? p1 : p2;
    }
    public:
    SegTree(const vi &_A){
        A=_A; n=(int)A.size();
        st.assign(4*n,0);
        build(1,0,n-1);
    }
    int rmq(int i, int j){ return
rmq(1,0,n-1,i,j); }
};
```

BIT

```
int LSOne(int s){ return s&(-s); }

class FenwickTree{
    private: vi ft;
    public:
    FenwickTree(int n) { ft.assign(n + 1,
0);}
    int rsq(int b){
        int sum=0; for(;b;b-=LSOne(b))
sum+=ft[b];
        return sum;}
    int rsq(int a, int b){
        return rsq(b) - (a==1? 0 :
rsq(a-1));}
    void adjust(int k, int v){
        for(; k< (int)ft.size(); k+=
LSOne(k)) ft[k]+=v;}};
```

KMP

```
int lps[N];
void KMPPreProcess(char P[], int m){
    int i = 0, j = -1; lps[0] = -1;
    while(i<m){
        while(j>=0 && P[i] != P[j])
j=lps[j];
        i++; j++;
        lps[i]=j;
    }
}
void KMPSearch(char T[], char P[], int n,
int m){
    int i=0,j=0;
    while(i<n){
        while(j>=0 && T[i]!= P[j])
j=lps[j];
        i++; j++;
        if(j==m){
            printf("P is found at index
%d in T\n",i - j);
            j=lps[j];
        }
    }
```

GREEDY STRATEGY

- Coin change for Indian Currency or when $m_1, m_2, m_3, .., m_n$ *where* $m_i | m_{i+1}\ \forall i$
- Above strategy can be manipulated by listing ratios.
- Check load balancing.

DFS

```
vi dfs_num;
void dfs(vvi& AdjList, int u){
    dfs_num[u] = 1; int j;
    for(auto j: AdjList[u]) if(!
dfs_num[j]) dfs(AdjList,j);
}
```

BFS

```
vi d;
void bfs(vvi& AdjList, int s){
    queue<int> q; q.push(s); int j;
    while(!q.empty()){
        int u=q.front(); q.pop();
        for(auto j: AdjList[u]) if(d[j]==MOD)
d[j]=d[u]+1, q.push(j),cout<<j<<endl;
    }}
```

BRIDGES AND ARTICULATION POINTS

```
int dfsNumCounter,rootChildren,dfsRoot;
void BandA(vvi& AdjList,int u, int dfs_num[], int
dfs_low[], int dfs_parent[],bool art_vertex[]){
    dfs_low[u] = dfs_num[u] = dfsNumCounter++; int i;
    for(auto i: AdjList[u]){
        if(dfs_num[i]==0){
            dfs_parent[i]=u;
            if(u == dfsRoot) rootChildren++;

BandA(AdjList,i,dfs_num,dfs_low,dfs_parent,art_vertex
);
            if (dfs_low[i] >= dfs_num[u])
art_vertex[u] = true;
            if (dfs_low[i] > dfs_num[u]) printf("
Edge (%d, %d) is a bridge\n", u, i);
            dfs_low[u] = min(dfs_low[u], dfs_low[i]);
        }
        else if (i != dfs_parent[u]) dfs_low[u] =
min(dfs_low[u], dfs_num[i]);
    }
}
void BridgesandArticulation(vvi& AdjList,int V){
    int dfs_num[V],dfs_low[V],dfs_parent[V];
    bool art_vertex[V];
    int dfsNumCounter = 0;

fill(dfs_num),fill(dfs_low),fill(dfs_parent),fill(art
_vertex);
    rep(i,V) if (dfs_num[i] == 0){//BRIDGES
        dfsRoot = i; rootChildren = 0;
BandA(AdjList,i,dfs_num,dfs_low,dfs_parent,art_vertex
);
        art_vertex[dfsRoot] = (rootChildren > 1);}
    rep(i,V) if (art_vertex[i]){//ARTICULATIONS
        printf("Vertex %d\n",i);
    }
}
```

DJIKSTRA

```
void djikstra(vector<vpii>& AdjList, int s, int
V){
    vi dist(V,MOD); dist[s] = 0;
    //min-heap
    priority_queue< pii, vector<pii>,
greater<pii> > pq; pq.push(pii(0,s));
    while(!pq.empty()){
        pii front = pq.top(); pq.pop();
        int d = front.fst, u = front.snd;
        if(d > dist[u]) continue; pii v;
        for(auto v: AdjList[u]) if(dist[u] +
v.snd < dist[v.fst]){
            dist[v.fst] = dist[u] + v.snd;
            pq.push(pii(dist[v.fst],v.fst));
        }}}
```

BELLMAN FORD

```
void bellmanford(vector<vpii>& AdjList, int s,
int V){
    vi dist(V, MOD); dist[s]=0; pii v;
    rep(i,V-1) rep(u,V) for(auto v: AdjList[u])
dist[v.fst]=min(dist[v.fst],dist[u]+v.snd);
    bool NegCycle = false;
    rep(u,V) for(auto v: AdjList[u])
if(dist[v.fst] > dist[u] + v.snd) NegCycle =
true;
```

FLOYD WARSHALL

```
int V=400;
void floydwarshall(int AdjMat[400][400], int
V){
    rep(k,V) rep(i,V) rep(j,V) AdjMat[i][j]
= min(AdjMat[i][j], AdjMat[i][k]+AdjMat[k]
[j]);
```

In Nim game, two players take turns to remove objects from distinct heaps. On each turn, a player must remove at least one object and may remove any number of objects provided they all come from the same heap. The initial state of the game is the number of objects n_i at each of the k heaps: $\{n_1,n_2,...,n_k\}$. There is a nice solution for this game. For the first (starting) player to win, the value of n_1 ^ n_2 ^ ... ^ n_k must be non zero where ^ is the bit operator xor (exclusive or)—proof omitted.

GEOMETRY

```
struct point_i{
    int x,y;
    point_i(){ x = y = 0; }
    point_i(int _x, int _y) : x(_x), y(_y)
{}
};
struct point{
    point(){ x = y = 0.0; }
    point(ld _x, ld _y): x(_x), y(_y) {}
    bool operator < (point other) const{
        if(fabs(x-other.x) > eps) return x <
other.x;
        return y < other.y;
    }
    bool operator == (point other) const{
        return (fabs(x - other.x) < eps &&
fabs(y - other.y) < eps);
    }
};
double dist(point p1, point p2) return
hypot(p1.x-p2.x, p1.y-p2.y);
point rotate(point p, ld theta){
    ld rad = DEG_to_RAD(theta);
    return point(p.x*cos(rad) -
p.y*sin(rad), p.x*sin(rad) + p.y*cos(rad));
}
double area(const vector<point>& P){
    ld result = 0.0, x1, y1, x2, y2;
    rep(i,(int)P.size()-1){
        x1 = P[i].x; x2 = P[i+1].x;
        y1 = P[i].y; y2 = P[i+1].y;
        result += (x1 * y2 - x2 * y1);
    }
    return fabs(result) / 2.0 ;
}
```

ACM/ICPC CheatSheet

Puzzles

Contents

1 STL Useful Tips

1.1 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
```

1.2 Modifying sequence operations

```
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2); // swap range
void replace(first, last, old_value, new_value); // replace in range
void replace_if(first, last, pred, new_value); // replace in conditions
  // pred can be represented in function
  // e.x. bool IsOdd (int i) { return ((i%2)==1); }
void reverse(first, last); // reverse a range of elements
void reverse_copy(first, last, result); // copy a reverse of range of elements
void random_shuffle(first, last); // using built-in random generator to shuffle array
```

1.3 Merge

```
// merge sorted ranges
void merge(first1, last1, first2, last2, result, comp);
// union of two sorted ranges
void set_union(first1, last1, first2, last2, result, comp);
// intersection of two sorted ranges
void set_interaction(first1, last1, first2, last2, result, comp);
// difference of two sorted ranges
void set_difference((first1, last1, first2, last2, result, comp);
```

1.4 String

```
// Searching
unsigned int find(const string &s2, unsigned int pos1 = 0);
unsigned int rfind(const string &s2, unsigned int pos1 = end);
unsigned int find_first_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_of(const string &s2, unsigned int pos1 = end);
unsigned int find_first_not_of(const string &s2, unsigned int pos1 = 0);
unsigned int find_last_not_of(const string &s2, unsigned int pos1 = end);
// Insert, Erase, Replace
string& insert(unsigned int pos1, const string &s2);
string& insert(unsigned int pos1, unsigned int repetitions, char c);
string& erase(unsigned int pos = 0, unsigned int len = npos);
string& replace(unsigned int pos1, unsigned int len1, const string &s2);
string& replace(unsigned int pos1, unsigned int len1, unsigned int repetitions, char c);
// String streams
stringstream s1;
int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

1.5 Heap

```
template <class RandomAccessIterator>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void push_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
```

```
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void pop_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);

template <class RandomAccessIterator>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void make_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp );

template <class RandomAccessIterator>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  void sort_heap (RandomAccessIterator first, RandomAccessIterator last,
          Compare comp);
template <class RandomAccessIterator>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last);
template <class RandomAccessIterator, class Compare>
  RandomAccessIterator is_heap_until (RandomAccessIterator first,
                    RandomAccessIterator last
                    Compare comp);
```

1.6 Sort

```
void sort(iterator first, iterator last);
void sort(iterator first, iterator last, LessThanFunction comp);
void stable_sort(iterator first, iterator last);
void stable_sort(iterator first, iterator last, LessThanFunction comp);
void partial_sort(iterator first, iterator middle, iterator last);
void partial_sort(iterator first, iterator middle, iterator last, LessThanFunction comp);
bool is_sorted(iterator first, iterator last);
bool is_sorted(iterator first, iterator last, LessThanOrEqualFunction comp);
// example for sort, if have array x, start_index, end_index;
sort(x+start_index, x+end_index);

/** sort a map **/
// You cannot directly sort a map<key type, mapped data type>
// if you only want to sort in key type
// you can use insert method to copy map into another map
// b.insert(make_pair(it->first, it->second) /* it is a map iterator */
// this will result a map which sorts key type in increasing order
// if you want to sort key type in decreasing order, then declare your map as
// something like:
// map<char, int, greater<char> >

// if you want to sort based on key, you need to copy the data to a vector
// where elements of vector are pair.
// you can define a PAIR type by using:
typedef pair<char, int> PAIR;

// suppose this is the map
map<char, int> a;

// sort vector in decreasing order
bool cmp_by_value(const PAIR& lhs, const PAIR& rhs) {
  return lhs.second > rhs.second;
}

// sort key in increasing order
bool cmp_by_char(const PAIR& lhs, const PAIR& rhs) {
  return lhs.first < rhs.first;
}

// copy map data to vector
vector<PAIR> b(a.begin(), a.end());
```

```
// sort data
sort(b.begin(), b.end(), cmp_by_value);

// you can still call your data by b[i].first and b[i].second.
// THE ABOVE CODES ARE EXAMPLE FOR SORTING A MAP.
// PLEASE USE IT FOR YOUR OWN DEMANDS.
```

1.7 Permutations

```
bool next_permutation(iterator first, iterator last);
bool next_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
bool prev_permutation(iterator first, iterator last);
bool prev_permutation(iterator first, iterator last, LessThanOrEqualFunction comp);
```

1.8 Searching

```
// will return address of iterator, call result as *iterator;
iterator find(iterator first, iterator last, const T &value);
iterator find_if(iterator first, iterator last, const T &value, TestFunction test);
bool binary_search(iterator first, iterator last, const T &value);
bool binary_search(iterator first, iterator last, const T &value, LessThanOrEqualFunction comp);
```

1.9 Random algorithm

```
srand(time(NULL));
// generate random numbers between [a,b)
rand() % (b - a) + a;
// generate random numbers between [0,b)
rand() % b;
// generate random permutations
random_permutation(anArray, anArray + 10);
random_permutation(aVector, aVector + 10);
```

2 Number Theory

2.1 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;
}
```

2.2 Generate combinations

```cpp
// 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;
  }
}
```

2.3 10-ary to m-ary

```cpp
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;
}
```

2.4 m-ary to 10-ary

```cpp
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;
}
```

2.5 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];
}
```

2.6 Catalan numbers

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

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.

2.7 Eulerian numbers

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

```
// 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];
}
```

2.8 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;
}
```

3 Searching Algorithms

3.1 KMP Algorithm

```cpp
#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();
}

int main(void)
{
  string a = (string) "The example above illustrates the general technique for assembling "+
    "the table with a minimum of fuss. The principle is that of the overall search: "+
    "most of the work was already done in getting to the current position, so very "+
    "little needs to be done in leaving it. The only minor complication is that the "+
    "logic which is correct late in the string erroneously gives non-proper "+
    "substrings at the beginning. This necessitates some initialization code.";

  string b = "table";

  int p = KMP(a, b);
  cout << p << ": " << a.substr(p, b.length()) << " " << b << endl;

  return 0;
}
```

3.2 Manachar's Algorithm

```
#include <bits/stdc++.h>
using namespace std;
#define SIZE 100000 + 1

int P[SIZE * 2];

// Transform S into new string with special characters inserted.
string convertToNewString(const string &s) {
    string newString = "@";

    for (int i = 0; i < s.size(); i++) {
        newString += "#" + s.substr(i, 1);
    }

    newString += "#$";
    return newString;
}

string longestPalindromeSubstring(const string &s) {
    string Q = convertToNewString(s);
    int c = 0, r = 0;                // current center, right limit

    for (int i = 1; i < Q.size() - 1; i++) {
        // find the corresponding letter in the palidrome subString
        int iMirror = c - (i - c);

        if(r > i) {
            P[i] = min(r - i, P[iMirror]);
        }

        // expanding around center i
        while (Q[i + 1 + P[i]] == Q[i - 1 - P[i]]){
            P[i]++;
        }

        // Update c,r in case if the palindrome centered at i expands past r,
        if (i + P[i] > r) {
            c = i;              // next center = i
            r = i + P[i];
        }
    }

    // Find the longest palindrome length in p.

    int maxPalindrome = 0;
    int centerIndex = 0;

    for (int i = 1; i < Q.size() - 1; i++) {

        if (P[i] > maxPalindrome) {
            maxPalindrome = P[i];
            centerIndex = i;
        }
    }

    cout << maxPalindrome << "\n";
    return s.substr( (centerIndex - 1 - maxPalindrome) / 2, maxPalindrome);
}

int main() {
    string s = "kiomaramol\n";
    cout << longestPalindromeSubstring(s);
    return 0;
}
```

4 Dynamic Programming

4.1 0/1 Knapsack problems

```
#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

```
#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;
}
```

4.3 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];
}
```

4.4 Longest increasing common sequence (LICS)

```
#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)

```
#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;
}

int main(void)
{
   int n=0;
   cin >> n;
   for (int i=1;i<=n;i++)
      for (int j=1;j<=n;j++)
         cin >> a[i][j];

   cout << maxmatrix(n);
   return 0;
}
```

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 Trees

5.1 Depth and width of tree

```
#include <iostream>
#include <queue>
#include <stack>

using namespace std;

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);
  }
}

int main(void)
{
  cin >> n;

  for (int i=1;i<=n;i++)
    cin >> l[i] >> r[i];

  d=depth(1);
  mystack.push(1);
  width(1);

  cout << w << " " << d << endl;

  return 0;
}
```

6 Segment Trees

6.1 Segment Tree

```
template <class T>
class Segtree {
//Non-Lazy Segment Tree
private :
  vector<T> _tree;
  int _size;
  T _basis;
  T (*_relator)(T, T);

  void make(const vector<T>& arr, int l, int r, int i) {
  if (l == r) { _tree[i] = arr[l]; return; }
      int m = (l + r) / 2;
    make(arr, l, m, 2 * i + 1);
    make(arr, m + 1, r, 2 * i + 2);
    _tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  T rq(int l1, int r1, int l, int r, int i) {
    if (r1 < l1) { return _basis; }
    if (l1 == l && r1 == r) { return _tree[i]; }
    int m = (l + r) / 2;
    T a = rq(l1, min(m, r1), l, m, 2 * i + 1);
    T b = rq(max(m + 1, l1), r1, m + 1, r, 2 * i + 2);
    return _relator(a, b);
  }
  void ru(int l1, int r1, T x, int l, int r, int i) {
    //increases all values in [l1, r1] by x
    if (r1 < l1) { return; }
    if (l == r) { _tree[i] += x; return; }
    int m = (l + r) / 2;
    ru(l1, min(m, r1), x, l, m, 2 * i + 1);
    ru(max(m + 1, l1), r1, x, m + 1, r, 2 * i + 2);
    _tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  void upd(int p, T x, int l, int r, int i) {
    //updates value at position p to x
    if (p < l && r < p) { return; }
    if (l == r) { _tree[i] = x; return; }
    int m = (l + r) / 2;
    upd(p, x, l, m, 2 * i + 1);
    upd(p, x, m + 1, r, 2 * i + 2);
    _tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  void pr(int l, int r, int i) {
    if (l == r) { cout << _tree[i] << ' '; return; }
    int m = (l + r) / 2;
    pr(l, m, 2 * i + 1);
    pr(m + 1, r, 2 * i + 2);
  }

public :
  Segtree(const vector<T>& arr, T f(T, T), T b) {
    _size = arr.size();
    _tree = vector<T>(4 * _size);
    _basis = b;
    _relator = f;
    //_tree[0] = _relator(arr[0], arr[1]);
    make(arr, 0, _size - 1, 0);
  }
  T rq(int l1, int r1) {
    return rq(l1, r1, 0, _size - 1, 0);
  }
  void ru(int l1, int r1, T x) {
    ru(l1, r1, x, 0, _size - 1, 0);
  }
  void upd(int p, T x) {
    upd(p, x, 0, _size - 1, 0);
  }
  void pr_tree() {
    for (int i = 0; i < _tree.size(); i++) { cout << _tree[i] << ' '; }
    cout << endl;
  }
  void pr() {
    pr(0, _size - 1, 0);
    cout << endl;
  }
};
```

6.2 Segment Tree with Lazy Propagation

```
template <class T>
class Lazytree {
//Lazy Segment Tree
//It is self assumed that relator is adder
//value stored in a lazy node means that node has been updated
//but the update needs to passed to its children
//this is made so because since we never descend further down
//from the leaves so the lazy values stored at the leaves
//must mean that the leaves have been updated and the update
//value needs to be passed down
private :
  vector<T> _tree;
  vector<T> _lazy;
  int _size;
  T _basis;
  T (*_relator)(T, T);

  void make(const vector<T>& arr, int l, int r, int i) {
    if (l == r) { _tree[i] = arr[l]; return; }
    int m = (l + r) / 2;
    make(arr, l, m, 2 * i + 1);
    make(arr, m + 1, r, 2 * i + 2);
    _tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  void passdown(int l, int r, int i) {
    int m = (l + r) / 2;
    _tree[2 * i + 1] += (m - l + 1) * _lazy[i];
    _tree[2 * i + 2] += (r - m) * _lazy[i];
    _lazy[2 * i + 1] += _lazy[i];
    _lazy[2 * i + 2] += _lazy[i];
    _lazy[i] = 0;
  }
  T rq(int l1, int r1, int l, int r, int i) {
    if (r1 < l1) { return _basis; }
    if (l1 == l && r1 == r) { return _tree[i]; }
    passdown(l, r, i);
    int m = (l + r) / 2;
    T a = rq(l1, min(m, r1), l, m, 2 * i + 1);
    T b = rq(max(m + 1, l1), r1, m + 1, r, 2 * i + 2);
    return _relator(a, b);
  }
  void ru(int l1, int r1, T x, int l, int r, int i) {
    //increases all values in [l1, r1] by x
    if (r1 < l1) { return; }
    if (l == l1 && r == r1) {
      _tree[i] += (r - l + 1) * x;
      _lazy[i] += x;
      return;
    }
    passdown(l, r, i);
    int m = (l + r) / 2;
    ru(l1, min(m, r1), x, l, m, 2 * i + 1);
    ru(max(m + 1, l1), r1, x, m + 1, r, 2 * i + 2);
    //_tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  void upd(int p, T x, int l, int r, int i) {
    //updates value at position p to x
    if (p < l || r < p) { return; }
    if (l == r) { _tree[i] = x; return; }
    passdown(l, r, i);
    int m = (l + r) / 2;
    upd(p, x, l, m, 2 * i + 1);
    upd(p, x, m + 1, r, 2 * i + 2);
    _tree[i] = _relator(_tree[2 * i + 1], _tree[2 * i + 2]);
  }
  void pr(int l, int r, int i) {
    if (l == r) { cout << _tree[i] << ' '; return; }
    passdown(l, r, i);
    int m = (l + r) / 2;
    pr(l, m, 2 * i + 1);
    pr(m + 1, r, 2 * i + 2);
  }

public :
  Lazytree(const vector<T>& arr, T f(T, T), T b) {
    _size = arr.size();
    _tree = vector<T>(4 * _size);
    _lazy = vector<T>(4 * _size, 0);
    _basis = b;
    _relator = f;
    //_tree[0] _relator(arr[0], arr[1]);
    make(arr, 0, _size - 1, 0);
  }
  T rq(int l1, int r1) {
    return rq(l1, r1, 0, _size - 1, 0);
  }
  void ru(int l1, int r1, T x) {
    ru(l1, r1, x, 0, _size - 1, 0);
  }
  void upd(int p, T x) {
    upd(p, x, 0, _size - 1, 0);
  }
  void pr_tree() {
    for (int i = 0; i < _tree.size(); i++) { cout << _tree[i] << ' '; }
    cout << endl;
  }
  void pr() {
    pr(0, _size - 1, 0);
    cout << endl;
  }
};
```

7 Graph Theory

7.1 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)
```

7.2 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;
  }
}
```

7.3 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];
}
```

7.4 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;
}
```

7.5 Eulerian circuit

```
// 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;
}
```

7.6 Topological sort

```
// 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;
}
```

```
// 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;
}
```