

# CodeBook by 櫛風

## CodeBook by 櫛風

### Python

slice

swap

input

### Dark\_Code

rope

thread

default\_code

faster\_input

random\_shuffle

### OOP

function

BigInteger

Matrix

fraction

### JAVA

BigInteger

default\_code

build\_and\_run

### Algorithm

dijkstra

0-1Knapsack

Math

Newton\_Raphson\_Method

chinese\_remainder

fast\_power

gcd\_extended

pollard\_rho

fast\_multi

gcd

Miller\_Rabin

mod\_inverse

SquareNumber

# Python

## slice

```
1 | a[begin:end:step]
```

## swap

```
1 | a,b=(b,a)
```

## input

```
1 | a,b=input().split(" ")
```

# Dark\_Code

## rope

```
1 #include<ext/rope>
2 using namespace __gnu_cxx;
3 rope<char>str;
```

## thread

```
1 #include<Thread>
2 thread t([](int i){cout<<i<<endl;},1);
3 t.join();
```

## default\_code

```
1 #include<bits/stdc++.h>
2
3 #define debug(x) #x<<": "<<x<<" "
4 #define endl "\n"
5 #define num long long
6
7 using namespace std;
8
9 int main(){
10     cin.tie(0);
11     ios_base::sync_with_stdio(0);
12
13     return 0;
14 }
```

## faster\_input

```
1 template<typename T>inline T input(){
2     T sum=0,fl=1;
3     int ch=getchar();
4     for(;!isdigit(ch);ch=getchar())if(ch=='-')fl=-1;
5     for(;isdigit(ch);ch=getchar())sum=sum*10+ch-'0';
6     return sum*fl;
7 }
8 int a=input<int>();
```

## random\_shuffle

```
1 #include<algorithm>
2 #include<cstdlib>
3 #include<ctime>
4 srand(time(0));
5 random_shuffle(a.begin(),a.end());
```

# OOP

## function

```
1 #include<vector>
2 class func:std::vector<double>{
3 public:
4     func(int n=0){resize(n);}
5     func(std::vector<double>a){
6         resize(a.size());
7         for(int i=0;i<a.size();i++)at(i)=a[i];
8     }
9     void print(){
10         bool first=true;
11         for(int i=size()-1;i>=0;i--){
12             if(at(i)){
13                 std::cout<<((first)?(first=false,""):"+")
14                     <<at(i)<<((i>=1)?"X":"" );
15                 if(i>1)std::cout<<"^"<<i;
16             }
17         }
18         std::cout<<std::endl;
19     }
20     double setX(double x){
21         double ans=0;
22         double t=1;
23         for(int i=0;i<size();i++){
24             ans+=at(i)*t;
25             t*=x;
26         }
27         return ans;
28     }
29     friend func prime(func a){
30         func b;
31         for(int i=1;i<a.size();i++){
32             b.push_back(a[i]*i);
33         }
34         return b;
35     }
```

## BigInteger

```

1  #include<algorithm>
2  #include<sstream>
3  #include<vector>
4  #include<cmath>
5  #include<iomanip>
6  class bigN:std::vector<long long>{
7  private:
8      const static int base=1000000000,width=log10(base);
9      bool negative;
10     bigN convert_base(int old_width,int new_width)const{
11         vector<long long>p(std::max(old_width,new_width)+1,1);
12         for(size_t i=1;i<p.size();++i)p[i]=p[i-1]*10;
13         bigN ans;
14         long long cur=0;
15         int cur_id=0;
16         for(size_t i=0;i<size();++i){
17             cur+=at(i)*p[cur_id];
18             cur_id+=old_width;
19             while(cur_id>=new_width){
20                 ans.push_back(cur%p[new_width]);
21                 cur/=p[new_width];
22                 cur_id-=new_width;
23             }
24         }
25         return ans.push_back(cur),ans.trim(),ans;
26     }
27     bigN karatsuba(const bigN &b)const{
28         bigN res;
29         res.resize(size()*2);
30         if(size()<=32){
31             for(size_t i=0;i<size();++i){
32                 for(size_t j=0;j<size();++j){
33                     res[i+j]+=at(i)*b[j];
34                 }
35             }
36             return res;
37         }
38         size_t k=size()/2;
39         bigN a1(begin(),begin()+k);
40         bigN a2(begin()+k,end());
41         bigN b1(b.begin(),b.begin()+k);
42         bigN b2(b.begin()+k,b.end());
43         bigN a1b1=a1.karatsuba(b1);
44         bigN a2b2=a2.karatsuba(b2);
45         for(size_t i=0;i<k;++i)a2[i]+=a1[i];
46         for(size_t i=0;i<k;++i)b2[i]+=b1[i];
47         bigN r=a2.karatsuba(b2);

```

```

48     for(size_t i=0;i<a1b1.size();++i)r[i]-=a1b1[i];
49     for(size_t i=0;i<a2b2.size();++i)r[i]-=a2b2[i];
50     for(size_t i=0;i<r.size();++i)res[i+k]+=r[i];
51     for(size_t i=0;i<a1b1.size();++i)res[i]+=a1b1[i];
52     for(size_t i=0;i<a2b2.size();++i)res[i+size()]+=a2b2[i];
53     return res;
54 }
55 void trim(){
56     while(size()&&!back())pop_back();
57     if(empty())negative=0;
58 }
59 void carry(int _base=base){
60     for(size_t i=0;i<size();++i){
61         if(at(i)>=0&&at(i)<_base)continue;
62         if(i+1u==size())push_back(0);
63         int r=at(i)%_base;
64         if(r<0)r+=_base;
65         at(i+1)+=(at(i)-r)/_base;
66         at(i)=r;
67     }
68 }
69 int abscmp(const bigN &b)const{
70     if(size()>b.size())return 1;
71     if(size()<b.size())return -1;
72     for(int i=int(size())-1;i>=0;--i){
73         if(at(i)>b[i])return 1;
74         if(at(i)<b[i])return -1;
75     }
76     return 0;
77 }
78 int cmp(const bigN &b)const{
79     if(negative!=b.negative)return negative?-1:1;
80     return negative?-abscmp(b):abscmp(b);
81 }
82 bigN abs()const{
83     bigN res=*this;
84     return res.negative=0,res;
85 }
86 public:
87     bigN():negative(0){}
88     bigN(const_iterator a,const_iterator b):vector<long long>(a,b){}
89     bigN(std::string s){
90         if(s.empty())return;
91         if(s[0]=='-')negative=1,s=s.substr(1);
92         else negative=0;
93         for(int i=int(s.size())-1;i>=0;i-=width){
94             long long t=0;
95             for(int j=std::max(0,i-width+1);j<=i;++j)
96                 t=t*10+s[j]-'0';
97             push_back(t);
98         }
99         trim();

```

```

100     }
101     template<typename T> bigN(const T &x){
102         std::stringstream ss;
103         ss<<x;
104         *this=ss.str();
105     }
106     bool operator<(const bigN&b)const{return cmp(b)<0;}
107     bool operator>(const bigN&b)const{return cmp(b)>0;}
108     bool operator<=(const bigN&b)const{return cmp(b)<=0;}
109     bool operator>=(const bigN&b)const{return cmp(b)>=0;}
110     bool operator==(const bigN&b)const{return !cmp(b);}
111     bool operator!=(const bigN&b)const{return cmp(b)!=0;}
112     bigN operator-(const bigN res=*this;return res.negative=!negative,res.trim(),res;}
113     bigN operator+(const bigN &b)const{
114         if(negative)return -(-( *this)+(-b));
115         if(b.negative)return *this-(-b);
116         bigN res=*this;
117         if(b.size()>size())res.resize(b.size());
118         for(size_t i=0;i<b.size();++i)res[i]+=b[i];
119         return res.carry(),res.trim(),res;
120     }
121     bigN operator-(const bigN &b)const{
122         if(negative)return -(-( *this)-(-b));
123         if(b.negative)return *this+(-b);
124         if(abscmp(b)<0)return -(b-( *this));
125         bigN res=*this;
126         if(b.size()>size())res.resize(b.size());
127         for(size_t i=0;i<b.size();++i)res[i]-=b[i];
128         return res.carry(),res.trim(),res;
129     }
130     bigN operator*(const bigN &b)const{
131         const static int mul_base=1000000,mul_width=log10(mul_base);
132         bigN A=convert_base(width,mul_width);
133         bigN B=b.convert_base(width,mul_width);
134         int n=std::max(A.size(),B.size());
135         while(n&(n-1))++n;
136         A.resize(n),B.resize(n);
137         bigN res=A.karatsuba(B);
138         res.negative=negative!=b.negative;
139         res.carry(mul_base);
140         res=res.convert_base(mul_width,width);
141         return res.trim(),res;
142     }
143     bigN operator*(long long b)const{
144         bigN res=*this;
145         long long a;
146         if(b<0)res.negative=!negative,b=-b;
147         for(size_t i=0,is=0;i<res.size()||is;++i){
148             if(i==res.size())res.push_back(0);
149             a=res[i]*b+is;
150             is=a/base;
151             res[i]=a%base;

```

```

152     }
153     return res.trim(),res;
154 }
155 bigN operator/(const bigN &b)const{
156     int norm=base/(b.back()+1);
157     bigN x=abs()*norm;
158     bigN y=b.abs()*norm;
159     bigN q,r;
160     q.resize(x.size());
161     for(int i=int(x.size())-1;i>=0;--i){
162         r=r*base+x[i];
163         int s1=r.size()<=y.size()?0:r[y.size()];
164         int s2=r.size()<y.size()?0:r[y.size()-1];
165         int d=((long long)(base)*s1+s2)/y.back();
166         r=r-y*d;
167         while(r.negative)r=r+y,--d;
168         q[i]=d;
169     }
170     q.negative=negative!=b.negative;
171     return q.trim(),q;
172 }
173 bigN operator%(const bigN &b)const{return *this-(*this/b)*b;}
174 bigN operator<<(const int &b)const{
175     bigN res=*this;
176     for(int i=0;i<b;i++)res*=2;
177     return res.carry(),res.trim(),res;
178 }
179 bigN operator>>(const int &b)const{
180     bigN res=*this;
181     for(int i=0;i<b;i++)res/=2;
182     return res.carry(),res.trim(),res;
183 }
184 friend std::istream& operator>>(std::istream &ss,bigN &b){
185     std::string s;
186     return ss>>s,b=s,ss;
187 }
188 friend std::ostream& operator<<(std::ostream &ss,const bigN &b){
189     if(b.negative)ss<<"-";
190     ss<<(b.empty()?0:b.back());
191     for(int i=int(b.size())-2;i>=0;--i)
192         ss<<std::setw(width)<<std::setfill('0')<<b[i];
193     return ss;
194 }
195 template<typename T>operator T(){
196     std::stringstream ss;
197     ss<<*this;
198     T res;
199     return ss>>res,res;
200 }
201 friend bigN abs(bigN a){return a.abs();}
202 bigN operator+=(const bigN &other){*this=(*this)+(other);return *this;}
203 bigN operator-=(const bigN &other){*this=(*this)-(other);return *this;}

```



```

204     bigN operator*=(const bigN &other){*this=(*this)*(other);return *this;}
205     bigN operator/=(const bigN &other){*this=this->operator/(other);return *this;}
206     bigN operator<=<=(const bigN &other){*this=(*this)<<(other);return *this;}
207     bigN operator>>=(const bigN &other){*this=(*this)>>(other);return *this;}
208     bigN operator++(){(*this)+=1;return *this;}
209     bigN operator++(int){*this+=1;return *this;}
210     bigN operator--(){*this-=1;return *this;}
211     bigN operator--(int){*this-=1;return *this;}
212     bool operator!(){return (*this)==0;}
213 };

```

## Matrix

```

1  #include <iostream>
2  #include <iomanip>
3  #include <cmath>
4  #include <vector>
5  #include <cassert>
6
7  class Matrix:std::vector<std::vector<double>>{
8  private:
9      int maxL;
10 public:
11     Matrix(int n,int m,double a=0){
12         resize(n);
13         maxL=1;
14         for(int i=0;i<size();i++){
15             at(i).resize(m);
16             if(n<=m)at(i).at(i)=a;
17         }
18     }
19     Matrix(std::vector<std::vector<double>>a){
20         int t;
21         resize(a.size());
22         for(int i=0;i<size();i++){
23             at(i).resize(a[i].size());
24             for(int j=0;j<a[i].size();j++){
25                 at(i).at(j)=a[i][j];
26                 t=log10(abs(at(i).at(j)))+1;
27                 if(at(i).at(j)<0)t++;
28                 if(t>maxL)maxL=t;
29             }
30         }
31     }
32     int row()const{return size();}
33     int column()const{return at(0).size();}
34     bool isSquire(){return row()==column();}
35     void operator+=(Matrix other){*this=*this+other;}
36     void operator-=(Matrix other){*this=*this-other;}
37     void operator*=(Matrix other){*this=*this*other;}
38     void operator^=(int times){*this=*this^times;}

```

```

39 void print(){
40     for(int i=0;i<row();i++){
41         if(i==0 && i==row()-1)std::cout<<"[";
42         else if(i==0)std::cout<<"[";
43         else if(i==row()-1)std::cout<<"[";
44         else std::cout<<"|";
45         for(int j=0;j<column();j++){
46             std::cout<<std::setw(maxL)<<at(i).at(j);
47             if(j==column()-1){
48                 if(i==0 && i==row()-1)std::cout<<"]";
49                 else if(i==0)std::cout<<"]";
50                 else if(i==row()-1)std::cout<<"]";
51                 else std::cout<<"|";
52             }
53             else std::cout<<" ";
54         }
55         std::cout<<std::endl;
56     }
57 }
58 friend std::ostream & operator<<(std::ostream &out,const Matrix &a){
59     for(int i=0;i<a.row();i++){
60         for(int j=0;j<a.column();j++){
61             out<<std::setw(a.maxL)<<a[i][j]<<" \n"[j==a.column()-1];
62         }
63     }
64     return out;
65 }
66 friend std::istream & operator>>(std::istream &in,Matrix &a){
67     int t;
68     for(int i=0;i<a.row();i++){
69         for(int j=0;j<a.column();j++){
70             in>>a[i][j];
71             t=log10(abs(a[i][j]))+1;
72             if(a[i][j]<0)t++;
73             if(t>a.maxL)a.maxL=t;
74         }
75     }
76     return in;
77 }
78 friend int operator==(Matrix a,Matrix b){
79     if(a.row()!=b.row() || a.column()!=b.column())return 0;
80     for(int i=0;i<a.row();i++){
81         for(int j=0;j<a.column();j++){
82             if(a[i][j]!=b[i][j])return -1;
83         }
84     }
85     return 1;
86 }
87 friend bool operator!=(Matrix a,Matrix b){return !(a==b);}
88 friend Matrix operator+(Matrix a,Matrix b){
89     assert(a==b);
90     std::vector<std::vector<double>>>c;

```

```

91     c.resize(a.row());
92     for(int i=0;i<a.row();i++){
93         c[i].resize(a.column());
94         for(int j=0;j<a.column();j++){
95             c[i][j]=a[i][j]+b[i][j];
96         }
97     }
98     Matrix ans(c);
99     return ans;
100 }
101 friend Matrix operator-(Matrix a,Matrix b){return a+(-1*b);}
102 friend Matrix operator*(Matrix a,double t){
103     Matrix b(a.row(),a.column(),t);
104     return a*b;
105 }
106 friend Matrix operator*(double t,Matrix a){return a*t;}
107 friend Matrix operator*(Matrix a,Matrix b){
108     assert(a==b);
109     std::vector<std::vector<double>>>c;
110     c.resize(a.row());
111     for(int i=0;i<a.row();i++){
112         c[i].resize(a.column());
113         for(int j=0;j<a.column();j++){
114             for(int z=0;z<a.column();z++){
115                 c[i][j]+=a[i][z]*b[z][j];
116             }
117         }
118     }
119     Matrix ans(c);
120     return ans;
121 }
122 friend Matrix operator^(Matrix a,int t){
123     if(t==-1)return inverse(a);
124     assert(t>0);
125     Matrix b=a;
126     while(--t)b=b*a;
127     return b;
128 }
129 friend Matrix T(Matrix a){
130     std::vector<std::vector<double>>> c;
131     c.resize(a.column());
132     for(int i=0;i<a.column();i++){
133         c[i].resize(a.row());
134         for(int j=0;j<a.row();j++){
135             c[i][j]=a[j][i];
136         }
137     }
138     Matrix ans(c);
139     return ans;
140 }
141 friend Matrix inverse(Matrix a){
142     assert(a.isSquire());

```

```

143     double d=det(a);
144     assert(d);
145     return (1/d)*adj(a);
146 }
147 friend double det(Matrix a){
148     assert(a.isSquire());
149     double ans=0;
150     if(a.row()==1)ans=a[0][0];
151     else for(int i=0;i<a.column();i++){
152         ans+=pow(-1,i)*a[0][i]*det(cof(a,0,i));
153     }
154     return ans;
155 }
156 friend Matrix cof(Matrix a,int x,int y){
157     assert(a.isSquire());
158     std::vector<std::vector<double>>>c;
159     c.resize(a.row()-1);
160     int q=0,w=0;
161     for(int i=0;i<a.row()-1;i++){
162         c[i].resize(a.column()-1);
163         w=0;
164         if(q==x)q++;
165         for(int j=0;j<a.column()-1;j++){
166             if(w==y)w++;
167             c[i][j]=a[q][w];
168             w++;
169         }
170         q++;
171     }
172     Matrix ans(c);
173     return ans;
174 }
175 friend Matrix adj(Matrix a){
176     assert(a.isSquire());
177     std::vector<std::vector<double>>>c;
178     c.resize(a.row());
179     for(int i=0;i<a.row();i++){
180         c[i].resize(a.column());
181         for(int j=0;j<a.column();j++){
182             c[i][j]=pow(-1,i+j)*det(cof(a,i,j));
183         }
184     }
185     Matrix ans(c);
186     return T(ans);
187 }
188 };

```

## fraction

```
1  #include<algorithm>
2  class Frac:std::pair<int,int>{
3  public:
4      Frac(){first=0;second=1;}
5      Frac(int a,int b=1){
6          int g=std::__gcd(a,b);
7          if(second<0){first*=-1;second*=-1;}
8          first=a/g;second=b/g;
9      }
10     Frac operator=(Frac b){first=b.first;second=b.second;return *this;}
11     friend Frac operator+(Frac a,Frac b){
12         return Frac(a.first*b.second+b.first*a.second,a.second*b.second);
13     }
14     friend Frac operator-(Frac a,Frac b){return a+(-1*b);}
15     friend Frac operator*(Frac a,Frac b){ return Frac(a.first*b.first,a.second*b.second);}
16     friend Frac operator/(Frac a,Frac b){return a*inverse(b);}
17     friend void operator+=(Frac a,Frac b){a=a+b;return;}
18     friend void operator-=(Frac a,Frac b){a=a-b;return;}
19     friend void operator*=(Frac a,Frac b){a=a*b;return;}
20     friend void operator/=(Frac a,Frac b){a=a/b;return;}
21     friend bool operator==(Frac a,Frac b){return a.first==b.first && a.second==b.second;}
22     friend bool operator!=(Frac a,Frac b){return !(a==b);}
23     friend bool operator<(Frac a,Frac b){return a.first*b.second<b.first*a.second;}
24     friend bool operator>(Frac a,Frac b){return b<a;}
25     friend bool operator<=(Frac a,Frac b){return !(b<a);}
26     friend bool operator>=(Frac a,Frac b){return !(a<b);}
27     friend std::ostream & operator<<(std::ostream &out,const Frac &x){
28         out<<x.first;
29         if(x.second!=1)out<<"/"<<x.second;
30         return out;
31     }
32     friend std::istream & operator>>(std::istream &in,Frac &x){
33         int a,b;
34         in>>a>>b;
35         x=Frac(a,b);
36         return in;
37     }
38     friend Frac inverse(Frac a){return Frac(a.second,a.first);}
39 };
```

# JAVA

## BigInteger

```
1 import java.math.BigInteger;
2 BigInteger n1,n2,ans;
3 n1=new BigInteger(keyboard.next());
4 n2=new BigInteger(keyboard.next());
5 ans=n1.add(n2));
6 ans=n1.subtract(n2);
7 ans=n1.multiply(n2));
8 ans=n1.divide(n2));
9 ans=n1.mod(n2));
```

## default\_code

```
1 import java.util.Scanner;
2 public class Main{
3     public static void main(String[] args){
4         Scanner keyboard=new Scanner(System.in);
5         String str;
6         int num;
7         while(keyboard.hasNext()){
8             str=keyboard.next();
9             System.out.println(str);
10            num=keyboard.nextInt();
11            System.out.println(num);
12        }
13    }
14 }
```

## build\_and\_run

```
1 $java Main.java
2 $javac Main
```

# Algorithm

## dijkstra

```
1 #include<iostream>
2 using namespace std;
3 int map[1005][1005];
4 int dis[1005];
5 bool vis[1005];
6 void dijkstra(int start,int end,int n){
7     int pos;
```

```

8     for(int i=0;i<n;i++)dis[i]=map[i][start];
9     vis[start]=true;
10    for(int i=0;i<n-1;i++){
11        pos=s;
12        int min=inf;
13        for(int j=0;j<n;j++){
14            if(!vis[j] && dis[j]<min){
15                min=dis[j];
16                pos=j;
17            }
18        }
19        vis[pos]=true;
20        for(int j=0;j<n;j++){
21            if(!vis[j] && dis[j]>dis[pos]+map[j][pos]){
22                dis[j]=dis[pos]+map[j][pos];
23            }
24        }
25    }
26 }
27 int main(){
28     int n,m;
29     int a,b,x;
30     int start,end;
31     cin>>n>>m;
32     for(int i=0;i<n;i++){
33         for(int j=0;j<n;j++){
34             map[i][j]=1000000;
35         }
36         map[i][i]=0;
37     }
38     for(int i=0;i<m;i++){
39         cin>>a>>b>>x;
40         map[a][b]=map[b][a]=x;
41     }
42     cin>>start>>end;
43     dijkstra(start,end,n);
44     cout<<(dis[end]!=1000000)?dis[end]:-1<<endl;
45     return 0;
46 }

```

## 0-1Knapsack

```

1  #include<iostream>
2  using namespace std;
3  struct item{
4      int weight,value;
5  }items[1005];
6  int dp[100005];
7  int main(){
8      int n,m;
9      cin>>n>>m;

```

```

10     for(int i=0;i<n;i++)cin>>items[i].weight>>items[i].value;
11     for(int i=0;i<n;i++){
12         for(int j=m;j>=0;j--){
13             if(j-items[i].weight>=0){
14                 dp[j]=max(dp[j],dp[j-items[i].weight]+items[i].value);
15             }
16         }
17     }
18     cout<<dp[m]<<endl;
19     return 0;
20 }

```

# Math

## Newton\_Raphson\_Method

```

1 double Newton_Raphson_Method(func f,double x=1){
2     while(abs(f.setX(x))>0.000001)
3         x-=(f.setX(x)/prime(f).setX(x));
4     return x;
5 }

```

## chinese\_remainder

```

1 //need: gcdExtended
2 num chineseRemainder(num a[],num w[],int len){
3     num d,x,y,m,n=1,ret=0;
4     for(int i=0;i<len;i++)n*=w[i];
5     for(int i=0;i<len;i++){
6         m=n/w[i];
7         d=gcdExtended(w[i],m,&x,&y);
8         ret=(ret+y*m*a[i])%n;
9     }
10     return (n+ret%n)%n;
11 }

```

## fast\_power



```

1  inline num fastPower(num a,num b,num mod=0){
2      num ans=1;
3      while(b){
4          if(b&1)ans=fastMulti(ans,a,mod);
5          a=fastMulti(a,a,mod);
6          b>>=1;
7      }
8      return ans;
9  }

```

## gcd\_extended

```

1  num gcdExtended(num a,num b,num *x,num *y){
2      if(!a){*x=0,*y=1;return b;}
3      num x1,y1;
4      num gcd=gcdExtended(b%a,a,&x1,&y1);
5      *x=y1-(b/a)*x1;
6      *y=x1;
7      return gcd;
8  }

```

## pollard\_rho

```

1  //need: gcd、fastMulti、fastPower、millerRabin
2  inline num pollardRho(num n){
3      if(millerRabin(n)){cout<<n<<" ";return n;}
4      num p=n;
5      while(p==n)p=[](int n){
6          num c=rand()%(n-1)+1;
7          num k=2,x=1ll*rand()%n+1,y=x;
8          for(num i=2;;i++){
9              x=(fastMulti(x,x,n)+c)%n;
10             num d=gcd(x-y,n);
11             if(d!=1 && d!=n)return d;
12             if(y==x)return n;
13             if(i==k)k<<=1,y=x;
14         }
15     }(p);
16     return max(pollardRho(p),pollardRho(n/p));
17 }

```

## fast\_multi

```

1 inline num fastMulti(num a,num b,num mod=0){
2     num ans=0;
3     while(b){
4         if(b&1)ans=(mod)?(ans+a)%mod:ans+a;
5         a=(mod)?(a<<1)%mod:a<<1;
6         b>>=1;
7     }
8     return ans;
9 }

```

## gcd

```

1 inline num gcd(num a,num b){
2     return a<0?gcd(-a,b):(a?1:(!b?a:gcd(b,a%b)));
3 }// lcm=a*b/gcd(a,b)

```

## Miller\_Rabin

```

1 //need: fastMulti、fastPower
2 bool millerRabin(num n,int times=20){
3     if(n==2)return true;
4     if(n<2||n%2==0)return false;
5     for(int i=1;i<=times;){
6         num tem=n-1;
7         int j=0;
8         while(tem%2==0){tem/=2;j++;}
9         num a=((double)rand()/RAND_MAX*(n-2)+0.5)+1;
10        num x=fastPower(a,tem,n);
11        if(x!=1 && x!=n-1){
12            while(j--){
13                x=fastMulti(x,x,n);
14                if(x==n-1)goto next;
15            }
16            return false;
17        }
18        next:i++;
19    }
20    return true;
21 }

```

## mod\_inverse

```
1 //need: gcdExtended
2 num modInverse(num a,num m){
3     num x,y;
4     num g=gcdExtended(a,m,&x,&y);
5     if(g!=1)return 0;
6     else return (x%m+m)%m;
7 }
```

## SquareNumber

```
1 bool isSquareNumber(long long n){
2     if(n<1)return false;
3     for(long long i=1;n;i+=2)n-=i;
4     return !n;
5 }
```