# CodeBook by 櫛風

- 0-1Knapsack
- Math
  - Newton_Raphson_Method
  - chinese_remainder
  - fast_power
  - gcd_extended
  - pollard_rho
  - fast_multi
  - gcd
  - Miller_Rabin
  - mod_inverse
  - SquareNumber

# Python

## slice

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

## swap

```
a,b=(b,a)
```

## input

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

# Dark_Code

## rope

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

## thread

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

## default_code

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

#define debug(x) #x<<": "<<x<<" "
#define endl "\n"
#define num long long

using namespace std;

int main(){
    cin.tie(0);
    ios_base::sync_with_stdio(0);

    return 0;
}
```

## faster_input

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

## random_shuffle

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

# OOP

## function

```
#include<vector>
class func:std::vector<double>{
public:
    func(int n=0){resize(n);}
    func(std::vector<double>a){
        resize(a.size());
        for(int i=0;i<a.size();i++)at(i)=a[i];
    }
    void print(){
        bool first=true;
        for(int i=size()-1;i>=0;i--){
            if(at(i)){
                std::cout<<((first)?(first=false,""):"+")
                         <<at(i)<<((i>=1)?"x":"");
                if(i>1)std::cout<<"^"<<i;
            }
        }
        std::cout<<std::endl;
    }
    double setX(double x){
        double ans=0;
        double t=1;
        for(int i=0;i<size();i++){
            ans+=at(i)*t;
            t*=x;
        }
        return ans;
    }
    friend func prime(func a){
        func b;
        for(int i=1;i<a.size();i++){
            b.push_back(a[i]*i);
        }
        return b;
    }
```

```
};
```

## BigInteger

```
#include<algorithm>
#include<sstream>
#include<vector>
#include<cmath>
#include<iomanip>
class bigN:std::vector<long long>{
private:
    const static int base=1000000000,width=log10(base);
    bool negative;
    bigN convert_base(int old_width,int new_width)const{
        vector<long long>p(std::max(old_width,new_width)+1,1);
        for(size_t i=1;i<p.size();++i)p[i]=p[i-1]*10;
            bigN ans;
            long long cur=0;
            int cur_id=0;
            for(size_t i=0;i<size();++i){
                cur+=at(i)*p[cur_id];
                cur_id+=old_width;
                while(cur_id>=new_width){
                    ans.push_back(cur%p[new_width]);
                    cur/=p[new_width];
                    cur_id-=new_width;
                }
            }
            return ans.push_back(cur),ans.trim(),ans;
    }
    bigN karatsuba(const bigN &b)const{
        bigN res;
        res.resize(size()*2);
        if(size()<=32){
            for(size_t i=0;i<size();++i){
                for(size_t j=0;j<size();++j){
                    res[i+j]+=at(i)*b[j];
                }
            }
            return res;
        }
        size_t k=size()/2;
        bigN a1(begin(),begin()+k);
        bigN a2(begin()+k,end());
        bigN b1(b.begin(),b.begin()+k);
        bigN b2(b.begin()+k,b.end());
        bigN a1b1=a1.karatsuba(b1);
        bigN a2b2=a2.karatsuba(b2);
        for(size_t i=0;i<k;++i)a2[i]+=a1[i];
        for(size_t i=0;i<k;++i)b2[i]+=b1[i];
        bigN r=a2.karatsuba(b2);
```

```
        for(size_t i=0;i<a1b1.size();++i)r[i]-=a1b1[i];
        for(size_t i=0;i<a2b2.size();++i)r[i]-=a2b2[i];
        for(size_t i=0;i<r.size();++i)res[i+k]+=r[i];
        for(size_t i=0;i<a1b1.size();++i)res[i]+=a1b1[i];
        for(size_t i=0;i<a2b2.size();++i)res[i+size()]+=a2b2[i];
        return res;
    }
    void trim(){
        while(size()&&!back())pop_back();
        if(empty())negative=0;
    }
    void carry(int _base=base){
        for(size_t i=0;i<size();++i){
            if(at(i)>=0&&at(i)<_base)continue;
            if(i+1u==size())push_back(0);
            int r=at(i)%_base;
            if(r<0)r+=_base;
            at(i+1)+=(at(i)-r)/_base;
            at(i)=r;
        }
    }
    int abscmp(const bigN &b)const{
        if(size()>b.size())return 1;
        if(size()<b.size())return -1;
        for(int i=int(size())-1;i>=0;--i){
            if(at(i)>b[i])return 1;
            if(at(i)<b[i])return -1;
        }
        return 0;
    }
    int cmp(const bigN &b)const{
        if(negative!=b.negative)return negative?-1:1;
        return negative?-abscmp(b):abscmp(b);
    }
    bigN abs()const{
        bigN res=*this;
        return res.negative=0,res;
    }
public:
    bigN():negative(0){}
    bigN(const_iterator a,const_iterator b):vector<long long>(a,b){}
    bigN(std::string s){
        if(s.empty())return;
        if(s[0]=='-')negative=1,s=s.substr(1);
        else negative=0;
        for(int i=int(s.size())-1;i>=0;i-=width){
            long long t=0;
            for(int j=std::max(0,i-width+1);j<=i;++j)
                t=t*10+s[j]-'0';
            push_back(t);
        }
        trim();
```

```
    }
    template<typename T>bigN(const T &x){
        std::stringstream ss;
        ss<<x;
        *this=ss.str();
    }
    bool operator<(const bigN&b)const{return cmp(b)<0;}
    bool operator>(const bigN&b)const{return cmp(b)>0;}
    bool operator<=(const bigN&b)const{return cmp(b)<=0;}
    bool operator>=(const bigN&b)const{return cmp(b)>=0;}
    bool operator==(const bigN&b)const{return !cmp(b);}
    bool operator!=(const bigN&b)const{return cmp(b)!=0;}
    bigN operator-()const{bigN res=*this;return res.negative=!negative,res.trim(),res;}
    bigN operator+(const bigN &b)const{
        if(negative)return -(-(*this)+(-b));
        if(b.negative)return *this-(-b);
        bigN res=*this;
        if(b.size()>size())res.resize(b.size());
        for(size_t i=0;i<b.size();++i)res[i]+=b[i];
        return res.carry(),res.trim(),res;
    }
    bigN operator-(const bigN &b)const{
        if(negative)return -(-(*this)-(-b));
        if(b.negative)return *this+(-b);
        if(abscmp(b)<0)return -(b-(*this));
        bigN res=*this;
        if(b.size()>size())res.resize(b.size());
        for(size_t i=0;i<b.size();++i)res[i]-=b[i];
        return res.carry(),res.trim(),res;
    }
    bigN operator*(const bigN &b)const{
        const static int mul_base=1000000,mul_width=log10(mul_base);
        bigN A=convert_base(width,mul_width);
        bigN B=b.convert_base(width,mul_width);
        int n=std::max(A.size(),B.size());
        while(n&(n-1))++n;
        A.resize(n),B.resize(n);
        bigN res=A.karatsuba(B);
        res.negative=negative!=b.negative;
        res.carry(mul_base);
        res=res.convert_base(mul_width,width);
        return res.trim(),res;
    }
    bigN operator*(long long b)const{
        bigN res=*this;
        long long a;
        if(b<0)res.negative=!negative,b=-b;
        for(size_t i=0,is=0;i<res.size()||is;++i){
            if(i==res.size())res.push_back(0);
            a=res[i]*b+is;
            is=a/base;
            res[i]=a%base;
```

```
        }
        return res.trim(),res;
    }
    bigN operator/(const bigN &b)const{
        int norm=base/(b.back()+1);
        bigN x=abs()*norm;
        bigN y=b.abs()*norm;
        bigN q,r;
        q.resize(x.size());
        for(int i=int(x.size())-1;i>=0;--i){
            r=r*base+x[i];
            int s1=r.size()<=y.size()?0:r[y.size()];
            int s2=r.size()<y.size()?0:r[y.size()-1];
            int d=((long long)(base)*s1+s2)/y.back();
            r=r-y*d;
            while(r.negative)r=r+y,--d;
            q[i]=d;
        }
        q.negative=negative!=b.negative;
        return q.trim(),q;
    }
    bigN operator%(const bigN &b)const{return *this-(*this/b)*b;}
    bigN operator<<(const int &b)const{
        bigN res=*this;
        for(int i=0;i<b;i++)res*=2;
        return res.carry(),res.trim(),res;
    }
    bigN operator>>(const int &b)const{
        bigN res=*this;
        for(int i=0;i<b;i++)res/=2;
        return res.carry(),res.trim(),res;
    }
    friend std::istream& operator>>(std::istream &ss,bigN &b){
        std::string s;
        return ss>>s,b=s,ss;
    }
    friend std::ostream& operator<<(std::ostream &ss,const bigN &b){
        if(b.negative)ss<<'-';
        ss<<(b.empty()?0:b.back());
        for(int i=int(b.size())-2;i>=0;--i)
        ss<<std::setw(width)<<std::setfill('0')<<b[i];
        return ss;
    }
    template<typename T>operator T(){
        std::stringstream ss;
        ss<<*this;
        T res;
        return ss>>res,res;
    }
    friend bigN abs(bigN a){return a.abs();}
    bigN operator+=(const bigN &other){*this=(*this)+(other);return *this;}
    bigN operator-=(const bigN &other){*this=(*this)-(other);return *this;}
```

```
    bigN operator*=(const bigN &other){*this=(*this)*(other);return *this;}
    bigN operator/=(const bigN &other){*this=this->operator/(other);return *this;}
    bigN operator<<=(const bigN &other){*this=(*this)<<(other);return *this;}
    bigN operator>>=(const bigN &other){*this=(*this)>>(other);return *this;}
    bigN operator++(){(*this)+=1;return *this;}
    bigN operator++(int){*this+=1;return *this;}
    bigN operator--(){*this-=1;return *this;}
    bigN operator--(int){*this-=1;return *this;}
    bool operator!(){return (*this)==0;}
};
```

## Matrix

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <cassert>

class Matrix:std::vector<std::vector<double>>{
private:
    int maxL;
public:
    Matrix(int n,int m,double a=0){
        resize(n);
        maxL=1;
        for(int i=0;i<size();i++){
            at(i).resize(m);
            if(n<=m)at(i).at(i)=a;
        }
    }
    Matrix(std::vector<std::vector<double>>a){
        int t;
        resize(a.size());
        for(int i=0;i<size();i++){
            at(i).resize(a[i].size());
            for(int j=0;j<a[i].size();j++){
                at(i).at(j)=a[i][j];
                t=log10(abs(at(i).at(j)))+1;
                if(at(i).at(j)<0)t++;
                if(t>maxL)maxL=t;
            }
        }
    }
    int row()const{return size();}
    int column()const{return at(0).size();}
    bool isSqure(){return row()==column();}
    void operator+=(Matrix other){*this=*this+other;}
    void operator-=(Matrix other){*this=*this-other;}
    void operator*=(Matrix other){*this=*this*other;}
    void operator^=(int times){*this=*this^times;}
```

```
    void print(){
        for(int i=0;i<row();i++){
            if(i==0 && i==row()-1)std::cout<<"[";
            else if(i==0)std::cout<<"⎡";
            else if(i==row()-1)std::cout<<"⎣";
            else std::cout<<"⎢";
            for(int j=0;j<column();j++){
                std::cout<<std::setw(maxL)<<at(i).at(j);
                if(j==column()-1){
                    if(i==0 && i==row()-1)std::cout<<"]";
                    else if(i==0)std::cout<<"⎤";
                    else if(i==row()-1)std::cout<<"⎦";
                    else std::cout<<"⎥";
                }
                else std::cout<<" ";
            }
            std::cout<<std::endl;
        }
    }
    friend std::ostream & operator<<(std::ostream &out,const Matrix &a){
        for(int i=0;i<a.row();i++){
            for(int j=0;j<a.column();j++){
                out<<std::setw(a.maxL)<<a[i][j]<<" \n"[j==a.column()-1];
            }
        }
        return out;
    }
    friend std::istream & operator>>(std::istream &in,Matrix &a){
        int t;
        for(int i=0;i<a.row();i++){
            for(int j=0;j<a.column();j++){
                in>>a[i][j];
                t=log10(abs(a[i][j]))+1;
                if(a[i][j]<0)t++;
                if(t>a.maxL)a.maxL=t;
            }
        }
        return in;
    }
    friend int operator==(Matrix a,Matrix b){
        if(a.row()!=b.row() || a.column()!=b.column())return 0;
        for(int i=0;i<a.row();i++){
            for(int j=0;j<a.column();j++){
                if(a[i][j]!=b[i][j])return -1;
            }
        }
        return 1;
    }
    friend bool operator!=(Matrix a,Matrix b){return !(a==b);}
    friend Matrix operator+(Matrix a,Matrix b){
        assert(a==b);
        std::vector<std::vector<double>>c;
```

```
        c.resize(a.row());
        for(int i=0;i<a.row();i++){
            c[i].resize(a.column());
            for(int j=0;j<a.column();j++){
                c[i][j]=a[i][j]+b[i][j];
            }
        }
        Matrix ans(c);
        return ans;
    }
    friend Matrix operator-(Matrix a,Matrix b){return a+(-1*b);}
    friend Matrix operator*(Matrix a,double t){
        Matrix b(a.row(),a.column(),t);
        return a*b;
    }
    friend Matrix operator*(double t,Matrix a){return a*t;}
    friend Matrix operator*(Matrix a,Matrix b){
        assert(a==b);
        std::vector<std::vector<double>>c;
        c.resize(a.row());
        for(int i=0;i<a.row();i++){
            c[i].resize(a.column());
            for(int j=0;j<a.column();j++){
                for(int z=0;z<a.column();z++){
                    c[i][j]+=a[i][z]*b[z][j];
                }
            }
        }
        Matrix ans(c);
        return ans;
    }
    friend Matrix operator^(Matrix a,int t){
        if(t==-1)return inverse(a);
        assert(t>0);
        Matrix b=a;
        while(--t)b=b*a;
        return b;
    }
    friend Matrix T(Matrix a){
        std::vector<std::vector<double>> c;
        c.resize(a.column());
        for(int i=0;i<a.column();i++){
            c[i].resize(a.row());
            for(int j=0;j<a.row();j++){
                c[i][j]=a[j][i];
            }
        }
        Matrix ans(c);
        return ans;
    }
    friend Matrix inverse(Matrix a){
        assert(a.isSqure());
```

```
        double d=det(a);
        assert(d);
        return (1/d)*adj(a);
    }
    friend double det(Matrix a){
        assert(a.isSqure());
        double ans=0;
        if(a.row()==1)ans=a[0][0];
        else for(int i=0;i<a.column();i++){
            ans+=pow(-1,i)*a[0][i]*det(cof(a,0,i));
        }
        return ans;
    }
    friend Matrix cof(Matrix a,int x,int y){
        assert(a.isSqure());
        std::vector<std::vector<double>>c;
        c.resize(a.row()-1);
        int q=0,w=0;
        for(int i=0;i<a.row()-1;i++){
            c[i].resize(a.column()-1);
            w=0;
            if(q==x)q++;
            for(int j=0;j<a.column()-1;j++){
                if(w==y)w++;
                c[i][j]=a[q][w];
                w++;
            }
            q++;
        }
        Matrix ans(c);
        return ans;
    }
    friend Matrix adj(Matrix a){
        assert(a.isSqure());
        std::vector<std::vector<double>>c;
        c.resize(a.row());
        for(int i=0;i<a.row();i++){
            c[i].resize(a.column());
            for(int j=0;j<a.column();j++){
                c[i][j]=pow(-1,i+j)*det(cof(a,i,j));
            }
        }
        Matrix ans(c);
        return T(ans);
    }
};
```

## fraction

```cpp
#include<algorithm>
class Frac:std::pair<int,int>{
public:
    Frac(){first=0;second=1;}
    Frac(int a,int b=1){
        int g=std::__gcd(a,b);
        if(second<0){first*=-1;second*=-1;}
        first=a/g;second=b/g;
    }
    Frac operator=(Frac b){first=b.first;second=b.second;return *this;}
    friend Frac operator+(Frac a,Frac b){
        return Frac(a.first*b.second+b.first*a.second,a.second*b.second);
    }
    friend Frac operator-(Frac a,Frac b){return a+(-1*b);}
    friend Frac operator*(Frac a,Frac b){ return Frac(a.first*b.first,a.second*b.second);}
    friend Frac operator/(Frac a,Frac b){return a*inverse(b);}
    friend void operator+=(Frac a,Frac b){a=a+b;return;}
    friend void operator-=(Frac a,Frac b){a=a-b;return;}
    friend void operator*=(Frac a,Frac b){a=a*b;return;}
    friend void operator/=(Frac a,Frac b){a=a/b;return;}
    friend bool operator==(Frac a,Frac b){return a.first==b.first && a.second==b.second;}
    friend bool operator!=(Frac a,Frac b){return !(a==b);}
    friend bool operator<(Frac a,Frac b){return a.first*b.second<b.first*a.second;}
    friend bool operator>(Frac a,Frac b){return b<a;}
    friend bool operator<=(Frac a,Frac b){return !(b<a);}
    friend bool operator>=(Frac a,Frac b){return !(a<b);}
    friend std::ostream & operator<<(std::ostream &out,const Frac &x){
        out<<x.first;
        if(x.second!=1)out<<"/"<<x.second;
        return out;
    }
    friend std::istream & operator>>(std::istream &in,Frac &x){
        int a,b;
        in>>a>>b;
        x=Frac(a,b);
        return in;
    }
    friend Frac inverse(Frac a){return Frac(a.second,a.first);}
};
```

# JAVA

### BigInteger

```
import java.math.BigInteger;
BigInteger n1,n2,ans;
n1=new BigInteger(keyboard.next());
n2=new BigInteger(keyboard.next());
ans=n1.add(n2));
ans=n1.subtract(n2);
ans=n1.multiply(n2));
ans=n1.divide(n2));
ans=n1.mod(n2));
```

### default_code

```
import java.util.Scanner;
public class Main{
    public static void main(String[] args){
        Scanner keyboard=new Scanner(System.in);
        String str;
        int num;
        while(keyboard.hasNext()){
            str=keyboard.next();
            System.out.println(str);
            num=keyboard.nextInt();
            System.out.println(num);
        }
    }
}
```

### build_and_run

```
$java Main.java
$javac Main
```

# Algorithm

### dijkstra

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

```
    for(int i=0;i<n;i++)dis[i]=map[i][start];
    vis[start]=true;
    for(int i=0;i<n-1;i++){
        pos=s;
        int min=inf;
        for(int j=0;j<n;j++){
            if(!vis[j] && dis[j]<min){
                min=dis[j];
                pos=j;
            }
        }
        vis[pos]=true;
        for(int j=0;j<n;j++){
            if(!vis[j] && dis[j]>dis[pos]+map[j][pos]){
                dis[j]=dis[pos]+map[j][pos];
            }
        }
    }
}
int main(){
    int n,m;
    int a,b,x;
    int start,end;
    cin>>n>>m;
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            map[i][j]=1000000;
        }
        map[i][i]=0;
    }
    for(int i=0;i<m;i++){
        cin>>a>>b>>x;
        map[a][b]=map[b][a]=x;
    }
    cin>>start>>end;
    dijkstra(start,end,n);
    cout<<(dis[end]!=1000000)?dis[end]:-1<<endl;
    return 0;
}
```

## 0-1Knapsack

```
#include<iostream>
using namespace std;
struct item{
    int weight,value;
}items[1005];
int dp[100005];
int main(){
    int n,m;
    cin>>n>>m;
```

```
    for(int i=0;i<n;i++)cin>>items[i].weight>>items[i].value;
        for(int i=0;i<n;i++){
            for(int j=m;j>=0;j--){
            if(j-items[i].weight>=0){
                dp[j]=max(dp[j],dp[j-items[i].weight]+items[i].value);
            }
        }
    }
    cout<<dp[m]<<endl;
    return 0;
}
```

# Math

## Newton_Raphson_Method

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

## chinese_remainder

```
//need: gcdExtended
num chineseRemainder(num a[],num w[],int len){
    num d,x,y,m,n=1,ret=0;
        for(int i=0;i<len;i++)n*=w[i];
        for(int i=0;i<len;i++){
            m=n/w[i];
            d=gcdExtended(w[i],m,&x,&y);
            ret=(ret+y*m*a[i])%n;
        }
        return (n+ret%n)%n;
}
```

## fast_power

```
inline num fastPower(num a,num b,num mod=0){
    num ans=1;
    while(b){
        if(b&1)ans=fastMulti(ans,a,mod);
        a=fastMulti(a,a,mod);
        b>>=1;
    }
    return ans;
}
```

## gcd_extended

```
num gcdExtended(num a,num b,num *x,num *y){
    if(!a){*x=0,*y=1;return b;}
    num x1,y1;
    num gcd=gcdExtended(b%a,a,&x1,&y1);
    *x=y1-(b/a)*x1;
    *y=x1;
    return gcd;
}
```

## pollard_rho

```
//need: gcd、fastMulti、fastPower、millerRabin
inline num pollardRho(num n){
    if(millerRabin(n)){cout<<n<<" ";return n;}
    num p=n;
    while(p==n)p=[](int n){
        num c=rand()%(n-1)+1;
        num k=2,x=1ll*rand()%n+1,y=x;
        for(num i=2;;i++){
            x=(fastMulti(x,x,n)+c)%n;
            num d=gcd(x-y,n);
            if(d!=1 && d!=n)return d;
            if(y==x)return n;
            if(i==k)k<<=1,y=x;
        }
    }(p);
    return max(pollardRho(p),pollardRho(n/p));
}
```

## fast_multi

```
inline num fastMulti(num a,num b,num mod=0){
    num ans=0;
    while(b){
        if(b&1)ans=(mod)?(ans+a)%mod:ans+a;
        a=(mod)?(a<<1)%mod:a<<1;
        b>>=1;
    }
    return ans;
}
```

## gcd

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

## Miller_Rabin

```
//need: fastMulti、fastPower
bool millerRabin(num n,int times=20){
    if(n==2)return true;
    if(n<2||n%2==0)return false;
    for(int i=1;i<=times;){
        num tem=n-1;
        int j=0;
        while(tem%2==0){tem/=2;j++;}
        num a=((double)rand()/RAND_MAX*(n-2)+0.5)+1;
        num x=fastPower(a,tem,n);
        if(x!=1 && x!=n-1){
            while(j--){
                x=fastMulti(x,x,n);
                if(x==n-1)goto next;
            }
            return false;
        }
        next:i++;
    }
    return true;
}
```

## mod_inverse

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

## SquareNumber

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