Template

jslijin_3

May 17, 2019

目录

1 图论

1.1 spfa_双端优化最长路_判正环

```
const int _N=1e6+5,_M=1e6+5;
template <class V>
struct Tu{
   int head[_N],nxt[_M],e[_M],n,tot; V v[_N],w[_M];
   void Init(int _n) { n=_n,mem(head,0),tot=0; }
   void I(int x,int y,V _w) { e[++tot]=y,w[tot]=_w; nxt[tot]=head[x],head[x]=tot; }
};

struct SPFA{
   int x,cnt[_N]; bool v[_N]; template <class V>
   bool solve(const Tu<V> &T,int s,V dist[],V _oo=oo) {
      deque <int> Q; rep(i,0,T.n+1) dist[i]=_oo;
      mem(v,0),mem(cnt,0),dist[s]=0,cnt[s]=v[s]=1,Q.push_back(s);
      while (!Q.empty()) {
         x=Q.front(),Q.pop_front(),v[x]=0;
         for (int i=T.head[x]; i; i=T.nxt[i]) {
            if ((ll)dist[x]+T.w[i]<=dist[T.e[i]]) continue;
            dist[T.e[i]]=dist[x]+T.w[i];
            if (v[T.e[i]]) continue; v[T.e[i]]=1,cnt[T.e[i]]++;
            if (cnt[T.e[i]]>=T.n+1) return 0;
            if (!Q.empty() && dist[T.e[i]]<dist[Q.front()])
               Q.push_front(T.e[i]); else Q.push_back(T.e[i]);
         }
      }
      return 1;
   }
};
```

1.2 spfa_朴素最长路_判正环

```
const int _N=1e6+5,_M=1e6+5;
template <class V>
struct Tu{
   int head[_N],nxt[_M],e[_M],n,tot; V v[_N],w[_M];
   void Init(int _n) { n=_n,mem(head,0),tot=0; }
   void I(int x,int y,V _w) { e[++tot]=y,w[tot]=_w; nxt[tot]=head[x],head[x]=tot; }
};

struct SPFA{
   int x,cnt[_N]; bool v[_N]; template <class V>
   bool solve(const Tu<V> &T,int s,V dist[],V _oo=oo) {
      queue <int> Q; rep(i,0,T.n+1) dist[i]=_oo;
      mem(v,0),mem(cnt,0),dist[s]=0,cnt[s]=v[s]=1,Q.push(s);
      while (!Q.empty()) {
         x=Q.front(),Q.pop(),v[x]=0;
         for (int i=T.head[x]; i; i=T.nxt[i])
            if ((ll)dist[x]+T.w[i]>dist[T.e[i]]) {
               dist[T.e[i]]=dist[x]+T.w[i];
               if (!v[T.e[i]]) {
                  if (cnt[T.e[i]]>=T.n) return 0;
                  Q.push(T.e[i]),v[T.e[i]]=1,cnt[T.e[i]]++;
               }
            }
      }
      return 1;
   }
};
```

2 数学

2.1 FT_fft_2D

```
const int _M=2050,_N=N;
template <class V>
struct FT{
   struct cp{ double x,y; } tmp[_M*2+5]; cp aa[_M][_M],bb[_M][_M];
   friend cp operator + (cp &a,cp &b) { return cp{a.x+b.x,a.y+b.y}; }
   friend cp operator - (cp &a,cp &b) { return cp{a.x-b.x,a.y-b.y}; }
   friend cp operator * (cp &a,cp &b) { return cp{a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x}; }
   cp get(double x) { return cp{cos(x),sin(x)}; }
   void FFT(cp *a,int n,int op){
      for(int i=(n>>1),j=1;j<n;j++){
         if(i<j) swap(a[i],a[j]);
         int k; for(k=(n>>1);k&i;i^=k,k>>=1); i^=k;
      }
      for(int m=2;m<=n;m<<=1){
         cp w=get(2*PI*op/m); tmp[0]=cp{1,0};
         for(int j=1;j<(m>>1);j++) tmp[j]=tmp[j-1]*w;
         for(int i=0;i<n;i+=m)
            for(int j=i;j<i+(m>>1);j++){
               cp u=a[j],v=a[j+(m>>1)]*tmp[j-i];
               a[j]=u+v,a[j+(m>>1)]=u-v;
            }
      }
      if(op==-1) rep(i,0,n) a[i]=cp{a[i].x/n,a[i].y/n};
   }
   void FFT(cp a[][_M],int n,int op) { rep(i,0,n) FFT(a[i],n,op); }
   template <class T>
   void Transpose(T a[][_M],int n) {
       rep(i,0,n) rep(j,0,i) swap(a[i][j],a[j][i]);
    }
    void Reverse(V a[][_M],int n,int m) {
        rep(i,0,(n-1>>1)+1) rep(j,0,m) swap(a[i][j],a[n-1-i][j]);
        rep(i,0,n) rep(j,0,(m-1>>1)+1) swap(a[i][j],a[i][m-1-j]);
    }
    void Shift(V a[][_M],int n,int m,int p,int q) {
        rep(i,n,n+p) rep(j,m,m+q) a[i-n][j-m]=a[i][j];
    }
    void In(cp p[][_M],int len,V a[][_M],int n,int m) {
        rep(i,0,len) rep(j,0,len) p[i][j]=cp{i<n&&j<m?(double)a[i][j]:0,0};
    }
    void Out(V a[][_M],int n,int m,cp p[][_M],int len) {
        rep(i,0,n) rep(j,0,m) a[i][j]=(V)(p[i][j].x+0.5)%_p;
    }
```

```
   void Multiply(V A[][_M],int n,V B[][_M],int m,V C[][_M],int &len,int op=0) {
       if (op) Reverse(A,n,n);
      len=1; while (len<n+m-1) len<<=1;
      In(aa,len,A,n,n),In(bb,len,B,m,m),FFT(aa,len,1),FFT(bb,len,1);
        Transpose(aa,len),Transpose(bb,len),FFT(aa,len,1),FFT(bb,len,1);
      rep(i,0,len) rep(j,0,len) aa[i][j]=aa[i][j]*bb[i][j];
      FFT(aa,len,-1),Transpose(aa,len),FFT(aa,len,-1),Out(C,len,len,aa,len);
      if (op) Shift(C,n-1,n-1,m,m),len=m,Reverse(A,n,n);
   }
};

inline void Random(int a[][_M],int n) {
    rep(i,0,n) rep(j,0,n) a[i][j]=rand();
}
```

2.2 FT__fft__2D__使用说明

```
/*
* FT_fft_2D 使用说明
*
* 【接口说明】
*
* cp get(double x) ： 获取一个辐角为 x 的复数
*
* void FFT(cp *a,int n,int op) ： 变换接口，注意： op=1 为正卷积， op=-1 为逆卷积
*
* void FFT(cp a[ ][_M],int n,int op) ： 行变换接口，逐行进行正 / 逆变换
*
* void Transpose(T a[ ][_M],int n) ： 转置接口
*
* void Reverse(V a[ ][_M],int n,int m) ： 翻转接口
*
* void Shift(V a[ ][_M],int n,int m,int p,int q) ： 移位接口，将矩阵 a 的 (n,m) 整体移位
到 (0,0) 长度保留 p 和 q ( 长和宽 )
*
* void In(cp p[][_M],int len,V a[][_M],int n,int m) ： 数据填充接口
*
* void Out(V a[][_M],int n,int m,cp p[][_M],int len) ： 数据提取接口
*
* void Multiply(V A[][_M],int n,V B[][_M],int m,V C[][_M],int &len,int op=0) ：
*
* 乘法接口，表示 n * n 的矩阵 A 乘上 m*m 的矩阵 B ，结果放到 C 中，规模为 len * len 且计算并
返回到 len 中， op=1 为差卷积
*/
```

2.3 FT__fft 基础版本

```
const int _M=N,_N=N;
template <class V>
struct FT{
   struct cp{ double x,y; } tmp[_M*2+5];
   friend cp operator + (cp &a,cp &b) { return cp{a.x+b.x,a.y+b.y}; }
   friend cp operator - (cp &a,cp &b) { return cp{a.x-b.x,a.y-b.y}; }
   friend cp operator * (cp &a,cp &b) { return cp{a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x}; }
   cp get(double x) { return cp{cos(x),sin(x)}; }
   vector <cp> aa,bb;
   void FFT(vector<cp> &a,int n,int op){
      for(int i=(n>>1),j=1;j<n;j++){
         if(i<j) swap(a[i],a[j]);
         int k; for(k=(n>>1);k&i;i^=k,k>>=1); i^=k;
      }
      for(int m=2;m<=n;m<<=1){
         cp w=get(2*PI*op/m); tmp[0]=cp{1,0};
         for(int j=1;j<(m>>1);j++) tmp[j]=tmp[j-1]*w;
         for(int i=0;i<n;i+=m)
            for(int j=i;j<i+(m>>1);j++){
               cp u=a[j],v=a[j+(m>>1)]*tmp[j-i];
               a[j]=u+v,a[j+(m>>1)]=u-v;
            }
      }
      if(op==-1) rep(i,0,n) a[i]=cp{a[i].x/n,a[i].y/n};
   }
   vector<V> multiply(vector<V> A,vector<V> B,int op=0){
       if (op) reverse(all(A));
      int lena=A.size(),lenb=B.size(),len=1;
      while (len<lena+lenb) len<<=1;
      aa=vector<cp>(len),bb=vector<cp>(len);
      rep(i,0,lena) aa[i]=cp{(double)A[i],0};
      rep(i,0,lenb) bb[i]=cp{(double)B[i],0};
      FFT(aa,len,1),FFT(bb,len,1);
      rep(i,0,len) aa[i]=aa[i]*bb[i];
      FFT(aa,len,-1); A.clear();
        if (!op) rep(i,0,len) A.pb((ll)(aa[i].x+0.5)); else
        rep(i,lena-1,lena+lenb-2+1) A.pb((ll)(aa[i].x+0.5));
      return A;
   }
};
```

2.4 FWT__k 进制__xor 版本

```
const int _p=998244353;
ll add(ll x,ll y) { x+=y; return x%_p; }
ll mul(ll x,ll y) { return x*y%_p; }
ll Pow(ll x,ll k) {
   ll ret=1;
   for (;k;k>>=1,x=mul(x,x)) if (k&1) ret=mul(ret,x);
   return ret;
}

template <int K>
struct Num{
   ll a[K];
   Num(int x=0) { mem(a,0),a[0]=x; }
   inline Num& operator = (const Num &t) {
      rep(i,0,K) a[i]=t.a[i];
      return *this;
   }
   inline Num& operator = (int x) { mem(a,0),a[0]=x; return *this; }
   inline friend Num operator + (const Num &a,const Num &b) {
```

```
    Num c;
    rep(i,0,K) c.a[i]=add(a.a[i],b.a[i]);
    return c;
  }
  inline friend Num operator - (const Num &a,const Num &b) {
    Num c;
    rep(i,0,K) c.a[i]=add(a.a[i],-b.a[i]);
    return c;
  }
  inline friend Num operator * (const Num &a,const Num &b) {
    Num c;
    rep(i,0,K) rep(j,0,K) (c.a[(i+j)%K]+=mul(a.a[i],b.a[j]))%=_p;
    return c;
  }
  inline friend Num operator >> (const Num &a,int k) {
    Num c;
    rep(i,0,K) c.a[(i+k)%K]=a.a[i];
    return c;
  }
  inline friend Num operator ^ (Num x,ll k) {
     Num ret=1;
     for (;k;k>>=1,x=x*x) if (k&1) ret=ret*x;
     return ret;
  }
  inline ll Value() {
    int cnt=K&-K,L=K/cnt; ll ret=add(a[0],-(L>1)*a[cnt]);
    if (K&1^1) ret-=add(a[K>>1],-(L>1)*a[(K>>1)+cnt]),ret%=_p;
    return ret;
  }
  inline void print(string s="") {
    printf("\n\n\n%s\n",s.c_str());
    rep(i,0,K) printf("a[%d] => %d\n",i,a[i]);
  }
};

template <int M,int N,int K>
struct FT{
  Num<K> tmp[M<<1],a[N],b[N]; int t;
  void FWT(Num<K> a[],int S,int n,int op) {
    if (n==1) return; int L=n/M;
    rep(i,0,M) FWT(a,S+L*i,n/M,op);
    rep(i,0,L) {
      rep(j,0,M) tmp[j]=0;
      rep(j,0,M) rep(k,0,M) {
        t=op*j*k%M,t<0?t+=M:0;
        tmp[j]=tmp[j]+(a[S+L*k+i]>>t);
      }
      rep(j,0,M) a[S+L*j+i]=tmp[j];
    }
  }
  void Multiply(ll A[],ll B[],int n,ll C[]) {
    rep(i,0,n) a[i]=A[i],b[i]=B[i];
    FWT(a,0,n,1),FWT(b,0,n,1);
    rep(i,0,n) a[i]=a[i]*b[i];
    FWT(a,0,n,-1); ll inv=Pow(n,_p-2);
    rep(i,0,n) C[i]=mul(a[i].Value(),inv),C[i]<0?C[i]+=_p:0;
  }
  int get(int x,int y) {
    int ret=0,B=1;
    for (; x||y; x/=M,y/=M,B*=M) ret+=(x%M+y%M)%M*B;
    return ret;
  }
  void Multiply_B(ll A[],ll B[],int n,ll C[]) {
    rep(i,0,n) rep(j,0,n) (C[get(i,j)]+=mul(A[i],B[j]))%=_p;
    rep(i,0,n) C[i]<0?C[i]+=_p:0;
  }
};
```

2.5 NTT_2D

```
typedef vector<vector<int>> vii;

const int _M=N,_N=N;
template <class V>
struct FT{
  vector <V> aa,bb; int _p,K,_m,N; V w[2][_M*2+5],rev[_M*2+5],tmp,w0;
  inline void Init(int _K,int p) { K=_K,_p=p; }
  ll Pow(ll x,ll k,ll _p) { ll ans=1; for (;k;k>>=1,x=x*x%_p) if (k&1) (ans*=x)%=_p;
  return ans; }
  inline void get_len(int a,int b,int &len,int &L) { len=1,L=0; while (len<a+b) len
  <<=1,++L; }
  inline void init_w(int m) {
    N=1<<m,w0=Pow(3,(_p-1)/N,_p); w[0][0]=w[1][0]=1;
    rep(i,1,N) w[0][i]=w[1][N-i]=(ll)w[0][i-1]*w0%_p;
    rep(i,1,N) rev[i]=(rev[i>>1]>>1)|(i&1)<<m-1;
  }
  inline void FFT(vector<V>& A,int m,int op){
    if (m!=_m) init_w(_m=m);
    rep(i,0,N) if (i<rev[i]) swap(A[i],A[rev[i]]);
    for (int i=1; i<N; i<<=1)
      for (int j=0,t=N/(i<<1); j<N; j+=i<<1)
        for (int k=j,l=0; k<j+i; k++,l+=t) {
          V x=A[k],y=(ll)w[op][l]*A[k+i]%_p;
          A[k]=(x+y)%_p,A[k+i]=(x-y+_p)%_p;
        }
    if (op) { tmp=Pow(N,_p-2,_p); rep(i,0,N) A[i]=1ll*A[i]*tmp%_p; }
  }
  inline void multiply(const vector<V>& A,const vector<V>& B,vector<V> *C){
    int lena=A.size(),lenb=B.size(),len=1,L=0; aa=A,bb=B;
    get_len(lena,lenb,len,L),aa.resize(len),bb.resize(len);
    FFT(aa,L,0),FFT(bb,L,0),(*C).resize(len);
    rep(i,0,len) (*C)[i]=(ll)aa[i]*bb[i]%_p;
    FFT(*C,L,1); if (K<len-1) (*C).resize(K+1);
  }
};

struct Matrix{
  int n,m; vii a;
  inline void Set_m(int _m,int x=0) { m=_m; rep(i,0,n) a[i].resize(m,x); }
  inline void Set_n(int _n) { n=_n,a.resize(n); }
```

```
    inline void Set(int _n,int _m,int x=0) { Set_n(_n),Set_m(_m,x); }
    Matrix(int n=0,int m=0,int x=0):n(n),m(m) {
        a.clear(),a.resize(n); Set_m(m,x);
    }
    inline void Transpose() {
        Matrix t=Matrix(m,n,0);
        rep(i,0,n) rep(j,0,m) t.a[j][i]=a[i][j];
        *this=t;
    }
    inline void Reverse() {
        Matrix t=Matrix(n,m,0);
        rep(i,0,n) rep(j,0,m) t.a[n-1-i][m-1-j]=a[i][j];
        *this=t;
    }
    inline void Shift(int x,int y) {
        rep(i,x,n-1+x+1) rep(j,y,m-1+y+1) a[i-x][j-y]=(i<n&&j<m)?a[i][j]:0;
    }
    inline void FFT(FT<int> &T,int len,int op) {
        if (!op) Set_m(1<<len,0);
        rep(i,0,n) T.FFT(a[i],len,op);
    }
    inline void print() const;
    inline void Normalize(int _p);
    inline void Random();
};

inline void Matrix::print() const {
    printf("\n\n\n\n\nn => %d     m => %d\n",n,m);
    debug_arr2(a,n-1,m-1);
}

inline void Matrix::Normalize(int _p) {
    rep(i,0,n) rep(j,0,m) if (a[i][j]<0) a[i][j]+=_p;
}

inline void Matrix::Random() {
    rep(i,0,n) rep(j,0,m) a[i][j]=(rand()<<15)+rand();
}

inline bool operator==(const Matrix &A,const Matrix &B) {
    if (A.n!=B.n || A.m!=B.m) return 0;
    rep(i,0,A.n) rep(j,0,A.m) if (A.a[i][j]!=B.a[i][j]) return 0;
    return 1;
}

struct Calculator{
    Matrix aa,bb,cc; FT<int> T; int len,L,_p;
    inline void Init(int p) { _p=p; }
    inline void Multiply(const Matrix &A,const Matrix &B,Matrix &C,int op=0) {
        aa=A,bb=B; if (op) aa.Reverse(); T.get_len(A.m,B.m,len,L),T.Init(cc.m=A.n+B.n-1,
        _p);
        aa.FFT(T,L,0),bb.FFT(T,L,0),aa.Transpose(),bb.Transpose(),cc.Set_n(aa.n);
        rep(i,0,aa.n) T.multiply(aa.a[i],bb.a[i],&cc.a[i]);
        cc.Transpose(),cc.FFT(T,L,1),cc.Set_m(A.m+B.m-1),C=cc;
        if (op) C.Shift(A.n-1,A.m-1);
    }
    inline void add(int &x,int y) { x+=y,x%=_p; }
    inline int mul(int x,int y) { return (ll)x*y%_p; }
    inline void Multiply_B(const Matrix &A,const Matrix &B,Matrix &C) {
        C.Set(A.n+B.n-1,A.m+B.m-1);
        rep(xa,0,A.n) rep(ya,0,A.m) rep(xb,0,B.n) rep(yb,0,B.m)
            add(C.a[xa+xb][ya+yb],mul(A.a[xa][ya],B.a[xb][yb]));
    }
    inline void Multiply_B_sub(const Matrix &A,const Matrix &B,Matrix &C) {
        C.Set(A.n+B.n-1,A.m+B.m-1);
        rep(xa,0,A.n) rep(ya,0,A.m) rep(xb,xa,B.n) rep(yb,ya,B.m)
            add(C.a[xb-xa][yb-ya],mul(A.a[xa][ya],B.a[xb][yb]));
    }
};
```

2.6 一维 FFT_单模式串_模式串带通配符匹配

```
struct cp{ double x,y; };
inline cp operator + (cp &a,cp &b) { return cp{a.x+b.x,a.y+b.y}; }
inline cp operator - (cp &a,cp &b) { return cp{a.x-b.x,a.y-b.y}; }
inline cp operator * (cp &a,cp &b) { return cp{a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x}; }
inline cp get(double x) { return cp{cos(x),sin(x)}; }
inline ostream& operator<<(ostream &out,const cp &t) { out<<"("<<t.x<<","<<t.y<<")";
return out; }

const int _M=1<<18,_N=N;
struct FT{
   cp tmp[_M*2+5],aa[_M],bb[_M];
   void FFT(cp *a,int n,int op){
      for(int i=(n>>1),j=1;j<n;j++){
         if(i<j) swap(a[i],a[j]);
         int k; for(k=(n>>1);k&i;i^=k,k>>=1); i^=k;
      }
      for(int m=2;m<=n;m<<=1){
         cp w=get(2*PI*op/m); tmp[0]=cp{1,0};
         for(int j=1;j<(m>>1);j++) tmp[j]=tmp[j-1]*w;
         for(int i=0;i<n;i+=m)
            for(int j=i;j<i+(m>>1);j++){
               cp u=a[j],v=a[j+(m>>1)]*tmp[j-i];
               a[j]=u+v,a[j+(m>>1)]=u-v;
            }
      }
      if(op==-1) rep(i,0,n) a[i]=cp{a[i].x/n,a[i].y/n};
   }
    void In(cp p[],int len,cp a[],int n) {
        rep(i,0,len) p[i]=i<n?a[i]:cp{0,0};
    }
    void Out(int a[],int n,cp p[],int len) {
        rep(i,0,n) a[i]=(int)(p[i].x+eps);
    }
    void Shift(int a[],int n,int p) { rep(i,n,n+p) a[i-n]=a[i]; }
   void Multiply(cp A[],int n,cp B[],int m,int C[],int &len,int op=0) {
       if (op) reverse(A,A+n);
      len=1; while (len<n+m-1) len<<=1;
      In(aa,len,A,n),In(bb,len,B,m),FFT(aa,len,1),FFT(bb,len,1);
```

```
      rep(i,0,len) aa[i]=aa[i]*bb[i];
      FFT(aa,len,-1),Out(C,n+m-1,aa,len);
      if (op) Shift(C,n-1,m),len=m,reverse(A,A+n);
   }
};

void Build(cp A[],int n,char s[],int M,int op,int cc='a') {
   rep(i,0,n) A[i]=(s[i]=='?')?cp{0,0}:get(2*PI/M*(s[i]-cc)*op);
}

int n,m,len,tot=0,tt; char s[N],t[N]; FT T; cp A[_M],B[_M]; int C[_M]; vi ans;

int main() {
    //file_put();

    scanf("%s%s",s,t),n=strlen(s),m=strlen(t);
    rep(i,0,m) tot+=(t[i]!='?');
    Build(A,n,s,26,1),Build(B,m,t,26,-1);
   T.Multiply(B,m,A,n,C,len,1);
   //debug_arr(C,len-1);
    rep(i,0,n-m+1) if (C[i]>=tot) ans.pb(i);
    printf("%d\n",tt=ans.size());
    rep(i,0,tt) printf("%d\n",ans[i]);

    return 0;
}
```

2.7 二维 FFT__单模式串__模式串带通配符匹配

```
struct cp{ double x,y; };
inline cp operator + (cp &a,cp &b) { return cp{a.x+b.x,a.y+b.y}; }
inline cp operator - (cp &a,cp &b) { return cp{a.x-b.x,a.y-b.y}; }
inline cp operator * (cp &a,cp &b) { return cp{a.x*b.x-a.y*b.y,a.x*b.y+a.y*b.x}; }
inline cp get(double x) { return cp{cos(x),sin(x)}; }
inline ostream& operator<<(ostream &out,const cp &t) { out<<"("<<t.x<<","<<t.y<<")";
return out; }

const int _M=2048,_N=N;
template <class V>
struct FT{
   cp tmp[_M*2+5],aa[_M][_M],bb[_M][_M];
   void FFT(cp *a,int n,int op){
      for(int i=(n>>1),j=1;j<n;j++){
         if(i<j) swap(a[i],a[j]);
         int k; for(k=(n>>1);k&i;i^=k,k>>=1); i^=k;
      }
      for(int m=2;m<=n;m<<=1){
         cp w=get(2*PI*op/m); tmp[0]=cp{1,0};
         for(int j=1;j<(m>>1);j++) tmp[j]=tmp[j-1]*w;
         for(int i=0;i<n;i+=m)
            for(int j=i;j<i+(m>>1);j++){
               cp u=a[j],v=a[j+(m>>1)]*tmp[j-i];
               a[j]=u+v,a[j+(m>>1)]=u-v;
            }
      }
      if(op==-1) rep(i,0,n) a[i]=cp{a[i].x/n,a[i].y/n};
   }
   void FFT(cp a[][_M],int n,int op) { rep(i,0,n) FFT(a[i],n,op); }
   template <class T>
   void Transpose(T a[][_M],int n) {
       rep(i,0,n) rep(j,0,i) swap(a[i][j],a[j][i]);
    }
    void Reverse(V a[][_M],int n,int m) {
        rep(i,0,(n-1>>1)+1) rep(j,0,m) swap(a[i][j],a[n-1-i][j]);
        rep(i,0,n) rep(j,0,(m-1>>1)+1) swap(a[i][j],a[i][m-1-j]);
    }
    void Shift(int a[][_M],int n,int m,int p,int q) {
        rep(i,n,n+p) rep(j,m,m+q) a[i-n][j-m]=a[i][j];
    }
    void In(cp p[][_M],int len,V a[][_M],int n,int m) {
        rep(i,0,len) rep(j,0,len) p[i][j]=i<n&&j<m?a[i][j]:cp{0,0};
    }
    void Out(int a[][_M],int n,int m,cp p[][_M],int len) {
        rep(i,0,n) rep(j,0,m) a[i][j]=(int)(p[i][j].x+eps);
    }
   void Multiply(V A[][_M],int n,V B[][_M],int m,int C[][_M],int &len,int op=0) {
       if (op) Reverse(A,n,n);
      len=1; while (len<n+m-1) len<<=1;
      In(aa,len,A,n,n),In(bb,len,B,m,m),FFT(aa,len,1),FFT(bb,len,1);
        Transpose(aa,len),Transpose(bb,len),FFT(aa,len,1),FFT(bb,len,1);
      rep(i,0,len) rep(j,0,len) aa[i][j]=aa[i][j]*bb[i][j];
      FFT(aa,len,-1),Transpose(aa,len),FFT(aa,len,-1),Out(C,len,len,aa,len);
      if (op) Shift(C,n-1,n-1,m,m),len=m,Reverse(A,n,n);
   }
};

void Build(cp A[][_M],int n,int m,char s[][405],int M,int op,int cc='a') {
   rep(i,0,n) rep(j,0,m) A[i][j]=(s[i][j]=='?')?cp{0,0}:get(2*PI/M*(s[i][j]-cc)*op);
}

int n1,n2,m1,m2,nn,mm,len,tot=0; char s[405][405],t[405][405]; FT<cp> T; cp A[_M][_M],B[
_M][_M]; int C[_M][_M];

int main() {
    //file_put();

    scanf("%d%d",&n1,&m1);
    rep(i,0,n1) scanf("%s",s[i]);
    scanf("%d%d",&n2,&m2),nn=n1+n2,mm=m1+m2;
    rep(i,0,n2) scanf("%s",t[i]);
    rep(i,0,n2) rep(j,0,m2) tot+=(t[i][j]!='?');
    Build(A,n1,m1,s,26,1),Build(B,n2,m2,t,26,-1);
    rep(i,0,nn) rep(j,0,mm) {
        if (i<n1 && j<m1) continue;
        A[i][j]=A[i%n1][j%m1];
    }
    //debug_arr2(A,nn-1,mm-1);
    //debug_arr2(B,n2-1,m2-1);
   T.Multiply(B,max(n2,m2),A,max(nn,mm),C,len,1);
    //debug_arr2(C,len-1,len-1);
```

```
    rep(i,0,n1) {
        rep(j,0,m1) printf("%c","01"[C[i][j]>=tot]);
        printf("\n");
    }

    return 0;
}
```

3 数据结构

3.1 BIT_二维_区间增减_区间查询

```
template <class V>
struct BIT{
   static const int N=1005,M=1005; int n,m;
   V t1[N][M], t2[N][M], t3[N][M], t4[N][M];
   void init(int _n,int _m) { mem(t1,0),mem(t2,0),mem(t3,0),mem(t4,0); n=_n,m=_m;  }
   void add(int x,int y,V z){
      for (int X=x; X<=n; X+=X&-X)
         for (int Y=y; Y<=m; Y+=Y&-Y) {
            t1[X][Y]+=z, t2[X][Y]+=z*x;
            t3[X][Y]+=z*y, t4[X][Y]+=z*x*y;
         }
   }
   void add(int xa,int ya,int xb,int yb,V z){
      add(xa,ya,z),add(xa,yb+1,-z);
      add(xb+1,ya,-z),add(xb+1,yb+1,z);
   }
   V ask(int x, int y){
      V res=0;
      for(int i=x; i; i-=i&-i)
         for(int j=y; j; j-=j&-j)
            res+=(x+1)*(y+1)*t1[i][j]
               -(y+1)*t2[i][j]-(x+1)*t3[i][j]+t4[i][j];
      return res;
   }
   V ask(int xa,int ya,int xb,int yb){
      return ask(xb,yb)-ask(xb,ya-1)-ask(xa-1,yb)+ask(xa-1,ya-1);
   }
};
```

3.2 BIT_二维_区间增减_单点查询

```
template <class V>
struct BIT{
   static const int N=1005,M=1005; int n,m; V t[N][M];
   void init(int _n,int _m) { mem(t,0); n=_n,m=_m;  }
   void add(int x,int y,V z) {
      for (int i=x; i<=n; i+=i&-i)
         for (int j=y; j<=m; j+=j&-j)
            t[i][j]+=z;
   }
   void add(int xa,int ya,int xb,int yb,V z) {
      add(xa,ya,z),add(xa,yb+1,-z),add(xb+1,ya,-z),add(xb+1,yb+1,z);
   }
   V ask(int x, int y){
      V ret=0;
      for (int i=x; i; i-=i&-i)
         for (int j=y; j; j-=j&-j)
            ret+=t[i][j];
      return ret;
   }
};
```

3.3 BIT_二维_单点修改_区间查询

```
template <class V>
struct BIT{
   static const int N=1005,M=1005; int n,m; V t[N][M];
   void init(int _n,int _m) { mem(t,0); n=_n,m=_m;  }
   void add(int x,int y,V z) {
      for (int i=x; i<=n; i+=i&-i)
         for (int j=y; j<=m; j+=j&-j)
            t[i][j]+=z;
   }
   V ask(int x, int y){
      V ret=0;
      for (int i=x; i; i-=i&-i)
         for (int j=y; j; j-=j&-j)
            ret+=t[i][j];
      return ret;
   }
   V ask(int xa,int ya,int xb,int yb) {
      return ask(xb,yb)-ask(xb,ya-1)-ask(xa-1,yb)+ask(xa-1,ya-1);
   }
};
```

3.4 BIT_二维_说明

```
/*
*
* BIT_ 二维 _ 说明
*
* 【说明】
*
* 1. 凡是区间 / 矩形的传参，都是左上、右下表示法，点的坐标是 ( 行，列 ) ，二维矩阵模型
*
* 2. 根据实际需求，修改 N 和 M 的大小，表示二维树状数组维护的矩阵规模
*
* 3. 当空间巨大时，考虑分块 / 分治等算法，想办法优化一维，减小规模
*/
```

3.5 LIS_T

```
struct LIS_T{
   int n,*a,dep[N],f[N],c[N],cnt,t; vi V[N];
```

```
   inline void Clear() { rep(i,0,n+1) dep[i]=0,f[i]=-1,c[i]=0,V[i].clear(); cnt=0; }
   inline void I(int x) {
      t=dep[x]=lower_bound(c+1,c+cnt+1,x)-c;
      c[t]=x,V[t].pb(x),f[x]=c[t-1],cnt+=(t>cnt);
   }
   inline void Build() { rep(i,1,n+1) I(a[i]); }
   inline void Init(int _a[],int _n) { a=_a,n=_n; Clear(); Build(); }
};

/*
*
* 注： LIS 树模板
*
* 用户可以调用接口 Init(a,n)+Build()
*
* 详细说明请看 md 文件
*
*/
```

3.6 Tree_森林_寻找中心_构造直径

```
struct D{ int x,y,len; }; struct C{ int c,len; };
struct D{ int x,y,len; }; struct C{ int c,len; };
typedef vector<pair<D,C> > VPDC;
struct Tree{
   static const int N=100005; vi V[N]; bool v[N]; int n,f[N],len,now;
   void Init(int _n) { n=_n; rep(i,0,n+1) V[i].clear(); mem(v,0),mem(f,0); }
   void I(int x,int y) { V[x].pb(y),V[y].pb(x); }
   void dfs(int x,int p,int d,int op) {
      if (op) f[x]=p; else v[x]=1;
      if (d>=len) len=d,now=x;
      for (auto y:V[x]) if (y!=p) dfs(y,x,d+1,op);
   }
   C find_center() {
      int x=now,k=(len+1)>>1,kk=k;
      while (k--) x=f[x]; return C{x,kk};
   }
   VPDC solve() {
      VPDC ans; int xx,yy;
      rep(x,1,n+1) if (!v[x]) {
         len=0,dfs(now=x,0,0,0),xx=now;
         dfs(now,0,0,1),yy=now,ans.pb(mp(D{xx,yy,len},find_center()));
      }
      return ans;
   }
};


/*
*
* 森林中求每棵树的中心编号，并且构造出一条直径
*
* 直径是指最长链，中心是指以此为根，树高最小
*
* 注：
*
* 1. 注意记得初始化 T.Init(n) 里面带清空,
*
* 2.D 是直径结构体， x 和 y 是端点， len 是直径长度
*
* 3.C 是中心结构体， c 是中心点编号， len 是以中心 c 为根，该子树树高 ( 根深度是 0) ；此树高 = 直径长 /2 上取整
*
* 4. 用户调用 solve() ，返回一个 VPDC 表示所有子树中心和直径的信息集合，该 vector 的 size 是森林中树的个数
*
* 5. 该代码中， dfs 里 len 表示当前最长路长度， now 为最远走到的结点编号；
*
* f[] 用于从最远点回溯寻找中心，回溯时打印途径结点，可以构造完整直径路径
*
*/
```

gcd_lcm卷积

卷积是一种比较耗费计算资源的基础数学运算，在数学上我们处理卷积的一般方法都是通过各种变换或者转化，使得问题转化为比较容易计算的计算类型(比如序列点积)，想方设法加速计算的效率，当然这需要卷积本身具有一定的数学性质。

***gcd*卷积**是说我们需要计算的卷积公式为：

$$c_k = \sum_{(i,j)=k} a_i * b_j$$

令 A, B, C 分别为 a, b, c 的类点值序列 $(\text{闭区间}[1, n], \text{下标超过} n \text{值为} 0)$，

其中一个定义如下：

$$A_k = \sum_{k|d} a_d$$

那么即有：

$$C_k = \sum_{k|d} c_d = \sum_{k|d} \sum_{(i,j)=d} a_i * b_j = \sum_{k|(i,j)} a_i * b_j$$

$$= \sum_{k|i} \sum_{k|j} a_i * b_j = (\sum_{k|i} a_i) * (\sum_{k|i} b_i) = A_k * B_k$$

我们得到：

$$C = A \bullet B$$

序列反演公式：

$$a_k = \sum_{k|d} A_d * \mu_{\frac{d}{k}}$$

***lcm*卷积**是说我们需要计算的卷积公式为：

$$c_k = \sum_{[i,j]=k} a_i * b_j$$

令A,B,C分别为a,b,c的类点值序列，其中一个定义如下：

$$A_k=\sum_{d|k}a_d \Leftrightarrow A=a\circ I$$

那么即有：

$$C_k=\sum_{d|k}c_d=\sum_{d|k}\sum_{[i,j]=d}a_i*b_j=\sum_{[i,j]|k}a_i*b_j$$

$$=\sum_{i|k}\sum_{j|k}a_i*b_j=(\sum_{i|k}a_i)*(\sum_{i|k}b_i)=A_k*B_k$$

我们得到：

$$C=A\bullet B$$

序列反演公式：

$$a_k=\sum_{d|k}A_d*\mu_{\frac{k}{d}}=\sum_{x*y=k}A_x*\mu_y\Leftrightarrow a=A\circ\mu$$

LIS_T说明

给定一个排列，1-n中任何一个数对应于树上一个节点，根为0元素，***根到节点i的有序路径表示以i这个值为结尾的字典序最小的最长LIS方案***，其构造方法如下：

维护一个**单调迭代数组**(是这样一个数据结构，不妨设单调递增，支持末端插入元素的动态数组，单点替换，修改操作不影响数组单调性)，s[1..cnt]，其中某个时刻T时，插入元素x，**lower_bound**查找s中第一个严格大于x的下标i，s[i]由y替换为x(y>x)表示以值x结尾的LIS长度为i，且存在一个***字典序最小***的方案，方案中x的前驱元素应该是此时的s[i-1]，利用前驱关系构建出来的显然是一棵有序树(**儿子们相对有序**)，***元素y被x替代，表明x和y在同一层，且x在y的右侧***；这个数组在某个时刻的数学意义是这棵树的右侧投影节点编号，这棵树的深度是整个排列的LIS长度，同层元素递减；

这个结构可以很方便的证明排列的***最小递增子序列的切割定理***，一个排列最少可以分成k个单调递减的子序列，其中k为最长上升序列的长度；这是因为，*只要把树上每层的元素拿出来，就是一个方案*，这表明k至多是LIS长度，而树上深度最大的叶子节点，显然要至少深度个递减序列覆盖，所以k至少是LIS长度，故k=len(LIS)得证；

NTT_2D使用说明

接口说明

1）FT部分：

void Init(int _K,int p)：两个参数为用户所期望的结果最大次数，和模数

void init_w(int m)：预处理长度为1<<m的w数组和rev数组

void FFT(vector& A,int m,int op)：变换接口，长度为1<<m，op为0表示普通和卷积，op为1表示差卷积

void multiply(const vector& A,const vector& B,vector *C)：

乘法接口，注意第三个参数传入指针

2）Matrix部分：

void Set_m(int _m,int x=0)：将每列resize为m，不足元素填充为x

void Set_n(int _n)：将行扩充为n

void Set(int _n,int _m,int x=0)：设置矩阵规模

void Transpose()：转置接口

void Reverse()：翻转接口，不光行翻转，列也翻转

void Shift(int x,int y)：移位接口，将(x,y)设置为矩阵左上角元素，整体移位，不足部分填充0

void FFT(FT &T,int len,int op)：批量行变换，对每行进行正/逆变换

void print() const：打印接口

void Normalize(int _p)：规范化接口，保证矩阵中每个元素都严格非负且已模

void Random()：产生随机数，填充此矩阵

==：矩阵可以直接比较判等

3）Calculator部分：

void Init(int p)：预处理模数

void Multiply(const Matrix &A,const Matrix &B,Matrix &C,int op=0)：

乘法接口，op=0为和卷积，op=1为差卷积

void Multiply_B(const Matrix &A,const Matrix &B,Matrix &C)：

暴力和卷积接口

void Multiply_B_sub(const Matrix &A,const Matrix &B,Matrix &C)：

暴力差卷积接口