Cartilla de código piola

UTN FRSF - El Rejunte

2017

Índice

1. C/C++ **2**
1.1. I/O . . . 2
1.1.1. scanf Format Strings . . . 2
1.1.2. printf Format Strings . . . 2

2. Template del Rejunte **3**

3. Constantes y Tablas **4**
3.1. Constantes . . . 4

4. Estructuras de datos **4**
4.1. Disjoint Sets . . . 4
4.1.1. Union Find (OOP) . . . 4
4.1.2. Union Find (C Style/Static) . . . 4
4.2. Segment Trees . . . 5
4.2.1. Range Sum Query (lazy) . . . 5
4.2.2. Range Min/Max Query (lazy) . . . 5

5. Strings **6**

6. Geometría **6**

7. Matemática **6**
7.1. GCD & LCM . . . 6
7.2. Combinatoria . . . 6
7.3. Exponenciación de Matrices y Fibonacci . . . 7
7.4. Operaciones Modulares . . . 7
7.5. Funciones de Primos . . . 7
7.6. Phollard's Rho . . . 8

8. Grafos **9**

9. Flow **9**
9.1. Edmond Karp . . . 9
9.2. Push Relabel . . . 9

10. Juegos **10**
10.1. Ajedrez . . . 10
10.1.1. Non-Attacking N Queen . . . 10

11. Utils **10**
11.1. Convertir string a num e viceversa . . . 10

1. C/C++

1.1. I/O

1.1.1. scanf Format Strings

`%[*][width][length]specifier`

spec	Tipo	Descripción
i	int	Dígitos dec. [0-9], oct. (0)[0-7], hexa (0x|0X)[0-9a-fA-F]. Con signo.
d, u	int, unsigned	Dígitos dec. [+-0-9].
o	unsigned	Dígitos oct. [+-0-7].
x	unsigned	Dígitos hex. [+-0-9a-fA-F]. Prefijo 0x,0X opcional.
f, e, g	float	Dígitos dec. c/punto flotante [+-.0-9]. Prefijo 0x,0X y sufijo e,E opcionales.
c, [width]c	char, char*	Siguiente carácter. Lee width chars y los almacena contiguamente. No agrega \0.
s	char*	Secuencia de chars hasta primer espacio. Agrega \0.
p	void*	Secuencia de chars que representa un puntero.
[chars]	Scanset, char*	Caracteres especificados entre corchetes.] debe ser primero en la lista, - primero o último. Agrega \0
[^chars]	!Scanset, char*	Caracteres no especificados entre corchetes.
n	int	No consume entrada. Almacena el número de chars leídos hasta el momento.
%		%% consume un %

sub-specifier	Descripción
*	Indica que se leerá el dato pero se ignorará. No necesita argumento.
width	Cantidad máxima de caracteres a leer.
lenght	Uno de hh, h, l, ll, j, z, t, L. Ver tabla siguiente.

length	d i	u o x
(none)	int*	unsigned int*
hh	signed char*	unsigned char*
h	short int*	unsigned short int*
l	long int*	unsigned long int*
ll	long long int*	unsigned long long int*

Continuación

length	d i	u o x
j	intmax_t*	uintmax_t*
z	size_t*	size_t*
t	ptrdiff_t*	ptrdiff_t*
L		

length	f e g a	c s [ ] [^]	p	n
(none)	float*	char*	void**	int*
hh				signed char*
h				short int*
l	double*	wchar_t*		long int*
ll				long long int*
j				intmax_t*
z				size_t*
t				ptrdiff_t*
L	long double*			

1.1.2. printf Format Strings

`%[flags][width][.precision][length]specifier`

specifier	Descripción	Ejemplo
d or i	Entero decimal con signo	392
u	Entero decimal sin signo	7235
o	Entero octal sin signo	610
x	Entero hexadecimal sin signo	7fa
X	Entero hexadecimal sin signo (mayúsculas)	7FA
f	Decimal punto flotante (minúsculas)	392.65
F	Decimal punto flotante (mayúsculas)	392.65
e	Notación científica (mantisa/exponente), (minúsculas)	3.9265e+2
E	Notación científica (mantisa/exponente), (mayúsculas)	3.9265E+2
g	Utilizar la representaciíon más corta: %e ó %f	392.65
G	Utilizar la representaciíon más corta: %E ó %F	392.65
a	Hexadecimal punto flotante (minúsculas)	-0xc.90fep-2
A	Hexadecimal punto flotante (mayúsculas)	-0XC.90FEP-2
c	Caracter	a
s	String de caracteres	sample
p	Dirección de puntero	b8000000
n	No imprime nada. El argumento debe ser int*, almacena el número de caracteres imprimidos hasta el momento.	
%	Un % seguido de otro % imprime un solo %	%

flag	Descripción
_	Justificación a la izquierda dentro del campo width (ver width sub-specifier).
+	Forza a preceder el resultado de texttt+ o texttt-.
(espacio)	Si no se va a escribir un signo, se inserta un espacio antes del valor.
#	Usado con o, x, X specifiers el valor es precedido por 0, 0x, 0X respectivamente para valores distintos de 0.
0	Rellena el número con texttt0 a la izquierda en lugar de espacios cuando se especifica width.

width	Descripción
(número)	Número mínimo de caracteres a imprimir. Si el valor es menor que número, el resultado es rellando con espacios. Si el valor es mayor, no es truncado.
*	No se especifica width, pero se agrega un argumento entero precediendo al argumento a ser formateado. Ej. printf("---%*d----\n", 3, 2); ⇒ "---- 5----".

precision	Descripción
.(número)	Para d, i, o, u, x, X: número mínimo de dígitos a imprimir. Si el valor es más chico que número se rellena con 0. Para a, A, e, E, f, F: número de dígitos a imprimir después de la coma (default 6). Para g, G: Número máximo de cifras significativas a imprimir. Para s: Número máximo de caracteres a imprimir. Trunca.
.*	No se especifica precision pero se agrega un argumento entero precediendo al argumento a ser formateado.

length	d i	u o x X
(none)	int	unsigned int
hh	signed char	unsigned char
h	short int	unsigned short int
l	long int	unsigned long int
ll	long long int	unsigned long long int
j	intmax_t	uintmax_t
z	size_t	size_t
t	ptrdiff_t	ptrdiff_t
L		

length	f F e E g G a A	c	s	p	n
(none)	double	int	char*	void*	int*
hh					signed char*
h					short int*
l		wint_t	wchar_t*		long int*
ll					long long int*
j					intmax_t*
z					size_t*
t					ptrdiff_t*
L	long double				

2. Template del Rejunte

```cpp
#include <bits/stdc++.h>
#define sqr(a) ((a)*(a))
#define rsz resize
#define forr(i,a,b) for(int i=(a);i<(b);i++)
#define forn(i,n) forr(i,0,n)
#define dforn(i,n) for(int i=n-1;i>=0;i--)
#define forall(it,v) for(auto it=v.begin();it!=v.end();it++)
#define sz(c) ((int)c.size())
#define zero(v) memset(v, 0, sizeof(v))
#define pb push_back
#define mp make_pair
#define lb lower_bound
#define ub upper_bound
#define fst first
#define snd second
using namespace std;

typedef long long ll;
typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

int main()
{
    //freopen("input","r",stdin);
    //freopen("output","w",stdout);
  ios::sync_with_stdio(false);
  cin.tie(NULL);
  cout.tie(NULL);

    return 0;
}
```

3. Constantes y Tablas

3.1. Constantes

```cpp
#define INF 1000000000 // 1 billon, entra en int
#define EPS 1e-12
#define PI 3.14159265358979323846
```

4. Estructuras de datos

4.1. Disjoint Sets

4.1.1. Union Find (OOP)

Utiliza: <vector>

Notas: Rangos [i,j] (*0 based*). No recomendable si se tienen que crear y destruir muchos objetos. Probar funcionamiento en casos límites.

```cpp
class UnionFind
{
private:
  vector<int> p, rank, setSize;
  int numSets;
public:
  UnionFind(int N)
  {
    setSize.assign(N, 1);
    numSets = N;
    rank.assign(N, 0);
    p.assign(N, 0);
    for (int i = 0; i < N; i++) p[i] = i;
  }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j)
  {
    int x = findSet(i), y = findSet(j);
    if (!(x==y))
    {
      numSets--;
      if (rank[x] > rank[y])
      {
        p[y] = x;
        setSize[x] += setSize[y];
      }
      else
      {
        p[x] = y;
        setSize[y] += setSize[x];
        if (rank[x] == rank[y]) rank[y]++;
      }
    }
  }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};
```

4.1.2. Union Find (C Style/Static)

Utiliza: <cstring>

Notas: Rangos [i,j] (*0 based*). En init(n), n $\leq$ MAXN

```cpp
#define MAXN ...
int p[MAXN], rank[MAXN], setSize[MAXN], numSets;

inline void init(int n)
{
    memset(rank,0, sizeof(int)*n);
    for(int i=0;i<n;++i) p[i]=i, setSize[i]=1;
    numSets = n;
}
inline int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
inline bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
void unionSet(int i, int j)
{
    int x = findSet(i), y = findSet(j);
    if (!(x==y))
    {
        numSets--;
        if (rank[x] > rank[y])
        {
            p[y] = x;
            setSize[x] += setSize[y];
        }
        else
        {
            p[x] = y;
            setSize[y] += setSize[x];
            if (rank[x] == rank[y]) rank[y]++;
        }
    }
}
inline int numDisjointSets() { return numSets; }
inline int sizeOfSet(int i) { return setSize[findSet(i)]; }
```

4.2. Segment Trees

4.2.1. Range Sum Query (lazy)

Utiliza: <cmath>
Notas: Limpiar st y lazy en cada testcase. En main():

- N es la cantidad de elementos del problema.
- $(0 \leq i,j < N)$ build(1,0,N-1), update(1,0,N-1,i,j,value), query(1,0,N-1,i,j). Los elementos en arr tienen que estar desde arr[0].
- $(1 \leq i,j \leq N)$ build(1,1,N), update(1,1,N,i,j,value), query(1,1,N,i,j). Los elementos en arr tienen que estar desde arr[1].

```
const int MAXN = ...;
const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);

#define OP(a,b) ((a) + (b))
#define NEUTRO 0LL

#define left(x) ((x)<<1)
#define right(x) (((x)<<1)+1)
#define middle(a,b) (((a)+(b))>>1)

ll st[MAXST], lazy[MAXST]; int arr[MAXN+1];

void build(int node, int a, int b)
{
    if(a>b) return;
    if(a==b){st[node] = arr[a]; return;}
    int l=left(node), r=right(node), m=middle(a,b);
    build(l,a,m);
    build(r,m+1,b);
    st[node]=OP(st[l],st[r]);
}

void update(int node, int a, int b, int i, int j, ll value)
{
    int l=left(node), r=right(node), m=middle(a,b);

    if(lazy[node]!=0LL)
    {
        st[node] = (b-a+1)*lazy[node]; // SET
        //      += (b-a+1)*lazy[node]; // ADD
        if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
        //       lazy[l]+=lazy[node],lazy[r]+=lazy[node]; // ADD
        lazy[node] = 0LL;
    }

    if(a>b || a>j || b<i) return;

    if(a>=i && b<=j)
    {
        st[node] = (b-a+1)*value; // SET
        //      += (b-a+1)*value; // ADD
        if(a!=b) lazy[l] = lazy[r] = value;
        //       lazy[l]+=value,lazy[r]+=value; // ADD
        return;
    }

    update(l,a,m,i,j,value);
    update(r,m+1,b,i,j,value);

    st[node] = OP(st[l],st[r]);
}

ll query(int node, int a, int b, int i, int j)
{
    if(a>b || a>j || b<i) return NEUTRO;

    int l=left(node), r=right(node), m=middle(a,b);

    if(lazy[node]!=0)
    {
        st[node] = (b-a+1)*lazy[node];
        if(a!=b) lazy[l] = lazy[r] = lazy[node];
        lazy[node] = 0;
    }
    if(a>=i && b<=j) return st[node];

    int ql = query(l,a,m,i,j);
    int qr = query(r,m+1,b,i,j);
    return OP(ql,qr);
}
```

4.2.2. Range Min/Max Query (lazy)

Utiliza: <cmath>
Notas: Limpiar st y lazy en cada testcase. En main():

- N es la cantidad de elementos del problema.
- $(0 \leq i,j < N)$ build(1,0,N-1), update(1,0,N-1,i,j,value), query(1,0,N-1,i,j). Los elementos en arr tienen que estar desde arr[0].
- $(1 \leq i,j \leq N)$ build(1,1,N), update(1,1,N,i,j,value), query(1,1,N,i,j). Los elementos en arr tienen que estar desde arr[1].

```
const int MAXN  = ...;
const int MAXST = (int) (2*exp2(ceil(log2(MAXN)))+1);

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))

#define OP(a,b) (max(a,b)) // operador binario asociativo
```

```
#define NEUTRO 0       // elemento neutro del operador

#define left(x) ((x)<<1)
#define right(x) (((x)<<1)+1)
#define middle(a,b) (((a)+(b))>>1)

int st[MAXST], lazy[MAXST], arr[MAXN+1];

void build(int node, int a, int b)
{
  if(a>b) return;
  if(a==b) {st[node] = arr[a]; return;}
  int l = left(node), r = right(node), m = middle(a,b);
  build(l,a,m);
  build(r,m+1,b);
  st[node] = OP(st[l], st[r]);
}

void update(int node, int a, int b, int i, int j, int value)
{
  int l = left(node), r = right(node), m = middle(a,b);

  if(lazy[node]!=0)
  {
    st[node] = lazy[node]; // SET [a,b] to value
    //      += lazy[node]; // ADD value to [a,b]
    if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
    //       lazy[l] +=lazy[node], lazy[r] += lazy[node]; // ADD
    lazy[node] = 0;
  }

  if(a>b || a>j || b<i) return;

  if(a>=i && b<=j)
  {
    st[node] = value; // SET
    //    += value; // ADD
    if(a!=b) lazy[l] = lazy[r] = value; // SET
    //     lazy[l] += value, lazy[r] += value; // ADD
    return;
  }

  update(l,a,m,i,j,value);
  update(r,m+1,b,i,j,value);

  st[node] = OP(st[l], st[r]);
}

int query(int node, int a, int b, int i, int j)
{
  if(a>b || a>j || b<i) return NEUTRO;

  int l = left(node), r = right(node), m = middle(a,b);

  if(lazy[node]!=0)
  {
    st[node] = lazy[node]; // SET [a,b] to value
    //      += lazy[node]; // ADD value to [a,b]
    if(a!=b) lazy[l] = lazy[r] = lazy[node]; // SET
    //     lazy[l] +=lazy[node], lazy[r] += lazy[node]; // ADD
    lazy[node] = 0;
  }

  if(a>=i && b<=j) return st[node];

  int ql = query(l,a,m,i,j);
  int qr = query(r,m+1,b,i,j);

  return OP(ql, qr);
}
```

5. Strings

6. Geometría

7. Matemática

7.1. GCD & LCM

```
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
int lcm(int a, int b) { return a * (b / gcd(a, b));}
```

7.2. Combinatoria

```
void cargarComb()//O(MAXN^2)
{
  forn(i, MAXN+1) //comb[i][k]=i tomados de a k = i!/(k!*(i-k)!)
  {
    comb[0][i]=0;
    comb[i][0]=comb[i][i]=1;
    forr(k, 1, i) comb[i][k]=(comb[i-1][k-1]+comb[i-1][k]) %MOD;
  }
}
ll lucas (ll n, ll k, int p)
{ //Calcula (n,k) %p teniendo comb[p][p] precalculado.
  ll aux = 1;
  while (n + k)
  {
```

```
    aux = (aux * comb[n %p][k %p]) %p;
    n/=p, k/=p;
  }
  return aux;
}
```

7.3. Exponenciación de Matrices y Fibonacci

```
#define SIZE 350
int NN;
void mul(double a[SIZE][SIZE], double b[SIZE][SIZE])
{
  double res[SIZE][SIZE] = {{0}};
  forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
  forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
}
void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE])
{
  forn(i, NN) forn(j, NN) res[i][j]=(i==j);
  while(n)
  {
    if(n&1) mul(res, a), n--;
    else mul(a, a), n/=2;
  }
}

struct M22{    // |a b|
  tipo a,b,c,d;// |c d| -- TIPO
  M22 operator*(const M22 &p) const {
  return (M22){a*p.a+b*p.c, a*p.b+b*p.d, c*p.a+d*p.c,c*p.b+d*p.d};}
};
M22 operator^(const M22 &p, int n)
{//VER COMO SE PUEDE PONER DENTRO DEL STRUCT
  if(!n) return (M22){1, 0, 0, 1};//identidad
  M22 q=p^(n/2); q=q*q;
  return n %2? p * q : q;
}

ll fibo(ll n)//calcula el fibonacci enesimo en O(logN)
{
  M22 mat=(M22){0, 1, 1, 1}^n;
  return mat.a*f0+mat.b*f1;//f0 y f1 son los valores iniciales
}
```

7.4. Operaciones Modulares

```
ll sumMod(ll a,ll b,ll m=MOD)
{
  return (a%m+b%m)%m;
}
ll difMod(ll a,ll b,ll m=MOD)
{
  ll ret=a%m-b%m;
  if(ret<0) ret+=m;
  return ret;
}
ll mulMod(ll a,ll b,ll m=MOD) //O(log b)
{ //returns (a*b) %c, and minimize overfloor
  ll x=0, y=a%m;
  while(b>0)
  {
    if(b%2==1) x=(x+y)%m;
    y=(y*2)%m;
    b/=2;
  }
  return x%m;
}
ll divMod(ll a,ll b,ll m=MOD)
{
  return mulMod(a,inverso(b),m);
}
ll expMod(ll b,ll e,ll m=MOD) //O(log b)
{
  if(!e) return 1;
  ll q= expmod(b,e/2,m); q=mulmod(q,q,m);
  return e%2? mulmod(b,q,m) : q;
}
```

7.5. Funciones de Primos

```
#define MAXP 100000 //no necesariamente primo
int criba[MAXP+1];
void crearCriba()
{
  int w[] = {4,2,4,2,4,6,2,6};
  for(int p=25;p<=MAXP;p+=10) criba[p]=5;
  for(int p=9;p<=MAXP;p+=6) criba[p]=3;
  for(int p=4;p<=MAXP;p+=2) criba[p]=2;
  for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
  for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
}
vector<int> primos;
void buscarPrimos()
{
  crearCriba();
  forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
}

//factoriza bien numeros hasta MAXP^2
```

```
void fact(ll n,map<ll,ll> &f) //O (cant primos)
{ //llamar a buscarPrimos antes
  forall(p, primos){
    while(!(n %*p))
    {
      f[*p]++;//divisor found
      n/=*p;
    }
  }
  if(n>1) f[n]++;
}

//factoriza bien numeros hasta MAXP
void fact2(ll n,map<ll,ll> &f) //O (lg n)
{ //llamar a crearCriba antes
  while (criba[n])
  {
    f[criba[n]]++;
    n/=criba[n];
  }
  if(n>1) f[n]++;
}

//Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
void divisores(map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::iterator it, ll
    n=1)
{
  if(it==f.begin()) divs.clear();
  if(it==f.end())
  {
    divs.pb(n);
    return;
  }
  ll p=it->fst, k=it->snd; ++it;
  forn(_, k+1) divisores(f, divs, it, n), n*=p;
}
ll cantDivs(map<ll,ll> &f)
{
  ll ret=1;
  forall(it, f) ret*=(it->second+1);
  return ret;
}
ll sumDivs(map<ll,ll> &f)
{
  ll ret=1;
  forall(it, f)
  {
    ll pot=1, aux=0;
    forn(i, it->snd+1) aux+=pot, pot*=it->fst;
    ret*=aux;
  }
  return ret;
}

ll eulerPhi(map<ll,ll> &f) // con criba: O(lg n)
{
  ll ret=n;
  forall(it, f) ret-=ret/it->first;
  return ret;
}
ll eulerPhi2(ll n) // O (sqrt n)
{
  ll r = n;
  forr(i,2,n+1)
  {
    if((ll)i*i>n) break;
    if(n%i==0)
    {
      while(n%i==0) n/=i;
      r -= r/i;
    }
  }
  if (n != 1) r-= r/n;
  return r;
}
```

7.6. Phollard's Rho

```
bool es_primo_prob(ll n, int a)
{
  if(n==a) return true;
  ll s=0,d=n-1;
  while(d%2==0) s++,d/=2;
  ll x=expmod(a,d,n);
  if((x==1) || (x+1==n)) return true;
  forn(i,s-1)
  {
    x=mulmod(x, x, n);
    if(x==1) return false;
    if(x+1==n) return true;
  }
  return false;
}
bool rabin (ll n)  //devuelve true si n es primo
{
  if(n==1) return false;
  const int ar[]={2,3,5,7,11,13,17,19,23};
  forn(j,9) if(!es_primo_prob(n,ar[j])) return false;
  return true;
}
ll rho(ll n)
{
  if((n&1)==0) return 2;
  ll x=2,y=2,d=1;
```

```
  ll c=rand()%n+1;
  while(d==1)
  {
    x=(mulmod(x,x,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    y=(mulmod(y,y,n)+c)%n;
    if(x-y>=0) d=gcd(x-y,n);
    else d=gcd(y-x,n);
  }
  return d==n? rho(n):d;
}
void factRho (ll n,map<ll,ll> &f) //O (lg n)^3 un solo numero
{
  if (n == 1) return;
  if (rabin(n))
  {
    f[n]++;
    return;
  }
  ll factor = rho(n);
  factRho(factor);
  factRho(n/factor);
}
```

8. Grafos

9. Flow

9.1. Edmond Karp

```
#define MAX_V 1000
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto -- unordered_map mejora
//To add an edge use
#define add(a, b, w) G[a][b]=w
int f, p[MAX_V];
void augment(int v, int minE)
{
  if(v==SRC) f=minE;
  else if(p[v]!=-1)
  {
    augment(p[v], min(minE, G[p[v]][v]));
    G[p[v]][v]-=f, G[v][p[v]]+=f;
  }
}
ll maxflow()//O(min(VE^2,Mf*E))
{
  ll Mf=0;
  do
  {
    f=0;
    char used[MAX_V]; queue<int> q; q.push(SRC);
    zero(used), memset(p, -1, sizeof(p));
    while(sz(q))
    {
      int u=q.front(); q.pop();
      if(u==SNK) break;
      forall(it, G[u])
        if(it->snd>0 && !used[it->fst])
        used[it->fst]=true, q.push(it->fst), p[it->fst]=u;
    }
    augment(SNK, INF);
    Mf+=f;
  }while(f);
  return Mf;
}
```

9.2. Push Relabel

```
#define MAX_V 1000
int N;//valid nodes are [0...N-1]
#define INF 1e9
//special nodes
#define SRC 0
#define SNK 1
map<int, int> G[MAX_V];//limpiar esto -- unordered_map mejora
//To add an edge use
#define add(a, b, w) G[a][b]=w
ll excess[MAX_V];
int height[MAX_V], active[MAX_V], count[2*MAX_V+1];
queue<int> Q;

void enqueue(int v)
{
  if (!active[v] && excess[v] > 0) active[v]=true, Q.push(v);
}
void push(int a, int b)
{
  int amt = min(excess[a], ll(G[a][b]));
  if(height[a] <= height[b] || amt == 0) return;
  G[a][b]-=amt, G[b][a]+=amt;
  excess[b] += amt, excess[a] -= amt;
  enqueue(b);
}
void gap(int k)
{
  forn(v, N)
```

```
  {
    if (height[v] < k) continue;
    count[height[v]]--;
    height[v] = max(height[v], N+1);
    count[height[v]]++;
    enqueue(v);
  }
}
void relabel(int v)
{
  count[height[v]]--;
  height[v] = 2*N;
  forall(it, G[v])
  if(it->snd) height[v] = min(height[v], height[it->fst] + 1);
  count[height[v]]++;
  enqueue(v);
}
ll maxflow() //O(V^3)
{
  zero(height), zero(active), zero(count), zero(excess);
  count[0]=N-1; count[N]=1;
  height[SRC] = N;
  active[SRC] = active[SNK] = true;
  forall(it, G[SRC])
  {
    excess[SRC] += it->snd;
    push(SRC, it->fst);
  }
  while(sz(Q))
  {
    int v = Q.front(); Q.pop();
    active[v]=false;
    forall(it, G[v]) push(v, it->fst);
    if(excess[v] > 0)
    count[height[v]] == 1? gap(height[v]):relabel(v);
  }
  ll mf=0;
  forall(it, G[SRC]) mf+=G[it->fst][SRC];
  return mf;
}
```

10. Juegos

10.1. Ajedrez

10.1.1. Non-Attacking N Queen

Utiliza: <algorithm>

Notas: todo es $O(!N \cdot N^2)$.

```
#define NQUEEN 8
#define abs(x) ((x)<0?(-(x)):(x))

int board[NQUEEN];
void inline init(){for(int i=0;i<NQUEEN;++i)board[i]=i;}
bool check(){
    for(int i=0;i<NQUEEN;++i)
        for(int j=i+1;i<NQUEEN;++j)
            if(abs(i-j)==abs(board[i]-board[j]))
                return false;
    return true;
}
//en main
init();
do{
    if(check()){
        //process solution
    }
}while(next_permutation(board,board+NQUEEN));
```

11. Utils

11.1. Convertir string a num e viceversa

```
#include <sstream>
string num_to_str(int x){
  ostringstream convert;
  return (convert << entero).str();
}

int str_to_num(string x){
  int ret;
  istringstream (x) >> ret;
  return ret;
}
```