Algorithm's Spellbook

henviso

Abstract

This will someday become a very organized and structured notebook with nice algorithms and data structures to be used as reference...

Contents

1 Macros

macros.cpp

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cstdlib>
#include <stack>
#include <algorithm>
#include <cctype>
#include <vector>
#include <queue>
#include <tr1/unordered_map>
#include <cmath>
#include <map>
#include <bitset>
#include <set>
#include <iomanip>
#include <utility>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector< ii > vii;
//////////////////////////////////UTIL//////////////////////////////////
#define ALL(x) (x).begin(),x.end()
#define CLEAR0(v) memset(v, 0, sizeof(v))
#define CLEAR(v, x) memset(v, x, sizeof(v))
#define COPY(a, b) memcpy(a, b, sizeof(a))
#define CMP(a, b) memcmp(a, b, sizeof(a))
#define REP(i,n) for(int i = 0; i<n; i++)
#define REPP(i,a,n) for(int i = a; i<n; i++)
#define REPD(i,n) for(int i = n-1; i>-1; i--)
#define REPDP(i,a,n) for(int i = n-1; i>=a; i--)
#define pb push_back
#define pf push_front
#define sz size()
#define mp make_pair
//////////////////////////////NUMERICAL//////////////////////////////
#define INF 0x3f3f3f3f
#define EPS 1e-9
//////////////////////////////BITWISE//////////////////////////////
#define CHECK(S, j) (S & (1 << j))
#define CHECKFIRST(S) (S & (-S))
#define SET(S, j) S |= (1 << j)
#define SETALL(S, j) S = (1 << j)-1
#define UNSET(S, j) S &= ~(1 << j)
#define TOOGLE(S, j) S ^= (1 << j)
//////////////////////////////64 BITS//////////////////////////////
#define LCHECK(S, j) (S & (1ULL << j))
#define LSET(S, j) S |= (1ULL << j)
#define LSETALL(S, j) S = (1ULL << j)-1ULL
#define LUNSET(S, j) S &= ~(1ULL << j)
```

```
#define LTOOGLE(S, j) S ^= (1ULL << j)
//__builtin_popcount(m)
//scanf(" %d ", &t);
//L[i]=L[i/2]+1;

int primes[] = {2, 3, 5, 7};
int p = 4;
int cnt[100];

vi decompose(int x){
	REP(i, p){
		while(x%primes[i]){
			x /= primes[i];
		}
	}
}

int main(){
	REP(i, 10) printf("LA\n");



}
```

Basic

basic.cpp

```
#define SIEVE_MAX 100000000
bitset<SIEVE_MAX+1> _prime;
vi primes;

void sieve(){
	_prime.set();
	_prime[0] = _prime[1] = 0; primes.pb(2);
	for(ll i = 4; i<SIEVE_MAX; i+=2) _prime[i] = 0;
	for(ll i = 3; i<SIEVE_MAX; i+=2){
		if(_prime[i]){
			for(ll j = i*i; j<SIEVE_MAX; j+=2*i) _prime[j] = 0;
			primes.push_back((int) i);
		}
	}
}

bool isPrime(ll N){
	if(N <= SIEVE_MAX) return _prime[N];
	for(int i = 0; i<(int) primes.size(); i++){
		if(N%primes[i] == 0) return false;
	}
	return true;
}

vi primeFactors(ll N) {
	vi factors;	//TROCAR PRA vll SE O NUMERO FOR > QUE INT
	ll PF_idx = 0, PF = primes[PF_idx];
	while(N > 1 && (PF*PF <= N)){
		while(N%PF == 0){ N /= PF; factors.push_back((int) PF); }
		PF = primes[++PF_idx];
	}
	if(N > 1) factors.push_back((int) N);
	return factors;
}

ll numPf(ll N){
	ll PF_idx = 0, PF = primes[PF_idx], ans = 0;
	while(N > 1 && (PF*PF <= N)){
		while(N%PF == 0){ N /= PF; ans++; }
		PF = primes[++PF_idx];
	}
	if(N > 1) ans++;
	return ans;
}

ll numDiv(ll N){
	ll PF_idx = 0, PF = primes[PF_idx], ans = 1;
	while(N > 1 && (PF*PF <= N)){
		ll power = 0;
		while(N%PF == 0){ N /= PF; power++; }
		ans *= (power+1);
		PF = primes[++PF_idx];
	}
	if(N > 1) ans *= 2;
	else return N;
	return ans;
}

ll EulerPhi(ll N){
	ll PF_idx = 0, PF = primes[PF_idx], ans = N;
	while(N != 1 && (PF * PF <= N)){
		if(N%PF == 0) ans -= ans / PF;
		while(N%PF == 0) N /= PF;
		PF = primes[++PF_idx];
	}
	if(N != 1) ans -= ans/N;
	return ans;
}

ll fastExp(ll base, ll p, ll m){
	if(p == 0LL) return 1LL;
	else if(p == 1LL) return base%m;
	else{
		unsigned long long res = fastExp(base, p/2LL, m);
		res = (res*res)%m;
		if(p%2LL == 1LL) res = (res*base)%m;
		return res;
	}
}

int64 expo(int64 a, int64 b, int64 m)
{
int64 y = a, x = 1;
while (b > 0) {
if (b % 2 == 1) {
x = (x*y) % m;
```

```
}
y = (y*y) % m;
b = b/2;
}
return x % m;
}

ll x, y, d;

//inv modular de a mod b = m eh o x
void extendedEuclid(int a, int b){
	if(b == 0){ x = 1; y = 0; d = a; return; }
	extendedEuclid(b, a%b);
	ll x1 = y;
	ll y1 = x - (a/b) * y;
	x = x1;
	y = y1;
}

ll invMult(ll a, ll m){
	extendedEuclid(a, m);
	x = x%m;
	if(x < 0) x += m;
	return x;
}

ll weakComp(int n, int k){
	ll res = fat[n+k-1];
	ll a = (fat[n] * fat[k-1])%M;
	if(res == 0 || a == 0) return 0;
	invMult(a, M);
	//cout << "FAT DEU " << fat[n+k-1] << " D DEU " << d << " INV DEU " << x << endl;
	if(a != 1) if(d == 1) res = (res*x)%M;
	return res;
}

int grid[360][360];
int R, C;

int dr[] = {1,-1,0,0};
int dc[] = {0,0,1,-1};

void floodfill(int r, int c){
	//cout << "CHAMANDO FLOOD EM " << r << " " << c << " E O N " << n << endl;
	if(r < 0 || r >= R || c < 0 || c >= C) return;
	if(classify(grid[r][c]) != classe) return;
	grid[r][c] = -1;
	//cout << " DESBLOQUEANDO " << r << " " << c << endl;
	for(int d = 0; d<4; d++) floodfill(r+dr[d], c+dc[d], classe);
	return;
}

ll mdc(ll a, ll b) {
  if(a<0LL) a = -a;
  if(b<0LL) b = -b;
  if(b == 0LL)
    return a;
  else
    return mdc(b, a%b);
}

ll mmc(ll a, ll b){
	return (a*b)/mdc(a,b);
}
```

Recorrencia Linear

reclog.cpp

```
typedef vector<vector<ll> > matrix;
const ll MOD = 1000000007;
const int K = 2;

// computes A * B
matrix mul(matrix A, matrix B)
{
   matrix C(K, vector<ll>(K));
   REP(i, K) REP(j, K) REP(k, K)
      C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
   return C;
}

// computes A ^ p
matrix pow(matrix A, int p)
{
   if (p == 1)
      return A;
   if (p % 2)
      return mul(A, pow(A, p-1));
   matrix X = pow(A, p/2);
   return mul(X, X);
}

// returns the N-th term of Fibonacci sequence
int fib(int N)
{
   // create vector F1
   vector<ll> F1(K);
   F1[0] = 1;
   F1[1] = 1;

   // create matrix T
   matrix T(K, vector<ll>(K));
   T[0][0] = 0, T[0][1] = 1;
   T[1][0] = 1, T[1][1] = 1;

   // raise T to the (N-1)th power
```

```
    if (N == 1)
        return F1[0];
    T = pow(T, N−1);

    // the answer is the first row of T . F1
    ll res = 0;
    REP(i, K)
        res = (res + T[0][i] ∗ F1[i]) % MOD;
    if(res < 0) res += MOD;
    return res;
}

int main(){
        int t, x;
        scanf("␣%d␣", &t);
        while(t−−){
                scanf("␣%d␣", &x);
                printf("%d\n", fib(x));
        }
}
```

Strings Hash

hashstrings.cpp

```
typedef long long unsigned hash;

#define MAXS 10
#define MAXN 300100
#define B 33ULL
#define C 5381ULL

ull power[MAXN];
hash p[MAXS][MAXN];
char str[MAXS][MAXN];
int t[MAXS], n;

void precalc(){
        power[0] = 1ULL;
        REPP(i, 1, MAXN) power[i] = power[i−1]∗B;
        REP(i, n){
                p[i][0] = 0ULL;
                REP(j, t[i]){
                        p[i][j+1] = p[i][j]∗B + str[i][j];
                }
        }
}

void print(int e, int a, int b){
        for(int i = a; i<=b; i++){
                printf("%c", str[e][i]);
        }
        printf("\n");
}

hash calc_dhash(int e, int a, int b) {
        if (a > b) return 0;
        return p[e][b+1] − p[e][a] ∗ power[b − a + 1];
}
```

2 Data Structures

Arbitrary Precision Integers

bigint.cpp

```
const int DIG = 1; //numero de algarismos de cada digito na entrada
const int BASE = 10; // BASE**3 < 2**51
const int TAM = 2048;

struct bigint {
    int v[TAM], n;
    bigint(int x = 0): n(1) {
        memset(v, 0, sizeof(v)); v[n++] = x; fix();
    }
    bigint(char ∗s): n(1) {
        memset(v, 0, sizeof(v));
        int sign = 1;
        while (∗s && !isdigit(∗s)) if (∗s++ == '−') sign ∗= −1;
        char ∗t = strdup(s), ∗p = t + strlen(t);
        while (p > t) {
            ∗p = 0; p = max(t, p − DIG);
            sscanf(p, "%d", &v[n]);
            v[n++] ∗= sign;
        }
        free(t); fix();
    }
    bigint& fix(int m = 0) {
        n = max(m, n);
        int sign = 0;
        for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
            v[i] += e; e = v[i] / BASE; v[i] %= BASE;
            if (v[i]) sign = (v[i] > 0) ? 1 : −1;
        }
        for (int i = n − 1; i > 0; i−−)
            if (v[i] ∗ sign < 0) { v[i] += sign ∗ BASE; v[i+1] −= sign; }
        while (n && !v[n]) n−−;
        return ∗this;
    }
    int cmp(const int a, const int b) const{
```

```
        if(a < b) return -1;
        if(a > b) return 1;
        return 0;
    }

int cmp(const bigint& x = 0) const {
    int i = max(n, x.n), t = 0;
    while (1) if ((t = cmp(v[i], x.v[i])) || i-- == 0) return t;
}
bool operator <(const bigint& x) const { return cmp(x) < 0; }
bool operator ==(const bigint& x) const { return cmp(x) == 0; }
bool operator !=(const bigint& x) const { return cmp(x) != 0; }
operator string() const {
    ostringstream s; s << v[n];
    for (int i = n - 1; i > 0; i--) {
        s.width(DIG); s.fill('0'); s << abs(v[i]);
    }
    return s.str();
}
friend ostream& operator <<(ostream& o, const bigint& x) {
    return o << (string) x;
}
bigint& operator +=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
    return fix(x.n);
}
bigint operator +(const bigint& x) { return bigint(*this) += x; }
bigint& operator -=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
    return fix(x.n);
}
bigint operator -(const bigint& x) { return bigint(*this) -= x; }
bigint operator -() { bigint r = 0; return r -= *this; }
void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
    for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
        v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
    }
}
bigint operator *(const bigint& x) const {
    bigint r;
    for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
    return r;
}
bigint& operator *=(const bigint& x) { return *this = *this * x; }
// cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
bigint div(const bigint& x) {
    if (x == 0) return 0;
    bigint q; q.n = max(n - x.n + 1, 0);
    int d = x.v[x.n] * BASE + x.v[x.n-1];
    for (int i = q.n; i > 0; i--) {
        int j = x.n + i - 1;
        q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
        ams(x, -q.v[i], i-1);
        if (i == 1 || j == 1) break;
        v[j-1] += BASE * v[j]; v[j] = 0;
    }
    fix(x.n); return q.fix();
}
bigint& operator /=(const bigint& x) { return *this = div(x); }
bigint& operator %=(const bigint& x) { div(x); return *this; }
bigint operator /(const bigint& x) { return bigint(*this).div(x); }
bigint operator %(const bigint& x) { return bigint(*this) %= x; }
bigint pow(int x) {
    if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
    bigint r = 1;
    for (int i = 0; i < x; i++) r *= *this;
    return r;
}
bigint root(int x) {
    if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
    if (*this == 1 || x == 1) return *this;
    if (cmp() < 0) return -(-*this).root(x);
    bigint a = 1, d = *this;
    while (d != 1) {
        bigint b = a + (d /= 2);
        if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
    }
    return a;
}
};
```

Fenwick Tree

fenwick.cpp

```
#define MAX_N 1000010

int B1[MAX_N], B2[MAX_N];

void ft_create(int ft[MAX_N], int n){ CLEAR0(ft); }

int ft_query(int ft[MAX_N], int b){
    int sum = 0; while(b){ sum += ft[b]; b -= LSOne(b);}
    return sum;
}

void ft_update(int ft[MAX_N], int k, int v){
    while(k <= MAX_N){ ft[k] += v; k += LSOne(k); }
}

int query(int b) {
  return ft_query(B1, b) * b - ft_query(B2, b);
}

int range_query(int i, int j)  {
  return query(j) - query(i - 1);
```

```
}

void range_update(int i, int j, ll v)   {
        ft_update(B1, i, v);
        ft_update(B1, j + 1, −v);
        ft_update(B2, i, v ∗ (i − 1));
        ft_update(B2, j + 1, −v ∗ j);
}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

void ft_create(vi &t, int n){ t.assign(n+1, 0); }

int ft_rsq(const vi &t, int b){
        int sum = 0; for(; b; b −= CHECKFIRST(b)) sum += t[b];
        return sum;
}

int ft_rsq(const vi &t, int a, int b){
        return ft_rsq(t, b) − (a == 1? 0 : ft_rsq(t, a−1));
}

void ft_adjust(vi &t, int k, int v){
        for(; k  <= (int)t.size(); k += CHECKFIRST(k)) t[k] += v;
}


/∗−−−−−−−−−−−−3D−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−∗/
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;

#define MAX 100
long long tree[MAX+10][MAX+10][MAX+10]; /∗nao usar posicao em 0∗/
int N, M;

long long q(int x, int y, int z) {
    long long sum=0;
    int yy=y, zz = z;
    if (x==0 || y==0 || z == 0) return 0LL;
    while (x) {
        while (y) {
            while (z){
                sum+=tree[x][y][z];
                z−=z & (−z);
            }
            y−=y & (−y);
            z = zz;
        }
        x−=x & (−x);
        y=yy;
    }
    return sum;
}

void update(int x, int y, int z, long long v) {
    int yy=y, zz = z;
    if(x==0 || y==0 || z == 0) return;
    while (x<=MAX) {
        while (y<=MAX) {
            while (z<=MAX){
                tree[x][y][z]+=v;
                z+=z & (−z);
            }
            y+=y & (−y);
            z = zz;
        }
        x+=x & (−x);
        y=yy;
    }
}

long long sum(int r1, int c1, int d1, int r2, int c2, int d2){
    return (q(r2,c2,d2) − q(r1−1,c2,d2) − q(r2,c1−1,d2) + q(r1−1,c1−1,d2) + q(r2,c1−1,d1−1) − q(r2,c2,d1−1) + q(r1−1,c2,d1−1) − q(
}
```

---

3 Programação Dinâmica

Lis Logarítmico

lislog.cpp

```
#define MAX

struct myGreater {
  bool operator()(int A, int B) const {
    return A <= B;
  }
};

int M[MAX+1], P[MAX+1], X[MAX+1];

void print(int a){
        if(P[a] != −1) print(P[a]);
        printf("%d\n", X[a]);
}

void LIS(int n){ //N = NUMERO DE ELEMENTOS DA SEQUENCIA INICIAL

        int A[n+1], len, i, lnis, j;
```

```
    P[0] = −1;
    M[1] = 0;
    for (A[0] = X[0], i = lnis = 1; i < n; i++) { // O(n)

      int *l = upper_bound(A, A + lnis, X[i], myGreater()); // find insertion point, O(log k)
      *l = X[i];
      len = (int)((l − A)+1);
      P[i] = (len > 1)? M[len−1] : −1;
      //cout << "SETANDO P DE " << i << " = " << P[i] << endl;

      if(len >= lnis || X[i] < X[M[len]]){
          M[len] = i;
          lnis = max(lnis, len);
      }
  }

  printf("%d\n", lnis);
  printf("−\n");
  print(M[lnis]);
}
```

Knapsack

knapsack.cpp

```
#include <iostream>
#define MAT(a, b) *(mem + (a) * (cap + 1) + b)
#define PRECO(a) pesos[0][0]
#define PESO(a) pesos[0][1]
#define PRECOK(a) *(pesos + (a) * 2)
#define PESOK(a) *(pesos + (a) * 2 + 1)

using namespace std;

int cap;

int knapsack(int n, int *mem, int *pesos)
{
    int max, maxgeral = 0;
    for(int i = 1; i<n; i++)
    {
        for(int j = 0; j<cap+1; j++)
        {
            max = MAT(i−1, j);
            if(j−PESOK(i) > −1 && MAT(i−1, j−PESOK(i)) != −1)
            {
                if( (MAT(i−1, j−PESOK(i)) + PRECOK(i)) > max )
                    max = MAT(i−1, j−PESOK(i)) + PRECOK(i);
            }
            MAT(i, j) = max;
            if(MAT(i, j) > maxgeral) maxgeral = MAT(i,j);
        }
    }
    return maxgeral;
}

int main()
{
    int n;
    cin >> n;
    while(n)
    {
        int pesos[n][2];
        for(int i = 0; i<n; i++)
        {
            cin >> pesos[i][0] >> pesos[i][1];
        }
        cin >> cap;
        int mem[n][cap+1];
        for(int i = 0; i<n; i++){ mem[i][0] = 0; for(int j = 1; j<cap+1; j++) mem[i][j] = −1; };
        mem[0][PESO(0)] = PRECO(0);

        cout << knapsack(n, &mem[0][0], &pesos[0][0]) << endl;
        cin >> n;
    }
}
```

Edit Distance

editdist.cpp

```
int EditDistance(string word1, string word2)
{
  int i, j, l1, l2, m;
  l1 = word1.length();
  l2 = word2.length();
  vector< vector<int> > t(l1 + 1, vector<int>(l2 + 1));

  for (i = 0; i <= l1; i++)
    t[i][0] = i;
  for (i = 1; i <= l2; i++)
    t[0][i] = i;

  for (i = 1; i <= l1; i++)
  {
    for (j = 1; j <= l2; j++)
    {
      m = min(t[i−1][j], t[i][j−1]) + 1;
      t[i][j] = min(m, t[i−1][j−1] + (word1[i−1] == word2[j−1] ? 0 : 1));
    }
  }
  return t[l1][l2];
}
```

4 Graphs

Maxflow EDMONDS-KARP

fluxo.cpp

```
#define MAXV 110 //COMPLEXIDADE O(V*E^2)

int g[MAXV][MAXV], grau[MAXV], n, m, res[MAXV][MAXV]; //PREENCHER
int mf, f, s, t, p[MAXV];
bitset<MAXV> vis;

void augment(int v, int minEdge, int totalW){
    if(v == s){ f = minEdge; return; }
    else if(p[v] != -1){
        augment(p[v], min(minEdge, res[p[v]][v]), totalW + g[p[v]][v]);
        res[p[v]][v] -= f; //cout << " DIMINUINDO ARESTA " << p[v] << " " << v << " EM " << f << endl;
        res[v][p[v]] += f;
    }
}

int maxflow(){
    mf = 0;
    while(1){
        f = 0;          //cout << "INICIANDO NOVO FLUXO " << endl;
        vis.reset(); vis[s] = true; CLEAR(p, -1);
        int q[MAXV+10], ini, fim;
        ini = fim = 0; q[fim++] = s;
        while(ini != fim){
            int u = q[ini++]; //cout << "visitando vertice " << u << " E T EH " << t << endl;
            if(u == t) break;
            REP(i, grau[u]){
                int v = g[u][i];
                //cout << "tentando ir pra " << v << " no residual, peso eh " << res[u][v] << " VIS EH " << vis[v] << endl;
                if(res[u][v] > 0 && !vis[v]){
                    vis[v] = true; q[fim++] = v; p[v] = u;
                    //cout << " COLOCANDO " << v << " NA FILA " << endl;
                }
            }
        }
        augment(t, INF, 0);
        if(f == 0) break;
        mf += f;
    }
    return mf;
}

int main(){ //EXEMPLO
    scanf("%d", &n);
    if(n == 0) break;
    scanf("%d", &m);
    s = 1;
    t = n;

    CLEAR0(g); CLEAR0(grau); CLEAR0(res);

    int u, v, x;
    REP(i, m){
        scanf("%d%d%d", &u, &v, &x);
        res[u][v] = res[v][u] = x;
        g[u][grau[u]++] = v;
        g[v][grau[v]++] = u;
    }
    printf("%d\n", maxflow());
}
```

Max Card Bip Matching

MCBM.cpp

```
#define MAXV 510

int V;
vii G[MAXV];

int dijkstra(int s, int t){
    vi dist(V, INF); dist[s] = 0;
    priority_queue< ii, vii, greater<ii> > pq; pq.push(mp(0, s));
    while(!pq.empty()){
        ii f = pq.top(); pq.pop();
        int d = f.first, u = f.second;
        if(d > dist[u]) continue;
        REP(j, G[u].size()){
            ii v = G[u][j];
            if(dist[u] + v.second < dist[v.first]){
                dist[v.first] = dist[u] + v.second;
                pq.push(mp(dist[v.first], v.first));
            }
        }
    }
    return dist[t];
}

int mate[MAXV], n;
bitset<MAXV> vis;

//BIPARTITE MATCHING
int aug(int v){
  if(vis[v]) return false;
  vis[v] = true;
  REP(j, grau[v]){
    int w = g[v][j];
    if(mate[w] == -1 || aug(mate[w])){
```

```
            mate[w] = v;
            return 1;
        }
    }
    return 0;
}


int MCBM(){
    CLEAR(mate, -1);
    int mcbm = 0;
    REPP(i, 1, n+1){
        vis.reset();
        mcbm += aug(i);
    }
    return mcbm;
}
```

Heavy Light Decomposition

hld.cpp

```
#define MAX_NODES 30100
#define LOG2_MAX_NODES 15

int chainNo=0,chainHead[MAX_NODES],chainPos[MAX_NODES],chainInd[MAX_NODES],chainSize[MAX_NODES],chainAc[MAX_NODES];
int subsize[MAX_NODES];
int p[MAX_NODES], l[MAX_NODES];
int ac[MAX_NODES][LOG2_MAX_NODES];

void hld(int cur) {
        if(chainHead[chainNo] == -1){
                chainHead[chainNo] = cur;
                chainAc[chainNo] = (chainNo == 0)? 0 : chainAc[chainNo-1] + chainSize[chainNo-1];
        }
    chainInd[cur] = chainNo;
    chainPos[cur] = chainSize[chainNo];
    ft_adjust(ft, chainAc[chainNo] + chainPos[cur]+1, P[cur]);
    chainSize[chainNo]++;

    int ind = -1, mai = -1;
    for(int i = 0; i < g[cur].size(); i++) {
                if(g[cur][i] != p[cur] && subsize[g[cur][i]] > mai) {
            mai = subsize[ g[cur][i] ];
            ind = i;
        }
    }

    if(ind >= 0) hld( g[cur][ind] );

    for(int i = 0; i < g[cur].size(); i++) {
        if(g[cur][i] != p[cur] && i != ind) {
            chainNo++;
            hld( g[cur][i] );
        }
    }
}

int dfs(int v, int P, int L){
        if(p[v]) return 0;
        p[v] = P;
        l[v] = L;
        subsize[v] = 1;
        REP(i, g[v].size()){
                subsize[v] += dfs(g[v][i], v, L+1);
        }
        return subsize[v];
}

void precalc(){
        CLEAR0(ac);
        REPP(i, 1, n+1) ac[i][0] = p[i];
        REPP(j, 1, 15) REPP(i, 1, n+1) if(ac[i][j-1]) ac[i][j] = ac[ac[i][j-1]][j-1];
}

int lca(int u, int v){
        int tmp, log, i;
        if(l[u] < l[v]) swap(u, v);

    for (log = 1; 1 << log <= l[u]; log++);
        log--;

    for (i = log; i >= 0; i--)
          if (l[u] - (1 << i) >= l[v])
              u = ac[u][i];
        if (u == v)
          return u;
        for (i = log; i >= 0; i--)
         if (ac[u][i] && ac[u][i] != ac[v][i])
            u = ac[u][i], v = ac[v][i];
        return p[u];
}
```

2 Sat

2sat.cpp

```
#define MAXCL 2100
#define MAXV 2100

#define N(x) (2*x + 1)
#define Y(x) (2*x)
```

```
#define NEG(x) (x%2 == 1 ? x-1 : x+1) //positive variables has even index and negative variables has odd index

int x[MAXCL], y[MAXCL]; //clauses
int nc, nv;             //number of clauses and number of variables
int n, m;               //number of vertices and edges of the graph
int numSCC;
vi g[4*MAXCL], ng[4*MAXCL];  //implication graph and its complement
stack<int> st;
bitset<4*MAXCL> vis;        //bitset to mark visited vertices

int pos[4*MAXCL], comp[4*MAXCL], idx;
bool val[4*MAXCL];

void dfs(int v){
    vis[v] = true;
    REP(i, g[v].size()) if(!vis[g[v][i]]) dfs(g[v][i]);
    st.push(v);
}

void dfs2(int v){
    vis[v] = true;
    REP(i, ng[v].size()) if(!vis[ng[v][i]]) dfs2(ng[v][i]);
    comp[v] = numSCC;
    pos[v] = idx++;
}

bool twoSat(){
    numSCC = 0;
    vis.reset();
    REP(i, V) if(!vis[i]) dfs(i);
    vis.reset();
    idx = 0;
    while(!st.empty()){
        int v = st.top(); st.pop();
        if(!vis[v]){
            numSCC++;
            dfs2(v);
        }
    }
    REP(i, nv){
        if(comp[Y[i]] == comp[N[i]]) return false;
        val[Y[i]] = (pos[Y[i]] > pos[N[i]]);
        val[N[i]] = !val[Y[i]];
    }
    return true;
}
```