Universidad Peruana de Ciencias Aplicadas

Monaschinas

Dobby, Sebastian, Jack

2023-10-07

Contest (1)

template.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

void solve(){

}

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

    int tc = 1;
    // cin >> tc;
    while(tc--){
        solve();
    }
}
```

.bashrc

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt

```
Wrong answer:
- Formato de salida (falta un 'n' o ' ').
- Debuggear el codigo con couts (sin fast input).
- Checkear si limpia memoria por cada test case.

Runtime error:
- Lectura fuera del vector.
- Division entre 0.

Time limit exceeded:
- Recursion infinita.
- Escribir la complejidad temporal.

Tips:
- Trabajar en equipo (discutir ideas, revisar codigo, etc).
- Tomar un descanzo (ej. ir a los servicios higienicos).
```

Dynamic Prog (2)

2.1 Fundamentals

SumaRango.h

Description: Suma en rangos.
Time: $\mathcal{O}(N)$

```cpp
int a[N];
int dif[N];

int main(){
    cin >> n >> q;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        dif[i] = a[i] - a[i-1];
    }

    while(q--){
        int l, r, x;
        cin >> l >> r >> x;
        dif[l] += x;
        dif[r+1] -= x;
    }
    for(int i = 1; i <= n; i++) a[i] = dif[i] + a[i-1];
    for(int i = 1; i <= n; i++){
        cout << a[i] << " ";
    }
    return 0;
}
```

KnapsackRec.h

Description: Knapsack recursivo.
Time: $\mathcal{O}(N * C)$

```cpp
const int N = 1e2 + 5;  // Limite de "n"
const int W = 1e4 + 5;   // Limite de capacidad
int n, C;
int v[N];
int w[N];

// Memoria O(N*C)
int dp[N][W];
bool vis[N][W];

int solve(int pos, int left){
    if(pos == n) return 0;
    if(vis[pos][left]) return dp[pos][left];

    int ans = solve(pos+1, left);
    if(left >= w[pos]) {
        ans = max(ans, v[pos] + solve(pos+1, left-w[pos]));
    }

    vis[pos][left] = 1;
    return dp[pos][left] = ans;
}

int main(){
    solve(0, C);
}
```

KnapsackIter.h

Description: Knapsack iterativo.
Time: $\mathcal{O}(N * C)$

```cpp
int n, C;
int v[N];
int w[N];

// Memoria O(N)
int solve(){
    vector<int> last(C+1);
    vector<int> dp(C+1);
    for(int pos = n-1; pos >= 0; pos--){
        for(int left = 0; left <= C; left++){

            int ans = last[left];
            if(left >= w[pos]){
                ans = max(ans, v[pos] + last[left - w[pos]]);
            }
            dp[left] = ans;
        }
        last = dp;
    }
    return last[C];
}
int main(){
    solve();
}
```

maxsubarray1D.h

Description: Terminando en 'i'.
Time: $\mathcal{O}(N)$

```cpp
// memo[i]: Maxima suma de arreglo que termina en la posicion 'i'.
// memo[i]: Toma solo a[i] o (..., a[i-1], a[i])
// memo[i]: max(a[i], a[i] + memo[i-1])
// memo[i]: a[i] + max(0, memo[i-1])
const int N = 1e5 + 5;
const int inf = 1e9;

int n;
int a[N];
int memo[N];

int solve(){
    for(int i = 0; i < n; i++){
        if(i == 0) {
            memo[i] = a[i];
        } else {
            memo[i] = max(a[i], a[i] + memo[i-1]);
        }
    }
    return *max_element(memo, memo+n);
}
```

maxsubarray1Dprefix.h
Description: Diferencia de prfijos
Time: $\mathcal{O}(N)$
db4a4e, 31 lines

```
// S[i] = a[1] + ... + a[i]
// suma(l, r) = S[r] - S[l-1]

// F[i]: Maxima suma de arreglo que termina en pos 'i'
// F[i]: max(S[r] - S[l-1]) donde 0 <= l < r
// F[i]: S[r] + max(-S[l-1]) donde S[r] se itera y max(-S[l-1])
      se procesa en marcha

// S[r]: prefix
// S[l-1]: bestprefix
const int N = 1e5 + 5;
const int inf = 1e9;

int n;
int a[N];
int memo[N];

int solve(){
    ll prefix = 0;
    ll mini = 0;
    ll ans = -inf;
    for(int i = 0; i < n; i++){
        prefix += a[i];
        if(ans < prefix - mini){
            ans = prefix - mini;
        }
        if(mini > prefix){   // bestprefix para i+1
            mini = prefix;
        }
    }
    return ans;
}
```

maxsubarray2D.h
Description: S[i][j]: Suma hasta fila i, columna j.
Time: $\mathcal{O}(N)$
769e50, 30 lines

```
ll S[N][N];

void solve(){
    // preprocesar
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            int suma = a[i][j];
            if(i) suma += S[i-1][j];
            if(j) suma += S[i][j-1];
            if(i and j) suma -= S[i-1][j-1];
            S[i][j] = suma;
        }
    }
    // suma en 2D
    int ans = INT_MIN;
    for(int r1 = 0; r1 < n; r1++){
        for(int r2 = r1; r2 < n; r2++){
            for(int c1 = 0; c1 < n; c1++){
                for(int c2 = c1; c2 < n; c2++){
                    int suma = S[r2][c2];
                    if(r1) suma -= S[r1-1][c2];
                    if(c1) suma -= S[r2][c1-1];
                    if(r1 and c1) suma += S[r1-1][c1-1];
                    ans = max(ans, suma);
                }
            }
        }
    }
    cout << ans;
}
```

maxsubarray2Dprefix.h
Description: F[i][j]: Suma hasta fila i de la columna j-
Time: $\mathcal{O}(N^3)$
4e693e, 28 lines

```
ll F[N][N];

void solve(){
    // preprocesar
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            F[i][j] = a[i][j];
            if(i) F[i][j] += F[i-1][j];
        }
    }
    // suma en 2D
    int ans = INT_MIN;
    for(int r1 = 0; r1 < n; r1++){
        for(int r2 = r1; r2 < n; r2++){
            int prefix = 0, mini = 0;
            for(int i = 0; i < n; i++){
                int col = F[r2][i];
                if(r1){
                    col -= F[r1-1][i];
                }
                prefix += col;
                if(ans < prefix - mini) ans = prefix - mini;
                if(mini > prefix) mini = prefix;
            }
        }
    }
    cout << ans;
}
```

SumaStaticQuery.h
Description: Query en un array estático.
Time: $\mathcal{O}(N)$
6186d8, 14 lines

```
int main(){
    for(int i = 0; i < n; i++){
        cin >> a[i];
        a[i] += (i-1 >= 0 ? a[i-1] : 0);
    }
    while(q--){
        int l, r;
        cin >> l >> r;
        r--;
        ll suma = a[r] - (l-1 >= 0 ? a[l-1] : 0);
        cout << suma << "\n";
    }
    return 0;
}
```

2.2 Subsequence

LCS.h
Description: Longest Common Subsequence
Time: $\mathcal{O}(N * C)$
300365, 12 lines

```
int solve(){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            int ans = 0;
            // Cuidado con memo [-1]
            if(s[i-1] == t[j-1]) ans = max(ans, 1 + memo[i-1][j
                -1]);
            else  ans = max(memo[i-1][j], memo[i][j-1]);
            memo[i][j] = ans;
        }
    }
    return memo[n][m];
}
```

LCSfind.h
Description: Find the path.
Time: $\mathcal{O}(N^2)$
4c3fb7, 33 lines

```
const int N = 100, M = 100, inf = 1e9;
int memo[N][N];

string solve(){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            int cur = 0;
            // Cuidado con memo [-1]
            if(s[i-1] == t[j-1]) cur = max(cur, 1 + memo[i-1][j
                -1]);
            else  cur = max(memo[i-1][j], memo[i][j-1]);

            memo[i][j] = cur;
        }
    }

    string ans = "";
    int i = n, j = m;
    while(i > 0 and j > 0){
        if(s[i-1] == t[j-1]){
            ans.push_back(s[i-1]);
            i -= 1;
            j -= 1;
        } else {
            if(memo[i-1][j] > memo[i][j-1]){
                i -= 1;
            } else {
                j -= 1;
            }
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

LCSfollow.h
Description: Follow the path.
Time: $\mathcal{O}(N^2)$
ffa3d3, 41 lines

```
const int N = 100, M = 100, inf = 1e9;

int memo[N][N];
int choice[N][N];

string solve(){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= m; j++){
            // Cuidado con memo [-1]
            if(s[i-1] == t[j-1]) {
                memo[i][j] = 1 + memo[i-1][j-1];
                choice[i][j] = 0;
            }
            else {
                if(memo[i-1][j] > memo[i][j-1]){
                    memo[i][j] = memo[i-1][j];
                    choice[i][j] = 1;
                } else {
                    memo[i][j] = memo[i][j-1];
                    choice[i][j] = 2;
                }
            }
```

```
        }
    }

    string ans = "";
    int i = n, j = m;
    while(i > 0 and j > 0){
        if(choice[i][j] == 0){
            ans.push_back(s[i-1]);
            i -= 1;
            j -= 1;
        } else if(choice[i][j] == 1){
            i -= 1;
        } else {
            j -= 1;
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

LIS.h

Description: Longest Increasing Subsequence
Time: $\mathcal{O}(N * logN)$

ca343f, 41 lines

```
const int N = 100;
int n;
int a[N];
vector<int> vec;

void add(int left, int right, int val){
    if(left == right){
        if(vec[left] >= val){
            vec[left] = val;
        } else {
            vec.emplace_back(val);
        }
        return;
    }
    int mid = (right - left)/2;

    if(val <= vec[mid]){
        add(left, mid, val);
    } else {
        add(mid+1, right, val);
    }
}
void imprime_vector(){
    for(int i = 0; i < vec.size(); i++){
        cout << vec[i] << " ";
    }
    cout << "\n";
}
int main(){
    scanf("%d", &n);
    for(int i = 0; i < n; i++) cin >> a[i];
    vec.push_back(a[0]);    // inicial

    for(int i = 1; i < n; i++){
        add(0, vec.size()-1, a[i]);
        imprime_vector();
    }

    cout << vec.size() << "\n";
    return 0;
}
```

Data structures (3)

SegmentTree.h

Description: Estructura de consulta y actualizacion en rangos. ST indexado en 0, pero 'pos' inicia en 1.
Time: $\mathcal{O}(\log N)$

4fd1d0, 35 lines

```
const int N = 1e5 + 5;
const int MAX = 4*N;
ll st[MAX];

void build(int pos=1, int l=0, int r=n-1){
    if(l == r){
        st[pos] = a[l];
    }
    int mi = (l+r)/2;
    build(2*pos, l, mi);
    build(2*pos+1, mi+1, r);
    st[pos] = st[2*pos] + st[2*pos+1];
}

// x, y (query)
// l, r (subarray)
ll query(int x, int y, int pos, int l, int r){
    if(y < l or x > r) return 0;
    if(l >= x and r <= y) return st[pos];
    int mi = (l+r)/2;
    ll left = query(x, y, 2*pos, l, mi);
    ll right = query(x, y, 2*pos+1, mi+1, r);
    return left + right;
}

void update(int x, int y, int pos, int l, int r){
    if(l == r){
        st[pos] = y;
        return;
    }
    int mi = (l+r)/2;
    if(x <= mi) update(x, y, 2*pos, l, mi);
    else update(x, y, 2*pos+1, mi+1, r);
    st[pos] = st[2*pos] + st[2*pos+1];
}
```

FenwickTree.h

Description: Estructura de prefijo que permite modificaciones y consulta en O(log N).
Time: $\mathcal{O}(\log N)$

e2642f, 18 lines

```
const int N = 1000000 + 5;
ll ft[N];

void update(int pos, int val){
    while(pos <= MAXI){
        ft[pos] += val;
        pos += (pos & -pos);
    }
}

ll query(int pos){
    ll res = 0;
    while(pos > 0){
        res += ft[pos];
        pos -= (pos & -pos);
    }
    return res;
}
```

MinQueue.h

Description: Estructura de consulta en O(1) amortizado y actualización O(1).
Time: $\mathcal{O}(\log N)$

5a7abe, 54 lines

```
struct MinStack {
    stack<pair<int, int>> S;
    void push(int x){
        S.push(make_pair(x, min(x, S.empty() ? INT_MAX : S.top
            ().second)));
    }
    void pop(){
        if(!S.empty()){
            S.pop();
        }
    }
    bool empty(){
        return S.empty();
    }
    int top(){
        return S.empty() ? -1 : S.top().first;
    }
    int get_min(){
        return S.empty() ? -1 : S.top().second;
    }
};

struct MinQueue{
    MinStack in, out;
    void push(int x){
        in.push(x);
    }
    void pop(){
        if(out.empty()){
            while(!in.empty()){
                out.push(in.top());
                in.pop();
            }
        }
        if(!out.empty()){
            out.pop();
        }
    }
    bool empty(){
        return in.empty() and out.empty();
    }
    int front(){
        if(out.empty()){
            while(!in.empty()){
                out.push(in.top());
                in.pop();
            }
        }
        return out.empty() ? -1 : out.top();
    }
    int get_min(){
        return min(in.empty() ? INT_MAX : in.get_min(),
                out.empty() ? INT_MAX : out.get_min());
    }
} minqueue;
```

SQRT.h

Description: Estructura de consulta y actualizacion en raices.
Time: $\mathcal{O}\left(\sqrt{N}\right)$

912018, 38 lines

```
// N,Q<= 1e5
const int N = 1e5 + 5;
const int SQRT = 450;

int n, q;
int a[N];
int bsize;
ll sum[SQRT];
```

```
void build(){
    for(int i = 0; i < n; i++){
        sum[i / bsize] += a[i];
    }
}
ll query(int l, int r) {
    ll res = 0;

    while(l % bsize != 0 and l <= r){
        res += a[l];
        l += 1;
    }
    // l + bsize - 1 <= r
    while(l + bsize - 1 <= r){
        res += sum[l/bsize];
        l += bsize;
    }
    while(l <= r){
        res += a[l];
        l += 1;
    }
    return res;
}
void update(int pos, int val){
    int idx = pos / bsize;
    sum[idx] -= a[pos];
    sum[idx] += val;
    a[pos] = val;
}
```

Number theory (4)

Primo.h

Description: Verifica si un numero es primo.
Time: $\mathcal{O}(logN)$

2f50f9, 8 lines

```
bool primo(int n){
    for(int i = 2; i*i <= n; i++){
        if(n % i == 0){
            return 0;
        }
    }
    return !;
}
```

FactoresPrimos.h

Description: Encuentra los factores primos.
Time: $\mathcal{O}\left(\sqrt{N}\right)$

7ade7a, 13 lines

```
void solve(){
    for(int i = 2; 1ll*i*i <= n; i++){
        if(n % i == 0){
            while(n % i == 0){
                cout << i << " ";
                n /= i;
            }
        }
    }
    if(n > 1){
        cout << n << "\n";
    }
}
```

Tdprimes.h

Description: Encuentra los factores primos.
Time: $\mathcal{O}(logN)$

0bdcc4, 17 lines

```
const int N = 100000000 + 1;

vector<int> primes;
bitset<N> composite;

int main() {
  for(int i = 2; i < N; i++) {
    if(not composite[i]) primes.emplace_back(i);
    for(int p : primes) {
      if(i * p >= N) break;
      composite[i * p] = true;
      if(i % p == 0) break;
    }
  }
  for(int i = 0; i < primes.size(); i += 100) printf("%d\n",
       primes[i]);
  return 0;
}
```

Criba.h

Description: Criba para hallar numeros primos.
Time: $\mathcal{O}(N)$

3be3f3, 11 lines

```
void solve(int n){
    vector<int> prime(n+1, true);

    for(int i = 2; i <= n; i++){
        if(prime[i]){
            for(int j = i*2; j <= n; j += i){
                prime[j] = false;
            }
        }
    }
}
```

Euclides.h

Description: Euclides para hallar mcd (gcd).
Time: $\mathcal{O}(logN)$

21d9fc, 4 lines

```
int gcd(int a, int b){
    if(a == 0) return b;
    return gcd(b % a, a);
}
```

EuclidesExtended.h

Description: Euclides extendido ax + by = gcd(a, b)
Time: $\mathcal{O}(logN)$

343534, 20 lines

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

    *x = y1 - (b/a)*x1;
    *y = x1;
    return gcd;
}

int main(){
    int x, y;
    int gcd = gcdExtended(a, b, &x, &y);
    cout << x << y << gcd;
}
```

Graph (5)

5.1 Fundamentals

UniversalSink.h

Description: Find universal sink interactivo.
Time: $\mathcal{O}(3 * N)$

23ce8d, 43 lines

```
int main(){
    int n;
    cin >> n;

    int j = 0;
    for(int i = 1; i < n; i++){
        cout << "? " << i+1 << " " << j+1 << "\n";
        fflush(stdout);
        bool e;
        cin >> e;
        if(e){
            continue;
        } else {
            j = i;
        }
    }
    int row = 0;
    for(int i = 0; i < n; i++){
        cout << "? " << i+1 << " " << j+1 << "\n";
        fflush(stdout);

        bool e;
        cin >> e;
        row += e;
    }
    int col = 0;
    for(int i = 0; i < n; i++){
        cout << "? " << j+1 << " " << i+1 << "\n";
        fflush(stdout);

        bool e;
        cin >> e;
        col += e;
    }
    if(row == n-1 && col == 0){
        cout << "! " << j+1 << "\n";
        fflush(stdout);
    } else {
        cout << "! " << -1 << "\n";
        fflush(stdout);
    }
    return 0;
}
```

NumberPaths.h

Description: Encuentra el número de caminos con longitud l, para ello usa binary exponentation.
Time: $\mathcal{O}(N * N * logN)$

1f59af, 54 lines

```
const int MOD = 1e9 + 7;

typedef vector<int> vi;
typedef vector<vector<int>> vvi;

vvi multiply(vvi a, vvi b, int n){
    vvi c (n, vi(n));
    // multiply
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            for(int k = 0; k < n; k++){
```

```
                c[i][j] = (c[i][j] + 1ll * a[i][k] * b[k][j]) %
                     MOD;
            }
        }
    }
    return c;
}

vvi exp_power(vvi a, int n, int l){
    vvi result (n, vi(n));
    // identity matrix
    for(int i = 0; i < n; i++)  result[i][i] = 1;

    while(l > 0){
        if(l & 1){
            result = multiply(result, a, n);
        }
        l /= 2;
        a = multiply(a, a, n);
    }
    return result;
}

int main(){
    int n, l;
    cin >> n >> l;
    vvi a(n, vi(n));

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cin >> a[i][j];
        }
    }
    vvi al = exp_power(a, n, l);
    // cout << "result matrix \n";
    // for(int i = 0; i < n; i++){
    //     for(int j = 0; j < n; j++){
    //         cout << al[i][j] << " ";
    //     }
    //     cout << "\n";
    // }
    cout << al[0][n-1];
    return 0;
}
```

BFS.h

Description: Encuentra el camino màs corto de un nodo hacia los demas. Todas las aristas tienen el mismo peso..
Time: $\mathcal{O}(|V| + |E|)$

3be3b8, 45 lines

```
const int N = 1e2 + 5;
const int inf = 1e9 + 5;

int n;
vector<int> G[N];
int dis[N];
bool vis[N];

void bfs(int s){
    for(int u = 0; u < n; u++) dis[u] = inf;
    queue<int> Q;
    dis[s] = 0;
    vis[s] = 1;
    Q.push(s);
    while (!Q.empty()){
        int u = Q.front();
        Q.pop();
        for(int v : G[u]){
            if(vis[v]) continue;
            dis[v] = dis[u] + 1;
            vis[v] = 1;
            Q.push(v);
        }
    }
}

int main(){
    cin >> n;
    for(int i = 0; i < n; i++){
        int u, k;
        cin >> u >> k;
        u--;
        for(int j = 0; j < k; j++){
            int v;
            cin >> v;
            v--;
            G[u].push_back(v);
        }
    }
    bfs(0);
    for(int i = 0; i < n; i++){
        cout << i+1 << " " << (dis[i] == inf ? -1 : dis[i]) <<
             "\n";
    }
    return 0;
}
```

BFSfrontier.h

Description: Encuentra el camino màs corto de un nodo hacia los demas. Todas las aristas tienen el mismo peso.
Time: $\mathcal{O}(|V| + |E|)$

120650, 26 lines

```
const int inf = 1e9 + 10;
const int N = 1e5 + 5;

vector<int> G[N];
bool vis[N];
int dis[N];

void bfs(int s){
    for(int i = 0; i < n; i++) dis[s] = inf;
    vector<int> front = {s};
    dis[s] = 0;
    vis[s] = 1;

    while (!front.empty()){
        vector<int> cur;
        for(int u : front){
            for(int v : G[u]){
                if(vis[v]) continue;
                dis[v] = dis[u] + 1;
                vis[v] = 1;
                cur.push_back(v);
            }
        }
        swap(cur, front);
    }
}
```

Dijkstra.h

Description: Distancia más corta de un nodo hacia los demas. No acepta aristas negativas.
Time: $\mathcal{O}(E * logE)$

f166fe, 42 lines

```
int n, m;
vector<pair<int, int>> G[N];

void Dijsktra(int s){
    // O(ElogE)
    vector<ll> d(n, inf);
    d[s] = 0;

    priority_queue<pair<ll, int>, vector<pair<ll, int>>,
        greater<pair<ll, int>>> Q;
    Q.emplace(make_pair(0, s));

    while(!Q.empty()){
        int u;
        ll dis;
        tie(dis, u) = Q.top();
        Q.pop();
        if(dis != d[u]) continue;

        //for(auto [v, w] : G[u]){ // Feature in C++ 17
        for(auto pair : G[u]){
            int v;
            ll w;
            tie(v, w) = pair;

            if(d[v] > d[u]+  w){
                d[v] = d[u] + w;
                Q.emplace(make_pair(d[v], v));
            }
        }
    }
    for(int i = 0; i < n; i++){
        printf("%lld%c", d[i], " \n"[i+1 == n]);
    }
}

int main(){
    // Inout
    G[u].emplace_back(make_pair(v, w));
    G[v].emplace_back(make_pair(u, w));
    // Process
    Dijsktra(0);
}
```

BellmanFord.h

Description: Calcula el camino más corto de s en un grafo, este puede tener aristas negativas, pero no puede tener ciclos negativos.
Time: $\mathcal{O}(V * E)$

340af8, 32 lines

```
const int N = 100000 + 5;
const long long  inf = 1e18 + 10;
int n, m;
long long d[N];
vector<tuple <int, int, int>> edges;

bool Bellman(int s){
    for(int i = 0; i < n; i++) d[i] = inf * (i != s);
    int last = 0;

    for(int i = 1; i <= n; i++){
        bool relaxed = false;
        for(auto e : edges){
            int u, v, w;
            tie(u, v, w) = e;
            if(d[u] < inf and d[v] > d[u] + w){
                d[v] = d[u] + w;
                relaxed = true;
            }
        }
        if(not relaxed) break;
        last = i;
    }
```

```
    return last < n;
}

int main(){
    edges.emplace_back(make_tuple(u, v, w));
    Bellman(0);
    return 0;
}
```

FloydWarshall.h

Description: Calcula todos los caminos más cortos en un grafo dirigido que puede tener aristas de peso negativo. La entrada es una matriz $n * n$ donde indica los pesos de las aristas, por otro lado, sino existe tiene valor inf.
Time: $\mathcal{O}(N^3)$

a0559c, 37 lines

```
const int N = 1e2 + 5;
const int inf = 2e9 + 10;
int n, m;
int d[N][N];

int main(){
    cin >> n >> m;
    // preprocess
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            d[i][j] = i == j ? 0 : inf;
        }
    }
    // input
    for(int i = 0; i < m; i++){
        int u, v, w;
        cin >> u >> v >> w;
        d[u][v] = min(d[u][v], w);
    }
    // Floyd warshall
    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                if(d[i][k] == inf or d[k][j] == inf) continue;
                if(d[i][j] > d[i][k] + d[k][j]) d[i][j] = d[i][
                    k] + d[k][j];
            }
        }
    }
    // Find negative cycle
    for(int i = 0; i < n; i++){
        if(d[i][i] < 0){
            cout << "NEGATIVE CYCLE";
            return 0;
        }
    }
    return 0;
}
```

Kruskal.h

Description: Calcula el árbol de expansión minima (MST), usa la optimización en rango (DSU).
Time: $\mathcal{O}(E * logE)$

fbe3d7, 56 lines

```
const int N = 200000 + 5;
int n, m;
int comp[N];
vector<int> nodes[N];

void init() {
  for(int i = 1; i <= n; i++) {
    nodes[i].push_back(i);
    comp[i] = i;
  }
}

void join(int u, int v) {
  u = comp[u];
  v = comp[v];
  if(nodes[u].size() > nodes[v].size()) swap(u, v);
  while(!nodes[u].empty()) {
    int x = nodes[u].back();
    nodes[u].pop_back();
    nodes[v].emplace_back(x);
    comp[x] = v;
  }
}

int get_component(int x) {
  return comp[x];
}

int main() {
    cin >> n >> m;
  vector<tuple<int, int, int>> edges;
  for(int i = 0; i < m; i++) {
    int u, v, w;
        cin >> u >> v >> w;
    edges.push_back({w, u, v});
  }
  sort(edges.begin(), edges.end());
  init();
  int ans = 0;
  vector<pair<int, int>> mst;
  for(auto e : edges) {
    int w, u, v;
    tie(w, u, v) = e;
    if(get_component(u) != get_component(v)) {
      ans += w;
      mst.emplace_back(u, v);
      join(u, v);
    }
  }
    cout << ans << "\n";
    cout << n-1 << "\n";
  for(auto e : mst) {
    cout << e.first << " " << e.second << "\n";
  }
  return 0;
}
```

Boruvka.h

Description: Calcula el árbol de expansión minima (MST), usa la optimización en rango (DSU).
Time: $\mathcal{O}(E * logE)$

d6a6f8, 72 lines

```
int n, m;
int comp[N];
vector<pair<int, int>> G[N];
vector<int> nodes[N];

void init() {
  for(int i = 1; i <= n; i++) {
    nodes[i].emplace_back(i);
    comp[i] = i;
  }
}

void join(int u, int v) {
  u = comp[u];
  v = comp[v];
  if(nodes[u].size() > nodes[v].size()) swap(u, v);
  while(!nodes[u].empty()) {
    int x = nodes[u].back();
    nodes[u].pop_back();
    nodes[v].emplace_back(x);
    comp[x] = v;
  }
}

int main() {
  scanf("%d %d", &n, &m);
  for(int i = 0; i < m; i++) {
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    G[u].emplace_back(v, w);
    G[v].emplace_back(u, w);
  }
  init();
  int ans = 0;
  vector<pair<int, int>> mst;
  int comps = n;
  while(comps > 1) {
    vector<tuple<int, int, int>> L;
    for(int i = 1; i <= n; i++) {
      if(nodes[i].empty()) continue;
      int best_cost = INT_MAX;
      tuple<int, int, int> light_edge;
      for(auto u : nodes[i]) {
        for(auto e : G[u]) {
          int v, w;
          tie(v, w) = e;
          if(comp[u] == comp[v]) continue;
          if(w < best_cost) {
            best_cost = w;
            light_edge = make_tuple(u, v, w);
          }
        }
      }
      L.emplace_back(light_edge);
    }
    for(auto e : L) {
      int u, v, w;
      tie(u, v, w) = e;
      if(comp[u] == comp[v]) continue;
      comps -= 1;
      join(u, v);
      ans += w;
      mst.emplace_back(u, v);
    }
  }
  printf("%d\n", ans);
  printf("%d\n", n - 1);
  for(auto e : mst) {
    printf("%d %d\n", e.first, e.second);
  }
  return 0;
}
```

5.2 DFS algorithms

DFS.h

Time: $\mathcal{O}(|V| + |E|)$

706c0e, 36 lines

```
#pragma once
 const int N = 1e5 + 5;

int n, m;
vector<int> G[N];
int dis[N];
int vis[N];
int par[N];
```

```
int timer = 1;
int in[N], out[N];

void dfs_visit(int u, int p){
    vis[u] = 1;
    par[u] = p;
    in[u] = timer++;

    for(int v : G[u]){
        if(vis[v] == 2) continue;
        else if(vis[v] == 1){
            // hay un ciclo en el grafo, no llamar al dfs_visit
                (v)
            continue;
        } else {
            dfs_visit(v, u);
        }
    }
    out[u] = timer++;
    vis[u] = 2;
}

void dfs(){
    for(int u = 0; u < n; u++){
        if(vis[u]) continue;
        dfs_visit(u, -1);
    }
}
```

TopoSortBFS.h

Description: Ordena los vertices de manera que las arista van de izquierda a derecha. Si existen ciclos retorna un vector vacio.

Time: $\mathcal{O}(|V| + |E|)$

f91a1e, 33 lines

```
int n, m;
int in[N];
vector<int> G[N];

vector<int> toposort(){
    queue<int> Q;
    vector<int> order;

    for(int u = 0; u < n; u++){
        if(in[u] == 0){
            Q.push(u);
        }
    }
    while(!Q.empty()){
        int u = Q.front();
        Q.pop();
        order.push_back(u);
        for(int v : G[u]){
            in[v]--;
            if(in[v] == 0){
                Q.push(v);
            }
        }
    }

    // deteccion de un ciclo
    return order.size() < n ? vector<int>() : order;
}

int main(){
    // input
    in[v]++;
}
```

TopoSortDFS.h

Description: Ordena los vertices de manera que las arista van de izquierda a derecha. Si existen ciclos retorna un vector vacio.

Time: $\mathcal{O}(|V| + |E|)$

4e868b, 30 lines

```
bool cycle;
int vis[N];
vector<int> G[N];
vector<int> order;

void DFS(int u){
    vis[u] = 1;
    // in[u] = timer++;
    for(int v : G[u]){
        if(vis[v] == 2) continue;
        if(vis[v] == 1) {
            cycle = true;
            continue;
        }
        DFS(v);
    }

    vis[u] = 2;
    order.emplace_back(u);
}

vector<int> topological_sort(){
    for(int i = 1; i <= n; i++){
        if(vis[i]) continue;
        DFS(i);
    }
    if(cycle) order.clear();
    reverse(order.begin(), order.end());
    return order;
}
```

SCC.h

Description: Encuentra las componentes fuertemente conectadas. Cabe resaltar que el algoritmos encuentra los componentes ordenados topologicamente.

Time: $\mathcal{O}(E + V)$

5dc184, 45 lines

```
vector<int> G[2][N];
bool vis[N];
stack<int> S;
vector<int> comp;

void dfs(int u, int id){
    vis[u] = 1;
    for(int v : G[id][u]){
        if(vis[v]) continue;
        dfs(v, id);
    }
    if(id == 0) S.push(u);
    else comp.push_back(u);
}

void get_scc(){
    for(int i = 0; i < n; i++){
        if(vis[i]) continue;
        dfs(i, 0);
    }
    for(int i = 0; i < n; i++) vis[i] = 0;

    vector<vector<int>> res;
    while(!S.empty()){
        int i = S.top();
        S.pop();
        comp.clear();
        if(vis[i]) continue;
        dfs(i, 1);
        res.push_back(comp);
    }
    cout << res.size() << "\n";
    for(auto grupo : res){
        cout << grupo.size() << " ";
        for(int x : grupo) {
            cout << x << " ";
        }
        cout << "\n";
    }
}

int main(){
    G[0][u].push_back(v);
    G[1][v].push_back(u);
}
```

Bridges.h

Description: Encuentra las aristas puente o bridges, es decir, si se elimina la arista el grafo se divide.

Time: $\mathcal{O}(|V| + |E|)$

b1683e, 45 lines

```
int n, m;
vector<int> G[N];
bool vis[N];

int timer;
int in[N];
int low[N];
vector<pair<int, int>> bridges;

void dfs(int u, int p){
    vis[u] = 1;
    in[u] = timer++;
    low[u] = n+1;
    for(int v : G[u]){
        if(v == p) continue;
        if(vis[v] == 1){
            low[u] = min(low[u], in[v]);
        }
        else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if(low[v] > in[u]){
                if(u < v) bridges.push_back({u, v});
                else bridges.push_back({v, u});
            }
        }
    }
}
int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int u, v;
        cin >> u >> v;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    dfs(0, -1);
    sort(bridges.begin(), bridges.end());
    for(auto e : bridges){
        cout << e.first << " " << e.second << "\n";
    }
    return 0;
}
```

Cutpoints.h

Description: Encuentra los nodos o cutpoints, es decir, si se elimina el nodo el grafo se divide.
Time: $\mathcal{O}(|V| + |E|)$ f2124a, 61 lines

```
const int N = 1e5;

int n, m;
vector<int> G[N];
bool vis[N];

int timer;
int in[N];
int low[N];

void is_cutpoint(int u){
    cout << u << "\n";
}

void dfs(int u, int p = -1){
    vis[u] = 1;
    in[u] = timer++;
    low[u] = n + 1;
    int children = 0;

    for(int v : G[u]){
        if(v == p) continue; // procesado or ignorar el padre
                                                // (u, v) -> (v, u)
        if(vis[v] == 1){
            // si es backedge, aporta a low[u]
            low[u] = min(low[u], in[v]);
        }
        else{
            // aun no procesado (tree edge)
            // verificar si u es cut point
            dfs(v, u);
            low[u] = min(low[u], low[v]);

            // Verificar para nodos diferentes a la raiz
            if(p != -1 and low[v] >= in[u]){
                is_cutpoint(u);
            }
            children++;
        }
    }

    // Si 'u' es raiz, es articulacion si tiene 2 hijos a mas
    if(p == -1 and children > 1) is_cutpoint(u);
}

void dfs_visit(){
    for(int i = 0; i < n; i++){
        if(vis[i] == 0){
            dfs(i, -1);
        }
    }
}

int main(){
    // input
    G[u].push_back(v);
    G[v].push_back(u);

    // process
    dfs_visit();
}
```

2sat.h

Description: Calcula una expresión booleana a variabels a, b, c,... de un 2SAT problem, tal que la expresión es verdadera, o que la expresión no tiene solución.
Time: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of clauses. 5e3f6d, 120 lines

```
struct SATSolver {
    // Assumes that nodes are 0 indexed
    int n, m;
    vector<bool> vis;
    vector<int> comp;
    vector<int> order;
    vector<int> component;
    vector<vector<int>> G, Gt;

    SATSolver(int n, int m) : n(n), m(m){
        // x_i = 2i
        // ~x_i = 2i+1
        comp.resize(2*n);
        vis.resize(2*n, false);
        G.resize(2*n, vector<int>());
        Gt.resize(2*n, vector<int>());
    }

    void add_edge(int u, int v){
        // u or v
        G[u^1].push_back(v);
        G[v^1].push_back(u);
        Gt[v].push_back(u^1);
        Gt[u].push_back(v^1);
    }

    void dfs1(int u){
        vis[u] = 1;
        for(int v : G[u]){
            if(!vis[v]){
                dfs1(v);
            }
        }
        order.push_back(u);
    }
    void dfs2(int u){
        vis[u] = 1;
        for(int v : Gt[u]){
            if(!vis[v]){
                dfs2(v);
            }
        }
        component.push_back(u);
    }
    void get_scc(){
        for(int i = 0; i < 2*n; i++){
            if(!vis[i]){
                dfs1(i);
            }
        }
        reverse(order.begin(), order.end());
        fill(vis.begin(), vis.end(), false);

        int id = 0;
        for(int u : order){
            if(!vis[u]){
                component.clear();
                dfs2(u);
                for(int x : component){
                    comp[x] = id;
                }
                id++;
            }
        }
    }
    vector<int> solve(){
        vector<int> res(n);
        get_scc();
        for(int i = 0; i < n; i++){
            int val = 2*i;
            if(comp[val] == comp[val^1]) return vector<int>();
            if(comp[val] < comp[val^1]) res[i] = 0;
            else res[i] = 1;
        }
        return res;
    }
};

int main(){
    int n, m;
    cin >> n >> m;
    SATSolver Solver(n, m);
    for(int i = 0; i < m; i++){
        // Leemos nodos 1-indexed
        // x_i = i
        // ~x_i = -i
        int u, v;
        cin >> u >> v;
        if(u < 0){
            u = -u;
            u--;
            u = 2*u+1;
        }
        else {
            u--;
            u = 2*u;
        }
        if(v < 0){
            v = -v;
            v--;
            v = 2*v+1;
        }
        else {
            v--;
            v = 2*v;
        }
        Solver.add_edge(u, v);
    }
    vector<int> res = Solver.solve();
    if(res.empty()){
        cout << "There is no solution" << "\n";
    } else {
        cout << "One possible solution is: " << "\n";
        for(int x : res){
            cout << x << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

5.3 Sparse table tree

LCA.h

Description: Estructura de datos que calcula el lowest common ancestor en un arbol (raiz en 0). Usa sparse table.
Time: $\mathcal{O}(N \log N + Q)$ da2830, 54 lines

```
const int N = 1e4 + 5;
const int LOG = 14;
int n, q;
int a[N];
```

```
int h[N];
vector<int> G[N];
int ST[N][LOG];

void compute(int u, int p){
    ST[u][0] = p;
    for(int d = 1; 1 << d <= h[u]; d++){
        int y = ST[u][d-1];
        ST[u][d] = ST[y][d-1];
    }
}
void dfs(int u, int p = -1){
    compute(u, p);
    for(int v : G[u]){
        h[v] = h[u] + 1;
        dfs(v, u);
    }
}
void go_up(int &a, int d){
    while(d){
        int k = __builtin_ctz(d);
        a = ST[a][k];
        d &= d - 1;
    }
}
int lca(int u, int v){
    if(h[u] < h[v]) swap(u, v);
    // h[u] >= h[v]
    go_up(u, h[u] - h[v]);  // Nos movemos h[u] - h[v] aristas
         hacia arriba
    if(u == v) return u;
    for(int i = 31 - __builtin_clz(h[u]); i >= 0; i--){
        if((1 << i) > h[u]) continue;
        if(ST[u][i] != ST[v][i]){
            u = ST[u][i];
            v = ST[v][i];
        }
    }
    return ST[u][0];
}

int main(){
    // input
    G[u].push_back(v);
    // preprocess
    dfs(0);
    // querys
    while(q--){
        lca(u, v);
    }
}
```

Strings (6)

Hashing.h

c1a601, 67 lines

```
ll m0 = 1e9+7, m1=1e9+9;
ll base = 37;
const int LIM = 1e5+5;
int n;
char a[LIM];
char b[LIM];
ll ha [LIM][2], hb[LIM][2], pot[LIM][2];

struct hashingiunc {
    size_t operator()(const pair<ll,ll>& x) coast { return x.
         first;//this is good enough
    }
};

ll subhash(ll hashtab[LIM][2], it i, int k, int part, ll mod){
    return ((hashtab[i+k-1][part]-hashtab[i-1][part]*pot[k][
         part])%mod+mod)%mod;
}

int check(int k){
    if(k==0) return true;
    unordered_set<pair<ll, ll>, hashing_func> sa(n-k+1);
    for(int i=1; i+k-1<=n; ++i){
        sa.emplace(subhash(ha, i, k, 0, m0),
                   subhash(ha, i, k, 1, m1));
    }
    for(int i=1; i+k-1<=n; ++i){
        if(sa.count(make_pair(subhash(hb,i,k,0,m0),
                              subhash(hb,i,k,1,m1)))){
                                  return i;
        }
    }
    return -1;
}
void chash(char* str, ll hashtab[LIM][2]){
    hashtab[0][0] = hashtab[0][1] = 0;
    for(int i=1; i<=n; ++i){
        hashtab[i][0] = (hashtab[i-1][0]*base+(str[i-1]-'A'))%
             m0;
        hashtab[i][1] = (hashtab[i-1][1]*base+(str[i-1]-'A'))%
             m1;
    }
}
int main() {
    scanf("%d", &n);
    scanf("%s%s", a, b);
    pot[0][0] = pot[0][1] = 1;

    for(int i=1;i<LIM;++i){
        pot[i][0] = pot[i-1][0]*base%m0;
        pot[i][1] = pot[i-1][1]*base%m1;
    }
    chash(a, ha);
    chash(b, hb);

    int lo = 0, hi = n;
    int pos = 0, len =0;
    while(lo<=hi){
        int mid = (lo+hi)/2;
        int res = check(mid);
        if(res>=0){
            pos = res-1;
            len = mid;
            lo = mid+1;
        } else {
            hi = mid-1;
        }
    }
    b[pos+len]=0;
    printf("%s\n", b+pos);
}
```

Trie.h

Description: Estructura Trie natural.
Time: $\mathcal{O}(N * D)$

248775, 51 lines

```
const int N = 1e5 + 1; // Number * size of words
const int D = 26;

int nodo;
int trie[N][D];
bool fin[N];
int cnt[N];

void init(){
  nodo = 1;
  for(int i = 0; i < N; i++){
    fin[i] = 0;
        cnt[i] = 0;
    for(int j = 0; j < D; j++){
      trie[i][j] = 0; // nodo no existe
    }
  }
}

void addWord(string s){
  int cur = 0;  // nodo raiz
  for(char ch : s){
    int c = ch - 'a';
    if(!trie[cur][c]){
      trie[cur][c] = nodo++;
    }
    cur = trie[cur][c];
        cnt[cur]++;
  }
  fin[cur] = 1;
}

bool isPrefix(string s){
  int cur = 0;
  for(char ch : s){
    int c = ch - 'a';
    if(!trie[cur][c]) return 0;
    cur = trie[cur][c];
  }
  return 1;
}

bool isWord(string s){
  int cur = 0;
  for(char ch : s){
    int c = ch - 'a';
    if(!trie[cur][c]) return 0;
    cur = trie[cur][c];
  }
  return fin[cur];
}
```

Zfunc.h

680adb, 45 lines

```
const int MAXPATLEN = 30 + 5;
const int MAXTEXLEN = 100000 + 5;

int z[MAXPATLEN + MAXTEXLEN + 5];

void Z(string &s){
    int n = s.size();
    int L = 0, R = 0;
    for(int i = 0; i < n; i++){
        if(i > R){
            L = R = i;
            while(R < n && s[R-L] == s[R]) R++;
            z[i] = R - L;
            R--;
        } else {
            int k = i - L;
            if(z[k] < R-i+1) z[i] = z[k];
            else {
                L = i;
                while(R < n && s[R - L] == s[R]) R++;
```

```
                z[i] = R - L;
                R--;
            }
        }
    }
}

int main(){
    string text, pat;
    cin >> text >> pat;
    string concat = pat + "$" + text;
    Z(concat);

    int cnt = 0;
    int at = pat.size() + 1;
    while(at < concat.size()){
        if(z[at] == pat.size()){
            cnt++;
            at += pat.size() - 1;
        }
        at++;
    }
    cout << cnt << "\n";
    return 0;
}
```

Game theory (7)

Snim.h

Description: Suma de juegos con grundy.
Time: $\mathcal{O}(?)$

cfcf1e, 44 lines

```
int grundy(int n, vector<int> moves){
    if(n == 0) return 0;
    vector<bool> used(100);
    for(int i = 0; i < moves.size(); i++){
        int x = moves[i];
        if(n - x >= 0){
            used[grundy(n - x, moves)] = 1;
        }
    }
    int ret = 0;
    for(int i = 0; i < 1e4 + 5; i++){
        if(!used[i]) {
            ret = i;
            break;
        }
    }
    return ret;
}

void solve(int k){
    vector<int> S(k);
    for(int i = 0; i < k; i++){
        cin >> S[i];
    }

    int m;
    cin >> m;
    while(m--){
        int l;
        cin >> l;
        int ans;
        for(int i = 0; i < l; i++){
            int heap;
            cin >> heap;
            if(i == 0){
                ans = grundy(heap, S);
            } else {
                ans ^= grundy(heap, S);
            }
        }
        cout << (ans == 0 ? 'L' : 'W');
    }
    cout << "\n";
}
```

TimeTakeStones.h

Description: Juego de nim con movimientos especificos (moves).
Time: $\mathcal{O}(N)$

992983, 14 lines

```
const int N = 1e4 + 5;

vector<int> moves;
vector<int> dp(N, 0);

bool isWinning(int n){
    dp[0] = 1;
    for(int i = 1; i <= n; i++){
        for(int x : moves){
            if(n - x >= 0 and !dp[n-x]) dp[n] = 1;
        }
    }
    return dp[n];
}
```

Various (8)

Mergesort.h

Description: Merge Sort

9e3507, 22 lines

```
vector<int> merge(vector<int> &L, vector<int> &R) {
  vector<int> res;
  int at = 0;
  for(auto x : L) {
    while(at < R.size() and R[at] < x) res.emplace_back(R[at
        ++]);
    res.emplace_back(x);
  }
  while(at < R.size()) res.emplace_back(R[at++]);
  return res;
}

vector<int> merge_sort(vector<int> &a) {
  if(a.size() <= 1) return a;
  int n = a.size();
    int mid = n/2;
  vector<int> L(a.begin(), a.begin() + mid);
    vector<int> R(a.begin() + mid, a.end());

    L = merge_sort(L);
  R = merge_sort(R);
  return merge(L, R);
}
```

Techniques (A)

techniques.txt 159 lines

```
Recursion
Divide and conquer
  Finding interesting points in N log N
Algorithm analysis
  Master theorem
  Amortized time complexity
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
  Invariants
  Huffman encoding
Graph theory
  Dynamic graphs (extra book-keeping)
  Breadth first search
  Depth first search
  * Normal trees / DFS trees
  Dijkstra's algorithm
  MST: Prim's algorithm
  Bellman-Ford
  Konig's theorem and vertex cover
  Min-cost max flow
  Lovasz toggle
  Matrix tree theorem
  Maximal matching, general graphs
  Hopcroft-Karp
  Hall's marriage theorem
  Graphical sequences
  Floyd-Warshall
  Euler cycles
  Flow networks
  * Augmenting paths
  * Edmonds-Karp
  Bipartite matching
  Min. path cover
  Topological sorting
  Strongly connected components
  2-SAT
  Cut vertices, cut-edges and biconnected components
  Edge coloring
  * Trees
  Vertex coloring
  * Bipartite graphs (=> trees)
  * 3^n (special case of set cover)
  Diameter and centroid
  K'th shortest path
  Shortest cycle
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
  Dynprog over intervals
  Dynprog over subsets
  Dynprog over probabilities
  Dynprog over trees
  3^n set cover
  Divide and conquer
  Knuth optimization
  Convex hull optimizations
  RMQ (sparse table a.k.a 2^k-jumps)
  Bitonic cycle
  Log partitioning (loop over most restricted)
Combinatorics
  Computation of binomial coefficients
  Pigeon-hole principle
  Inclusion/exclusion
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Chinese remainder theorem
  Fermat's little theorem
  Euler's theorem
  Phi function
  Frobenius number
  Quadratic reciprocity
  Pollard-Rho
  Miller-Rabin
  Hensel lifting
  Vieta root jumping
Game theory
  Combinatorial games
  Game trees
  Mini-max
  Nim
  Games on graphs
  Games on graphs with loops
  Grundy numbers
  Bipartite games without repetition
  General games without repetition
  Alpha-beta pruning
Probability theory
Optimization
  Binary search
  Ternary search
  Unimodality and convex functions
  Binary search on derivative
Numerical methods
  Numeric integration
  Newton's method
  Root-finding with binary/ternary search
  Golden section search
Matrices
  Gaussian elimination
  Exponentiation by squaring
Sorting
  Radix sort
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  KD-trees
  All segment-segment intersection
Sweeping
  Discretization (convert to events and sweep)
  Angle sweeping
  Line sweeping
  Discrete second derivatives
Strings
  Longest common substring
  Palindrome subsequences
  Knuth-Morris-Pratt
  Tries
  Rolling polynomial hashes
  Suffix array
  Suffix tree
  Aho-Corasick
  Manacher's algorithm
  Letter position lists
Combinatorial search
  Meet in the middle
  Brute-force with pruning
  Best-first (A*)
  Bidirectional search
  Iterative deepening DFS / A*
Data structures
  LCA (2^k-jumps in trees in general)
  Pull/push-technique on trees
  Heavy-light decomposition
  Centroid decomposition
  Lazy propagation
  Self-balancing trees
  Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
  Monotone queues / monotone stacks / sliding queues
  Sliding queue using 2 stacks
  Persistent segment tree
```