Team notebook

Nicolas Alba

March 22, 2022

Contents

1 graph

1.1 1 - DFS

```
const int n = 1e6;
vector<int> adj[n + 1];
bool visited[n + 1];

void dfs(int x) {
	if (visited[x]) return;
	visited[x] = true;
	for (int &a : adj[x]) {
		dfs(x);
	}
}
```

1.2 10 - Union Find

```
10. Union Find

int link[n];
int score[n];

void find(int a) {
	if (link[a] == a) return a;
	return link[a] = find(link[a]);
}

void group(int a, int b) {
	int pa = find(a);
	int pb = find(b);
	if (pa != pb) {
		if (score[pa] > score[pb]) {
			link[pb] = pa;
		} else if (score[pa] < score[pb]) {
			link[pa] = pb;
		} else {
			score[pa]++;
			link[pb] = pa;
```

```
                }
        }
}

void init() {
        for (int i = 0; i < n; i++) {
                link[i] = i;
                score[i] = 0;
        }
}
```

1.3 2 - BFS

```
2. BFS

vector<int> adj[n + 1];
bool visited[n + 1];

void bfs() {
        queue<int> q;
        q.push(0); // initial node
        visited[0] = true;
        while(q.size() > 0) {
                int c = q.front();
                q.pop();
                for (int a : adj[c]) {
                        if (visited[a]) continue;
                        q.push(a);
                        visited[a] = true;
                }
        }
}
```

1.4 3 - Dijkstra

```
3. Dijkstra

const int inf = 1e9;
vector<pair<int, int>> adj[n];
bool processed[n];
ll distance[n];

void dijkstra() {
        priority_queue<pair<int, int>> q;
        for (int i = 0; i < n; i++) {
                distance[i] = inf;
        }
        distance[start] = 0;
        q.push({0, start});
        while (q.size() > 0) {
                int c = q.top().second;
                q.pop();
                if (processed[c]) continue;
                processed[c] = true;
                for (auto& a : adj[c]) {
                        int u = a.first;
                        int w = a.second;
                        if (distance[c] + w < distance[u]) {
                                distance[u] = distance[c] + w;
                                q.push({-distance[u], u});
                        }
                }
        }
}
```

1.5 4 - BellmanFord

```
4. BellmanFord

const int inf = 1e9;
vector<tuple<int, int, int>> edges;
ll distance[n];


void bellmanFord() {
        for (int i = 0; i < n; i++) {
                distance[i] = inf;
        }
        distance[start] = 0;
        for (int i = 0; i < n - 1; i++) {
                //bool changed = false; add one iteration (i < n) to valide
                    negative cicles
```

```
                for (auto& edge : edges) {
                        int a, b, w;
                        tie(a, b, w) = edge;
                        if (distance[a] + w < distance[b]) {
                                distance[b] = distance[a] + w;
                                //changed = true;
                        }
                }
        }
}
```

1.6 5 - Floyd Warshall

```
5. Floyd Warshall

const int inf = 1e9;
vector<pair<int, int>> adj[n];
ll distance[n][n];

void floydWarshall() {
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        distance[i][j] = inf;
                }
        }
        for (int i = 0; i < n; i++) {
                for (auto p : adj[i]) {
                        int b = p.first;
                        int w = p.second;
                        distance[i][b] = w;
                }
        }
        for (int k = 0; k < n; k++) {
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                distance[i][j] = min(distance[i][j],
                                    distance[i][k] + distance[k][j]);
                        }
                }
        }
}
```

1.7 6 - Euler Path and Cycle

```
6. Euler Path and Cycle

// TODO
```

1.8 7 - Topological Sort

```
7. Topological Sort

stack<int> topo;
vector<int> adj[n + 1];
bool visited[n + 1];

void dfs(int x) {
        if (visited[x]) return;
        visited[x] = true;
        for (int a : adj[x]) {
                dfs(a);
        }
        topo.push(x);
}
```

1.9 8 - Transitive Closure

```
8. Transitive Closure

const int inf = 1e9;
vector<int> adj[n];
ll distance[n][n];

void floydWarshall() {
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        distance[i][j] = false;
                }
        }
        for (int i = 0; i < n; i++) {
                for (int b : adj[i]) {
```

```
                        distance[i][b] = true;
                }
        }
        for (int k = 0; k < n; k++) {
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                distance[i][j] |= distance[i][k] & distance[k][j];
                        }
                }
        }
}
```

1.10 9 - Kruskal

```
9. Kruskal

vector<tuple<int, int, int> edges;


void kruskal() {
        ll res = 0;
        sort(edges.begin(), edges.end());// sorted by weight
        for (auto edge : edges) {
                int a, b, w;
                tie(w, a, b) = edge;
                if (find(a) != find(b)) {
                        group(a, b);
                        res += w;
                }
        }
}
```

2 math

2.1 Extended Euclides

```
// It finds X and Y in equation:
// a * X + b * Y = gcd(a, b)
int x, y;

int euclid(int a, int b) {
        if (b == 0) {
                x = 1;
                y = 0;
                return a;
        }
        int aux = x;
        x = y;
        y = aux - a/b*y;
        return euclid(b, a % b);
}
```

2.2 Greatest Common Divisor

```
// Alternative: __gcd(a, b);
// O(log(max(a, b)))

ll gcd(ll a, ll b) {
        return b == 0 ? a : gcd(b, a % b);
}
```

2.3 Lowest Common Multiple

```
// O(log(max(a, b)))
int lcm(int a, int b) {
        return a/gcd(a, b) * b;
}
```

2.4 primes

```
// O(sqrt(n))
bool isPrime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
```

```
    }
    return true;
}

// O(nloglogn)
// sieve[X] == 0 if it is prime
int const N = 1e6;
bool sieve[N + 1];
vector<int> primes;

void calculate() {
    for (int p = 2; p <= N; p++) {
        if (sieve[p]) continue;
        primes.PB(p);
        for (ll i = 1ll*p*p; i <= N; i += p)
            sieve[i] = true;
    }
}


// For 64-bit integers
// O((ln n)^2)
// 32 bits bases: 2, 3, 5, 7.
// 64 bits bases: 2 ... 37

using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
}

bool MillerRabin(u64 n) {
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```