

## Contents

1 Math	1	2 Graph	1
1.1 快速幂	1	2.1 Tarjan SCC	1
1.2 擴展歐幾里得	1	2.2 AP/Bridge	1
		3 String	2
		3.1 Hash	2
		3.2 Zvalue	2

## 1 Math

### 1.1 快速幂

```
//x^y % p
int func(int x,int y,int p){
    int res = 1;
    while(y != 0){
        if(y%2==1){
            res *= x;
            res %=p;
        }
        x *= x;
        y /= 2; // 5^8 => (5^2)^4
        x %= p; // ((5^2) % 7)^4
    }
    return res;
}
```

### 1.2 擴展歐幾里得

```
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);
}

pair<int, int> ext_gcd
(int a, int b) //擴展歐幾里得 ax+by = gcd(a,b)
{
    if (b == 0)
        return {1, 0};
    if (a == 0)
        return {0, 1};
    int x, y;
    tie(x, y) = ext_gcd(b % a, a);
    return make_pair(y - b * x / a, x);
}
```

## 2 Graph

### 2.1 Tarjan SCC

```
class tarjan{
    int time = 1;
    stack<int> s;
    vector<int> dfn;
    vector<int> low;
    vector<bool> in_stack;
    vector<vector<int>> ans;
    void dfs(int node, vector<vector<int>> &graph){
        in_stack[node] = true;
        s.push(node);
        dfn[node] = low[node] = time++;
        for(auto &j : graph[node]){
            if(dfn[j] == 0){
                dfs(j, graph);
                // 看看往下有沒有辦法回到更上面的點
                low[node] = min(low[node], low[j]);
            }
            else if(in_stack[j]){
                low[node] = min(low[node], low[j]);
            }
        }
        vector<int> t; // 儲存這個強連通分量
        if(dfn[node] == low[node]){
            while(s.top() != node){
                t.push_back(s.top());
                in_stack[s.top()] = false;
                s.pop();
            }
        }
    }
}
```

```
t.push_back(s.top());
in_stack[s.top()] = false;
s.pop();
}
if(!t.empty()) ans.push_back(t);
}
public:
vector
    <vector<int>> scc(vector<vector<int>> &graph){
        int num = graph.size();
        dfn.resize(num, 0);
        low.resize(num, 0);
        in_stack.resize(num, false);
        for(int i = 1; i < num; i++){
            if(dfn[i] == 0) dfs(i, graph);
        }
        return ans;
    }
};
```

### 2.2 AP/Bridge

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u

int dfsAP(int u, int p) {
    int children = 0;
    low[u] = disc[u] = ++Time;
    for (int& v : adj[u]) {
        if (v == p) continue; //
        // we don't want to go back through the same path.
        // if we go back is because
        // we found another way back

        if (!disc[v]) { // if v has not been discovered before
            children++;
            dfsAP(v, u); // recursive DFS call
            if (disc[u] <= low[v]) // condition #1
                ap[u] = 1;
            low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
        } else // if v was already
            // discovered means that we found an ancestor
            low[u] = min(low[u], disc[v]); // finds
            // the ancestor with the least discovery time

    }
    return children;
}

void AP() {
    ap = low = disc = vector<int>(adj.size());
    Time = 0;
    for (int u = 0; u < adj.size(); u++)
        if (!disc[u])
            ap[u] = dfsAP(u, u) > 1; // condition #2
}

// br = bridges, p = parent

vector<pair<int, int>> br;

int dfsBR(int u, int p) {
    low[u] = disc[u] = ++Time;
    for (int& v : adj[u]) {
        if (v == p) continue; //
        // we don't want to go back through the same path.
        // if we go back is because
        // we found another way back

        if (!disc[v]) { // if v has not been discovered before
            dfsBR(v, u); // recursive DFS call
            if (disc[u] < low[v]) // condition to find a bridge
                br.push_back({u, v});
            low[u] = min(low[u], low[v]); // low[v] might be an ancestor of u
        } else // if v was already
            // discovered means that we found an ancestor
            low[u] = min(low[u], disc[v]); // finds
            // the ancestor with the least discovery time

    }
}
```

```

void BR() {
    low = disc = vector<int>(adj.size());
    Time = 0;
    for (int u = 0; u < adj.size(); u++)
        if (!disc[u])
            dfsBR(u, u)
}

```

## 3 String

### 3.1 Hash

```

vector<int> Pow(int num){
    int p = 1e9 + 7;
    vector<int> ans = {1};
    for(int i = 0; i < num; i++)
        ans.push_back(ans.back() * b % p);
    return ans;
}

vector<int> Hash(string s){
    int p = 1e9 + 7;
    vector<int> ans = {0};
    for(char c:s){
        ans.push_back((ans.back() * b + c) % p);
    }
    return ans;
}

// 閉區間[l, r]
int query
(vector<int> &vec, vector<int> &pow, int l, int r){
    int p = 1e9 + 7;
    int length = r - l + 1;
    return
        (vec[r + 1] - vec[l] * pow[length] % p + p) % p;
}

```

### 3.2 Zvalue

```

vector<int> z_func(string s1){
    int l = 0, r = 0, n = s1.size();
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i
            <= r and z[i - l] < r - i + 1) z[i] = z[i - l];
        else{
            z[i] = max(z[i], r - i + 1);
            while(i + z
                [i] < n and s1[i + z[i]] == s1[z[i]]) z[i]++;
        }
        if(i + z[i] - 1 > r){
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

```