Team Reference Document

cprefer

August 23, 2023

Contents

1 String

1.1 Z

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

// Note : z[0] = 0.
vector<int> get_z(const string &s) {
  int n = s.size();
  vector<int> z(n);
  for (int i = 1, j = 0; i < n; i += 1) {
    z[i] = max(min(z[i - j], j + z[j] - i), 0);
    while (i + z[i] < n and s[i + z[i]] == s[z[i]]) {
      z[i] += 1;
    }
    if (i + z[i] > j + z[j]) {
      j = i;
    }
  }
  return z;
}
```

1.2 Manacher

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

// Find palindromes with odd length.
vector<int> manacher(const string &s) {
  int n = s.size();
  vector<int> p(n);
  for (int i = 0, j = 0; i < n; i += 1) {
    if (j + p[j] > i) {
      p[i] = min(p[j * 2 - i], j + p[j] - i);
    }
    while (i >= p[i] and i + p[i] < n and s[i - p[i]] == s[i + p[i]]) {
      p[i] += 1;
    }
    if (i + p[i] > j + p[j]) {
      j = i;
    }
  }
  return p;
}
```

1.3 Suffix Array

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

pair<vector<int>, vector<int>> binary_lifting(const string &s) {
  int n = s.size(), k = 0;
  vector<int> p(n), rank(n), q, count;
  iota(p.begin(), p.end(), 0);
  ranges::sort(p, {}, [&](int i) { return s[i]; });
  for (int i = 0; i < n; i += 1) {
    rank[p[i]] = i and s[p[i]] == s[p[i - 1]] ? rank[p[i - 1]] : k++;
  }
  for (int m = 1; m < n; m *= 2) {
    q.resize(m);
    iota(q.begin(), q.end(), n - m);
    for (int i : p) {
      if (i >= m) {
        q.push_back(i - m);
      }
    }
    count.assign(k, 0);
    for (int i : rank) {
      count[i] += 1;
    }
    partial_sum(count.begin(), count.end(), count.begin());
    for (int i = n - 1; i >= 0; i -= 1) {
      p[count[rank[q[i]]] -= 1] = q[i];
    }
    auto previous = rank;
    previous.resize(2 * n, -1);
    k = 0;
    for (int i = 0; i < n; i += 1) {
      rank[p[i]] = i and previous[p[i]] == previous[p[i - 1]] and
                         previous[p[i] + m] == previous[p[i - 1] + m]
                     ? rank[p[i - 1]]
                     : k++;
    }
  }
  vector<int> lcp(n);
  k = 0;
  for (int i = 0; i < n; i += 1) {
    if (rank[i]) {
      k = max(k - 1, 0);
      int j = p[rank[i] - 1];
      while (i + k < n and j + k < n and s[i + k] == s[j + k]) {
        k += 1;
      }
      lcp[rank[i]] = k;
    }
  }
  return {p, lcp};
}
```

1.4 Lyndon Factorization

```cpp
#include <bits/stdc++.h>
```

```cpp
using namespace std;
vector<int> lyndon(const string &s) {
  int n = s.size();
  vector<int> res;
  for (int i = 0, j, k; i < n;) {
    j = (k = i) + 1;
    for (j = (k = i) + 1; j < n and s[k] <= s[j]; j += 1) {
      k = s[k] < s[j] ? i : k + 1;
    }
    while (i <= k) {
      res.push_back(i += (j - k));
    }
  }
  return res;
}
```

2 Graph

2.1 Tarjan

2.1.1 Strongly Connected Component

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

vector<vector<int>> scc(const vector<vector<int>> &g) {
  int n = g.size();
  vector<bool> done(n);
  vector<int> pos(n, -1), stack;
  vector<vector<int>> res;
  function<int(int)> dfs = [&](int u) {
    int low = pos[u] = stack.size();
    stack.push_back(u);
    for (int v : g[u]) {
      if (not done[v]) {
        low = min(low, ~pos[v] ? pos[v] : dfs(v));
      }
    }
    if (low == pos[u]) {
      res.emplace_back(stack.begin() + low, stack.end());
      for (int v : res.back()) {
        done[v] = true;
      }
      stack.resize(low);
    }
    return low;
  };
  for (int i = 0; i < n; i += 1) {
    if (not done[i]) {
      dfs(i);
    }
  }
  ranges::reverse(res);
  return res;
}
```