Acm Algorithm Templates

Epoche

Contents

1 Introduction

Personal Acm Templates

2 Planing

Problem Set https://www.luogu.com.cn/training/9391

- 矩阵
  1. 乘法 ✓
  2. 逆矩阵
- 稀疏表 ✓
- Disjoint union
  1. 带权并查集
  2. 判连通块个数 ✓
- 树状数组
  1. 权值树状数组
  2. 单点修改，区间查询，区间修改，单点查询
- 线段树
- 图
  1. Dijkstra ✓
  2. Bellman-ford
- 最小生成树 (Minimal spanning tree)
  1. Kruskal
  2. prim
- 素数筛
  1. 埃氏筛
  2. 欧拉筛

3 File Header

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

typedef long long i64;
```

Or

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <cmath>
#include <queue>
#include <stack>
#include <set>
#include <functional>

// #define rs ranges
#define fst first
#define snd second

typedef long long i64;
```

Optimize

```cpp
#pragma GCC optimize(2)
```

4 Graph

4.1 Storage

adjacency list

```cpp
struct Graph {
    int N;
    std::vector<std::vector<std::pair<int, int>>> adj;

    Graph(int n) {
        N = n;
        adj.resize(n);
    }

    void add(int u, int v, int d) {
        adj[u - 1].emplace_back(v - 1, d);
    }
};
```

4.2 Traverse

DFS

```cpp
void dfs(int u) {
    std::vector<int> vis(N, 1);

    auto recursive = [&](auto self, auto cur) -> void {
        vis[cur] = 0;

        for (auto [v, w] : adj[cur]) {
            if (vis[v] == 0) continue;
            self(self, v);
        }
    };

    recursive(recursive, u);
}
```

BFS

```cpp
void bfs(int origin) {
    std::vector<int> vis(N, 1);
    std::queue<int> q;
    q.emplace(origin);

    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        vis[u] = 0;

        for (auto [v, _] : adj[u]) {
            if (vis[v] == 0) continue;
            q.emplace(v);
        }
    }
}
```

4.3 Shortest path

Heap optimized Dijkstra

Storage the minus edge weight, so that we don't need write the declaration: `std::greater<int>`.

```
void dijkstra() {
    std::vector<int> dis(N, -1);
    std::priority_queue<std::pair<int, int>> h;
    h.emplace(0, 0);

    while (!h.empty()) {
        auto [d, u] = h.top();
        h.pop();

        if (dis[u] != -1) continue;
        dis[u] = -d;

        for (auto [v, w] : adj[u]) {
            h.emplace(d - w, v);
        }
    }
}
```

5 Math

5.1 sieve prime number

Liner sieve

```
void sieveLiner (int n) {
    for (int i = 2; i <= n; i++) {
        if (st[i] == 0) pri[cnt++] = i;
        for (int j = 0; pri[j] <= n / i; j++) {
            st[pri[j] * i] = true;
            if (i % pri[j] == 0) break;
        }
    }
}
```

trial division determin prime number

```
bool isPrime (int n) {
    if (n < 2) return false;
    for (int i = 2; i <= n / i; i++)
        if (n % i == 0) return false;
    return true;
}
```

5.2 Fast power

```
constexpr i64 MODP = 1000000007;
constexpr i64 ksm (i64 a, i64 b, i64 p = MODP) {
    i64 res = 1;
    for (; b; b >>= 1, a = a * a % p)
        if (b & 1) res = res * a % p;
    return res;
}
```

5.3 GCD

```
i64 gcd(i64 a, i64 b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

5.4 Matrix

```cpp
constexpr i64 MOD = 1000000007;
struct Matrix {
    int row, column;
    std::vector<std::vector<int>> data;

    Matrix() {}
    Matrix(int n, int m) {
        init(n, m);
    }

    void init(int n, int m) {
        row = n, column = m;
        data.assign(n, std::vector<int>(m));
    }

    std::vector<int> operator[](const int &x) {
        return data[x];
    }

    Matrix operator*(const Matrix &other) const {
        Matrix res(row, other.column);
        int r = row, c = other.column, l = column;

        for (int i = 0; i < r; i++) {
            for (int k = 0; k < l; k++) {
                int cur = data[i][k];
                for (int j = 0; j < c; j++) {
                    res[i][j] += cur * other.data[k][j];
                    res[i][j] %= MOD;
                }
            }
        }

        return res;
    }
};
```

6 Data Structure

6.1 Monotonic queue

```cpp
std::deque<int> mq;
for (int i = 0; i < n; i++) {
    if (mq.front() < i - len + 1) mq.pop_front();
    while (!mq.empty() && mq.back() > a[i]) mq.pop_back();
    mq.emplace_back(a[i]);
}
```

6.2 Sparse Table (Rrange Maximum/Minimum Query)

```cpp
template<class T>
struct ST {
    int n;
    std::vector<std::vector<T>> st;

    using optFunction = std::function<T(const T &, const T &)>;
    optFunction opt;
    static T defaultOpt(const T & a, const T & b) {
        return std::max(a, b);
    }

```

```
    ST(const std::vector<T> & _init, optFunction _opt = defaultOpt) {
        opt = _opt;
        init(_init);
    }

    void init(const std::vector<T> & _init) {
        n = _init.size();
        int cap = std::log2(n) + 1;
        st.assign(n, std::vector<T>(cap));

        for (int i = 0; i < n; i++) {
            st[i][0] = _init[i];
        }

        for (int j = 1; j < cap; j++) {
            int cur = 1 << (j - 1);
            for (int i = 0; i + cur < n; i++) {
                st[i][j] = opt(st[i][j - 1], st[i + cur][j - 1]);
            }
        }
    }

    T query(int l, int r) {
        int k = std::log2(r - l + 1);
        return opt(st[l][k], st[r - (1 << k) + 1][k]);
    }

    T query(int l, int r, int k) {
        return opt(st[l][k], st[r - (1 << k) + 1][k]);
    }
};
```

6.3 Disjoint union

```
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }

    int find(int x) {
        return f[x] == x ? x : f[x] = find(f[x]);
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int a, int b) {
        int fa = find(a), fb = find(b);
        if (fa == fb) {
            return false;
        }
        siz[fb] += siz[fa];
        f[fa] = fb;
```

```
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

6.4 Fenwick Tree

```
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n = 0) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

    T sum(int x) {
        auto ans = T();
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }
};
```

6.5 Segment Tree

```
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;

    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
```

```
        init(init_);
    }

    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }

    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }

    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
       return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }

    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
```

```
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F pred) {
        return findFirst(1, 0, n, l, r, pred);
    }

    template<class F>
    int findLast(int p, int l, int r, int x, int y, F pred) {
        if (l >= y || r <= x || !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
```

7 Misc

7.1 Leap year

Leap years are **evenly divisible by 4**. The most recent leap year was 2020 and the next leap year will be 2024. However, any year that is **evenly divided by 100 would not be a leap year unless it is evenly divided by 400**. This is why 1600, 2000, and 2400 are leap years, while 1700, 1800, 1900, 2100, 2200, and 2300 are common years, even though they are all divisible by 4.

```
bool isLeapYear (int year) {
    return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
}
```

7.2 Sum of Prefix and Difference

2D

```
void build(std::vector<std::vector<int>> sum, auto a, int n, int m) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            sum[i][j] = sum[i][j - 1] + a[i - 1][j - 1] -
                        sum[i - 1][j - 1] + sum[i - 1][j];
        }
    }
}

int query(auto sum, int a, int b, int c, int d) {
    return sum[c][d] + sum[a - 1][b - 1] - sum[a - 1][d] - sum[c][b - 1];
}
```

```
void build(auto dif, auto a, auto n, auto m) {
    for (int i = n; i > 0; i--) {
        for (int j = m; j > 0; j--) {
```

```
            dif[i][j] = a[i][j] - dif[i - 1][j] -
                        dif[i][j - 1] + dif[i - 1][j - 1];
        }
    }
}
void modi(auto dif, int x1, int x2, int y1, int y2, int x) {
    dif[x1][y1] += x, dif[x2 + 1][y2 + 1] += x;
    dif[x1][y2 + 1] -= x, dif[x2 + 1][y1] -= x;
}
```

7.3 Fast IO

```
namespace io {
  struct read {
      static constexpr int M = 1 << 23;
      char buf[M], *S = buf, *P = buf, c, l;

      inline char gc() {
          return (S == P && (P = (S = buf) + fread(buf, 1, M, stdin), S == P) ? EOF : *S++);
      }

      template<typename T> read &operator>>(T &x) {
          for (c = 0; !isdigit(c); c = gc()) {
              l = c;
          }

          for (x = 0; isdigit(c); c = gc()) {
              x = x * 10 + c - '0';
          }

          return x = (l ^ 45) ? x : -x, *this;
      }
  } in;
}
```

7.4 CPU Information

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

using u32 = uint32_t;

static void cpu(u32 X, u32 Y, u32 msg[4]) {
    __cpuid_count(X, Y, msg[0], msg[1], msg[2], msg[3]);
}

int main() {
    u32 data[4];
    char msg[50];

    for (int i = 0; i < 3; ++i) {
        cpu(0x80000002 + i, 0, data);

        for (int j = 0; j < 4; ++j) {
            reinterpret_cast<u32 *>(msg)[i * 4 + j] = data[j];
        }
    }

    std::cout << msg;
}
```

7.5 STL

```
std::vector<T,Allocator>::assign

void assign( size_type count, const T& value );

template< class InputIt >
void assign( InputIt first, InputIt last );
```