

Acm Algorithm Templates

Epoche

Contents

1 Introduction	3
2 Planing	3
3 File Header	3
4 Graph	4
4.1 Storage	4
4.2 Traverse	4
4.3 Shortest path	4
5 Math	5
5.1 sieve prime number	5
5.2 Fast power	5
5.3 GCD	5
5.4 Matrix	6
6 Data Structure	6
6.1 Monotonic queue	6
6.2 Sparse Table (Rrange Maximum/Minimum Query)	6
6.3 Disjoint union	7
6.4 Fenwick Tree	8
6.5 Segment Tree	8
7 Misc	10
7.1 Leap year	10
7.2 Sum of Prefix and Difference	10
7.3 Fast IO	11
7.4 CPU Information	11

1 Introduction

Personal Acem 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

```
1 #include <bits/stdc++.h>
2 #define rs ranges
3
4 typedef long long i64;
```

Or

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <numeric>
5 #include <cmath>
6 #include <queue>
7 #include <stack>
8 #include <set>
9 #include <functional>
10
11 // #define rs ranges
12 #define fst first
13 #define snd second
14
15 typedef long long i64;
```

Optimize

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

4 Graph

4.1 Storage

adjacency list

```
1 struct Graph {
2     int N;
3     std::vector<std::vector<std::pair<int, int>>> adj;
4
5     Graph(int n) {
6         N = n;
7         adj.resize(n);
8     }
9
10    void add(int u, int v, int d) {
11        adj[u - 1].emplace_back(v - 1, d);
12    }
13 };
```

4.2 Traverse

DFS

```
1 void dfs(int u) {
2     std::vector<int> vis(N, 1);
3
4     auto recursive = [&](auto self, auto cur) -> void {
5         vis[cur] = 0;
6
7         for (auto [v, w] : adj[cur]) {
8             if (vis[v] == 0) continue;
9             self(self, v);
10        }
11    };
12
13    recursive(recursive, u);
14 }
```

BFS

```
1 void bfs(int origin) {
2     std::vector<int> vis(N, 1);
3     std::queue<int> q;
4     q.emplace(origin);
5
6     while (!q.empty()) {
7         auto u = q.front();
8         q.pop();
9         vis[u] = 0;
10
11        for (auto [v, _] : adj[u]) {
12            if (vis[v] == 0) continue;
13            q.emplace(v);
14        }
15    }
16 }
```

4.3 Shortest path

Heap optimized Dijkstra

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

```
1 void dijkstra() {
2     std::vector<int> dis(N, -1);
3     std::priority_queue<std::pair<int, int>> h;
4     h.emplace(0, 0);
5
6     while (!h.empty()) {
7         auto [u, d] = h.top();
8         h.pop();
9
10        if (dis[u] != -1) continue;
11        dis[u] = -d;
12
13        for (auto [v, w] : adj[u]) {
14            h.emplace(v, d - w);
15        }
16    }
17 }
```

5 Math

5.1 sieve prime number

Liner sieve

```
1 void sieveLiner (int n) {
2     for (int i = 2; i <= n; i++) {
3         if (st[i] == 0) pri[cnt++] = i;
4         for (int j = 0; pri[j] <= n / i; j++) {
5             st[pri[j] * i] = true;
6             if (i % pri[j] == 0) break;
7         }
8     }
9 }
```

trial division determin prime number

```
1 bool isPrime (int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++)
4         if (n % i == 0) return false;
5     return true;
6 }
```

5.2 Fast power

```
1 constexpr i64 MODP = 1000000007;
2 constexpr i64 ksm (i64 a, i64 b, i64 p = MODP) {
3     i64 res = 1;
4     for (; b >>= 1, a = a * a % p)
5         if (b & 1) res = res * a % p;
6     return res;
7 }
```

5.3 GCD

```
1 i64 gcd(i64 a, i64 b) {
2     return b == 0 ? a : gcd(b, a % b);
3 }
```

5.4 Matrix

```
1 constexpr i64 MOD = 1000000007;
2 struct Matrix {
3     int row, column;
4     std::vector<std::vector<int>> data;
5
6     Matrix() {}
7     Matrix(int n, int m) {
8         init(n, m);
9     }
10
11     void init(int n, int m) {
12         row = n, column = m;
13         data.assign(n, std::vector<int>(m));
14     }
15
16     std::vector<int> operator[](const int &x) {
17         return data[x];
18     }
19
20     Matrix operator*(const Matrix &other) const {
21         Matrix res(row, other.column);
22         int r = row, c = other.column, l = column;
23
24         for (int i = 0; i < r; i++) {
25             for (int k = 0; k < l; k++) {
26                 int cur = data[i][k];
27                 for (int j = 0; j < c; j++) {
28                     res[i][j] += cur * other.data[k][j];
29                     res[i][j] %= MOD;
30                 }
31             }
32         }
33
34         return res;
35     }
36 };
```

6 Data Structure

6.1 Monotonic queue

```
1 std::deque<int> mq;
2 for (int i = 0; i < n; i++) {
3     if (mq.front() < i - len + 1) mq.pop_front();
4     while (!mq.empty() && mq.back() > a[i]) mq.pop_back();
5     mq.emplace_back(a[i]);
6 }
```

6.2 Sparse Table (Range Maximum/Minimum Query)

```
1 template<class T>
2 struct ST {
3     int n;
4     std::vector<std::vector<T>> st;
5
6     using optFunction = std::function<T(const T &, const T &>>;
7     optFunction opt;
8     static T defaultOpt(const T & a, const T & b) {
9         return std::max(a, b);
10    }
11 }
```

```

12     ST(const std::vector<T> & _init, optFunction _opt = defaultOpt) {
13         opt = _opt;
14         init(_init);
15     }
16
17     void init(const std::vector<T> & _init) {
18         n = _init.size();
19         int cap = std::log2(n) + 1;
20         st.assign(n, std::vector<T>(cap));
21
22         for (int i = 0; i < n; i++) {
23             st[i][0] = _init[i];
24         }
25
26         for (int j = 1; j < cap; j++) {
27             int cur = 1 << (j - 1);
28             for (int i = 0; i + cur < n; i++) {
29                 st[i][j] = opt(st[i][j - 1], st[i + cur][j - 1]);
30             }
31         }
32     }
33
34     T query(int l, int r) {
35         int k = std::log2(r - l + 1);
36         return opt(st[l][k], st[r - (1 << k) + 1][k]);
37     }
38
39     T query(int l, int r, int k) {
40         return opt(st[l][k], st[r - (1 << k) + 1][k]);
41     }
42 };

```

6.3 Disjoint union

```

1  struct DSU {
2      std::vector<int> f, siz;
3
4      DSU() {}
5      DSU(int n) {
6          init(n);
7      }
8
9      void init(int n) {
10         f.resize(n);
11         std::iota(f.begin(), f.end(), 0);
12         siz.assign(n, 1);
13     }
14
15     int find(int x) {
16         return f[x] == x ? x : f[x] = find(f[x]);
17     }
18
19     bool same(int x, int y) {
20         return find(x) == find(y);
21     }
22
23     bool merge(int a, int b) {
24         int fa = find(a), fb = find(b);
25         if (fa == fb) {
26             return false;
27         }
28         siz[fb] += siz[fa];
29         f[fa] = fb;

```

```

30         return true;
31     }
32
33     int size(int x) {
34         return siz[find(x)];
35     }
36 };

```

6.4 Fenwick Tree

```

1  template <typename T>
2  struct Fenwick {
3      int n;
4      std::vector<T> a;
5
6      Fenwick(int n = 0) {
7          init(n);
8      }
9
10     void init(int n) {
11         this->n = n;
12         a.assign(n, T());
13     }
14
15     void add(int x, T v) {
16         for (int i = x + 1; i <= n; i += i & -i) {
17             a[i - 1] += v;
18         }
19     }
20
21     T sum(int x) {
22         auto ans = T();
23         for (int i = x; i > 0; i -= i & -i) {
24             ans += a[i - 1];
25         }
26         return ans;
27     }
28
29     T rangeSum(int l, int r) {
30         return sum(r) - sum(l);
31     }
32
33     int kth(T k) {
34         int x = 0;
35         for (int i = 1 << std::lg(n); i; i /= 2) {
36             if (x + i <= n && k >= a[x + i - 1]) {
37                 x += i;
38                 k -= a[x - 1];
39             }
40         }
41         return x;
42     }
43 };

```

6.5 Segment Tree

```

1  template<class Info>
2  struct SegmentTree {
3      int n;
4      std::vector<Info> info;
5
6      SegmentTree() : n(0) {}
7      SegmentTree(int n_, Info v_ = Info()) {
8          init(n_, v_);
9      }
10     template<class T>
11     SegmentTree(std::vector<T> init_) {

```



```

12     init(init_);
13 }
14
15 void init(int n_, Info v_ = Info()) {
16     init(std::vector(n_, v_));
17 }
18
19 template<class T>
20 void init(std::vector<T> init_) {
21     n = init_.size();
22     info.assign(4 << std::lg(n), Info());
23     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
24         if (r - l == 1) {
25             info[p] = init_[l];
26             return;
27         }
28         int m = (l + r) / 2;
29         build(2 * p, l, m);
30         build(2 * p + 1, m, r);
31         pull(p);
32     };
33     build(1, 0, n);
34 }
35
36 void pull(int p) {
37     info[p] = info[2 * p] + info[2 * p + 1];
38 }
39
40 void modify(int p, int l, int r, int x, const Info &v) {
41     if (r - l == 1) {
42         info[p] = v;
43         return;
44     }
45     int m = (l + r) / 2;
46     if (x < m) {
47         modify(2 * p, l, m, x, v);
48     } else {
49         modify(2 * p + 1, m, r, x, v);
50     }
51     pull(p);
52 }
53 void modify(int p, const Info &v) {
54     modify(1, 0, n, p, v);
55 }
56
57 Info rangeQuery(int p, int l, int r, int x, int y) {
58     if (l >= y || r <= x) {
59         return Info();
60     }
61     if (l >= x && r <= y) {
62         return info[p];
63     }
64     int m = (l + r) / 2;
65     return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
66 }
67 Info rangeQuery(int l, int r) {
68     return rangeQuery(1, 0, n, l, r);
69 }
70
71 template<class F>
72 int findFirst(int p, int l, int r, int x, int y, F pred) {
73     if (l >= y || r <= x || !pred(info[p])) {
74         return -1;
75     }
76     if (r - l == 1) {
77         return l;
78     }
79     int m = (l + r) / 2;
80     int res = findFirst(2 * p, l, m, x, y, pred);
81     if (res == -1) {

```

```

82         res = findFirst(2 * p + 1, m, r, x, y, pred);
83     }
84     return res;
85 }
86 template<class F>
87 int findFirst(int l, int r, F pred) {
88     return findFirst(1, 0, n, l, r, pred);
89 }
90
91 template<class F>
92 int findLast(int p, int l, int r, int x, int y, F pred) {
93     if (l >= y || r <= x || !pred(info[p])) {
94         return -1;
95     }
96     if (r - l == 1) {
97         return l;
98     }
99     int m = (l + r) / 2;
100    int res = findLast(2 * p + 1, m, r, x, y, pred);
101    if (res == -1) {
102        res = findLast(2 * p, l, m, x, y, pred);
103    }
104    return res;
105 }
106 template<class F>
107 int findLast(int l, int r, F pred) {
108     return findLast(1, 0, n, l, r, pred);
109 }
110 };

```

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.

```

1 bool isLeapYear (int year) {
2     return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
3 }

```

7.2 Sum of Prefix and Difference

2D

```

1 void build(std::vector<std::vector<int>> sum, auto a, int n, int m) {
2     for (int i = 1; i <= n; i++) {
3         for (int j = 1; j <= m; j++) {
4             sum[i][j] = sum[i][j - 1] + a[i - 1][j - 1] -
5                 sum[i - 1][j - 1] + sum[i - 1][j];
6         }
7     }
8 }
9
10 int query(auto sum, int a, int b, int c, int d) {
11     return sum[c][d] + sum[a - 1][b - 1] - sum[a - 1][d] - sum[c][b - 1];
12 }

```

```

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

```

```

4         dif[i][j] = a[i][j] - dif[i - 1][j] -
5             dif[i][j - 1] + dif[i - 1][j - 1];
6     }
7 }
8 }
9 void modi(auto dif, int x1, int x2, int y1, int y2, int x) {
10     dif[x1][y1] += x, dif[x2 + 1][y2 + 1] += x;
11     dif[x1][y2 + 1] -= x, dif[x2 + 1][y1] -= x;
12 }

```

7.3 Fast IO

```

1 namespace io {
2     struct read {
3         static constexpr int M = 1 << 23;
4         char buf[M], *S = buf, *P = buf, c, l;
5
6         inline char gc() {
7             return (S == P && (P = (S = buf) + fread(buf, 1, M, stdin), S == P) ?
8 EOF : *S++);
9         }
10
11         template<typename T> read &operator>>(T &x) {
12             for (c = 0; !isdigit(c); c = gc()) {
13                 l = c;
14             }
15
16             for (x = 0; isdigit(c); c = gc()) {
17                 x = x * 10 + c - '0';
18             }
19
20             return x = (l ^ 45) ? x : -x, *this;
21         }
22     } in;
23 }

```

7.4 CPU Information

```

1 #include <bits/stdc++.h>
2 #include <cpuid.h>
3
4 using u32 = uint32_t;
5
6 static void cpu(u32 X, u32 Y, u32 msg[4]) {
7     __cpuid_count(X, Y, msg[0], msg[1], msg[2], msg[3]);
8 }
9
10 int main() {
11     u32 data[4];
12     char msg[50];
13
14     for (int i = 0; i < 3; ++i) {
15         cpu(0x80000002 + i, 0, data);
16
17         for (int j = 0; j < 4; ++j) {
18             reinterpret_cast<u32 *>(msg)[i * 4 + j] = data[j];
19         }
20     }
21
22     std::cout << msg;
23 }

```