**KMP字符串匹配**

#include <iostream>

#include <string>

#include <array>

std::array<int, 1000000 + 5> f{ 0 };

void GetFail(const std::string &P)

{

for (std::size\_t i = 1; i < P.size(); ++i)

{

auto j = f[i];

while (j && P[i] != P[j])

{

j = f[j];

}

f[i + 1] = P[i] == P[j] ? j + 1 : 0;

}

}

void Find(const std::string &T, const std::string &P)

{

GetFail(P);

std::size\_t j = 0;

for (std::size\_t i = 0; i < T.size(); ++i)

{

while (j && P[j] != T[i])

{

j = f[j];

}

if (P[j] == T[i])

{

++j;

}

if (j == P.size())

{

std::cout << i - P.size() + 2 << std::endl;

}

}

}

int main()

{

std::string T, P;

std::cin >> P >> T;

Find(P, T);

for (std::size\_t i = 1; i <= T.size(); ++i)

{

std::cout << f[i] << ' ';

}

std::putchar('\n');

return 0;

}

**Manacher算法**

#include <iostream>

#include <cstring>

#include <algorithm>

char s[21000001];

char s\_new[42000002];

int p[21000001];

int Init()

{

auto len = std::strlen(s);

s\_new[0] = '$';

 s\_new[1] = '#';

int j = 2;

for (std::size\_t i = 0; i < len; ++i)

{

s\_new[j++] = s[i];

s\_new[j++] = '#';

}

s\_new[j] = '\0';

return j;

}

int Manacher()

{

int len = Init();

int MaxLen = -1;

int id;

int mx = 0;

for (int i = 1; i < len; ++i)

{

if (i < mx)

{

p[i] = std::min(p[2 \* id - i], mx - i);

}

else

{

p[i] = 1;

}

while (s\_new[i - p[i]] == s\_new[i + p[i]])

{

++p[i];

}

if (mx < i + p[i])

{

id = i;

mx = i + p[i];

}

MaxLen = std::max(MaxLen, p[i] - 1);

}

return MaxLen;

}

int main()

{

std::scanf("%s", s);

std::cout << Manacher() << std::endl;

return 0;

}

**并查集（Rank优化）**

#include <iostream>

#include <array>

std::array<int, 100001> Par{ 0 }, Rank{ 0 };

template <typename T>

int Find(const T &x)

{

if (Par[x] == x)

return x;

else

return Par[x] = Find(Par[x]);

}

template <typename T>

void Unite(T &x, T &y)

{

x = Find(x);

y = Find(y);

if (x == y)

return;

if (Rank[x] < Rank[y])

{

Par[x] = y;

}

else

{

Par[y] = x;

if (Rank[x] == Rank[y])

++Rank[x];

}

}

template <typename T>

void IsSame(const T &x, const T &y)

{

if (Find(x) == Find(y))

std::cout << 'Y' << std::endl;

else

std::cout << 'N' << std::endl;

}

int main()

{

int N, M;

std::cin >> N >> M;

for (int i = 0; i < N; ++i)

{

Par[i] = i;

Rank[i] = 0;

}

int Z, X, Y;

while (M--)

{

std::cin >> Z >> X >> Y;

switch (Z)

{

case 1:

Unite(X, Y);

break;

case 2:

IsSame(X, Y);

break;

default:

break;

}

}

return 0;

}

**乘法逆元（递推）**

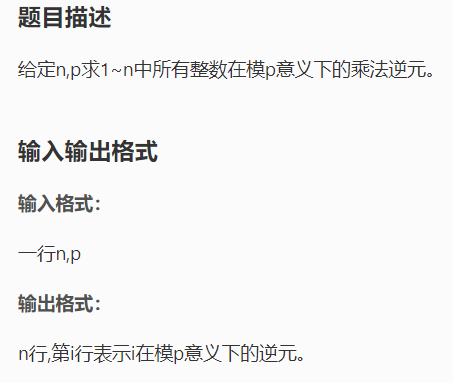
#include <iostream>

#include <array>

int main()

{

int n, p;

 std::cin >> n >> p;

static std::array<long long, 3000001> Arr{ 0 };

Arr[1] = 1;

std::cout << 1 << std::endl;

for (long long i = 2; i <= n; ++i)

{

Arr[i] = static\_cast<long long>((p - p / i) \* Arr[p % i] % p);

std::printf("%lld\n", Arr[i]);

}

return 0;

}

**单源最短路径（优先队列优化）**

#include <iostream>

#include <vector>

#include <queue>

#include <array>

constexpr int MaxN = 500000 + 5;

constexpr int INF = 2147483647;

class Edge

{

public:

int From, To, Dist;

Edge() = default;

Edge(int u, int v, int d) :

From(u), To(v), Dist(d)

{

}

};

std::vector<Edge> Edges;

std::array<std::vector<int>, MaxN> G;

std::array<bool, MaxN> Done{ false };

std::array<int, MaxN> d{ 0 }, p{ 0 };

void AddEdge(int From, int To, int Dist)

{

Edges.emplace\_back(From, To, Dist);

G[From].push\_back(Edges.size() - 1);

}

class HeapNode

{

public:

int d, u;

HeapNode() = default;

HeapNode(int d, int u) :

d(d), u(u)

{

}

bool operator<(const HeapNode &rhs) const

{

return d > rhs.d;

}

};

int Read()

{

int n = 0, k = 1;

char ch = getchar();

while ((ch > '9' || ch < '0') && ch != '-')

{

ch = getchar();

}

if (ch == '-')

{

k = -1;

ch = getchar();

}

while (ch <= '9' && ch >= '0')

{

n = n \* 10 + ch - '0';

ch = getchar();

}

return n \* k;

}

int main()

{

int N, M, S;

N = Read();

M = Read();

S = Read();

for (int i = 0; i < M; ++i)

{

int From, To, Cost;

From = Read();

To = Read();

Cost = Read();

AddEdge(From, To, Cost);

}

std::priority\_queue<HeapNode> Q;

for (int i = 1; i <= N; ++i)

{

d[i] = INF;

}

d[S] = 0;

Q.push(HeapNode(0, S));

while (!Q.empty())

{

auto x = Q.top();

Q.pop();

auto u = x.u;

if (Done[u])

{

continue;

}

Done[u] = true;

for (std::size\_t i = 0; i < G[u].size(); ++i)

{

auto &e = Edges[G[u][i]];

if (d[e.To] > d[u] + e.Dist)

{

d[e.To] = d[u] + e.Dist;

p[e.To] = G[u][i];

Q.push(HeapNode(d[e.To], e.To));

}

}

}

for (int i = 1; i <= N; ++i)

{

std::cout << d[i] << ' ';

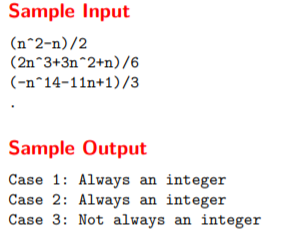
}

std::putchar('\n');

return 0;

}

**多项式解析（最简式）**

#include <iostream>

#include <vector>

#include <cctype>

#include <string>

using ll = long long;

class Polynomial

{

public:

std::vector<int> a, p;

void ParsePolynomial(const std::string &Expr)

{

auto Len = Expr.size();

decltype(Len) i = 0;

while (i < Len)

{

int Sign = 1;

if (Expr[i] == '+')

{

++i;

}

if (Expr[i] == '-')

{

Sign = -1;

++i;

}

int v = 0;

while (i < Len && std::isdigit(Expr[i]))

{

v = v \* 10 + Expr[i++] - '0';

}

if (i == Len)

{

a.push\_back(v);

p.push\_back(0);

}

else

{

if (v == 0)

{

v = 1;

}

v \*= Sign;

if (Expr[++i] == '^')

{

a.push\_back(v);

v = 0;

++i;

while (i < Len && std::isdigit(Expr[i]))

{

v = v \* 10 + Expr[i++] - '0';

}

p.push\_back(v);

}

else

{

a.push\_back(v);

p.push\_back(1);

}

}

}

}

int Mod(int x, int MOD)

{

auto n = a.size();

int Ans = 0;

for (std::size\_t i = 0; i < n; ++i)

{

auto m = a[i];

for (int j = 0; j < p[i]; ++j)

{

m = static\_cast<ll>(m) \* x % MOD;

}

Ans = (static\_cast<ll>(Ans) + m) % MOD;

}

return Ans;

}

};

bool Check(const std::string &Expr)

{

auto p = Expr.find('/');

Polynomial Poly;

Poly.ParsePolynomial(Expr.substr(1, p - 2));

auto D = std::stoi(Expr.substr(p + 1));

for (int i = 1; i <= Poly.p[0] + 1; ++i)

{

if (Poly.Mod(i, D) != 0)

{

return false;

}

}

return true;

}

int main()

{

int NO = 0;

std::string Expr;

while (std::cin >> Expr && Expr[0] != '.')

{

std::cout << "Case " << ++NO << ": ";

if (Check(Expr))

{

std::puts("Always an integer");

}

else

{

std::puts("Not always an integer");

}

}

return 0;

}

**二分图匹配（Dinic）**

#include <iostream>

#include <array>

#include <vector>

#include <algorithm>

#include <queue>

constexpr int INF = 999999999;

int N, M, E;

std::array<std::array<bool, 2002>, 2002> Can{ false };

class Edge

{

public:

int To, Cap, Rev;

Edge() = default;

Edge(int to, int cap, int rev) :

To(to), Cap(cap), Rev(rev)

{

}

};

std::array<std::vector<Edge>, 2002> G;

std::array<int, 2002> Level{ 0 }, Iter{ 0 };

void AddEdge(int From, int To, int Cap)

{

G[From].emplace\_back(To, Cap, G[To].size());

G[To].emplace\_back(From, 0, G[From].size() - 1);

}

void BFS(int s)

{

std::fill(Level.begin(), Level.end(), -1);

std::queue<int> Que;

Level[s] = 0;

Que.push(s);

while (!Que.empty())

{

auto v = Que.front();

Que.pop();

for (std::size\_t i = 0; i < G[v].size(); ++i)

{

auto &e = G[v][i];

if (e.Cap > 0 && Level[e.To] < 0)

{

Level[e.To] = Level[v] + 1;

Que.push(e.To);

}

}

}

}

int DFS(int v, int t, int f)

{

if (v == t)

{

return f;

}

for (int &i = Iter[v]; i < static\_cast<int>(G[v].size()); ++i)

{

auto &e = G[v][i];

if (e.Cap > 0 && Level[v] < Level[e.To])

{

int d = DFS(e.To, t, std::min(f, e.Cap));

if (d > 0)

{

e.Cap -= d;

G[e.To][e.Rev].Cap += d;

return d;

}

}

}

return 0;

}

int MaxFlow(int s, int t)

{

int Flow = 0;

while (1)

{

BFS(s);

if (Level[t] < 0)

{

return Flow;

}

std::fill(Iter.begin(), Iter.end(), 0);

int F;

while ((F = DFS(s, t, INF)) > 0)

{

Flow += F;

}

}

}



int main()

{

N = Read();

M = Read();

E = Read();

for (int i = 0; i < E; ++i)

{

int From, To;

From = Read();

To = Read();

if (To <= M)

{

Can[From][To] = true;

}

}

int S = N + M, T = S + 1;

for (int i = 0; i < N; ++i)

{

AddEdge(S, i, 1);

}

for (int i = 0; i < M; ++i)

{

AddEdge(N + i, T, 1);

}

for (int i = 0; i < N; ++i)

{

for (int j = 0; j < M; ++j)

{

if (Can[i + 1][j + 1])

{

AddEdge(i, N + j, 1);

}

}

}

std::cout << MaxFlow(S, T) << std::endl;

return 0;

}

**高斯消元法**

#include <iostream>

#include <vector>

#include <utility>

#include <cmath>

using vec = std::vector<double>;

using mat = std::vector<vec>;

constexpr double EPS = 1e-8;

vec GaussJordan(const mat &A, const vec &b)

{

auto n = A.size();

mat B(n, vec(n + 1));

for (std::size\_t i = 0; i < n; ++i)

{

for (std::size\_t j = 0; j < n; ++j)

{

B[i][j] = A[i][j];

}

B[i][n] = b[i];

}

for (std::size\_t i = 0; i < n; ++i)

{

auto Privot = i;

for (auto j = i; j < n; ++j)

{

if (std::abs(B[j][i]) > std::abs(B[Privot][i]))

{

Privot = j;

}

}

std::swap(B[i], B[Privot]);

if (std::abs(B[i][i]) < EPS)

{

return vec();

}

for (auto j = i + 1; j <= n; ++j)

{

B[i][j] /= B[i][i];

}

for (std::size\_t j = 0; j < n; ++j)

{

if (i != j)

{

for (auto k = i + 1; k <= n; ++k)

{

B[j][k] -= B[j][i] \* B[i][k];

}

}

}

}

vec x(n);

for (std::size\_t i = 0; i < n; ++i)

{

x[i] = B[i][n];

}

return x;

}

int main()

{

int n;

std::cin >> n;

mat A(n, vec(n));

vec B(n);

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

A[i][j] = Read();

}

B[i] = Read();

}

auto Ans = GaussJordan(A, B);

if (Ans.empty())

{

std::cout << "No Solution" << std::endl;

}

else

{

for (const auto &r : Ans)

{

std::printf("%.2lf\n", r);

}

}

return 0;

}

**滑动窗口（单调队列）**

#include <iostream>

#include <array>

int n, k;

std::array<int, 1000000 + 5> a{ 0 };

void QueryMax()

{

static std::array<int, 1000000 + 5> p{ 0 }, q{ 0 };

int Head = 1, Tail = 0;

for (int i = 1; i <= n; ++i)

{

while (Head <= Tail && q[Tail] <= a[i])

{

--Tail;

}

q[++Tail] = a[i];

p[Tail] = i;

while (p[Head] < i - k + 1)

{

++Head;

}

if (i >= k)

{

std::cout << q[Head] << ' ';

}

}

}

void QueryMin()

{

static std::array<int, 1000000 + 5> p{ 0 }, q{ 0 };

int Head = 1, Tail = 0;

for (int i = 1; i <= n; ++i)

{

while (Head <= Tail && q[Tail] >= a[i])

{

--Tail;

}

q[++Tail] = a[i];

p[Tail] = i;

while (p[Head] < i - k + 1)

{

 ++Head;

}

if (i >= k)

{

std::cout << q[Head] << ' ';

}

}

}

int main()

{

n = Read();

k = Read();

for (int i = 1; i <= n; ++i)

{

a[i] = Read();

}

QueryMin();

std::putchar('\n');

QueryMax();

std::putchar('\n');

return 0;

}

**矩阵加速（数列）**

#include <iostream>

#include <vector>

#include <utility>

using ll = long long;

using vec = std::vector<ll>;

using mat = std::vector<vec>;

constexpr int mod = 1000000007;

mat operator\*(const mat &A, const mat &B)

{

mat C(A.size(), vec(B[0].size()));

for (int i = 0; i < A.size(); ++i)

{

for (int k = 0; k < B.size(); ++k)

{

for (int j = 0; j < B[0].size(); ++j)

{

C[i][j] = (C[i][j] + A[i][k] \* B[k][j]) % mod;

}

}

}

return C;

}

mat operator^(mat &A, ll n)

{

mat B(A.size(), vec(A.size()));

 for (int i = 0; i < A.size(); ++i)

{

B[i][i] = 1;

}

while (n > 0)

{

if (n & 1)

{

B = B \* A;

}

A = A \* A;

n >>= 1;

}

return B;

}

int main()

{

int t;

std::cin >> t;

while (t--)

{

mat A(3, vec(3));

A[0][0] = 1; A[0][1] = 0; A[0][2] = 1;

A[1][0] = 1; A[1][1] = 0; A[1][2] = 0;

A[2][0] = 0; A[2][1] = 1; A[2][2] = 0;

ll n;

std::cin >> n;

if (n <= 3)

{

std::cout << 1 << std::endl;

}

else

{

A = std::move(A ^ (n - 3));

std::cout << (A[0][0] + A[0][1] + A[0][2]) % 1000000007 << std::endl;

}

}

return 0;

}

**矩阵快速幂**

#include <iostream>

#include <vector>

using ll = long long;

using vec = std::vector<ll>;

using mat = std::vector<vec>;

constexpr int mod = 1e9 + 7;

mat operator\*(const mat &A, const mat &B)

{

mat C(A.size(), vec(B[0].size()));

for (ll i = 0; i < A.size(); ++i)

{

for (ll k = 0; k < B.size(); ++k)

{

for (ll j = 0; j < B[0].size(); ++j)

{

C[i][j] = (C[i][j] + A[i][k] \* B[k][j]) % mod;

}

}

}

return C;

}

mat operator^(mat A, ll n)

{

mat B(A.size(), vec(A.size()));

for (ll i = 0; i < A.size(); ++i)

 {

B[i][i] = 1;

}

while (n > 0)

{

if (n & 1)

{

B = B \* A;

}

A = A \* A;

n >>= 1;

}

return B;

}

int main()

{

ll n, k;

std::cin >> n >> k;

mat A(n, vec(n));

for (ll i = 0; i < n; ++i)

{

for (ll j = 0; j < n; ++j)

{

std::cin >> A[i][j];

}

}

A = A ^ k;

for (ll i = 0; i < n; ++i)

{

for (ll j = 0; j < n; ++j)

{

std::cout << A[i][j] << ' ';

}

putchar('\n');

}

return 0;

}

**莫队**

#include <iostream>

#include <array>

#include <utility>

#include <algorithm>

constexpr int MaxN = 50000 + 5;

class Query

{

public:

int Left, Right, ID, Pos;

Query() = default;

Query(int l, int r, int id, int pos) :

Left(l), Right(r), ID(id), Pos(pos)

{

}

bool operator<(const Query &rhs) const noexcept

{

if (Pos == rhs.Pos)

 {

return Right < rhs.Right;

}

else

{

return Pos < rhs.Pos;

}

}

};

int main()

{

int n, m, k;

n = Read();

m = Read();

k = Read();

static std::array<int, MaxN> Num;

for (int i = 1; i <= n; ++i)

{

Num[i] = Read();

}

int s = std::sqrt(n);

static std::array<Query, MaxN> a;

for (int i = 1; i <= m; ++i)

{

int l, r;

l = Read();

r = Read();

a[i] = std::move(Query(l, r, i, l / s));

}

std::sort(a.begin() + 1, a.begin() + m + 1);

static std::array<int, MaxN> cnt{ 0 }, Res;

int Left = 1, Right = 0, Ans = 0;

for (int i = 1; i <= m; ++i)

{

while (Left > a[i].Left)

{

--Left;

++cnt[Num[Left]];

Ans += 2 \* cnt[Num[Left]] - 1;

}

while (Right < a[i].Right)

{

++Right;

++cnt[Num[Right]];

Ans += 2 \* cnt[Num[Right]] - 1;

}

while (Left < a[i].Left)

{

--cnt[Num[Left]];

Ans -= 2 \* cnt[Num[Left]] + 1;

++Left;

}

while (Right > a[i].Right)

{

--cnt[Num[Right]];

Ans -= 2 \* cnt[Num[Right]] + 1;

--Right;

}

Res[a[i].ID] = Ans;

}

for (int i = 1; i <= m; ++i)

{

std::cout << Res[i] << std::endl;

}

return 0;

}

**强连通分量（Tarjan）**

#include <iostream>

#include <vector>

#include <algorithm>

#include <array>

int V;

std::array<std::vector<int>, 50001> G, RG;

std::array<bool, 50001> Used{ false };

std::array<int, 50001> Cmp{ 0 };

std::vector<int> VS;

void AddEdge(int From, int To)

{

G[From].push\_back(To);

RG[To].push\_back(From);

}

void DFS(int v)

{

Used[v] = true;

for (std::size\_t i = 0; i < G[v].size(); ++i)

{

if (!Used[G[v][i]])

{

DFS(G[v][i]);

}

}

VS.push\_back(v);

}

void RDFS(int v, int k)

{

Used[v] = true;

Cmp[v] = k;

for (std::size\_t i = 0; i < RG[v].size(); ++i)

{

if (!Used[RG[v][i]])

{

RDFS(RG[v][i], k);

}

}

}

int SCC()

{

for (int v = 0; v < V; ++v)

{

if (!Used[v])

{

DFS(v);

}

}

std::fill(Used.begin(), Used.end(), false);

int k = 0;

for (int i = static\_cast<int>(VS.size()) - 1; i >= 0; --i)

{

if (!Used[VS[i]])

{

RDFS(VS[i], k++);

}

}

 return k;

}

int main()

{

int N, M;

std::cin >> N >> M;

std::array<int, 50001> A{ 0 };

std::array<int, 50001> B{ 0 };

for (int i = 0; i < M; ++i)

{

std::cin >> A[i] >> B[i];

}

V = N;

for (int i = 0; i < M; ++i)

{

AddEdge(A[i] - 1, B[i] - 1);

}

int n = SCC();

int u = 0, num = 0;

for (int v = 0; v < V; ++v)

{

if (Cmp[v] == n - 1)

{

u = v;

++num;

}

}

std::fill(Used.begin(), Used.end(), false);

RDFS(u, 0);

for (int v = 0; v < V; ++v)

{

if (!Used[v])

{

num = 0;

break;

}

}

std::cout << num << std::endl;

return 0;

}

**三分法（秦九韶公式优化）**

#include <iostream>

#include <array>

std::array<double, 15> a{ 0 };

int n;

double Left, Right, MidL, MidR;

constexpr double EPS = 1e-6;

double f(double x)

{

double Sum = 0.0;

 for (int i = 0; i <= n; ++i)

{

Sum = Sum \* x + a[i];

}

return Sum;

}

int main()

{

std::cin >> n >> Left >> Right;

for (int i = 0; i <= n; ++i)

{

std::cin >> a[i];

}

while (Right - Left > EPS)

{

MidL = (Left + Right) / 2;

MidR = (MidL + Right) / 2;

if (f(MidL) >= f(MidR))

{

Right = MidR;

}

else

{

Left = MidL;

}

}

std::printf("%.5lf\n", Left);

return 0;

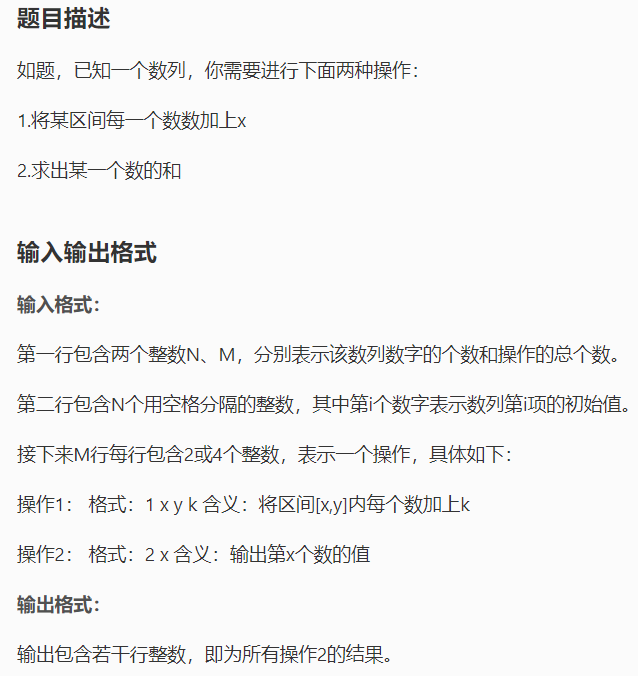
}

**树状数组（差分&单点询问）**

#include <iostream>

#include <array>

std::array<int, 500001> Bit{ 0 };

int n;

int Sum(int i)

{

int s = 0;

while (i > 0)

{

s += Bit[i];

i -= i & -i;

}

return s;

}

void Add(int i, int x)

{

while (i <= n)

{

Bit[i] += x;

i += i & -i;

}

}

int main()

{

int m;

std::cin >> n >> m;

int Last = 0, Now;

for (int i = 1; i <= n; ++i)

{

std::cin >> Now;

Add(i, Now - Last);

Last = Now;

}

while (m--)

{

int Command;

std::cin >> Command;

if (Command == 1)

{

int x, y, k;

std::cin >> x >> y >> k;

Add(x, k);

Add(y + 1, -k);

}

else if (Command == 2)

{

int x;

std::cin >> x;

std::cout << Sum(x) << std::endl;

}

}

return 0;

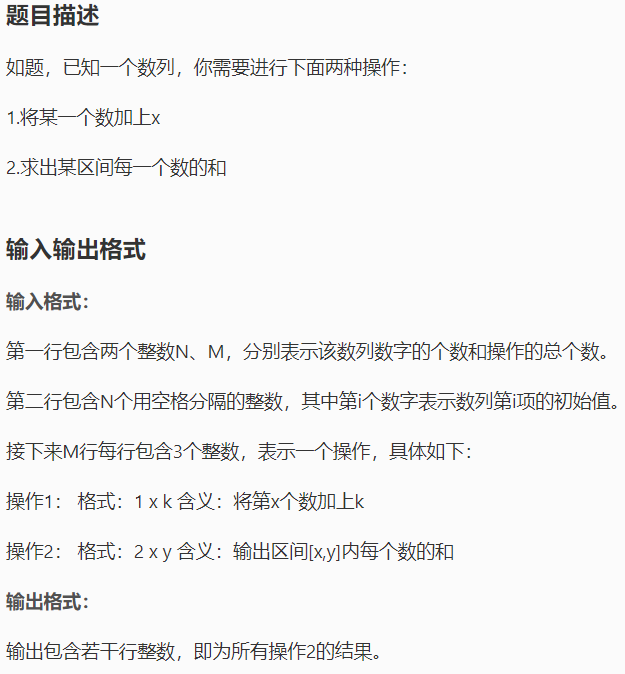
}

**树状数组（区间和）**

#include <iostream>

#include <array>

std::array<int, 500001> Bit{ 0 };

int n;

int Sum(int i)

{

int s = 0;

while (i > 0)

{

s += Bit[i];

i -= i & -i;

}

return s;

}

void Add(int i, int x)

{

while (i <= n)

{

Bit[i] += x;

i += i & -i;

}

}

int main()

{

int m;

std::cin >> n >> m;

int t;

for (int i = 1; i <= n; ++i)

{

t = Read();

Add(i, t);

}

while (m--)

{

int Command;

std::cin >> Command;

if (Command == 1)

{

int x, k;

x = Read();

k = Read();

Add(x, k);

}

else if (Command == 2)

{

int x, y;

x = Read();

y = Read();

std::cout << Sum(y) - Sum(x - 1) << std::endl;

}

}

return 0;

}

**凸包**

#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

constexpr double EPS = 1e-10;

double Add(double a, double b)

{

if (std::abs(a + b) < EPS \* (std::abs(a) + std::abs(b)))

{

return 0;

}

else

{

return a + b;

}

}

struct Point

{

public:

double x, y;

Point() = default;

Point(double X, double Y) :

x(X), y(Y)

{

}

Point operator+(const Point &p)

{

return Point(Add(x, p.x), Add(y, p.y));

}

Point operator-(const Point &p)

{

return Point(Add(x, -p.x), Add(y, -p.y));

}

Point operator\*(double d)

{

return Point(x \* d, y \* d);

}

double Dot(const Point &p)

{

return Add(x \* p.x, y \* p.y);

}

double Det(const Point &p)

{

return Add(x \* p.y, -y \* p.x);

}

};

std::vector<Point> ConvexHull(std::vector<Point> &ps)

{

std::sort(ps.begin(), ps.end(), [](const Point &p, const Point &q)

{

if (p.x != q.x)

{

return p.x < q.x;

}

else

{

return p.y < q.y;

}

});

int k = 0;

std::vector<Point> qs(ps.size() \* 2);

for (std::size\_t i = 0; i < ps.size(); ++i)

{

while (k > 1 && (qs[k - 1] - qs[k - 2]).Det(ps[i] - qs[k - 1]) <= 0)

{

--k;

}

qs[k++] = ps[i];

}

for (int i = static\_cast<int>(ps.size() - 2), t = k; i >= 0; --i)

{

while (k > t && (qs[k - 1] - qs[k - 2]).Det(ps[i] - qs[k - 1]) <= 0)

{

--k;

}

qs[k++] = ps[i];

}

qs.resize(k - 1);

return qs;

}

double Dist(const Point &p, const Point &q)

{

return (p.x - q.x) \* (p.x - q.x) + (p.y - q.y) \* (p.y - q.y);

}

int main()

{

int n;

std::cin >> n;

std::vector<Point> ps;

for (int i = 0; i < n; ++i)

{

double x, y;

std::cin >> x >> y;

ps.emplace\_back(x, y);

}

auto qs = ConvexHull(ps);

double Res = 0.0;

if (qs.size() == 2)

{

std::printf("%.0lf\n", Dist(qs[0], qs[1]));

}

else

{

for (std::size\_t i = 0; i < qs.size(); ++i)

{

for (std::size\_t j = 0; j < i; ++j)

{

Res = std::max(Res, Dist(qs[i], qs[j]));

}

}

std::printf("%.0lf\n", Res);

 }

return 0;

}

**网络最大流（Dinic）**

#include <iostream>

#include <queue>

#include <array>

#include <vector>

#include <algorithm>

#include <climits>

constexpr int INF = INT\_MAX;

int N, M, S, T;

class Edge

{

public:

int To, Cap, Rev;

Edge() = default;

Edge(int to, int cap, int rev) : To(to), Cap(cap), Rev(rev)

{

}

};

std::array<std::vector<Edge>, 10001> G;

std::array<int, 10001> Level, Iter;

void AddEdge(int From, int To, int Cap)

{

G[From].emplace\_back(To, Cap, G[To].size());

G[To].emplace\_back(From, 0, G[From].size() - 1);

}

void BFS(int S)

{

std::fill(Level.begin(), Level.end(), -1);

Level[S] = 0;

std::queue<int> Que;

Que.push(S);

while (!Que.empty())

{

const auto v = Que.front();

Que.pop();

for (int i = 0; i < G[v].size(); ++i)

{

auto &e = G[v][i];

if (e.Cap > 0 && Level[e.To] < 0)

{

Level[e.To] = Level[v] + 1;

Que.push(e.To);

}

}

}

}

int DFS(int v, int t, int f)

{

if (v == t)

{

return f;

}

for (int &i = Iter[v]; i < G[v].size(); ++i)

{

auto &e = G[v][i];

if (e.Cap > 0 && Level[v] < Level[e.To])

{

int d = DFS(e.To, t, std::min(f, e.Cap));

if (d > 0)

{

e.Cap -= d;

G[e.To][e.Rev].Cap += d;

return d;

}

}

}

return 0;

}

int MaxFlow(int s, int t)

{

int Flow = 0;

while (1)

{

BFS(s);

if (Level[t] < 0)

{

return Flow;

}

std::fill(Iter.begin(), Iter.end(), 0);

int f;

while ((f = DFS(s, t, INF)) > 0)

{

Flow += f;

}

}

}

int main()

{

std::cin >> N >> M >> S >> T;

for (int i = 0; i < M; ++i)

{

int u, v, w;

std::cin >> u >> v >> w;

AddEdge(u, v, w);

}

std::cout << MaxFlow(S, T) << std::endl;

return 0;

}

**线段树（RMQ）**

#include <iostream>

#include <array>

#include <climits>

#include <algorithm>

using ll = long long;

constexpr int MaxM = 200000;

std::array<ll, MaxM << 2> Tree{ 0 };

void Update(ll Root, ll l, ll r, ll Pos, ll k)

{

if (l == r)

{

Tree[Root] = k;

return;

}

auto Mid = (l + r) / 2;

if (Mid >= Pos)

{

Update(Root \* 2, l, Mid, Pos, k);

}

else

{

Update(Root \* 2 + 1, Mid + 1, r, Pos, k);

}

Tree[Root] = std::max(Tree[Root \* 2], Tree[Root \* 2 + 1]);

}

ll Query(ll l, ll r, ll ql, ll qr, ll Root)

{

if (l >= ql && r <= qr)

{

return Tree[Root];

}

else if (qr < l || ql > r)

{

return 0;

}

else

{

auto Mid = (l + r) / 2;

return std::max(Query(l, Mid, ql, qr, Root \* 2), Query(Mid + 1, r, ql, qr, Root \* 2 + 1));

}

}

int main()

{

ll M;

ll D;

std::cin >> M >> D;

ll Last = 0;

ll x = 0;

for (ll i = 0; i < M; ++i)

{

char Command;

std::cin >> Command;

if (Command == 'A')

{

ll k;

std::cin >> k;

Update(1, 1, MaxM, ++Last, (x + k) % D);

}

else

{

ll r;

std::cin >> r;

x = Query(1, MaxM, Last - r + 1, Last, 1);

std::cout << x << std::endl;

}

}

return 0;

}

**线段树（区间和&乘）**

#include <iostream>

#include <array>

constexpr int MaxN = 400000 + 5;

using ll = long long;

struct tree

{

public:

ll Val, LazyTagAdd, LazyTagMul;

};

std::array<tree, MaxN> Tree;

std::array<ll, MaxN> Arr{ 0 };

ll Mod;

void Build(ll Root, ll l, ll r)

{

Tree[Root].LazyTagAdd = 0;

Tree[Root].LazyTagMul = 1;

if (l == r)

{

Tree[Root].Val = Arr[l];

}

else

{

auto Mid = (l + r) / 2;

Build(Root \* 2, l, Mid);

Build(Root \* 2 + 1, Mid + 1, r);

Tree[Root].Val = Tree[Root \* 2].Val + Tree[Root \* 2 + 1].Val;

}

Tree[Root].Val %= Mod;

}

void PushDown(ll Root, ll l, ll r)

{

auto Mid = (l + r) / 2;

Tree[Root \* 2].Val = (Tree[Root \* 2].Val \* Tree[Root].LazyTagMul

+ Tree[Root].LazyTagAdd \* (Mid - l + 1)) % Mod;

Tree[Root \* 2 + 1].Val = (Tree[Root \* 2 + 1].Val \* Tree[Root].LazyTagMul

+ Tree[Root].LazyTagAdd \* (r - Mid)) % Mod;

Tree[Root \* 2].LazyTagMul = (Tree[Root \* 2].LazyTagMul \* Tree[Root].LazyTagMul) % Mod;

Tree[Root \* 2 + 1].LazyTagMul = (Tree[Root \* 2 + 1].LazyTagMul \* Tree[Root].LazyTagMul) % Mod;

Tree[Root \* 2].LazyTagAdd = (Tree[Root \* 2].LazyTagAdd \* Tree[Root].LazyTagMul + Tree[Root].LazyTagAdd) % Mod;

Tree[Root \* 2 + 1].LazyTagAdd = (Tree[Root \* 2 + 1].LazyTagAdd \* Tree[Root].LazyTagMul + Tree[Root].LazyTagAdd) % Mod;

Tree[Root].LazyTagAdd = 0;

Tree[Root].LazyTagMul = 1;

}

void UpdateMul(ll Root, ll l, ll r, ll rl, ll rr, ll k)

{

if (l > rr || rl > r)

{

return;

}

if (rl <= l && rr >= r)

{

Tree[Root].Val = (Tree[Root].Val \* k) % Mod;

Tree[Root].LazyTagMul = (Tree[Root].LazyTagMul \* k) % Mod;

Tree[Root].LazyTagAdd = (Tree[Root].LazyTagAdd \* k) % Mod;

return;

}

PushDown(Root, l, r);

auto Mid = (l + r) / 2;

UpdateMul(Root \* 2, l, Mid, rl, rr, k);

UpdateMul(Root \* 2 + 1, Mid + 1, r, rl, rr, k);

Tree[Root].Val = (Tree[Root \* 2].Val + Tree[Root \* 2 + 1].Val) % Mod;

}

void UpdateAdd(ll Root, ll l, ll r, ll rl, ll rr, ll k)

{

if (rr < l || rl > r)

{

return;

}

if (rl <= l && rr >= r)

{

Tree[Root].LazyTagAdd = (Tree[Root].LazyTagAdd + k) % Mod;

Tree[Root].Val = (Tree[Root].Val + k \* (r - l + 1)) % Mod;

return;

}

PushDown(Root, l, r);

auto Mid = (l + r) / 2;

UpdateAdd(Root \* 2, l, Mid, rl, rr, k);

UpdateAdd(Root \* 2 + 1, Mid + 1, r, rl, rr, k);

Tree[Root].Val = (Tree[Root \* 2].Val + Tree[Root \* 2 + 1].Val) % Mod;

}

ll Query(ll Root, ll l, ll r, ll rl, ll rr)

{

if (rr < l || rl > r)

{

return 0;

}

if (rl <= l && r <= rr)

{

return Tree[Root].Val;

}

PushDown(Root, l, r);

auto Mid = (l + r) / 2;

return (Query(Root \* 2, l, Mid, rl, rr) + Query(Root \* 2 + 1, Mid + 1, r, rl, rr)) % Mod;

}

int main()

{

ll N, M;

std::cin >> N >> M >> Mod;

for (ll i = 1; i <= N; ++i)

{

std::cin >> Arr[i];

}

Build(1, 1, N);

while (M--)

{

ll Command;

std::cin >> Command;

ll l, r, k;

if (Command == 1)

{

std::cin >> l >> r >> k;

UpdateMul(1, 1, N, l, r, k);

}

else if (Command == 2)

{

std::cin >> l >> r >> k;

UpdateAdd(1, 1, N, l, r, k);

}

else

{

std::cin >> l >> r;

std::cout << Query(1, 1, N, l, r) << std::endl;

}

}

return 0;

}

**线段树（区间和）**

#include <iostream>

#include <array>

using ll = long long;

std::array<int, 100000> Arr{ 0 };

struct Node

{

ll Val, LazyTag;

};

std::array<Node, 400000> SegTree;

template <typename T>

void Build(int Root, const T &Arr, int Start, int End)

{

SegTree[Root].LazyTag = 0;

if (Start == End)

{

SegTree[Root].Val = Arr[Start];

}

else

{

int Mid = (Start + End) / 2;

Build(Root \* 2, Arr, Start, Mid);

Build(Root \* 2 + 1, Arr, Mid + 1, End);

SegTree[Root].Val = SegTree[Root \* 2].Val + SegTree[Root \* 2 + 1].Val;

}

}

void PushDown(int Root, int Start, int End)

{

if (SegTree[Root].LazyTag != 0)

{

SegTree[Root \* 2].LazyTag += SegTree[Root].LazyTag;

SegTree[Root \* 2 + 1].LazyTag += SegTree[Root].LazyTag;

int Mid = (Start + End) / 2;

SegTree[Root \* 2].Val += SegTree[Root].LazyTag \* (Mid - Start + 1);

SegTree[Root \* 2 + 1].Val += SegTree[Root].LazyTag \* (End - Mid);

SegTree[Root].LazyTag = 0;

}

}

ll Query(int Root, int NStart, int NEnd, int QStart, int QEnd)

{

if (QStart > NEnd || QEnd < NStart)

{

return 0;

}

else if (QStart <= NStart && QEnd >= NEnd)

{

return SegTree[Root].Val;

}

PushDown(Root, NStart, NEnd);

int Mid = (NStart + NEnd) / 2;

return Query(Root \* 2, NStart, Mid, QStart, QEnd)

+ Query(Root \* 2 + 1, Mid + 1, NEnd, QStart, QEnd);

}

void Update(int Root, int NStart, int Nend, int UStart, int UEnd, int AddVal)

{

if (UStart > Nend || UEnd < NStart)

{

return;

}

else if (UStart <= NStart && UEnd >= Nend)

{

SegTree[Root].LazyTag += AddVal;

SegTree[Root].Val += AddVal \* (Nend - NStart + 1);

return;

}

PushDown(Root, NStart, Nend);

int Mid = (NStart + Nend) / 2;

Update(Root \* 2, NStart, Mid, UStart, UEnd, AddVal);

Update(Root \* 2 + 1, Mid + 1, Nend, UStart, UEnd, AddVal);

SegTree[Root].Val = SegTree[Root \* 2].Val + SegTree[Root \* 2 + 1].Val;

}

int main()

{

int n, m;

std::cin >> n >> m;

for (int i = 1; i <= n; ++i)

{

std::cin >> Arr[i];

}

Build(1, Arr, 1, n);

while (m--)

{

int Command;

std::cin >> Command;

if (Command == 1)

{

int x, y, k;

std::cin >> x >> y >> k;

Update(1, 1, n, x, y, k);

}

else if (Command == 2)

{

int x, y;

std::cin >> x >> y;

std::cout << Query(1, 1, n, x, y) << std::endl;

}

}

return 0;

}

**最小生成树**

#include <iostream>

#include <algorithm>

#include <array>

std::array<int, 200001> Par{ 0 }, Rank{ 0 };

template <typename T>

int Find(const T &x)

{

if (Par[x] == x)

{

return x;

}

else

{

return Par[x] = Find(Par[x]);

}

}

template <typename T>

void Init(const T &n)

{

for (int i = 0; i < n; ++i)

{

Par[i] = i;

}

}

template <typename T>

void Unite(T x, T y)

{

x = Find(x);

y = Find(y);

if (x == y)

{

return;

}

if (Rank[x] < Rank[y])

{

Par[x] = y;

}

else

{

Par[y] = x;

if (Rank[x] == Rank[y])

{

++Rank[x];

}

}

}

template <typename T>

bool IsSame(const T &x, const T &y)

{

return Find(x) == Find(y);

}

struct Edge

{

int From, To, Cost;

};

std::array<struct Edge, 200001> Edges;

int main()

{

int N, M;

std::cin >> N >> M;

for (int i = 0; i < M; ++i)

{

std::cin >> Edges[i].From >> Edges[i].To >> Edges[i].Cost;

}

std::sort(Edges.begin(), Edges.begin() + M, [](const Edge &a, const Edge &b)

{

return a.Cost < b.Cost;

});

Init(N);

int Res = 0;

for (int i = 0; i < M; ++i)

{

auto e = Edges[i];

if (!IsSame(e.From, e.To))

{

Unite(e.From, e.To);

Res += e.Cost;

}

}

std::cout << Res << std::endl;

return 0;

}

**最长上升子序列（o(nlogn))**

#include <iostream>

#include <array>

#include <climits>

#include <algorithm>

int main()

{

int n;

std::cin >> n;

static std::array<int, 100002> Origin{ 0 }, Stack{ 0 }, dp{ 0 };

for (int i = 0; i < n; ++i)

{

int Num;

std::cin >> Num;

Origin[Num] = i;

}

for (int i = 0; i < n; ++i)

{

int Num;

std::cin >> Num;

Stack[i] = Origin[Num];

}

std::fill(dp.begin(), dp.begin() + n, INT\_MAX);

for (int i = 0; i < n; ++i)

{

\*std::lower\_bound(dp.begin(), dp.begin() + n, Stack[i]) = Stack[i];

}

std::cout << std::lower\_bound(dp.begin(), dp.begin() + n, INT\_MAX) - dp.begin() << std::endl;

return 0;

}

**// Trie（左儿子右兄弟表示法）：P210(UVa11732)**

#include<cstring>

#include<vector>

using namespace std;

const int maxnode = 4000 \* 1000 + 10;

const int sigma\_size = 26;

// 字母表为全体小写字母的Trie

struct Trie {

int head[maxnode]; // head[i]为第i个结点的左儿子编号

int next[maxnode]; // next[i]为第i个结点的右兄弟编号

char ch[maxnode]; // ch[i]为第i个结点上的字符

int tot[maxnode]; // tot[i]为第i个结点为根的子树包含的叶结点总数

int sz; // 结点总数

long long ans; // 答案

void clear() { sz = 1; tot[0] = head[0] = next[0] = 0; } // 初始时只有一个根结点

// 插入字符串s（包括最后的'\0'），沿途更新tot

void insert(const char \*s) {

int u = 0, v, n = strlen(s);

tot[0]++;

for(int i = 0; i <= n; i++) {

// 找字符a[i]

bool found = false;

for(v = head[u]; v != 0; v = next[v])

if(ch[v] == s[i]) { // 找到了

found = true;

break;

}

if(!found) {

v = sz++; // 新建结点

tot[v] = 0;

ch[v] = s[i];

next[v] = head[u];

head[u] = v; // 插入到链表的首部

head[v] = 0;

}

u = v;

tot[u]++;

}

}

// 统计LCP=u的所有单词两两的比较次数之和

void dfs(int depth, int u) {

if(head[u] == 0) // 叶结点

ans += tot[u] \* (tot[u] - 1) \* depth;

else {

int sum = 0;

for(int v = head[u]; v != 0; v = next[v])

sum += tot[v] \* (tot[u] - tot[v]); // 子树v中选一个串，其他子树中再选一个

ans += sum / 2 \* (2 \* depth + 1); // 除以2是每种选法统计了两次

for(int v = head[u]; v != 0; v = next[v])

dfs(depth+1, v);

}

}

// 统计

long long count() {

ans = 0;

dfs(0, 0);

return ans;

}

};

#include<cstdio>

const int maxl = 1000 + 10; // 每个单词最大长度

int n;

char word[maxl];

Trie trie;

int main() {

int kase = 1;

while(scanf("%d", &n) == 1 && n) {

trie.clear();

for(int i = 0; i < n; i++) {

scanf("%s", word);

trie.insert(word);

}

printf("Case %d: %lld\n", kase++, trie.count());

}

return 0;

}

**// Trie（普通表示法）：P209(LA3492)**

#include<cstring>

#include<vector>

using namespace std;

const int maxnode = 4000 \* 100 + 10;

const int sigma\_size = 26;

// 字母表为全体小写字母的Trie

struct Trie {

int ch[maxnode][sigma\_size];

int val[maxnode];

int sz; // 结点总数

void clear() { sz = 1; memset(ch[0], 0, sizeof(ch[0])); } // 初始时只有一个根结点

int idx(char c) { return c - 'a'; } // 字符c的编号

// 插入字符串s，附加信息为v。注意v必须非0，因为0代表“本结点不是单词结点”

void insert(const char \*s, int v) {

int u = 0, n = strlen(s);

for(int i = 0; i < n; i++) {

int c = idx(s[i]);

if(!ch[u][c]) { // 结点不存在

memset(ch[sz], 0, sizeof(ch[sz]));

val[sz] = 0; // 中间结点的附加信息为0

ch[u][c] = sz++; // 新建结点

}

u = ch[u][c]; // 往下走

}

val[u] = v; // 字符串的最后一个字符的附加信息为v

}

// 找字符串s的长度不超过len的前缀

void find\_prefixes(const char \*s, int len, vector<int>& ans) {

int u = 0;

for(int i = 0; i < len; i++) {

if(s[i] == '\0') break;

int c = idx(s[i]);

if(!ch[u][c]) break;

u = ch[u][c];

if(val[u] != 0) ans.push\_back(val[u]); // 找到一个前缀

}

}

};

#include<cstdio>

const int maxl = 300000 + 10; // 文本串最大长度

const int maxw = 4000 + 10; // 单词最大个数

const int maxwl = 100 + 10; // 每个单词最大长度

const int MOD = 20071027;

int d[maxl], len[maxw], S;

char text[maxl], word[maxwl];

Trie trie;

int main() {

int kase = 1;

while(scanf("%s%d", text, &S) == 2) {

trie.clear();

for(int i = 1; i <= S; i++) {

scanf("%s", word);

len[i] = strlen(word);

trie.insert(word, i);

}

memset(d, 0, sizeof(d));

int L = strlen(text);

d[L] = 1;

for(int i = L-1; i >= 0; i--) {

vector<int> p;

trie.find\_prefixes(text+i, L-i, p);

for(int j = 0; j < p.size(); j++)

d[i] = (d[i] + d[i+len[p[j]]]) % MOD;

}

printf("Case %d: %d\n", kase++, d[0]);

}

return 0;

}

**// 线段树（推平一段区间）：P207(UVa11992)**

// 注意：所有叶子上总是保留set标记而不会被清除（pushdown只能针对非叶结点），因此maintain函数对于叶子来说并不会重复累加addv[o]

// 本程序在query的时候没有进行标记传递（即pushdown），而是用其他方法完成了查询。虽然执行效率提高，但在一定程度上牺牲了可读性

// 有兴趣的读者请参考代码仓库中的uva11992b.cpp，那个写法更具一般性，只是执行效率较低

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxnode = 1<<17;

int \_sum, \_min, \_max, op, x1, x2, y1, y2, x, v;

struct IntervalTree {

int sumv[maxnode], minv[maxnode], maxv[maxnode], setv[maxnode], addv[maxnode];

// 维护信息

void maintain(int o, int L, int R) {

int lc = o\*2, rc = o\*2+1;

if(R > L) {

sumv[o] = sumv[lc] + sumv[rc];

minv[o] = min(minv[lc], minv[rc]);

maxv[o] = max(maxv[lc], maxv[rc]);

}

if(setv[o] >= 0) { minv[o] = maxv[o] = setv[o]; sumv[o] = setv[o] \* (R-L+1); }

if(addv[o]) { minv[o] += addv[o]; maxv[o] += addv[o]; sumv[o] += addv[o] \* (R-L+1); }

}

// 标记传递

void pushdown(int o) {

int lc = o\*2, rc = o\*2+1;

if(setv[o] >= 0) {

setv[lc] = setv[rc] = setv[o];

addv[lc] = addv[rc] = 0;

setv[o] = -1; // 清除本结点标记

}

if(addv[o]) {

addv[lc] += addv[o];

addv[rc] += addv[o];

addv[o] = 0; // 清除本结点标记

}

}

void update(int o, int L, int R) {

int lc = o\*2, rc = o\*2+1;

if(y1 <= L && y2 >= R) { // 标记修改

if(op == 1) addv[o] += v;

else { setv[o] = v; addv[o] = 0; }

} else {

pushdown(o);

int M = L + (R-L)/2;

if(y1 <= M) update(lc, L, M); else maintain(lc, L, M);

if(y2 > M) update(rc, M+1, R); else maintain(rc, M+1, R);

}

maintain(o, L, R);

}

void query(int o, int L, int R, int add) {

if(setv[o] >= 0) {

int v = setv[o] + add + addv[o];

\_sum += v \* (min(R,y2)-max(L,y1)+1);

\_min = min(\_min, v);

\_max = max(\_max, v);

} else if(y1 <= L && y2 >= R) {

\_sum += sumv[o] + add \* (R-L+1);

\_min = min(\_min, minv[o] + add);

\_max = max(\_max, maxv[o] + add);

} else {

int M = L + (R-L)/2;

if(y1 <= M) query(o\*2, L, M, add + addv[o]);

if(y2 > M) query(o\*2+1, M+1, R, add + addv[o]);

}

}

};

const int maxr = 20 + 5;

const int INF = 1000000000;

IntervalTree tree[maxr];

int main() {

int r, c, m;

while(scanf("%d%d%d", &r, &c, &m) == 3) {

memset(tree, 0, sizeof(tree));

for(x = 1; x <= r; x++) {

memset(tree[x].setv, -1, sizeof(tree[x].setv));

tree[x].setv[1] = 0;

}

while(m--) {

scanf("%d%d%d%d%d", &op, &x1, &y1, &x2, &y2);

if(op < 3) {

scanf("%d", &v);

for(x = x1; x <= x2; x++) tree[x].update(1, 1, c);

} else {

\_sum = 0; \_min = INF; \_max = -INF;

for(x = x1; x <= x2; x++) tree[x].query(1, 1, c, 0);

printf("%d %d %d\n", \_sum, \_min, \_max);

}

}

}

return 0;

}

**// 线段树（动态最大连续和）：P201(LA3938)**

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn = 500000 + 10;

const int maxnode = 1000000 + 10;

typedef long long LL;

typedef pair<int,int> Interval;

LL prefix\_sum[maxn];

LL sum(int L, int R) {

return prefix\_sum[R] - prefix\_sum[L-1];

}

LL sum(Interval p) {

return sum(p.first, p.second);

}

Interval better(Interval a, Interval b) {

if(sum(a) != sum(b)) return sum(a) > sum(b) ? a : b;

return a < b ? a : b; // 利用pair自带的字典序

}

int qL, qR;

struct IntervalTree {

int max\_prefix[maxnode];

int max\_suffix[maxnode];

Interval max\_sub[maxnode];

void build(int o, int L, int R) {

if(L == R) {

max\_prefix[o] = max\_suffix[o] = L;

max\_sub[o] = make\_pair(L, L);

} else {

int M = L + (R-L)/2;

// 递归创建子树

int lc = o\*2, rc = o\*2+1;

build(lc, L, M);

build(rc, M+1, R);

// 递推max\_prefix

LL v1 = sum(L, max\_prefix[lc]);

LL v2 = sum(L, max\_prefix[rc]);

if(v1 == v2) max\_prefix[o] = min(max\_prefix[lc], max\_prefix[rc]);

else max\_prefix[o] = v1 > v2 ? max\_prefix[lc] : max\_prefix[rc];

// 递推max\_suffix

v1 = sum(max\_suffix[lc], R);

v2 = sum(max\_suffix[rc], R);

if(v1 == v2) max\_suffix[o] = min(max\_suffix[lc], max\_suffix[rc]);

else max\_suffix[o] = v1 > v2 ? max\_suffix[lc] : max\_suffix[rc];

// 递推max\_sub

max\_sub[o] = better(max\_sub[lc], max\_sub[rc]); // 完全在左子树或者右子树

max\_sub[o] = better(max\_sub[o], make\_pair(max\_suffix[lc], max\_prefix[rc])); // 跨越中线

}

}

Interval query\_prefix(int o, int L, int R) {

if(max\_prefix[o] <= qR) return make\_pair(L, max\_prefix[o]);

int M = L + (R-L)/2;

int lc = o\*2, rc = o\*2+1;

if(qR <= M) return query\_prefix(lc, L, M);

Interval i = query\_prefix(rc, M+1, R);

i.first = L;

return better(i, make\_pair(L, max\_prefix[lc]));

}

Interval query\_suffix(int o, int L, int R) {

if(max\_suffix[o] >= qL) return make\_pair(max\_suffix[o], R);

int M = L + (R-L)/2;

int lc = o\*2, rc = o\*2+1;

if(qL > M) return query\_suffix(rc, M+1, R);

Interval i = query\_suffix(lc, L, M);

i.second = R;

return better(i, make\_pair(max\_suffix[rc], R));

}

Interval query(int o, int L, int R) {

if(qL <= L && R <= qR) return max\_sub[o];

int M = L + (R-L)/2;

int lc = o\*2, rc = o\*2+1;

if(qR <= M) return query(lc, L, M);

if(qL > M) return query(rc, M+1, R);

Interval i1 = query\_prefix(rc, M+1, R); // 右半的前缀

Interval i2 = query\_suffix(lc, L, M); // 左半的后缀

Interval i3 = better(query(lc, L, M), query(rc, M+1, R));

return better(make\_pair(i2.first, i1.second), i3);

}

};

IntervalTree tree;

int main() {

int kase = 0, n, a, Q;

while(scanf("%d%d", &n, &Q) == 2) {

prefix\_sum[0] = 0;

for(int i = 0; i < n; i++) {

scanf("%d", &a);

prefix\_sum[i+1] = prefix\_sum[i] + a;

}

tree.build(1, 1, n);

printf("Case %d:\n", ++kase);

while(Q--) {

int L, R;

scanf("%d%d", &L, &R);

qL = L; qR = R;

Interval ans = tree.query(1, 1, n);

printf("%d %d\n", ans.first, ans.second);

}

}

return 0;

}