MIST_EagleForces Team Notebook

Syed Mafijul Islam, Saif Ahmed,
Rafsan Hasnan Ratul

Contents

1 000

1.1 Remember [9 lines] - 575b8cdd

```
# Don't stuck in one approach
# Bruteforce is sometimes the best way to go
# Reset global variables
# Check for overflow
# Read the problem carefully
# Break the problem into smaller parts (go step by step)
# Check for negative numbers, zero, one or small cases
# For problems like making string A to B, try to work
    backwards from B to A or it may be easier to make
    both A and B to some C
# L > R can occur
```

2 Data Structure

2.1 DSU [31 lines] - ca62a11d

```
class DisjointSet {
  vector<int> par, sz, minElmt, maxElmt, cntElmt;

 public:
  DisjointSet(int n) {
    par.resize(n + 1);
    sz.resize(n + 1, 1);
    minElmt.resize(n + 1);
    maxElmt.resize(n + 1);
    cntElmt.resize(n + 1, 1);
    for (int i = 1; i <= n; i++) par[i] = minElmt[i] =
        maxElmt[i] = i;
  }
  int findUPar(int u) {
    if (u == par[u]) return u;
    return par[u] = findUPar(par[u]);
  }
  void unionBySize(int u, int v) {
    int pU = findUPar(u);
    int pV = findUPar(v);
    if (pU == pV) return;
    if (sz[pU] < sz[pV]) swap(pU, pV);
    par[pV] = pU;
    sz[pU] += sz[pV];
    cntElmt[pU] += cntElmt[pV];
    minElmt[pU] = min(minElmt[pU], minElmt[pV]);
    maxElmt[pU] = max(maxElmt[pU], maxElmt[pV]);
  }
  int getMinElementIntheSet(int u) { return
      minElmt[findUPar(u)]; }
  int getMaxElementIntheSet(int u) { return
      maxElmt[findUPar(u)]; }
  int getNumofElementIntheSet(int u) { return
      cntElmt[findUPar(u)]; }
};
```

2.2 Fenwick Tree [38 lines] - 8d0c1e90

```
int fenwick[N];
void update(int ind, int val) {
  while (ind < N) {
    fenwick[ind] += val;
    ind += ind & -ind;
  }
}
int query(int ind) {
  int sum = 0;
  while (ind > 0) {
    sum += fenwick[ind];
    ind -= ind & -ind;
  }
  return sum;
}

struct Fenwick2D {
  vector<vector<int>> t;
  int n, m;
  Fenwick2D(int n, int m) : n(n), m(m), t(n + 1,
      vector<int>(m + 1, 0)) {}

  void update(int i, int j, int v) {
    for (int x = i; x <= n; x += x & -x)
      for (int y = j; y <= m; y += y & -y) t[x][y] += v;
  }

  int query(int i, int j) {
    int s = 0;
    for (int x = i; x > 0; x -= x & -x)
      for (int y = j; y > 0; y -= y & -y) s += t[x][y];
    return s;
  }

  int query(int r1, int c1, int r2, int c2) {
    return query(r2, c2) - query(r1 - 1, c2) -
        query(r2, c1 - 1) +
          query(r1 - 1, c1 - 1);
  }
};
```

2.3 MO with Update [61 lines] - f9eca766

```
// 1 indexed
// Complexity:O(S × Q + Q × N^2/S^2)
// S = (2*n^2)^(1/3)
const int block_size = 2720;  // 4310 for 2e5
const int mx = 1e5 + 5;
struct Query {
  int L, R, T, id;
  Query() {}
  Query(int _L, int _R, int _T, int _id) : L(_L),
      R(_R), T(_T), id(_id) {}
  bool operator<(const Query& x) const {
    if (L / block_size == x.L / block_size) {
      if (R / block_size == x.R / block_size) return T <
          x.T;
      return R / block_size < x.R / block_size;
    }
    return L / block_size < x.L / block_size;
  }
} Q[mx];
struct Update {
  int pos;
  int old, cur;
  Update() {};
  Update(int _p, int _o, int _c) : pos(_p), old(_o),
      cur(_c) {};
} U[mx];
int ans[mx];
inline void add(int id) {}
inline void remove(int id) {}
inline void update(int id, int L, int R) {
  // If update position is in current range [L,R]
  if (L <= U[id].pos && U[id].pos <= R) {
    remove(U[id].pos);             // Remove old value
    arr[U[id].pos] = U[id].cur;   // Apply update
    add(U[id].pos);                // Add new value
  } else {
    arr[U[id].pos] = U[id].cur;   // Just update array
  }
}
inline void undo(int id, int L, int R) {
  // Similar to update but revert to old value
  if (L <= U[id].pos && U[id].pos <= R) {
    remove(U[id].pos);
    arr[U[id].pos] = U[id].old;
    add(U[id].pos);
  } else {
    arr[U[id].pos] = U[id].old;
  }
}
inline int get() {}
void MO(int nq, int nu) {
  sort(Q + 1, Q + nq + 1);
  int L = 1, R = 0, T = nu;
  for (int i = 1; i <= nq; i++) {
    Query q = Q[i];
    while (T < q.T) update(++T, L, R);
    while (T > q.T) undo(T--, L, R);
    while (L > q.L) add(--L);
    while (R < q.R) add(++R);
    while (L < q.L) remove(L++);
    while (R > q.R) remove(R--);
    ans[q.id] = get();
  }
}
```

2.4 MO [58 lines] - 727b1684

```
const int N = 1e6 + 5;
int arr[N], freq[N], cnt = 0;

void remove(int idx) {
  int v = arr[idx];
  freq[v]--;
  if (freq[v] == 0) cnt--;
}
void add(int idx) {
  int v = arr[idx];
  freq[v]++;
  if (freq[v] == 1) cnt++;
}
int get_answer() {
  return cnt;
}

int block_size = 700;

struct Query {
  int l, r, idx;
  bool operator<(Query other) const {
    if (l / block_size != other.l / block_size) return l
        / block_size < other.l / block_size;
    return r < other.r;
  }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
  vector<int> answers(queries.size());
  sort(queries.begin(), queries.end());

  // TODO: initialize data structure

  int cur_l = 0;
  int cur_r = -1;
  // invariant: data structure will always reflect the
      range [cur_l, cur_r]
  for (Query q : queries) {
    while (cur_l > q.l) {
      cur_l--;
      add(cur_l);
    }
    while (cur_r < q.r) {
      cur_r++;
      add(cur_r);
    }
    while (cur_l < q.l) {
      remove(cur_l);
      cur_l++;
    }
    while (cur_r > q.r) {
      remove(cur_r);
      cur_r--;
    }
    answers[q.idx] = get_answer();
    }
    return answers;
}
```

2.5 Merge Sort Tree [41 lines] - 7c32da88

```
constexpr int N = 100005;
struct Node {
    vector<int> nums;
};
int arr[N];
Node seg[4 * N];

// Combine: Merge two sorted vectors
Node combine(Node& left, Node& right) {
    Node res;
    res.nums.resize(left.nums.size() +
        right.nums.size());
    merge(left.nums.begin(), left.nums.end(),
        right.nums.begin(),
          right.nums.end(), res.nums.begin());
    return res;
}
// O(n log n)
void build(int ind, int low, int high) {
    if (low == high) {
        seg[ind].nums = {arr[low]};  // Single element
        return;
    }
    int mid = (low + high) / 2;
    build(2 * ind, low, mid);
    build(2 * ind + 1, mid + 1, high);
    seg[ind] = combine(seg[2 * ind], seg[2 * ind + 1]);
}
// Query: Count elements <= k in range [l, r]
// O(log^2 n)
int query(int ind, int low, int high, int l, int r, int
    k) {
    if (low > r || high < l) return 0;  // No overlap
    if (low >= l && high <= r) {
        // Entire segment is in query range
        // Count elements <= k in this sorted segment
        auto it = upper_bound(seg[ind].nums.begin(),
            seg[ind].nums.end(), k);
        return it - seg[ind].nums.begin();
    }
    int mid = (low + high) / 2;
    int left = query(2 * ind, low, mid, l, r, k);
    int right = query(2 * ind + 1, mid + 1, high, l, r,
        k);
    return left + right;
}
```

2.6 SQRT Decomposition [69 lines] - a6182018

```
// https://www.spoj.com/problems/GIVEAWAY/
// 0 l r k : count of numbers >= k in [l,r]
// 1 idx val : update a[idx] = val

const int N = 5e5 + 5, BLK = 700;
vector<int> block[N / BLK + 5];
int get_block(int idx) { return idx / BLK; }

void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];
    for (int i = 0; i < n; i++) {
        block[get_block(i)].push_back(a[i]);
    }
    int last_blk = get_block(n - 1);
    for (int i = 0; i <= last_blk; i++) {
        sort(block[i].begin(), block[i].end());
    }
    int q;
    cin >> q;
    while (q--) {
        int ty;
        cin >> ty;
        if (ty == 0) {
            // query
            // O(sqrt(n) log(sqrt(n)))
            int l, r, k;
            cin >> l >> r >> k;
            l--, r--;
            int bl = get_block(l);
            int br = get_block(r);
            int ans = 0;
            if (bl == br) {
                // just brute force
                for (int i = l; i <= r; i++) {
                    if (a[i] >= k) ans++;
                }
            } else {
                for (int i = l; get_block(i) < (bl +
                    1); i++) {
                    if (a[i] >= k) ans++;
                }
                for (int i = r; get_block(i) > (br -
                    1); i--) {
                    if (a[i] >= k) ans++;
                }
                for (int b = bl + 1; b < br; b++) {
                    // sorted vector, [1, 5, 10, 20], k
                        = 6, ans += 2
                    auto it =
                        lower_bound(block[b].begin(),
                        block[b].end(), k);
                    ans += block[b].end() - it;
                }
            }
            cout << ans << '\n';
        } else {
            // update
            // O(sqrt(n) log(sqrt(n)))
            int idx, val;
            cin >> idx >> val;
            idx--;
            int old = a[idx];
            int b = get_block(idx);
            auto it = lower_bound(block[b].begin(),
                block[b].end(), old);
            *it = val;
            sort(block[b].begin(), block[b].end());
            a[idx] = val;
        }
    }
}
```

2.7 Segment Tree Lazy [95 lines] - 3f731717

```
const int N = 1e5 + 5;
i64 arr[N], seg[N << 2], lz[N << 2];

i64 combine(i64 l, i64 r) { return l + r; }
void push(int node, int l, int r) {
    if (lz[node] == 0) return;
    int mid = (l + r) >> 1;
    lz[node << 1] += lz[node];
    seg[node << 1] += lz[node] * (mid - l + 1);
    lz[node << 1 | 1] += lz[node];
    seg[node << 1 | 1] += lz[node] * (r - mid);
    lz[node] = 0;
}
void build(int node, int l, int r) {
    if (l == r) {
        seg[node] = arr[l];
        lz[node] = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(node << 1, l, mid);
    build(node << 1 | 1, mid + 1, r);
    seg[node] = combine(seg[node << 1], seg[node << 1 |
        1]);
}
void update(int node, int l, int r, int ql, int qr, int
    val) {
    if (qr < l || r < ql) return;
    if (ql <= l && r <= qr) {
        seg[node] += 1LL * val * (r - l + 1);  // check
            it!
        lz[node] += val;
        return;
    }
    push(node, l, r);
    int mid = (l + r) >> 1;
    update(node << 1, l, mid, ql, qr, val);
    update(node << 1 | 1, mid + 1, r, ql, qr, val);
    seg[node] = combine(seg[node << 1], seg[node << 1 |
        1]);
}
i64 query(int node, int l, int r, int ql, int qr) {
    if (qr < l || r < ql) return 0;  // check it!
    if (ql <= l && r <= qr) return seg[node];
    push(node, l, r);
    int mid = (l + r) >> 1;
    return combine(query(node << 1, l, mid, ql, qr),
                   query(node << 1 | 1, mid + 1, r, ql,
                       qr));
}

// Range Inversion Count and Toggle Update
void push(int node, int l, int r) {
    if (lz[node] == 0) return;
  // need to update the children while passing the lazy
      value
    int mid = (l + r) >> 1;
    lz[node << 1] ^= lz[node];
    {
        swap(seg[node << 1].one, seg[node << 1].zero);
        int t = seg[node << 1].total;
        int o = seg[node << 1].one;
        int z = seg[node << 1].zero;
        seg[node << 1].ans =
            (t * (t - 1) / 2 - o * (o - 1) / 2 - z * (z
              - 1) / 2) -
            seg[node << 1].ans;
    }
    lz[node << 1 | 1] ^= lz[node];
    {
        swap(seg[node << 1 | 1].one, seg[node << 1 |
            1].zero);
        int t = seg[node << 1 | 1].total;
        int o = seg[node << 1 | 1].one;
        int z = seg[node << 1 | 1].zero;
        seg[node << 1 | 1].ans =
            (t * (t - 1) / 2 - o * (o - 1) / 2 - z * (z
              - 1) / 2) -
            seg[node << 1 | 1].ans;
    }
    lz[node] = 0;
}
void update(int node, int l, int r, int ql, int qr, int
    val) {
    if (qr < l || r < ql) return;
    if (ql <= l && r <= qr) {
        lz[node] ^= val;
        { // need to update the segment now
            swap(seg[node].one, seg[node].zero);
            int t = seg[node].total;
            int o = seg[node].one;
            int z = seg[node].zero;
            seg[node].ans =
                (t * (t - 1) / 2 - o * (o - 1) / 2 - z
                  * (z - 1) / 2) -
                seg[node].ans;
        }
        return;
    }
    push(node, l, r);
    int mid = (l + r) >> 1;
    update(node << 1, l, mid, ql, qr, val);
    update(node << 1 | 1, mid + 1, r, ql, qr, val);

    seg[node] = combine(seg[node << 1], seg[node << 1 |
        1]);
}
```

2.8 Segment Tree [53 lines] - b0adef5e

```
constexpr int N = 100005;
i64 arr[N], seg[4 * N];

i64 combine(i64 l, i64 r) { return l + r; }
void build(int ind, int low, int high) {
  if (low == high) {
    seg[ind] = arr[low];
    return;
  }
  int mid = (low + high) / 2;
  build(2 * ind, low, mid);
  build(2 * ind + 1, mid + 1, high);
  seg[ind] = combine(seg[2 * ind], seg[2 * ind + 1]);
}
i64 query(int ind, int low, int high, int l, int r) {
  if (low >= l && high <= r) return seg[ind];
  if (low > r || high < l) return 0;  // check it!
  int mid = (low + high) / 2;
  i64 left = query(2 * ind, low, mid, l, r);
  i64 right = query(2 * ind + 1, mid + 1, high, l, r);
  return combine(left, right);
}
void update(int ind, int low, int high, int node, int
    val) {
  if (low == high) {
    seg[ind] = val;
    return;
  }
  int mid = (low + high) / 2;
  if (low <= node && node <= mid)
    update(2 * ind, low, mid, node, val);
  else
    update(2 * ind + 1, mid + 1, high, node, val);
  seg[ind] = combine(seg[2 * ind], seg[2 * ind + 1]);
}

// Maximum subarray sum in range [l, r]
struct Node {
  i64 sum, pref, suff, ans;
  Node() {
    sum = 0;
    pref = suff = ans = -inf;
  }
};
Node seg[4 * N];
Node combine(Node l, Node r) {
  Node res;
  res.sum = l.sum + r.sum;
  res.pref = max(l.pref, l.sum + r.pref);
  res.suff = max(r.suff, r.sum + l.suff);
  res.ans = max({l.ans, r.ans, res.pref, res.suff,
      res.sum});
  res.ans = max(res.ans, l.suff + r.pref);
  return res;
}
```

2.9 Special Range Query Example Fixing Right Position

[84 lines] - 5a580ece

```
/*
https://codeforces.com/gym/106107/problem/B
We need to count "good subarrays" [l, r] where:
1. Length >= 2
2. a[l] exists somewhere in b[l...r]
3. a[r] does NOT exist in b[l...r]
*/

#include <bits/stdc++.h>
using namespace std;
#define i64 long long
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

const int N = 5e5 + 5;
i64 fenwick[N];

void update(int ind, int val) {
  while (ind < N) {
    fenwick[ind] += val;
    ind += ind & -ind;
  }
}
i64 query(int ind) {
  i64 sum = 0;
  while (ind > 0) {
    sum += fenwick[ind];
    ind -= ind & -ind;
  }
  return sum;
}
void solve() {
  int n;
  cin >> n;
  vector<int> comp;
  vector<int> a(n + 1), b(n + 1);
  gp_hash_table<int, vector<int>> mp;
  for (int i = 1; i <= n; i++) cin >> a[i];
  for (int i = 1; i <= n; i++) {
    cin >> b[i];
    mp[b[i]].push_back(i);
  }
  vector<int> rend(n + 1, n + 1);
  for (int i = 1; i <= n; i++) {
    if (mp[a[i]].empty()) continue;
    auto it = lower_bound(mp[a[i]].begin(),
        mp[a[i]].end(), i);
    if (it == mp[a[i]].end()) continue;
    rend[i] = max(i + 1, *it); // length >= 2, l < r
  }
  vector<vector<int>> active(n + 2);
  for (int i = 1; i <= n; i++) {
    if (rend[i] <= n) {
      active[rend[i]].push_back(i); // i to rend[i] is a
          segment
    }
  }
  gp_hash_table<int, int> last;
  i64 seg_count = 0, ans = 0;
  for (int r = 1; r <= n; r++) {
    last[b[r]] = r;
    for (auto l : active[r]) {
      update(l, 1);
      seg_count++;
    }
    int last_pos = (last.find(a[r]) != last.end()) ?
        last[a[r]] : 0;
    i64 bad = query(last_pos);
    ans += seg_count - bad;
  }
  cout << ans << '\n';

  // clear fenwick tree
  for (int r = 1; r <= n; r++) {
    for (auto l : active[r]) update(l, -1);
  }
}
int32_t main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t = 1;
  cin >> t;
  for (int i = 1; i <= t; i++) {
    // cout << "Case " << i << ": ";
    solve();
  }
}
```

2.10 Sqrt Tricks [8 lines] - 8f8b610b

```
1. Size of the block is not always Sqrt, adjust it as
    necessary. if o(n/b+b) then take n/b = b and
    calculate b.
2. MO's Algorithm
   *it is possible to solve a Mo problem without any
       remove operation. For L in one block R only
       increases, for every range we can start L from
       the last of that block
3. Sqrt Decomposition by time of queries.
   *keep overall solution and sqrt(n) updates in a
       vector and for a query iterate over all of them,
       when the vector size exceeds sqrt(n) you can add
       these updates with overall solution using o(n)
4. If sum of N positive numbers are S, there are at most
    sqrt(S) distinct values.
5. Randomization
6. Baby step, gaint step
```

2.11 Trie Bit [53 lines] - 0faa3353

```
const int BT = 32;
class Node {
  public:
    Node* child[2];
    int cnt;
    Node() {
      cnt = 0;
      for (int i = 0; i < 2; i++) child[i] = NULL;
    }
};
Node *root = new Node();

void insert(Node* node, int x) {
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1LL;
    if (node->child[r] == NULL) node->child[r] = new
        Node();
    node = node->child[r];
    node->cnt++;
  }
} // insert(root, x);

int query(Node* node, int x) {
  int mx = 0;
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1LL;
    if (node->child[r ^ 1] == NULL || node->child[r ^
        1]->cnt == 0) {
      if (node->child[r] == NULL) return mx;
      node = node->child[r];
    } else {
      mx |= (1 << i);
      node = node->child[r ^ 1];
    }
  }
  return mx;
} // query(root, x), get max xor

void remove(Node* node, int x) {
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1LL;
    if (node->child[r] == NULL) return;
    node = node->child[r];
    node->cnt--;
  }
} // remove(root, x)

void clearTrie(Node* node) {
    if (node == nullptr) return;
    clearTrie(node->child[0]);
    clearTrie(node->child[1]);
    delete node;
}
// clearTrie(root), delete full trie
// root = new Node()
```

3 Dynamic Programming

3.1 All Possible Subarray Sum [6 lines] - 18ee4984

```
bitset<100005> bs = 1;
for (auto i : a) {
  bs |= (bs << i);  // if previous 1 value pos is
      possible now ith bit or ith sm
                    // is also possible
}
cout << bs.count() - 1 << endl;
```

3.2 CHT [48 lines] - bc8514ba

```
#define x first
#define y second
// Warning: replace __int128 with long double if using
    floating point lines or if unsure about the
    compiler
// Queries maximum by default, pass false in constructor
    to query minimum
// Line = pair (m, c) for a line y = mx + c
// Make sure lines are inserted in non-decreasing order
    for maximum queries, and non-increasing order for
    minimum queries
template<typename T, class Line = pair<T, T>> // Make
    sure multiplications fit in T
class CHT {
private:
  vector<Line> lines;
  bool maximum;
public:
  CHT(bool maximum = true) { this->maximum = maximum; }
  void addLine(Line line) {
    if (!maximum) line.first = -line.first, line.second
        = -line.second;
    insert(line);
  }
  T query(const T &x) {
    assert(!lines.empty());

    int L = 0, R = lines.size() - 1;
    while (L != R) {
      int mid1 = L + (R - L) / 3;
      int mid2 = R - (R - L) / 3;
      if (lines[mid1].x * x + lines[mid1].y
          >= lines[mid2].x * x + lines[mid2].y
      ) R = mid2 - 1;
      else L = mid1 + 1;
    }

    T res = lines[L].x * x + lines[L].y;
    return maximum ? res : -res;
  }
private:
  bool bad(const Line &line1, const Line &line2, const
      Line &line3) {
    return __int128(line3.y - line1.y) *
        __int128(line1.x - line2.x)
          <= __int128(line2.y - line1.y) *
              __int128(line1.x - line3.x);
  }
  void insert(const Line &line) {
    while (lines.size() > 0 && lines.back().x == line.x)
        lines.pop_back();
    lines.push_back(line);
    int sz = lines.size();
    while (sz >= 3 && bad(lines[sz - 3], lines[sz - 2],
        lines[sz - 1])) {
      lines.erase(lines.end() - 2);
      sz--;
    }
  }
};
```

3.3 Digit DP [15 lines] - 5ea603bd

```
int f(int pos, int up, int under, int nz) {
  if (pos == n) return 0;
  int& ans = dp[pos][under][up][nz];
  if (ans != -1) return ans;
  int ans = 0;
  int start = up ? 0 : s[pos] - '0';
  int end = under ? 9 : t[pos] - '0';
  for (int d = start; d <= end; d++) {
    int n_up = up || (d > s[pos] - '0');
    int n_under = under || (d < t[pos] - '0');
    int n_nz = nz || (d != 0);
    ans += f(pos + 1, n_under, n_up, n_nz);
  }
  return ans;
}
```

3.4 Divide and Conquer DP [26 lines] - 1949a056

```
ll G,L;///total group,cell size
ll dp[8001][801],cum[8001];
ll C[8001];///value of each cell
inline ll cost(ll l,ll r){
  return(cum[r]-cum[l-1])*(r-l+1);
}
void fn(ll g,ll st,ll ed,ll r1,ll r2){
  if(st>ed)return;
  ll mid=(st+ed)/2,pos=-1;
  dp[mid][g]=inf;
  for(int i=r1;i<=r2;i++){
    ll tcost=cost(i,mid)+dp[i-1][g-1];
    if(tcost<dp[mid][g]){
        dp[mid][g]=tcost,pos=i;
    }
  }
  fn(g,st,mid-1,r1,pos);
  fn(g,mid+1,ed,pos,r2);
}
int main(){
  for(int i=1;i<=L;i++)
    cum[i]=cum[i-1]+C[i];
  for(int i=1;i<=L;i++)
    dp[i][1]=cost(1,i);
  for(int i=2;i<=G;i++)fn(i,1,L,1,L);
}
```

3.5 LCIS o(n*m) [22 lines] - b70a45ab

```
int a[100] = {0}, b[100] = {0}, f[100] = {0};
int n = 0, m = 0;
int main(void) {
  cin >> n;
  for (int i = 1; i <= n; i++) cin >> a[i];
  cin >> m;
  for (int i = 1; i <= m; i++) cin >> b[i];
  for (int i = 1; i <= n; i++) {
    int k = 0;
    for (int j = 1; j <= m; j++) {
      if (a[i] > b[j] && f[j] > k)
        k = f[j];
      else if (a[i] == b[j] && k + 1 > f[j])
        f[j] = k + 1;
    }
  }
  int and = 0;
  for (int i = 1; i <= m; i++)
    if (f[i] > ans) ans = f[i];
  cout << and<< endl;
  return 0;
}
```

3.6 LCS [62 lines] - 2e950914

```
// LCS DP Table and LCS Length
vector<vector<int>> dp(s.size() + 1,
    vector<int>(t.size() + 1));
for (int i = 1; i <= s.size(); i++) {
  for (int j = 1; j <= t.size(); j++) {
    if (s[i - 1] == t[j - 1])
      dp[i][j] = dp[i - 1][j - 1] + 1;
    else
      dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
  }
}
cout << dp[s.size()][t.size()] << "\n";

// Any LCS String
string a;
int i = s.size();
int j = t.size();
while (i > 0 && j > 0) {
  if (s[i - 1] == t[j - 1]) {
    a += s[i - 1];
    i--, j--;
  } else if (dp[i][j - 1] > dp[i - 1][j])
    j--;
  else
    i--;
}
reverse(a.begin(), a.end());
cout << a << "\n";

// Lexicographically smallest LCS
vector<vector<string>> dpp(s.size() + 1,
    vector<string>(t.size() + 1));
for (int i = 1; i <= s.size(); i++) {
  for (int j = 1; j <= t.size(); j++) {
    if (s[i - 1] == t[j - 1])
      dpp[i][j] = dpp[i - 1][j - 1] + s[i - 1];
    else if (dp[i][j - 1] < dp[i - 1][j])
      dpp[i][j] = dpp[i - 1][j];
    else if (dp[i][j - 1] > dp[i - 1][j])
      dpp[i][j] = dpp[i][j - 1];
    else
      dpp[i][j] = min(dpp[i][j - 1], dpp[i - 1][j]);
  }
}
cout << dpp[s.size()][t.size()] << "\n";

// number of distinct LCS sequences
int MOD = 1e9 + 7;
vector<vector<int>> cnt(s.size() + 1,
    vector<int>(t.size() + 1, 1));
for (int i = 1; i <= s.size(); i++) {
  for (int j = 1; j <= t.size(); j++) {
    if (s[i - 1] == t[j - 1])
      cnt[i][j] = cnt[i - 1][j - 1];
    else if (dp[i][j - 1] < dp[i - 1][j])
      cnt[i][j] = cnt[i - 1][j];
    else if (dp[i][j - 1] > dp[i - 1][j])
      cnt[i][j] = cnt[i][j - 1];
    else {
      cnt[i][j] = cnt[i - 1][j] + cnt[i][j - 1];
      if (dp[i][j] == dp[i - 1][j - 1]) cnt[i][j] -=
          cnt[i - 1][j - 1];
      cnt[i][j] = (cnt[i][j] + MOD) % MOD;
    }
  }
}
```

3.7 LIS O(nlogn) with full path [50 lines] - d910146b

```
vector<int> v = {7, 3, 5, 3, 6, 2, 9, 8};
vector<int> seq;
for (auto i : v) {
  auto id = lower_bound(seq.begin(), seq.end(), i);
  if (id == seq.end())
    seq.push_back(i);
  else
    seq[id - seq.begin()] = i;
}
cout << seq.size() << endl;

void lis(vector<int> &v) {
  int n = v.size();
  vector<int> dp(n + 1, 1), hash(n);
  int mx = 1, lastInd = 0;
  for (int i = 0; i < n; i++) {
    hash[i] = i;
    for (int prev = 0; prev < i; prev++) {
      if (v[i] > v[prev] && 1 + dp[prev] > dp[i]) {
        dp[i] = 1 + dp[prev];
        hash[i] = prev;
      }
    }
    if (mx < dp[i]) {
      mx = dp[i];
      lastInd = i;
    }
  }
  vector<int> printSeq;
  printSeq.push_back(v[lastInd]);
  while (hash[lastInd] != lastInd) {
    lastInd = hash[lastInd];
    printSeq.push_back(v[lastInd]);
  }
  reverse(printSeq.begin(), printSeq.end());
  cout << mx << "\n";
  for (int i : printSeq) cout << i << " ";
  cout << "\n";

  // nlogn. segment tree max
  vector<int> dp(n + 1);

  for (int i = 0; i < n; i++) {
    dp[i] = 1;
    int mx = query(0, 1, N, 1, a[i] - 1);
    dp[i] = max(dp[i], mx + 1);
    update(0, 1, N, a[i], dp[i]);
  }
  int ans = *max_element(dp.begin(), dp.end());
}
```

3.8 MCM [26 lines] - e8d610dd

```
const int N = 1005;
vector<int> v;
int dp[N][N], mark[N][N];
int MCM(int i, int j) {
  if (i == j) return dp[i][j] = 0;
  if (dp[i][j] != -1) return dp[i][j];
  int mn = INT_MAX;
  for (int k = i; k < j; k++) {
    int x = mn;
    mn = min(mn, MCM(i, k) + MCM(k + 1, j) + v[i - 1] *
        v[k] * v[j]);
    if (x != mn) mark[i][j] = k;
  }
  return dp[i][j] = mn;
}
void print_order(int i, int j) {
  if (i == j)
    cout << "X" << i;
  else {
    cout << "(";
    print_order(i, mark[i][j]);
    print_order(mark[i][j] + 1, j);
    cout << ")";
  }
}
// memset(dp, -1, sizeof dp);
// print_order(1, n);
```

3.9 Maximum Submatrix [34 lines] - 30548291

```
int a[150][150] = {0};
int c[200] = {0};
int maxarray(int n) {
  int b = 0, sum = -100000000;
  for (int i = 1; i <= n; i++) {
    if (b > 0)
      b += c[i];
    else
      b = c[i];
    if (b > sum) sum = b;
  }
  return sum;
}

int maxmatrix(int n) {
  int sum = -100000000, max = 0;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) c[j] = 0;
    for (int j = i; j <= n; j++) {
      for (int k = 1; k <= n; k++) c[k] += a[j][k];
      max = maxarray(n);
      if (max > sum) sum = max;
    }
  }
  return sum;
}
int main(void) {
  int n = 0;
  cin >> n;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++) cin >> a[i][j];
  cout << maxmatrix(n);
  return 0;
}
```

3.10 SOS DP [63 lines] - 2e55972d

```
// sum over subsets
// fact: need sum of submasks
void SOS(vector<int>& v) {
  const int BITS = log2(*max_element(v.begin(),
      v.end())) + 1;
  vector<int> freq(1 << BITS, 0);
  for (int mask : v) freq[mask]++;

  vector<vector<int>> dp(BITS + 1, vector<int>(1 <<
      BITS, 0));
  for (int mask = 0; mask < 1 << BITS; mask++) {
    dp[0][mask] = freq[mask];
  }

  for (int bits = 1; bits <= BITS; bits++) {
    for (int mask = 0; mask < 1 << BITS; mask++) {
      if ((mask & (1 << (bits - 1))) == 0) {
        dp[bits][mask] = dp[bits - 1][mask];
      } else {
        int other_mask = mask - (1 << (bits - 1));
        dp[bits][mask] = dp[bits - 1][mask] + dp[bits -
            1][other_mask];
      }
    }
  }

  for (int mask : v) cout << dp[BITS][mask] << '\n';
}
// O(BITS * 2 ^ BITS)
// dp[bits][mask] means left most 'bits' of submasks can
    be differ (submask)
// for dp[1][11] 01, 00 are not allow because leftmost 1
    bit can be differ. 10
// and 11 are allowed. for travarsing all the submask of
    a mask we can use
// submask = mask
// do {
// submask = (submask - 1) & mask
// } while (submask)
// O(3 ^ BITS)

const int MAXN = 1 << 20, MLOG = 20;
int dp[MAXN], freq[MAXN];

// Sum over subsets (SOS Down)
void forward1() {
  for (int bit = 0; bit < MLOG; bit++)
    for (int i = 0; i < MAXN; i++)
      if (i & (1 << bit)) dp[i] += dp[i ^ (1 << bit)];
}

void backward1() {
  for (int bit = 0; bit < MLOG; bit++)
    for (int i = MAXN - 1; i >= 0; i--)
      if (i & (1 << bit)) dp[i] -= dp[i ^ (1 << bit)];
}

// Sum over supersets (SOS Up)
void forward2() {
  for (int bit = 0; bit < MLOG; bit++)
    for (int i = MAXN - 1; i >= 0; i--)
      if (i & (1 << bit)) dp[i ^ (1 << bit)] += dp[i];
}

void backward2() {
  for (int bit = 0; bit < MLOG; bit++)
    for (int i = 0; i < MAXN; i++)
      if (i & (1 << bit)) dp[i ^ (1 << bit)] -= dp[i];
}
```

4 Game Theory

4.1 Points to be noted [14 lines] - 85ddc883

```
>[First Write a Brute Force solution]
>Nim = all xor
>Misere Nim = Nim + corner case: if all piles are 1,
    reverse(nim)
>Bogus Nim = Nim
>Staircase Nim = Odd indexed pile Nim (Even indexed pile
    doesnt matter, as one player can give bogus moves to
    drop all even piles to ground)
>Sprague Grundy: [Every impartial game under the normal
    play convention is equivalent to a one-heap game of
    nim]
Every tree = one nim pile = tree root value; tree leaf
    value = 0; tree node value = mex of all child nodes.
[Careful: one tree node can become multiple new tree
    roots(multiple elements in one node), then the value
    of that node = xor of all those root values]
>Hackenbush(Given a rooted tree; cut an edge in one
    move; subtree under that edge gets removed; last
    player to cut wins):
 Colon:  //G(u) = (G(v1) + 1) ⊕ (G(v2) + 1) ⊕ ··· [v1, v2, ···
    are childs of u]
For multiple trees ans is their xor
>Hackenbush on graph (instead of tree given an rooted
    graph):
fusion: All edges in a cycle can be fused to get a tree
    structure; build a super node, connect some single
    nodes with that super node, number of single nodes
    is the number of edges in the cycle.
Sol: [Bridge component tree] mark all bridges, a group
    of edges that are not bridges, becomes one component
    and contributes number of edges to the hackenbush.
    (even number of edges contributes 0, odd number of
    edges contributes 1)
```

5 Geometry

5.1 2D Convex Hull [67 lines] - 0b5204d1

```
struct point {
  ll x, y;
  bool operator==(point const& t) const { return (x ==
      t.x && y == t.y); }
};
int orientation(point a, point b, point c) {
  ll d = (c.y - b.y) * (b.x - a.x) - (c.x - b.x) * (b.y
      - a.y);
  if (d < 0)
    return -1;  // clockwise
  else if (d > 0)
    return 1;  // anticlockwise
  return 0;    // collinear
}

bool clockwise(point a, point b, point c, bool
    include_collinear) {
  int o = orientation(a, b, c);
  if (o < 0) return true;
  if (o > 0) return false;
  return include_collinear;
}
bool collinear(point a, point b, point c) { return
    orientation(a, b, c) == 0; }
bool cmp_points(point& p1, point& p2) {
  return make_pair(p1.x, p1.y) < make_pair(p2.x, p2.y);
}
ll distance_sq(point a, point b) {
  ll d = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y
      - b.y);
  return d;
}

void print(vector<point>& v) {
  for (auto it : v) {
    cout << "(" << it.x << ", " << it.y << ")" << "\n";
  }
}
vector<point> convex_hull(vector<point>& points,
                          bool include_collinear =
                              false) {
  int sz = (int)points.size();
  point p0 = *min_element(points.begin(), points.end(),
      cmp_points);
  vector<point> convex_hull_vector = points;
  sort(convex_hull_vector.begin(),
      convex_hull_vector.end(),
       [&p0](const point& p1, const point& p2) {
         int o = orientation(p0, p1, p2);
         if (o == 0) {
           return distance_sq(p0, p1) < distance_sq(p0,
               p2);
         }
         return o < 0;
       });
  if (include_collinear) {
    int idx = sz - 1;
    while (idx >= 0 &&
           collinear(p0, convex_hull_vector[idx],
               convex_hull_vector.back()))
      idx--;
    reverse(convex_hull_vector.begin() + idx + 1,
        convex_hull_vector.end());
  }
  vector<point> st;
  for (int i = 0; i < sz; i++) {
    while (st.size() >= 2 &&
        !clockwise(st[st.size() - 2], st.back(),
            convex_hull_vector[i],
                 include_collinear)) {
      st.pop_back();
    }
    st.push_back(convex_hull_vector[i]);
  }
  if (!include_collinear && st.size() == 2 && st[0] ==
      st[1]) {
    st.pop_back();
  }
  return st;
}
```

5.2 Computational Geometry [74 lines] - a0efcafa

```
typedef long long int T;
typedef double Tf;
Tf eps = 1e-10;

// int sgn(Tf x)
// {
//     if (fabs(x) < eps) return 0;
//     if (x > 0) return 1;
//     return -1;
// }

int sgn(T x) {
  if (x == 0) return 0;
  if (x > 0)
    return 1;
  else
    return -1;
}
struct point {
  T x, y;
  point(T x = 0, T y = 0) {
    this->x = x;
    this->y = y;
  }
  point operator+(point p) { return point(x + p.x, y +
      p.y); }
  point operator-(point p) { return point(x - p.x, y -
      p.y); }
  Tf len() { return sqrt(x * x + y * y); }
  T norm() { return x * x + y * y; }
  point operator*(T a) { return point(x * a, y * a); }
  point operator/(T a) { return point(x / a, y / a); }
  // integer
  bool operator==(point p) { return p.x == x && p.y ==
      y; }
  // float
  // bool operator==(point p) {
  //     return sgn(p.x - x) == 0 && sgn(p.y - y) == 0;
  // }
  bool operator<(point p) {
    if (x == p.x) return y < p.y;
    return x < p.x;
  }
  // float
  // bool operator<(point p) {
  //     if (sgn(p.x - x) == 0) return sgn(y - p.y) ==
    -1;
  //     return sgn(x - p.x) == -1;
  // }
};
T dot(point a, point b) { return a.x * b.x + a.y * b.y;
    }
T cross(point a, point b) { return a.x * b.y - b.x *
    a.y; }
T orien(point a, point b, point c) { return cross(a -
    b, a - c); }

Tf area(point a, point b, point c) { return
    abs(orien(a, b, c)) / (Tf)2; }

struct segment {
  point a, b;
  bool onSegment(point p) {
    if (sgn(orien(a, b, p)) != 0) return false;
    return sgn(dot(p - a, p - b)) <= 0;
  }
  bool intersection(segment s) {
    if (onSegment(s.a)) return true;
    if (onSegment(s.b)) return true;
    if (s.onSegment(a)) return true;
    if (s.onSegment(b)) return true;
    int s1 = sgn(orien(a, b, s.a));
    int s2 = sgn(orien(a, b, s.b));
    int s3 = sgn(orien(s.a, s.b, a));
    int s4 = sgn(orien(s.a, s.b, b));
    if (s1 * s2 < 0 && s3 * s4 < 0) return true;
    return false;
  }
};
ostream& operator<<(ostream& os, point p) {
  return os << "(" << p.x << "," << p.y << ")";
}
```

6 Graph

6.1 Articulation Bridges [41 lines] - b120c1d4

```
const int N = 2e5 + 5;
vector < pair < int, int >> g[N];
int vis[N], st[N], low[N];
int clk, n;
vector < pair < int, int >> bridges;

void dfs(int u, int p, int edg) {
  vis[u] = 1;
  ++clk;
  st[u] = clk;
  low[u] = clk;
  for (auto[v, id]: g[u]) {
    // handled parallel edges between u, v
    if (v == p && id == edg) continue;
    if (vis[v]) {
      low[u] = min(low[u], st[v]);
    } else {
      dfs(v, u, id);
      low[u] = min(low[u], low[v]);
      if (st[u] < low[v]) {
        // edge u, v has bridge
        bridges.push_back({
          u,
          v
        });
      }
    }
  }
}
void find_bridges() {
  clk = 0;
  for (int i = 1; i <= n; i++) {
    vis[i] = 0;
    st[i] = low[i] = -1;
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, 0, -1);
    }
  }
}
```

6.2 Articulation Points [40 lines] - 94a1ef65

```
const int N = 2e5 + 5;
vector < int > g[N];
int vis[N], st[N], low[N], adjComp[N];
int clk, n;

void dfs(int u, int p) {
  vis[u] = 1;
  ++clk;
  st[u] = clk;
  low[u] = clk;
  int child = 0;
  for (int v: g[u]) {
    if (v == p) continue;
    if (vis[v]) {
      low[u] = min(low[u], st[v]);
    } else {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= st[u] && p != 0) {
        adjComp[u]++;
      }
      child++;
    }
  }
  if (p == 0 && child > 1) {
    adjComp[u] = child - 1;
  }
}
void find_vertices() {
  clk = 0;
  for (int i = 1; i <= n; i++) {
    vis[i] = 0;
    st[i] = low[i] = -1;
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, 0);
    }
  }
}
```

6.3 Bellmanford [22 lines] - b8e770fd

```
void bellmanFord(int num_of_nd, int src) {
  dist[src] = 0;
  for (int step = 0; step < num_of_nd; step) {
    for (int i = 1; i <= num_of_nd; i++) {
      for (auto it : adj[i]) {
        int u = i;
        int v = it.first;
        int wt = it.second;
        if (dist[u] != inf && ((dist[u] + wt) <
            dist[v])) {
          if (step == num_of_nd - 1) {
            cout << "Negative cycle found\n ";
            return;
          }
          dist[v] = dist[u] + wt;
          parent[v] = u;
        }
      }
    }
  }
  for (int i = 1; i <= num_of_nd; i++) cout << dist[i]
      << " ";
  cout << endl;
}
```

6.4 Block Cut Tree [83 lines] - b31a5c97

```
const int N = 2e5 + 5;

vector<int> g[N], bct[2 * N];
int vis[N], low[N], disc[N], is_art[N];
stack<int> stk;
vector<int> block_of[N];
int clk, n, m, bct_nodes, root;

void dfs(int u, int p) {
  vis[u] = 1;
  disc[u] = low[u] = ++clk;
  stk.push(u);
  int child = 0;

  for (int v : g[u]) {
    if (v == p) continue;
    if (!vis[v]) {
      child++;
      dfs(v, u);
      low[u] = min(low[u], low[v]);

      if ((p == -1 && child > 1) || (p != -1 && low[v]
          >= disc[u])) {
        is_art[u] = 1;
        vector<int> block;
        while (stk.top() != v) {
          block.push_back(stk.top());
          stk.pop();
        }
        block.push_back(stk.top());
        stk.pop();
        block.push_back(u);

        int block_id = ++bct_nodes;
        for (int x : block) {
          block_of[x].push_back(block_id);
          if (is_art[x]) {
            bct[x].push_back(block_id);
            bct[block_id].push_back(x);
          }
        }
      }
    } else {
      low[u] = min(low[u], disc[v]);
    }
  }
}

void build_bct() {
  clk = 0;
  bct_nodes = n;
  for (int i = 1; i <= n; i++) {
    vis[i] = is_art[i] = 0;
    block_of[i].clear();
    bct[i].clear();
  }
  for (int i = n + 1; i <= 2 * n; i++) bct[i].clear();

  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, -1);
      if (!stk.empty()) {
        int block_id = ++bct_nodes;
        while (!stk.empty()) {
          int x = stk.top();
          stk.pop();
          block_of[x].push_back(block_id);
          if (is_art[x]) {
            bct[x].push_back(block_id);
            bct[block_id].push_back(x);
          }
        }
      }
    }
  }

  root = n + 1;
  for (int i = 1; i <= n; i++) {
    if (is_art[i]) {
      root = i;
      break;
    }
  }
}
```

6.5 Centroid Decomposition [30 lines] - 2ab608d6

```
const int N = 1e5 + 5;
vector<int> g[N];
int sz[N], dead[N], cenpar[N];
void dfs_sz(int u, int p) {
  sz[u] = 1;
  for (int v : g[u]) {
    if (v == p || dead[v]) continue;
    dfs_sz(v, u);
    sz[u] += sz[v];
  }
}
int find_centroid(int num, int u, int p) {
  for (int v : g[u]) {
    if (v != p && !dead[v] && 2 * sz[v] > num) {
      return find_centroid(num, v, u);
    }
  }
  return u;
}
void decompose(int u, int p) {
  dfs_sz(u, p);
  int cent = find_centroid(sz[u], u, p);
  dead[cent] = true;
  cenpar[cent] = p;
  for (int v : g[cent]) {
    if (!dead[v]) {
      decompose(v, cent);
    }
  }
}
```

6.6 Dijkstra [22 lines] - 3621521c

```
void Dijkstra(int start) {
  // vector<pair<int, int>> adj[N];
  priority_queue<pair<int, int>, vector<pair<int,
      int>>, greater<pair<int, int>>> pq;
  pq.push({0, start});
  while (!pq.empty()) {
    auto it = pq.top();
    pq.pop();
    int wt = it.first;
    int u = it.second;
    if (vis[u])
      continue;
    vis[u] = 1;
    for (pair<int, int> i : adj[u]) {
      int adjWt = i.second;
      int adjNode = i.first;
      if (dist[adjNode] > wt + adjWt) {
        dist[adjNode] = wt + adjWt;
        pq.push({dist[adjNode], adjNode});
      }
    }
  }
}
```

6.7 Eularian Circuit [16 lines] - b13e8b7b

```
unordered_map<int, int> Start, End, Val;
unordered_map<int, pair<int, int>> Range;
int start = 0;
void dfs(int node) {
  visited[node] = true;
  Start[node] = start++;
  for (auto child : adj[node]) {
    if (!visited[child]) dfs(child);
  }
  End[node] = start - 1;
}
dfs(1);
vector<int> FlatArray(start + 5);
for (auto i : Start) {
  FlatArray[i.second] = Val[i.first];
  Range[i.first] = {i.second, End[i.first]};
```

6.8 Floyed Warshall [13 lines] - 7e17addb

```
const int inf = 1e9;
const int MAXN = 505;
int dist[MAXN][MAXN];
void floydWarshall(int n) {
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < n; ++j) {
        if (dist[i][k] < inf && dist[k][j] < inf)
          dist[i][j] = min(dist[i][j], dist[i][k] +
              dist[k][j]);
      }
    }
  }
}
```

6.9 Heavy Light Decomposition [102 lines] - 89c9595e

```
vector<int> g[N];
int par[N], dep[N], sz[N];
void dfs(int u, int p = 0) {
  sz[u] = 1, par[u] = p;
  dep[u] = dep[p] + 1;
  int mx = 0;
  for (auto& v : g[u]) {
    if (v == p) continue;
    dfs(v, u);
    sz[u] += sz[v];
    if (sz[v] > mx) {
      mx = sz[v];
      swap(v, g[u][0]);
    }
  }
}
int T, head[N], st[N], en[N];

void dfs_hld(int u, int p = 0) {
  st[u] = ++T;
  // arr[T] = a[u];
  head[u] = (p != 0 && g[p][0] == u) ? head[p] : u;
  for (auto v : g[u]) {
    if (v == p) continue;
    dfs_hld(v, u);
  }
  en[u] = T;
}
int lca(int a, int b) {
  for (; head[a] != head[b]; b = par[head[b]]) {
    if (dep[head[a]] > dep[head[b]]) swap(a, b);
  }
  if (dep[a] > dep[b]) swap(a, b);
  return a;
}
int n;
// process node u upto it's ancestor a
// if excl is true, a won't process
i64 chain_process(int a, int u, bool excl = false) {
  i64 ans = 0;
  for (; head[u] != head[a]; u = par[head[u]]) {
    ans = ans + query(1, 1, n, st[head[u]], st[u]);
  }
  ans = ans + query(1, 1, n, st[a] + excl, st[u]);
  return ans;
}

// process path from node u to v
// if excl is true, lca won't process
i64 path_process(int a, int b, bool excl = false) {
  i64 ans = 0;
  for (; head[a] != head[b]; b = par[head[b]]) {
    if (dep[head[a]] > dep[head[b]]) swap(a, b);
    ans = ans + query(1, 1, n, st[head[b]], st[b]);
  }
  if (dep[a] > dep[b]) swap(a, b);
  ans = ans + query(1, 1, n, st[a] + excl, st[b]);
  return ans;
}
// update path from node u to v
// if excl is true, lca won't update
void path_update(int a, int b, int val, bool excl =
    false) {
  for (; head[a] != head[b]; b = par[head[b]]) {
    if (dep[head[a]] > dep[head[b]]) swap(a, b);
    update(1, 1, n, st[head[b]], st[b], val);
  }
  if (dep[a] > dep[b]) swap(a, b);
  update(1, 1, n, st[a] + excl, st[b], val);
}
/*
     n is global
     dfs(1);
     head[1] = 1;
     dfs_hld(1);
     build(1, 1, n);
*/
void solve() {
  cin >> n;
  int q;
  cin >> q;
  for (int i = 1; i < n; i++) {
    int u, v, c;
    cin >> u >> v >> c;
    g[u].push_back({v, i});
    g[v].push_back({u, i});
    edge_cost[i] = c;
  }
  dfs(1);
  head[1] = 1;
  dfs_hld(1);
  for (int i = 1; i < n; i++) {
    int u = edge_to_ch[i];
    arr[st[u]] = edge_cost[i];
  }
  build(1, 1, n);
  while (q--) {
    int u, v, val;
    cin >> u >> v >> val;
    path_update(u, v, val, true);
    cout << path_process(u, v, true) << '\n';
  }
}
```

6.10 K'th Shortest path [40 lines] - 5a50a704

```
int m,n,deg[MM],source,sink,K,val[MM][12];
struct edge{
  int v,w;
}adj[MM][500];
struct info{
  int v,w,k;
  bool operator<(const info &b)const{
    return w>b.w;
  }
};
priority_queue<info,vector<info>>Q;
void kthBestShortestPath(){
  int i,j;
  info u,v;
  for(i=0;i<n;i++)
    for(j=0;j<K;j++)val[i][j]=inf;
  u.v=source,u.k=0,u.w=0;
  Q.push(u);
  while(!Q.empty()){
    u=Q.top();
    Q.pop();
    for(i=0;i<deg[u.v];i++){
      v.v=adj[u.v][i].v;
      int cost=adj[u.v][i].w+u.w;
      for(v.k=u.k;v.k<K;v.k++){
        if(cost==inf)break;
        if(val[v.v][v.k]>cost){
          swap(cost,val[v.v][v.k]);
          v.w=val[v.v][v.k];
          Q.push(v);
          break;
        }
      }
      for(v.k++;v.k<K;v.k++){
        if(cost==inf)break;
        if(val[v.v][v.k]>cost)swap(cost, val[v.v][v.k]);
      }
    }
  }
}
```

6.11 Kruskal [13 lines] - 7c746046

```
vector<pair<int, pair<int, int>>> Krushkal(
    vector<pair<int, pair<int, int>>>& edges, int n) {
  sort(edges.begin(), edges.end());
  vector<pair<int, pair<int, int>>> ans;
  DisjointSet D(n);
  for (auto it : edges) {
    if (D.findUPar(it.second.first) !=
        D.findUPar(it.second.second)) {
      ans.push_back({it.first, {it.second.first,
          it.second.second}});
      D.unionBySize(it.second.first, it.second.second);
    }
  }
  return ans;
}
```

6.12 LCA using Segment Tree [89 lines] - fa80d47b

```
struct LCA {
  vector<int> height, euler, first, segtree, parent;
  vector<bool> visited;
  vector<vector<int>> jump;

  int n;

  LCA(vector<vector<int>> &adj, int root = 0) {
    n = adj.size();
    height.resize(n);
    first.resize(n);
    parent.resize(n);
    euler.reserve(n * 2);
    visited.assign(n, false);
    dfs(adj, root);
    int m = euler.size();
    segtree.resize(m * 4);
    build(1, 0, m - 1);

    jump.resize(n, vector<int>(32, -1));

    for(int i=0;i<n;i++) {
      jump[i][0] = parent[i];
    }

    for(int j=1;j<20;j++) {
      for(int i=0;i<n;i++) {
        int mid = jump[i][j-1];
```

```cpp
      if(mid != -1) jump[i][j] = jump[mid][j-1];
    }
  }
}

void dfs(vector<vector<int>> &adj, int node, int h =
    0) {
  visited[node] = true;
  height[node] = h;
  first[node] = euler.size();
  euler.push_back(node);
  for (auto to : adj[node]) {
    if (!visited[to]) {
      parent[to] = node;
      dfs(adj, to, h + 1);
      euler.push_back(node);
    }
  }
}

void build(int node, int b, int e) {
  if (b == e) {
    segtree[node] = euler[b];
  } else {
    int mid = (b + e) / 2;
    build(node << 1, b, mid);
    build(node << 1 | 1, mid + 1, e);
    int l = segtree[node << 1], r = segtree[node << 1
        | 1];
    segtree[node] = (height[l] < height[r]) ? l : r;
  }
}

int query(int node, int b, int e, int L, int R) {
  if (b > R || e < L)
    return -1;
  if (b >= L && e <= R)
    return segtree[node];
  int mid = (b + e) >> 1;

  int left = query(node << 1, b, mid, L, R);
  int right = query(node << 1 | 1, mid + 1, e, L, R);
  if (left == -1)
    return right;
  if (right == -1)
    return left;
  return height[left] < height[right] ? left : right;
}

int lca(int u, int v) {
  int left = first[u], right = first[v];
  if (left > right)
    swap(left, right);
  return query(1, 0, euler.size() - 1, left, right);
}

int kthParent(int u, int k) {
  for(int i = 20; i >= 0; i--) {
    if(k & (1LL << i)) u = jump[u][i];
  }
  return u;
}
};
```

6.13 LCA [40 lines] - e4c6dfdc

```cpp
const int N = 1e5 + 5, LG = 17;
vector<int> g[N];
int depth[N], parent[N][LG];
void dfs(int u, int p) {
  depth[u] = depth[p] + 1;
  parent[u][0] = p;
  for (int i = 1; i < LG; i++) {
    parent[u][i] = parent[parent[u][i - 1]][i - 1];
  }
  for (int v : g[u]) {
    if (v == p) continue;
    dfs(v, u);
  }
}
int lca(int u, int v) {
  if (depth[u] < depth[v]) swap(u, v);
  int diff = depth[u] - depth[v];
  for (int i = 0; i < LG; i++) {
    if (diff >> i & 1) {
      u = parent[u][i];
    }
  }
  if (u == v) return u;
  for (int i = LG - 1; i >= 0; i--) {
    if (parent[u][i] != parent[v][i]) {
      u = parent[u][i];
      v = parent[v][i];
    }
  }
  return parent[u][0];
}
int dist(int u, int v) { return depth[u] + depth[v] - 2
    * depth[lca(u, v)]; }
int kthParent(int u, int k) {
  for (int i = 0; i < LG; i++) {
    if (k >> i & 1) {
      u = parent[u][i];
    }
  }
  return u;
}
```

6.14 SCC [57 lines] - 060054f3

```cpp
vector<bool> visited; // keeps track of which vertices
    are already visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector
    when dfs leaves it.
void dfs(int v, vector<vector<int>> const &adj,
    vector<int> &output) {
  visited[v] = true;
  for (auto u : adj[v])
    if (!visited[u])
      dfs(u, adj, output);
  output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongy connected
    components in G
// output: adj_cond -- adjacency list of G^SCC (by root
    vertices)
void scc(vector<vector<int>> const &adj,
    vector<vector<int>> &components,
    vector<vector<int>> &adj_cond) {
  int n = adj.size();
  components.clear(), adj_cond.clear();

  vector<int> order; // will be a sorted list of G's
      vertices by exit time

  visited.assign(n, false);

  // first series of depth first searches
  for (int i = 0; i < n; i++)
    if (!visited[i])
      dfs(i, adj, order);

  // create adjacency list of G^T
  vector<vector<int>> adj_rev(n);
  for (int v = 0; v < n; v++)
    for (int u : adj[v])
      adj_rev[u].push_back(v);

  visited.assign(n, false);
  reverse(order.begin(), order.end());

  vector<int> roots(n, 0); // gives the root vertex of a
      vertex's SCC

  // second series of depth first searches
  for (auto v : order)
    if (!visited[v]) {
      std::vector<int> component;
      dfs(v, adj_rev, component);
      components.push_back(component);
      int root = *min_element(begin(component),
          end(component));
      for (auto u : component)
        roots[u] = root;
    }

  // add edges to condensation graph
  adj_cond.assign(n, {});
  for (int v = 0; v < n; v++)
    for (auto u : adj[v])
      if (roots[v] != roots[u])
        adj_cond[roots[v]].push_back(roots[u]);
}
```

6.15 SPFA O(V*E) [57 lines] - 7cd7bdcb

```cpp
int q[3001] = {0};  // queue for node
it d[1001] = {0};   // record shortest path
from start to ith node bool f[1001] = {0};
int a[1001][1001] = {0};  // adjacency list
int w[1001][1001] = {0};  // adjacency matrix
int main(void) {
  int n = 0, m = 0;
  cin >> n >> m;
  for (int i = 1; i <= m; i++) {
    int x = 0, y = 0, z = 0;
    cin >> x >> y >> z;
    // node x to node y has weight z
    a[x][0]++;
    a[x][a[x][0]] = y;
    w[x][y] = z;
    /*
    // for undirected graph
     a[x][0]++;
     a[y][a[y][0]]=x;
    w[y][x]=z;
    */
  }
  int s = 0, e = 0;
  cin >> s >> e;  // s: start, e: end
  SPFA(s);
  cout << d[e] << endl;
  return 0;
}
void SPFA(int v0) {
  int t, h, u, v;
  for (int i = 0; i < 1001; i++) d[i] = INT_MAX;
  for (int i = 0; i < 1001; i++) f[i] = false;
  d[v0] = 0;
  h = 0;
  t = 1;
  q[1] = v0;
  f[v0] = true;
  while (h != t) {
    h++;
    if (h > 3000) h = 1;
    u = q[h];
    for (int j = 1; j <= a[u][0]; j++) {
      v = a[u][j];
      if (d[u] + w[u][v] < d[v])  // change to > if
          calculating longest path
      {
        d[v] = d[u] + w[u][v];
        if (!f[v]) {
          t++;
          if (t > 3000) t = 1;
          q[t] = v;
          f[v] = true;
        }
      }
    }
    f[u] = false;
  }
}
```

6.16 Topological Sort [33 lines] - 3ddc7df6

```
vector<int> topologicalSort(int V, vector<vector<int>>&
    g) {
  vector<int> inDegree(V, 0);
  vector<int> result;
  queue<int> q;
  for (int u = 0; u < V; u++) {
    for (int v : g[u]) {
      inDegree[v]++;
    }
  }
  for (int i = 0; i < V; i++) {
    if (inDegree[i] == 0) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    result.push_back(u);
    for (int v : g[u]) {
      inDegree[v]--;
      if (inDegree[v] == 0) {
        q.push(v);
      }
    }
  }
  // You may need to check inDegree of all the vertices
      is 0
  if (result.size() != V) {
    cout << "Graph contains a cycle, topological sort
        not possible!" << endl;
    return {};
  }

  return result;
}
```

6.17 Tree Rerooting [99 lines] - c3cafbac

```
namespace reroot {
const auto exclusive = [](const auto& a, const auto&
    base,
                          const auto& merge_into, int
                              vertex) {
  int n = (int)a.size();
  using Aggregate = std::decay_t<decltype(base)>;
  std::vector<Aggregate> b(n, base);
  for (int bit = (int)std::__lg(n); bit >= 0; --bit) {
    for (int i = n - 1; i >= 0; --i) b[i] = b[i >> 1];
    int sz = n - (n & !bit);
    for (int i = 0; i < sz; ++i) {
      int index = (i >> bit) ^ 1;
      b[index] = merge_into(b[index], a[i], vertex, i);
    }
  }
  return b;
};
// MergeInto : Aggregate * Value * Vertex(int) *
    EdgeIndex(int) -> Aggregate
// Base : Vertex(int) -> Aggregate
// FinalizeMerge : Aggregate * Vertex(int) *
    EdgeIndex(int) -> Value
const auto rerooter = [](const auto& g, const auto&
    base,
                         const auto& merge_into, const
                             auto& finalize_merge) {
  int n = (int)g.size();
  using Aggregate = std::decay_t<decltype(base(0))>;
  using Value =
      std::decay_t<decltype(finalize_merge(base(0), 0,
      0))>;
  std::vector<Value> root_dp(n), dp(n);
  std::vector<std::vector<Value>> edge_dp(n),
      redge_dp(n);

  std::vector<int> bfs, parent(n);
  bfs.reserve(n);
  bfs.push_back(0);
  for (int i = 0; i < n; ++i) {
    int u = bfs[i];
    for (auto v : g[u]) {
      if (parent[u] == v) continue;
      parent[v] = u;
      bfs.push_back(v);
    }
  }
  for (int i = n - 1; i >= 0; --i) {
    int u = bfs[i];
    int p_edge_index = -1;
    Aggregate aggregate = base(u);
    for (int edge_index = 0; edge_index <
        (int)g[u].size(); ++edge_index) {
      int v = g[u][edge_index];
      if (parent[u] == v) {
        p_edge_index = edge_index;
        continue;
      }
      aggregate = merge_into(aggregate, dp[v], u,
          edge_index);
    }
    dp[u] = finalize_merge(aggregate, u, p_edge_index);
  }

  for (auto u : bfs) {
    dp[parent[u]] = dp[u];
    edge_dp[u].reserve(g[u].size());
    for (auto v : g[u]) edge_dp[u].push_back(dp[v]);
    auto dp_exclusive = exclusive(edge_dp[u], base(u),
        merge_into, u);
    redge_dp[u].reserve(g[u].size());
    for (int i = 0; i < (int)dp_exclusive.size(); ++i) {
      auto merge_value =
          finalize_merge(dp_exclusive[i], u, i);
      redge_dp[u].push_back(merge_value);
    }
    root_dp[u] = finalize_merge(
        n > 1 ? merge_into(dp_exclusive[0],
            edge_dp[u][0], u, 0) : base(u), u,
        -1);
    for (int i = 0; i < (int)g[u].size(); ++i) {
      dp[g[u][i]] = redge_dp[u][i];
    }
  }

  return std::make_tuple(std::move(root_dp),
      std::move(edge_dp),
                         std::move(redge_dp));
};
}  // namespace reroot

void solve() {
  using Aggregate = int;
  using Value = int;

  auto base = [](int vertex) -> Aggregate { return 0; };
  auto merge_into = [](Aggregate vertex_dp, Value
      neighbor_dp, int vertex,
                       int edge_index) -> Aggregate {
    return vertex_dp + std::max(neighbor_dp, 0);
  };
  auto finalize_merge = [&](Aggregate vertex_dp, int
      vertex,
                            int edge_index) -> Value {
    return vertex_dp + 2 * color[vertex] - 1;
  };

  auto [reroot_result, edge_dp, redge_dp] =
      reroot::rerooter(g, base, merge_into,
          finalize_merge);
  for (auto dp : reroot_result) {
    std::cout << dp << ' ';
  }
  std::cout << '\n';
}
```

7 Misc

7.1 Bit count in O(1) [5 lines] - 8f60ac14

```
int BitCount(unsigned int u) {
  unsigned int uCount;
  uCount = u - ((u >> 1) & 033333333333) - ((u >> 2) &
      011111111111);
  return ((uCount + (uCount >> 3)) & 030707070707) % 63;
}
```

7.2 Bit hacks [16 lines] - 96eab638

```
# x & -x is the least bit in x.
# iterate over all the subsets of the mask
for (int s=m; ; s=(s-1)&m) {
 ... you can use s ...
 if (s==0)  break;
}
# c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the
next number after x with the same number of bits set.
# __builtin_popcount(x) //number of ones in binary
  __builtin_popcountll(x) // for long long
# __builtin_clz(x) // number of leading zeros
  __builtin_ctz(x) // number of trailing zeros, they
      also have long long version
# bitset<N> b; //N is size of bitset
  b.count(); // return number of bits set O(N / 64)
  b.to_string(); // return string representation of
      bitset
  b.any() // return true if any bit is set O(N / 64)
```

7.3 Bitset C++ [13 lines] - 28766a3c

```
bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_first()<<endl; // prints 1
bs._Find_next(idx). This function returns first set bit
    after index idx.for example:

bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_next(1)<<','<<BS._Find_next(3)<<endl; //
    prints 7,7
So this code will print all of the set bits of BS:

for(int i=BS._Find_first();i< BS.size();i =
    BS._Find_next(i))
    cout<<i<<endl;
//Note that there isn't any set bit after idx,
    BS._Find_next(idx) will return BS.size(); same as
    calling BS._Find_first() when bitset is clear;
```

7.4 Knight Moves [2 lines] - 6c4e6fbc

```
int X[8]={2,1,-1,-2,-2,-1,1,2};
int Y[8]={1,2,2,1,-1,-2,-2,-1};
```

7.5 Random Generator [6 lines] - 2b02ac44

```
#define accuracy
    chrono::steady_clock::now().time_since_epoch().count()
mt19937 rng(accuracy);
int rand(int l, int r) {
  uniform_int_distribution<int> ludo(l, r);
  return ludo(rng);
}
```

7.6 Special DS [14 lines] - 0cb955b1

```
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using o_set = tree<T, null_type,
    less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
// o_set<int> st;
// *(st.find_by_order(index))
// st.order_of_key(value)

// Faster hash
gp_hash_table<int, int> mp;
// Or
unordered_map<int, int> mp;
mp.reserve(n * 2);
mp.max_load_factor(0.7);
```

7.7 Usefull Things [9 lines] - c67addfa

```
C++:
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
ios_base::sync_with_stdio(false), cin.tie(nullptr),
    cout.tie(nullptr);
freopen("input.txt", "r", stdin);
Python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
```

7.8 VScode setup [7 lines] - 3e225a41

```
{
   "key": "f5",
   "command": "workbench.action.terminal.sendSequence",
   "args": {
      "text": "clear && g++
          ${fileBasenameNoExtension}.cpp -o
          ${fileBasenameNoExtension} &&
          ./${fileBasenameNoExtension} < in.txt >
          out.txt\n"
    }
} // need "files.autoSaveDelay": 100
```

7.9 stress [25 lines] - 4a84e7fd

```
#!/usr/bin/env bash
g++ -g "$1".cpp -DONPC -o "$1"
g++ -g "$2".cpp -DONPC -o "$2"
g++ -g "$3".cpp -DONPC -o "$3"
for ((testNum=0;testNum<$4;testNum++))
do
  ./$3 > stdinput
  ./$2 < stdinput > outSlow
  ./$1 < stdinput > outWrong
  H1=`md5sum outWrong`
  H2=`md5sum outSlow`
  if !(cmp -s "outWrong" "outSlow")
  then
      echo "Error found!"
      echo "Input:"
      cat stdinput
      echo "Wrong Output:"
      cat outWrong
      echo "Slow Output:"
      cat outSlow
      exit
  fi
done
echo Passed $4 tests
# ./stress.sh wrong correct gen times
```

7.10 vimrc [12 lines] - 1f47305c

```
"install xclip, vim-gtk3
set nocp ai bs=2 hls ic is lbr ls=2 mouse=a nu ru sc scs
    smd so=3 sw=2 ts=2 rnu
filetype plugin indent on
syn on
map gA m'ggVG"+y''
inoremap {<CR> {<CR>}<Esc>ko
nnoremap = mzgg=G`z
vnoremap <C-k> :s/^\s*\zs/\/\/ /<CR>:nohl<CR>
nnoremap <C-k> :s/^\s*\zs/\/\/ /<CR>:nohl<CR>
vnoremap <C-l> :s/^\s*\zs\/\/ //<CR>:nohl<CR>
nnoremap <C-l> :s/^\s*\zs\/\/ //<CR>:nohl<CR>
autocmd FileType cpp map <F9> :w<CR> :!clear; g++
    --std=c++17 % -DONPC -o %:r && ./%:r<CR>
```

8 Number Theory

8.1 10-ary to m-ary [22 lines] - e6c26ef1

```
char a[16]={'0','1','2','3','4','5','6','7',
     '8','9','A','B','C','D','E','F'};
string tenToM(int n, int m){
    int temp=n;
    string result="";
    while (temp!=0){
        result=a[temp%m]+result;
        temp/=m;
    }
    return result;
}
// m-ary to 10-ary
string num = "0123456789ABCDE";
int mToTen(string n, int m){
    int multi=1;
    int result=0;
    for (int i=n.size()-1; i>=0; i--)    {
        result += num.find(n[i])*multi;
        multi*=m;
    }
    return result;
}
```

8.2 Binary Exponentiation [9 lines] - 8b74ed31

```
int power(long long n, long long k, int mod) {
  int ans = 1 % mod; n %= mod; if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
```

8.3 Catalan Numbers [8 lines] - 17405386

```cpp
void catalan(int n) {
  int res = 1;
  cout << res << " ";
  for (int i = 1; i < n; i++) {
    res = (res * (4 * i - 2)) / (i + 1);
    cout << res << " ";
  }
}
```

8.4 Count 1s 0 to n [9 lines] - 6b5041db

```cpp
int cntOnes(int n) {
  int cnt = 0;
  for (int i = 1; i <= n; i <<= 1) {
    int x = (n + 1) / (i << 1);
    cnt += x * i;
    if ((n + 1) % i && n & i) cnt += (n + 1) % i;
  }
  return cnt;
}
```

8.5 Divisors multiple info [36 lines] - 3aff6f8b

```cpp
constexpr int N = 1000005;

void Divisors(int n) {
  int sum = 1, total = 1;
  int mnP = INT_MAX, mxP = INT_MIN, cntP = 0, totalP =
      0;
  for (int i = 0; i < N && Prime[i] * Prime[i] <= n;
      i++) {
    if (n % Prime[i] == 0) {
      mnP = min(mnP, Prime[i]);
      mxP = max(mnP, Prime[i]);
      int k = 0;
      cntP++;
      while (n % Prime[i] == 0) {
        k++;
        n /= Prime[i];
      }

      sum *= (k + 1);  // NOD
      totalP += k;
      int s = 0, p = 1;
      while (k-- >= 0) {
        s += p;
        p *= Prime[i];
      };
      total *= s;  // SOD
    }
  }
  if (n > 1) {
    cntP++, totalP++;
    sum *= 2;
    total *= (1 + n);
    mnP = min(mnP, n);
    mxP = max(mnP, n);
  }
  cout << mnP << " " << mxP << " " << cntP << " " <<
      totalP << " " << sum << " "
       << total << "\n";
}
```

8.6 Extended GCD [8 lines] - 5998ac10

```cpp
// return {x,y} such that ax+by=gcd(a,b)
int exgcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1; y = 0;    return a;  }
  int g = exgcd(b, a % b, y, x);
  y -= a / b * x;
  return g;
} // bezout's identity: a.x +b.y = gcd(a,b)
```

8.7 Find numbers in L to R divisible by all array eleements [12 lines] - 12567a3c

```cpp
solve(int* arr, int N, int L, int R) {
  int LCM = arr[0];
  for (int i = 1; i < N; i++) {
    LCM = (LCM * arr[i]) / (__gcd(LCM, arr[i]));
  }
  if ((LCM < L && LCM * 2 > R) || LCM > R) {
    return;
  }
  int k = (L / LCM) * LCM;
  if (k < L) k = k + LCM;
  for (int i = k; i <= R; i = i + LCM) cout << i << ' ';
}
```

8.8 GaussElimination [39 lines] - 8a474ce2

```cpp
template<typename ld>
int gauss(vector<vector<ld>>& a, vector<ld>& ans) {
  const ld EPS = 1e-9;
  int n = a.size();///number of equations
  int m = a[0].size() - 1;///number of variables
  vector<int>where(m, -1);///indicates which row
      contains the solution
  int row, col;
  for (col = 0, row = 0;col < m && row < n;++col) {
    int sel = row;///which row contains the maximum
        value/
    for (int i = row + 1;i < n;i++)
      if (abs(a[i][col]) > abs(a[sel][col]))
        sel = i;
    if (abs(a[sel][col]) < EPS) continue;///it's
        basically 0.
    a[sel].swap(a[row]);///taking the max row up
    where[col] = row;
    ld t = a[row][col];
    for (int i = col;i <= m;i++) a[row][i] /= t;
    for (int i = 0;i < n;i++) {
      if (i != row) {
        ld c = a[i][col];
        for (int j = col;j <= m;j++)
          a[i][j] -= a[row][j] * c;
      }
    }
    row++;
  }
  ans.assign(m, 0);
  for (int i = 0;i < m;i++)
    if (where[i] != -1)
      ans[i] = a[where[i]][m] / a[where[i]][i];
  for (int i = 0;i < n;i++) {
    ld sum = 0;
    for (int j = 0;j < m;j++)
      sum += ans[j] * a[i][j];
    if (abs(sum - a[i][m]) > EPS) ///L.H.S!=R.H.S
      ans.clear();//No solution
  }
  return row;
}
```

8.9 GaussMod2 [43 lines] - af4935b7

```cpp
template <typename T>
struct Gauss {
    int bits;
    vector<T> table;
    int rank = 0;  // track number of basis/independent
        vectors
    Gauss() : bits(60) { table = vector<T>(bits, 0); }
    Gauss(int _bits) : bits(_bits) { table =
        vector<T>(bits, 0); }
    // Return rank/size of basis
    int basis_size() { return rank; }
    // Check if x can be obtained from the basis
    bool can(T x) {
        for (int i = bits - 1; i >= 0; i--) {
            if (!(x >> i & 1)) continue;
            if (!table[i]) return false;
            x ^= table[i];
        }
        return true;
    }
    void add(T x) {
        for (int i = bits - 1; i >= 0; i--) {
            if (!(x >> i & 1)) continue;  // Skip if bit
                i is 0
            if (!table[i]) {
                table[i] = x;
                rank++;
                return;
            }
            x ^= table[i];  // Eliminate this bit using
                XOR
        }
        // x became 0 (linearly dependent)
    }
    T get_max_xor() {
        T ans = 0;
        for (int i = bits - 1; i >= 0; i--) {
            if ((ans ^ table[i]) > ans) {
                ans ^= table[i];
            }
        }
        return ans;
    }
};
// Gauss<long long> g(60);
// g.add(x);
// g.get_max_xor();
```

8.10 Largest Power of n! [11 lines] - ffed3d40

```cpp
// Returns largest power of p that divides n!
int largestPower(int n, int p)  {
    int res = 0;

    // Calculate res = n/p + n/(p^2) + n/(p^3) + ....
    while (n > 0) {
        n /= p;
        res += n;
    }
    return res;
}
```

8.11 Leap Year [13 lines] - 8ff2b7cb

```
bool isLeap(int n){
    if (n%100==0)
        if (n%400==0) return true;
        else return false;
    if (n%4==0) return true;
    else return false;
}
// Num of Leap year in between
int calNum(int year) {
    return (year / 4) - (year / 100) +
    (year / 400);
}
int leapNum(int l, int r) { return calNum(r) -
    calNum(--l);}
```

8.12 Logab [3 lines] - 1b0f688c

```
int logab (int a, int b){
  return log2(a) / log2(b);
}
```

8.13 Matrix [105 lines] - 0cf635bb

```
template<typename T>
struct Matrix {
  T MOD = 1e9 + 7;///change if necessary
  T add(T a, T b) const {
    T res = a + b;
    if (res >= MOD) return res - MOD;
    return res;
  }
  T sub(T a, T b) const {
    T res = a - b;
    if (res < 0) return res + MOD;
    return res;
  }
  T mul(T a, T b) const {
    T res = a * b;
    if (res >= MOD) return res % MOD;
    return res;
  }
  int R, C;
  vector<vector<T>>mat;
  Matrix(int _R = 0, int _C = 0) {
    R = _R, C = _C;
    mat.resize(R);
    for (auto& v : mat) v.assign(C, 0);
  }
  void print() {
    for (int i = 0;i < R;i++)
      for (int j = 0;j < C;j++)
        cout << mat[i][j] << " \n"[j == C - 1];
  }
  void createIdentity() {
    for (int i = 0; i < R; i++)
      for (int j = 0; j < C; j++)
        mat[i][j] = (i == j);
  }
  Matrix operator+(const Matrix& o) const {
    Matrix res(R, C);
    for (int i = 0; i < R; i++)
      for (int j = 0; j < C; j++)
        res[i][j] = add(mat[i][j] + o.mat[i][j]);
  }
  Matrix operator-(const Matrix& o) const {
    Matrix res(R, C);
    for (int i = 0; i < R; i++)
      for (int j = 0; j < C; j++)
        res[i][j] = sub(mat[i][j] + o.mat[i][j]);
  }
  Matrix operator*(const Matrix& o) const {
    Matrix res(R, o.C);
    for (int i = 0; i < R; i++)
      for (int j = 0; j < o.C; j++)
        for (int k = 0;k < C;k++)
          res.mat[i][j] = add(res.mat[i][j],
            mul(mat[i][k], o.mat[k][j]));
    return res;
  }
  Matrix pow(long long x) {
    Matrix res(R, C);
    res.createIdentity();
    Matrix<T> o = *this;
    while (x) {
      if (x & 1) res = res * o;
      o = o * o;
      x >>= 1;
    }
    return res;
  }
  Matrix inverse()///Only square matrix && non-zero
      determinant
  {
    Matrix res(R, R + R);
    for (int i = 0;i < R;i++) {
      for (int j = 0;j < R;j++)
        res.mat[i][j] = mat[i][j];
      res.mat[i][R + i] = 1;
    }
    for (int i = 0;i < R;i++) {
      ///find row 'r' with highest value at [r][i]
      int tr = i;
      for (int j = i + 1;j < R;j++)
        if (abs(res.mat[j][i]) > abs(res.mat[tr][i]))
          tr = j;
      ///swap the row
      res.mat[tr].swap(res.mat[i]);
      ///make 1 at [i][i]
      T val = res.mat[i][i];
      for (int j = 0;j < R + R;j++) res.mat[i][j] /=
          val;
      ///eliminate [r][i] from every row except i.
      for (int j = 0;j < R;j++) {
        if (j == i) continue;
        for (int k = R + R - 1;k >= i;k--) {
          res.mat[j][k] -= res.mat[i][k] * res.mat[j][i]
              / res.mat[i][i];
        }
      }
    }
    Matrix ans(R, R);
    for (int i = 0;i < R;i++)
      for (int j = 0;j < R;j++)
        ans.mat[i][j] = res.mat[i][R + j];
    return ans;
  }
};
// Matrix<long long> base(2, 2);
// base.mat[0][0] = 1LL; base.mat[0][1] = 1LL;
// base.mat[1][0] = 1LL; base.mat[1][1] = 0LL;
// Matrix<long long> result = base.pow(n - 1);
// cout << result.mat[0][0] << '\n'; // nth Fibonacci
    number
```

8.14 Miller-Rabin-Pollard-Rho [36 lines] - c62079b7

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
    if (e & 1) result = (u128)result * base % mod;
    base = (u128)base * base % mod;
    e >>= 1;
  }
  return result;
}
bool check_composite(u64 n, u64 a, u64 d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1) return false;
  }
  return true;
};
bool MillerRabin(u64 n, int iter = 5) {  // returns true
    if n is probably prime, else returns false.
  if (n < 4) return n == 2 || n == 3;
  int s = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    s++;
  }

  for (int i = 0; i < iter; i++) {
    int a = 2 + rand() % (n - 3);
    if (check_composite(n, a, d, s)) return false;
  }
  return true;
}
```

8.15 Mobius [12 lines] - f7c84b43

```
const int N = 11;
int mob[N];
void mobius() {
  mob[1] = 1;
  for (int i = 2; i < N; i++){
    mob[i]--;
    for (int j = i + i; j < N; j += i) {
      mob[j] -= mob[i];
    }
  }
}
```

8.16 Mod Inverse [5 lines] - 6e1326cf

```
int modInv(int a, int m) {
    int x, y; //if g==1 Inverse doesn't exist
    int g = gcdExt(a, m, x, y);
    return (x % m + m) % m;
}
```

8.17 Modular Operations [12 lines] - da7829be

```
int add(int x, int y) {
    x += y;
    if (x >= mod) x -= mod;
    if (x < 0) x += mod;
    return x;
}
int mul(int x, int y) {
    return x * 1LL * y % mod;
}
int divide(int x, int y) {
    return mul(x, power(y, mod - 2));
}
```

8.18 NOD of n! [13 lines] - e777f444

```
ull factorialDivisors(ull n) {
  ull result = 1;
  for (int i = 0; i < allPrimes.size(); i++) {
    ull p = allPrimes[i];
    ull exp = 0;
    while (p <= n) {
      exp = exp + (n / p);
      p = p * allPrimes[i];
    }
    result = result * (exp + 1);
  }
  return result;
}
```

8.19 No of Digits in n! in base B [7 lines] - 1f8efcc5

```
ll NoOfDigitInNFactInBaseB(ll N,ll B){
  ll i;
  double ans=0;
  for(i=1;i<=N;i++)ans+=log(i);
  ans=ans/log(B),ans=ans+1;
  return(ll)ans;
}
```

8.20 Principle of Inclusion Exclusion [21 lines] - cb2ccfa4

```
// count the numbers between 1 and n (inclusive) that
    are not divisible by any
// of the integers in the given array a
// is n - (div by one number - div by two number + div
    by three number - ...)
int PIE(int div[], int n, int num) {
  int sum = 0;
  for (int msk = 1; msk < (1 << n); ++msk) {
    int bit_cnt = 0;
    int cur_lcm = 1;
    for (int i = 0; i < n; ++i) {
      if (msk & (1 << i)) {
        ++bit_cnt;
        cur_lcm = LCM(cur_lcm, div[i]);
      }
    }
    int cur = num / cur_lcm;
    if (bit_cnt & 1) sum += cur;
    else sum -= cur;
  }
  return num - sum;
}
```

8.21 Print Calander of Any Year [44 lines] - 1a53cdae

```
int dayNumber(int day, int month, int year) {
  static int t[] = {0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4};
  year -= month < 3;
  return (year + year / 4 - year / 100 + year / 400 +
      t[month - 1] + day) % 7;
}
string getMonthName(int monthNumber) {
  string months[] = {"January",   "February", "March",
      "April",
                     "May",        "June",      "July",
                         "August",
                     "September", "October",
                         "November", "December"};
  return (months[monthNumber]);
}
int numberOfDays(int monthNumber, int year) {
  if (monthNumber == 1 && isLeapYear(year)) return 29;
  int monthDays[] = {31, 28, 31, 30, 31, 30, 31, 31,
      30, 31, 30, 31};
  return (monthDays[monthNumber]);
}
void printCalendar(int year) {
  printf("       Calendar - %d\n\n", year);
  int days;
  int current = dayNumber(1, 1, year);
  // i|-> Iterate through all the months
  // j|-> Iterate through all the days of
  the month - i for (int i = 0; i < 12; i++) {
    days = numberOfDays(i, year);
    cout << "          |" << getMonthName(i).c_str() <<
        "|" << endl;
        printf(" Sun Mon Tue Wed Thu Fri
             Sat\n");
        int k;
        for (k = 0; k < current; k++)
            printf("    ");
        for (int j = 1; j <= days; j++) {
      printf("%4d", j);
      if (++k > 6) {
        k = 0;
        cout << endl;
      }
        }
        if (k)
            cout << endl;
        cout <<
      "----------------------------\n";
        current = k;
  }
}
```

8.22 SOD, NOD Upto N [30 lines] - aa4e95d9

```
int SOD_upto_N(int N) { // sqrt(N)
  int res = 0;
  for (int i = 1; i <= N; i++) {
    int j = N / (N / i);  // Last number with same
        quotient
    res += (N / i) * (i + j) * (j - i + 1) / 2;
    i = j;
  }
  return res;
}
int NOD_upto_N(ll N) { // sqrt(N)
  int res = 0;
  int u = sqrt(N);
  for (int i = 1; i <= u; i++) {
    res += (N / i) - i;
  }
  res = res * 2 + u;
  return res;
}
void NOD_per_number_upto_N(int n) { // n loglog(n)
  for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j += i) {
      d[j]++;
      // d[j] += i // for sum of divisors
    }
  }
  for (int i = 1; i <= n; i++) {
    cout << d[i] << ' ';
  }
  return 0;
}
```

8.23 SPF [10 lines] - 802defe6

```
int spf[N];
void buildSpf() {
  for (int i = 2; i < N; i++) {
    spf[i] = i;
  }
  for (int i = 2; 1LL * i * i < N; i++)
    if (spf[i] == i)
      for (int j = i * i; j < N; j += i)
        if (spf[j] == j) spf[j] = i;
}
```

8.24 Segmented Sieve [17 lines] - b1ac5ad7

```
void segmentedSieve(int L, int R) {
  bool isPrime[R - L + 1];
  for (int i = 0; i <= R - L + 1; i++) isPrime[i] =
      true;
  if (L == 1) isPrime[0] = false;
  for (int i = 0; primes[i] * primes[i] <= R; i++) {
    int curPrime = primes[i];
    int base = curPrime * curPrime;
    if (base < L) {
      base = ((L + curPrime - 1) / curPrime) * curPrime;
    }
    for (int j = base; j <= R; j += curPrime) isPrime[j
        - L] = false;
  }
  for (int i = 0; i <= R - L; i++) {
    if (isPrime[i] == true) cout << L + i << '\n';
  }
  cout << '\n';
}
```

8.25 Sieve Phi Mobius [13 lines] - fd92a8a5

```
const int N = 5000005;
int phi[N];
unsigned long long phiSum[N];
void phiCalc() {
  for (int i = 1; i < N; i++) phi[i] = i;
  for (int i = 2; i < N; i++) {
  if (phi[i] == i) {
    for (int j = i; j < N; j += i) {
      phi[j] -= phi[j] / i;    }  }  }
  for (int i = 2; i < N; i++) {
  phiSum[i] = (unsigned long long)phi[i] * (unsigned
      long long)phi[i] + phiSum[i - 1];
  }
}
```

8.26 Sieve upto 1e9 [71 lines] - 5fa0292a

```
vector<int> sieve(const int N, const int Q = 17, const
    int L = 1 << 15) {
  static const int rs[] = {1, 7, 11, 13, 17, 19, 23,
      29};
  struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
  };
  auto approx_prime_count = [] (const int N) -> int {
    return N > 60184 ? N / (log(N) - 1.1)
                     : max(1., N / (log(N) - 1.11)) + 1;
  };
  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i) if (isp[i]) {
    for (int j = i * i; j <= v; j += i) isp[j] = false;
  }
  const int rsize = approx_prime_count(N + 30);
  vector<int> primes = {2, 3, 5}; int psize = 3;
  primes.resize(rsize);

  vector<P> sprimes; size_t pbeg = 0;
  int prod = 1;
  for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg, primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
      int j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }

  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi]; const int p = pp.p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (int i = pp.pos[t]; i < prod; i += p) pre[i]
          &= m;
    }
  }
  const int block_size = (L + prod - 1) / prod * prod;
  vector<unsigned char> block(block_size); unsigned
      char* pblock = block.data();
  const int M = (N + 29) / 30;

  for (int beg = 0; beg < M; beg += block_size, pblock
      -= block_size) {
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock + i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi < sprimes.size(); ++pi) {
      auto& pp = sprimes[pi];
      const int p = pp.p;
      for (int t = 0; t < 8; ++t) {
        int i = pp.pos[t]; const unsigned char m = ~(1
            << t);
        for (; i < end; i += p) pblock[i] &= m;
        pp.pos[t] = i;
      }
    }
    for (int i = beg; i < end; ++i) {
      for (int m = pblock[i]; m > 0; m &= m - 1) {
        primes[psize++] = i * 30 + rs[__builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N) --psize;
  primes.resize(psize);
  return primes;
}
```

8.27 Sieve [17 lines] - bc172e7b

```
const int N = 1e7 + 3;
vector<int> primes;
int notprime[N];
void sieve() {
  primes.push_back(2);
  for (int i = 4; i < N; i += 2) {
    notprime[i] = true;
  }
  for (int i = 3; i < N; i += 2) {
    if (!notprime[i]) {
      primes.push_back(i);
      for (int j = i * i; j < N; j += 2 * i) {
        notprime[j] = true;
      }
    }
  }
}
```

8.28 String Mod [5 lines] - fce25f9e

```
int mod(string& num, int a) {
  int res = 0;
  for (int i = 0; i < num.length(); i++) res = (res * 10
      + num[i] - '0') % a;
  return res;
}
```

8.29 nCr [39 lines] - 9bd950f6

```
const int N = 1e6 + 5, mod = 1e9 + 7;
int fact[N], ifact[N];

void preFact() {
    fact[0] = 1;
    for (int i = 1; i < N; i++) {
        fact[i] = 1LL * fact[i - 1] * i % mod;
    }
    ifact[N - 1] = power(fact[N - 1], mod - 2);
    for (int i = N - 2; i >= 0; i--) {
        ifact[i] = 1LL * ifact[i + 1] * (i + 1) % mod;
    }
}
int nCr(int n, int r) {
    if (n < r || n < 0) return 0;
    return 1LL * fact[n] * ifact[r] % mod * ifact[n - r]
        % mod;
}
// more space less time
const int MOD = 1e9 + 7;
const int MAX = 1e7+10;
vector<int> fact(MAX), inv(MAX);
void factorial(){
    fact[0] = 1;
    for (int i = 1; i < MAX; i++)
        fact[i] = (i * fact[i - 1]) % MOD;
}
binaryExp(int a, int n, int M = MOD); //needs to
    implement
void inverse(){
    for (int i = 0; i < MAX; ++i)
        inv[i] = bigmod(fact[i], MOD - 2);
}
int nCr(int a, int b){
    if (a < b or a < 0 or b < 0)
        return 0;
    int de = (inv[b] * inv[a - b]) % MOD;
    return (fact[a] * de) % MOD;
}
// nCr ends here
int ModInv(int a, int M){ return bigmod(a, M - 2, M);}
```

9 Some Algorithms

9.1 2D Prefix Sum [2 lines] - 00be5300

```
pref[i][j] = a[i][j] + pref[i - 1][j] + pref[i][j - 1]
    - pref[i - 1][j - 1];
Sum of region = pref[row2][col2] - pref[row2][col1 - 1]
    - pref[row1 - 1][col2] + pref[row1 - 1][col1 - 1];
```

9.2 Big Integer [92 lines] - ec448d52

```
struct BigInteger {
  string str;
  // Constructor to initialize
  // BigInteger with a string
  BigInteger(string s) { str = s; }
  // Overload + operator to add
  // two BigInteger objects
  BigInteger operator+(const BigInteger& b) {
    string a = str, c = b.str;
    int alen = a.length(), clen = c.length();
    int n = max(alen, clen);
    if (alen > clen)
      c.insert(0, alen - clen, '0');
    else if (alen < clen)
      a.insert(0, clen - alen, '0');
    string res(n + 1, '0');
    int carry = 0;
    for (int i = n - 1; i >= 0; i--) {
        int digit=(a[i -'0')+(c[i]-'0')
    +carry;
        carry = digit / 10;
        res[i + 1] = digit % 10 + '0';
    }
    if (carry == 1) {
      res[0] = '1';
      return BigInteger(res);
    } else
      return BigInteger(res.substr(1));
  }

  // Overload - operator to subtract
  // first check which number is greater and then
      subtract
  BigInteger operator-(const BigInteger& b) {
    string a = str;
    string c = b.str;
    int alen = a.length(), clen = c.length();
    int n = max(alen, clen);
    if (alen > clen)
      c.insert(0, alen - clen, '0');
    else if (alen < clen)
      a.insert(0, clen - alen, '0');
    if (a < c) {
      swap(a, c);
      swap(alen, clen);
    }
    string res(n, '0');
    int carry = 0;
    for (int i = n - 1; i >= 0; i--) {
      int digit = (a[i] - '0') - (c[i] - '0') - carry;
      if (digit < 0)
        digit += 10, carry = 1;
      else
        carry = 0;
      res[i] = digit + '0';
    }
    // remove leading zeros
    int i = 0;
    while (i < n && res[i] == '0') i++;
    if (i == n) return BigInteger("0");
    return BigInteger(res.substr(i));
  }

  // Overload * operator to multiply
  // two BigInteger objects
  BigInteger operator*(const BigInteger& b) {
    string a = str, c = b.str;
    int alen = a.length(), clen = c.length();
    int n = alen + clen;
    string res(n, '0');
    for (int i = alen - 1; i >= 0; i--) {
      int carry = 0;
      for (int j = clen - 1; j >= 0; j--) {
                int digit = (a[i] - '0') *
     (c[j-'0')+(res[i+j+1]-'0')+carry;
                carry = digit / 10;
                res[i + j + 1] = digit % 10 + '0';
      }
      res[i] += carry;
    }
    int i = 0;
    while (i < n && res[i] == '0') i++;
    if (i == n) return BigInteger("0");
    return BigInteger(res.substr(i));
  }

  // Overload << operator to output
  // BigInteger object
  friend ostream& operator<<(ostream& out, const
      BigInteger& b) {
    out << b.str;
    return out;
  }
};
```

9.3 Binary Search [33 lines] - 6d0146ba

```
// Pattern 1: Finding roots
// Find x where f(x) = target
while (high - low > precision) {
  double mid = low + (high - low) / 2.0;
  if (f(mid) <= target) {
    low = mid;
  } else {
    high = mid;
  }
}

// Pattern 2: Maximizing a function
// Find x that maximizes f(x)
while (high - low > precision) {
  double mid1 = low + (high - low) / 3.0;
  double mid2 = high - (high - low) / 3.0;
  if (f(mid1) < f(mid2)) {
    low = mid1;
  } else {
    high = mid2;
  }
}

// Pattern 3: Search using fixed iterations
// Find x that satisfies condition check(x)
double low = 0.0, high = 1e9;
for (int i = 0; i < 100; i++) {
  double mid = (low + high) / 2.0;
  if (check(mid))
    low = mid;
  else
    high = mid;
}
```

9.4 Expression Parsing [92 lines] - c4a451ed

```
bool delim(char c) { return c == ' '; }

bool is_op(char c) { return c == '+' || c == '-' || c
    == '*' || c == '/'; }

bool is_unary(char c) { return c == '+' || c == '-'; }

int priority(char op) {
  if (op < 0)  // unary operator
    return 3;
  if (op == '+' || op == '-') return 1;
  if (op == '*' || op == '/') return 2;
  return -1;
}

void process_op(stack<int>& st, char op) {
  if (op < 0) {
    int l = st.top();
    st.pop();
    switch (-op) {
      case '+':
        st.push(l);
        break;
      case '-':
        st.push(-l);
        break;
    }
  } else {
    int r = st.top();
    st.pop();
    int l = st.top();
    st.pop();
    switch (op) {
      case '+':
        st.push(l + r);
        break;
      case '-':
        st.push(l - r);
        break;
      case '*':
        st.push(l * r);
        break;
      case '/':
        st.push(l / r);
        break;
    }
  }
}

int evaluate(string& s) {
  stack<int> st;
  stack<char> op;
  bool may_be_unary = true;
  for (int i = 0; i < (int)s.size(); i++) {
    if (delim(s[i])) continue;

    if (s[i] == '(') {
      op.push('(');
      may_be_unary = true;
    } else if (s[i] == ')') {
      while (op.top() != '(') {
        process_op(st, op.top());
        op.pop();
      }
      op.pop();
      may_be_unary = false;
    } else if (is_op(s[i])) {
      char cur_op = s[i];
      if (may_be_unary && is_unary(cur_op)) cur_op =
          -cur_op;
      while (!op.empty() &&
             ((cur_op >= 0 && priority(op.top()) >=
                 priority(cur_op)) ||
              (cur_op < 0 && priority(op.top()) >
                  priority(cur_op)))) {
        process_op(st, op.top());
        op.pop();
      }
      op.push(cur_op);
      may_be_unary = true;
    } else {
      int number = 0;
      while (i < (int)s.size() && isalnum(s[i]))
        number = number * 10 + s[i++] - '0';
      --i;
      st.push(number);
      may_be_unary = false;
    }
  }

  while (!op.empty()) {
    process_op(st, op.top());
    op.pop();
  }
  return st.top();
}
```

9.5 Infix to Postfix [96 lines] - 92843ab9

```
bool delim(char c) { return c == ' '; }
bool is_op(char c) {
    return c == '+' || c == '-' || c == '*'
     || c == '/' || c == '^';
}
bool is_unary(char c) {
return c == '+' || c == '-';
}
int priority(char op) {
    if (op < 0) return 3;
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 4;
    return -1;
}

void process_op(string& output, char op) {
    if (op < 0) {
        switch (-op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
        }
    }
    else {
        switch (op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
            case '*':
                output += "* ";
                break;
            case '/':
                output += "/ ";
                break;
            case '^':
                output += "^ ";
                break;
        }
    }
}

string InfixToPostFix(string& s) {
    string output;
    stack<char> op;
    bool may_be_unary = true;
    for (int i = 0; i <(int)s.size(); i++){
        if (delim(s[i]))
            continue;
        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        }
        else if (s[i] == ')') {
            while (op.top() != '(') {
                process_op(output, op.top());
                op.pop();
            }
            op.pop();
            may_be_unary = false;
        }
        else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary && is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() &&
                    ((cur_op >= 0 && priority(op.top())
                        >= priority(cur_op)) ||
                    (cur_op < 0 && priority(op.top()) >
                        priority(cur_op)))) {
                process_op(output, op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        }
        else {
            char number;
            while (i < (int)s.size() && isalnum(s[i]))
                number = s[i++];
            --i;
            output.push_back(number);
            output.push_back(' ');
            may_be_unary = false;
        }
    }
    while (!op.empty()) {
        process_op(output, op.top());
        op.pop();
    }
    return output;
}
```

9.6 Kadanes [9 lines] - 6304cec2

```
// return maximum subarray sum.
int kadanes(int arr[], int n) {
    int maxsum = arr[0], curr_sum = arr[0];
    for (int i = 1; i < n; i++) {
        curr_sum = max(arr[i], curr_sum + arr[i]);
        maxsum = max(maxsum, curr_sum);
    }
    return maxsum;
}
```

9.7 Monotonic Stack (next greater) [7 lines] - b69c792c

```
for (int i = n - 1; i >= 0; i--) {
  while (!stk.empty() && v[i] >= v[stk.top()])
      stk.pop();
  ind[i] = stk.empty() ? -1 : stk.top();
  stk.push(i);
}
// 3 1 5 4 10
// 2 2 4 4 -1
```

9.8 N Queen [32 lines] - ac6d465b

```
// It just counts the number of ways to place the
    order.
const int N = 32;
int mark[N][N];
char grid[N][N];
int n, cnt;
void fillup(int row, int col) {
  for (int i = 1; i < n - row + 1; i++) {
    mark[row + i][col]++;
    if (col - i >= 0) mark[row + i][col - i]++;
    if (col + i < n) mark[row + i][col + i]++;
  }
}
void fillout(int row, int col) {
  for (int i = 1; i < n - row + 1; i++) {
    mark[row + i][col]--;
    if (col - i >= 0) mark[row + i][col - i]--;
    if (col + i < n) mark[row + i][col + i]--;
  }
}
void find_way(int row) {
  if (row == n) {
    cnt++;
    return;
  }
  for (int j = 0; j < n; j++) {
    if (grid[row][j] == '*' or mark[row][j]) continue;
    fillup(row, j);
    find_way(row + 1);
    fillout(row, j);
  }
}
// input in grid. call find_way(0);
```

9.9 Spirally Traverse [25 lines] - 04cd0672

```
vector<int> spirallyTraverse(vector<vector<int>>& mat) {
  int m = mat.size();
  int n = mat[0].size();
  vector<int> res;
  vector<vector<bool>> vis(m, vector<bool>(n, false));
  vector<int> dr = {0, 1, 0, -1};
  vector<int> dc = {1, 0, -1, 0};
  int r = 0, c = 0;
  int idx = 0;
  for (int i = 0; i < m * n; ++i) {
    res.push_back(mat[r][c]);
    vis[r][c] = true;
    int newR = r + dr[idx];
    int newC = c + dc[idx];
    if (0 <= newR && newR < m && 0 <= newC && newC < n
        && !vis[newR][newC]) {
      r = newR;
      c = newC;
    } else {
      idx = (idx + 1) % 4;
      r += dr[idx];
      c += dc[idx];
    }
  }
  return res;
}
```

9.10 Ternary Search [10 lines] - d44f71ce

```
double ternary_search(double l, double r) {
  double eps = 1e-9;  // error limit
  while (r - l > eps) {
    double m1 = l + (r - l) / 3, m2 = r - (r - l) / 3;
    double f1 = f(m1), f2 = f(m2);  // evaluates the
        function at m1, m2
    if (f1 < f2) l = m1;
    else r = m2;
  }
  return f(l);  // return the maximum of f(x) in [l, r]
}
```

10 String

10.1 Hashing [71 lines] - 0e25eadc

```
const int N = 2e6 + 9;
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 %
        MOD2;
  }
}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i]
          % MOD1) % MOD1;
      p.second = (hs[i].second + 1LL * pw[i].second *
          s[i] % MOD2) % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { // 1 -
      indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l - 1].first + MOD1)
        * 1LL * ipw[l - 1].first % MOD1;
    ans.second = (hs[r].second - hs[l - 1].second +
        MOD2) * 1LL * ipw[l - 1].second % MOD2;
    return ans;
  }
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
string s;
Hashing hs;
int lcp(int i1, int j1, int i2, int j2) {
  int l = 1, r = min(j1 - i1 + 1, j2 - i2 + 1), ans = 0;
  while (l <= r) {
    int mid = (l + r) / 2;
    if (hs.get_hash(i1, i1 + mid) == hs.get_hash(i2
        + 1, i2 + mid)) {
      ans = mid;
      l = mid + 1;
    }
    else {
      r = mid - 1;
    }
  }

  return ans;
}
int compare(int i1, int j1, int i2, int j2) {
  int l = lcp(i1, j1, i2, j2);
  int len1 = j1 - i1 + 1, len2 = j2 - i2 + 1;
  if (len1 == len2 && l == len1) return 0;
  if (l == len1) return -1;
  if (l == len2) return 1;
  return s[i1 + l] < s[i2 + l] ? -1 : 1;
}
```

10.2 KMP [22 lines] - c877552b

```
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
vector<int> find_matches(string text, string pat) {
  int n = pat.length(), m = text.length();
  string s = pat + "$" + text;
  vector<int> pi = prefix_function(s), ans;
  for (int i = n; i <= n + m; i++) {
    if (pi[i] == n) {
      ans.push_back(i - 2 * n);
    }
  }
  return ans;
}
```

10.3 Suffix Array [177 lines] - 3df3c95c

```
struct SuffixArray {
  // p suffix array 1 base, 0 index for dollar
  // rank will show 1 index value
  // 0 base suffix array -> suf[rank[i] - 1] = i
  vector<int> p, c, rank, lcp;
  vector<vector<int>> st;
  SuffixArray(string const& s) {
    build_suffix(s + char(1));
    build_rank(p.size());
    build_lcp(s + char(1));
    build_sparse_table(lcp.size());
  }
  void build_suffix(string const& s) {
    int n = s.size();
    const int MX_ASCII = 256;
    vector<int> cnt(max(MX_ASCII, n), 0);
    p.resize(n); c.resize(n);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i=1; i<MX_ASCII; i++) cnt[i]+=cnt[i-1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
      if (s[p[i]] != s[p[i-1]]) classes++;
      c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
      for (int i = 0; i < n; i++) {
        pn[i] = p[i] - (1 << h);
        if (pn[i] < 0) pn[i] += n;
      }
      fill(cnt.begin(), cnt.begin() + classes, 0);
      for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
      for (int i=1; i<classes; i++) cnt[i]+=cnt[i-1];
      for (int i=n-1;i>=0;i--) p[--cnt[c[pn[i]]]]=pn[i];
      cn[p[0]] = 0; classes = 1;
      for (int i = 1; i < n; i++) {
        pair<int, int> cur = {c[p[i]], c[(p[i] + (1 <<
            h)) % n]};
        pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1
            << h)) % n]};
        if (cur != prev) ++classes;
        cn[p[i]] = classes - 1;
      }
      c.swap(cn);
    }
  }
  void build_rank(int n) {
    rank.resize(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] = i;
  }
  void build_lcp(string const& s) {
    int n = s.size(), k = 0;
    lcp.resize(n - 1, 0);
    for (int i = 0; i < n; i++) {
      if (rank[i] == n - 1) {
        k = 0;
        continue;
      }
      int j = p[rank[i] + 1];
      while (i + k < n && j + k < n && s[i+k] == s[j+k])
        k++;
      lcp[rank[i]] = k;
      if (k) k--;
    }
  }
  void build_sparse_table(int n) {
    int lim = __lg(n);
    st.resize(lim + 1, vector<int>(n)); st[0] = lcp;
    for (int k = 1; k <= lim; k++)
      for (int i = 0; i + (1 << k) <= n; i++)
        st[k][i] = min(st[k - 1][i], st[k - 1][i + (1 <<
            (k - 1))]);
  }
  int get_lcp(int i) { return lcp[i]; }
  int get_lcp(int i, int j) {
    if (j < i) swap(i, j);
    j--; /*for lcp from i to j we don't need last lcp*/
    int K = __lg(j - i + 1);
    return min(st[K][i], st[K][j - (1 << K) + 1]);
  }
  // Compare two substrings (l1, r1) and (l2, r2)
  int compare(int l1, int r1, int l2, int r2) {
    int len1 = r1 - l1 + 1;
    int len2 = r2 - l2 + 1;
    int pos1 = rank[l1];
    int pos2 = rank[l2];
    if (pos1 == pos2) {
      if (len1 != len2) return len1 < len2 ? -1 : 1;
      return 0;
    }
    int common = get_lcp(min(pos1, pos2), max(pos1,
        pos2));
    int compare_len = min(min(len1, len2), common);
    if (compare_len == len1 && compare_len == len2)
        return 0;
    if (compare_len == len1) return -1;
    if (compare_len == len2) return 1;
    return pos1 < pos2 ? -1 : 1;
  }
};
// s.compare(suf[mid], min(n - suf[mid], m), t) ->
    -1(small), 0(equal), 1(large)

// https://cses.fi/problemset/task/2109/
// You are given a string of length n. If all of its
    substrings (not necessarily distinct) are ordered
    lexicographically, what is the kth smallest of
    them?

void solve() {
  string s;
  cin >> s;
  int k, n = s.size();
  cin >> k;
  SuffixArray suff(s);
  vector<int> sa(n), pref(n);
  for (int i = 0; i < n; i++) {
    sa[suff.rank[i] - 1] = i;
  }
  pref[0] = n - sa[0];
  for (int i = 1; i < n; i++) {
    pref[i] = pref[i - 1] + n - sa[i];
  }

  auto get_string = [&](int l, int r) {
    int sum = pref[r];
    if (l - 1 >= 0) sum -= pref[l - 1];
    return sum;
  };

  int L = 0, R = n - 1;
  int depth = 0;
  while (true) {
    if (depth > 0) {
      int count = R - L + 1;
      if (k <= count) {
        cout << s.substr(sa[L], depth) << '\n';
        return;
      }
      k -= count;
    }
    int cur = L;
    // skip all suffixes which are already ended
    while (cur <= R && sa[cur] + depth >= n) cur++;
    assert(cur <= R);

    while (cur <= R) {
      // extend to next level with a character c
      // cur = starting index of current character
        branch
      // cur_end = ending index of current character
        branch
      // binary search to find cur_end
      char c = s[sa[cur] + depth];
      int low = cur, high = R, cur_end = cur;
      while (low <= high) {
        int mid = (low + high) / 2;
        if (sa[mid] + depth < n && s[sa[mid] + depth]
            == c) {
          cur_end = mid;
          low = mid + 1;
        } else if (sa[mid] + depth >= n) {
          low = mid + 1;
        } else {
          high = mid - 1;
        }
      }

      int total = get_string(cur, cur_end);
      int branch = total - (cur_end - cur + 1) * depth;

      if (k <= branch) {
        // search in this branch
        L = cur;
        R = cur_end;
        depth++;
        goto next_level;
      } else {
        // search in next branch
        k -= branch;
        cur = cur_end + 1;
      }
    }

next_level:;
  }
}
```

10.4 Trie [56 lines] - 5399d31c

```
const int N = 26;
char BASE = 'A';
class Node {
 public:
  int EoW;
  Node* child[N];
  Node() {
    EoW = 0;
    for (int i = 0; i < N; i++) child[i] = NULL;
  }
};
Node* root = new Node();

void insert(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i++) {
    int r = s[i] - BASE;
    if (node->child[r] == NULL) node->child[r] = new
        Node();
    node = node->child[r];
  }
  node->EoW += 1;
}  // insert(root, s);

int search(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i++) {
    int r = s[i] - BASE;
    if (node->child[r] == NULL) return 0;
    node = node->child[r];
  }
  return node->EoW;
}  // search(root, s);

void print(Node* node, string s = "") {
  if (node->EoW) cout << s << "\n";
  for (int i = 0; i < N; i++) {
    if (node->child[i] != NULL) {
      char c = i + BASE;
      print(node->child[i], s + c);
    }
  }
}  // print whole trie

void remove(Node* node, string s) {
  for (size_t i = 0; i < s.size(); i++) {
    int r = s[i] - base;
    if (node->child[r] == NULL) return;
    node = node->child[r];
  }
  node->EoW--;
}  // remove(root, s)

void delete_trie(Node* node) {
  for (int i = 0; i < N; i++) {
    if (node->child[i] != NULL)
        delete_trie(node->child[i]);
  }
  delete node;
}  // delete full trie
```

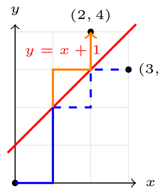
11 Random

11.1 Combinatorics

- $\sum_{k=0}^{n} \binom{n-k}{k} = Fib_{n+1}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- Number of binary sequences of length n such that no two 0's are adjacent $= Fib_{n+1}$
- Number of non-negative solution of $x_1 + x_2 + x_3 + ... + x_k = n$ is $\binom{n+k-1}{n}$

11.1.1 Catalan Number

- **Reflection Principle** $(n = 3)$
  Count paths $(0, 0) \to (n, n)$ *below* $y = x$.



- **Bad Path:** Touches $y = x + 1$.
- **Reflection:** Flip path after first touch.
- **Mapping:** Bad path to $(n, n) \iff$ Path to $(n-1, n+1)$.

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$
- $C_0 = 1, C_1 = 1, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786$
- Number of correct bracket sequences consisting of n opening brackets.
- Number of ways to completely parenthesize n+1 factors.
- The number of triangulations of a convex polygon with +2 sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the 2n points on a circle to form n disjoint i.e. non-intersecting chords.
- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal
- Number of permutation of length n that can be stack sorted.
- The number of non-crossing partitions of a set of n elements.
- The number of rooted full binary tree with n+1 leaves.
- The number of Dyck words of length 2n. A string consisting of n X's and n Y's such that no string prefix has more Y's than X's.
- Number of permutation of length n with no three-term increasing subsequence.
- Number of ways to tile a stairstep shape of height n with n rectangle.
- $C_n^k = \frac{k+1}{n+1}\binom{2n-k}{n-k}$ denote the number of bracket sequences of size 2n with the first k elements being (.

- $N(n, k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}$
- The number of expressions containing n pairs of correct parentheses, which contain k distinct nestings. $N(4, 2) = 6$
  $()((())), (())(()), (()(())), ((()())), ((())()), ((()))()$
- The number of paths from (0,0) to (2n, 0) with steps only northeast and southeast, not staying below the x-axis with k peaks. And sum of all number of peaks is Catalan number.

11.1.2 Stirling Number of the First Kind

- Count permutation according to their number of cycles.
- $S(n, k)$ count the number of permutation of n elements with k disjoint cycles.
- $S(n, k) = (n-1) \times S(n-1, k) + S(n-1, k-1), S(0, 0) = 1, S(n, 0) = S(0, n) = 0$
- $S(n, 1) = (n-1)!$
- $S(n, n-1) = \binom{n}{2}$
- $\sum_{k=0}^{n} S(n, k) = n!$

11.1.3 Stirling Numbers of the Second Kind

- Number of ways to partition a set of n objects into k non-empty subsets.
- $S(n, k) = k * S(n-1, k) + S(n-1, k-1), S(0, 0) = 1, S(n, 0) = S(0, n) = 0$
- $S(n, 2) = 2^{n-1} - 1$
- $S(n, k) = \frac{1}{k!}\sum\limits_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$
- $S(n, k) * k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

11.1.4 Bell Number

- Counts the number of partitions of a set.
- $B_{n+1} = \sum_{k=0}^{n}\binom{n}{k} * B_k$
- $B_n = \sum_{k=0}^{n} S(n, k)$, where S is Stirling number of second kind.
- The number of multiplicative partitions of a square free number with i prime factors is the i-th Bell number.
- $B(p^m + n) \equiv mB(n) + B(n+1) (\mod p)$
- If a deck is shuffled by removing and reinserting the top card n times, there are n^n possible shuffles. The number of shuffles that return the deck to its original order is B_n, so the probability of returning to the original order is B_n/n^n.

11.1.5 Lucas Theorem

- If p is prime then $\binom{p^a}{k} \equiv 0 \mod p$
- For non-negative integers m and n and a prime p: $\binom{m}{n} = \prod\limits_{i=0}^{k}\binom{m_i}{n_i}(\mod p)$ where $m = m_k p^k + m_{k-1}p^{k-1} + ... + m_1 p + m_0$ $n = n_k p^k + n_{k-1}p^{k-1} + ... + n_1 p + n_0$ are the base p expansion.
- Let n and k be non-negative integers, and let p be a prime. Let the base-p expansions of n and k be:

  $$n = n_m p^m + n_{m-1}p^{m-1} + \cdots + n_1 p^1 + n_0 p^0$$

  $$k = k_m p^m + k_{m-1}p^{m-1} + \cdots + k_1 p^1 + k_0 p^0$$

  where $0 \le n_i < p$ and $0 \le k_i < p$ are the base-p digits of n and k, respectively.

  The theorem states:

  $$\binom{n}{k} \equiv \prod_{i=0}^{m}\binom{n_i}{k_i} \pmod p$$

  Note: The individual terms $\binom{n_i}{k_i}$ are calculated using standard arithmetic, and the final product is taken modulo p.

- **Example:** $\binom{10}{2} \pmod 2$

  $$n = 10_{10} = 1010_2$$
  $$k = 2_{10} = 0010_2$$

  Applying Lucas's Theorem:

  $$\binom{10}{2} \equiv \binom{1}{0}\cdot\binom{0}{0}\cdot\binom{1}{1}\cdot\binom{0}{0} \pmod 2$$

  $$\equiv 1 \cdot 1 \cdot 1 \cdot 1 \equiv 1 \pmod 2$$

  Bitwise check: $1010_2 \& 0010_2 = 0010_2$. Since $2 = 2$, the result is 1.

  $$\binom{n}{k} \pmod 2 = \begin{cases} 1 & \text{if } (n \& k) = k \\ 0 & \text{otherwise} \end{cases}$$

11.1.6 Derangement

- A permutation such that no element appears in its original position.
- $d(n) = (n-1)*(d(n-1)+d(n-2)), d(0)=1, d(1)=0$
- $d(n) = nd(n-1) + (-1)^n = \lfloor \frac{n!}{e} \rfloor, n \geq 1$

11.1.7 Burnside Lemma

Given a group G of symmetries and a set X, the number of elements of X up to symmetry equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|$$

where X^g are the elements fixed by $g(g.x = x)$ If f(n) counts "configurations" of some sort of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k)$$

11.1.8 Eulerian Number

- $E(n,k)$ is the number of permutations of the numbers 1 to n in which exactly k elements are greater than the previous element.
- $E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k), E(n,0) = E(n,n-1) = 1$
- $E(n,k) = \sum\limits_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$
- $E(n,k) = E(n,n-1-k)$
- $E(0,k) = [k=0]$
- $E(n,1) = 2^n - n - 1$

11.2 Number Theory

11.2.1 Mobius Function and Inversion

- define $\mu(n)$ as the sum of the primitive nth roots of unity depending on the factorization of n into prime factors:

$$\mu(x) = \begin{cases} 0 & \text{n is not square free} \\ 1 & \text{n has even number of prime factors} \\ -1 & \text{n has odd number of prime factors} \end{cases}$$

- Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \leftrightarrow f(n) = \sum_{d|n}\mu(d)g(n/d)$$

- $\sum\limits_{d|n}\mu(d) = [n=1]$
- $\phi(n) = \sum\limits_{d|n}\mu(d).\frac{n}{d} = n\sum\limits_{d|n}\frac{\mu(d)}{d} = \sum\limits_{d|n} d.\mu(\frac{n}{d})$
- $a|b \rightarrow \phi(a)|\phi(b)$
- $\phi(mn) = \phi(m).\phi(n).\frac{d}{\phi(d)}$ where $d = gcd(m,n)$
- $\phi(n^m) = n^{m-1}\phi(n)$
- $\sum\limits_{i=1}^{n}[gcd(i,n)=k] = \phi(\frac{n}{k})$
- $\sum\limits_{i=1}^{n} gcd(i,n) = \sum\limits_{d|n} d.\phi(\frac{n}{d})$
- $\sum\limits_{i=1}^{n}\frac{1}{gcd(i,n)} = \sum\limits_{d|n}\frac{1}{d}.\phi(\frac{n}{d}) = \frac{1}{n}\sum\limits_{d|n} d.\phi(d)$
- $\sum\limits_{i=1}^{n}\frac{i}{gcd(i,n)} = \frac{n}{2}.\sum\limits_{d|n}\frac{1}{d}.\phi(\frac{n}{d}) = \frac{n}{2}.\frac{1}{n}\sum\limits_{d|n} d.\phi(d)$
- $\sum\limits_{i=1}^{n}\frac{n}{gcd(i,n)} = 2.\sum\limits_{i=1}^{n}\frac{i}{gcd(i,n)} - 1$

11.2.2 GCD and LCM

- gcd(a,b) = gcd(b, a mod b)
- If $a|b.c$, and gcd(a,b) = d, then $(a/d)|c$.
- GCD is a multiplicative function.
- gcd(a, lcm(b,c)) = lcm(gcd(a,b), gcd(a,c))
- $gcd(n^a-1, n^b-1) = n^{gcd(a,b)} - 1$

11.2.3 Gauss Circle Theorem

- Determine the number of lattice points in a circle centered at the origin with radius r.
- number of pairs (m,n) such that $m^2 + n^2 \leq r^2$
- $N(r) = 1 + 4\sum\limits_{i=0}^{\infty}(\lfloor\frac{r^2}{4i+1}\rfloor - \lfloor\frac{r^2}{4i+3}\rfloor)$

11.2.4 Pick's Theorem

According to Pick's Theorem We can calculate the area of any polygon by just counting the number of Interior and Boundary lattice points of that polygon. If number of interior points are I and number of boundary lattice points are B then Area (A) of polygon will be:

$$Area = I + B/2 - 1$$

where I is the number of points in the interior shape, B stands for the number of points on the boundary of the shape.

11.2.5 Formula Cheatsheet

- $\sum\limits_{i=1}^{n} = \frac{1}{m+1}[(n+1)^{m+1} - 1 - \sum\limits_{i=1}^{n}((i+1)^{m+1} - i^{m+1} - (m+1)i^m)]$
- $\sum\limits_{i=0}^{n} c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1$
- $\sum\limits_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum\limits_{i=1}^{\infty} c^i = \frac{c}{1-c}, |c| < 1$
- $H_n = \sum\limits_{i=1}^{n}\frac{1}{n}, \sum\limits_{i=1}^{n} iH_i = \frac{n(n+1)}{2}H_n - \frac{n(n-1)}{4}$
- $\sum\limits_{k=0}^{n}\binom{r+k}{k} = \binom{r+n+1}{n}$

11.2.6 Geometry and Formulas

- **Area Formulas**
  - Rectangle: $A = l \times w$
  - Square: $A = s^2$
  - Triangle: $A = \frac{1}{2}bh$
  - Circle: $A = \pi r^2$
  - Parallelogram: $A = bh$
  - Pyramid Base: $A = \frac{1}{2}b \times h_{slant}$
  - Polygon: $A = \frac{1}{2}|\sum_{i=1}^{n-1}(x_i y_{i+1})|$
  - Pick's formula: $A = I + \frac{B}{2} - 1$ where I = interior points, B = boundary points
- **Perimeter Formulas**
  - Rectangle: $P = 2(l+w)$
  - Square: $P = 4s$
  - Triangle: $P = a+b+c$
  - Circle: $P = 2\pi r$
- **Volume Formulas**
  - Cube: $V = s^3$
  - Rect. Prism: $V = lwh$
  - Cylinder: $V = \pi r^2 h$
  - Sphere: $V = \frac{4}{3}\pi r^3$
  - Pyramid: $V = \frac{1}{3}A_{base}h$
- **Surface Area Formulas**
  - Cube: $SA = 6s^2$
  - Rect. Prism: $SA = 2(lw + lh + wh)$

- Cylinder: $SA = 2\pi r(r+h)$
- Sphere: $SA = 4\pi r^2$
- Pyramid: $SA = A_{base} + \frac{1}{2}P_{base}h_{slant}$

- **Triangles**
  - Side Lengths: a, b, c
  - Semi Perimeter: $p = \frac{a+b+c}{2}$
  - Area (Heron's): $A = \sqrt{p(p-a)(p-b)(p-c)}$
  - Circumradius: $R = \frac{abc}{4A}$
  - Inradius: $r = \frac{A}{p}$
- **Summation Of Series**
  - $c^k + c^{k+1} + \cdots + c^n = \frac{c^{n+1}-c^k}{c-1}$
  - $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$
  - $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$
  - $1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$
  - $1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
  - **Arithmetic**: $S_n = \frac{n}{2}\left[2a + (n-1)d\right]$
  - **Geometric**: $S_n = a \cdot \frac{r^n-1}{r-1}$
- **Triangle Laws**
  - **Law of Sines**: $\dfrac{a}{\sin A} = \dfrac{b}{\sin B} = \dfrac{c}{\sin C} = 2R$
  - **Law of Cosines**: $a^2 = b^2 + c^2 - 2bc\cos A$
  - **Law of Tangents**: $\dfrac{a+b}{a-b} = \dfrac{\tan\left(\frac{A+B}{2}\right)}{\tan\left(\frac{A-B}{2}\right)}$
- **Angle Sum and Difference**

$$\sin(A+B) = \sin A\cos B + \cos A\sin B$$
$$\cos(A+B) = \cos A\cos B - \sin A\sin B$$
$$\sin(A-B) = \sin A\cos B - \cos A\sin B$$
$$\cos(A-B) = \cos A\cos B + \sin A\sin B$$
$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A\tan B}$$
$$\tan(A-B) = \frac{\tan A - \tan B}{1 + \tan A\tan B}$$

- **Double Angle Formulas**

$$\sin 2\theta = 2\sin\theta\cos\theta$$
$$\cos 2\theta = \cos^2\theta - \sin^2\theta$$
$$\tan 2\theta = \frac{2\tan\theta}{1-\tan^2\theta}$$

- **Half Angle Formulas**

$$\sin\left(\frac{\theta}{2}\right) = \pm\sqrt{\frac{1-\cos\theta}{2}}$$
$$\cos\left(\frac{\theta}{2}\right) = \pm\sqrt{\frac{1+\cos\theta}{2}}$$
$$\tan\left(\frac{\theta}{2}\right) = \frac{1-\cos\theta}{\sin\theta}$$

- **Sum-to-Product Formulas**

$$\sin r + \sin w = 2\sin\left(\frac{r+w}{2}\right)\cos\left(\frac{r-w}{2}\right)$$
$$\cos r + \cos w = 2\cos\left(\frac{r+w}{2}\right)\cos\left(\frac{r-w}{2}\right)$$
$$(V+W)\tan\left(\frac{r-w}{2}\right) = (V-W)\tan\left(\frac{r+w}{2}\right)$$

- **Linear Combinations**

$$a\cos x + b\sin x = r\cos(x-\phi)$$
$$a\sin x - b\cos x = r\sin(x-\phi)$$
$$\text{where } r = \sqrt{a^2+b^2}, \quad \phi = \arctan(b/a)$$

11.2.7 Most Number of Divisors

Max Value	Number	Divisors
10^3	83,160	128
10^6	720,720	240
10^7	9,609,600	640
10^8	98,280,000	672
10^9	735,134,400	1,344
10^{10}	7,242,460,800	2,688
10^{11}	73,346,256,000	5,376
10^{12}	936,966,912,400	10,752

11.2.8 Pascal's Triangle

```
                         1
                       1   1
                     1   2   1
                   1   3   3   1
                 1   4   6   4   1
               1   5  10  10   5   1
             1   6  15  20  15   6   1
           1   7  21  35  35  21   7   1
         1   8  28  56  70  56  28   8   1
       1   9  36  84 126 126  84  36   9   1
     1  10  45 120 210 252 210 120  45  10   1
```

11.2.9 Logarithm Rules

- $\log_b(xy) = \log_b x + \log_b y$
- $\log_b\left(\dfrac{x}{y}\right) = \log_b x - \log_b y$
- $\log_b(x^k) = k\log_b x$
- $\log_b b = 1$
- $\log_b 1 = 0$
- $\log_b x = \dfrac{\log_k x}{\log_k b}$

11.2.10 Miscellaneous

- $2^{100} = 2^{50} \times 2^{50}$
- Fibonacci Matrix:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

- $\log(n!) = \log 1 + \log 2 + \cdots + \log n$
- $\gcd(a,b) = \gcd(a-b, b)$
- Prime p in $n!$:

$$\left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots$$

- Catalan:

$$\binom{2n}{n} - \binom{2n}{n-1}$$

- Divisors of $p^x q^y$: $(x+1)(y+1)$
- Sum of divisors of $p^x q^y$:

$$(1 + p + p^2 + \cdots + p^x)(1 + q + q^2 + \cdots + q^y)$$

- Divisibility rules:
  - **3**: Sum of digits divisible by 3
  - **4**: Last two digits divisible by 4
  - **5**: Last digit 0 or 5
  - **6**: Divisible by both 2 and 3
  - **7**: Double last digit, subtract from rest. Continue till manageable.
  - **8**: Last three digits divisible by 8
  - **9**: Sum of digits divisible by 9

11: $(sum\ of\ odd\ digits) - (sum\ of\ even\ digits) \equiv 0 \pmod{11}$

- Derangement: $D_n = (n-1)D_{n-2} + (n-1)D_{n-1}$
- Golden Ratio: $\Phi = \frac{1+\sqrt{5}}{2} \approx 1.618034$
- n-th Fibonacci:

$$F_n = \frac{\Phi^n - (1-\Phi)^n}{\sqrt{5}}$$

- Stars & Bars: Solutions of $x_1+x_2+\cdots+x_k = n$ $x_i > 0$: $\binom{n-1}{k-1}$, $x_i \geq 0$: $\binom{n+k-1}{k-1}$
- **Throwing Dice (Matrix Exponentiation)**
  - **Recurrence Relation** The number of ways to achieve a sum n is given by the linear recurrence:

$$\begin{aligned} f(n) =& f(n-1) + f(n-2) + f(n-3) \\ &+ f(n-4) + f(n-5) + f(n-6) \end{aligned}$$

with the base case $f(0) = 1$ and $f(k) = 0$ for $k < 0$.

  - **State Transition** The relationship between the new state vector (V_i) and the old state vector (V_{i-1}) is:

$$\underbrace{\begin{bmatrix} f(i) \\ f(i-1) \\ f(i-2) \\ f(i-3) \\ f(i-4) \\ f(i-5) \end{bmatrix}}_{V_i} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{T} \times \underbrace{\begin{bmatrix} f(i-1) \\ f(i-2) \\ f(i-3) \\ f(i-4) \\ f(i-5) \\ f(i-6) \end{bmatrix}}_{V_{i-1}}$$

  - **Reason for Transition Matrix (T)** The constants (1/0) implement the recurrence and the shifting of the state vector:
    - **Row 1** $(f(n))$: **1 1 1 1 1 1** (Implements the sum: $f(n-1) + f(n-2) + f(n-3) + f(n-4) + f(n-5) + f(n-6)$).
    - **Row 2** $(f(n-1))$: **1 0 0 0 0 0** (Copies $f(n-1)$).
    - **Row 3** $(f(n-2))$: **0 1 0 0 0 0** (Copies $f(n-2)$).
    - **Row 4** $(f(n-3))$: **0 0 1 0 0 0** (Copies $f(n-3)$).
    - **Row 5** $(f(n-4))$: **0 0 0 1 0 0** (Copies $f(n-4)$).
    - **Row 6** $(f(n-5))$: **0 0 0 0 1 0** (Copies $f(n-5)$).
  - **Final Answer** The n-th term is found by: $f(n) = (T^n)_{1,1} \times f(0)$, where $f(0) = 1$.

$$f(n) = (T^n)_{1,1}$$

11.3 Probability and Expectation

- Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.
- Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

- **Discrete distributions**
  - **Binomial distribution:** The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n,p)$, $n = 1, 2, \ldots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np,\ \sigma^2 = np(1-p)$$

$\text{Bin}(n,p)$ is approximately $\text{Po}(np)$ for small p.

  - **First success distribution:** The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1-p)^{k-1},\ k = 1, 2, \ldots$$

$$\mu = \frac{1}{p},\ \sigma^2 = \frac{1-p}{p^2}$$

  - **Poisson distribution:** The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda}\frac{\lambda^k}{k!},\ k = 0, 1, 2, \ldots$$

$$\mu = \lambda,\ \sigma^2 = \lambda$$

- **Continuous distributions**
  - **Uniform distribution:** If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a,b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2},\ \sigma^2 = \frac{(b-a)^2}{12}$$

  - **Exponential distribution:** The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda},\ \sigma^2 = \frac{1}{\lambda^2}$$

  - **Normal distribution:** Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$