University of Central Florida

UCF Narcissus

Kyle Cochran, Tyler Mark, David Santamaria

2022-02-03

Contest (1)

template.cpp

10 lines

```
#include <bits/stdc++.h>
#define all(x) begin(x), end(x)
using namespace std;

using ll = long long;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

Data structures (2)

BIT.h

Description: Query [l, r] sums, and point updates. kth() returns the smallest index i s.t. query(0, i) >= k
Time: $\mathcal{O}(\log n)$ for all ops.

33f78c, 22 lines

```
template <typename T>
struct BIT {
  vector<T> s;
  int n;
  BIT(int n): s(n + 1), n(n) {}
  void update(int i, T v) {
    for (i++; i <= n; i += i & -i) s[i] += v;
  }
  T query(int i) {
    T ans = 0;
    for (i++; i; i -= i & -i) ans += s[i];
    return ans;
  }
  T query(int l, int r) { return query(r) - query(l - 1); }
  int kth(T k) { // returns n if k > sum of tree
    if (k <= 0) return -1;
    int i = 0;
    for (int pw = 1 << __lg(n); pw; pw >>= 1)
      if (i + pw <= n && s[i + pw] < k) k -= s[i += pw];
    return i;
  }
};
```

dsu.h

Description: Maintains a collection of disjoint sets.
Time: $\mathcal{O}(\alpha(1))$ amortized.

4d9c0b, 30 lines

```
class ufds {
    public:
    vi p, rank, size;
    int num_distincts;
    ufds(int n) {
        p.resize(n); rank.resize(n); size.resize(n);
        for (int i = 0; i < n; i++) {
            rank[i] = 0;
            size[i] = 1;
            p[i] = i;
        }
        distincts = n;
    }
    int find(int i) { return (p[i] == i) ? i : (p[i] = find(p[i
        ])); }
    bool same(int i, int j) { return find(i) == find(j); }
    void union_set(int i, int j) {
        int pi = find(i), pj = find_set(j);
        if (pi == pj) return;
        distincts--;
        size[pi] = size[pj] = size[pi] + size[pj];
        if (rank[pi] > rank[pj]) {
            p[pj] = pi;
        } else {
            p[pi] = p[pj];
            if (rank[pi] == rank[pj]) {
                rank[pj]++;
            }
        }
    }
};
```

fenwicktree.h

Description: Binary Indexed Tree to support logarithmic complexity on point update and range query
Memory: $\mathcal{O}(N)$
Time: $\mathcal{O}(log(M))$ update/query, $\mathcal{O}(nlog(n))$ build

6c20de, 24 lines

```
#define lso(x) ((x) & -(x))
class bit {
    public:
        int n;
        vector<ll> ft;
        bit(int n) {
            this->n = n;
            ft.resize(n + 1);
            fill(ft.begin(), ft.end(), 0);
        }
        void update(int i, int val) {
            for (; i < (int) ft.size(); i += lso(i))
                ft[i] += val;
        }
        ll rsq(int i) {
            ll ret = 0;
            for (; i; i -= lso(i))
                ret += ft[i];
            return ret;
        }
        ll rsq(int i, int j) {
            return rsq(j) - rsq(i - 1);
        }
};
```

Geometry (3)

closestpairpoints.cpp

993a04, 63 lines

```
struct vec{
    int x, y, id;
    explicit vec(int x=0, int y=0, int id=0) : x(x), y(y), id(
        id){}
};

int n;
vector<vec> a, t;
double mindist;
pair<int, int> best;

void updClosest(const vec& a, const vec& b){
    double dx = a.x - b.x, dy = a.y - b.y;
    double dist = sqrt(dx*dx + dy*dy);
    if(dist < mindist){
        mindist = dist;
        best = {a.id, b.id};
    }
}

bool cmpX(const vec& a, const vec& b){
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

bool cmpY(const vec& a, const vec& b){
    return a.y < b.y;
}

void solve(int l, int r){
    if(r-l <= 3){
        for(int i = l; i < r; i++){
            for(int j = i+1; j < r; j++){
                updClosest(a[i], a[j]);
            }
        }
        sort(a.begin()+l, a.begin()+r, cmpY);
        return;
    }

    int m = (l+r)/2;
    int midx = a[m].x;
    solve(l, m);
    solve(m, r);

    merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin
        () + r, t.begin(), cmpY);
    copy(t.begin(), t.begin() + (r-l), a.begin() + l);

    int tSz = 0;
    for(int i = l; i < r; i++){
        if(abs(a[i].x-midx) < mindist){
            for(int j = tSz - 1; j >= 0 && a[i].y - t[j].y <
                mindist; j--){
                updClosest(a[i], t[j]);
            }
            t[tSz++] = a[i];
        }
    }
}

void clstPts(){
    t.resize(n);
    sort(a.begin(), a.end(), cmpX);
    mindist = DBL_MAX;
    solve(0, n);
}
```

convexHull.cpp

Description: Given a set of vertices, find a set that creates a polygon such that all vertices lie within that polygon
Memory: $\mathcal{O}(n)$
Time: $\mathcal{O}(n)$

1e3255, 40 lines

```
using point = pair<int, int>;
#define xx first
#define yy second

int cross(point o, point a, point b){
    int dx1 = a.xx-o.xx, dy1 = a.yy-o.yy;
    int dx2 = o.xx-b.xx, dy2 = o.yy-b.yy;
    return dx1*dy2-dx2*dy1;
}

vector<point> convexHull(vector<point> p, int n){
    vector<point> hull(n);
    if(n <= 3){
        for(int i = 0; i < n; i++)
            hull[i] = p[i];

        return;
    }

    sort(p.begin(), p.end());

    int k = 0;
    for(int i = 0; i < n; i++){
        while(k >= 2 && cross(hull[k-2], hull[k-1], p[i]) <= 0)
            k--;

        hull[k++] = p[i];
    }

    for(int i = n-1, t = k+1; i > 0; i--){
        while(k >= t && cross(hull[k-2], hull[k-1], p[i-1]) <=
             0)
            k--;

        hull[k++] = p[i];
    }

    hull.resize(k);

    return hull;
}
```

seg.cpp

Description: Line segment geometry
Memory: $\mathcal{O}(1)$
Time: $\mathcal{O}(1)$

3199ec, 66 lines

```
#define eps 1e9

using vec = pair<double, double>;
#define xx first
#define yy second

vec operator+(const vec & v, const vec & u) { return {v.xx+u.xx
     , v.yy+u.yy}; }
vec operator-(const vec & v, const vec & u) { return {v.xx-u.xx
     , v.yy-u.yy}; }
vec operator*(const vec & v, const double & c) { return {v.xx *
      c, v.yy * c}; }

double dotProd(vec v, vec u) { return v.xx*u.xx + v.yy*u.yy; }
double crossProd(vec v, vec u) { return v.xx*u.yy - v.yy*u.xx;
     }
double mag2(vec v) { return dotProd(v, v); }
double mag(vec v) { return sqrt(mag2(v)); }
vec unit(vec v) { return v * (1.0/mag(v)); }

vec rotate(vec v, double th){
    double newX = v.xx*cos(th) + v.yy*sin(th);
    double newY = v.xx*sin(th) + v.yy*cos(th);
    return {newX, newY};
}

double angle(vec v) { return atan2(v.yy, v.xx); }

//start

using seg = pair<vec, vec>;

vec lineIntersection(seg a, seg b){
    vec dirA = a.second - a.first, dirB = b.second - b.first;
    double det = crossProd(dirB, dirA);
    if(det == 0) return {INT_MAX, INT_MAX};
    double t = (crossProd(dirB, b.first-a.first)) / det;
    return a.first + dirA * t;
}

bool containsPoint(seg s, vec p){
    vec dir = s.second-s.first;
    double dist = crossProd(dir, p-s.first)/mag(dir);
    if(abs(dist) < eps) return false;
    return (mag(dir)-mag(s.first-p)-mag(s.second-p) < eps);
}

vec segIntersection(seg a, seg b){
    vec intersect = lineIntersection(a, b);
    if(intersect.first == INT_MAX && intersect.first == INT_MAX
        )
        return {INT_MAX, INT_MAX};
    if(containsPoint(a, intersect) && containsPoint(b,
        intersect))
        return intersect;
    return {INT_MAX, INT_MAX};
}

//returns 1 if above, 0 if on, -1 if below
int side(seg s, vec p){
    vec dir = s.second-s.first;
    double dist = crossProd(dir, p-s.first)/mag(dir);
    if(abs(dist) < eps) return 0;
    if(dist < 0) return -1;
    else return 1;
}

bool intersects(seg a, seg b){
    return side(a, b.first)!=side(a, b.second) &&
           side(b, a.first)!=side(b, a.second);
}
```

vec.cpp

Description: Vector code
Memory: $\mathcal{O}(1)$
Time: $\mathcal{O}(1)$

00645f, 24 lines

```
using vec = pair<double, double>;
#define xx first
#define yy second

vec operator+(const vec & v, const vec & u) { return {v.xx+u.xx
     , v.yy+u.yy}; }
vec operator-(const vec & v, const vec & u) { return {v.xx-u.xx
     , v.yy-u.yy}; }
vec operator*(const vec & v, const double & c) { return {v.xx *
      c, v.yy * c}; }

double dotProd(vec v, vec u) { return v.xx*u.xx + v.yy*u.yy; }
double crossProd(vec v, vec u) { return v.xx*u.yy - v.yy*u.xx;
     }

double mag2(vec v) { return dotProd(v, v); }
double mag(vec v) { return sqrt(mag2(v)); }
vec unit(vec v) { return v * (1.0/mag(v)); }

vec rotate90(vec v){ return{-v.yy, v.xx}; }
vec rotate270(vec v){ return{v.yy, -v.xx}; }
vec rotate(vec v, double th){
    double newX = v.xx*cos(th) + v.yy*sin(th);
    double newY = v.xx*sin(th) + v.yy*cos(th);
    return {newX, newY};
}

double angle(vec v) { return atan2(v.yy, v.xx); }
```

Graphs (4)

bellmanFord.cpp

Description: Single Source Shortest Path (SSSP)
Memory: $\mathcal{O}(V^2)$
Time: $\mathcal{O}(V^3)$

e3cbb3, 24 lines

```
#define vv first
#define ww second
using edge = pair<int, int>;

void bellmanFord(vector<edge> g[], int v, int s){
    int dist[v];
    memset(dist, 0, sizeof(0));

    for(int i = 0; i < v-1; i++)
        for(int u = 0; u < v; u++)
            for(edge e : g[u])
                if(dist[u] + e.ww < dist[e.vv])
                    dist[e.vv] = dist[u] + e.ww;

    //check for negative cycles
    for(int u = 0; u < v; u++){
        for(edge e : g[u]){
            if(dist[u]!=INT_MAX && dist[u] + e.ww < dist[e.vv])
             {
                //negative cycle reached
                return;
            }
        }
    }
}
```

dijkstraTylerM.cpp

<bits/stdc++.h> 1faf48, 28 lines

```
using namespace std;

#define vv first
#define ww second
using edge = pair<int, int>;

void djikstras(vector<edge> g[], int v, int s){
    priority_queue<edge, vector<edge>, greater<edge>> pq;
    vector<int> dist(v, INT_MAX);

    dist[s] = 0;
    pq.push(make_pair(0,s));

    while(!pq.empty()){
        if(pq.top().first > dist[pq.top().first])
            continue;

        int u = pq.top().second;
        pq.pop();

        for(edge e : g[u]){
            if(dist[e.vv] > dist[u] + e.ww){
                dist[e.vv] = dist[u] + e.ww;
                pq.push(make_pair(dist[e.vv], e.vv));
            }
        }
    }
}
```

floydWarshall.cpp

Description: FindAll-Pairs Shortest Paths (APSP)
Memory: $\mathcal{O}(n^2)$
Time: $\mathcal{O}(n^3)$

ca1aff, 25 lines

```
#define vv first
#define ww second
using edge = pair<int, int>;

void floydWarshall(vector<edge> g[], int n){
    int d[n][n];
    memset(d, INT_MAX, sizeof(d));
    for(int i = 0; i < n; i++) d[i][i] = 0;

    for(int i = 0; i < n; i++){
        for(edge e : g[i]){
            if(e.ww < d[i][e.vv])
                d[i][e.vv] = d[e.vv][i] = e.ww;
        }
    }

    for(int k = 0; k < n; k++){
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }

}
```

kruskalMST.cpp

Description: Construct a Minimum Spanning Tree using Kruskal's algorithm
Time: $\mathcal{O}(E\alpha)$

30806f, 50 lines

```
#define vv first
#define ww second
using edge = tuple<int, int, int>;

struct disjoint_set{
    int n;
    int *par, *height;

    disjoint_set(int nn){
        n = nn;
        par = new int[n];
        memset(par, -1, sizeof(par));
        height = new int[n];
        memset(height, 1, sizeof(height));
    }

    int parent(int i){
        return par[i] == -1 ? i : (par[i] = parent(par[i]));
    }

    void unionize(int a, int b){
        a = parent(a);
        b = parent(b);

        if(a==b) return;
        if(height[a] == height[b])
            height[a]++;

        if(height[a] >= height[b])
            par[b] = a;
        else par[a] = b;
    }
};

vector<edge> kruskalMST(vector<edge> edges, int n){
    sort(edges.begin(), edges.end(), [&](edge & a, edge & b) ->
         bool { return get<2>(a) < get<2>(b); });
    disjoint_set ds(n);

    int tot = 0;
    vector<edge> out;
    for(edge e : edges){
        if(ds.parent(get<0>(e)) != ds.parent(get<1>(e))){
            tot += get<2>(e);
            out.push_back(e);
            ds.unionize(get<0>(e), get<1>(e));
        }
    }

    return out;
}
```

Dinic.cpp

Description: Compute maximum flow in a graph. The basic principle is that a Maximum flow = minimum cut and Breadth First Search is used as a sub-routine.
Memory: $\mathcal{O}(E+V)$
Time: $\mathcal{O}(EV^2)$

ef99cb, 47 lines

```
using ll = long long;

struct Dinic {
  struct Edge {
    int to, rev;
    ll c, oc;
    ll flow() { return max(oc - c, 0LL); } // if you need flows
      Edge(int tt, int rr, ll cc, ll oo){
        to = tt; rev = rr; c = cc; oc = oo;
      }
  };
  vector<int> lvl, ptr, q;
  vector<vector<Edge>> adj;
  Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
  void add(int a, int b, ll c, ll rcap = 0) {
    adj[a].push_back(Edge((ll)b, adj[b].size(), c, c));
    adj[b].push_back({a, (int)adj[a].size() - 1, rcap, rcap});
  }
  ll dfs(int v, int t, ll f) {
    if (v == t || !f) return f;
    for (int& i = ptr[v]; i < adj[v].size(); i++) {
      Edge& e = adj[v][i];
      if (lvl[e.to] == lvl[v] + 1)
        if (ll p = dfs(e.to, t, min(f, e.c))) {
          e.c -= p, adj[e.to][e.rev].c += p;
          return p;
        }
    }
    return 0;
  }
  ll calc(int s, int t) {
    ll flow = 0; q[0] = s;
    for(int L = 0; L < 31; L++) do { // 'int L=30' maybe faster
        for random data
      lvl = ptr = vector<int>(q.size());
      int qi = 0, qe = lvl[s] = 1;
      while (qi < qe && !lvl[t]) {
        int v = q[qi++];
        for (Edge e : adj[v])
          if (!lvl[e.to] && e.c >> (30 - L))
            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
      }
      while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
    } while (lvl[t]);
    return flow;
  }
  bool leftOfMinCut(int a) { return lvl[a] != 0; }
```

```
};
```

topSort.cpp

Description: Find a topsort of a directed graph
Memory: $\mathcal{O}(V)$
Time: $\mathcal{O}(V+E)$

36bfac, 27 lines

```
using namespace std;

#define vv first
#define ww second
using edge = pair<int, int>;

void topSortUtil(vector<edge> g[], int v, stack<int> s, bool
     seen[]){
    seen[v] = true;
    for(edge e : g[v])
        if(!seen[e.vv])
            topSortUtil(g, e.vv, s, seen);
    s.push(v);
}

vector<int> topSort(vector<edge> g[], int v){
    stack<int> out;
    bool seen[v];
    for(int i = 0; i < v; i++)
        if(!seen[i])
            topSortUtil(g, i, out, seen);
    vector<int> ts(v);
    for(int i = v-1; i >= 0; i--){
        ts[i] = out.top();
        out.pop();
    }
    return ts;
}
```

SCCTarjan.h

Description: Finds strongly connected components of a directed graph. Visits/indexes SCCs in reverse topological order.
Usage: `scc(graph) returns an array that has the ID of each node's SCC. scc(graph, [&](vector<int>& v) { ... }) calls the lambda on each SCC, and returns the same array.`
Time: $\mathcal{O}(|V|+|E|)$

358d18, 37 lines

```
namespace SCCTarjan {
  vector<int> val, comp, z, cont;
  int Time, ncomps;
  template <class G, class F>
  int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x;
    z.push_back(j);
    for (auto e : g[j])
      if (comp[e] < 0) low = min(low, val[e] ?: dfs(e, g, f));
    if (low == val[j]) {
      do {
        x = z.back();
        z.pop_back();
        comp[x] = ncomps;
        cont.push_back(x);
      } while (x != j);
      f(cont);
      cont.clear();
      ncomps++;
    }
    return val[j] = low;
  }
  template <class G, class F>
  vector<int> scc(G& g, F f) {
    int n = g.size();
    val.assign(n, 0);
    comp.assign(n, -1);
    Time = ncomps = 0;
    for (int i = 0; i < n; i++)
      if (comp[i] < 0) dfs(i, g, f);
    return comp;
  }
  template <class G> // convenience function w/o lambda
  vector<int> scc(G& g) {
    return scc(g, [](auto& v) {});
  }
} // namespace SCCTarjan
```

SCCKosaraju.h

Description: Finds strongly connected components of a directed graph. Visits/indexes SCCs in topological order.
Usage: `scc(graph) returns an array that has the ID of each node's SCC.`
Time: $\mathcal{O}(|V|+|E|)$

9b78e7, 29 lines

```
namespace SCCKosaraju {
  vector<vector<int>> adj, radj;
  vector<int> todo, comp;
  vector<bool> vis;
  void dfs1(int x) {
    vis[x] = 1;
    for (int y : adj[x])
      if (!vis[y]) dfs1(y);
    todo.push_back(x);
  }
  void dfs2(int x, int i) {
    comp[x] = i;
    for (int y : radj[x])
      if (comp[y] == -1) dfs2(y, i);
  }
  vector<int> scc(vector<vector<int>>& _adj) {
    adj = _adj;
    int time = 0, n = adj.size();
    comp.resize(n, -1), radj.resize(n), vis.resize(n);
    for (int x = 0; x < n; x++)
      for (int y : adj[x]) radj[y].push_back(x);
    for (int x = 0; x < n; x++)
      if (!vis[x]) dfs1(x);
    reverse(todo.begin(), todo.end());
    for (int x : todo)
      if (comp[x] == -1) dfs2(x, time++);
    return comp;
  }
}; // namespace SCCKosaraju
```

dijkstra.h

Description: Computes shortest paths from s to any node reachable from s. Pass in an adjacency list of pairs (node, weight) and a starting node s.
Time: $\mathcal{O}((|V|+|E|)\log|V|)$

e0bb66, 16 lines

```
constexpr int INF = (int) 1e9;
vector<int> dijkstra(
        vector<vector<ii>> adjlist, int s) {
    using ii = pair<int, int>;
    vector<int> dist(V, INF); dist[s] = 0;
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.push(ii(0, s));
    while (!pq.empty()) {
        auto [d, u] = pq.top(); pq.pop();
        if (d > dist[u]) continue;
        for (auto [v, w] : adjlist[u])
            if (d + w < dist[v])
                pq.push(ii(dist[v] = d + w, v));
    }
    return dist;
}
```

Mathematics (5)

Fraction.h

Description: Struct for representing fractions/rationals. All ops are $O(\log N)$ due to GCD in constructor. Uses cross multiplication.

a1de34, 27 lines

```
template <typename T>
struct Q {
  T a, b;
  Q(T p, T q = 1) {
    T g = gcd(p, q);
    a = p / g;
    b = q / g;
    if (b < 0) a = -a, b = -b;
  }
  T gcd(T x, T y) const { return __gcd(x, y); }
  Q operator+(const Q& o) const {
    return {a * o.b + o.a * b, b * o.b};
  }
  Q operator-(const Q& o) const {
    return *this + Q(-o.a, o.b);
  }
  Q operator*(const Q& o) const { return {a * o.a, b * o.b}; }
  Q operator/(const Q& o) const { return *this * Q(o.b, o.a); }
  Q recip() const { return {b, a}; }
  int signum() const { return (a > 0) - (a < 0); }
  bool operator<(const Q& o) const {
    return a * o.b < o.a * b;
  }
  friend ostream& operator<<(ostream& cout, const Q& o) {
    return cout << o.a << "/" << o.b;
  }
};
```

FractionOverflow.h

Description: Safer struct for representing fractions/rationals. Comparison is 100% overflow safe; other ops are safer but can still overflow. All ops are $O(\log N)$.

feba79, 43 lines

```
template <typename T>
struct QO {
  T a, b;
  QO(T p, T q = 1) {
    T g = gcd(p, q);
    a = p / g;
    b = q / g;
    if (b < 0) a = -a, b = -b;
  }
  T gcd(T x, T y) const { return __gcd(x, y); }
  QO operator+(const QO& o) const {
    T g = gcd(b, o.b), bb = b / g, obb = o.b / g;
    return {a * obb + o.a * bb, b * obb};
  }
  QO operator-(const QO& o) const {
    return *this + QO(-o.a, o.b);
  }
  QO operator*(const QO& o) const {
    T g1 = gcd(a, o.b), g2 = gcd(o.a, b);
    return {(a / g1) * (o.a / g2), (b / g2) * (o.b / g1)};
  }
  QO operator/(const QO& o) const {
    return *this * QO(o.b, o.a);
  }
  QO recip() const { return {b, a}; }
  int signum() const { return (a > 0) - (a < 0); }
```

```
  static bool lessThan(T a, T b, T x, T y) {
    if (a / b != x / y) return a / b < x / y;
    if (x % y == 0) return false;
    if (a % b == 0) return true;
    return lessThan(y, x % y, b, a % b);
  }
  bool operator<(const QO& o) const {
    if (this->signum() != o.signum() || a == 0) return a < o.a;
    if (a < 0)
      return lessThan(abs(o.a), o.b, abs(a), b);
    else
      return lessThan(a, b, o.a, o.b);
  }
  friend ostream& operator<<(ostream& cout, const QO& o) {
    return cout << o.a << "/" << o.b;
  }
};
```

PrimeSieve.h

Description: Prime sieve for generating all primes up to a certain limit. isprime[i] is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored.

dc4f55, 14 lines

```
const int MAX_PR = 5'000'000;
bitset<MAX_PR> isprime;
vector<int> primeSieve(int lim) {
  isprime.set();
  isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i * i < lim; i += 2)
    if (isprime[i])
      for (int j = i * i; j < lim; j += i * 2) isprime[j] = 0;
  vector<int> pr;
  for (int i = 2; i < lim; i++)
    if (isprime[i]) pr.push_back(i);
  return pr;
}
```

PrimeSieveFast.h

Description: Prime sieve for generating all primes smaller than LIM.
Time: LIM=1e9 $\approx$ 1.5s

a1933d, 23 lines

```
const int LIM = 1e8;
bitset<LIM> isPrime;
vector<int> primeSieve() {
  const int S = round(sqrt(LIM)), R = LIM / 2;
  vector<int> pr = {2}, sieve(S + 1);
  pr.reserve(int(LIM / log(LIM) * 1.1));
  vector<pair<int, int>> cp;
  for (int i = 3; i <= S; i += 2)
    if (!sieve[i]) {
      cp.push_back({i, i * i / 2});
      for (int j = i * i; j <= S; j += 2 * i) sieve[j] = 1;
    }
  for (int L = 1; L <= R; L += S) {
    array<bool, S> block{};
    for (auto& [p, idx] : cp)
      for (int i = idx; i < S + L; idx = (i += p))
        block[i - L] = 1;
    for (int i = 0; i < min(S, R - L); i++)
      if (!block[i]) pr.push_back((L + i) * 2 + 1);
  }
  for (int i : pr) isPrime[i] = 1;
  return pr;
}
```

Miscellaneous (6)

NDimensionalVector.h

3c0f61, 12 lines

```
template <int D, typename T>
struct Vec : public vector<Vec<D - 1, T>> {
  static_assert(D >= 1,
                "Vector dimension must be greater than zero!");
  template <typename... Args>
  Vec(int n = 0, Args... args):
    vector<Vec<D - 1, T>>(n, Vec<D - 1, T>(args...)) {}
};
template <typename T>
struct Vec<1, T> : public vector<T> {
  Vec(int n = 0, const T& val = T()): vector<T>(n, val) {}
};
```

Submasks.h

35424b, 3 lines

```
for (int mask = 0; mask < (1 << n); mask++)
  for (int sub = mask; sub; sub = (sub - 1) & mask)
// do thing
```

Strings (7)

ZValues.h

151ee3, 10 lines

```
vector<int> zValues(string& s) {
  int n = ( int )s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```