



Team Contest Reference

Team: stoptryharding

Moritz Hoffmann

Ian Pösse

Marcel Wienöbst

Contents

1	ds	2
1.1	Fenwick-Tree	2
1.2	Range maximum query	2
1.3	Suffix array	2
1.4	trie	3
1.5	Union-Find	3
2	graph	4
2.1	2SAT	4
2.2	BellmanFord	4
2.3	bipartite graph check	4
2.4	Maximum Bipartite Matching	4
2.5	shortest path for dags	5
2.6	Recursive Depth First Search	5
2.7	Dijkstra	5
2.8	FloydWarshall	6
2.9	kruskal algorithm	6
2.10	EdmondsKarp	6
2.11	find min cut edges	6
2.12	strongly connected components	7
2.13	topological sort	7
3	math	7
3.1	binomial coefficient	7
3.2	Iterative EEA	7
3.3	Fourier transform	7
3.4	Greatest Common Divisor	8
3.5	geometry lib	8
3.6	Least Common Multiple	9
3.7	phi function calculator	9
3.8	Sieve of Eratosthenes	9
3.9	successive squaring	9
4	misc	9
4.1	Binary Search	9
4.2	comparator in C++	10
4.3	hashing pair in C++	10
4.4	knuth-morris-pratt	10
4.5	LongestIncreasingSubsequence	10
4.6	Next number with n bits set	10
5	more math	11
5.1	Tree	11
5.2	Divisability Explanation	11
5.3	Combinatorics	11
5.4	Polynomial Interpolation	11

5.4.1	Theory	11
5.5	Fibonacci Sequence	11
5.5.1	Binet's formula	11
5.5.2	Generalization	11
5.5.3	Pisano Period	11
5.6	Reihen	11
5.7	Binomialkoeffizienten	11
5.8	Catalanzahlen	11
5.9	Geometrie	11
5.10	Zahlentheorie	12
5.11	Faltung	12

1 ds

1.1 Fenwick-Tree

Can be used for computing prefix sums.

```

1 //note that 0 can not be used
2 //globaly create array
3 int fwktree[1000001];
4 int read(int index) {
5     int sum = 0;
6     while (index > 0) {
7         sum += fenwickTree[index];
8         index -= (index & -index);
9     }
10    return sum;
11 }
12 // n is the actual size of the tree (e.g. the array is
13    used from 1 to n-1)
14 void update(int index, int addValue, int n) {
15     while (index <= n - 1) {
16         fenwickTree[index] += addValue;
17         index += (index & -index);
18     }
19 }
```

MD5: 9f2366fa36268df7f3bf1ac4d3772f91 | $\mathcal{O}(\log n)$

1.2 Range maximum query

finds maximum in range $[i,j]$ in $\mathcal{O}(1)$ preprocessing takes $\mathcal{O}(n \log n)$

```

1 // create A globally, contains the input
2 int A[10000];
3 // M is the DP table has size N*log N
4 int M[10000][20];
5 // N is the input size
6 void process(int N) {
7     for(int i = 0; i < N; i++)
8         M[i][0] = i;
9     // filling table M
10    // M[i][j] = max(M[i][j-1], M[i+(1<<(j-1))][j-1]),
11    // cause interval of length 2^j can be partitioned
12    // into two intervals of length 2^(j-1)
13    for(int j = 1; 1 << j <= N; j++) {
14        for(int i = 0; i + (1 << j) - 1 < N; i++) {
15            if(A[M[i][j-1]] >= A[M[i+(1 << (j-1))][j-1]])
16                M[i][j] = M[i][j-1];
17            else
18                M[i][j] = M[i + (1 << (j-1))][j-1];
19        }
20    }
```

```

20 }
21 }
22 // range is [i,j], returns index of max
23 int query(int N, int i, int j) {
24     // k = |_log_2(j-i+1) _|
25     int k = (int) (log(j - i + 1) / log(2));
26     if(A[M[i][k]] >= A[M[j - (1 << k) + 1][k]])
27         return M[i][k];
28     else
29         return M[j - (1 << k) + 1][k];
30 }
```

MD5: eae61471981a55989f42aa6631bb2f13 | $\mathcal{O}(?)$

1.3 Suffix array

```

1 vector<int> sa, pos, tmp, lcp;
2 string s;
3 int N, gap;
4
5 bool sufCmp(int i, int j) {
6     if(pos[i] != pos[j])
7         return pos[i] < pos[j];
8     i += gap;
9     j += gap;
10    return (i < N && j < N) ? pos[i] < pos[j] : i > j;
11 }
12
13 void buildSA()
14 {
15     N = s.size();
16     for(int i = 0; i < N; ++i) {
17         sa.push_back(i);
18         pos.push_back(s[i]);
19     }
20     tmp.resize(N);
21     for(gap = 1; gap <= N; gap *= 2) {
22         sort(sa.begin(), sa.end(), sufCmp);
23         for(int i = 0; i < N - 1; ++i) {
24             tmp[i+1] = tmp[i] + sufCmp(sa[i], sa[i+1]);
25         }
26         for(int i = 0; i < N; ++i) {
27             pos[sa[i]] = tmp[i];
28         }
29         if(tmp[N-1] == N-1) break;
30     }
31 }
32
33 void buildLCP()
34 {
35     lcp.resize(N);
36     for(int i = 0, k = 0; i < N; ++i) {
```

```

37     if(pos[i] != N - 1) {
38         for(int j = sa[pos[i] + 1]; s[i + k] == s[j + k]
39             );) {
40             ++k;
41         }
42         lcp[pos[i]] = k;
43         if (k) --k;
44     }
45 }
46
47 int main()
48 {
49     string r, t;
50     cin >> r >> t;
51     s = r + "$" + t;
52     buildSA();
53     buildLCP();
54     for(int i = 0; i < N; ++i) {
55         cout << sa[i] << " " << lcp[i] << endl;
56     }
57     //suffix arrays can be used for various things:
58     //for example: finding lcs between two strings
59 }

```

MD5: 47eb870ecfe9cb548eb96a15c077fab7 | $\mathcal{O}(?)$

1.4 trie

source: github -> SuprDewd

```

1 template <class T>
2 struct trie {
3     struct node {
4         map<T, node*> children;
5         int prefixes, words;
6         node() { prefixes = words = 0; } };
7     node* root;
8     trie() : root(new node()) { }
9     template <class I>
10    void insert(I begin, I end) {
11        node* cur = root;
12        while (true) {
13            cur->prefixes++;
14            if (begin == end) { cur->words++; break; }
15            else {
16                T head = *begin;
17                typename map<T, node*>::const_iterator it;
18                it = cur->children.find(head);
19                if (it == cur->children.end()) {
20                    pair<T, node*> nw(head, new node());
21                    it = cur->children.insert(nw).first;
22                    } begin++, cur = it->second; } } }
23    template<class I>
24    int countMatches(I begin, I end) {
25        node* cur = root;
26        while (true) {
27            if (begin == end) return cur->words;
28            else {
29                T head = *begin;
30                typename map<T, node*>::const_iterator it;
31                it = cur->children.find(head);
32                if (it == cur->children.end()) return 0;
33                begin++, cur = it->second; } } }
34    template<class I>
35    int countPrefixes(I begin, I end) {
36        node* cur = root;

```

```

37    while (true) {
38        if (begin == end) return cur->prefixes;
39        else {
40            T head = *begin;
41            typename map<T, node*>::const_iterator it;
42            it = cur->children.find(head);
43            if (it == cur->children.end()) return 0;
44            begin++, cur = it->second; } } }
45
46    // use as follows
47    trie<char> t;
48    string s = "aaa";
49    t.insert("aaa");

```

MD5: 4410c62ce77f58cf564ac6881096d200 | $\mathcal{O}(?)$

1.5 Union-Find

union joins the sets x and y are contained in. *find* returns the representative of the set x is contained in.

Input: number of elements n , element x , element y

Output: the representative of element x or a boolean indicating whether sets got merged.

```

1 // globally create arrays
2 int p[100000];
3 int r[100000];
4
5 int count() {
6     return count;
7 } // number of sets
8
9 int find(int x) {
10    int root = x;
11    while (p[root] >= 0) { // find root
12        root = p[root];
13    }
14    while (p[x] >= 0) { // path compression
15        tmp = p[x];
16        p[x] = root;
17        x = tmp;
18    }
19    return root;
20 }
21
22 // return true, if sets merged and false, if already
23 // from same set
24 bool union(int x, int y) {
25     int px = find(x);
26     int py = find(y);
27     if (px == py)
28         return false; // same set -> reject edge
29     if (r[px] < r[py]) { // swap so that always h[px]
30         ]>=h[py]
31     }
32     int tmp = px;
33     px = py;
34     py = tmp;
35     p[py] = px; // hang flatter tree as child of
36     // higher tree
37     r[px] = max(r[px], r[py] + 1); // update (worst-
38     // case) height
39     count--;
40     return true;
41 }

```

```

39 int main() {
40     // init count to number of nodes
41     int count = n;
42
43     for(int i = 0; i < n; ++i) {
44         p[i] = -1;
45     }
46     // do something
47 }

```

MD5: e5cb75e4854c060b0e08655fec44ae8 | $\mathcal{O}(\alpha(n))$

2 graph

2.1 2SAT

```

1 // create implication graph
2 // do SCC
3 // check if var and its negation are in the same
  component

```

MD5: a2e8b2ae500366ce942af79e0a3f4283 | $\mathcal{O}(V + E)$

2.2 BellmanFord

Finds shortest paths from a single source. Negative edge weights are allowed. Can be used for finding negative cycles.

```

1 // globally create arrays and graph
2 vector<vector<pair<int, int>>> g;
3 int dist[n];
4 int MAX_VALUE = (1 << 30);
5
6 bool bellmanFord() {
7     //source is 0
8     dist[0] = 0;
9     //calc distances
10    //the path has max length |V|-1
11    for(int i = 0; i < n-1; i++) {
12        //each iteration relax all edges
13        for(int j = 0; j < n; j++) {
14            for(int k = 0; k < g[j].size(); ++k) {
15                pair<int, int> e = g[j][k];
16                if(dist[j] != MAX_VALUE
17                    && dist[e.first] > dist[j] + e.second) {
18                    dist[e.first] = dist[j] + e.second;
19                }
20            }
21        }
22    }
23    //check for negative-weight cycle
24    for(int i = 0; i < n; i++) {
25        for(int j = 0; j < g[i].size(); ++j) {
26            if(dist[i] != Integer.MAX_VALUE
27                && dist[e.first] > dist[i] + e.second) {
28                return true;
29            }
30        }
31    }
32    return false;
33 }

```

MD5: 0dfb4089a47db73dbaad5add58fd2a0 | $\mathcal{O}(|V| \cdot |E|)$

2.3 bipartite graph check

```

1 // traverse through graph with bfs
2 // assign labels 0 and 1
3 // if child is unexplored it gets different label from
  parent and put in the queue
4 // if already visited check if labels are different

```

MD5: 0b64ac42e8b846e97c338eae7d73575 | $\mathcal{O}(?)$

2.4 Maximum Bipartite Matching

Finds the maximum bipartite matching in an unweighted graph using DFS.

Input: An unweighted adjacency matrix `boolean[M][N]` with M nodes being matched to N nodes.

Output: The maximum matching. (For getting the actual matching, little changes have to be made.)

```

1 // globally create graph array
2 // adjacency matrix but smaller as only edges between
  M and N exist
3 bool bpGraph[M][N];
4
5 // A DFS based recursive function
6 // that returns true if a matching
7 // for vertex u is possible
8 bool bpm(int u, vector<bool> &seen, vector<int> &
  matchR)
9 {
10    // Try every job one by one
11    for (int v = 0; v < N; v++)
12    {
13        // If applicant u is interested in
14        // job v and v is not visited
15        if (bpGraph[u][v] && !seen[v])
16        {
17            // Mark v as visited
18            seen[v] = true;
19
20            // If job v is not assigned to an
21            // applicant OR previously assigned
22            // applicant for job v (which is matchR[v]
23            // )
24            // has an alternate job available.
25            // Since v is marked as visited in
26            // the above line, matchR[v] in the
27            // following
28            // recursive call will not get job v again
29            if (matchR[v] < 0 || bpm(bpGraph, matchR[v]
30            ],
31                seen, matchR))
32            {
33                matchR[v] = u;
34                return true;
35            }
36        }
37    }
38    return false;
39 }
40
41 // Returns maximum number
42 // of matching from M to N
43 int maxBPM()
44 {
45     // An array to keep track of the

```

```

43 // applicants assigned to jobs.
44 // The value of matchR[i] is the
45 // applicant number assigned to job i,
46 // the value -1 indicates nobody is
47 // assigned.
48 vector<int> matchR (N);
49
50 // Initially all jobs are available
51 for(int i = 0; i < N; ++i) {
52 matchR[i] = -1;
53 }
54
55 // Count of jobs assigned to applicants
56 int result = 0;
57 for (int u = 0; u < M; u++)
58 {
59     // Mark all jobs as not seen
60     // for next applicant.
61     vector<int> seen (N);
62
63     // Find if the applicant u can get a job
64     if (bpm(bpGraph, u, seen, matchR))
65         result++;
66 }
67 return result;
68 }

```

MD5: 035f3ecf4735d724aad793ac4c1417c3 | $\mathcal{O}(M \cdot N)$

2.5 shortest path for dags

can also be applied to longest path problem in dags

```

1 // calc topological sorting
2 // go through nodes in ts order
3 // relaxate its neighbours

```

MD5: 337da9f825b3decf382ab7a8278b025c | $\mathcal{O}(?)$

2.6 Recursive Depth First Search

Recursive DFS with different options (storing times, connected/unconnected graph). this is very much pseudocode, needs a lot of problem adaption anyway

Input: A source vertex s , a target vertex t , and adjlist G and the time (0 at the start)

Output: Indicates if there is connection between s and t .

```

1 // globally create adj list etc
2 vector<vector<int>> g;
3 int dtime[n];
4 int ftime[n];
5 int vis[n];
6 int pre[n];
7 //first call with time = 0
8 void rec_dfs(int u, int time){
9     //it might be necessary to store the time of
10     //discovery
11     time = time + 1;
12     dtime[u] = time;
13
14     vis[u] = 1; //new vertex has been discovered
15     //For cycle check vis should be int and 0 are not
16     //vis nodes

```

```

15 //1 are vis nodes which havent been finished and 2
16 //are finished nodes
17 //cycle exists iff edge to node with vis=1
18 //when reaching the target return true
19 //not necessary when calculating the DFS-tree
20 for(int i = 0; i < g[u].size(); ++i) {
21     int v = g[u][i];
22     //exploring a new edge
23     if(!vis[v]) {
24         pre[v] = u;
25         if(rec_dfs(v, time)) return true;
26     }
27 }
28 //storing finishing time
29 time = time + 1;
30 ftime[s] = time;
31 vis[s] = 2;
32 return false;
33 }
34
35 //if we want to visit the whole graph, even if it is
36 //not connected we might use this
37 //make sure all vertices vis value is false etc
38 int time = 0;
39 for(int i = 0; i < n; i++) {
40     if(vis[i]) {
41         //note that we leave out t so this does not work
42         //with the below function
43         //adaption will not be too difficult though
44         //time should not always start at zero, change if
45         //needed
46         rec_dfs(i, 0);
47     }
48 }
49 }

```

MD5: c7de745b3c11151bfa0c9093b827cefc | $\mathcal{O}(|V| + |E|)$

2.7 Dijkstra

Finds the shortest paths from one vertex to every other vertex in the graph (SSSP).

For negative weights, add $|\min|+1$ to each edge, later subtract from result.

To get a different shortest path when edges are ints, add an $\varepsilon = \frac{1}{k+1}$ on each edge of the shortest path of length k , run again.

Input: A source vertex s and an adjacency list G .

Output: Modified adj. list with distances from s and predecessor vertices set.

```

1 int mxi = (1 << 25);
2
3 bool cmp(pair<int, int> a, pair<int, int> b)
4 {
5     // unclear if it should be > or <
6     return (a.second > b.second);
7 }
8
9 int dijkstra(vector<vector<pair<int, int>>> &g, int N)
10 {
11     priority_queue<pair<int, int>, vector<pair<int,
12         int>>, decltype(cmp)> *p> pq(cmp);
13     vector<int> dist (N, mxi);
14     dist[0] = 0;
15     pq.push({0, 0});

```

```

15 while(!pq.empty()) {
16     int u = pq.top().first;
17     int d = pq.top().second;
18     pq.pop();
19     if(d > dist[u]) continue;
20     if(u == N-1) return d;
21     for(auto it = g[u].begin(); it != g[u].end();
22         ++it) {
23         int v = it->first;
24         int w = it->second;
25         if(w + dist[u] < dist[v]) {
26             dist[v] = w + dist[u];
27             pq.push({v, dist[v]});
28         }
29     }
30     return dist[N-1];
31 }

```

MD5: d18a3423468af48dac03a85b22b35dec | $\mathcal{O}(|E| \log |V|)$

2.8 FloydWarshall

Finds all shortest paths. Paths in array next, distances in ans.

```

1 int MAX_VALUE = (1 << 30);
2
3 void floydWarshall(int[][] graph,
4     int[][] next, int[][] ans, int n) {
5     for(int i = 0; i < n; i++)
6         for(int j = 0; j < n; j++)
7             ans[i][j] = graph[i][j];
8
9     for (int k = 0; k < n; k++)
10        for (int i = 0; i < n; i++)
11            for (int j = 0; j < n; j++)
12                if (ans[i][k] + ans[k][j] < ans[i][j]
13                    && ans[i][k] < MAX_VALUE
14                    && ans[k][j] < MAX_VALUE) {
15                    ans[i][j] = ans[i][k] + ans[k][j];
16                    next[i][j] = next[i][k];
17                }
18 }

```

MD5: d93432a80b6b67952eedde97a4e7df79 | $\mathcal{O}(|V|^3)$

2.9 kruskal algorithm

finds the minimum spanning tree

```

1 // sort edges by increasing weight
2 // init union find (the nodes are the sets)
3 // go through the sorted edges and check if the
4   corresponding nodes
5 // are in the same set, if yes skip the edge, if no
6   the edge is part
7 // of the minimum spanning tree -> unite nodes

```

MD5: 82c91537f2425cfed1809d2f685dafcd | $\mathcal{O}(?)$

2.10 EdmondsKarp

Finds the greatest flow in a graph. Capacities must be positive.

```

1 #include<iostream>
2 #include<vector>
3 #include<queue>
4 #include<unordered_map>
5 #include<cmath>
6
7 using namespace std;
8
9 bool bfs(vector<unordered_map<int, long long>> &g, int
10     s, int t, vector<int> &pre)
11 {
12     int n = g.size();
13     for(int i = 0; i < n; ++i) {
14         pre[i] = -1;
15     }
16     vector<bool> vis (n);
17     queue<int> q;
18     vis[s] = true;
19     q.push(s);
20     while(!q.empty()) {
21         int u = q.front();
22         q.pop();
23         if(u == t) return true;
24         for(auto v = g[u].begin(); v != g[u].end(); ++
25             v) {
26             if(!vis[v->first] && (v->second) > 0) {
27                 vis[v->first] = true;
28                 pre[v->first] = u;
29                 q.push(v->first);
30             }
31         }
32     }
33     return vis[t];
34 }
35
36 long long ed_karp(vector<unordered_map<int, long long
37     >> &g, int s, int t)
38 {
39     long long mxf = 0;
40     int n = g.size();
41     vector<int> pre (n);
42     while(bfs(g, s, t, pre)) {
43         long long pf = (1L << 58);
44         for(int v = t; v != s; v = pre[v]) {
45             int u = pre[v];
46             pf = min(pf, g[u][v]);
47         }
48         for(int v = t; v != s; v = pre[v]) {
49             int u = pre[v];
50             g[u][v] -= pf;
51             g[v][u] += pf;
52         }
53         mxf += pf;
54     }
55     return mxf;
56 }

```

MD5: 7ea28f50383117106939588171692efe | $\mathcal{O}(|V|^2 \cdot |E|)$

2.11 find min cut edges

```

1 // do a maxflow
2 // go through residual graph with dfs or bfs
3   traversing edges with residual cap > 0 and
4   // back edges with flow > 0, mark all visited nodes

```

```
4 // then output all edges from a marked to an unmarked
   node (maybe another BFS or something)
```

MD5: fb27cd04a3f1ab0ea7e494c40be18fbe | $\mathcal{O}(?)$

2.12 strongly connected components

```
1 // use two DFSs
2 // 1. DFS: topological sort produces list l
3 // 2. DFS: go through sorting and for transposed graph
   (edges are flipped) do the DFS, all reached nodes
   get the same label (are in the same component),
   of course BFS could also be used
```

MD5: 8ba4235a4fe35b79c0c3d4a86341c525 | $\mathcal{O}(?)$

2.13 topological sort

```
1 //two options:
2 //1. remove nodes with in-degree 0
3 //2. do DFS and prepend nodes to list when they are
   done
4 // (so all the nodes they depend on have already been
   prepended as they already finished)
```

MD5: db8519c36fbafe6a952fa5c808a5932e | $\mathcal{O}(?)$

3 math

3.1 binomial coefficient

gives binomial coefficient (n choose k)

```
1 // note that if we have to calculate the bin coeff
   modulo some prime
2 // we cannot divide, but have to multiply by the
   inverse of k
3 // that can be easily computed as  $k^{p-2} \pmod p$  with
   modular exponentiation (use successive squaring)
4 // another approach would be to just calculate  $n! / ((n-k)! * k!)$  (again invert denominator and use mod in
   all steps)
5 long long bin(int n, int k) {
6     if (k == 0)
7         return 1;
8     else if (k > n/2)
9         return bin(n, n-k);
10    else
11        return n*bin(n-1, k-1)/k;
12 }
```

MD5: 610ff61f07eef70ca116e75e1b15cf7c | $\mathcal{O}(k)$

3.2 Iterative EEA

Calculates the gcd of a and b and their modular inverse $x = a^{-1} \pmod b$ and $y = b^{-1} \pmod a$.

```
1 // extended euclidean algorithm - iterativ
2 if (b > a) {
3     long tmp = a;
4     a = b;
```

```
5     b = tmp;
6 }
7 long x = 0, y = 1, u = 1, v = 0;
8 while (a != 0) {
9     long q = b / a, r = b % a;
10    long m = x - u * q, n = y - v * q;
11    b = a; a = r; x = u; y = v; u = m; v = n;
12 }
13 long gcd = b;
14 // x =  $a^{-1} \pmod b$ , y =  $b^{-1} \pmod a$ 
15 // ax + by = gcd
```

MD5: 737c57d8f09d748f54c57851ea1e759d | $\mathcal{O}(\log a + \log b)$

3.3 Fourier transform

calculates the fourier transform for a given vector here used for polynom multiplication in $\mathcal{O}(n \log n)$

```
1 // pol is the vector that should be transformed
2 // fft is the resulting vector (note the complex
   numbers)
3 // n is the size of pol and fft which has to be of the
   form  $2^k$  (just fill up with zeros and choose big
   enough size)
4 // if inv = true the inverse transform is calculated (
   here too the result can be found in fft!)
5 void iterativefft(const vector<long long> &pol, vector
   <complex<double>> &fft, int n, bool inv)
6 {
7     //copy pol into fft
8     if(!inv) {
9         for(int i = 0; i < n; ++i) {
10            complex<double> cp (pol[i], 0);
11            fft[i] = cp;
12        }
13    }
14    //swap positions accordingly
15    for(int i = 0, j = 0; i < n; ++i) {
16        if(i < j) swap(fft[i], fft[j]);
17        int m = n >> 1;
18        while(1 <= m && m <= j) j -= m, m >>= 1;
19        j += m;
20    }
21    for(int m = 1; m <= n; m <= 1) { //<= or <
22        double theta = (inv ? -1 : 1) * 2 * M_PI / m;
23        complex<double> wm(cos(theta), sin(theta));
24        for(int k = 0; k < n; k += m) {
25            complex<double> w = 1;
26            for(int j = 0; j < m/2; ++j) {
27                complex<double> t = w * fft[k + j + m
28                    /2];
29                complex<double> u = fft[k + j];
30                fft[k + j] = u + t;
31                fft[k + j + m/2] = u - t;
32                w = w*wm;
33            }
34        }
35    }
36    if(inv) {
37        for(int i = 0; i < n; ++i) {
38            fft[i] /= complex<double> (n);
39        }
40    }
41    // the polynom pol gets squared, the result is put in
   res
```



```

42 vector<complex<double>> fft (n);
43 iterativefft(pol, fft, n, false);
44 for(int i = 0; i < n; ++i) {
45     fft[i] *= fft[i];
46 }
47 iterativefft(pol, fft, n, true);
48 vector<long long> res(n);
49 for(int i = 0; i < n; ++i) {
50     res[i] = round(fft[i].real());
51 }

```

MD5: 9dd418b1bc3d7685c5c55b287cc8555e | $\mathcal{O}(?)$

3.4 Greatest Common Divisor

Calculates the gcd of two numbers a and b or of an array of numbers *input*.

Input: Numbers a and b or array of numbers *input*

Output: Greatest common divisor of the input

```

1 long long gcd(long long a, long long b) {
2     while (b > 0) {
3         long long temp = b;
4         b = a % b; // % is remainder
5         a = temp;
6     }
7     return a;
8 }
9
10 long long gcd(vector<long long> &input) {
11     long long result = input[0];
12     for(int i = 1; i < input.size(); i++)
13         result = gcd(result, input[i]);
14     return result;
15 }

```

MD5: 27f69f32d6e1f59d16b9c8ea0028a9fb | $\mathcal{O}(\log a + \log b)$

3.5 geometry lib

```

1 // this library has been copied from https://github.
  // com/SuprDewd/T-414-AFLV
2 #include <complex>
3 using namespace std;
4 #define P(p) const point &p
5 #define L(p0, p1) P(p0), P(p1)
6 #define C(p0, r) P(p0), double r
7 #define PP(pp) pair<point,point> &pp
8 typedef complex<double> point;
9 const double pi = acos(-1.0);
10 const double EPS = 1e-9;
11 double dot(P(a), P(b)) {
12     return real(conj(a) * b);
13 }
14 double cross(P(a), P(b)) {
15     return imag(conj(a) * b);
16 }
17 point rotate(P(p), double radians = pi / 2, P(about) =
  point(0,0)) {
18     return (p - about) * exp(point(0, radians)) +
  about;
19 }
20 point proj(P(u), P(v)) {
21     return dot(u, v) / dot(u, u) * u;
22 }

```

```

23 point normalize(P(p), double k = 1.0) {
24     return abs(p) == 0 ? point(0,0) : p / abs(p) * k;
25 }
26 bool parallel(L(a, b), L(p, q)) {
27     return abs(cross(b - a, q - p)) < EPS;
28 }
29 double ccw(P(a), P(b), P(c)) {
30     return cross(b - a, c - b);
31 }
32 bool collinear(P(a), P(b), P(c)) { return abs(ccw(a, b
  , c)) < EPS; }
33 double angle(P(a), P(b), P(c)) {
34     return acos(dot(b - a, c - b) / abs(b - a) / abs(c
  - b));
35 }
36 bool intersect(L(a, b), L(p, q), point &res, bool
  segment = false) {
37     // NOTE: check for parallel/collinear lines before
  calling this function
38     point r = b - a, s = q - p;
39     double c = cross(r, s), t = cross(p - a, s) / c, u
  = cross(p - a, r) / c;
40     if (segment && (t < 0-EPS || t > 1+EPS || u < 0-
  EPS || u > 1+EPS))
41         return false;
42     res = a + t * r;
43     return true;
44 }
45 point closest_point(L(a, b), P(c), bool segment =
  false) {
46     if (segment) {
47         if (dot(b - a, c - b) > 0) return b;
48         if (dot(a - b, c - a) > 0) return a;
49     }
50     double t = dot(c - a, b - a) / norm(b - a);
51     return a + t * (b - a);
52 }
53
54 typedef vector<point> polygon;
55 #define MAXN 1000
56 point hull[MAXN];
57 bool cmp(const point &a, const point &b) {
58     return abs(real(a) - real(b)) > EPS ?
59         real(a) < real(b) : imag(a) < imag(b); }
60 int convex_hull(vector<point> p) {
61     int n = p.size(), l = 0;
62     sort(p.begin(), p.end(), cmp);
63     for (int i = 0; i < n; i++) {
64         if (i > 0 && p[i] == p[i - 1])
65             continue;
66         while (l >= 2 && ccw(hull[l - 2], hull[l - 1],
  p[i]) >= 0)
67             l--;
68         hull[l++] = p[i];
69     }
70     int r = l;
71     for (int i = n - 2; i >= 0; i--) {
72         if (p[i] == p[i + 1])
73             continue;
74         while (r - l >= 1 && ccw(hull[r - 2], hull[r -
  1], p[i]) >= 0)
75             r--;
76         hull[r++] = p[i];
77     }
78     return l == 1 ? 1 : r - 1;
79 }

```

MD5: 3563f20cd2010aee48a137414d73506c | $\mathcal{O}(?)$

3.6 Least Common Multiple

Calculates the lcm of two numbers a and b or of an array of numbers $input$.

Input: Numbers a and b or array of numbers $input$

Output: Least common multiple of the input

```
1 long long lcm(long long a, long long b) {
2     return a * (b / gcd(a, b));
3 }
4
5 long long lcm(vector<long long> &input) {
6     long result = input[0];
7     for(int i = 1; i < input.size(); i++)
8         result = lcm(result, input[i]);
9     return result;
10 }
```

MD5: f9b4919c74ef3ca9c1e0e2964d59fd7b | $\mathcal{O}(\log a + \log b)$

3.7 phi function calculator

takes \sqrt{n} time

```
1 int phi(int n)
2 {
3     double result = n;
4     for(int p = 2; p * p <= n; ++p) {
5         if(n % p == 0) {
6             while(n % p == 0) n /= p;
7             result *= (1.0 - (1.0 / (double) p));
8         }
9     }
10    if(n > 1) result *= (1.0 - (1.0 / (double) n));
11    return round(result);
12 }
```

MD5: 2ec930cc10935f1638700bb74e3439d9 | $\mathcal{O}(?)$

3.8 Sieve of Eratosthenes

Calculates Sieve of Eratosthenes.

Input: A integer N indicating the size of the sieve.

Output: A boolean array, which is true at an index i iff i is prime.

```
1 vector<boolean> is_prime(n+1);
2 for(int i = 2; i <= n; i++) is_prime[i] = true;
3 for(int i = 2; i*i <= n; i++)
4     if(is_prime[i])
5         for(int j = i*i; j <= n; j+=i)
6             is_prime[j] = false;
```

MD5: 2b965443a98027ed7f531d5360e00b48 | $\mathcal{O}(n)$

3.9 successive squaring

calculates g^L here shown for matrix mult, but can be applied in other cases

```
1 void mult(int a[][nos], int b[][nos], int N)
2 {
3     int res[nos][nos] = {0};
4     for(int i = 0; i < N; i++) {
5         for(int j = 0; j < N; j++) {
```

```
        for(int k = 0; k < N; k++) {
            res[i][j] = (res[i][j] + a[i][k]*b[k][j]) % 10000;
        }
    }
}
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        a[i][j] = res[i][j];
    }
}
// res stores the result and is initialized to the
// identity matrix
int res[nos][nos] = {0};
for(int i = 0; i < N; i++) {
    for(int j = 0; j < N; j++) {
        if(i == j) res[i][j] = 1;
    }
}
for(int i = 0; (1 << i) <= L; i++) {
    if(((1 << i) & L) == (1 << i)) {
        mult(res, g, N);
    }
    mult(g, g, N);
}
```

MD5: f86c0e996e5eec0aedce9308951f2ddc | $\mathcal{O}(?)$

4 misc

4.1 Binary Search

Binary searches for an element in a sorted array.

Input: sorted *array* to search in, amount N of elements in *array*, element to search for a

Output: returns the index of a in *array* or -1 if *array* does not contain a

```
1 int lo = 0;
2 int hi = N-1;
3 // a might be in interval [lo,hi] while lo <= hi
4 while(lo <= hi) {
5     int mid = (lo + hi) / 2;
6     // if a > elem in mid of interval,
7     // search the right subinterval
8     if(array[mid] < a)
9         lo = mid+1;
10    // else if a < elem in mid of interval,
11    // search the left subinterval
12    else if(array[mid] > a)
13        hi = mid-1;
14    // else a is found
15    else
16        return mid;
17 }
18 // array does not contain a
19 return -1;
```

MD5: 2049104cd8aaced6ba8de166e9bd2abe | $\mathcal{O}(\log n)$

4.2 comparator in C++

```

1 bool myfunction (int i, int j) {return (i<j); }
2
3 int main() {
4     vector<int> vec;
5     sort(vec.begin(), vec.end(), myfunction);
6     priority_queue<int, vector<int>, decltype(
7         myfunction) *> pq(myfunction);
8 }

```

MD5: f4beb6e197be08977fd4f74b2537ae09 | $\mathcal{O}(?)$

4.3 hashing pair in C++

```

1 struct pairhash {
2     public:
3         template <typename T, typename U>
4         std::size_t operator()(const std::pair<T, U> &x)
5             const
6         {
7             return std::hash<T>()(x.first) ^ std::hash<U>()(x.
8                 second);
9         }
10 };
11
12 int main() {
13     unordered_map<pair<unsigned int, char>, double,
14         pairhash> T;
15 }

```

MD5: 49bde857f5a8078349cf97308bd8144c | $\mathcal{O}(?)$

4.4 knuth-morris-pratt

finds pattern in a string

```

1 //-----
2 // Returns a vector containing the zero based index of
3 // the start of each match of the string K in S.
4 // Matches may overlap
5 // source: wikipedia
6 //-----
7 vector<int> KMP(string S, string K)
8 {
9     vector<int> T(K.size() + 1, -1);
10    vector<int> matches;
11
12    if (K.size() == 0) {
13        matches.push_back(0);
14        return matches;
15    }
16
17    for (int i = 1; i <= K.size(); i++) {
18        int pos = T[i - 1];
19        while (pos != -1 && K[pos] != K[i - 1])
20            pos = T[pos];
21        T[i] = pos + 1;
22    }
23
24    int sp = 0;
25    int kp = 0;
26    while (sp < S.size()) {
27        while (kp != -1 && (kp == K.size() || K[kp] !=
28            S[sp]))

```

```

28        kp = T[kp];
29        kp++;
30        sp++;
31        if (kp == K.size())
32            matches.push_back(sp - K.size());
33    }
34
35    return matches;
36 }

```

MD5: 856843d59319d4adac8e62968cc7ccf0 | $\mathcal{O}(?)$

4.5 LongestIncreasingSubsequence

Input: array *arr* containing a sequence and empty array *p* of length *arr.length* for storing indices of the LIS

Output: array *s* containing the longest increasing subsequence

```

1 // p[k] stores index of the predecessor of arr[k]
2 // in the LIS ending at arr[k]
3 // m[j] stores index k of smallest value arr[k]
4 // so there is a LIS of length j ending at arr[k]
5 int m[n+1];
6 int l = 0;
7 for(int i = 0; i < n; i++) {
8     // bin search for the largest positive j <= l
9     // with arr[m[j]] < arr[i]
10    int lo = 1;
11    int hi = l;
12    while(lo <= hi) {
13        int mid = (int) ((lo + hi) / 2.0) + 0.6);
14        if(arr[m[mid]] <= arr[i])
15            lo = mid+1;
16        else
17            hi = mid-1;
18    }
19    // lo is 1 greater than length of the
20    // longest prefix of arr[i]
21    int newL = lo;
22    p[i] = m[newL-1];
23    m[newL] = i;
24    // if LIS found is longer than the ones
25    // found before, then update l
26    if(newL > l)
27        l = newL;
28 }
29 // reconstruct the LIS
30 vector<int> s (l);
31 int k = m[l];
32 for(int i = l-1; i >= 0; i--) {
33     s[i] = arr[k];
34     k = p[k];
35 }
36 //s is the resulting seq

```

MD5: 8eb64842ea26475286a264c3557c355d | $\mathcal{O}(n \log n)$

4.6 Next number with n bits set

From *x* the smallest number greater than *x* with the same amount of bits set is computed. Little changes have to be made, if the calculated number has to have length less than 32 bits.

Input: number *x* with *n* bits set ($x = (1 \ll n) - 1$)

Output: the smallest number greater than *x* with *n* bits set

```

1 int nextNumber(int x) {
2     //break when larger than limit here
3     if(x == 0) return 0;
4     int smallest = x & -x;
5     int ripple = x + smallest;
6     int new_smallest = ripple & -ripple;
7     int ones = ((new_smallest/smallest) >> 1) - 1;
8     return ripple | ones;
9 }

```

MD5: a70e3ab92156018533fa25fea2297214 | $\mathcal{O}(1)$

5 more math

5.1 Tree

Diameter: BFS from any node, then BFS from last visited node.
Max dist is then the diameter. Center: Middle vertex in second step from above.

5.2 Divisability Explanation

$D \mid M \Leftrightarrow D \mid \text{digit_sum}(M, k, \text{alt})$, refer to table for values of D, k, alt .

5.3 Combinatorics

- Variations (ordered): k out of n objects (permutations for $k = n$)
 - without repetition:

$$M = \{(x_1, \dots, x_k) : 1 \leq x_i \leq n, x_i \neq x_j \text{ if } i \neq j\},$$

$$|M| = \frac{n!}{(n-k)!}$$
 - with repetition:

$$M = \{(x_1, \dots, x_k) : 1 \leq x_i \leq n\}, |M| = n^k$$
- Combinations (unordered): k out of n objects
 - without repetition: $M = \{(x_1, \dots, x_n) : x_i \in \{0, 1\}, x_1 + \dots + x_n = k\}, |M| = \binom{n}{k}$
 - with repetition: $M = \{(x_1, \dots, x_n) : x_i \in \{0, 1, \dots, k\}, x_1 + \dots + x_n = k\}, |M| = \binom{n+k-1}{k-1}$
- Ordered partition of numbers: $x_1 + \dots + x_k = n$ (i.e. $1+3 = 3+1 = 4$ are counted as 2 solutions)
 - #Solutions for $x_i \in \mathbb{N}_0$: $\binom{n+k-1}{k-1}$
 - #Solutions for $x_i \in \mathbb{N}$: $\binom{n-1}{k-1}$
- Unordered partition of numbers: $x_1 + \dots + x_k = n$ (i.e. $1+3 = 3+1 = 4$ are counted as 1 solution)
 - #Solutions for $x_i \in \mathbb{N}$: $P_{n,k} = P_{n-k,k} + P_{n-1,k-1}$ where $P_{n,1} = P_{n,n} = 1$
- Derangements (permutations without fixed points): $!n = n! \sum_{k=0}^n \frac{(-1)^k}{k!} = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$

5.4 Polynomial Interpolation

5.4.1 Theory

Problem: for $\{(x_0, y_0), \dots, (x_n, y_n)\}$ find $p \in \Pi_n$ with $p(x_i) = y_i$ for all $i = 0, \dots, n$.

Solution: $p(x) = \sum_{i=0}^n \gamma_{0,i} \prod_{j=0}^{i-1} (x - x_j)$ where $\gamma_{j,k} = y_j$ for $k = 0$

and $\gamma_{j,k} = \frac{\gamma_{j+1,k-1} - \gamma_{j,k-1}}{x_{j+k} - x_j}$ otherwise.

Efficient evaluation of $p(x)$: $b_n = \gamma_{0,n}$, $b_i = b_{i+1}(x - x_i) + \gamma_{0,i}$ for $i = n-1, \dots, 0$ with $b_0 = p(x)$.

5.5 Fibonacci Sequence

5.5.1 Binet's formula

$\begin{pmatrix} f_n \\ f_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow f_n = \frac{1}{\sqrt{5}}(\phi^n - \tilde{\phi}^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$ and $\tilde{\phi} = \frac{1-\sqrt{5}}{2}$.

5.5.2 Generalization

$g_n = \frac{1}{\sqrt{5}}(g_0(\phi^{n-1} - \tilde{\phi}^{n-1}) + g_1(\phi^n - \tilde{\phi}^n)) = g_0 f_{n-1} + g_1 f_n$ for all $g_0, g_1 \in \mathbb{N}_0$

5.5.3 Pisano Period

Both $(f_n \bmod k)_{n \in \mathbb{N}_0}$ and $(g_n \bmod k)_{n \in \mathbb{N}_0}$ are periodic.

5.6 Reihen

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=0}^n c^i = \frac{c^{n+1}-1}{c-1}, c \neq 1, \sum_{i=0}^{\infty} c^i = \frac{1}{1-c}, \sum_{i=1}^n c^i = \frac{c}{1-c}, |c| < 1$$

$$\sum_{i=0}^n i c^i = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}, c \neq 1, \sum_{i=0}^{\infty} i c^i = \frac{c}{(1-c)^2}, |c| < 1$$

5.7 Binomialkoeffizienten

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}, \binom{n}{m} \binom{m}{k} = \binom{n}{k} \binom{n-k}{m-k},$$

$$\binom{m+n}{r} = \sum_{k=0}^r \binom{m}{k} \binom{n}{r-k} \text{ and in general, } n_1 + \dots + n_p = \sum_{k_1+\dots+k_p=m} \binom{n_1}{k_1} \dots \binom{n_p}{k_p}$$

5.8 Catalanzahlen

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, C_{n+1} = \sum_{k=0}^n C_k C_{n-k}, C_{n+1} = \frac{4n+2}{n+2} C_n$$

5.9 Geometrie

Polygonfläche: $A = \frac{1}{2}(x_1 y_2 - x_2 y_1 + x_2 y_3 - x_3 y_2 + \dots + x_{n-1} y_n - x_n y_{n-1} + x_n y_1 - x_1 y_n)$

5.10 Zahlentheorie

Chinese Remainder Theorem: Es existiert eine Zahl C , sodass:

$$C \equiv a_1 \pmod{n_1}, \dots, C \equiv a_k \pmod{n_k}, \text{ggT}(n_i, n_j) = 1, i \neq j$$

Fall $k = 2$: $m_1 n_1 + m_2 n_2 = 1$ mit EEA finden.

Lösung ist $x = a_1 m_2 n_2 + a_2 m_1 n_1$.

Allgemeiner Fall: iterative Anwendung von $k = 2$

Eulersche φ -Funktion: $\varphi(n) = n \prod_{p|n} (1 - \frac{1}{p}), p \text{ prim}$

$$\varphi(p) = p - 1, \varphi(pq) = \varphi(p)\varphi(q), p, q \text{ prim}$$

$$\varphi(p^k) = p^k - p^{k-1}, p, q \text{ prim}, k \geq 1$$

Eulers Theorem: $a^{\varphi(n)} \equiv 1 \pmod{n}$

Fermats Theorem: $a^p \equiv a \pmod{p}, p \text{ prim}$

5.11 Faltung

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$$