Team notebook

[ITESM Monterrey] O(1)

March 16, 2023

Contents

1 Combinatorial Optimization

1.1 Min Cost Matching

```
#include "../template.cpp"
template<class T>
using mat = vector<vector<T>>;

/**
 * Minimum Cost Matching in bipartite graph. Negate cost for Max.
 *
 * @tparam TCost Type of the edge cost
 * @param cost Cost matrix. cost[i][j] = cost of pairing left i with
     right j
 * @param Lmate Lmate[i] = index of right node that left node i pairs with
 * @param Rmate Lmate[j] = index of right node that left node k pairs with
 * @return TCost Minimum cost in perfect match.
 */
template<class TCost>
TCost MinCostMatching(const mat<TCost> &cost, vi &Lmate, vi &Rmate) {
    typedef vector<TCost> vc;
    int n = int(cost.size());
    // construct dual feasible solution
    vc u(n), v(n);
    for (int i = 0; i < n; i++) {
        u[i] = cost[i][0];
        for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
    }
    for (int j = 0; j < n; j++) {
        v[j] = cost[0][j] - u[0];
        for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
    }

    // construct primal solution satisfying complementary slackness
    Lmate = vi(n, -1);
    Rmate = vi(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (Rmate[j] != -1) continue;
            if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
                Lmate[i] = j;
                Rmate[j] = i;
                mated++;
                break;
            }
        }
    }

    vc dist(n);
    vi dad(n), seen(n);
    // repeat until primal solution is feasible
    while (mated < n) {
        // find an unmatched left node
        int s = 0;
        while (Lmate[s] != -1) s++;
        // initialize Dijkstra
        fill(dad.begin(), dad.end(), -1);
        fill(seen.begin(), seen.end(), 0);
        for (int k = 0; k < n; k++) dist[k] = cost[s][k] - u[s] - v[k];
        int j = 0;
        while (true) {
            // find closest
            j = -1;
            for (int k = 0; k < n; k++) {
                if (seen[k]) continue;
                if (j == -1 || dist[k] < dist[j]) j = k;
            }
            seen[j] = 1;
            // termination condition
            if (Rmate[j] == -1) break;
            // relax neighbors
            const int i = Rmate[j];
            for (int k = 0; k < n; k++) {
                if (seen[k]) continue;
                const ull new_dist = dist[j] + cost[i][k] - u[i] - v[k];
                if (dist[k] > new_dist) {
                    dist[k] = new_dist;
                    dad[k] = j;
                }
            }
        }
```

```
        // update dual variables
        for (int k = 0; k < n; k++) {
            if (k == j || !seen[k]) continue;
            const int i = Rmate[k];
            v[k] += dist[k] - dist[j];
            u[i] -= dist[k] - dist[j];
        }
        u[s] += dist[j];
        // augment along path
        while (dad[j] >= 0) {
            const int d = dad[j];
            Rmate[j] = Rmate[d];
            Lmate[Rmate[j]] = j;
            j = d;
        }
        Rmate[j] = s;
        Lmate[s] = j;
        mated++;
    }

    TCost value = 0;
    for (int i = 0; i < n; i++) value += cost[i][Lmate[i]];
    return value;
}
```

2 Data Structures

2.1 Disjoint Set Union

```
#include "../template.cpp"

class DSU {
   public:
    vector<int> parent, rank;
    DSU(int n) {
        parent = vector<int>(n);
        rank = vector<int>(n, 0);
        for (int i = 0; i < n; i++) parent[i] = i;
    }
    int find(int v) {
        return (v == parent[v]) ? v : parent[v] = find(parent[v]);
    }
    void insert(int v) { parent[v] = v; rank[v] = 0; }
    void merge(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b) {
            if (rank[a] < rank[b]) swap(a, b);
            parent[b] = a;
            if (rank[a] == rank[b]) rank[a]++; // size[a] += size[b];
        }
    }
};
```

2.2 Fenwick Tree

```
#include "../template.cpp"

#define LSOne(S) ((S) & -(S))
typedef vector<int> vi;

template <class T>
class FenwickTree {
    typedef vector<T> vt;
    vi ft;
    FenwickTree(int m) { ft.assign(m + 1, 0); }
    // f is frequency array
    FenwickTree(const vt &f) { build(f); }
    void build(const vt &f) {
        int m = f.size() - 1;
        ft.assign(m + 1, 0);
        for (int i = 1; i <= m; i++) {
            ft[i] += f[i];
            if (i + LSOne(i) <= m)
                ft[i + LSOne(i)] += ft[i];
        }
    }
    int rsq(int j) {
        int sum = 0;
        for (; j; j -= LSOne(j)) sum += ft[j];
        return sum;
    }
    int rsq(int i, int j) { return rsq(j) - rsq(i - 1); }
    void update(int i, int v) {
        for (; i < ft.size(); i += LSOne(i)) ft[i] += v;
```

```
    }
};


template<class T>
class RUPQ {
    FenwickTree<T> ft;
    RUPQ(int m) : ft(FenwickTree<T>(m)) {}
    void range_update(int ui, int uj, int v) {
        ft.update(ui, v);
        ft.update(uj + 1, -v);
    }
    T point_query(int i) { return ft.rsq(i);}
};
```

2.3 Indexed Set

```
#include "../template.cpp"

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

/**
 * @brief Set data structure with rank - know which element is the kth
 * smallest. O(logn) operations.
 *
 * find_by_order(n) => (n - 1)th smallest key
 * order_of_key(k) => rank of key k
 *
 * algorithm - nth_element for O(n) on any vector
 *
 * @tparam TKey Type of set key
 */
template <class TKey>
using indexed_set = tree<TKey, null_type, less<Tkey>, rb_tree_tag,
                         tree_order_statistics_node_update>;
```

2.4 Segment Tree

```
#include "../template.cpp"
#define MAXN 1000000

template <class T>
class SegmentTree {
    typedef vector<T> vt;
    vt tree;
    SegmentTree() : tree(vt(4 * MAXN)) {}
    SegmentTree(vt a) : tree(vt(4 * a.size())) {}
    void build(vt a, int v, int tl, int tr) {
        if (tl == tr) {
            tree[v] = a[tl];
        } else {
            int tm = (tl + tr) / 2;
            build(a, v * 2, tl, tm);
            build(a, v * 2 + 1, tm + 1, tr);
            tree[v] = tree[v * 2] + tree[v * 2 + 1];
        }
    }
    T sum(int v, int tl, int tr, int l, int r) {
        if (l > r) return 0;
        if (l == tl && r == tr) {
            return tree[v];
        }
        int tm = (tl + tr) / 2;
        return sum(v * 2, tl, tm, l, min(r, tm)) +
               sum(v * 2 + 1, tm + 1, tr, max(l, tm + 1), r);
    }
    void update(int v, int tl, int tr, int pos, T new_val) {
        if (tl == tr) {
            tree[v] = new_val;
        } else {
            int tm = (tl + tr) / 2;
            if (pos <= tm)
                update(v * 2, tl, tm, pos, new_val);
            else
                update(v * 2 + 1, tm + 1, tr, pos, new_val);
            tree[v] = tree[v * 2] + tree[v * 2 + 1];
        }
    }
};
```

3 Geometry

3.1 agujetas$_{g}auss$

```python
def agujetas_gauss(points):
    """
    Utiliza agujetas de Gauss para calcular el area del poligono dado en
        la lista de puntos del
    poligono orientados en contra de las manecillas del reloj.
    """
    area = 0
    n = len(points)
    for i in range(n):
        area += points[i][0] * (points[(i + 1) % n][1] - points[(i - 1) %
            n][1])

    return area / 2

p = [(0,0), (1, 0), (1, 1), (0, 1)]
print(agujetas_gauss(p))
```

3.2 colinear

```python
def colinear(x1, y1, x2, y2, x3, y3):
    m = (y2 - y1) / (x2 - x1)
    b = y1 - m*x1
    return y3 == m * x3 + b
```

3.3 line$_{f}rom_{2p}oints$

```python
def line(x1, y1, x2, y2):
    m = (y2 - y1) / (x2 - x1)
    b = y1 - m*x1
    return m, b
```

4 Graph

4.1 Dijkstra

```cpp
#include <queue>

#include "../template.cpp"

const int INF = 2000000000;

template <class TDist>
using w_edge = pair<TDist, int>;
template <class TDist>
using edge_arr = vector<vector<TDist>>;

template <class TDist>
TDist dijkstra(edge_arr<TDist> graph, int n, int src, int target) {
    priority_queue<pii, vector<edge>, greater<edge>> pq;
    vector<TDist> dist(n, INF);
    vi trace(n, -1);
    pq.push(make_pair(0, src));
    dist[s] = 0;
    while (!pq.empty()) {
        w_edge top = pq.top();
        pq.pop();
        int here = top.snd;
        if (here == target) break;
        if (dist[here] != top.fst) continue;

        for (auto& edge : graph) {
            if (dist[here] + edge.fst < dist[edge.snd]) {
                dist[edge.snd] = dist[here] + edge.fst;
                trace[eddge.snd] = here;
                pq.push(make_pair(dist[edge.snd], edge.snd));
            }
        }
    }
}
```

4.2 Fast-Bellman-Ford

```cpp
#include "../template.cpp"

const ll inf = LLONG_MAX;

struct Edge {
    int u, v;
    int s() { return u < v ? u : -u; };
    ll w;
};
struct Node {
    ll dist = inf;
    int prev = -1;
};

// shortest path s to all nodes in O(V * E)
void bellmanFord(vector<Node>& nodes, vector<Edge>& edges, int s) {
    nodes[s].dist = 0;
    sort(all(edges), [](Edge a, Edge b) { return a.s() < b.s(); });

    int lim = sz(nodes) / 2 + 2;
    rep(i, 0, lim) for (Edge& e : edges) {
        Node curr = nodes[e.u], &dest = nodes[e.v];
        if (abs(curr.dist) == inf) continue;
        ll d = curr.dist + e.w;
        if (d < dest.dist) {
            dest.prev = e.u;
            dest.dist = (i < lim - 1 ? d : -inf);
        }
    }
    rep(i, 0, lim) for (Edge& e : edges) {
        if (nodes[e.u].dist == -inf) nodes[e.v].dist = -inf;
    }
}
```

4.3 Kruskal

```cpp
#include "../template.cpp"

template <class T>
struct Edge {
    int u, v;
    T weight;
    bool operator<(Edge const& other) { return weight < other.weight; }
};

template <class T>
vector<Edge<T>> kruskal(vector<Edge<T>>& edges, int n) {
    int cost = 0;
    vector<int> tree_id(n);
    vector<Edge> result;
    for (int i = 0; i < n; i++) tree_id[i] = i;

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

    for (Edge& e : edges) {
        if (tree_id[e.u] != tree_id[e.v]) {
            cost += e.weight;
            result.push_back(e);

            int old_id = tree_id[e.u], new_id = tree_id[e.v];
            for (int i = 0; i < n; i++) {
                if (tree_id[i] == old_id) tree_id[i] = new_id;
            }
        }
    }

    return result;
}
```

4.4 Max-Flow

```cpp
#include <queue>

#include "../template.cpp"

struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
```

```
int s, t;
vector<int> level, ptr;
queue<int> q;

Dinic(int n, int s, int t) : n(n), s(s), t(t) {
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
}

void add_edge(int v, int u, long long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
}

bool bfs() {
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int id : adj[v]) {
            if (edges[id].cap - edges[id].flow < 1) continue;
            if (level[edges[id].u] != -1) continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
        }
    }
    return level[t] != -1;
}

long long dfs(int v, long long pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
        int id = adj[v][cid];
        int u = edges[id].u;
        if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow
             < 1)
            continue;
        long long tr = dfs(u, min(pushed, edges[id].cap -
             edges[id].flow));
        if (tr == 0) continue;
        edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs()) break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

4.5 Min-Cost Max-Flow

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972). This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]). For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:         O(|V|^3) augmentations
//     min cost max flow: O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.
```

```cpp
#include "../template.cpp"

typedef vector<ll> vll;
typedef vector<vll> vvll;
typedef vector<pii> vpii;

const ll INF = numeric_limits<ll>::max() / 4;

struct MinCostMaxFlow {
    int N;
    vvll cap, flow, cost;
    vi found;
    vll dist, pi, width;
    vpii dad;

    MinCostMaxFlow(int N)
        : N(N),
          cap(N, vll(N)),
          flow(N, vll(N)),
          cost(N, vll(N)),
          found(N),
          dist(N),
          pi(N),
          width(N),
          dad(N) {}

    void AddEdge(int from, int to, ll cap, ll cost) {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
    }

    void Relax(int s, int k, ll cap, ll cost, int dir) {
        ll val = dist[s] + pi[s] - pi[k] + cost;
        if (cap && val < dist[k]) {
            dist[k] = val;
            dad[k] = make_pair(s, dir);
            width[k] = min(cap, width[s]);
        }
    }

    ll Dijkstra(int s, int t) {
        fill(found.begin(), found.end(), false);
        fill(dist.begin(), dist.end(), INF);
        fill(width.begin(), width.end(), 0);
        dist[s] = 0;
        width[s] = INF;

        while (s != -1) {
            int best = -1;
            found[s] = true;
            for (int k = 0; k < N; k++) {
                if (found[k]) continue;
                Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
                Relax(s, k, flow[k][s], -cost[k][s], -1);
                if (best == -1 || dist[k] < dist[best]) best = k;
            }
            s = best;
        }

        for (int k = 0; k < N; k++) pi[k] = min(pi[k] + dist[k], INF);
        return width[t];
    }

    pair<ll, ll> GetMaxFlow(int s, int t) {
        ll totflow = 0, totcost = 0;
        while (ll amt = Dijkstra(s, t)) {
            totflow += amt;
            for (int x = t; x != s; x = dad[x].first) {
                if (dad[x].second == 1) {
                    flow[dad[x].first][x] += amt;
                    totcost += amt * cost[dad[x].first][x];
                } else {
                    flow[x][dad[x].first] -= amt;
                    totcost -= amt * cost[x][dad[x].first];
                }
            }
        }
        return make_pair(totflow, totcost);
    }
};
```

5 Inputs

5.1 fastScan

```cpp
void fastscan(int &x){
```

```
    char ch; bool f= 0; int a=0;
    while(!((((ch=getchar())>='0')&&(ch<='9'))||(ch=='-')));
    if(ch!='-')a*=10,a+=ch-'0';else f=1;
    while(((ch=getchar())>='0')&&(ch<='9'))a*=10, a+=ch-'0';
    if(f)a=-a;x=a;
}
```

5.2 inputs

```
lst = [int(i) for i in input().split()]
```

6 Mathematics

6.1 BaseToNumber

```
#include <unordered_map>
#include <string>

using namespace std;

/**
 * Converts a number in an arbitrary base stored in a string into an
     actual integer
 * For example, baseToNumber("C8", 16, "0123456789ABCDEF") returns 200
 * @param input A string containing a number in an arbitrary base
 * @param base The arbitrary base
 * @param alphabet A list of characters representing each digit from zero
     to the base - 1
 * @return An unsigned long integer which is equivalent to the input
     number
 */
unsigned long long int baseToNumber(string input, unsigned long long int
    base, string alphabet) {
       unsigned long long int output = 0;
       unordered_map<char, unsigned long long int> alphabetMap;
       for(unsigned long long int i = 0; i < alphabet.size(); i++) {
              alphabetMap[alphabet[i]] = i;
       }
       for(unsigned long long int i = 0; i < input.size(); i++) {
              output *= base;
              output += alphabetMap[input[i]];
       }
       return output;
}
```

6.2 Bell

```
def bellNumber(n):
    """
    Cuenta las posibles particiones de un conjunto.
    """
    bell = [[0 for i in range(n+1)] for j in range(n+1)]
    bell[0][0] = 1
    for i in range(1, n+1):

        # Explicitly fill for j = 0
        bell[i][0] = bell[i-1][i-1]

        # Fill for remaining values of j
        for j in range(1, i+1):
            bell[i][j] = bell[i-1][j-1] + bell[i][j-1]

    return bell[n][0]
```

6.3 BinomialCoefficient

```
#include <algorithm>
#include <cmath>

using namespace std;

/**
 * Finds the binomial coefficient for a given n and r in O(r).
 * @param n The top value of the binomial coefficient.
 * @param r The bottom value of the binomial coefficient.
 */
long long int BinomialCoefficient(int n, int r) {
    long double coefficent = 1;
    for(int i = 1; i <= r; i++) {
        coefficient *= (long double)(n - r + i) / i;
    }
```

```
    return round(coefficient);
}
```

6.4 Catalan

```python
import math
def catalan(n):
    return math.comb(2 * n, n) / (n + 1)

def Catalan(n):
    """
    - Cantidad de arboles binarios completos posibles de formar con n
        vertices.
    - Cantidad de Strings binarios de tamao 2n tales que tienen n 0's y n
        1's de forma que
    ningun substring inicial tiene mas 0's que 1's.
    """
    if (n == 0 or n == 1):
        return 1

    # Table to store results of subproblems
    catalan = [0]*(n+1)

    # Initialize first two values in table
    catalan[0] = 1
    catalan[1] = 1

    # Fill entries in catalan[]
    # using recursive formula
    for i in range(2, n + 1):
        for j in range(i):
            catalan[i] += catalan[j] * catalan[i-j-1]

    return catalan[n]
```

6.5 ExtendedEuclid

```cpp
#include <utility>

using namespace std;

/**
 * Applies the extended euclid algorithm to find a solution to the
    equation a*x + b*y = gcd(a, b).
 * @param a The first value.
 * @param b The second value.
 * @return A pair of long long ints with the solutions x and y found by
    the algorithm.
 */
pair<long long int, long long int> ExtendedEuclid(long long int a, long
    long int b) {
    if (b == 0) {
        return make_pair(1, 0);
    }
    pair<long long int, long long int> result = ExtendedEuclid(b, a % b);
    return make_pair(result.second, result.first - result.second * (a /
        b));
}
```

6.6 Fibonacci

```python
fibonacciList = {
        0: 0,
        1: 1,
        2: 1
}

def fibonacci(n):
        n = round(n)
        if n not in fibonacciList:
                if n % 2 == 0:
                        k = n // 2
                        fibonacciList[n] = fibonacci(k) * (2 * fibonacci(k
                            + 1) - fibonacci(k))
                else:
                        k = (n - 1) // 2
                        fibonacciList[n] = fibonacci(k + 1) * fibonacci(k +
                            1) + fibonacci(k) * fibonacci(k)
        return fibonacciList[n]
```

6.7 GrayCode

```
/**
 * Converts a number N into gray code
 */
unsigned long long int GrayCode(unsigned long long int n) {
	return n ^ (n / 2);
}

/**
 * Converts a number G from gray code
 */
unsigned long long int InverseGrayCode(unsigned long long int g) {
	unsigned long long int n = 0;
	while(g != 0) {
		n ^= g;
		g /= 2;
	}
	return n;
}
```

6.8 IsPrime

```
/**
 * Checks if a value is prime in O(log^3 N) time using a deterministic
    implementation of the Miller-Rabin algorithm
 * This code only works for values less than 2^64, and requires __int128
 * Written by Brett Hale (StackOverflow 24096332)
 * @param n The value that will be tested
 * @return True if n is prime, and false if n is composite
 */
bool IsPrime (unsigned long long int n) {
	if(n == 1) {
		return 0;
	}
	const unsigned long long int sprp32_base[] = {2, 7, 61, 0};
	const unsigned long long int sprp64_base[] = {2, 325, 9375, 28178,
	    450775, 9780504, 1795265022, 0};
	const unsigned long long int *sprp_base;
	if((n & 1) == 0) {
		return n == 2;
	}
	sprp_base = (n <= 4294967295U) ? sprp32_base : sprp64_base;
	for(; *sprp_base != 0; sprp_base++) {
		unsigned long long int a = *sprp_base, m = n - 1, r, y, s
		    = 1;
		while ((m & (1UL << s)) == 0) {
			s++;
		}
		r = m >> s;
		if ((a %= n) == 0) {
			continue;
		}
		unsigned __int128 u = 1, w = a;
		while (r != 0) {
			if ((r & 1) != 0) {
				u = (u * w) % n;
			}
			if ((r >>= 1) != 0) {
				w = (w * w) % n;
			}
		}
		if ((y = (unsigned long long int)u) == 1) {
			continue;
		}
		for (unsigned long long int j = 1; j < s && y != m; j++) {
			u = y;
			u = (u * u) % n;
			if ((y = (unsigned long long int)u) <= 1) {
				return false;
			}
		}
		if(y != m) {
			return false;
		}
	}
	return true;
}
```

6.9 ModularExponentiation

```
/**
 * Calculates the operation base ^ exponent % modulus for long integers
 */
long long int modExp(long long int base, long long int exponent, long
    long int modulus) {
	long long int power = 1;
```

```
	base %= modulus;
	while(exponent > 0) {
		if(exponent % 2 == 1) {
			power = (power * base) % modulus;
		}
		exponent /= 2;
		base = (base * base) % modulus;
	}
	return power;
}
```

6.10 ModularFibonacci

```
#include <unordered_map>

unordered_map<long long int, long long int> fibonacciMap({{0, 0}, {1, 1},
    {2, 1}});

/**
 * Returns F(n) % mod in O(log(n))
 */
long long int fibonacci(long long int n, long long int mod) {
	if(fibonacciMap.count(n) == 0) {
		long long int k = n / 2;
		if(n % 2 == 0) {
			fibonacciMap[n] = (fibonacci(k, mod) * (2 *
			    fibonacci(k + 1, mod) - fibonacci(k, mod) +
			    mod)) % mod;
		} else {
			fibonacciMap[n] = (fibonacci(k, mod) * fibonacci(k,
			    mod) + fibonacci(k + 1, mod) * fibonacci(k + 1,
			    mod)) % mod;
		}
	}
	return fibonacciMap[n];
}
```

6.11 ModularMultiplication

```
/**
 * Calculates the operation a * b % modulus for unsigned long integers
     without overflow
 */
unsigned long long int modMult(unsigned long long int a, unsigned long
    long int b, unsigned long long int modulus) {
	unsigned long long int product = 0, tempB;
	if(b >= modulus) {
		if(modulus > 18446744073709551615ULL / 2ULL) {
			b -= modulus;
		} else {
			b %= modulus;
		}
	}
	while(a != 0) {
		if(a % 2 == 1) {
			if(b >= modulus - product) {
				product -= modulus;
			}
			product += b;
		}
		a /= 2;
		tempB = b;
		if(b >= modulus - b) {
			tempB -= modulus;
		}
		b += tempB;
	}
	return product;
}
```

6.12 NumDivisors

```
#include <vector>
#include <algorithm>

long long int NumDivisors(vector<long long int> primeFactors) {
	sort(primeFactors.begin(), primeFactors.end());
	long long int divisors = 1;
	long long int prevValue = 0;
	long long int count = 0;
	for(int i = 0; i < primeFactors.size(); i++) {
		if(primeFactors[i] == prevValue) {
			count++;
```

```
        } else {
            prevValue = primeFactors[i];
            divisors *= count + 1;
            count = 1;
        }
    }
    divisors *= count + 1;
    return divisors;
}
```

6.13 NumberToBase

```
#include <string>

using namespace std;

/**
 * Converts an integer into a number in an arbitrary base stored in a
    string
 * For example, numberToBase(200, 16, "0123456789ABCDEF") returns "C8"
 * @param input An unsigned long integer to represent in the arbitrary
    base
 * @param base The arbitrary base
 * @param alphabet A list of characters representing each digit from zero
    to the base - 1
 * @return A string containing the representation of the number in the
    given base
 */
string numberToBase(unsigned long long int input, unsigned long long int
    base, string alphabet) {
    string output = "";
    while(input > 0) {
        output = alphabet[input % base] + output;
        input /= base;
    }
    return output;
}
```

6.14 PascalTriangle

```
#include <vector>

using namespace std;

/**
 * Constructs a pascal triangle of a given size.
 * @param size The maximum size of the triangle.
 * @return A 2D vector of long long ints with each of the coefficients of
    the triangle.
 */
vector<vector<long long int>> PascalTriangle(int size) {
    vector<vector<long long int>> triangle = {{1}};
    for(int y = 1; y < size; y++) {
        vector<long long int> row;
        row.push_back(1);
        for(int x = 1; x < y; x++) {
            row.push_back(triangle[y - 1][x - 1] + triangle[y -
                1][x]);
        }
        row.push_back(1);
        triangle.push_back(row);
    }
    return triangle;
}
```

6.15 PrimeFactorization

```
#include <vector>

long long int Brent(long long int N) {
    if(N % 2 == 0) {
        return 2;
    }
    long long int y = (rand() % (N - 1)) + 1;
    long long int c = (rand() % (N - 1)) + 1;
    long long int m = (rand() % (N - 1)) + 1;
    long long int x, k, ys, g = 1, r = 1, q = 1;
    while(g == 1) {
        x = y;
        for(int i = 0; i < r; i++) {
            y = (modMult(y, y, N) + c) % N;
        }
        k = 0;
        while(k < r and g == 1) {
```

```
                ys = y;
                for(int i = 0; i < min(m, r - k); i++) {
                    y = (modMult(y, y, N) + c) % N;
                    q = modMult(q, abs(x - y), N);
                }
                g = gcd(q, N);
                k = k + m;
            }
            r = r * 2;
        }
        if(g == N) {
            while(true) {
                ys = (modMult(ys, ys, N) + c) % N;
                g = gcd(abs(x - ys), N);
                if(g > 1) {
                    break;
                }
            }
        }
        return g;
}

/**
 * Finds all prime factors of the given value using Brent's method
 * A fast IsPrime algorithm like Miller-Rabin's deterministic algorithm
    should be used
 * Disclamer: factors are not always in order
 */
vector<long long int> PrimeFactorization(long long int value) {
        vector<long long int> factors;
        if(value < 2) {
            return factors;
        }
        vector<long long int> factorsToCheck;
        factorsToCheck.push_back(value);
        while(factorsToCheck.size() > 0) {
            long long int currentFactor = factorsToCheck.back();
            factorsToCheck.pop_back();
            if(currentFactor == 1) {
                continue;
            }
            if(IsPrime(currentFactor)) {
                factors.push_back(currentFactor);
            } else {
                long long int newFactor = Brent(currentFactor);
                factorsToCheck.push_back(newFactor);
                factorsToCheck.push_back(currentFactor / newFactor);
            }
        }
        return factors;
}
```

6.16 SieveOfAtkin

```
/**
 * Calculates and stores all primes less than or equal to the limit in
    O(N) time
 * @param primes An array where all the primes less than or equal to the
    limit shall be stored
 * @param limit The largest value that will be checked by the function
 * @return The amount of primes that are stored in the array of primes
 */
int SieveOfAtkin(long long int* primes, long long int limit) {
    int index = 0;
    limit++;
    if(limit < 3) {
        return index;
    }
    primes[index] = 2;
    index++;
    if(limit < 4) {
        return index;
    }
    primes[index] = 3;
    index++;
    bool* sieve = new bool[limit];
    for(long long int i = 0; i < limit; i++) {
        sieve[i] = false;
    }
    for(long long int i = 1; i * i < limit; i++) {
        for(long long int j = 1; j * j < limit; j++) {
            long long int a = 4 * i * i + j * j;
            long long int b = 3 * i * i + j * j;
            long long int c = 3 * i * i - j * j;
            if(a <= limit && (a % 12 == 1 || a % 12 == 5)) {
                sieve[a] = !sieve[a];
            }
            if(b <= limit && b % 12 == 7) {
```

```
                sieve[b] = !sieve[b];
            }
            if(c <= limit && i > j && c % 12 == 11) {
                sieve[c] = !sieve[c];
            }
        }
    }
    for(long long int i = 5; i * i < limit; i++) {
        if(sieve[i]) {
            for(long long int j = i * i; j < limit; j += i * i) {
                sieve[j] = false;
            }
        }
    }
    for(long long int i = 5; i < limit; i++) {
        if(sieve[i]) {
            primes[index] = i;
            index++;
        }
    }
    return index;
}
```

6.17 Sterling1

```
def sterling_first(n, k):
    """
    Calculates the Sterling number of the first kind, which represents
        the number of ways
    to partition a set of n distinct objects into k disjoint cycles.
    """
    if n == 0 and k == 0:
        return 1
    elif n == 0 or k == 0:
        return 0
    else:
        # Create a table to store the results of subproblems
        table = [[0 for x in range(k+1)] for y in range(n+1)]

        # Fill in the base cases
        for i in range(n+1):
            table[i][0] = 0
        for j in range(1, k+1):
            table[0][j] = 0
        table[0][0] = 1

        # Fill in the remaining entries using the recurrence relation
        for i in range(1, n+1):
            for j in range(1, k+1):
                table[i][j] = (i-1)*table[i-1][j] + table[i-1][j-1]

        return table[n][k]
```

6.18 Sterling2

```
def sterling_second(n, k):
    """
    Calculates the Sterling number of the second kind, which represents
        the number of ways
    to partition a set of n distinct objects into k non-empty
        indistinguishable subsets.
    Runs in O(nk)
    """
    if n == 0 and k == 0:
        return 1
    elif n == 0 or k == 0:
        return 0
    else:
        # Create a table to store the results of subproblems
        table = [[0 for x in range(k+1)] for y in range(n+1)]

        # Fill in the base cases
        for i in range(n+1):
            table[i][0] = 0
        for j in range(k+1):
            table[0][j] = 0
        table[0][0] = 1

        # Fill in the remaining entries using the recurrence relation
        for i in range(1, n+1):
            for j in range(1, k+1):
                table[i][j] = j*table[i-1][j] + table[i-1][j-1]

        return table[n][k]
```

6.19 factor

```
import math
def factor(n):
    pos = [2, 3]
    pos.extend([6 * k + i for k in range(1, math.ceil(n/12) + 1) for i in
        [-1, 1]])
    pos.append(n)
    primes = []
    exponents = []
    iter = 0
    while(n > 1):
        if(n % pos[iter] == 0):
            primes.append(pos[iter])
            ex = 0
            while(n % pos[iter] == 0):
                n = n / pos[iter]
                ex += 1
            exponents.append(ex)
        iter += 1
    return [primes, exponents]

fact=factor(int(input()))
print(fact[0])
print(fact[1])
```

6.20 gcd

```
/**
 * Calculates the greatest common divisor between two numbers
 */
long long gcd(long long a, long long b){
   return b == 0 ? a : gcd(b, a % b);
}
```

6.21 gcf

```
def gcf(m,n):
    if m<n:
        (m,n) = (n,m)
    while m%n !=0:
        (m,n) = (n, m%n)
    return n

print(gcf(13,95))
```

6.22 isFibonacci

```
#include <math.h>

bool isSquare(unsigned long long x){
    unsigned long long s = sqrt(x);
    return (s*s == x);
}

bool isFib(unsigned long long n){
    return isSquare(5*n*n+4)||isSquare(5*n*n-4);
}
```

6.23 isPrime

```
def isPrime(n):
    if(n == 1 or ((n % 2 == 0 or n % 3 == 0) and n != 2 and n != 3)):
        return False
    d = 5
    while(d**2 <= n):
        if(n % d == 0):
            return False
        d = d + 2
        if(n % d == 0):
            return False
        d = d + 4
    return True

print(isPrime(2017))
```

6.24 lcm

```
/**
 * Calculates the least common multiple between two numbers
```

```
 */
long long gcd(long long a, long long b){
   return b == 0 ? a : gcd(b, a % b);
}

long long lcm(long long a, long long b){
   return (a * b) / gcd(a, b);
}
```

6.25 mod

```
/**
 * Calculates the positive and negative modulus of two numbers
 */
long long int mod(long long int a, long long int b) {
       return (a % b + b) % b;
}
```

6.26 multiplicativeInverse

```
#define mod 1000000007

long long inve(long long a){
   long long b = mod-2, ans = 1;
   while (b) {
       if (b&1) {
           ans = (ans * a) % mod;
       }
       a = (a * a) % mod;
       b >>= 1;
   }
   return ans;
}
```

6.27 multiplicativeInverse

```
from operator import mul

def multiplicativeInverse(a, m):
    m0 = m
    y = 0
    x = 1
    if (m == 1):
        return 0
    while (a > 1):
        # q is quotient
        q = a // m
        t = m
        # m is remainder now, process
        # same as Euclid's algo
        m = a % m
        a = t
        t = y
        # Update x and y
        y = x - q * y
        x = t
    # Make x positive
    if (x < 0):
        x = x + m0
    return x

print(multiplicativeInverse(5,7))
```

6.28 numberDivisors

```
import math
def numberDivisors(n):
    counter=0
    sq=math.sqrt(n)
    # print(sq)
    for i in range(1,int(sq)+1):
        if(n%i==0):
            counter+=2
    if(sq-math.floor(sq)==0):
        counter-=1
    return counter

print(numberDivisors(10))
```

6.29 triangularNumbers

```cpp
#include <math.h>
using namespace std;

/**
 * check if a number is triangular in constant time
 */
bool isTriangular(unsigned long long n){
    return ((float)sqrt(8*n+1) == floor(floor((float)sqrt(8*n+1))));
}
```

7 Strings

7.1 Knuth-Morris-Pratt

```cpp
#include "../template.cpp"

void buildPi(string& p, vi& pi) {
    pi = vi(p.length());
    int k = -2;
    for (int i = 0; i < p.length(); i++) {
        while (k >= -1 && p[k + 1] != p[i]) k = (k == -1) ? -2 : pi[k];
        pi[i] = ++k;
    }
}

/** Finds all occurrences of the pattern string p within the
 * text string t.
 * Running time is O(n + m), where n and m are the lengths
 * of p and t, respectively.
 */
int KMP(string& t, string& p) {
    vi pi;
    buildPi(p, pi);
    int k = -1;
    for (int i = 0; i < t.length(); i++) {
        while (k >= -1 && p[k + 1] != t[i]) k = (k == -1) ? -2 : pi[k];
        k++;
        if (k == p.length() - 1) {
            // p matches t[i-m+1, ..., i]
            cout << "matched at index " << i - k << ": ";
            cout << t.substr(i - k, p.length()) << endl;
            k = (k == -1) ? -2 : pi[k];
        }
    }
    return 0;
}
```

7.2 KnuthMorrisPratt

```cpp
#include <string>
#include <vector>

using namespace std;

/**
 * Finds all occurrences of a pattern in a string of text
 * @param text The text to search
 * @param pattern The pattern to look for in the text
 * @return A vector containing all the indices of each occurrence
 */
vector<int> knuthMorrisPratt(string text, string pattern) {
        vector<int> output;
        int processTable[pattern.size()];
        for(int i = 0; i < pattern.size(); i++) {
                processTable[i] = 0;
        }
        int processLength = 0;
        int processIndex = 1;
        processTable[0] = 0;
        while(processIndex < pattern.size()) {
                if(pattern[processIndex] == pattern[processLength]) {
                        processTable[processIndex] = processLength + 1;
                        processLength++;
                        processIndex++;
                } else {
                        if(processLength == 0) {
                                processTable[processIndex] = 0;
                                processIndex++;
                        } else {
                                processLength = processTable[processLength -
                                    1];
                        }
```

```
            }
        }
        int textIndex = 0;
        int patternIndex = 0;
        while(textIndex < text.size()) {
            if(text[textIndex] == pattern[patternIndex]) {
                textIndex++;
                patternIndex++;
            } else {
                if(patternIndex == 0) {
                    textIndex++;
                } else {
                    patternIndex = processTable[patternIndex -
                        1];
                }
            }
            if(patternIndex == pattern.size()) {
                output.push_back(textIndex - patternIndex);
                patternIndex = processTable[patternIndex - 1];
            }
        }
        return output;
}
```

7.3 LargestPalindromes

```
#include <unordered_set>
#include <utility>
#include <string>

using namespace std;

/**
 * Returns all the largest palindromes in the string
 * @param input The input string
 * @return A pair which contains an int (the length of the largest
     palindrome) and an unordered set of these palindromes
 */
pair<int, unordered_set<string>> getLargestPalindromes(string input) {
        if(input.size() < 2) {
            unordered_set<string> palindrome;
            palindrome.insert(input);
            return make_pair(input.size(), palindrome);
        }
        if(input[0] == input[input.size() - 1]) {
            pair<int, unordered_set<string>> palindromeInfo =
                getLargestPalindromes(input.substr(1, input.size() -
                2));
            palindromeInfo.first += 2;
            unordered_set<string> newSet;
            for(const string &str: palindromeInfo.second) {
                newSet.insert(input[0] + str + input[input.size() -
                    1]);
            }
            palindromeInfo.second = newSet;
            return palindromeInfo;
        }
        pair<int, unordered_set<string>> palindromeInfo;
        pair<int, unordered_set<string>> a =
            getLargestPalindromes(input.substr(0, input.size() - 1));
        pair<int, unordered_set<string>> b =
            getLargestPalindromes(input.substr(1, input.size() - 1));
        palindromeInfo.first = max(a.first, b.first);
        if(a.first == palindromeInfo.first) {
            for(const string &str: a.second) {
                palindromeInfo.second.insert(str);
            }
        }
        if(b.first == palindromeInfo.first) {
            for(const string &str: b.second) {
                palindromeInfo.second.insert(str);
            }
        }
        return palindromeInfo;
}
```

7.4 NeedlemanWunsch

```
#include <utility>
#include <vector>
#include <string>

using namespace std;

/**
```

```
 * Algorithm that aligns two strings A and B by inserting spaces and
     deleting characters
 * Longest Common Subsequence can be found with the following result:
 * needlemanWunsch(a, b, true, '-', 1, -INF, 0)
 * @param a The first string
 * @param b The second string
 * @param firstResult Makes the function stop after finding the first
     alignment
 * @param empty The character to place when a space is inserted
 * @param match The score that is used when a match is found
 * @param mismatch The score that is used when a mismatch is found
 * @param indel The score that is used when a character is deleted or a
     space is inserted
 * @return A vector of pairs of best alignments
 */
vector<pair<string, string>> needlemanWunsch(string a, string b, bool
    firstResult = false, char empty = '-', int match = 1, int mismatch =
    -1, int indel = -1) {
       vector<pair<string, string>> output;
       int scores[a.size() + 1][b.size() + 1];
       int paths[a.size() + 1][b.size() + 1];
       for(int y = 0; y <= a.size(); y++) {
              scores[y][0] = y * indel;
              paths[y][0] = 1;
       }
       for(int x = 0; x <= b.size(); x++) {
              scores[0][x] = x * indel;
              paths[0][x] = 2;
       }
       paths[0][0] = 0;
       for(int y = 1; y <= a.size(); y++) {
              for(int x = 1; x <= b.size(); x++) {
                     int scoreTop = scores[y - 1][x] + indel;
                     int scoreLeft = scores[y][x - 1] + indel;
                     int scoreDiagonal = scores[y - 1][x - 1];
                     if(a[y - 1] == b[x - 1]) {
                            scoreDiagonal += match;
                     } else {
                            scoreDiagonal += mismatch;
                     }
                     scores[y][x] = max(max(scoreTop, scoreLeft),
                         scoreDiagonal);
                     paths[y][x] = 0;
                     if(scoreTop == scores[y][x]) {
                            paths[y][x] += 1;
                     }
                     if(scoreLeft == scores[y][x]) {
                            paths[y][x] += 2;
                     }
                     if(scoreDiagonal == scores[y][x]) {
                            paths[y][x] += 4;
                     }
              }
       }
       pair<int, int> corner = make_pair(a.size(), b.size());
       vector<pair<int, int>> path;
       vector<pair<int, int>> movements;
       vector<int> distances;
       movements.push_back(corner);
       distances.push_back(0);
       while(movements.size() > 0) {
              pair<int, int> current = movements.back();
              int distance = distances.back();
              movements.pop_back();
              distances.pop_back();
              int y = current.first;
              int x = current.second;
              while(path.size() > distance) {
                     path.pop_back();
              }
              path.push_back(current);
              int pathCode = paths[y][x];
              if(pathCode >= 4) {
                     pathCode -= 4;
                     movements.push_back(make_pair(y - 1, x - 1));
                     distances.push_back(distance + 1);
              }
              if(pathCode >= 2) {
                     pathCode -= 2;
                     movements.push_back(make_pair(y, x - 1));
                     distances.push_back(distance + 1);
              }
              if(pathCode >= 1) {
                     movements.push_back(make_pair(y - 1, x));
                     distances.push_back(distance + 1);
              }
              if(x == 0 && y == 0) {
                     string newStringA = "";
                     string newStringB = "";
                     for(int i = path.size() - 2; i >= 0; i--) {
```

```
                        int dy = path[i].first - path[i + 1].first;
                        int dx = path[i].second - path[i + 1].second;
                        if(dy == 1 && dx == 1) {
                                newStringA += a[path[i].first - 1];
                                newStringB += b[path[i].second - 1];
                        } else if (dy == 1) {
                                newStringA += a[path[i].first - 1];
                                newStringB += empty;
                        } else {
                                newStringA += empty;
                                newStringB += b[path[i].second - 1];
                        }
                }
                output.push_back(make_pair(newStringA, newStringB));
                if(firstResult) {
                        return output;
                }
            }
        }
        return output;
}
```

7.5 Permute

```
#include <vector>
#include <string>

using namespace std;

/**
 * Returns all permutations of str, including duplicates
 * By changing the vector into a set, the duplicates can be ommited
 * @param str The string to permute
 * @return A vector of permutations
 */
vector<string> Permute(string str) {
        vector<string> outputs;
        if(str.size() == 0) {
                outputs.push_back("");
                return outputs;
        }
        for(int i = 0; i < str.size(); i++) {
                string a = str.substr(0, i);
                string b = str.substr(i + 1, str.size() - i - 1);
                vector<string> subpermutations = Permute(a + b);
                for(int j = 0; j < subpermutations.size(); j++) {
                        outputs.push_back(str[i] + subpermutations[j]);
                }
        }
        return outputs;
}
```

7.6 Tokenize

```
#include <string>
#include <vector>

using namespace std;

/**
 * Splits the input string into a vector of tokens
 * @param input The string to split
 * @param delimiters A string containing all the characters to consider
     as delimiters
 * @return A vector of tokens
 */
vector<string> tokenize(string input, string delimiters) {
        vector<string> tokens;
        string curr = "";
        for(int i = 0; i < input.size(); i++) {
                bool isDelimiter = false;
                for(int j = 0; j < delimiters.size(); j++) {
                        if(input[i] == delimiters[j]) {
                                isDelimiter = true;
                                break;
                        }
                }
                if(isDelimiter) {
                        if(curr.size() > 0) {
                                tokens.push_back(curr);
                                curr = "";
                        }
                } else {
                        curr += input[i];
                }
        }
```

```
        if(curr.size() > 0) {
                tokens.push_back(curr);
                curr = "";
        }
        return tokens;
}
```

8 template

```
// #include <bits/extc++.h> // pbds

#include <iostream>
#include <vector>
#define fst first
#define snd snd
#define sz(x) x.size()
#define rep(i, a, b) for (int i = a; i < (b); i++)
#define sync ios_base::sync_with_stdio(false); cin.tie(NULL);
    cout.tie(NULL)
#define all(x) x.begin(), x.end()

using namespace std;

typedef unsigned long long ull;
typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int, int> pii;
```