Team37

Radboud University Nijmegen

1 Basic

Power (Fast Exponentiation)

```
int pwr(int a, int b){
        int result = 1;
    while (b){
        if (b % 2) result *= a;
        b /= 2;
        a *= a;
    }

    return result;
}
```

Prime Sieve

```
vector<int> prime_sieve(int n){

        if (n < 2) return vector<int>();

        vector<int> primes;
        vector<bool> l (n+1,true);
        primes.push_back(2);

    int i = 3;
        for(; i <= sqrt(n); i += 2){
                while(!l[i]) i += 2;

                primes.push_back(i);
                for(int j=i*i; j < n; j+=i) l[j] = false;
        }

        for(; i < n; i+=2){
        if(l[i]) primes.push_back(i);
        }

        return primes;
}
```

Extended Euclidean

```
//Input two numbers a and b
//Return gcd(a,b)
int64_t gcd(int64_t a, int64_t b){
    if(a < b) swap(a,b);
    while(b != 0){
        int64_t r = a % b;
        a = b;
        b = r;
    }
    return a;
}

//Input two numbers a and b;
//Return triple (x,y,c) satisfying:
//x * a + y * b = c, with c = gcd(a,b)
pair<pair<int64_t,int64_t>, int64_t> egcd(int64_t a, int64_t b){
```

```
    int64_t p_prev = 0, p_cur = 1;
    int64_t q_prev = 1, q_cur = 0;
    int m = 0;
    if(a < b) {
        m++;
        swap(a,b);
        swap(q_prev,p_prev);
        swap(q_cur,p_cur);
    }
    while(b != 0){
        m++;
        int64_t r = a % b;
        int64_t k = a / b;
        int64_t s_temp = k * q_cur + q_prev;
        q_prev = q_cur, q_cur = s_temp;
        int64_t t_temp = k * p_cur + p_prev;
        p_prev = p_cur, p_cur = t_temp;
        a = b;
        b = r;
    }
    if(m % 2 == 0) m = 1;
    else m = -1;
    return make_pair(make_pair(m*q_prev,-m*p_prev),a);
}

//solved: https://open.kattis.com/problems/modulararithmetic
```

2 Graphs

Dijsktra:

```
#define INF (1LL<<60)
#define endl '\n'
#define mp make_pair

pair<vector<int64_t>, vector<int64_t> > dijkstra(vector<vector<pair<
    int64_t,int64_t> > >& graph, int64_t u){

    vector<int64_t> dist (graph.size(), INF), prev (graph.size(), -1);
    dist[u] = 0;
    priority_queue<pair<int64_t,int64_t> > Q;
    Q.push(mp(-dist[u],u));
    vector<bool> seen (graph.size(), false);

    while(!Q.empty()){
        pair<int64_t,int64_t> p = Q.top();
        int64_t w = p.second;
        Q.pop();
        if(!seen[w]){
            seen[w] = true;
            for(auto to : graph[w]){
                if(dist[to.first] > dist[w] + to.second){
                    dist[to.first] = dist[w] + to.second;
                    prev[to.first] = w;
                    Q.push(mp(-dist[to.first],to.first));
                }
            }
        }
```

```
    }
    return  make_pair(dist,prev);
}

//Solved : https://open.kattis.com/problems/shortestpath1
//http://codeforces.com/problemset/problem/20/C
//http://www.spoj.com/problems/SHPATH/
```

Floyd Warshall

```
#define INF (1LL << 60)
#define endl '\n'
#define mp make_pair

void floyd_warshall (vector<vector<int64_t> >& dist){
    for(int64_t k = 0; k < dist.size(); k++){
        for(int64_t i = 0; i < dist.size(); i++){
            for(int64_t j = 0; j < dist.size(); j++){
                if(dist[i][k] != INF && dist[k][j] != INF){
                    if(dist[i][j] > dist[i][k] + dist[k][j]){
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
    }
}

/** The distance options
if(dist[u][v] == INF) cout << "Impossible" << endl;
else if(dist[u][u] != 0 || dist[v][v] != 0) cout << "-Infinity" << endl;
else cout << dist[u][v] << endl;
**/

//Solved : https://open.kattis.com/problems/allpairspath
```

Disjoint Union

```
// Union-Find Disjoint Sets Library written in OOP manner, using both
   path compression and union by rank heuristics
class UnionFind {                                               // OOP
   style
    private:
        vector<int> p, rank, setSize;                           // remember:
            vi is vector<int>
        int numSets;
    public:
        UnionFind(int N) {
            setSize.assign(N, 1);
            numSets = N;
            rank.assign(N, 0);
            p.assign(N, 0);
            for (int i = 0; i < N; i++) p[i] = i;
        }
        int findSet(int i) {
            return (p[i] == i) ? i : (p[i] = findSet(p[i]));
        }
```

```
        bool isSameSet(int i, int j) {
            return findSet(i) == findSet(j);
        }
        void unionSet(int i, int j) {
            if (!isSameSet(i, j)) {
                numSets--;
                int x = findSet(i), y = findSet(j);
                // rank is used to keep the tree short
                if (rank[x] > rank[y]) {
                    p[y] = x; setSize[x] += setSize[y];
                }
                else{
                    p[x] = y; setSize[y] += setSize[x];
                    if (rank[x] == rank[y]) rank[y]++;
                }
            }
        }
        int numDisjointSets() {
            return numSets;
        }
        int sizeOfSet(int i) {
            return setSize[findSet(i)];
        }
};

//Solved : https://open.kattis.com/problems/minspantree
```

2.1 MST

Kruskall

```
struct Edge{
    int64_t first, second, weight;
};

bool edge_compare(Edge l, Edge r){
    return (l.weight < r.weight);
}

vector<Edge> kruskal(vector<Edge> e, int64_t n){
    UnionFind UF((int)n);
    vector<Edge> A;
    sort(e.begin(),e.end(),edge_compare);
    for(int i = 0; i < e.size(); i++){
        Edge edge = e[i];
        int u = edge.first, v = edge.second;
        if(!UF.isSameSet(u,v)){
            A.push_back(edge);
            UF.unionSet(u,v);
        }
    }
    return A;
}

//Solved : https://open.kattis.com/problems/minspantree
```

Prim

```
struct Primdata {
    vector<int64_t> dist;
    vector<int64_t> prev;
    int64_t length;
};

Primdata prim(vector<vector<pair<int64_t,int64_t> > >& graph, int64_t
   start){
    vector<int64_t> dist (graph.size(),INF);
    vector<int64_t> prev (graph.size());
    int64_t length = 0;
    dist[start] = 0;

    priority_queue<pair<int64_t,int64_t> > Q;
    Q.push({-dist[start],start});

    vector<bool> seen (graph.size(), false);

    while(!Q.empty()){
        pair<int64_t,int64_t> p = Q.top();
        int64_t w = p.second;
        Q.pop();
        if(seen[w]) continue;
        seen[w] = true;
        length += dist[w];
        for(auto to : graph[w]){
            if(!seen[to.first] && dist[to.first] > to.second){
                dist[to.first] = to.second;
                prev[to.first] = w;
                Q.push({-dist[to.first],to.first});
            }
        }
    }
    return {dist,prev,length};
}

//Solved : https://open.kattis.com/problems/minspantree
```