Mu_Ji_Sung_-_-_- Team Note

Cheolmin Choi, Changho Lee, Jeongsoo Han

October 8, 2021

1 FFT

```
//have to include these headers
#include <vector>
#include <complex>
#include <cmath>

using namespace std;
typedef complex<double> cpx;

//Cooley-Tukey FFT
void FFT(vector<cpx>& A, cpx w) {
    //base case
    int n = (int)A.size();
    if(n == 1) return;

    //split to even and odd
    vector<cpx> even(n / 2), odd(n / 2);
    for(int i = 0; i < n; ++i) {
        if(i & 1) odd[i / 2] = A[i];
        else even[i / 2] = A[i];
    }

    //Divide
    FFT(even, w * w);
    FFT(odd, w * w);

    cpx w_e(1, 0);

    //conquer
    for(int i = 0; i < n / 2; ++i) {
        A[i] = even[i] + w_e * odd[i];
        A[i + n / 2] = even[i] - w_e * odd[i];
        w_e *= w;
    }

    //Time complexity = n log n
}

//get discrete convolution of A and B
void product(vector<cpx>& A, vector<cpx>& B) {
    //get n, which satisfies n > 2 ^ ceil(log_2(size)) (when n = 2^k)
    int n = (A.size() <= B.size()) ?
    ceil(log2((double)B.size())) : ceil(log2((double)A.size()));
    n = pow(2, n + 1);

    A.resize(n);
    B.resize(n);
    vector<cpx> C(n);

    //n th root of unity (euler formula)
    cpx w(cos(2 * acos(-1) / n), sin(2 * acos(-1) / n));
    FFT(A, w);
    FFT(B, w);

    //multiply DFT
    for(int i = 0; i < n; ++i) C[i] = A[i] * B[i];

    //Inverse FFT to get Coefficient
    FFT(C, cpx(1, 0) / w);
    for(int i = 0; i < n; ++i) {
        C[i] /= cpx(n, 0);
        C[i] = cpx(round(C[i].real()), round(C[i].imag()));
    }
}

/*
void FFT(vector<cpx>& A, bool invert) {
    int n = (int)A.size();

    for(int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;

        while(j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;

        if(i < j) swap(A[i], A[j]);
    }

    for(int length = 2; length <= n; length <<= 1) {
        double ang = 2 * PI / length * (invert ? -1 : 1);
        cpx w(cos(ang), sin(ang));

        for(int i = 0; i < n; i += length) {
            cpx w_i(1, 0);
            for(int j = 0; j < length / 2; ++j) {
                cpx u = A[i + j], v = A[i + j + length / 2] * w_i;
                A[i + j] = u + v, A[i + j + length / 2] = u - v;
                w_i *= w;
            }
        }
    }

    if(invert) {
        for(int i = 0; i < n; ++i) {
            A[i] /= cpx(n, 0);
```

```
            A[i] = cpx(round(A[i].real()), round(A[i].imag()));
        }
    }
} // referenced from https://blog.myungwoo.kr/54
*/ //faster version of FFT
```

2 Geometry

2.1 Convex Hull

```
//have to include this header
#include <vector>
#include <algorithm>

using namespace std;
typedef long long ll;

typedef struct _Point {
    int x;
    int y;
} Point;

//Standard Point to Sort
Point S;

Point getVector(const Point& A, const Point& B) {
    Point v = {B.x - A.x, B.y - A.y};
    return v;
}

//ccw test
int ccw(const Point& v, const Point& u) {
    ll val = (ll)v.x * u.y - (ll)v.y * u.x;
    if(val > 0) return 1;
    else if(val < 0) return -1;
    else return 0;
}

int ccw(const Point& A, const Point& B, const Point& C) {
    Point v = getVector(A, B);
    Point u = getVector(B, C);
    return ccw(v, u);
}

//to sort by ccw
bool comp(const Point& A, const Point& B) {
    Point v = getVector(S, A);
    Point u = getVector(S, B);

    if(ccw(v, u) > 0) return true;
    else if(ccw(v, u) < 0) return false;

    return (v.x == u.x) ? (v.y < u.y) : (v.x < u.x);
}

bool operator<(const Point& A, const Point& B) {
    return (A.x == B.x) ? (A.y < B.y) : (A.x < B.x);
}

//Graham's Scan Method
vector<Point> getConvexHull(vector<Point>& A) {
    S = *min_element(A.begin(), A.end());
    sort(A.begin(), A.end(), comp);
    int n = (int)A.size();

    vector<Point> convexHull;

    //get Convex Hull
    for(int i = 0; i < n; ++i) {
        while((int)convexHull.size() > 1
        && ccw(convexHull[(int)convexHull.size() - 2], convexHull.back(), A[i]) <= 0) {
            convexHull.pop_back();
        }
        convexHull.push_back(A[i]);
    }

    return convexHull;
}
```

2.2 Line Cross Test

```
//have to include this header
#include <vector>

using namespace std;
typedef long long ll;

typedef struct _Point {
    int x;
    int y;
} Point;

//should define these functions(implemented in convex hull source code)
Point getVector(const Point& A, const Point& B);
int ccw(const Point& v, const Point& u);
int ccw(const Point& A, const Point& B, const Point& C);

bool operator<=(const Point A, const Point B) {
    if(A.x < B.x) return true;
    else if(A.x == B.x && A.y <= B.y) return true;
    else return false;
}

//cross test
bool isCross(const Point& A, const Point& B, const Point& C, const Point& D) {
    if(ccw(A, B, C) * ccw(A, B, D) == 0 && ccw(C, D, A) * ccw(C, D, B) == 0) {
        Point _A(A), _B(B), _C(C), _D(D);
        if(_B <= _A) swap(_A, _B);
        if(_D <= _C) swap(_C, _D);

        if(_A <= _D && _C <= _B) return true;
        else return false;
    }
    else if(ccw(A, B, C) * ccw(A, B, D) <= 0 && ccw(C, D, A) * ccw(C, D, B) <= 0) return true;
    else return false;
}
```

2.3 Point in Convex Hull Test

```
//have to include this header
#include <vector>

using namespace std;
```

```
typedef struct _Point {
    int x;
    int y;
} Point;

Point getVector(const Point& A, const Point& B);
int ccw(const Point& v, const Point& u);
int ccw(const Point& A, const Point& B, const Point& C);

//convexHull size >= 3
bool isInside(vector<Point>& convexHull, Point& A) {
    int O = 0;
    int L = 1, R = (int)convexHull.size() - 1;
    int M = (L + R) / 2;

    Point vecOL = getVector(convexHull[O], convexHull[L]);
    Point vecOA = getVector(convexHull[O], A);
    Point vecOR = getVector(convexHull[O], convexHull[R]);
    Point vecOM = getVector(convexHull[O], convexHull[M]);

    if(ccw(vecOL, vecOA) < 0) return false;
    if(ccw(vecOR, vecOA) > 0) return false;

    while(L + 1 != R) {
        M = (L + R) / 2;
        vecOM = getVector(convexHull[O], convexHull[M]);

        if(ccw(vecOM, vecOA) > 0) L = M;
        else R = M;
    }

    if(ccw(convexHull[L], A, convexHull[R]) <= 0) return true;
    else return false;
}
```

2.4 Rotating Calipers

```
#include <vector>

using namespace std;
typedef long long ll;

typedef struct _Point {
    int x;
    int y;
} Point;

using namespace std;

//should be defined
int ccw(const Point& v, const Point& u);

double rotCalipers(vector<Point>& convexHull);
    double ret = 987654321.0;
    int b = 1, f = 0;
    int s = (int)convexHull.size();
    bool flag = false;
    while(true) {
        Point frontVector = getVector(convexHull[f], convexHull[(f + 1) % s]);
        Point backVector = getVector(convexHull[b], convexHull[(b + 1) % s]);
        ret = max(ret, getDist(convexHull[f], convexHull[b]));

        if(ccw(frontVecTor, backVector) > 0) b = (b + 1) % s;
        else {
            f = (f + 1) % s;
            flag = true;
        }

        if(f == 0 && flag) break;
    }
}
```

3 Graph Thory

3.1 Bellman-Ford

```
//have to include these headers
#include <vector>
#include <utility>
#define INF 987654321

using namespace std;

int N;
vector<int> dist(N, INF);

//O(VE) bellman ford algorithm
void bellman_ford(vector<vector<pair<int, int>>> adj, int start) {
    dist[start] = 0;

    //update dist
    for(int i = 1; i <= (N - 1); ++i) { //after repeat N - 1 times, it completes.
        for(int j = 1; j <= N; ++j) {
            for(auto& edge : adj[j]) { //edge.first = destination, edge.second = distance
                dist[edge.first] = min(dist[j] + edge.second, dist[edge.first]);
            }
        }
    }

    //check negative cycle
    for(int i = 1; i <= N; ++i) {
        for(auto edge : adj[i]) {
            if(dist[edge.first] > dist[i] + edge.second) {
            //if dist changes, it means that it has negative cycle
                //cout << "YES\n";
                return;
            }
        }
    }

    //cout << "NO\n";
}
```

3.2 Edmonds-Karp

```
//time complex of this algorithm: min(O(Ef), O(VE^2))

//have to include these headers
#include <vector>
#include <queue>

using namespace std;
```

```cpp
#define V_MAX 1000
#define MAX 987654321

//adj matrix
int capacity[V_MAX][V_MAX] = { 0,};
int flow[V_MAX][V_MAX] = { 0,};

//edmonds-karp
int edmonds_karp (int source, int sink, int numOfVertex) {
    int result = 0;

    //each case of finding a path
    while (true) {
        vector<int> parent(V_MAX, -1);
        queue<int> q;
        q.push(source);
        parent[source] = source;

        //find a path(bfs)
        while (!q.empty() && parent[sink] == -1) {
            int cur = q.front();
            q.pop();
            for (int to = 0; to < numOfVertex; to++) {
                if (capacity[cur][to] - flow[cur][to] > 0 && parent[to] == -1) {
                    q.push(to);
                    parent[to] = cur;
                }
            }
        }

        //if there's no more path
        if (parent[sink] == -1) break;

        //if there's a path
        int amount = MAX;

        //find minimum residual capacity
        int now;
        for (now = sink; now != source; now = parent[now]) {
            amount = min(capacity[parent[now]][now] - flow[parent[now]][now], amount);
        }

        //edit flow
        for (int now = sink; now != source; now = parent[now]) {
            flow[parent[now]][now] += amount;
            flow[now][parent[now]] -= amount;
        }

        result += amount;
    }

    return result;
}
```

4 Linear Algebra

4.1 Gauss-Jordan

```cpp
//have to include this header
#include <vector>

using namespace std;

typedef struct _Matrix{
    int N;
    vector<vector<double>> matrix;

    _Matrix(int X) {
        N = X;
        matrix.resize(N, vector<double>(N + 1));
    } // N by N + 1 matrix
} Matrix;

void rowSwap(Matrix& A, int i) {
    vector<double> temp = A.matrix[i];
    A.matrix.erase(A.matrix.begin() + i);
    A.matrix.push_back(temp);
}

//Gauss-Jordan Elmination
void gaussJordan(Matrix& A) {
    for(int i = 0; i < A.N; ++i) {
        while(A.matrix[i][i] == 0) rowSwap(A, i);
        //check diagonal components are non-zero, when if, rotate row(swap)

        for(int j = 0; j < A.N; ++j) { //make RREF
            if(i != j) {
                double ratio = A.matrix[j][i] / A.matrix[i][i];
                for(int k = 0; k <= A.N; ++k) {
                    A.matrix[j][k] = A.matrix[j][k] - ratio * A.matrix[i][k];
                }
            }
        }
    }
}
```

5 Number Theory

5.1 Euler Phi Function

Euler Phi, 오일러 피 함수 : a이하의 수 중 a와 서로소가 되는 수들의 개수

$$\phi(a)$$

$$\phi(a) = a - 1$$

(a가 소수일 경우)

$$\phi(ab) = (ab - 1) - (a - 1) - (b - 1) = \phi(a)\phi(b)$$

(a, b가 서로 서로소인 관계)

$$\phi(a^m) = a^m - a^{m-1} = a^m(1 - \frac{1}{a})$$

(a가 소수일 경우)

$$\phi(x) = x(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})(1 - \frac{1}{p_3})...$$

(

$$x = p_1^k p_2^{k'} \dots$$

, 즉 일반적인 경우)