De La Salle ICPC Team Notebook (2018-19)

Contents

December 10, 2018

1 Advanced Data Structures Without Libraries

1.1 Union Find Disjoint Set

```
#include <vector>

using namespace std;

typedef vector<int> vi;

class UnionFind {
        public:
        vi p, rank, size;
        UnionFind(int n) {
                p.assign(n,0);
                rank.assign(n,0);
                size.assign(n,1);
                for(int i = 0; i < n; i++) { p[i] = i; }
        }

        void unionSet(int i, int j) {
                if(!isSameSet(i,j)) {
                        int x = findSet(i), y = findSet(j);
                        if(rank[x] > rank[y]) {
                                p[y] = x;
                        } else {
                                p[x] = y;
                                if(rank[x] == rank[y]) { rank[y]++; }
                        }
                        size[x]+=size[y];
                        size[y] = size[x];
                }
        }

        int findSet(int i) {
                if(p[i] != i) {
                        p[i] = findSet(p[i]);
                }
                return p[i];
        }

        bool isSameSet(int i, int j) { return findSet(i) == findSet(j);}
};
```

1.2 Segment Tree

```
#include <vector>

using namespace std;

typedef vector<int> vi;

/**
 * Segment tree for Range Maximum Queries
 */
class SegmentTree {
        public:
        vi st, A;
        int n;
        SegmentTree(const vi &_A) {
                A = _A;
                n = (int)A.size();
                st.assign(4 * n, 0);
                build(1, 0, n - 1);
        }

        int left(int p) { return 1 << p; }
        int right(int p) { return (1 << p) + 1; }
        void build(int p, int l, int r) {
                if(l == r) {
                        st[p] = l;
                } else {
                        int m = l + (r - l) / 2;
                        build(left(p),l,m);
                        build(right(p),m + 1,r);
                        int l = st[left(p)];
                        int r = st[right(p)];
                        st[p] = A[l] >= A[r] ? l : r;
                }
        }
        int rmq(int i, int j) {
                return rmq(1,0,n-1,i,j);
        }
        int rmq(int p, int l, int r, int i, int j) {
                if(l > j || r < i) { return -1; }
                else if(l >= i && r <= j) { return st[p]; }
                else {
                        int m = l + (r - l) / 2;
                        int s1 = rmq(left(p),l,m,i,j);
                        int s2 = rmq(right(p),m + 1,r,i,j);
                        if(s1 == -1) {return s2;}
                        else if(s2 == -1) {return s1;}
                        else {return A[l] >= A[r] ? l : r; }
                }
        }
        void pointUpdate(int i, int val) {
                pointUpdate(1,0,n-1,i,val);
        }
        void pointUpdate(int p, int l, int r,int i, int val) {
                if(l == r) {
                        A[i] = val;
                } else {
```

```
                        int m = l + (r - l) / 2;
                        if(i >= l && i <= m) {
                                pointUpdate(left(p),l,m,i,val);
                        } else {
                                pointUpdate(right(p),m + 1,r,i,val);
                        }
                        int l = st[left(p)];
                        int r = st[right(p)];
                        st[p] = A[l] >= A[r] ? l : r;
                }
        }
};
```

1.3 Fenwick Tree/Binary Indexed Tree

```
#include <vector>

using namespace std;

typedef vector<int> vi;

class FenwickTree {
        public:
        vi ft;
        int n;
        FenwickTree(int _n) {
                n = _n;
                ft.assign(n + 1,0);
        }

        void update(int index, int increment) {
                while(index <= n) {
                        ft[index] += increment;
                        index += (index & (-index));
                }
        }

        int rmq(int end) {
                int res = 0;
                while(end) {
                        res += ft[end];
                        end -= (end & (-end));
                }
                return res;
        }

        int rmq(int i, int j) {
                if(i > j) { return 0; }
                else if(i <= 0) { return rmq(j); }
                else {return rmq(j) - rmq(i - 1)}
        }
};
```

2 Problem Solving Paradigms

2.1 Maximum 1D/2D Range Sum

```
#include <vector>

using namespace std;

typedef vector<int> vi;

int max1DRangeSum(const vi &arr) {
        int ans = 0;
        int cur = 0;
        for(int i = 0; i < (int)arr.size(); i++) {
                cur += arr[i];
                if(cur < 0) cur = 0;
                ans = max(ans,cur);
        }
        return ans;
}

int max2DRangeSum(const vector<vi> &arr) {
        vi rowSums(arr.size(),0);
        for(int i = 0; i < (int)arr.size(); i++) {
                for(int j = 0; j < (int)arr[i].size(); j++) {
                        rowSums[i] += arr[i][j];
                }
        }

        int ans = 0;
        for(int leftCol = 0; leftCol < (int)arr[0].size(); leftCol++) {
                for(int rightCol = leftCol; rightCol < (int)arr[0].size(); rightCol++) {
                        vi arr2(arr.size(),0);
                        for(int i = 0; i < (int)arr.size(); i++) {
                                arr2[i] = leftCol == 0 ? rowSums[rightCol] :
                                                                        (rowSums[rightCol] - rowSums[
                                                                                leftCol - 1]);
                        }
                        int temp = max1DRangeSum(arr2);
                        ans = max(ans,temp);
                }
        }
        return ans;
}
```

2.2 Longest Increasing Subsequence

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

using namespace std;

typedef vector<int> vi;

int lisSlow(const vi &arr) {
        vi memo(arr.size(),1);
        int ans = 1;
        for(int i = 1; i < (int)arr.size(); i++) {
                for(int j = 0; j < i; j++) {
                        if(arr[i] > arr[j]) {
                                memo[i] = max(memo[i],memo[j] + 1);
                        }
                }
                ans = max(ans,memo[i]);
        }
        return ans;
}

int lisNLogN(const vi &arr) {
        vi memo;

        for(int i = 0; i < (int)arr.size(); i++) {
                vi::iterator itr = lower_bound(memo.begin(),memo.end(),arr[i]);
                if(itr == memo.end()) {
                        memo.push_back(arr[i]);
                } else {
                        *itr = arr[i];
                }
        }
        return (int)memo.size();
}
```

2.3 0/1 Knapsack

```
#include <vector>

using namespace std;

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

/**
 * item.first - weight of item
 * item.second - value of item
 */
int zeroOneKnapsack(const vii &items, int maxWeight) {
        vector<vi> memo(items.size(),vi(maxWeight + 1,0));
        for(int i = 0; i < (int)items.size(); i++ ) {
                for(int j = 0; j <= maxWeight; j++) {
                        if(i == 0){
                                memo[i][j] = j >= items[0].first ? items[0].second : 0;
                        } else {
                                memo[i][j] = memo[i - 1][j];
                                if(j >= items[i].first) {
                                        memo[i][j] = max(memo[i][j],
                                                        memo[i - 1][j - items[i].first] + items[i].second);
                                }
                        }
                }
```

```
        }
        return memo[(int)items.size() - 1][maxWeight];
}
```

2.4 Coin Change

```
#include <vector>

using namespace std;

typedef pair<int,int> ii;
typedef vector<int> vi;

/**
 * returns minimum coins
 */
int coinChangeV1(const vi &denoms, int total) {
        vi memo(total + 1,1 << 29);
        memo[0] = 0;
        for(int i = 1; i <= total; i++) {
                for(int j = 0; j < (int)denoms.size(); j++) {
                        if(i - denoms[j] >= 0) {
                                memo[i] = min(memo[i],memo[i - denoms[j]] + 1);
                        }
                }
        }
        return memo[total];
}

/**
 * returns number of ways to give change
 */
int coinChangeV2(const vi &denoms, int total) {
        vector<vi> memo(denoms.size(),vi(total + 1,0));
        for(int i = 0; i < (int)denoms.size(); i++) {
                for(int j = 0; j <= total; j++) {
                        if(i == 0) {
                                memo[i][j] = j % denoms[0] ? 1 : 0;
                        } else {
                                memo[i][j] = 0;
                                int count = 0;
                                while(j - count * denoms[i] >= 0) {
                                        memo[i][j] += memo[i - 1][j - count * denoms[i]];
                                        count++;
                                }
                        }
                }
        }
        return memo[(int)denoms.size() - 1][total];
}
```

2.5 Traveling Salesman Problem

```
#include <vector>

using namespace std;

typedef vector<int> vi;

int n; // number of nodes
int start; //start node

vector<vi> memo(n,vi(1 << n, -1));
vector<vi> dists; // 2D distance matrix

int tsp(int pos, int mask) {
        if(mask == (1 << n) - 1) {
                return dists[pos][start];
        } else if(memo[pos][mask] != -1) {
                return memo[pos][mask];
        } else {
                int &res = memo[pos][mask];
                for(int i = 0; i < n; i++) {
                        if((mask & (1 << i)) == 0) {
                                res = min(res,dists[pos][i] + tsp(i,mask | (1 << i)));
                        }
                }
                return res;
        }
}
```

3 Graphs

3.1 Depth First Search

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

using namespace std;

#define UNVISITED 1
#define VISITED 2

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> adjList;
vi dfs_num;

void dfs(int u){
        dfs_num[u] = VISITED;
        for(int i = 0; i < (int)adjList[u].size(); i++) {
                ii v = adjList[u][i];
                if(dfs_num[v.first] == UNVISITED) {
                        dfs(v.first);
                }
        }
}
```

3.2 Breadth First Search

```
#include <vector>
#include <utility>
#include <queue>
#include <bitset>

using namespace std;

#define MAX_N 1000

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> adjList;

void bfs(int s){
        bitset<MAX_N> visited;
        queue<int> q;
        q.push(s);
        visited.set(s);
        while(!q.empty()) {
                int u = q.front(); q.pop();
                for(int i = 0; i < (int)adjList[u].size(); i++) {
                        ii v = adjList[u][i];
                        if(!visited.test(v.first)) {
                                visited.set(v.first);
                                q.push(v.first);
                        }
                }
        }
}
```

3.3 Connected Components

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

using namespace std;

#define UNVISITED 1
#define VISITED 2

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
```

```
vector<vii> adjList;
vi dfs_num;
int V;

void dfs(int u){
        dfs_num[u] = VISITED;
        for(int i = 0; i < (int)adjList[u].size(); i++) {
                ii v = adjList[u][i];
                if(dfs_num[v.first] == UNVISITED) {
                        dfs(v.first);
                }
        }
}

int countCC() {
        int cc = 0;
        for(int i = 0; i < V; i++) {
                if(dfs_num[i] == UNVISITED ){
                        cc++;
                        dfs(i);
                }
        }
        return cc;
}
```

3.4 Flood Fill

```
#include <vector>
#include <utility>
#include <queue>

using namespace std;

#define MAX_R 1000
#define MAX_C 1000

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

int grid[MAX_R][MAX_C];
int dr[] = {-1,-1,-1,0,0,1,1,1};
int dc[] = {-1,0,1,-1,1,-1,0,1};

int iterativeFloodFill(int r, int c, int v1, int v2){
        if(r >= 0 && r < MAX_R && c >= 0 && c < MAX_C && grid[r][c] == v1) {
                queue<int> q;
                q.push(r * MAX_C + c);
                int area = 0;
                while(!q.empty()) {
                        int u = q.front(); q.pop();
                        int inR = u / MAX_C;
                        int inC = u % MAX_C;
                        grid[inR][inC] = v2;
                        area++;
                        for(int i = 0; i < 4; i++) {
                                int nr = inR + dr[i];
                                int nc = inC + dc[i];
                                if(nr >= 0 && nr < MAX_R && nc >= 0 && nc < MAX_C &&
                                        grid[nr][nc] == v1) {
                                        q.push(nr * MAX_C + nc);
                                }
                        }
                }
                return area;
        }
        return 0;
}

int recursiveFloodFill(int r, int c, int v1, int v2) {
        if(r >= 0 && r < MAX_R && c >= 0 && c < MAX_C && grid[r][c] == v1) {
                grid[r][c] = v2;
                int area = 1;
                for(int i = 0; i < 4; i++) {
                        int nr = r + dr[i];
                        int nc = c + dc[i];
                        area += recursiveFloodFill(nr,nc,v1,v2);
                }
                return area;
        } else {
                return 0;
        }
}
```

3.5 Bipartite Graph Check

```
#include <vector>
#include <utility>
#include <queue>
#include <bitset>

using namespace std;

#define MAX_N 1000
#define NO_COLOR -1

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

int V;
vector<vii> adjList;

bool isBipartite(int s){
        queue<int> q;
        q.push(s);
        vi color(V,NO_COLOR);
        color[s] = 0;
        bool isBipartite = true;
        while(!q.empty() && isBipartite) {
                int u = q.front(); q.pop();
                for(int i = 0; i < (int)adjList[u].size(); i++) {
                        ii v = adjList[u][i];
                        if(color[v.first] == NO_COLOR) {
                                color[v.first] = 1 - color[u];
                                q.push(v.first);
                        } else if(color[v.first] == color[u]) {
                                isBipartite = false;
                        }
                }
        }
        return isBipartite;
}
```

3.6 Edge Property Check

```
#include <cstdio>
#include <vector>
#include <utility>

using namespace std;

#define UNVISITED 1
#define VISITED 2
#define EXPLORED 3

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> adjList;
vi dfs_parent;
vi dfs_num;

void dfs(int u){
        dfs_num[u] = VISITED;
        for(int i = 0; i < (int)adjList[u].size(); i++) {
                ii v = adjList[u][i];
                if(dfs_num[v.first] == UNVISITED) {
                        printf("Edge (%d,%d): Tree Edge\n",u,v.first);
                        dfs_parent[v.first] = u;
                        dfs(v.first);
                } else if(dfs_num[v.first] == VISITED) {
                        if(v.first == dfs_parent[u]) {
                                printf("Edge (%d,%d): Two-Way Edge\n",u,v.first);
                        } else {
                                printf("Edge (%d,%d): Back Edge\n",u,v.first);
                        }
                } else {
                        printf("Edge (%d,%d): Forward/Cross Edge\n",u,v.first);
                }
        }
        dfs_num[u] = EXPLORED;
}
```

3.7 Finding Articulation Points And Bridges

```
#include <vector>
#include <utility>
#include <bitset>

using namespace std;

#define MAX_N 1000

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> adjList;
vi dfs_num, dfs_low, dfs_parent;
bitset<MAX_N> isArticulationPoint;

int dfsCtr = 0, V, children, dfsRoot;

void dfs(int u,vii &bridges){
        dfs_num[u] = dfs_low[u] = dfsCtr++;
        for(int i = 0; i < (int)adjList[u].size(); i++) {
                ii v = adjList[u][i];
                if(dfs_num[v.first] == -1) {
                        dfs_parent[v.first] = u;
                        dfs(v.first,bridges);
                        if(u == dfsRoot) {
                                children++;
                        }

                        dfs_low[u] = min(dfs_low[u],dfs_low[v.first]);
                        if(dfs_num[u] <= dfs_low[v.first]) {
                                isArticulationPoint.set(u);
                        }
                        if(dfs_num[u] < dfs_low[v.first]) {
                                bridges.push_back(ii(u,v.first));
                        }
                } else if(v.first != dfs_parent[u]) {
                        dfs_low[u] = min(dfs_low[u],dfs_num[v.first]);
                }
        }
}

pair<vi,vii> findArticulationPointsAndBridges(){
        vii bridges;
        isArticulationPoint.reset();
        dfs_num.assign(V,-1);
        dfs_low.assign(V,-1);
        dfs_parent.assign(V,-1);
        for(int i = 0; i < V; i++) {
                if(dfs_num[i] == -1) {
                        children = 0;
                        dfsRoot = i;
                        dfs(i,bridges);
                        if(children > 1) {
                                isArticulationPoint.set(i);
                        } else {
                                isArticulationPoint.reset(i);
                        }
                }
        }
        vi artPoints;
        for(int i = 0; i < V; i++) {
                if(isArticulationPoint.test(i)) {
                        artPoints.push_back(i);
                }
        }
        return pair<vi,vii>(artPoints,bridges);
}
```

3.8 Finding Strongly Connected Components

```
#include <vector>
#include <utility>
#include <bitset>
#include <stack>

using namespace std;

#define MAX_N 1000

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

vector<vii> adjList;
vi dfs_num, dfs_low;
bitset<MAX_N> visited;

int dfsCtr = 0, V, scc;

void tarjan(int u,stack<int> &s){
        // printf("Tarjan(%d)\n",u);
        dfs_num[u] = dfs_low[u] = dfsCtr++;
        s.push(u);
        visited.set(u);
        for(int i = 0; i < (int)adjList[u].size(); i++) {
                ii v = adjList[u][i];
                if(dfs_num[v.first] == -1) {
                        tarjan(v.first,s);
                }
                if(visited.test(u)) {
                        dfs_low[u] = min(dfs_low[u],dfs_low[v.first]);
                }
        }
        if(dfs_low[u] == dfs_num[u]) {
                printf("SCC #%d:",++scc);
                while(s.top() != u) {
                        int temp = s.top();     s.pop();
                        printf(" %d",temp);
                }
                printf(" %d\n",s.top());
                s.pop();
        }
}

int printAndCountSCCs(){
        scc = 0;
        visited.reset();
        stack<int> s;
        dfs_num.assign(V,-1);
        dfs_low.assign(V,-1);
        for(int i = 0; i < V; i++) {
                // printf("dfs_num[%d] = %d\n",i,dfs_num[i]);
                if(dfs_num[i] == -1) {
                        tarjan(i,s);
                }
        }
        return scc;
}
```

3.9 Minimum Spanning Tree Using Kruskal's Algorithm

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

using namespace std;

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
typedef pair<int,ii> iii;

class UnionFind {
        public:
        vi p, rank;
        UnionFind(int n) {
                p.assign(n,0);
                rank.assign(n,0);
                for(int i = 0; i < n; i++) { p[i] = i; }
        }

        void unionSet(int i, int j) {
                if(!isSameSet(i,j)) {
                        int x = findSet(i), y = findSet(j);
                        if(rank[x] > rank[y]) {
                                p[y] = x;
                        } else {
                                p[x] = y;
                                if(rank[x] == rank[y]) { rank[y]++; }
                        }
                }
        }

        int findSet(int i) {
                if(p[i] != i) {
                        p[i] = findSet(p[i]);
                }
                return p[i];
```

```
        }

        bool isSameSet(int i, int j) { return findSet(i) == findSet(j);}
};

int V,E;
vector<iii> edgeList;

/**
 * Store edges in edge list where first contains the edge weight and second
 * contains a pair with the endpoints.
 */
int kruskalsMST() {
        int mst = 0;

        sort(edgeList.begin(),edgeList.end());
        UnionFind uf(V);
        for(int i = 0; i < E; i++) {
                iii edge = edgeList[i];
                if(!uf.isSameSet(edge.second.first,edge.second.second)) {
                        uf.unionSet(edge.second.first,edge.second.second);
                        mst += edge.first;
                }
        }
}
```

3.10 Minimum Spanning Tree Using Prim's Algorithm

```
#include <vector>
#include <utility>
#include <queue>
#include <bitset>

using namespace std;

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

#define MAX_N 1000

int V;
vector<vii> adjList;

int primsMST(){
        bitset<MAX_N> taken;
        priority_queue<ii> q;
        q.push(ii(0,0));
        int mst = 0;
        while(!q.empty()) {
                ii cur = q.top(); q.pop();
                int u = -cur.second, w = -cur.first;
                if(!taken.test(u)) {
                        taken.set(u);
                        for(int i = 0; i < (int)adjList[u].size(); i++) {
                                ii v = adjList[u][i];
                                if(!taken.test(v.first)) {
                                        q.push(ii(-v.second,-v.first));
                                }
                        }
                        mst += w;
                }
        }
        return mst;
}
```

3.11 Single Source Shortest Path using Dijkstra's Algorithm

```
#include <vector>
#include <utility>
#include <queue>

using namespace std;

typedef pair<int,int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;

#define INF 1000000000

int V;

vector<vii> adjList;

vi dijkstras(int s){
        vi dist(V,INF);
        priority_queue<ii,vii,greater<ii> > q;
        q.push(s);
        dist[s] = 0;
        while(!q.empty()) {
                ii u = q.top(); q.pop();
                if(u.first > dist[u.second]) {
                        continue;
                }
                for(int i = 0; i < (int)adjList[u.second].size(); i++) {
                        ii v = adjList[u.second][i];
                        if(dist[v.first] > dist[u.second] + v.second) {
                                dist[v.first] = dist[u.second] + v.second;
                        }
                }
        }
        return dist;
}
```

3.12 All Pairs Shortest Path using Floyd Warshall's Algorithm

```
#include <vector>

using namespace std;

typedef vector<int> vi;

#define INF 1000000000

int V;
vector<vi> adjMat;

vector<vi> dijkstras(int s){
        for(int k = 0; k < V; k++)
                for(int i = 0; i < V; i++)
                        for(int j = 0; j < V; j++)
                                adjMat[i][j] = min(adjMat[i][j],adjMat[i][k] + adjMat[k][j]);
        return adjMat;
}
```

3.13 Max Flow with Edmonds Karp Algorithm

```
#include <vector>
#include <bitset>
#include <queue>

using namespace std;

#define MAX_N 1000
#define INF 1000000000

typedef vector<int> vi;

class Edge {
        public:
        int u, v, cap, rem;
        Edge(int _u, int _v, int _cap) : u(_u), v(_v), cap(_cap), rem(_cap) {}
};

int V, f, s, t;
vi p;
vector<vector<Edge *> > adjList;
Edge *res[MAX_N][MAX_N];

void augment(int v, int minEdge) {
        // printf("augmenting %d\n",v);
        if(v == s) {
                f = minEdge;
        } else if(p[v] != -1) {
                augment(p[v],min(minEdge,res[p[v]][v]->rem));
                res[p[v]][v]->rem -= f;
                res[v][p[v]]->rem += f;
        }
}

int edmondsKarp(int source, int sink) {
        int mf = 0;
        s = source;
        t = sink;
```

```
        while(true) {
                f = 0;
                bitset<MAX_N> visited;
                queue<int> q;
                q.push(s);
                p.assign(V,-1);
                visited.set(s);
                while(!q.empty()) {
                        int u = q.front(); q.pop();
                        // printf("exploring %d\n",u);
                        if(u == t) {
                                break;
                        } else {
                                for(int i = 0; i < (int)adjList[u].size(); i++) {
                                        Edge *e = adjList[u][i];
                                        if(e->rem > 0 && !visited.test(e->v)){
                                                p[e->v] = u;
                                                visited.set(e->v);
                                                q.push(e->v);
                                        }
                                }
                        }
                }

                augment(t,INF);
                if( f == 0) {
                        break;
                }
                mf += f;
        }
        return mf;
}

typedef pair<int,int> ii;
typedef pair<int,ii> iii;

int main() {
        V = 7;
        adjList.assign(V,vector<Edge*>());

        /**
         * This is a sample graph.
         */
        vector<iii> edges;
        edges.push_back(iii(0,ii(1,10)));
        edges.push_back(iii(0,ii(4,10)));
        edges.push_back(iii(1,ii(2,20)));
        edges.push_back(iii(2,ii(3,10)));
        edges.push_back(iii(4,ii(1,10)));
        edges.push_back(iii(2,ii(5,5)));
        edges.push_back(iii(3,ii(6,10)));
        edges.push_back(iii(5,ii(6,10)));

        for(int i = 0; i < (int)edges.size(); i++) {
                /**
                 * This is how to add an edge.
                 */
                iii cur = edges[i];
                Edge *e = new Edge(cur.first,cur.second.first,cur.second.second);
                Edge *eRev = new Edge(cur.second.first,cur.first,0);
                adjList[cur.first].push_back(e);
                adjList[cur.second.first].push_back(eRev);
                res[cur.first][cur.second.first] = e;
                res[cur.second.first][cur.first] = eRev;
        }
        printf("maxflow: %d\n",edmondsKarp(0,6));
        return 0;
}
```

3.14 Max Flow with Dinic's Algorithm

```
#include <vector>
#include <bitset>
#include <queue>

using namespace std;

#define MAX_N 1000
#define INF 1000000000

typedef vector<int> vi;

class Edge {
        public:
        int u, v, cap, rem;
        Edge(int _u, int _v, int _cap) : u(_u), v(_v), cap(_cap), rem(_cap) {}
};

int V, f, s, t;
vi p;
vector<vector<Edge *> > adjList;
Edge *res[MAX_N][MAX_N];

void augment(int v, int minEdge) {
        // printf("augmenting %d\n",v);
        if(v == s) {
                f = minEdge;
        } else if(p[v] != -1) {
                augment(p[v],min(minEdge,res[p[v]][v]->rem));
                res[p[v]][v]->rem -= f;
                res[v][p[v]]->rem += f;
        }
}

int dinicsMaxFlow(int source, int sink) {
        int mf = 0;
        s = source;
        t = sink;
        bool hasFlow = true;
        while(hasFlow) {
                vi dist(V,INF);
                queue<int> q;
                q.push(s);
                dist[s] = 0;
                while(!q.empty()) {
                        int u = q.front(); q.pop();
                        // printf("exploring %d\n",u);
                        if(u == t) {
                                break;
                        } else {
                                for(int i = 0; i < (int)adjList[u].size(); i++) {
                                        Edge *e = adjList[u][i];
                                        if(e->rem > 0 && dist[e->v] == INF){
                                                dist[e->v] = dist[u] + 1;
                                                q.push(e->v);
                                        }
                                }
                        }
                }
                hasFlow = false;
                while(true) {
                        f = 0;
                        bitset<MAX_N> visited;
                        q = queue<int>();
                        q.push(s);
                        p.assign(V,-1);
                        visited.set(s);
                        while(!q.empty()) {
                                int u = q.front(); q.pop();
                                if(u == t) {
                                        break;
                                } else {
                                        for(int i = 0; i < (int)adjList[u].size(); i++) {
                                                Edge *e = adjList[u][i];
                                                if(e->rem > 0 && !visited.test(e->v) && dist[e->v] ==
                                                        dist[u] + 1){
                                                  p[e->v] = u;
                                                  visited.set(e->v);
                                                  q.push(e->v);
                                                }
                                        }
                                }
                        }
                        augment(t,INF);
                        if( f == 0) {
                                break;
                        }
                        mf += f;
                        hasFlow = true;
                }
        }
        return mf;
}

typedef pair<int,int> ii;
typedef pair<int,ii> iii;

int main() {
        V = 7;
        adjList.assign(V,vector<Edge*>());

        /**
         * This is a sample graph.
         */
        vector<iii> edges;
        edges.push_back(iii(0,ii(1,10)));
        edges.push_back(iii(0,ii(4,10)));
        edges.push_back(iii(1,ii(2,20)));
        edges.push_back(iii(2,ii(3,10)));
```

```
        edges.push_back(iii(4,ii(1,10)));
        edges.push_back(iii(2,ii(5,5)));
        edges.push_back(iii(3,ii(6,10)));
        edges.push_back(iii(5,ii(6,10)));

        for(int i = 0; i < (int)edges.size(); i++) {
                /**
                 * This is how to add an edge.
                 */
                iii cur = edges[i];
                Edge *e = new Edge(cur.first,cur.second.first,cur.second.second);
                Edge *eRev = new Edge(cur.second.first,cur.first,0);
                adjList[cur.first].push_back(e);
                adjList[cur.second.first].push_back(eRev);
                res[cur.first][cur.second.first] = e;
                res[cur.second.first][cur.first] = eRev;
        }
        printf("maxflow: %d\n",dinicsMaxFlow(0,6));
        return 0;
}
```

4 Combinatorics

4.1 Combinatorics

```
public class Combinatorics {
        static int[] fibo(int n) {
                int[] f = new int[n];
                f[0] = f[1] = 1;
                for(int i=2; i<n; i++) f[i] = f[i-1]+f[i-2];
                return f;
        }

        // alternatively:
        // nCk = n!/(k!*(n-k)!)
        // nPk = n!/(n-k)!
        // add a memo table if needed to reduce overlaps
        static int binomialCoefficient(int n, int k) {
                if(n==k || k==0) return 1;
                return binomialCoefficient(n-1,k-1)+binomialCoefficient(n-1,k);
        }

        static int[] catalan(int n) {
                int[] cat = new int[n];
                cat[0] = 1;
                for(int i=1; i<n; i++) cat[i] = ((2*i*(2*i-1)) / ((i+1)*i))*cat[i-1];
                return cat;
        }
}
```

5 Number Theory

5.1 NumberTheory

```
import java.util.*;

public class NumberTheory {
        static BitSet sieve; // used for generating primes
        static ArrayList<Integer> primes; // change to long if necessary
        static int[] nDiffPF; // for modified sieve
        static int bound; // array cap, usually 10^7
        static int[] e; // for modified sieve euler phi

        static void generatePrimes() {
                sieve.set(2, bound);
                for(int i=2; i<bound; i++)
                        if(sieve.get(i)) {
                                primes.add(i);
                                for(int j=i+i; i<bound; j+=i) sieve.set(j, false);
                        }
        }

        static boolean isPrime(int n) {
                if(n<bound) return sieve.get(n);
                for(int i=0; i<primes.size(); i++)
                        if(n%primes.get(i)==0) return false;
                return true;
        }

        // assumption: a>=b
        static int gcd(int a, int b) {
                return b==0 ? a : gcd(b, a%b);
        }

        // alternative: get max of powers across all prime factors of a and b
        static int lcm(int a, int b) {
                return a*(b/gcd(a, b));
        }

        // returns map of format <factor, power>
        // convert to TreeMap if you need order
        // easy to tweak to count all PF, count distinct PF, sum PF, num Div
        static HashMap<Integer, Integer> getPrimeFactors(int n) {
                HashMap<Integer, Integer> factors = new HashMap<>();
                int pIndex = 0, p = primes.get(pIndex);
                while(p*p<=n) {
                        while(n%p==0) {
                                factors.put(p, factors.containsKey(p) ? factors.get(p)+1 : 1);
                                n/=p;
                        }
                        p = factors.get(++pIndex);
                }
                if(n!=1) factors.put(n, factors.containsKey(n) ? factors.get(n)+1 : 1);
                return factors;
        }

        static int eulerPhi(int n) {
                int pIndex = 0, p = primes.get(pIndex), ans = n;
                while(p*p<=n) {
                        if(n%p==0) ans-=ans/p;
                        while(n%p==0) n/=p;
                        p = primes.get(++pIndex);
                }
                if(n!=1) ans-=ans/n;
                return ans;
        }

        // more efficient way to count num of different PF of many numbers
        static void modifiedSieve() {
                nDiffPF = new int[bound];
                for(int i=2; i<bound; i++)
                        if(nDiffPF[i]==0) for(int j=i; j<bound; j+=i) nDiffPF[j]++;
        }

        static void modifiedEulerPhi() {
                e = new int[bound];
                for(int i=1; i<bound; i++) e[i] = i;
                for(int i=2; i<bound; i++)
                        if(e[i]==i) for(int j=i; j<bound; j+=i) e[j] = (e[j]/i)*(i-1);
        }
}
```

6 Miscellaneous Mathematics

7 String Processing

8 Computational Geometry

8.1 One Dimensional Objects

```
#include <cmath>

class Point{
        public:
        double x, y;
        Point() { x = y = 0; }
        Point(double _x, double _y) : x(_x), y(_y) {}

        bool operator<(const Point &p) const {
                if(fabs(x - p.x) > 1e-10) {
                        return x < p.x;
                } else {
                        return y < p.y;
                }
        }
```

```
        bool operator==(const Point &p) const {
                return (fabs(x - p.x) < 1e-10 && fabs(y - p.y) < 1e-10);
        }

        /**
         * theta in radians
         */
        Point rotate(double theta) {
                return Point(cos(theta) * x - sin(theta) * y,
                                                sin(theta) * x + cos(theta) * y);
        }

        double dist(const Point &p) const {
                return hypot(x - p.x, y - p.y);
        }
};

class Line{
        public:
        double a, b, c;

        Line(Point from, Point to) {
                if(fabs(from.x - to.x) < 1e-10) {
                        a = 1;
                        b = 0;
                        c = -from.x;
                } else {
                        a = (to.y - from.y) / (to.x - from.x);
                        b = 1;
                        c = -(a * from.x) - from.y;
                }
        }

        bool areParallel(const Line &l2) const {
                return (fabs(a - l2.a) < 1e-10 && fabs(b - l2.b) < 1e-10);
        }

        bool areSame(const Line &l2) const {
                return areParallel(l2) && fabs(c - l2.c) < 1e-10;
        }

        bool areIntersect(const Line &l2, Point &p) {
                if(areParallel(l2)) {
                        return false;
                }
                p.x = (l2.b * c - b * l2.c) / (l2.a * b - a * l2.b);
                if(fabs(b) > 1e-10) {
                        p.y = -(a * p.x + c);
                } else {
                        p.y = -(l2.a * p.x + l2.c);
                }
                return true;
        }
};

class Vector {
        public:
        double x, y;

        Vector(double _x, double _y) : x(_x), y(_y) {}
        Vector(Point src, Point dest) {
                x = dest.x - src.x;
                y = dest.y - src.y;
        }
        Vector scale(double c) {
                return Vector(x * c,y * c);
        }
        Point translate(Point p) {
                return Point(p.x + x,p.y + y);
        }

        double dot(const Vector &b) const {
                return x * b.x + y * b.y;
        }

        double norm() const {
                return x * x + y * y;
        }
};

/**
 * distance of point p from line AB
 */
double distToLine(Point p, Point a, Point b, Point &c) {
        Vector ap = Vector(a,p), ab = Vector(a,b);
        double u = ap.dot(ab) / ab.norm();
        ab.scale(u).translate(a);
        return p.dist(c);
}
```

8.2 Circles

```
#include <cmath>

/**
 * Checks if p is in circle with center c and radius r.
 * Returns 0 if inside, 1 if on border, and 2 if outside.
 */
int insideCircle(Point p, Point c, double r) {
        double dx = p.x - c.x, dy = p.y - c.y;
        double dist = hypot(dx,dy);
        return dist < r - 1e-10 ? 0 : (dist < r + 1e-10 ? 1 : 2);
}

double getPI() {
        return acos(-1.0);
}

/**
 * returns area of circle with radius r
 */
double circleArea(double r) {
        return getPI() * r * r;
}

/**
 * returns arc length of circle with radius r and vertex angle theta (in radians)
 */
double arcLength(double r, double theta) {
        return theta * r;
}

/**
 * returns chord length of circle with radius r and vertex angle theta (in radians)
 */
double chordLength(double r, double theta) {
        return sqrt(2 * r * r * (1 - cos(theta)));
}

/**
 * returns area of sector of circle with radius r and vertex angle theta (in radians)
 */
double sectorArea(double r, double theta) {
        return r * r * theta / 2;
}

/**
 * returns area of segment of circle with radius r and vertex angle theta (in radians)
 * segment is section between chord and arc
 */
double segmentArea(double r, double theta) {
        return sectorArea(r,theta) - r * cos(theta / 2.0) * r * sin(theta / 2.0);
}

/**
 * given two points p1 and p2, find the center of circles, c, with radius r.
 */
bool circle2PtsRad(Point p1, Point p2, double r, Point &c) {
        double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
                    (p1.y - p2.y) * (p1.y - p2.y);
        double det = r * r / d2 - 0.25;
        if (det < 0.0) return false;
        double h = sqrt(det);
        c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
        c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
        return true;
}
```

8.3 Triangles

```
#include <cmath>
#include "OneDim.cpp"

/**
 * returns area of triangle with base b and height h.
 */
double area(double b, double h) {
        return b * h * 0.5;
}

/**
 * returns perimeter of triangle with sides a, b, and c.
 */
double perimeter(double a, double b, double c) {
```

```
        return a + b + c;
}

/**
 * returns semiperimeter of triangle with sides a, b, and c.
 */
double semiperimeter(double a, double b, double c) {
        return (a + b + c) / 2.0;
}

/**
 * returns area of triangle with sides a, b, and c.
 */
double area(double a, double b, double c) {
        double s = semiperimeter(a,b,c);
        return sqrt(s * (s - a) * (s - b) * (s - c));
}

/**
 * returns radius of incircle of triangle with sides a, b, and c.
 */
double incircleradius(double a, double b, double c) {
        return area(a,b,c) / semiperimeter(a,b,c);
}

/**
 * returns radius of incircle of triangle with sides a, b, and c.
 */
double rInCircle(Point p1, Point p2, Point p3) {
        return incircleradius(p1.dist(p2),p2.dist(p3),p3.dist(p1));
}

// assumption: the required points/lines functions have been written
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(Point p1, Point p2, Point p3, Point &ctr, double &r) {
        r = rInCircle(p1, p2, p3);
        if (fabs(r) < 1e-10)
                return 0; // no inCircle center
        double ratio = p1.dist(p2) / p1.dist(p3);
        Vector p2p3(p2, p3);
        Point p = p2p3.scale(ratio / (1 + ratio)).translate(p2);
        Line l1(p1,p);
        ratio = p2.dist(p1) / p2.dist(p3);
        Vector p1p3(p2, p3);
        p = p1p3.scale(ratio / (1 + ratio)).translate(p1);
        Line l2(p2, p);
        l1.areIntersect(l2, ctr); // get their intersection point
        return 1;
}

/**
 * returns radius of circumcircle of triangle with sides a, b, and c.
 */
double circumcircleradius(double a, double b, double c) {
        return a * b * c / 4 * area(a,b,c);
}

/**
 * returns radius of incircle of triangle with sides a, b, and c.
 */
double rCircumCircle(Point p1, Point p2, Point p3) {
        return circumcircleradius(p1.dist(p2),p2.dist(p3),p3.dist(p1));
}
```