implement

```
vector<pair<int,int>> adj[MAX];
struct edge{int u, v, w;};
edge edges[1001];
sort(vec.begin(),vec.end(),[](int a,int b){return a<b;});
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
struct cmp {
    bool operator () (const T &a, const T &b) {
       return a.x < b.x;
    }
};
priority_queue<T,vector<T>,cmp> pq;
```

shortest path

```
Bellman-Ford(negative edge is allowed)
memset(dist, 0x3f, sizeof(dist)),dist[s] = 0;
for (int i = 0; i < n - 1; i++){
    for (auto e: edges){
        relax(e.u, e.v, e.w);
    }
}// O(|V|·|E|)
```

```
Dijskra(negative edge is not allowed)
dist[s]=0,others:INF
priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;//first distance,second index
pq.push({0,s});
while(!pq.empty){
    int d=pq.top().first, u=pq.top().second;
    pq.pop();
    if(d>dist[u]) continue;
    for(auto e: adj[u]){
        int v = e.first, w = e.second;
        if(dist[v] > dist[u] + w){
            dist[v] = dist[u] + w;
            pq.push({dist[v], v});
        }
    }
}// O(|E|+|V|·log|E|)
// 最短路径数量
```

```
Floyd
//dist[k][i][j]=min(dist[k-1][i][j],
//dist[k-1][i][k]+dist[k-1][k][j])
dist_pre[i][i]=0,dist_pre[u][v]=w,others:INF
vector<vector<int>> dist(MAX,vector<int>(MAX,0));
for (int k = 1; k <= n;k++){
    for (int i = 1; i <= n;i++)
        for (int j = 1; j <= n;j++)
            dist_now[i][j]=min(dist_pre[i][j],
            dist_pre[i][k]+dist_pre[k][j]);
    dist_pre = dist_now;
}// O(|V|^3)  // 最小环
```

SCC

```
Kasaraju
vector<int> adj[MAX],adj_inv[MAX];
vector<vector<int>> scc;
vector<int> post_order,temp_scc,scc_index(MAX);
bool visit[MAX];
void DFS1(int s){// DFS on the initial graph
    visited[s] = true;
    for (int i = 0; i < adj[s].size(); i++){
        if(!visited[adj[s][i]]){
            DFS1(adj[s][i]);
        }
    }
    post_order.push_back(s);
}
void DFS2(int s){// DFS on the inverse graph
    visited[s] = true;
    temp_scc.push_back(s);
    for (int i = 0; i < adj_inv[s].size(); i++){
        if(!visited[adj_inv[s][i]]){
            DFS2(adj_inv[s][i]);
        }
    }
}
memset(visited, false, sizeof(visited));
// calculate the post_order in the initial graph
for (int i = 1; i <= n; i++)
    if(!visited[i])
        DFS1(i);
memset(visited, false, sizeof(visited));
for(int i = post_order.size()-1; i >= 0; i--){
    if(!visited[post_order[i]]){
        vector<int> ().swap(temp_scc);
        DFS2(post_order[i]);
        scc.push_back(temp_scc);
        for (int j = 0; j < temp_scc.size(); j++)
            scc_index[temp_scc[j]] = scc.size() - 1;
    }
}// scc_index[u]!=scc_index[v];
```

Toposort

```
// count_indegree
queue<int> q; q.push(i)// indeg==0
while(!q.empty()){
    int t=q.front();q.pop();
    order.push_back(t);
    for(int i: adj[t]){
        indeg[i]--;
        if(indeg[i]==0)
            q.push(i);
    }
}
// reverse_toposort_DFS: post_order
// mini-toposort: pq
```

Minimum spanning tree

```
Prim(稠密图)
pq.push({0,1});// first weight,second vertex
while(!pq.empty()){
    int u = pq.top().second, w = pq.top().first;
    pq.pop();
    if(visited[u]) continue;
    visited[u] = true;
    cost += w;
    for(auto p:adj[u]){
        int v = p.first;
        int weight = p.second;
        if(!visited[v])
            pq.push({weight, v});
    }
}
```

```
Kruskal(稀疏图, union-find set)
int find_root(int a){
    int t = a;
    while(parent[a]!=a)
        a = parent[a];
    parent[t] = a;
    return a;
}parent[i]=i;// initialize
sort(edges,edges+m,[](Edge a, Edge b){return a.w<b.w;});
for(auto e:edges){
    int u=e.u,v=e.v,w=e.w;
    int root_u = find_root(u), root_v = find_root(v);
    if(root_u!=root_v){
        parent[root_u] = root_v;
        parent[u] = root_v;
        cost += w;
    }
}
```

Max flow

```
Edmonds_Karp(稀疏)
int update(int parent[],int t);
int BFS(int s,int t){// add the parent to record  // ...
    if(i==t)
        return update(parent,t); // ...
    return 0;
}
long long int Edmonds_Karp(int s, int t) {
        long long int f = 0;
        while (1) {
                build_res();
                int k = BFS(s, t);
                if (k <= 0) return f;
                f += k;
        }
}
```

```
Dinic(稠密)
int cap[MAX][MAX], level[MAX];
void BFS(int s) {
        memset(level, -1, sizeof(level));
        queue<int> Q;
        Q.push(s);
        level[s] = 0;
        while (!Q.empty()) {
                int v = Q.front();
                Q.pop();
                for (int i = 1; i <= n; i++)
                        if (cap[v][i] > 0 && level[i] < 0) {
                                Q.push(i);
                                level[i] = level[v] + 1;
                        }
        }
}
vector<int> used(MAX, 1);
long long DFS(int s, int t, int limit) {
        if (s == t) return limit;
        for (int& i = used[s]; i <= n; i++)
                if (cap[s][i] > 0 && (level[i] == level[s] + 1)) {
                        int f_limit = DFS(i, t, min(cap[s][i], limit));
                        if (f_limit > 0) {
                                cap[s][i] -= f_limit;
                                cap[i][s] += f_limit;
                                return f_limit;
                        }
                }
        return 0;
}
long long int Dinic(int s, int t) {
        long long f = 0;
        while (1) {
                BFS(s);
                if (level[t] < 0)
                        return f;
                used.assign(MAX, 1);
                long long f1;
                while (1) {
                        f1 = DFS(s, t, INF);
                        if (f1 > 0) f += f1;
                        else break;
                }
        }
}
// Multiple sources and sinks: S,T, inf edge to s,...t,...
// Vertex capacities: Create two vertex (in) and (out), c(e)=c(vertex)
// Maximum Bipartite Matching: s,t, s with group1, group2 with t, c(e)=1
// Maximum weight closure: s,t, s to (pos)-c(e)=pos,(neg) to t-c(e)=-neg, ceiling=sum(pos), ans=ceiling-min_cut
```