Contents

Graphs

Data Structures

Union Find

```
struct UnionFind {
  int n;
  vector<int> dad, size;

  UnionFind(int N) : n(N), dad(N), size(N, 1) {
    while (N--) dad[N] = N;
  }

  // O(lg*(N))
  int root(int u) {
    if (dad[u] == u) return u;
    return dad[u] = root(dad[u]);
  }

  // O(1)
  void join(int u, int v) {
    int Ru = root(u), Rv = root(v);
    if (Ru == Rv) return;
    if (size[Ru] > size[Rv]) swap(Ru, Rv);
    --n, dad[Ru] = Rv;
    size[Rv] += size[Ru];
  }

  // O(lg*(N))
  bool areConnected(int u, int v) {
    return root(u) == root(v);
  }

  int getSize(int u) { return size[root(u)]; }

  int numberOfSets() { return n; }
};
```

Articulation Points And Bridges

```
// APB = articulation points and bridges
// Ap = Articulation Point
// br = bridges, p = parent
// disc = discovery time
// low = lowTime, ch = children
// nup = number of edges from u to p

typedef pair<int, int> Edge;
int Time;
vector<vector<int>> adj;
vector<int> disc, low, isAp;
vector<Edge> br;

void init(int N) { adj.assign(N, vector<int>()); }

void addEdge(int u, int v) {
  adj[u].push_back(v);
  adj[v].push_back(u);
}

int dfsAPB(int u, int p) {
  int ch = 0, nup = 0;
  low[u] = disc[u] = ++Time;
  for (int &v : adj[u]) {
    if (v == p && !nup++) continue;
    if (!disc[v]) {
      ch++, dfsAPB(v, u);
      if (disc[u] <= low[v]) isAp[u]++;
      if (disc[u] < low[v]) br.push_back({u, v});
      low[u] = min(low[u], low[v]);
    } else
      low[u] = min(low[u], disc[v]);
  }
  return ch;
}

// O(N)
void APB() {
  br.clear();
  isAp = low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u]) isAp[u] = dfsAPB(u, u) > 1;
}
```

Topological Sort

```
// vis = visited
vector<vector<int>> adj;
vector<int> vis, toposorted;

void init(int N) {
  adj.assign(N, vector<int>());
  vis.assign(N, 0), toposorted.clear();
}

void addEdge(int u, int v) { adj[u].push_back(v); }

// returns false if there is a cycle
// O(E)
bool toposort(int u) {
  vis[u] = 1;
  for (auto &v : adj[u])
    if (v != u && vis[v] != 2 &&
        (vis[v] || !toposort(v)))
      return false;
  vis[u] = 2;
  toposorted.push_back(u);
  return true;
}

// O(V + E)
bool toposort() {
  for (int u = 0; u < adj.size(); u++)
    if (!vis[u] && !toposort(u)) return false;
  return true;
}
```

Flow

Max Flow Min Cut Edmonds-Karp

```
// cap[a][b] = Capacity left from a to b
// iflow = initial flow, icap = initial capacity
// pathMinCap = capacity bottleneck for a path (s->t)

typedef int T;
vector<int> level;
vector<vector<int>> adj, cap;
T inf = 1 << 30;

void init(int N) {
  adj.assign(N, vector<int>());
  cap.assign(N, vector<int>(N));
}

void addEdge(int u, int v, T icap, T iflow = 0) {
  if (!cap[u][v])
    adj[u].push_back(v), adj[v].push_back(u);
  cap[u][v] = icap - iflow;
  // cap[v][u] = cap[u][v]; // if graph is undirected
}

// O(N)
T bfs(int s, int t, vector<int> &dad) {
  dad.assign(adj.size(), -1);
  queue<pair<int, T>> q;
  dad[s] = s, q.push(s);
  while (q.size()) {
    int u = q.front().first;
    T pathMinCap = q.front().second;
    q.pop();
    for (int v : adj[u])
      if (dad[v] == -1 && cap[u][v]) {
        dad[v] = u;
        T flow = min(pathMinCap, cap[u][v]);
        if (v == t) return flow;
        q.push({v, flow});
      }
  }
  return 0;
}

// O(E^2 * V)
T maxFlowMinCut(int s, int t) {
  T maxFlow = 0;
  vector<int> dad;
  while (T flow = bfs(s, t, dad)) {
    maxFlow += flow;
    int u = t;
    while (u != s) {
      cap[dad[u]][u] -= flow, cap[u][dad[u]] += flow;
      u = dad[u];
    }
  }
  return maxFlow;
}
```

Max Flow Min Cut Dinic

```
// cap[a][b] = Capacity from a to b
// flow[a][b] = flow occupied from a to b
// level[a] = level in graph of node a
// iflow = initial flow, icap = initial capacity
// pathMinCap = capacity bottleneck for a path (s->t)

typedef int T;
vector<int> level;
vector<vector<int>> adj;
vector<vector<T>> cap, flow;
T inf = 1 << 30;

void init(int N) {
  adj.assign(N, vector<int>());
  cap.assign(N, vector<int>(N));
  flow.assign(N, vector<int>(N));
}

void addEdge(int u, int v, T icap, T iflow = 0) {
  if (!cap[u][v])
    adj[u].push_back(v), adj[v].push_back(u);
  cap[u][v] += icap;
  // cap[v][u] = cap[u][v]; // if graph is undirected
  flow[u][v] += iflow, flow[v][u] -= iflow;
}

bool levelGraph(int s, int t) {
  level.assign(adj.size(), 0);
  level[s] = 1;
  queue<int> q;
  q.push(s);
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int &v : adj[u]) {
      if (!level[v] && flow[u][v] < cap[u][v]) {
        q.push(v);
        level[v] = level[u] + 1;
      }
    }
  }
  return level[t];
}

T blockingFlow(int u, int t, T pathMinCap) {
  if (u == t) return pathMinCap;
  for (int v : adj[u]) {
    T capLeft = cap[u][v] - flow[u][v];
    if (level[v] == (level[u] + 1) && capLeft > 0)
      if (T pathMaxFlow = blockingFlow(
          v, t, min(pathMinCap, capLeft))) {
        flow[u][v] += pathMaxFlow;
        flow[v][u] -= pathMaxFlow;
        return pathMaxFlow;
      }
  }
  return 0;
}
```

```
// O(E * V^2)
T maxFlowMinCut(int s, int t) {
  if (s == t) return inf;
  T maxFlow = 0;
  while (levelGraph(s, t))
    while (T flow = blockingFlow(s, t, inf))
      maxFlow += flow;
  return maxFlow;
}
```

Shortest Paths

Dijkstra

```
// s = source
typedef int T;
typedef pair<T, int> DistNode;
T inf = 1 << 30;
vector<vector<int>> adj;
unordered_map<int, unordered_map<int, T>> weight;

void init(int N) {
  adj.assign(N, vector<int>());
  weight.clear();
}

void addEdge(int u, int v, T w, bool isDirected = 0) {
  adj[u].push_back(v);
  weight[u][v] = w;
  if (isDirected) return;
  adj[v].push_back(u);
  weight[v][u] = w;
}

// ~ O(E * lg(V))
vector<T> dijkstra(int s) {
  vector<long long int> dist(adj.size(), inf);
  priority_queue<DistNode> q;
  q.push({0, s}), dist[s] = 0;
  while (q.size()) {
    DistNode top = q.top();
    q.pop();
    int u = top.second;
    if (dist[u] < -top.first) continue;
    for (int &v : adj[u]) {
      T d = dist[u] + weight[u][v];
      if (d < dist[v]) q.push({-(dist[v] = d), v});
    }
  }
  return dist;
}
```

Extras

Maths

Common Sums

$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$	$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$	$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{n} k^4 = \frac{n}{30}(n+1)(2n+1)(3n^2+3n-1)$		$\sum_{k=0}^{n} a^k = \frac{1-a^{n+1}}{1-a}$
$\sum_{k=0}^{n} ka^k = \frac{a[1-(n+1)a^n+na^{n+1}]}{(1-a)^2}$	$\sum_{k=0}^{n} k^2a^k = \frac{a[(1+a)-(n+1)^2a^n+(2n^2+2n-1)a^{n+1}-n^2a^{n+2}]}{(1-a)^3}$	
$\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}, \lvert a\rvert < 1$	$\sum_{k=0}^{\infty} ka^k = \frac{a}{(1-a)^2}, \lvert a\rvert < 1$	$\sum_{k=0}^{\infty} k^2a^k = \frac{a^2+a}{(1-a)^3}, \lvert a\rvert < 1$
$\sum_{k=0}^{\infty} \frac{1}{a^k} = \frac{a}{a-1}, \lvert a\rvert > 1$	$\sum_{k=0}^{\infty} \frac{k}{a^k} = \frac{a}{(a-1)^2}, \lvert a\rvert > 1$	$\sum_{k=0}^{\infty} \frac{k^2}{a^k} = \frac{a^2+a}{(a-1)^3}, \lvert a\rvert > 1$
$\sum_{k=0}^{\infty} \frac{a^k}{k!} = e^a$	$\sum_{k=0}^{n} \binom{n}{k} = 2^n$	$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$

Logarithm Rules

$\log_b(b^k) = k$	$\log_b(1) = 0$	$\log_b(X) = \frac{\log_c(X)}{\log_c(b)}$
$\log_b(X \cdot Y) =$ $\log_b(X) + \log_b(Y)$	$\log_b(\frac{X}{Y}) = \log_b(X) - \log_b(Y)$	$\log_b(X^k) = k \cdot \log_b(X)$