COMP3011 Assignment 1
XIANG Yufan 23096511d

1. (a) $f = \Omega(g)$

   (b) $f = O(g)$

   (c) $f = \Theta(g)$

2. (i) $(1) + (2) \rightarrow (3)$:

   $\because (1) \therefore E \geq n - 1$ (E denotes number of edges of G) (Do I need to prove that? Because it can always find a spanning tree so it's number $\geq n - 1$)

   Based on that, find its spanning tree, and if we need any edge it will contain a cycle, so it's $E = n - 1$

   (ii) $(1) + (3) \rightarrow (2)$

   As with (i), we know that it's a tree, so it does not contain a cycle.

   (iii) $(2) + (3) \rightarrow (1)$

   Suppose it is not connected, so it contains $k \geq 2$ connected components, suppose each contains V_i nodes and E_i edges, we know that $n = \sum V_i = 1 + \sum E_i = n - 1$。 So it must contain a component that $V_i \leq E_i$ (Suppose all $V_i \geq E_i + 1$, equation will never be true) So same as (1), we know that it's a tree and have extra edge, so it contains a cycle.

3. **It must need conditions that all weight ≥ 0**

   We use two times simple BFS (or DFS which is simple traversal) in tree

   The time is $O(n)$。

Algorithm

We define $bfs(u)$ that calculate all the $dis(u, x)$ for any y in $O(n)$ times and return a x that $dis(u, x)$ is biggest.

First we choose any point x (in C++ code it is 1) and find $u = bfs(1)$。

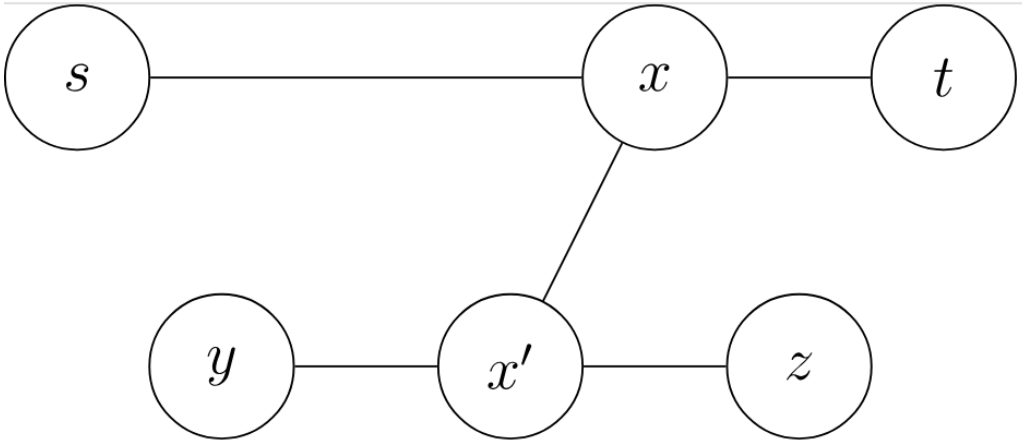
Then we use $v = bfs(u)$。 $path(u, v)$ is the diameter ($dis(u, v)$ is largest)

Prove

Suppose the diameter is (s, t), we only need to prove $z = dfs(y)$ must be s or t. So the next time we can find diameter.

So if $z \neq s$ and $z \neq t$。

1. if (y, z) don't have intersection with (s, t)

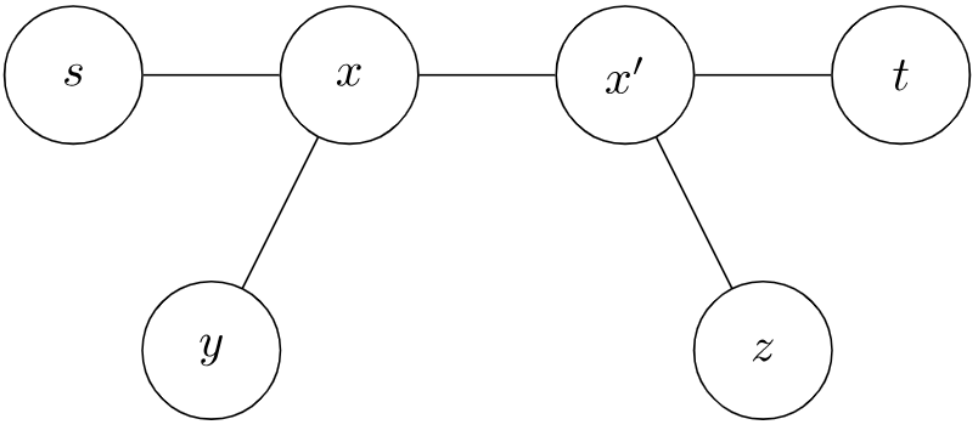


As you can see, x', x is a intersection. We have $d(y,z) = d(y,x') + d(x'z) > d(y,t) = d(y,x') + d(x' + x', t)$。So $d(x',z) > d(x',t)$。

So $d(x',z) + d(x',x) > d(x',z) > d(x,t)$。

Finally we have $d(s,z) > d(s,t)$. It creates contradiction.

2. From y to z， suppose x is the first point that on (s,t)



By symmetry, we can say z lies on the same side at t。So we have $d(x,z) > d(x,t)$

So $d(s,z) > d(s,t)$， it is also a conflict.

Code

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 200005;
// Number of node
int n, q[N], pre[N], a[N], m = 0;
LL dis[N], d[N], val[N];
bool vis[N];
int head[N], numE = 0;
```

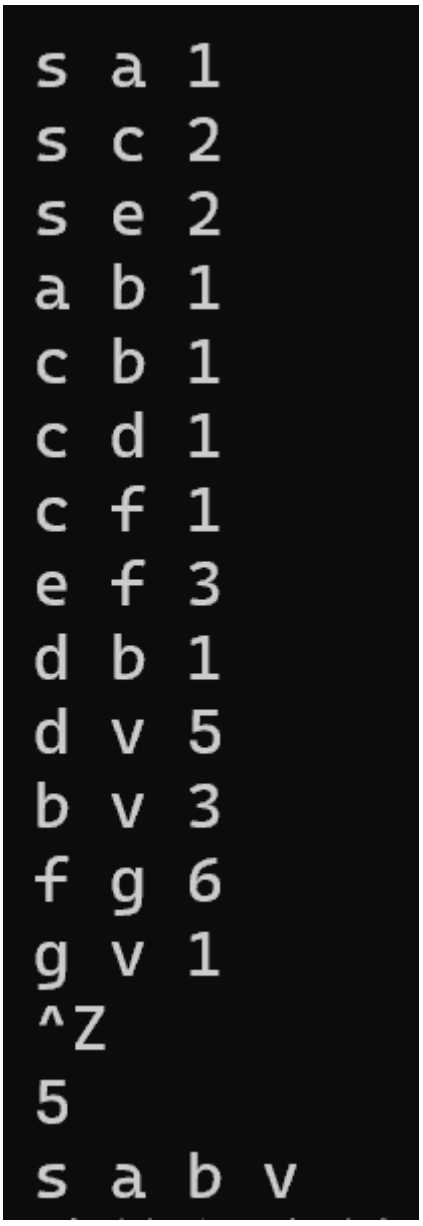
```
struct E{
    int next, v, w;
}e[N << 1];
void inline add(int u, int v, int w) {
    e[++numE] = (E) { head[u], v, w };
    head[u] = numE;
}
void inline bfs(int s) {
    int hh = 0, tt = -1;
    q[++tt] = s;
    dis[s] = 0;
    pre[s] = 0;
    while(hh <= tt) {
        int u = q[hh++];
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v;
            if(v == pre[u]) continue;
            dis[v] = dis[u] + e[i].w;
            pre[v] = u;
            q[++tt] = v;
        }
    }
}
int inline get() {
    int t = 1;
    for (int i = 2; i <= n; i++)
        if(dis[i] > dis[t]) t = i;
    return t;
}
LL inline bfs2(int s) {
    int hh = 0, tt = -1;
    d[s] = 0;
    q[++tt] = s;
    LL res = 0;
    while(hh <= tt) {
        int u = q[hh++];
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v;
            if(vis[v]) continue;
            d[v] = d[u] + e[i].w;
            res = max(res, d[v]);
            vis[v] = true;
            q[++tt] = v;
        }
    }
    return res;
}
int main() {
    scanf("%d", &n);
    for (int i = 1, u, v, w; i < n; i++)
        scanf("%d%d%d", &u, &v, &w), add(u, v, w), add(v, u, w);

    bfs(1);
    int u = get();
    bfs(u);
    int v = get(), p = v;
```

```
    printf("%lld\n", dis[v]);
    return 0;
}
```

4. We use $O(m \log m)$ Dijkstra algorithm program to show the path.

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> PII;
const int N = 100005, INF = 0x3f3f3f3f;
int n, m, dis[N], pre[N];
int head[N], numE = 0, s, t;
bool vis[N];
struct Edge{
    int next, to, dis;
}e[N];
map<char, int> w;

char pos[30];
void addEdge(int from, int to, int dis) {
    e[++numE].next = head[from];
    e[numE].to = to;
    e[numE].dis = dis;
    head[from] = numE;
}
priority_queue<PII, vector<PII>, greater<PII> > q;
int inline dijkstra() {
    memset(dis, 0x3f, sizeof dis);
    dis[s] = 0; q.push(make_pair(0, s));
    while(!q.empty()) {
        PII u = q.top(); q.pop();
        if(vis[u.second]) continue;
        vis[u.second] = true;
        if(u.second == t) return dis[u.second];
        for (int i = head[u.second]; i; i = e[i].next) {
            int v = e[i].to;
            if(dis[u.second] + e[i].dis < dis[v]) {
                dis[v] = dis[u.second] + e[i].dis;
                pre[v] = u.second;
                q.push(make_pair(dis[v], v));
            }
        }
    }
    return -1;
}

int get(char x) {
    if (!w.count(x)) {
        w[x] = ++n;
        pos[n] = x;
    }
    return w[x];
}

int main() {
```

```
    char a, b;
    int c;
    while (cin >> a >> b >> c) {
        int x = get(a), y = get(b);
        addEdge(x, y, c);
    }
    s = get('s');
    t = get('v');

    printf("%d\n", dijkstra());

    vector<int> p;
    int x = t;
    while (x) {
        p.push_back(x);
        x = pre[x];
    }
    reverse(p.begin(), p.end());
    for (int v: p)
        cout << pos[v] << " ";
    return 0;
}
```



The shortest path is $s \to a \to b \to v$。 Length is 5.

5. We define cnt_i as the answer of $w = i$。 At start, $cnt_v = 1$

   We use Dijkstra started from v. And when it comes to using x to extend y with weight w, ((u, v, w)) .

   - If $d_x + w = d_y$ Which means it could be in shortest path from v。 $cnt_y = cnt_y + cnt_x$
   - if $d_x + w < d_y$. Which means previous cnt_y will all be clear. So $cnt_y = cnt_x$。
   - The time complexity is $O(m \log n)$ or $O(m \log m)$ [In this code, I use priority_queue in C++, so it's $O(m \log m)$] (The bottleneck is Dijkstra itself.)

   The following C++ code shows solution of $v = 1$

```cpp
#include <cstdio>
#include <iostream>
#include <queue>
#include <cstring>
using namespace std;
typedef pair<int, int> PII;
const int N = 100005, M = 200005 * 2, P = 100003;
int n, m, dis[N], cnt[N];
bool vis[N];
```

```
priority_queue<PII, vector<PII>, greater<PII>> q;
int head[N], numE = 0;
struct E {
    int next, v, w;
} e[M];
void add(int u, int v, int w) {
    e[++numE] = (E) {
        head[u], v, w
    };
    head[u] = numE;
}
void inline dijkstra() {
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    q.push(make_pair(0, 1));

    while (!q.empty()) {
        PII u = q.top();
        q.pop();

        if (vis[u.second])
            continue;

        vis[u.second] = true;

        for (int i = head[u.second]; i; i = e[i].next) {
            int v = e[i].v;

            if (dis[u.second] + e[i].w < dis[v]) {
                dis[v] = dis[u.second] + e[i].w;
                cnt[v] = cnt[u.second];
                q.push(make_pair(dis[v], v));
            } else if (dis[u.second] + e[i].w == dis[v])
                (cnt[v] += cnt[u.second]) %= P;
        }
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
        add(v, u, w);
    }

    cnt[1] = 1;
    dijkstra();

    for (int i = 1; i <= n; i++)
        printf("%d\n", cnt[i]);
}
```

6. Because he can take any fractions. So we can use greedy. Suppose every item provide w_i, and it's average value is $p_i = v_i/w_i$。 We sort items by p_i in decreasing order and we pick in order until we can't pick anything. The time complexity is $O(n \log n)$, the bottle neck is sorting.
7.

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int N = 505, M = 200005, INF = 0x3f3f3f3f;
int n, m, dis[N];
int head[N], numE = 0;
struct Edge{
    int next, to, dis;
}e[M];
bool vis[N];
void inline addEdge(int from, int to, int dis) {
    e[++numE].next = head[from];
    e[numE].to = to;
    e[numE].dis = dis;
    head[from] = numE;
}
int prim() {
    memset(dis, 0x3f, sizeof dis);
    int res = 0;
    cerr << " First 1\n";
    for (int i = 1; i <= n; i++) {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if(!vis[j] && (t == -1 || dis[j] < dis[t])) t = j;
        vis[t] = true;
        if (t != 1) cerr << t << ":" << "minCost = " << dis[t] << endl;
        if(i > 1 && dis[t] == INF) return -1;
        if(i > 1) res += dis[t];
        for (int j = head[t]; j; j = e[j].next){
            dis[e[j].to] = min(dis[e[j].to], e[j].dis);
        }
    }
    return res;
}
int main() {
    //scanf("%d%d", &n, &m);
    n = 12;
    int u, v, w;
    while (cin >> u >> v >> w) {
        addEdge(u, v, w); addEdge(v, u, w);
    }
    int res = prim();
    cerr << res << endl;

    return 0;
}
```

```
1 2 5
1 5 5
2 3 2
2 5 4
2 7 7
3 4 4
3 7 1
4 7 3
4 8 6
5 6 3
5 9 1
6 7 2
6 10 1
7 8 8
7 10 9
7 11 1
8 11 1
8 12 4
9 10 5
10 11 2
11 12 8

// Output

 First 1
2:minCost = 5
3:minCost = 2
7:minCost = 1
11:minCost = 1
8:minCost = 1
6:minCost = 2
10:minCost = 1
4:minCost = 3
5:minCost = 3
9:minCost = 1
12:minCost = 4
24
```