8th Assignment – Graphs: Maximum flow in transport networks

a) Ford-Fulkerson algorithm

```
template <class T>
void Graph<T>::fordFulkerson(T source, T target) {
    // Obtain the source (s) and target (t) vertices
    Vertex<T>* s = findVertex(source);
    Vertex<T>* t = findVertex(target);
    if (s == NULL || t == NULL || s == t)
        throw "Invalid source and/or target vertex";

    // Apply algorithm as in slides
    resetFlows();
    while( findAugmentationPath(s, t) ) {
        double f = findMinResidualAlongPath(s, t);
        augmentFlowAlongPath(s, t, f);
    }
}

template <class T>
void Graph<T>::resetFlows() {
    for (auto v : vertexSet)
        for (auto e: v->outgoing)
            e->flow = 0;
}

template<class T>
bool Graph<T>::findAugmentationPath(Vertex<T> *s, Vertex<T> *t) {
    for(auto v : vertexSet)
        v->visited = false;
    s->visited = true;
    std::queue< Vertex<T>* > q;
    q.push(s);
    while( ! q.empty() && ! t->visited) {
        auto v = q.front();
        q.pop();
        for(auto e: v->outgoing)
            testAndVisit(q, e, e->dest, e->capacity - e->flow);
        for(auto e: v->incoming)
            testAndVisit(q, e, e->orig, e->flow);
    }
```

```
    return t->visited;
}

template<class T>
void Graph<T>::testAndVisit(std::queue< Vertex<T>*> &q, Edge<T>
*e, Vertex<T> *w, double residual) {
    if (! w-> visited && residual > 0) {
        w->visited = true;
        w->path = e;
        q.push(w);
    }
}

template<class T>
double Graph<T>::findMinResidualAlongPath(Vertex<T> *s, Vertex<T>
*t) {
    double f = INF;
    for (auto v = t; v != s; ) {
        auto e = v->path;
        if (e->dest == v) {
            f = std::min(f, e->capacity - e->flow);
            v = e->orig;
        }
        else {
            f = std::min(f, e->flow);
            v = e->dest;
        }
    }
    return f;
}

template<class T>
void Graph<T>::augmentFlowAlongPath(Vertex<T> *s, Vertex<T> *t,
double f) {
    for (auto v = t; v != s; ) {
        auto e = v->path;
        if (e->dest == v) {
            e->flow += f;
            v = e->orig;
        }
        else {
            e->flow -= f;
```

```
            v = e->dest;
        }
    }
}
```