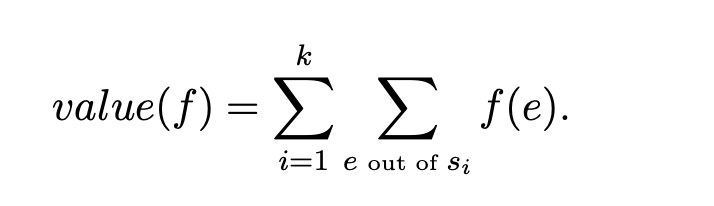
1. Consider a generalization of the max-flow problem with multiple sources and multiplesinks. Formally, we have a directed graph *G*(*V,E*) with *k* source vertices: *s*1*,...,sk*; and *`* sink vertices: *t*1*,...,t`*. Assume that no source is also a sink, and that sources have no incoming edges and sinks have no out going edges. A feasible flow obeys capacity constraints on each edge *e* ∈ *E*, and flow conservation (flow in = flow out) for all vertices that are not a source or a sink. The objective is to maximize the *total* flow leaving the sources:

*.*



Show that the multiple sources/sinks max-flow problem reduces to the single source/sink problem. That is, show how to convert a multiple source/sink instance into a single source/sink instance such that given a maximum flow in the single source/sink instance we can recover the a maximum flow in the original multi source/sink instance. [Hint: you might need to add some vertices and edges]

**Sol: Problem Statement**:

* + We have a directed graph **G(V, E)** with **k source vertices**: (s\_1, s\_2, \ldots, s\_k), and **t sink vertices**: (t\_1, t\_2, \ldots, t\_t).
  + Assumptions:
    - No source is also a sink.
    - Sources have no incoming edges.
    - Sinks have no outgoing edges.
  + Constraints:
    - Feasible flow obeys capacity constraints on each edge (e \in E).
    - Flow conservation (flow in = flow out) holds for all vertices that are not a source or a sink.
  + Objective:
    - Maximize the total flow leaving the sources.

1. **Reduction Approach**:
   * To convert the multiple source/sink instance into a single source/sink instance, we’ll add some vertices and edges to the graph.
   * We’ll create a **super-source** and a **super-sink**:
     + The super-source will be connected to all original sources with edges of infinite capacity.
     + The super-sink will be connected to all original sinks with edges of infinite capacity.
   * Additionally, we’ll add a new edge from the super-source to the super-sink with infinite capacity.
2. **Illustration**:

Suppose we have the following original graph:  
s1 s2 ... sk

\ | /

\ | /

\ | /

v v

t1 t2 ... tt

After adding the super-source and super-sink, the modified graph becomes:  
s1 s2 ... sk

\ | /

\ | /

\ | /

v v

super-source -> super-sink

^ ^

/ | \

/ | \

/ | \

t1 t2 ... tt

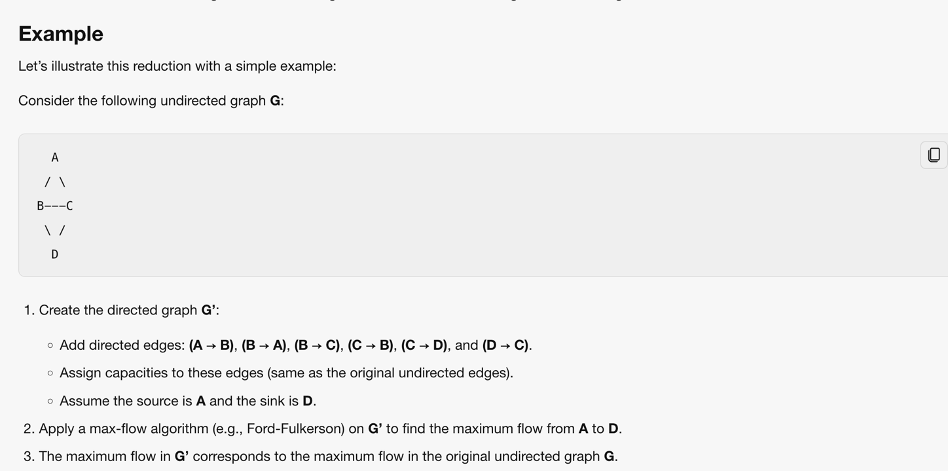
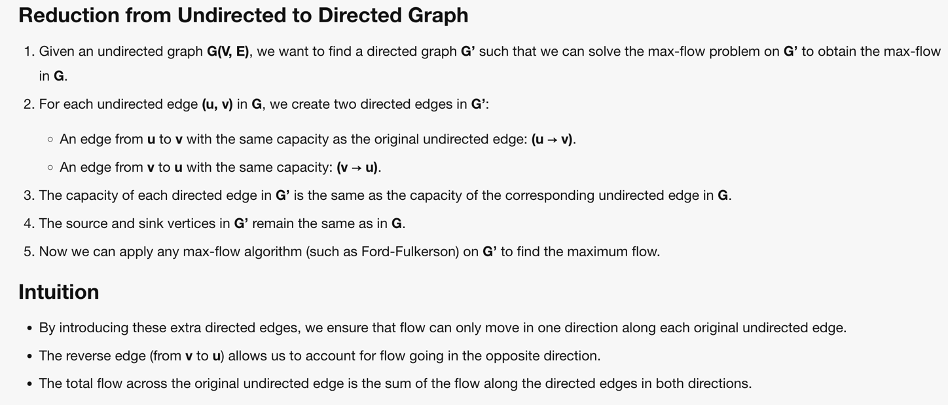
1. **Max-Flow Algorithm**:
   * Apply any max-flow algorithm (e.g., Ford-Fulkerson) to find the maximum flow from the super-source to the super-sink.
   * The flow on the edge connecting the super-source and super-sink represents the maximum flow leaving the original sources.
2. **Recovering Maximum Flow**:
   * Subtract the flow on the super-source to super-sink edge from the total flow leaving the super-source.
   * This gives us the maximum flow leaving the original sources.

In summary, by introducing the super-source, super-sink, and the connecting edge, we transform the multiple source/sink problem into a single source/sink problem. Solving the latter allows us to recover the maximum flow in the original multi source/sink instance.

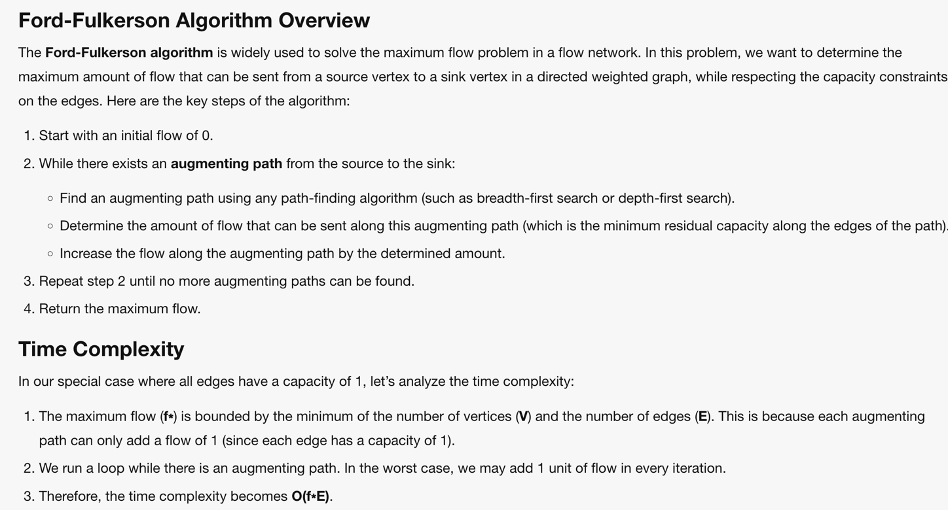
2. The max-flow problem generalizes to undirected graphs *G*(*V,E*) too. The problem setup is essentially the same. Flows are still *directed*. For each edge (*u,v*) the flow assigns two values: *f*(*u* → *v*) for flow going from *u* to *v*, and *f*(*v* → *u*) for flow going the opposite direction from *v* to *u*. Flow conservation (flow in = flow out) is defined as before. Capacity constraints require that the total amount of flow *f*(*u* → *v*) + *f*(*v* → *u*) crossing edge (*u,v*) is no more than *cu,v*. The objective is maximize the total flow leaving the source

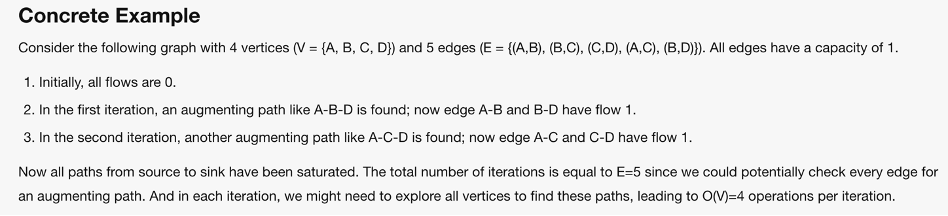
*.*

Show that the max-flow problem in an undirected graph reduces to the max-flow problem on a directed graph. That is, given a undirected graph show how to produce the directed problem instance such that we can use a max-flow in the directed instance to recover a max-flow in the undirected graph. [Hint: you might want to use extra edges.]

Sol; 

3. Suppose that all edges have capacity 1. Show that in this special case, the Ford-Fulkerson algorithm runs in *O*(*V E*). [Hint: can you bound *f*∗ in terms of *m* and *n*? You can assume that there is at most one edge connecting any two vertices.]

Sol: 



4. Suppose we are given any flow network *G* with a source *s*, a sink *t*, and positive integer capacities on each edge *e*. Let (*A,B*) be a minimum *s*-*t* cut with respect to these capacities. If we add 1 to every capacity (*c*(*e*) becomes *c*(*e*) + 1), is (*A,B*) still a min cut.

Sol: 