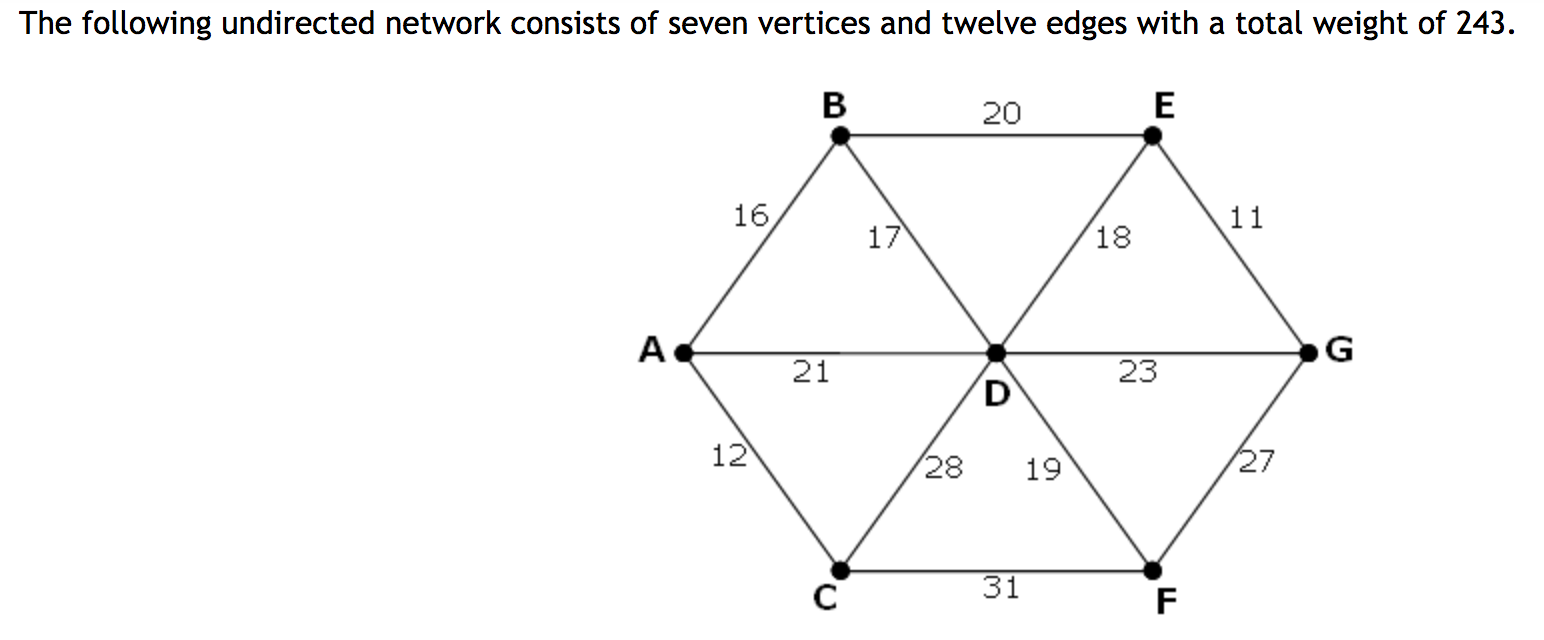
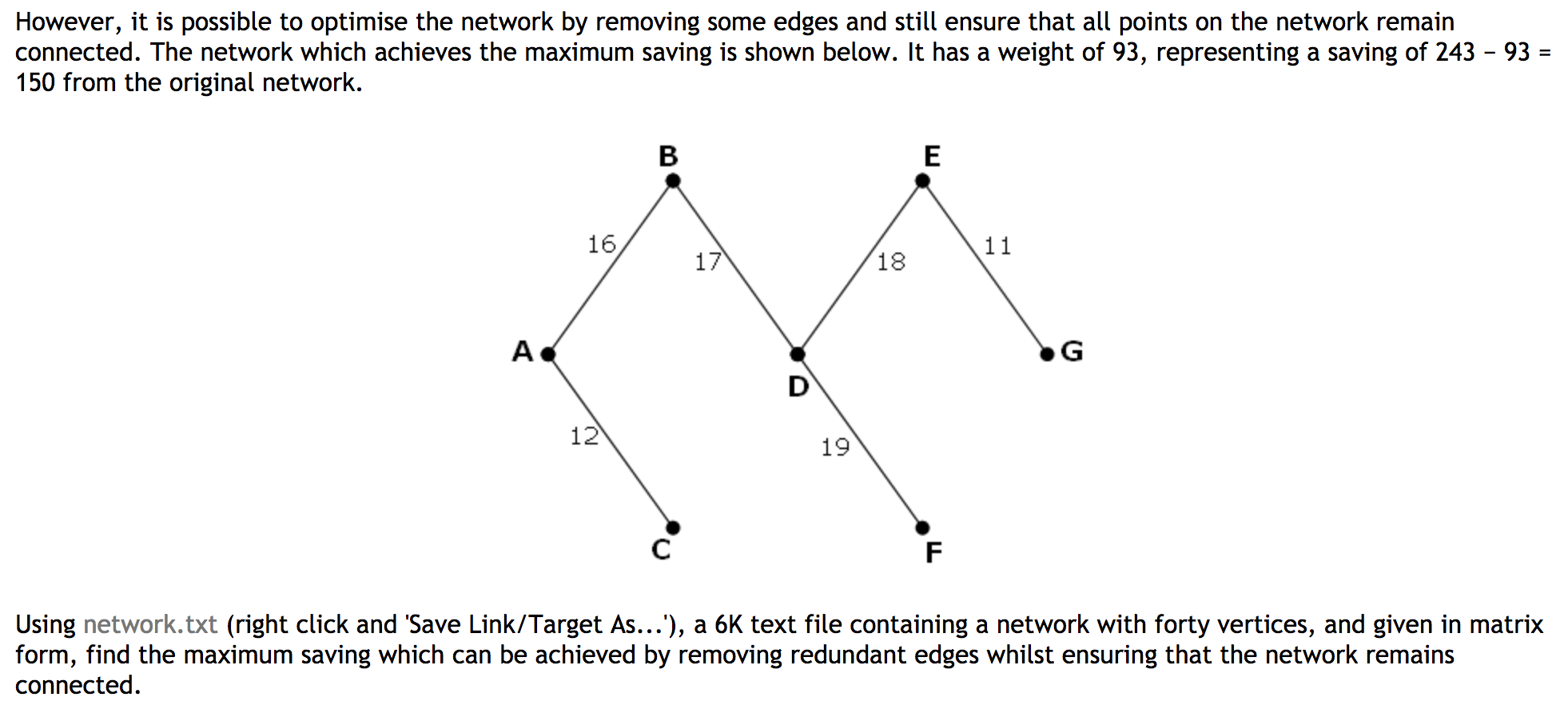
­ 



We develop such algorithm.

Define each iteration: *t*. Then for each *t*, we have a set of visited points *Vt*, and a set of unvisited (but known) nodes *Nt* and a set of unknown nodes *Ut*. Define edge *Et* be a function such that *Et: Vt→Nt*. Thus Ut are those nodes without corresponding function *E* yet.

From such definition we can derive that, for every iteration *t*, if we pick an edge *E*, we will increase *#Vt* by one (so *#Vt* = *t*), and remove a node nt in *Nt*. We also get a set of function E(nt\_i) → Ut. So #Nt will increase E(nt\_i) – 1.

If we assume graph is a **complete graph**, then every node ng in such graph will have at least one edge to other nodes. So *ng ∈ Nt* or *ng ∈ Vt* that *E: Nt* maps to, or *E: Ut* maps to.

We pick the minimum *Et* in each iteration, from the assumption of complete graph, we are guranteed to visit the whole graph, since each iteration is visiting some unknown Nodes *Nt*. If we consider the whole set *Vt* as a single Node *v*, we can see that travesal within v makes no contributes to the total time.

For a single node nt ∈ *Nt*, let’s assume there are multiple E(vi)=nt, and we pick all other *Et+i*, after t th iteration the minimum value. We end up with a time T’ > Tmin if we pick minimum the t th iteration.

Thus we prove that this alrotihm works (traversal all nodes, have minimum weight).