CSC 226

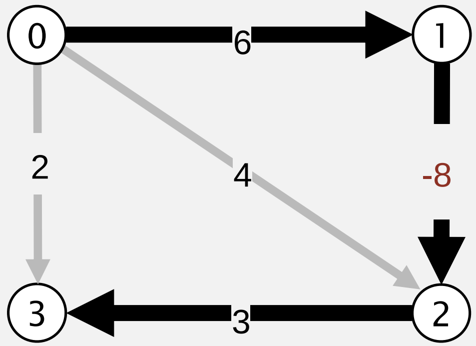
Algorithms and Data Structures: II

Rich Little
rlittle@uvic.ca
ECS 516

Question

- When solving the single source shortest path problem using Dijkstra’s algorithm for an undirected graph with positive edge weights, the paths found result in a spanning tree T.
- Is T a minimum spanning tree for G?

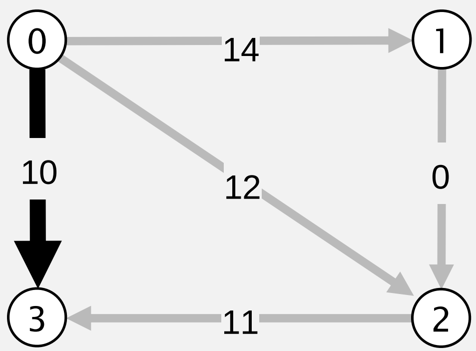
Dijkstra. Doesn't work with negative edge weights.



Dijkstra selects vertex 3 immediately after 0.

But shortest path from 0 to 3 is 0 → 1 → 2 → 3.

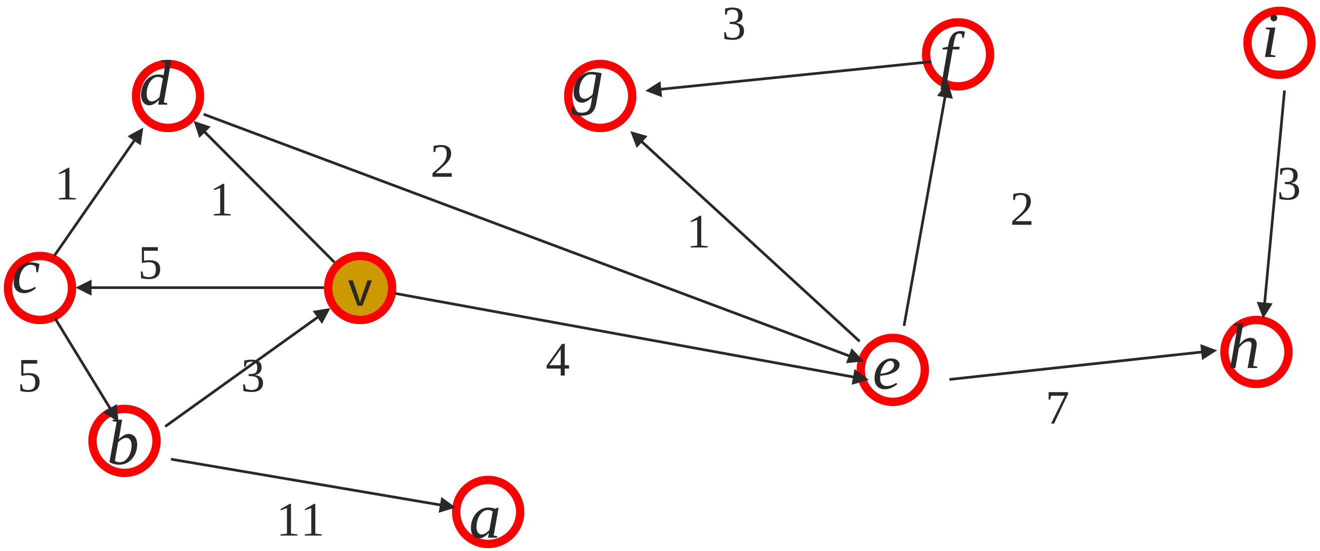
Re-weighting. Add a constant to every edge weight doesn't work.



Adding 8 to each edge weight changes the shortest path from 0 → 1 → 2 → 3 to 0 → 3.

Conclusion. Need a different algorithm.

Directed graphs with positive edge weights



Does Dijkstra’s algorithm work?

Single source shortest paths for *directed* graphs with positive edge weights

- Dijkstra's algorithm works without changes except here edges are directed, that is $(a,b) \neq (b,a)$

- The big-oh worst case running time remains the same

Dijkstra's algorithm: Java implementation

```
public class DijkstraSP
{
  private DirectedEdge[] edgeTo;
  private double[] distTo;
  private IndexMinPQ<Double> pq;

  public DijkstraSP(EdgeWeightedDigraph G, int s)
  {
    edgeTo = new DirectedEdge[G.V()];
    distTo = new double[G.V()];
    pq = new IndexMinPQ<Double>(G.V());

    for (int v = 0; v < G.V(); v++)
      distTo[v] = Double.POSITIVE_INFINITY;
    distTo[s] = 0.0;

    pq.insert(s, 0.0);
    while (!pq.isEmpty())
    {
       int v = pq.delMin();
       for (DirectedEdge e : G.adj(v))
         relax(e);
    }
```

relax vertices in order of distance from s

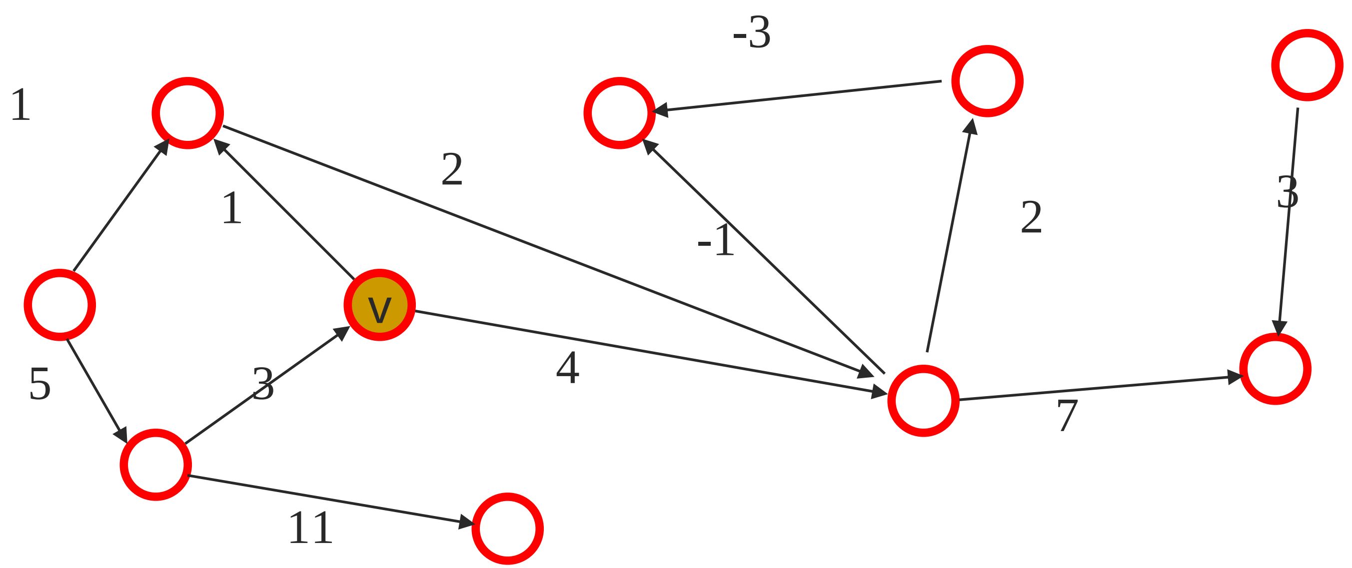
Dijkstra's algorithm: Java implementation

```
private void relax(DirectedEdge e)
{
   int v = e.from(), w = e.to();
   if (distTo[w] > distTo[v] + e.weight())
   {
      distTo[w] = distTo[v] + e.weight();
      edgeTo[w] = e;
      if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
      else                pq.insert (w, distTo[w]);
   }
}
```

update PQ

Shortest paths in graphs containing *negative* edges

- Not possible for undirected graphs
- What about directed graphs?



Negative edges and negative-weight cycles

- If G is directed, compute single-source shortest path problem using Bellman-Ford shortest path algorithm
- Negative-weight cycles are discovered

Algorithm `Bellman-Ford(G,v)`

Input: A simple undirected graph G with nonnegative edge-weights, a distinguished vertex v in G

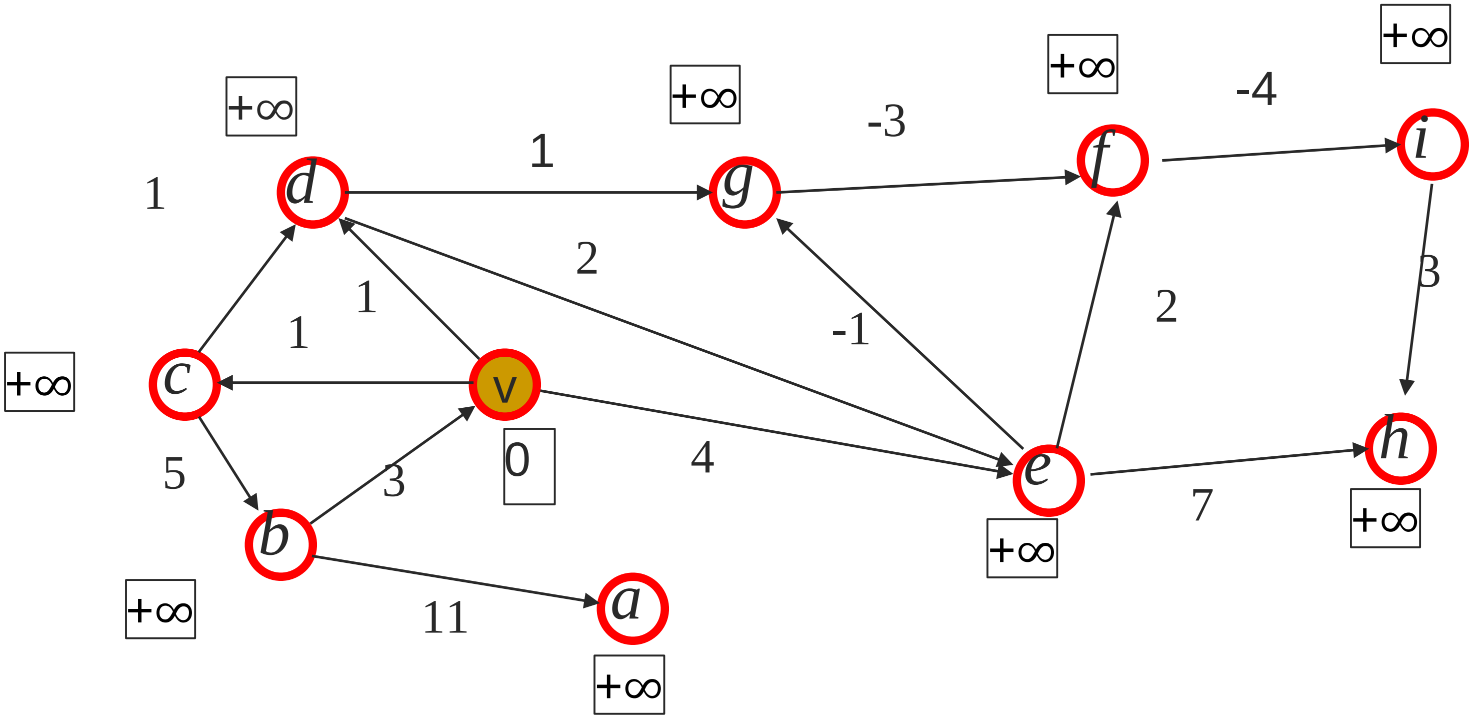
Output: A label D[u] for each vertex u in G such that D[u] is the distance from v to u in G.

Algorithm Bellman-Ford(G,v)

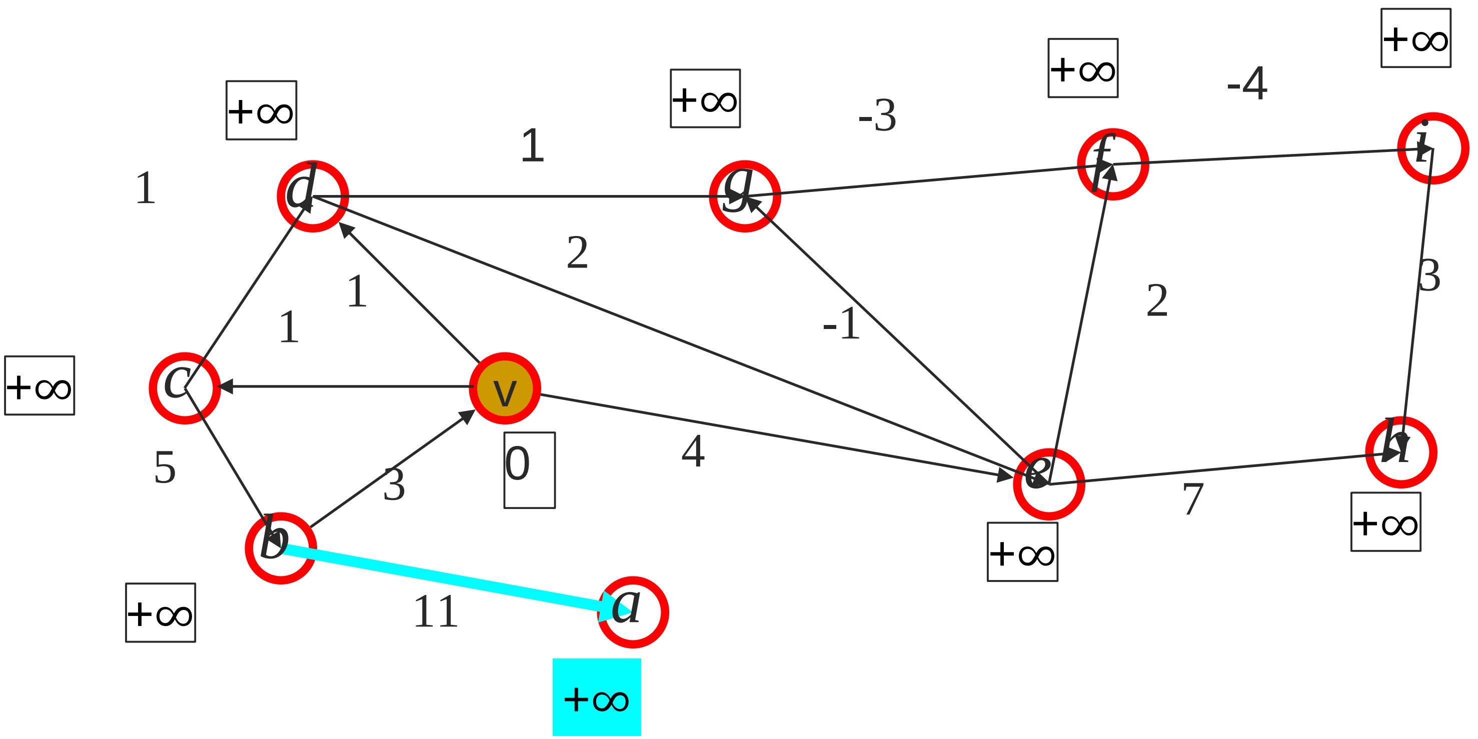
```
D[v]←0
for each vertex u ≠ v of G do
   D[u]← +∞
for i ←1 to n-1 do
   for each edge (u,z) in G do
      if D[u]+w((u,z)) < D[z] then
            D[z]←D[u]+w((u,z))
if there are no edges left with potential
   relaxation operations then
   return D
else
   return "G contains a negative cycle"
```

performs n-1 times a relaxation of every edge in the graph

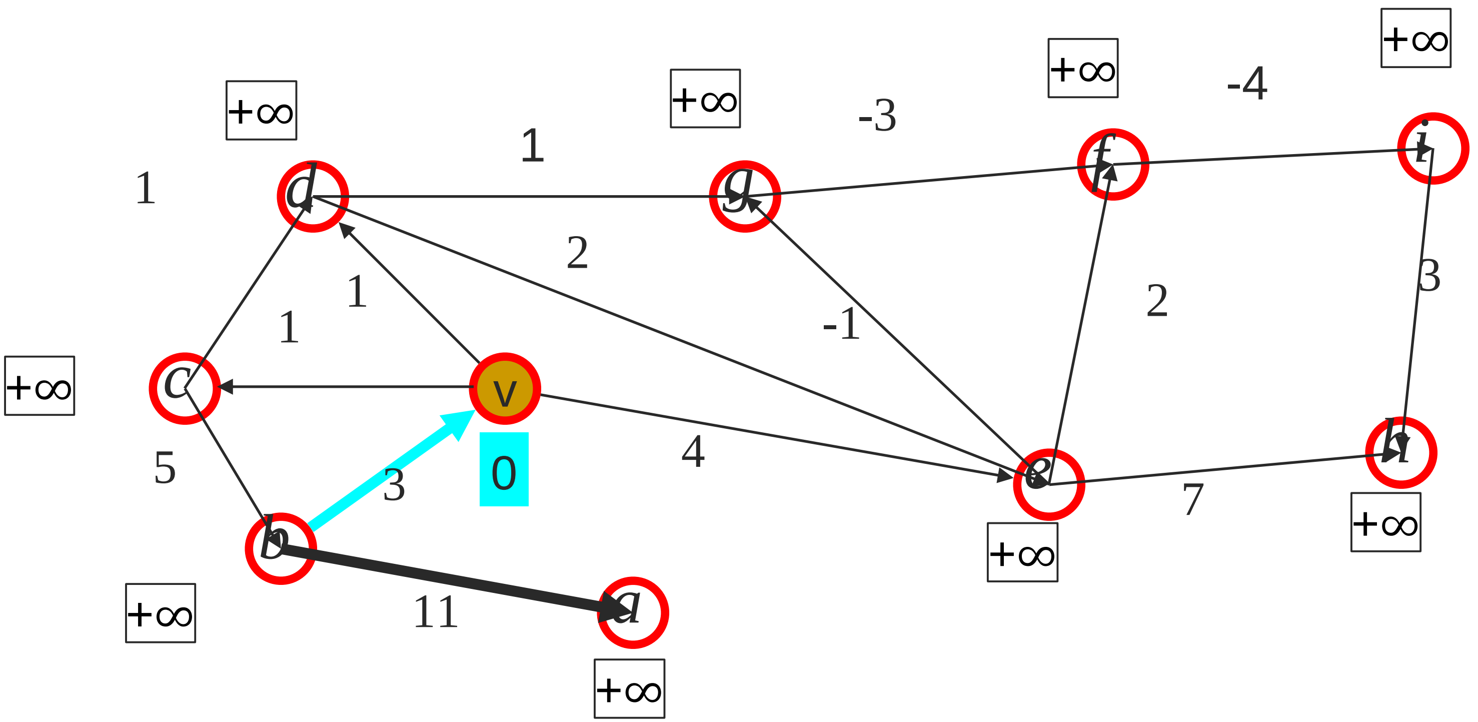
Initialize …



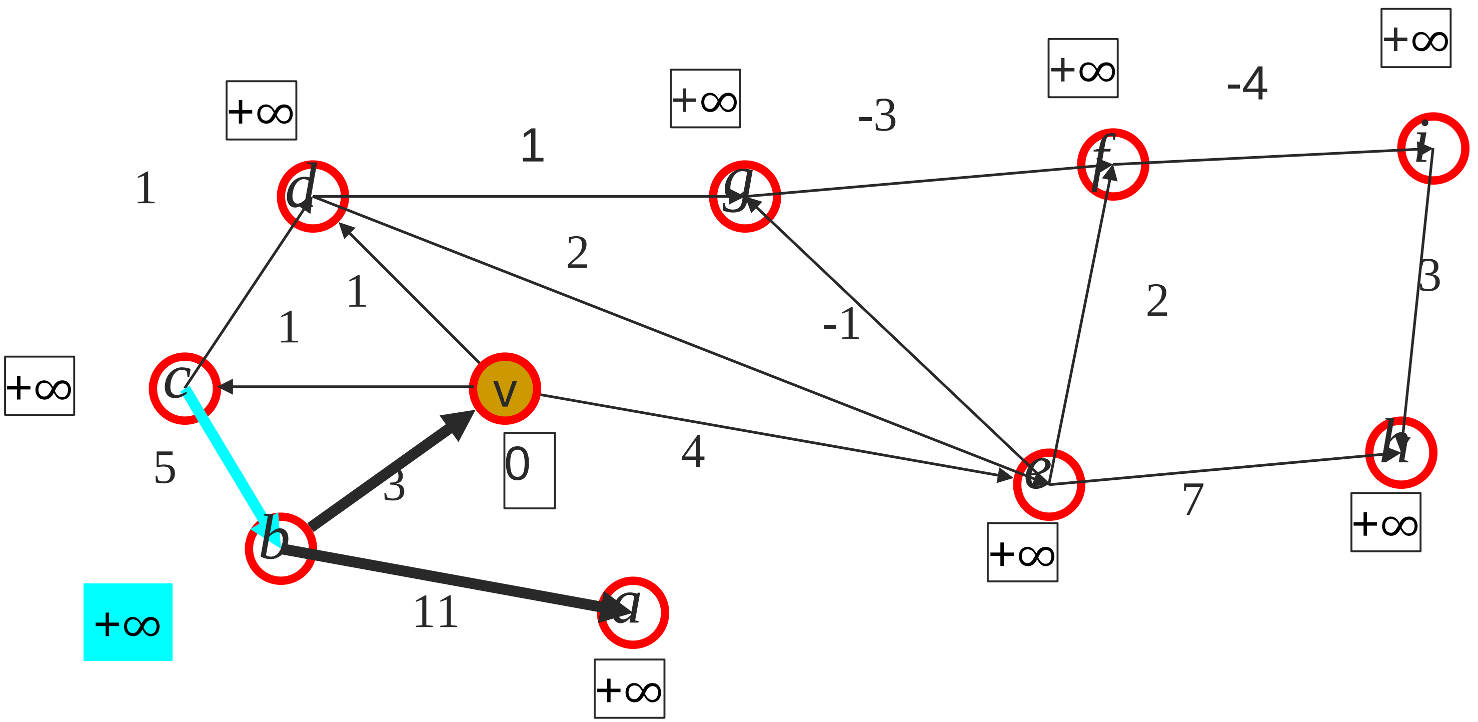
i=1



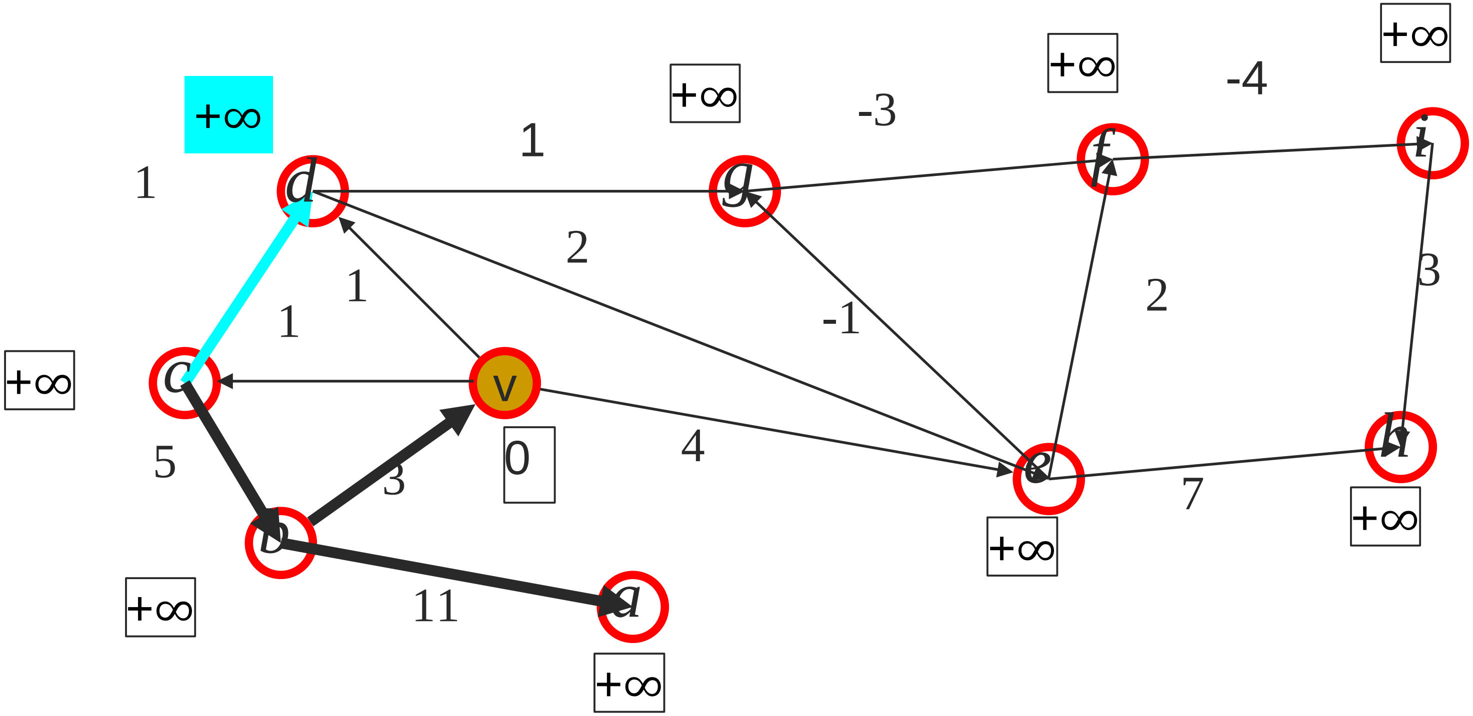
i=1



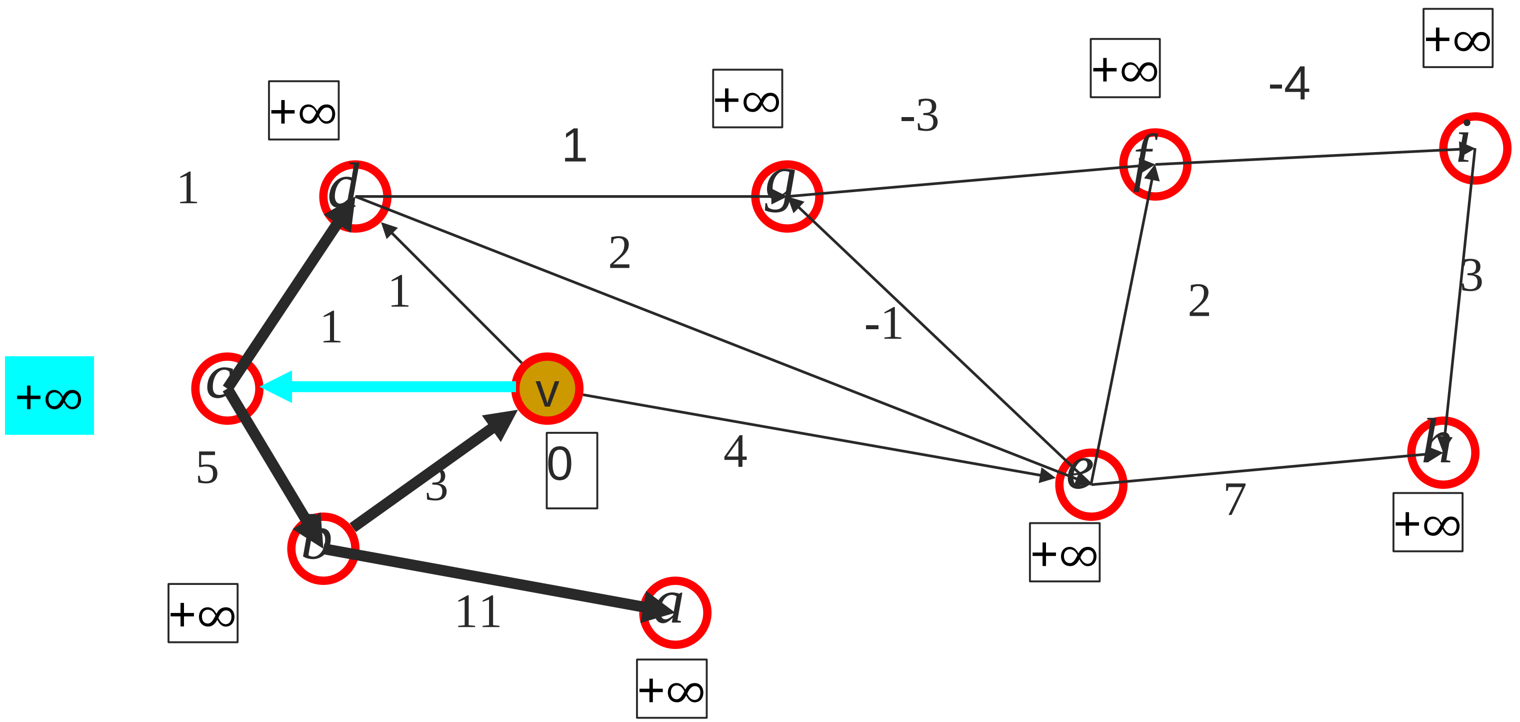
i=1



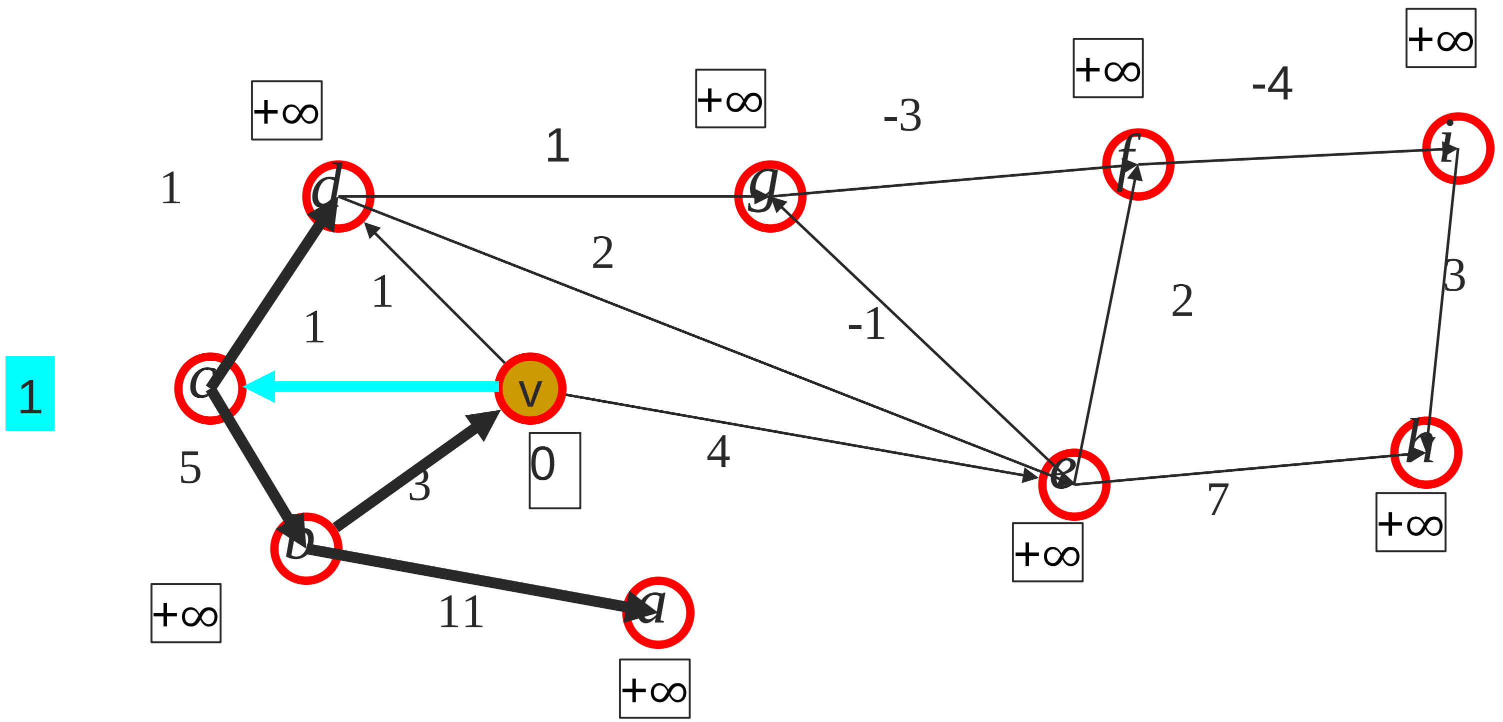
i=1



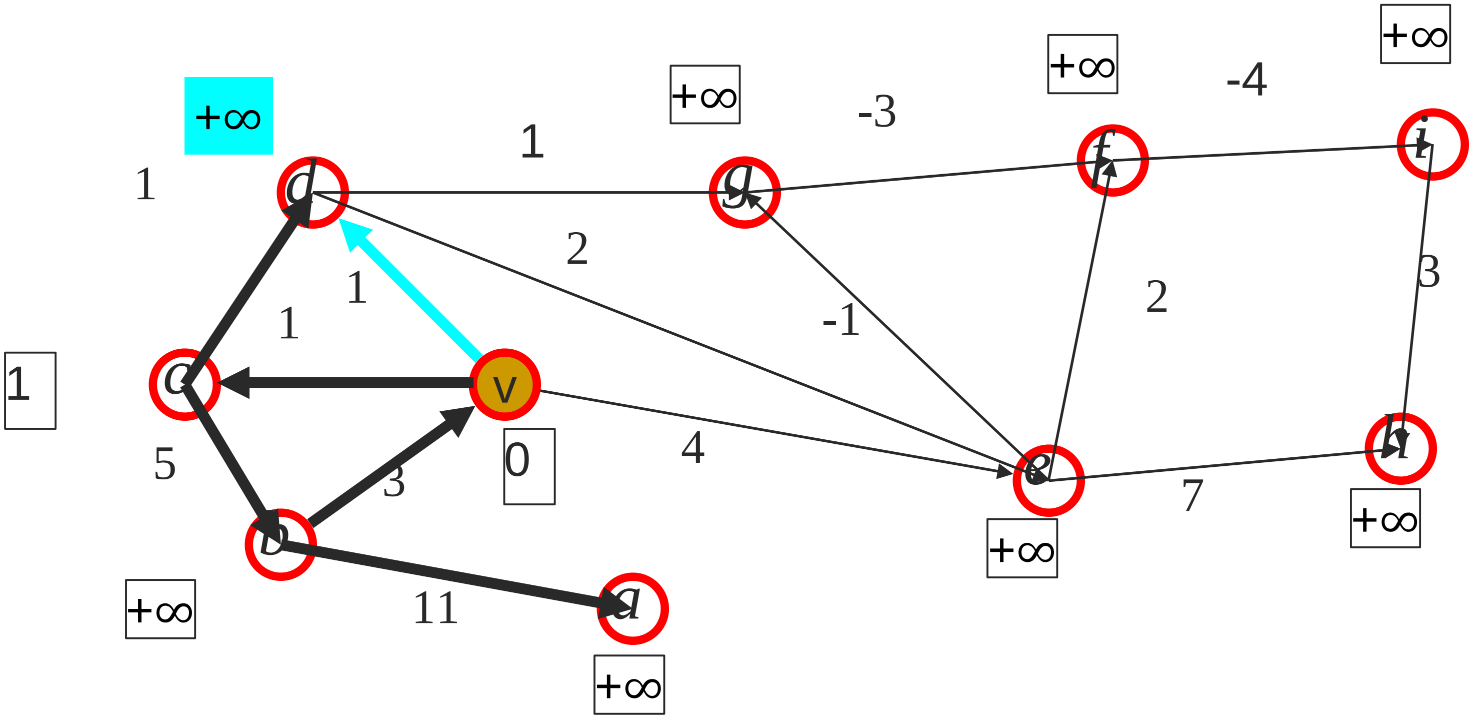
i=1



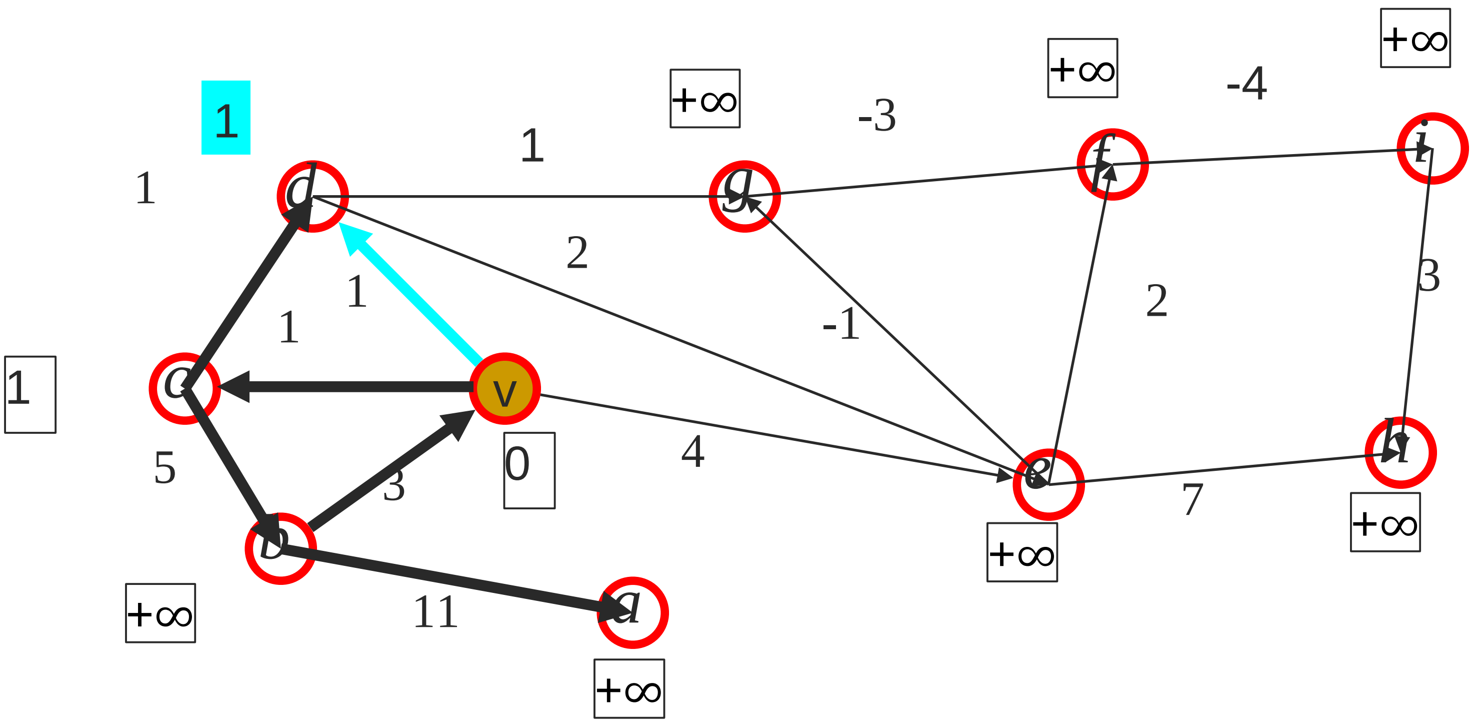
i=1



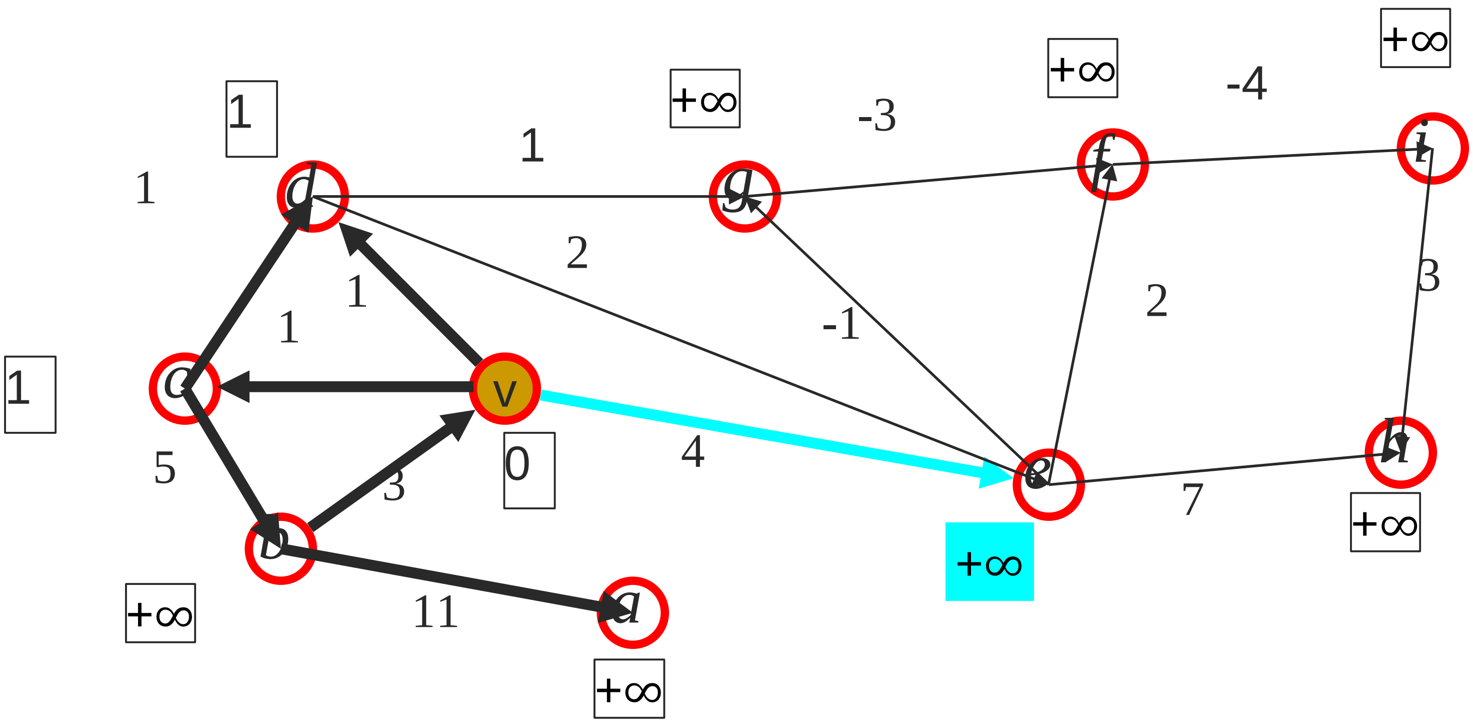
i=1



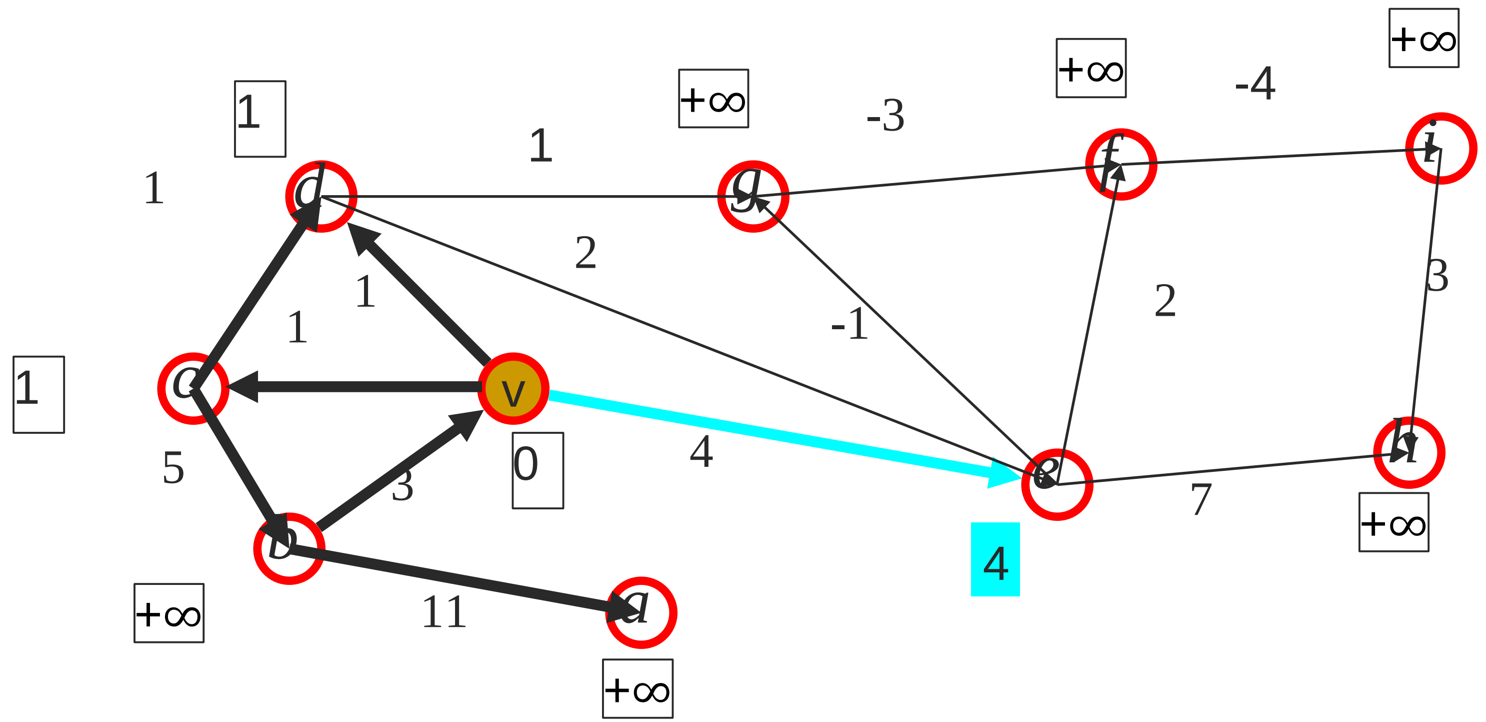
i=1



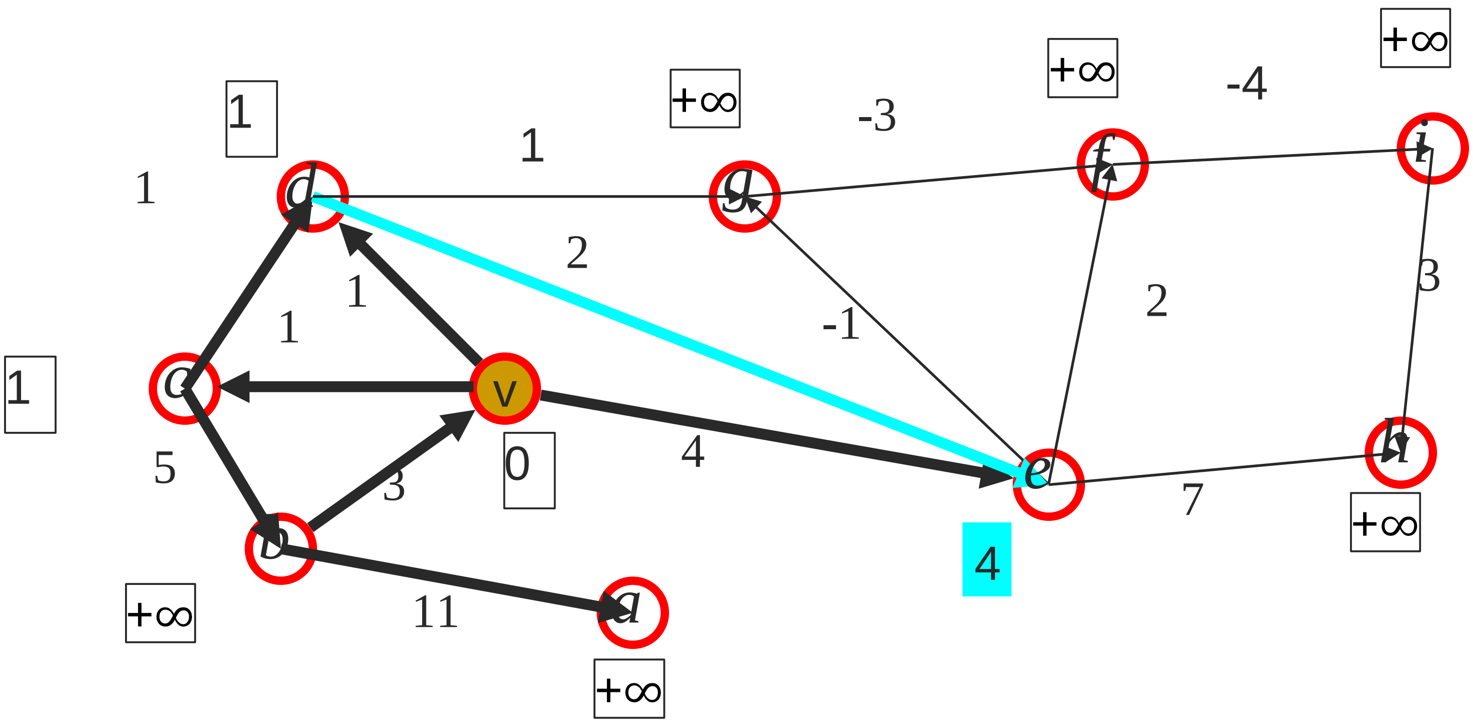
i=1



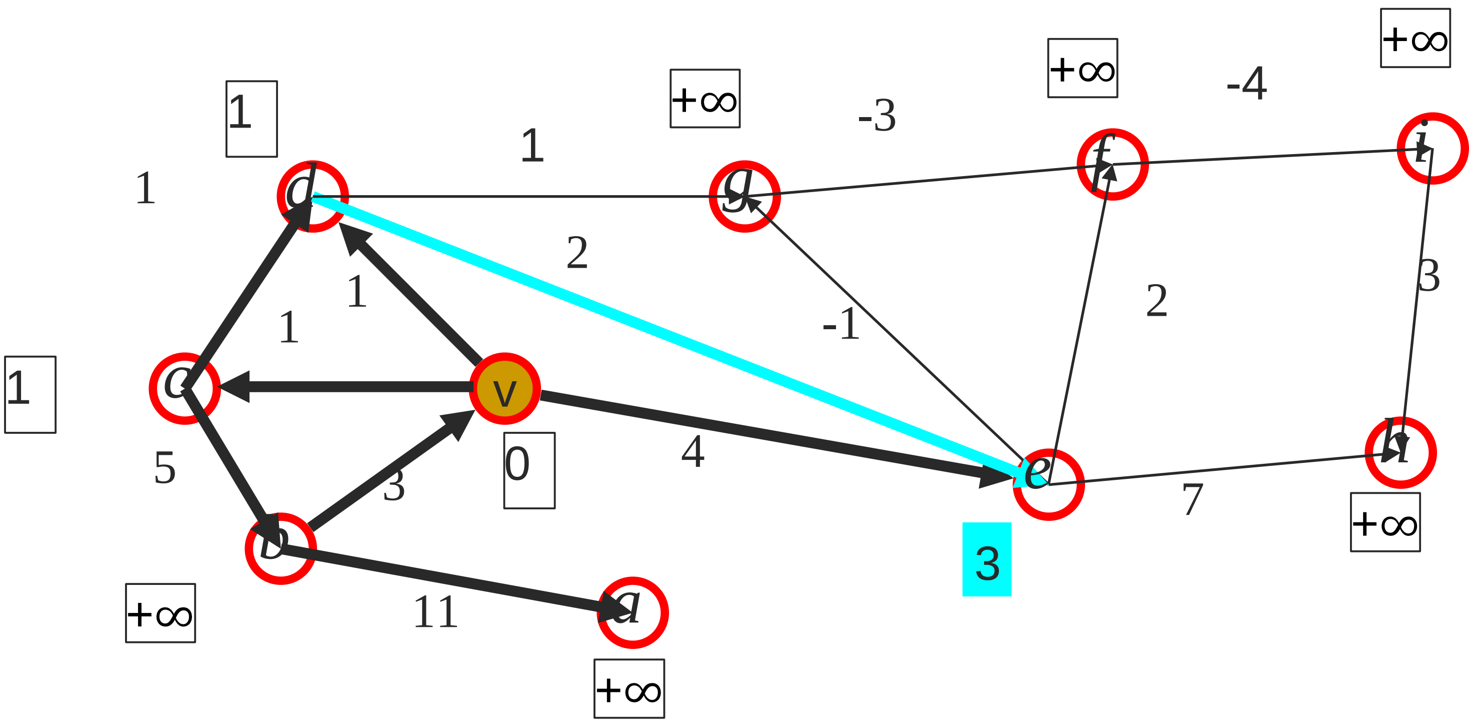
i=1



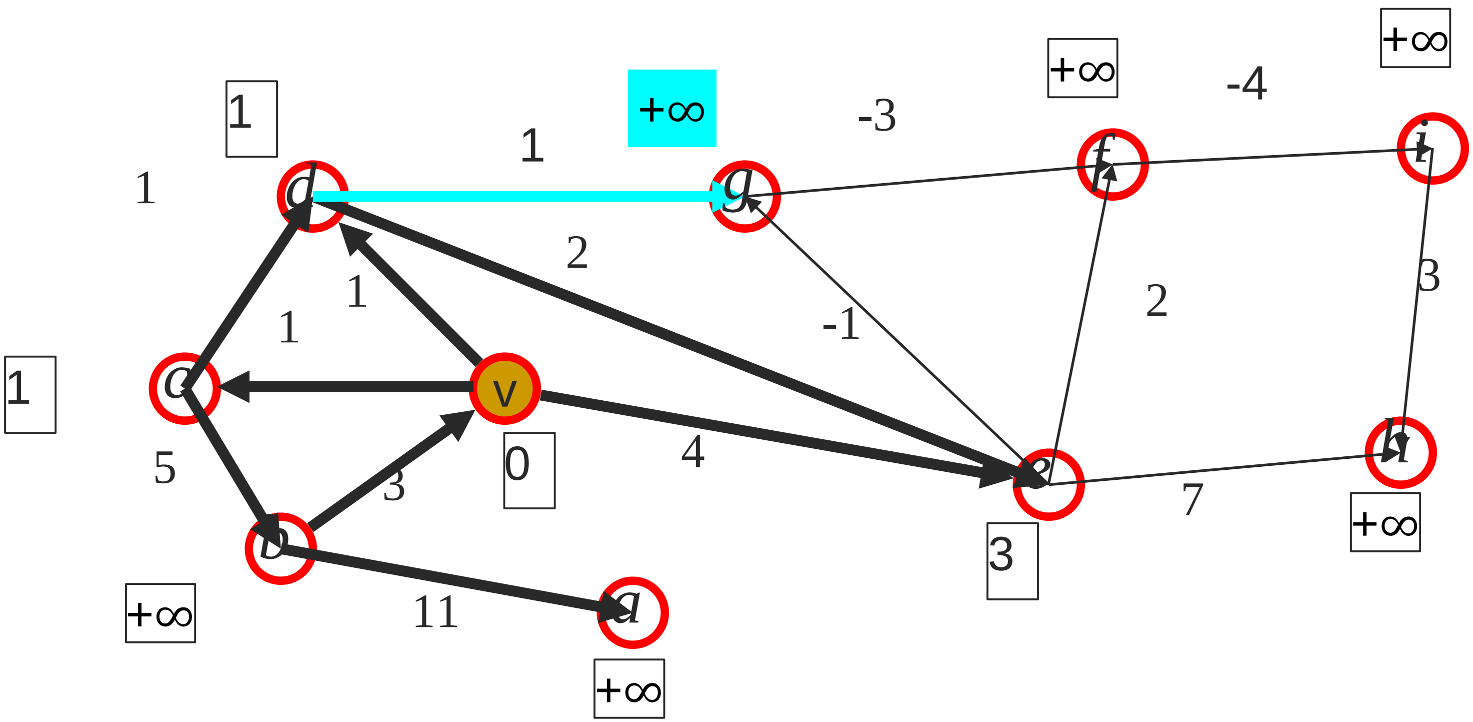
i=1



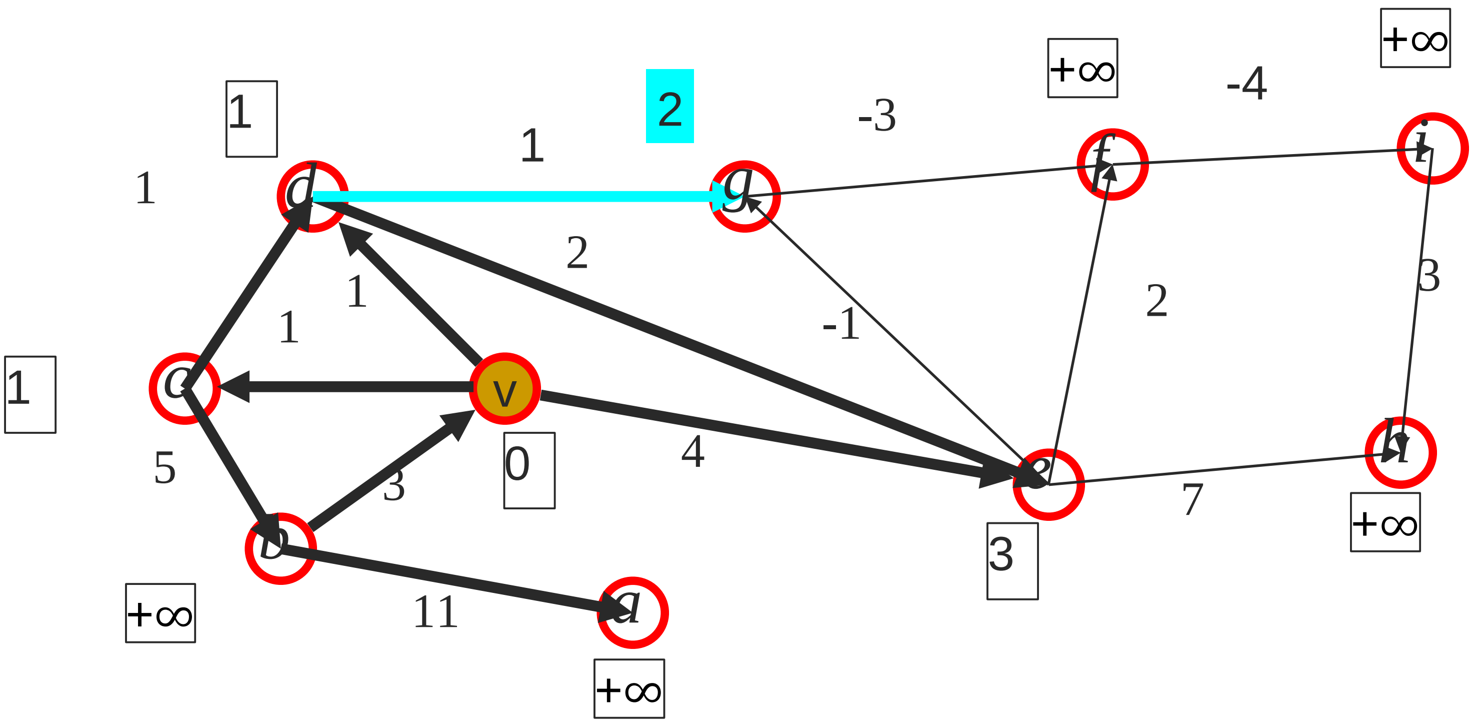
i=1



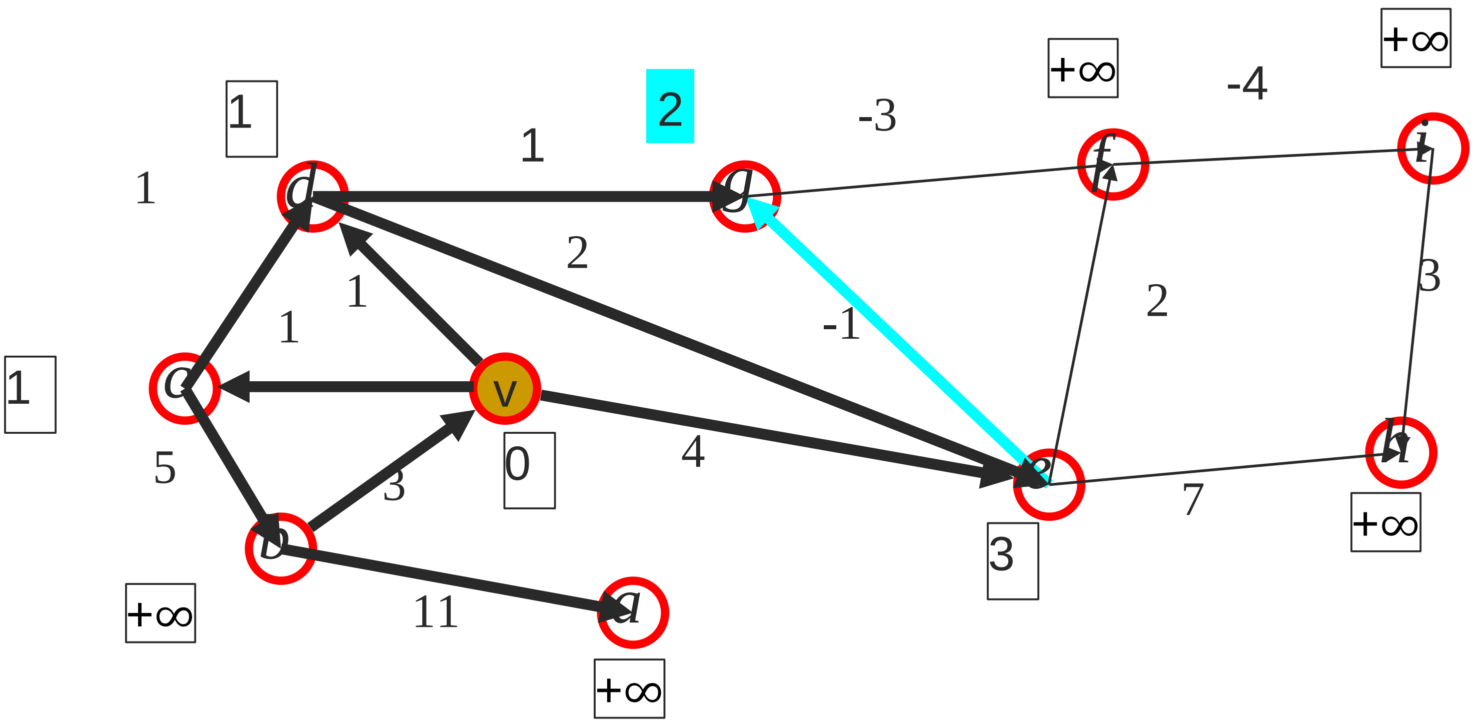
i=1



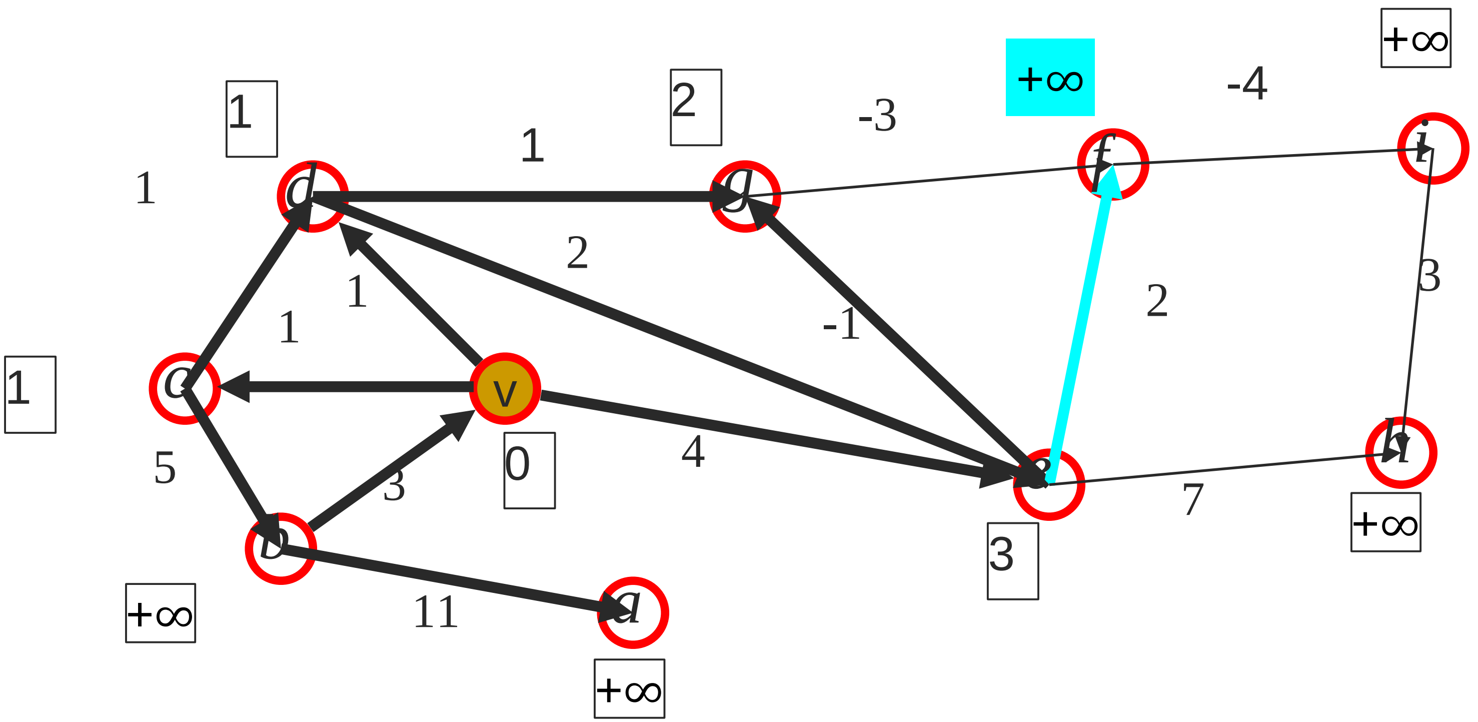
i=1



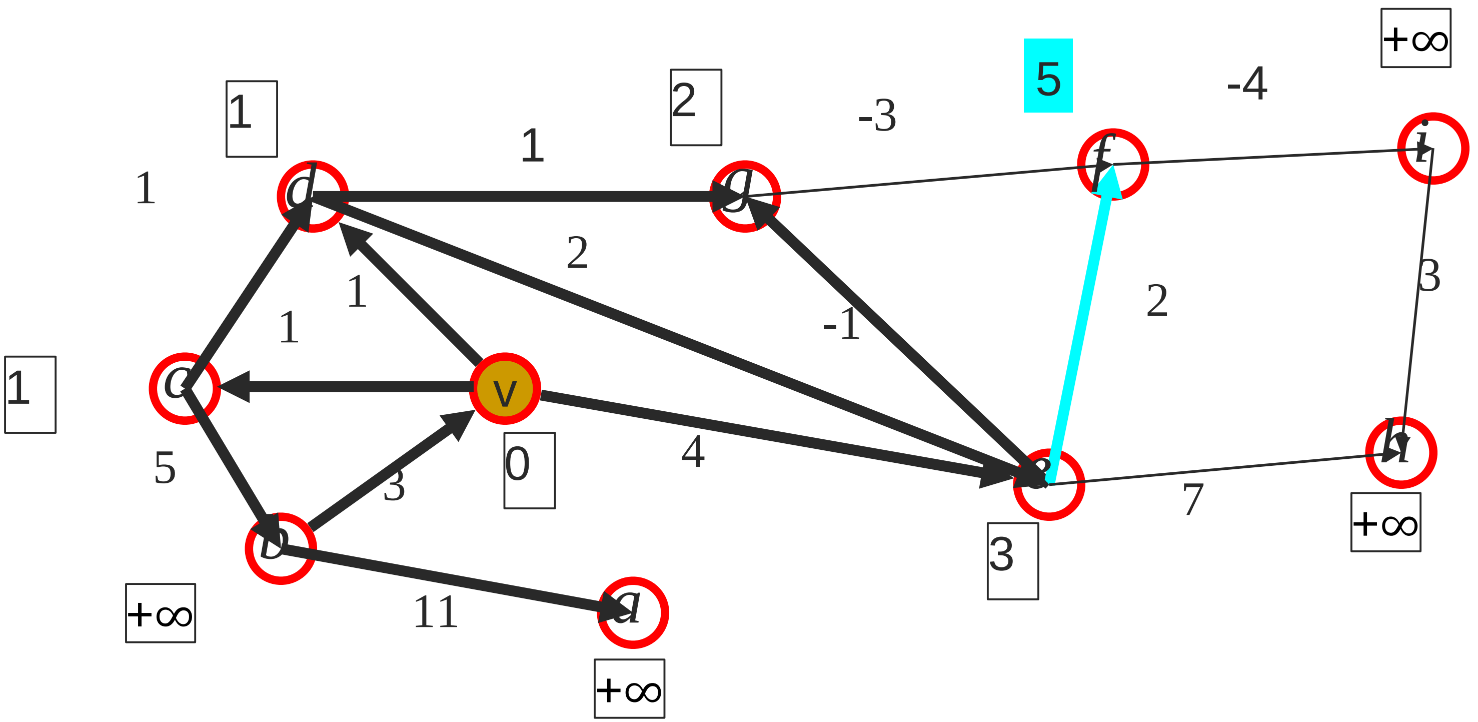
i=1



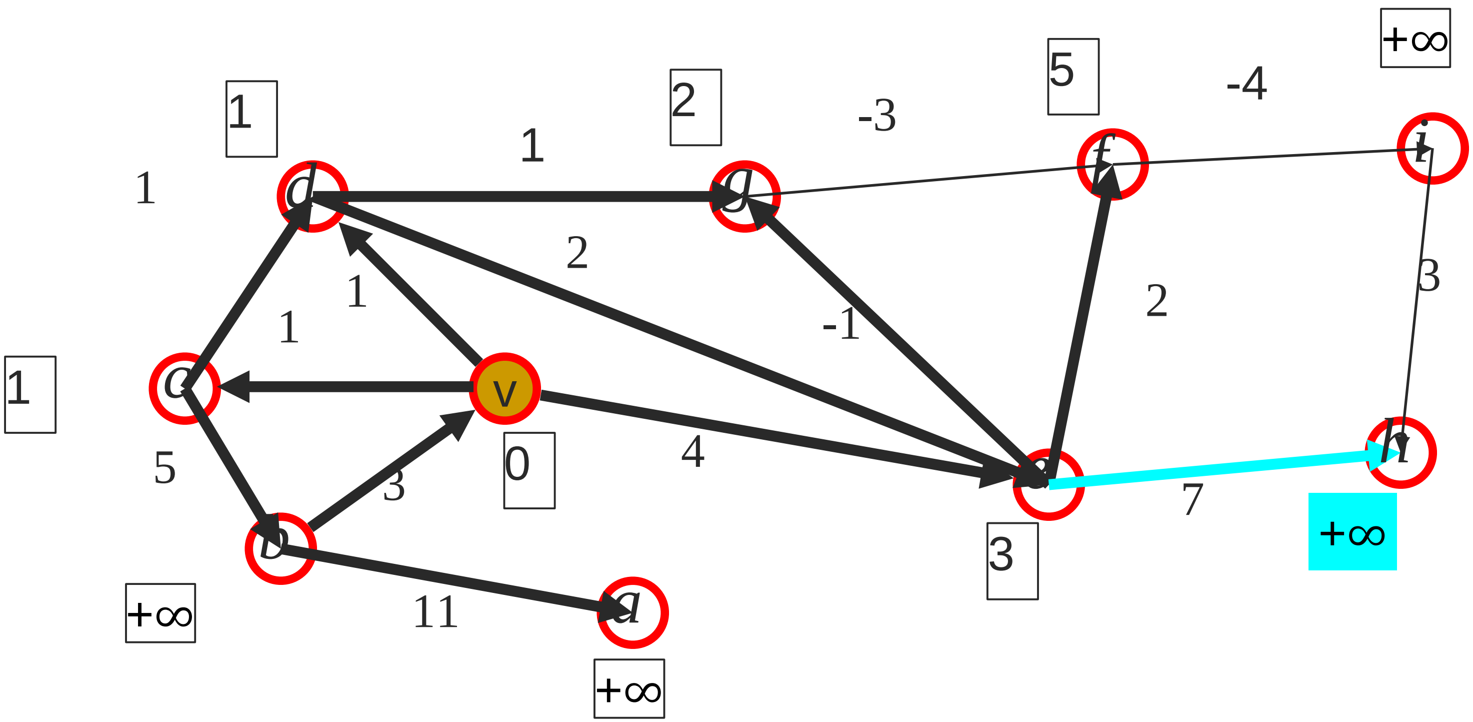
i=1



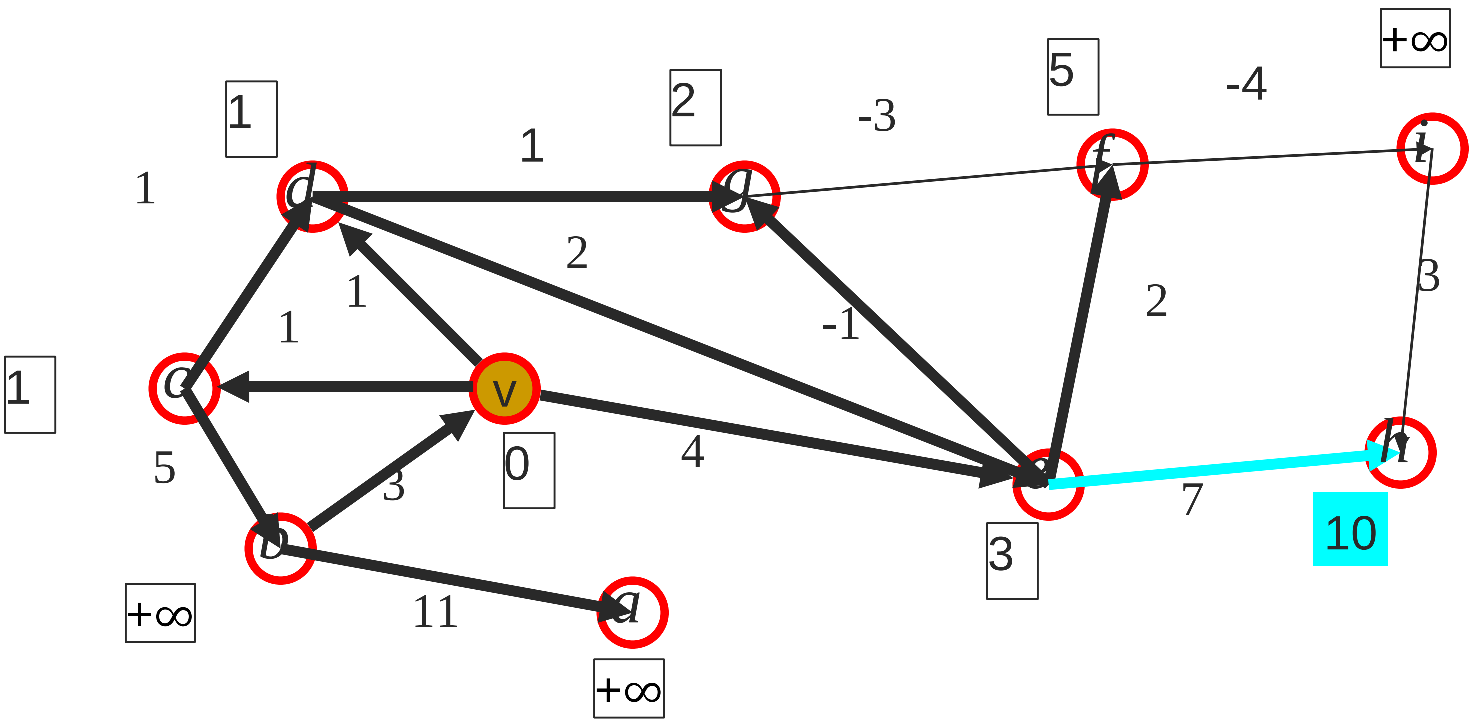
i=1



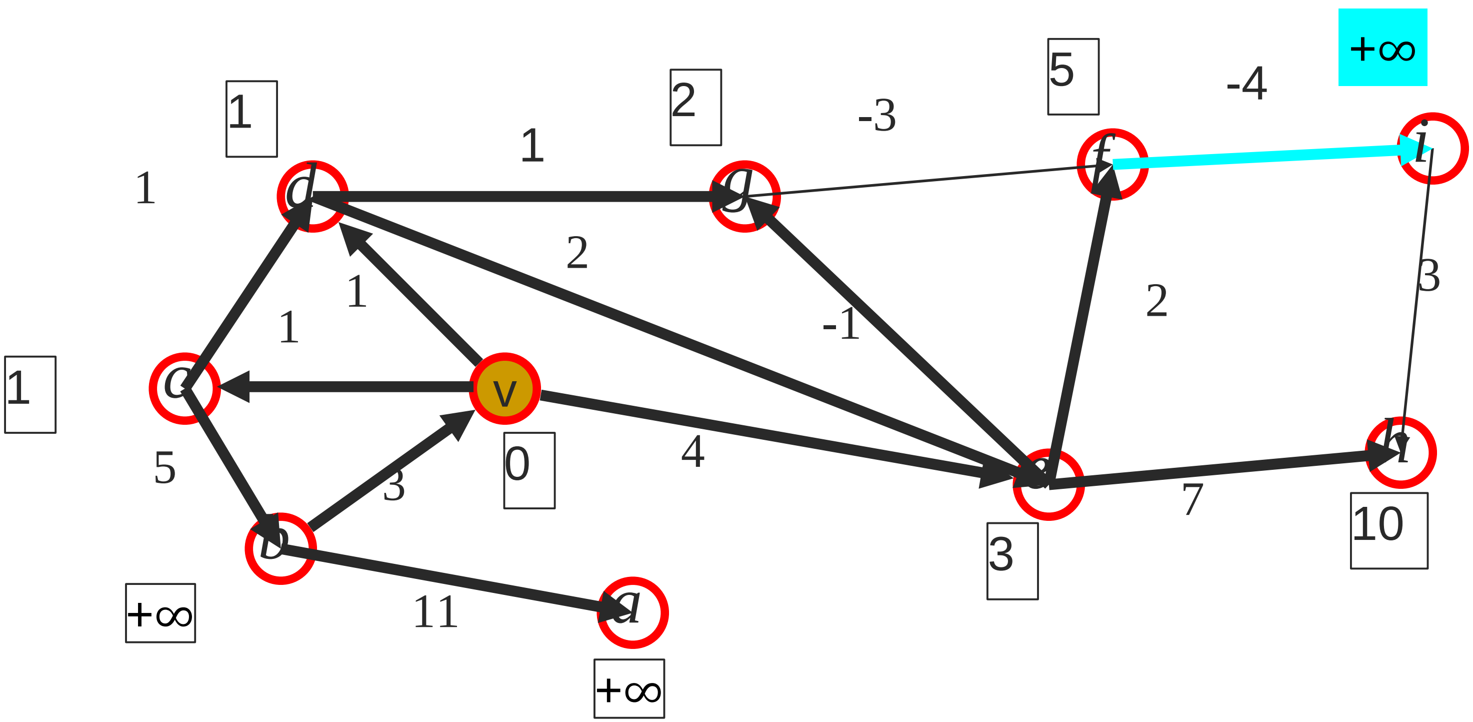
i=1



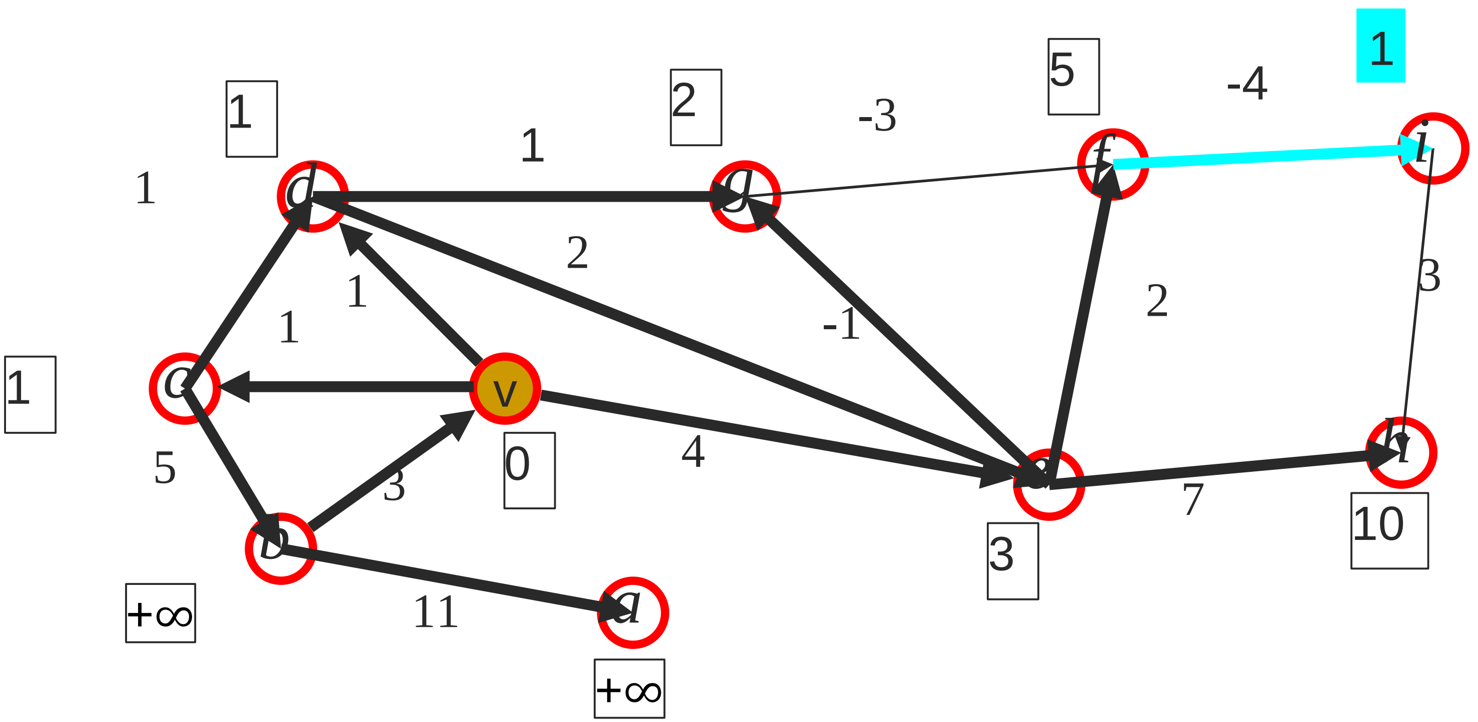
i=1



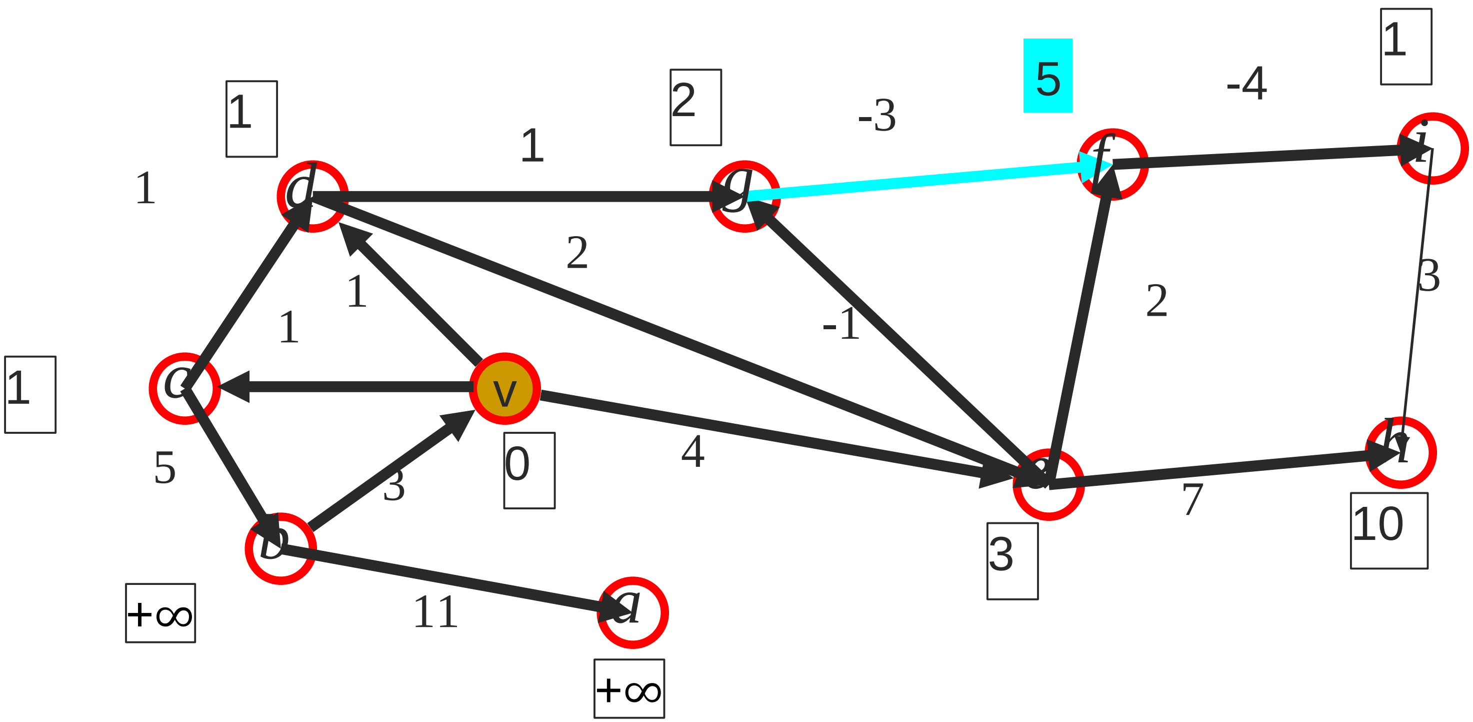
i=1



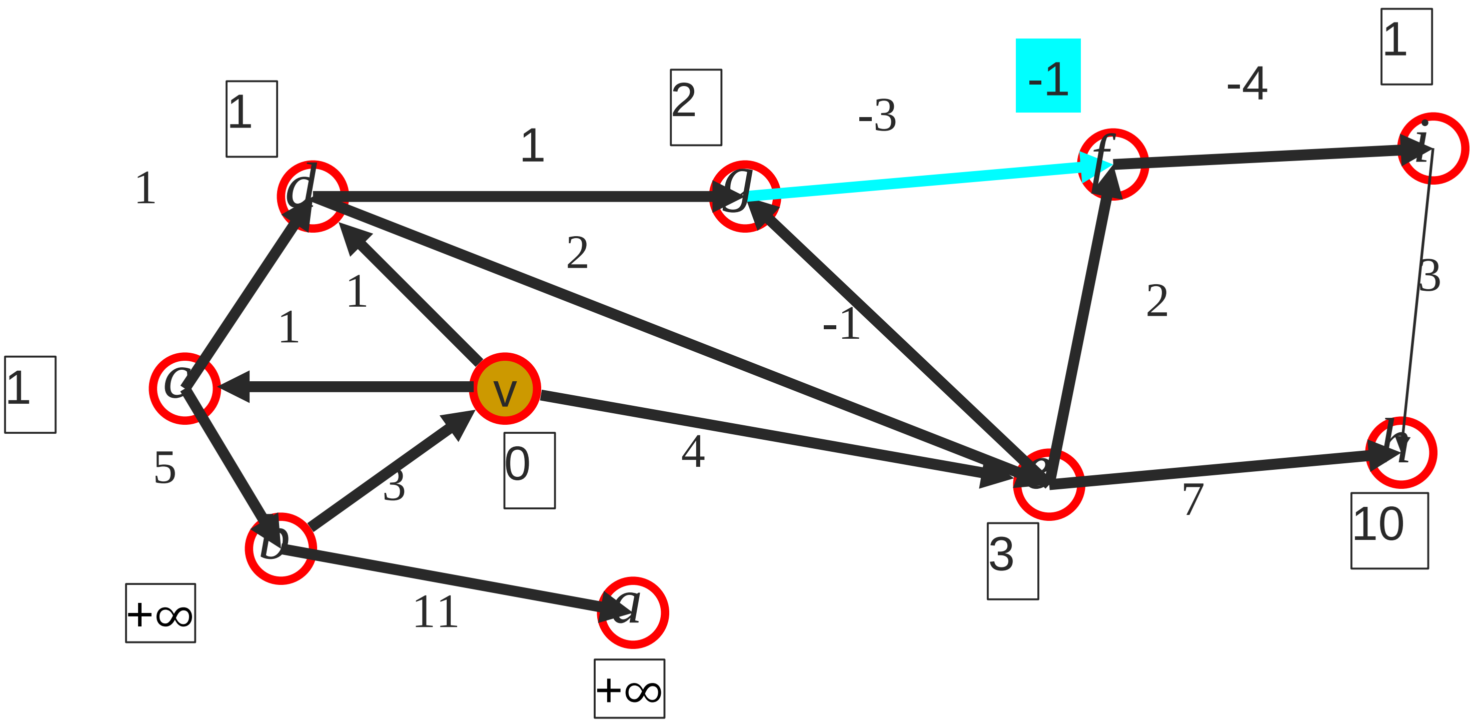
i=1



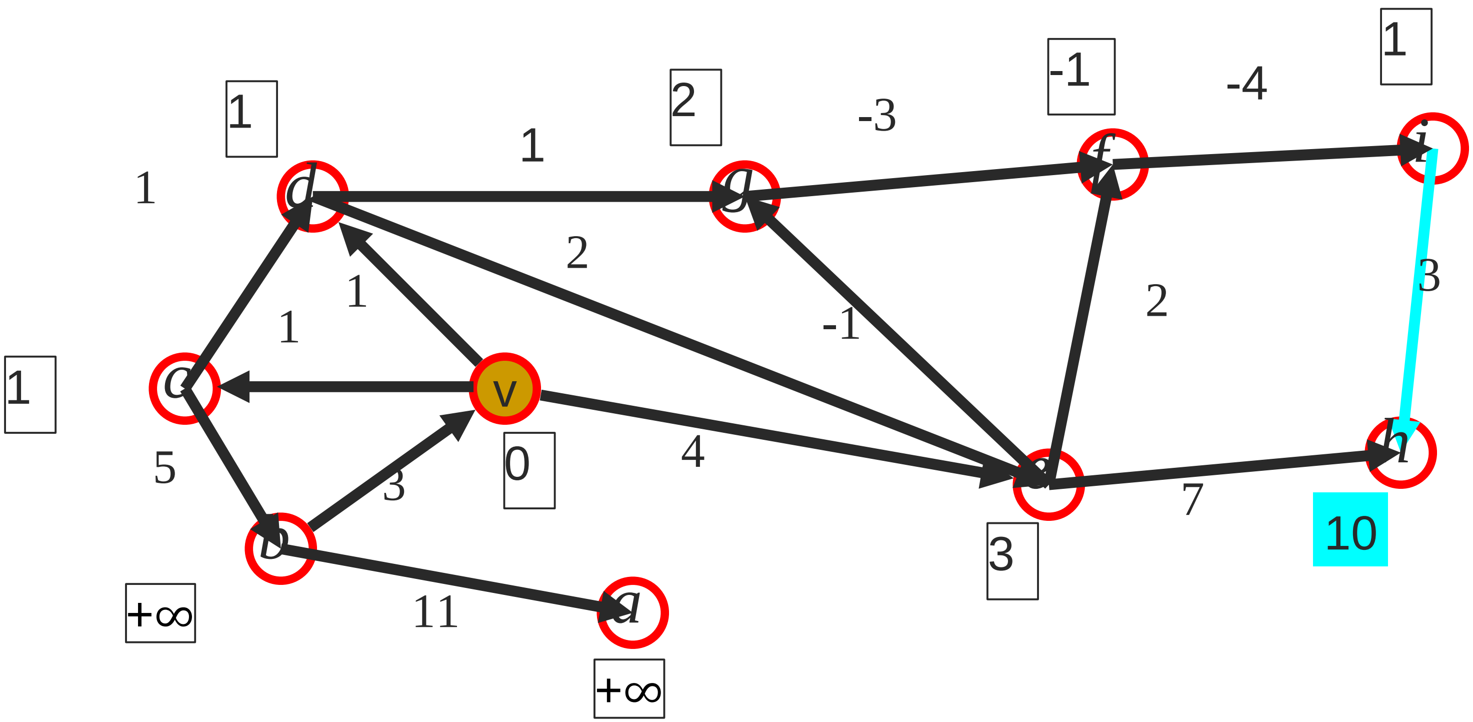
i=1



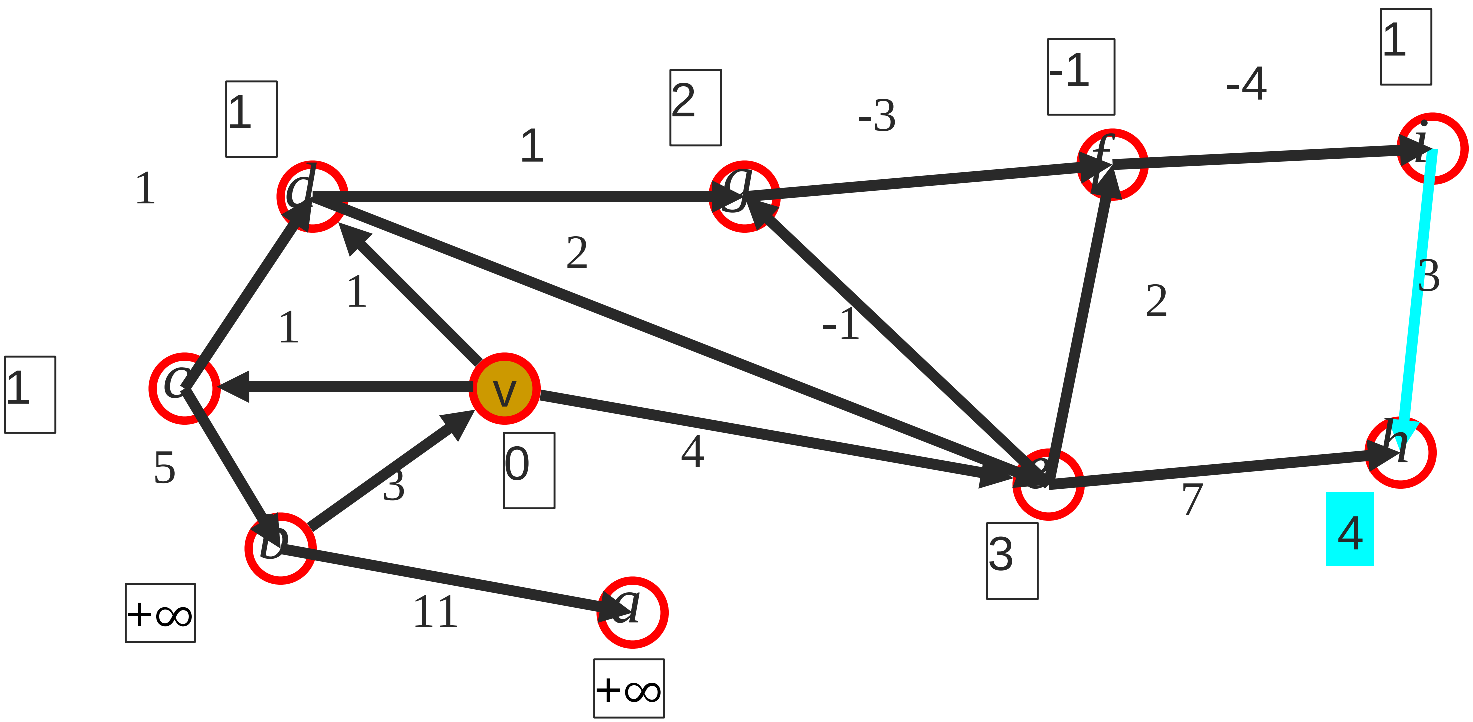
i=1



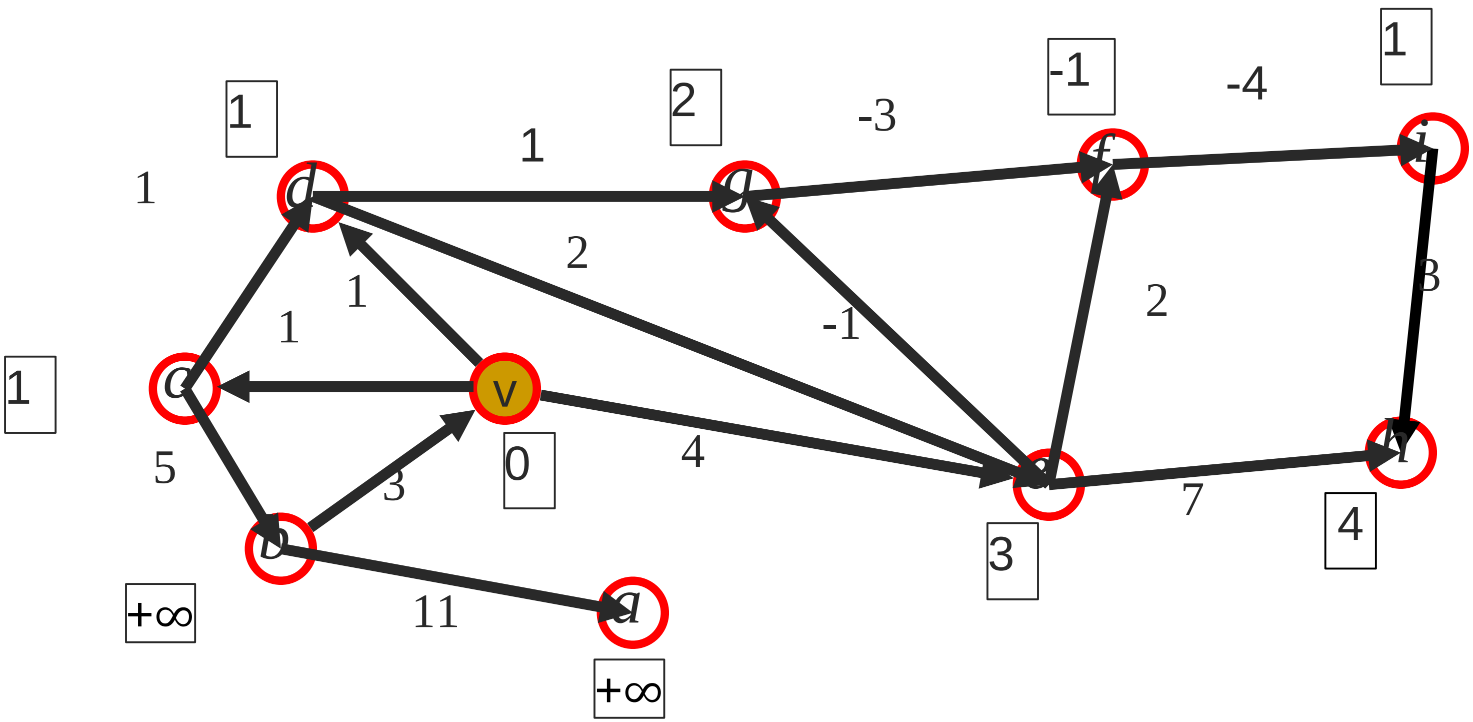
i=1



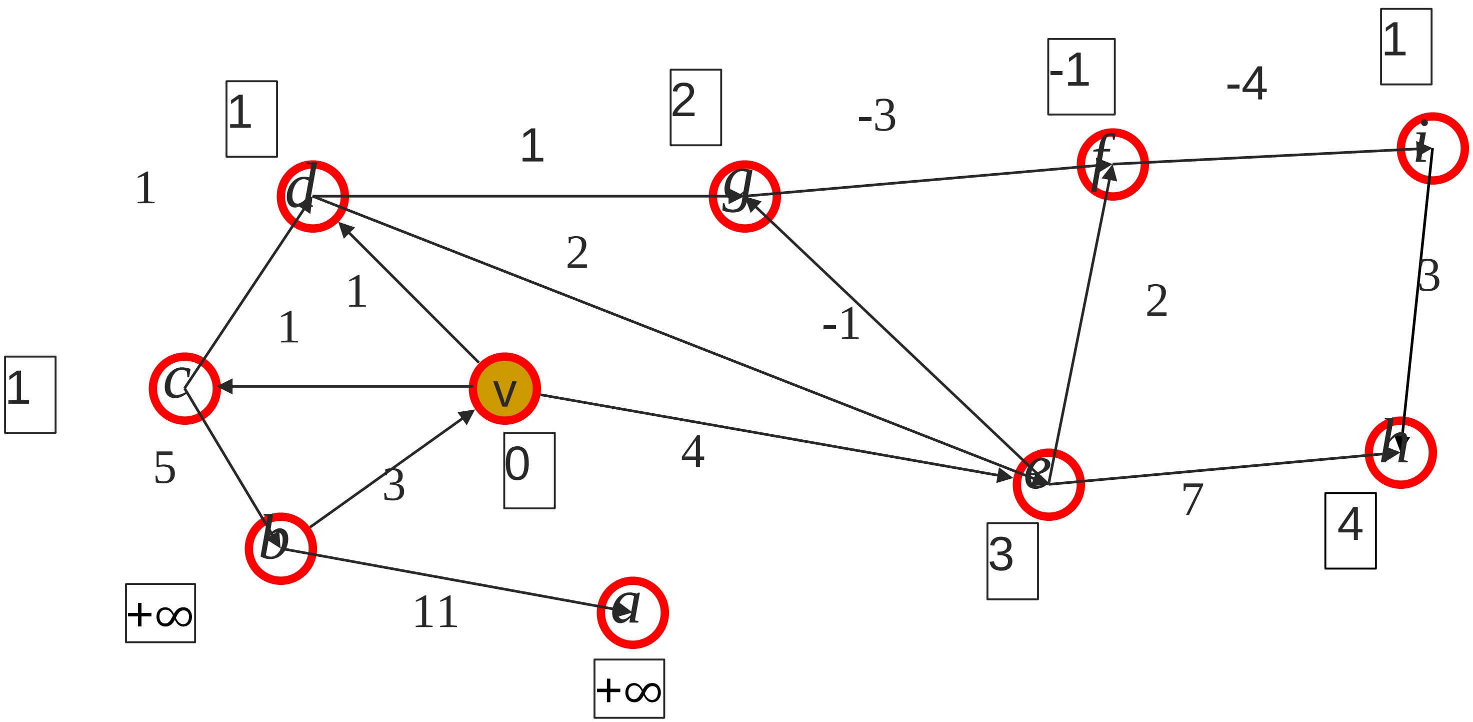
i=1



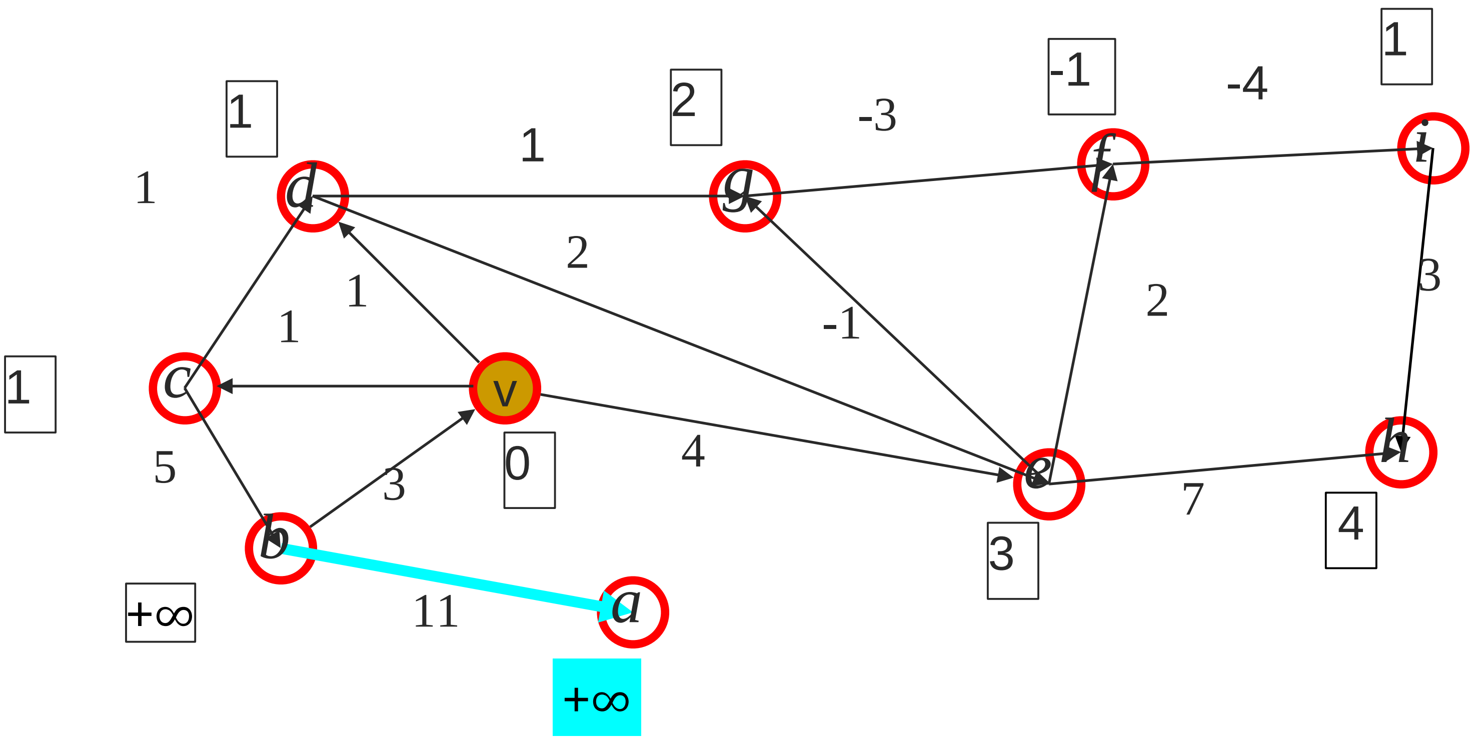
i=1



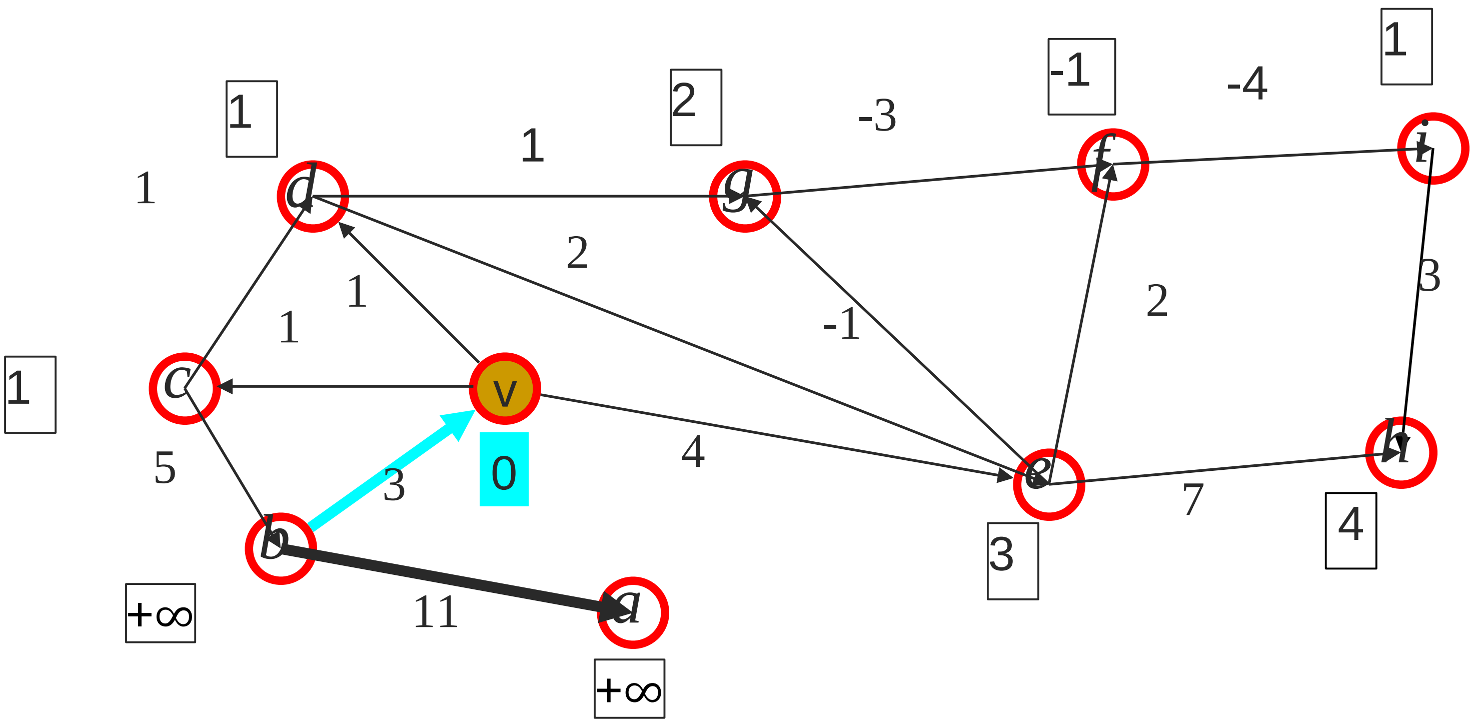
i=1



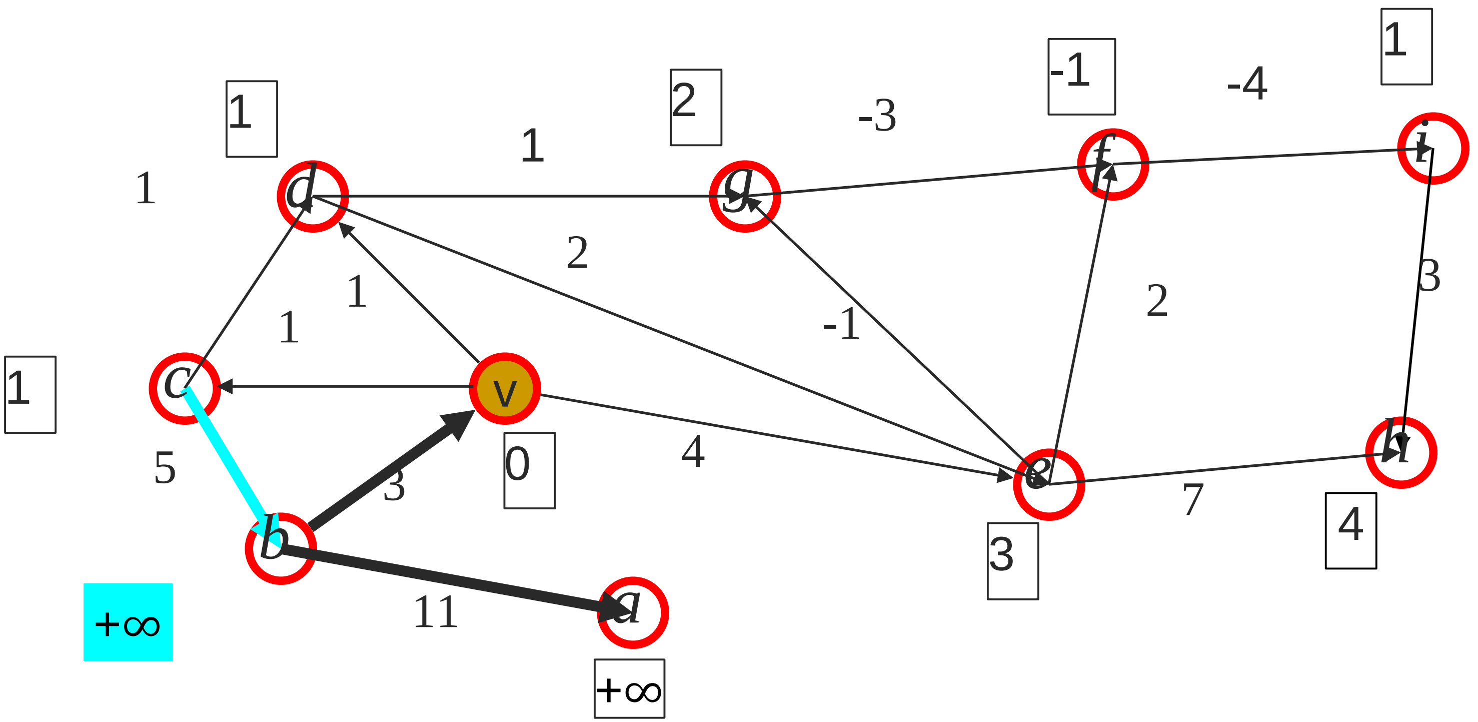
i=2



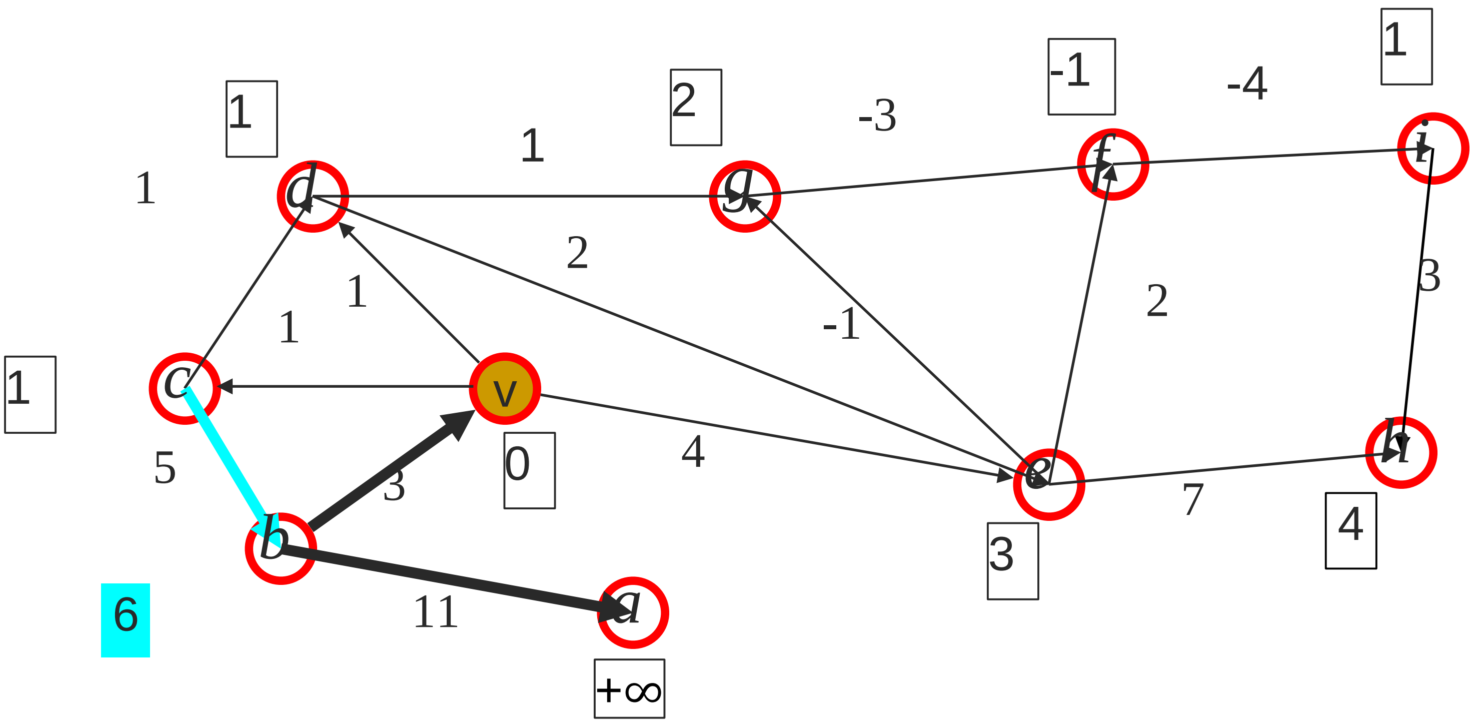
i=2



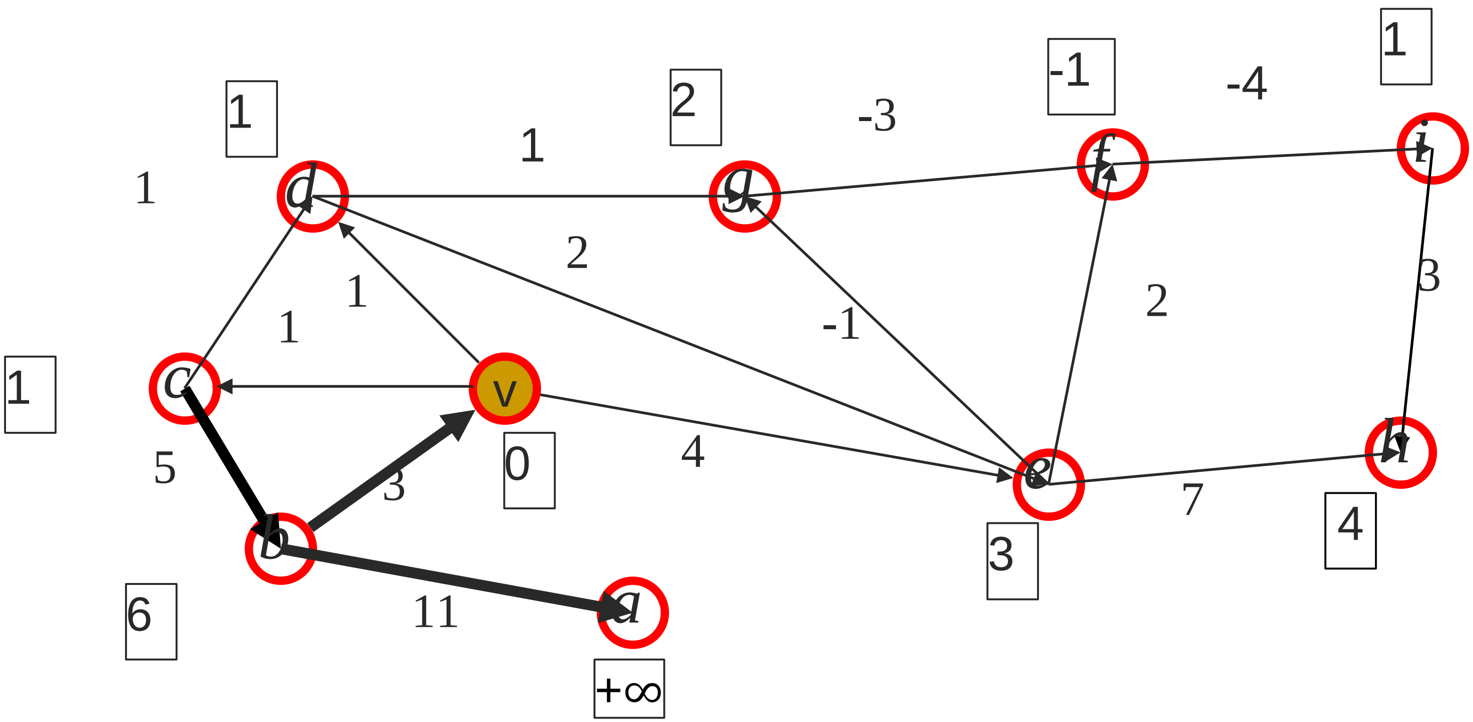
i=2



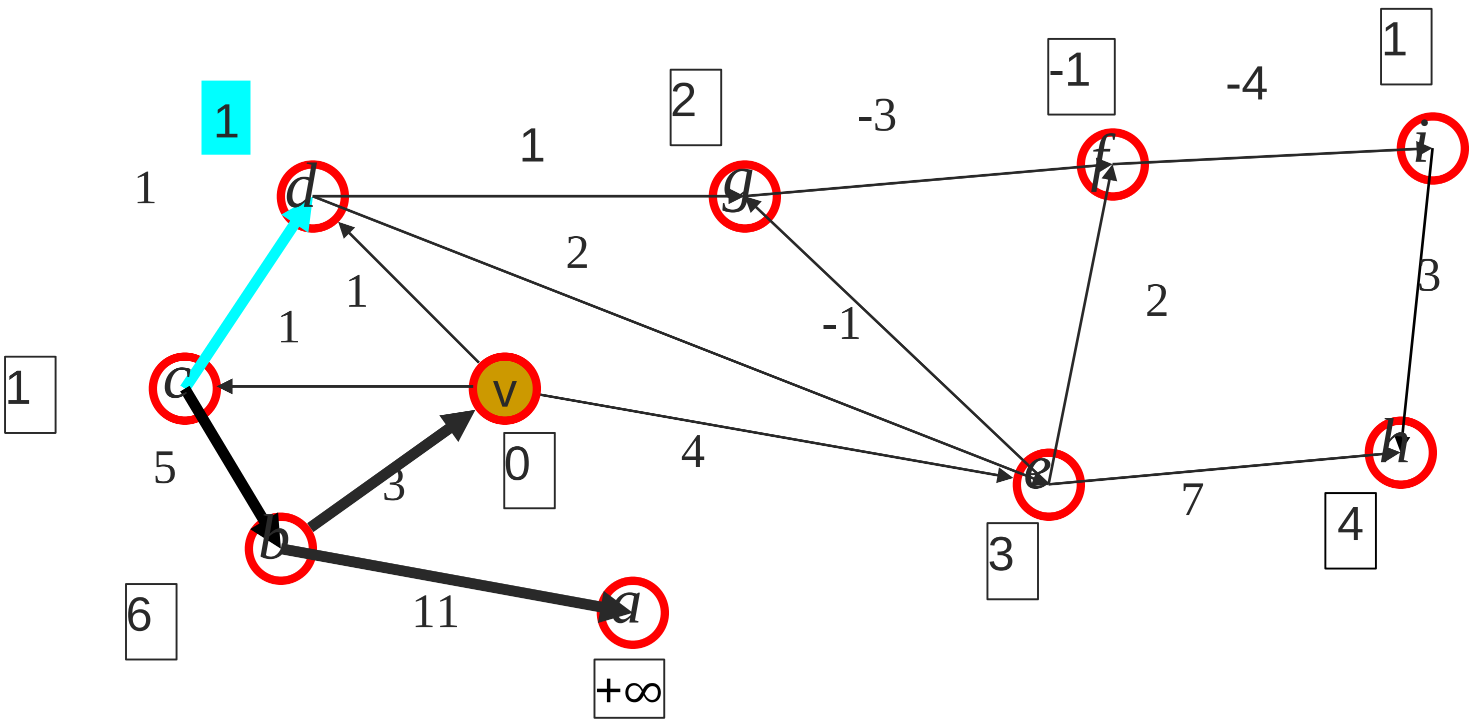
i=2



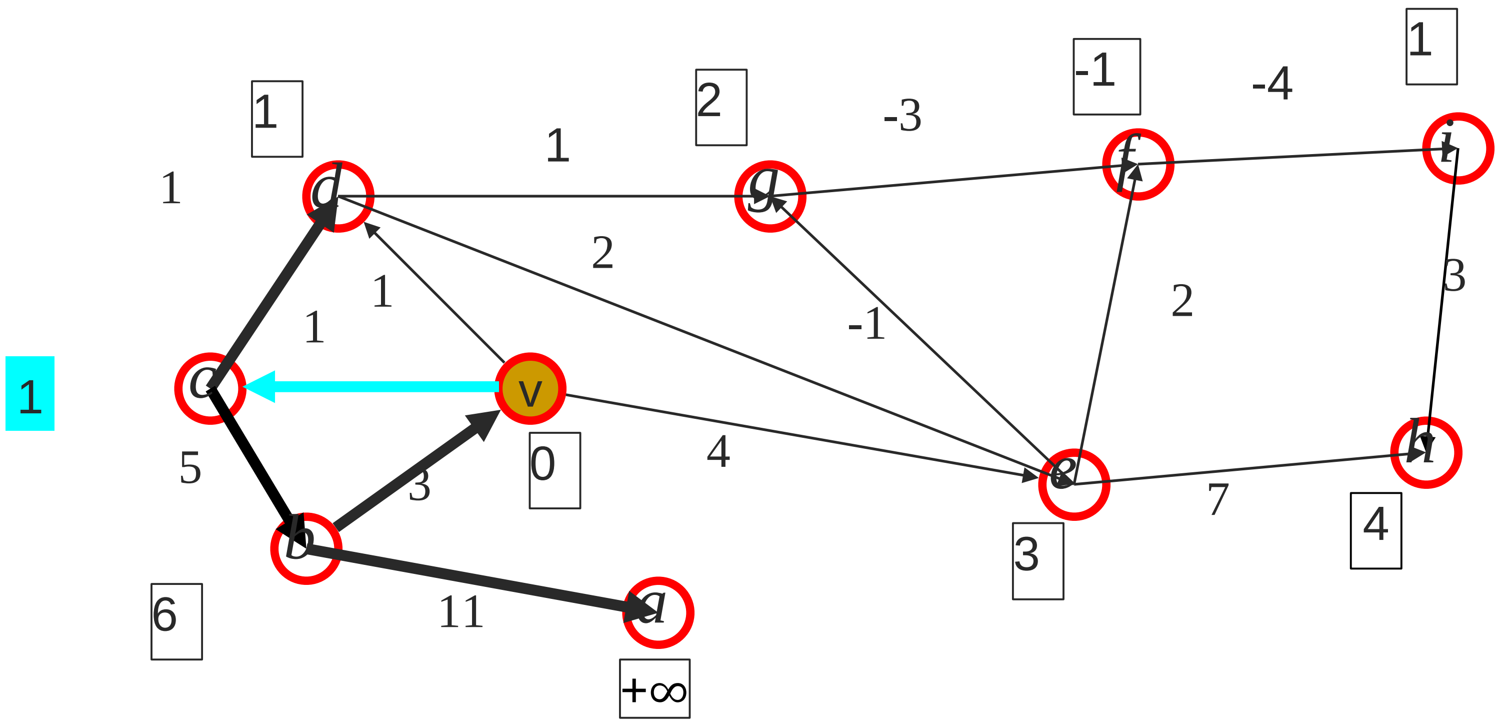
i=2



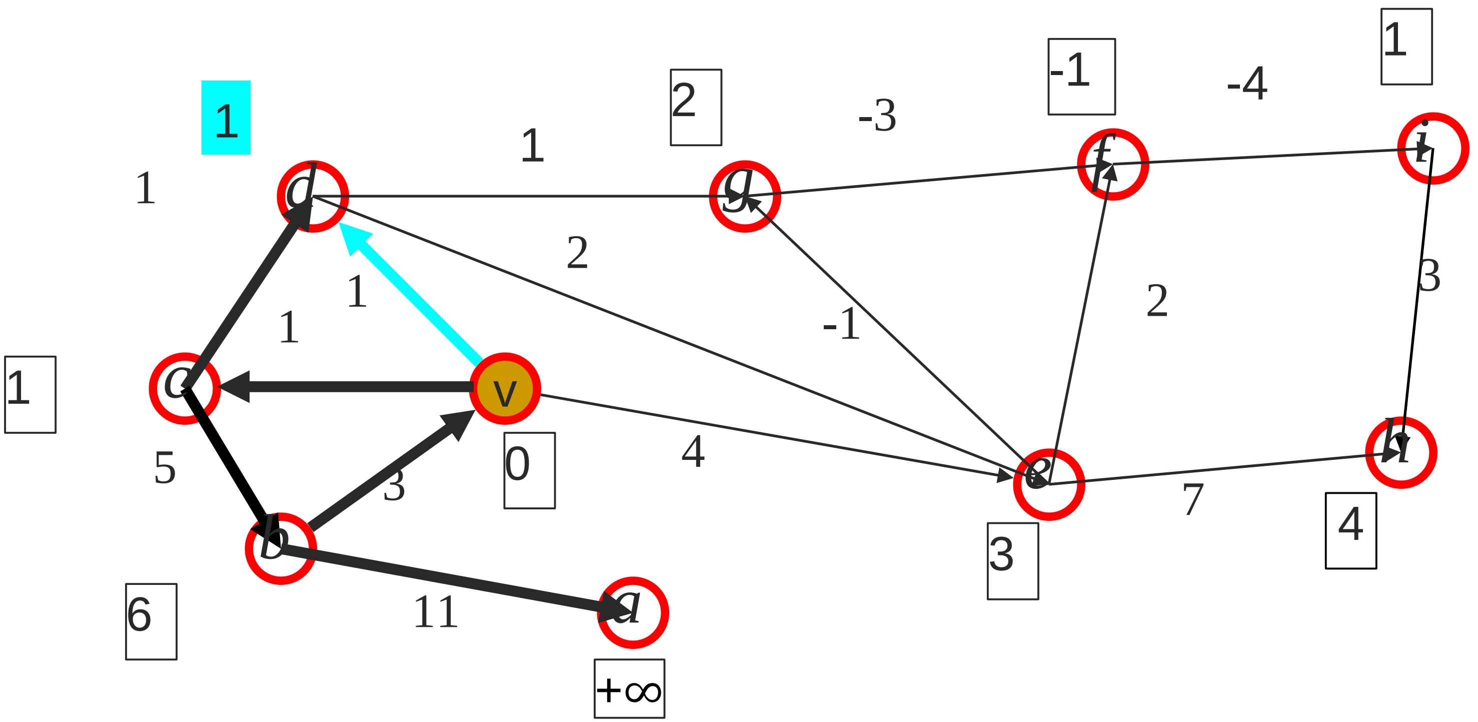
i=2



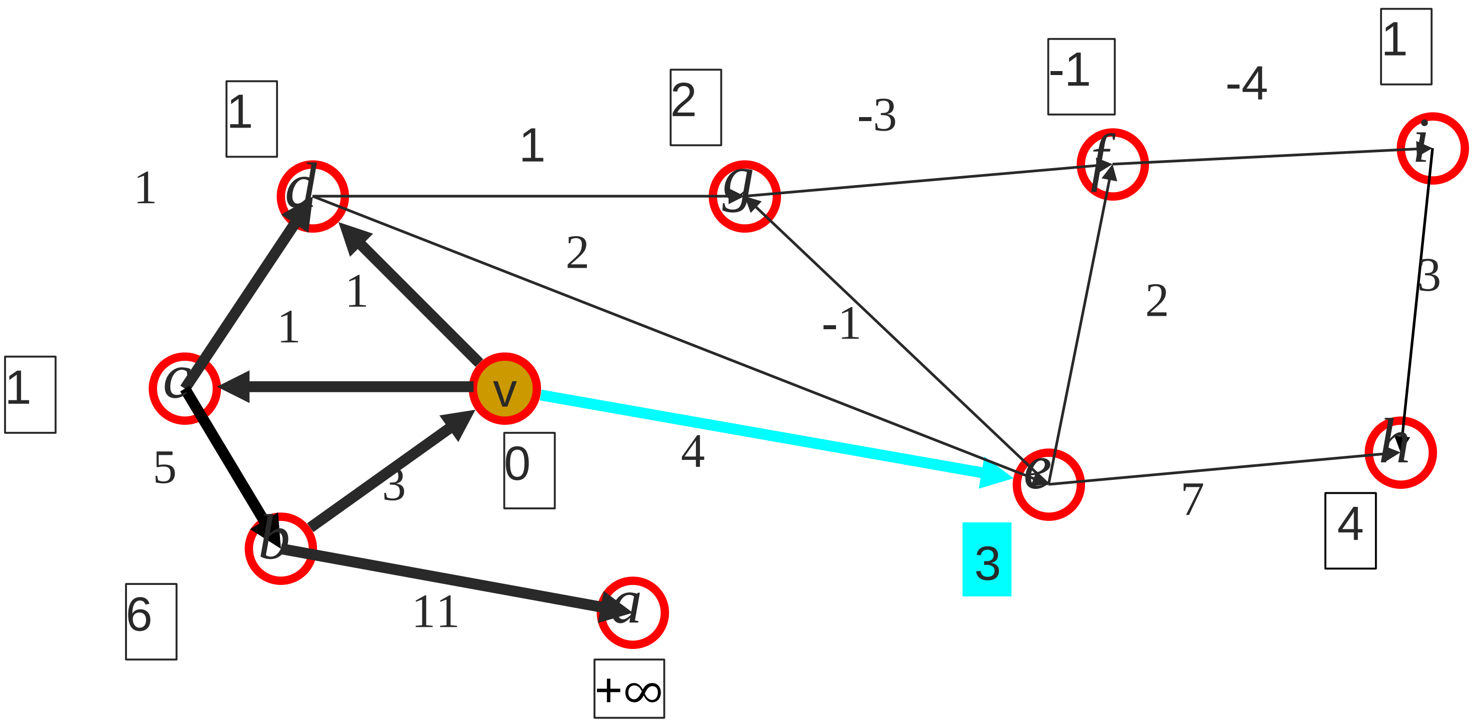
i=2



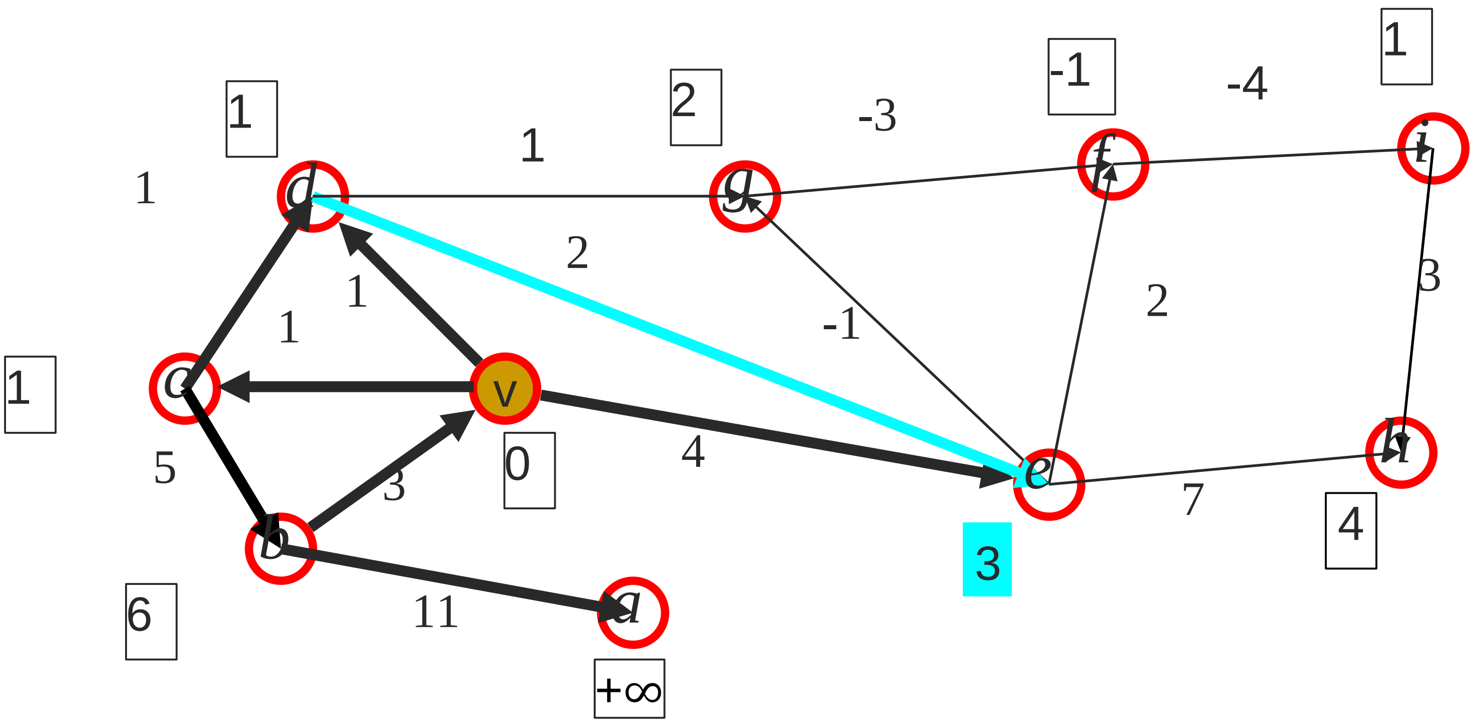
i=2



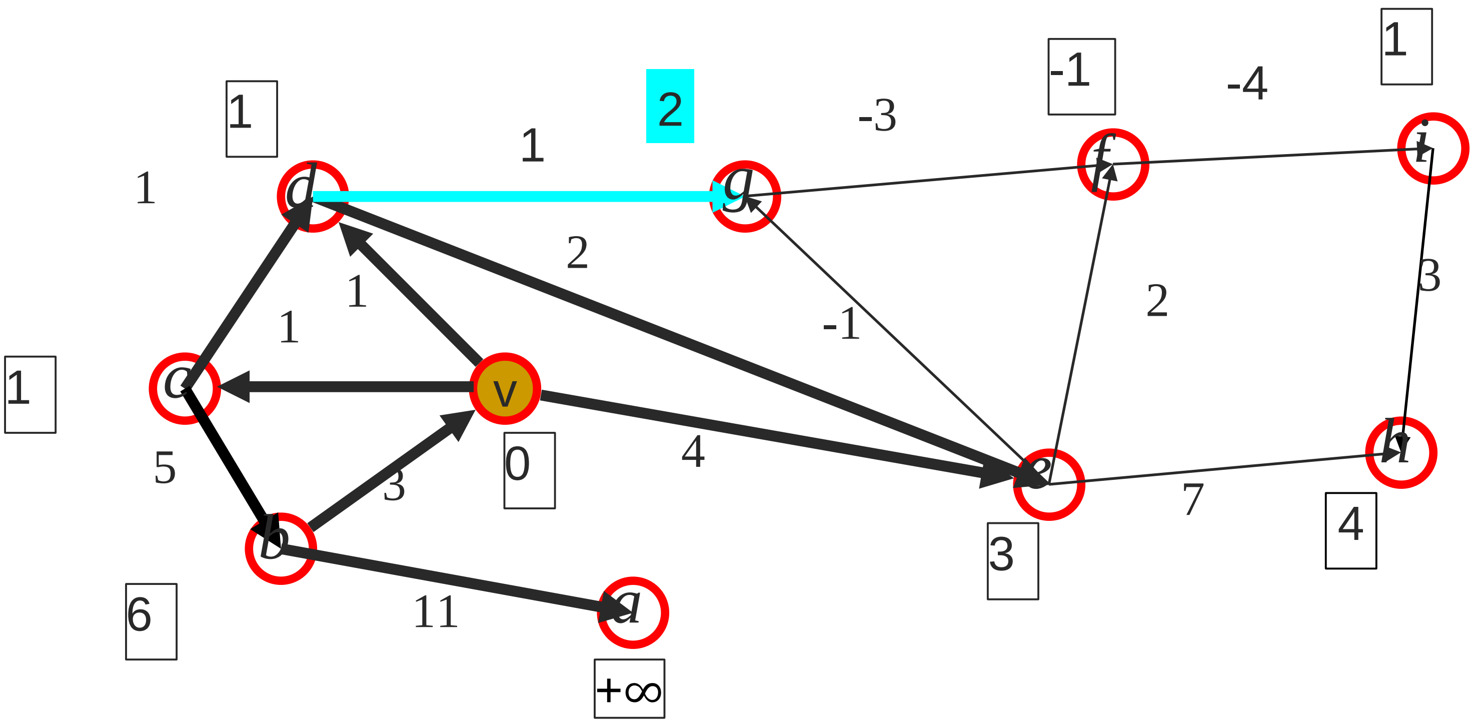
i=2



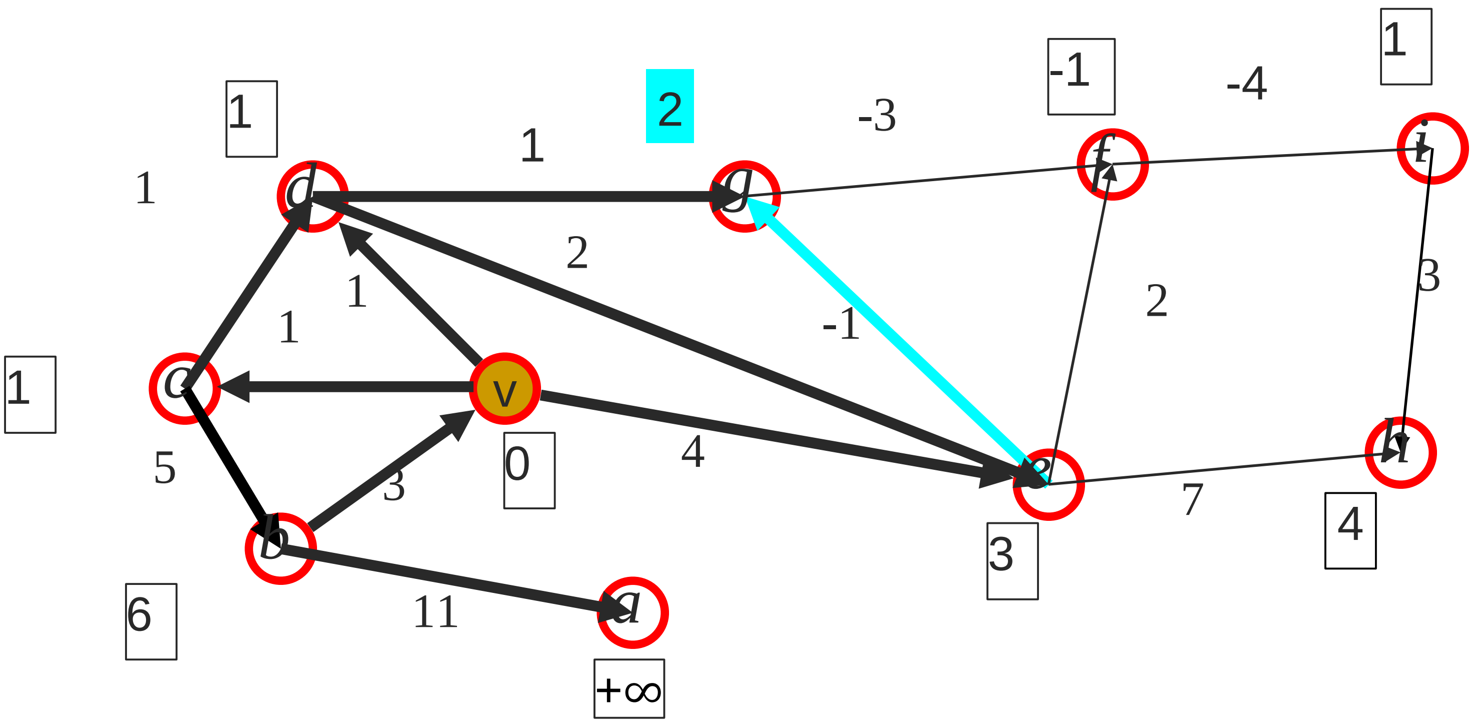
i=2



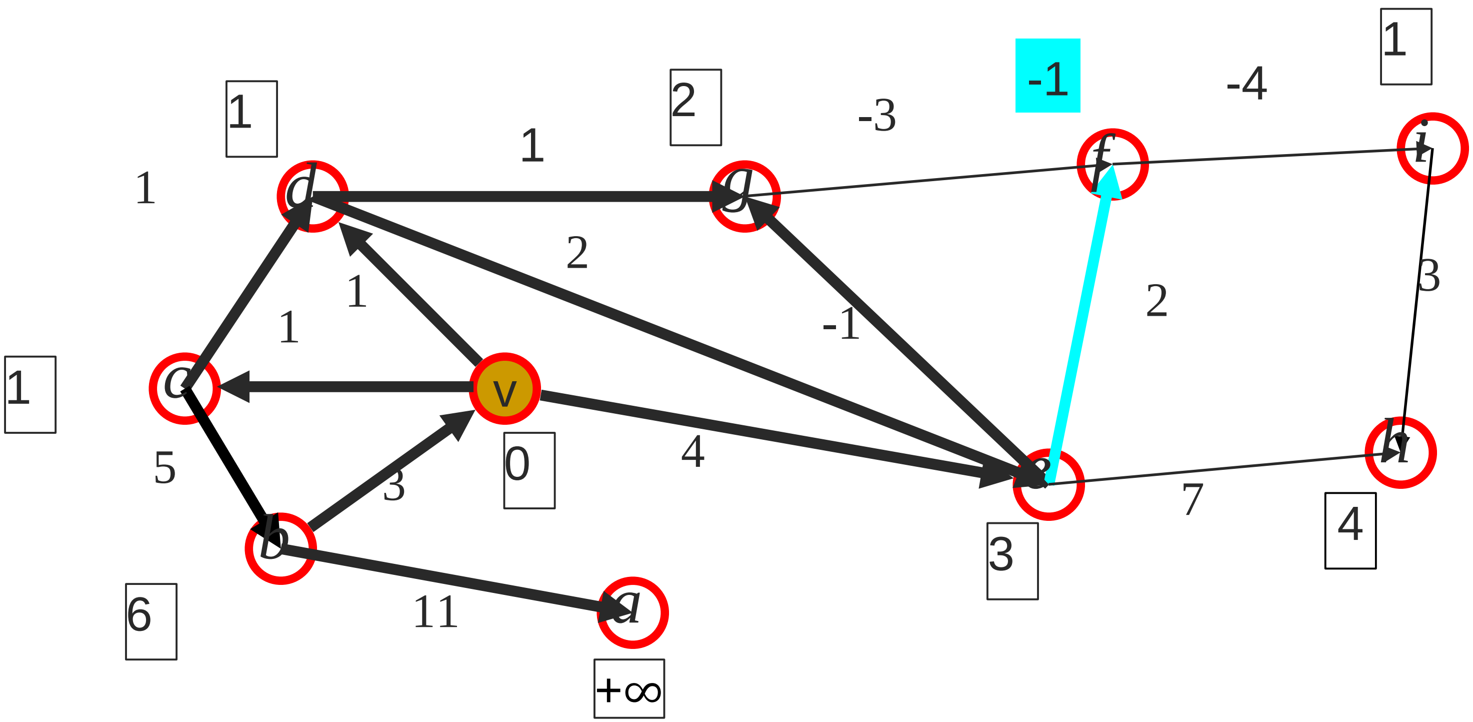
i=2



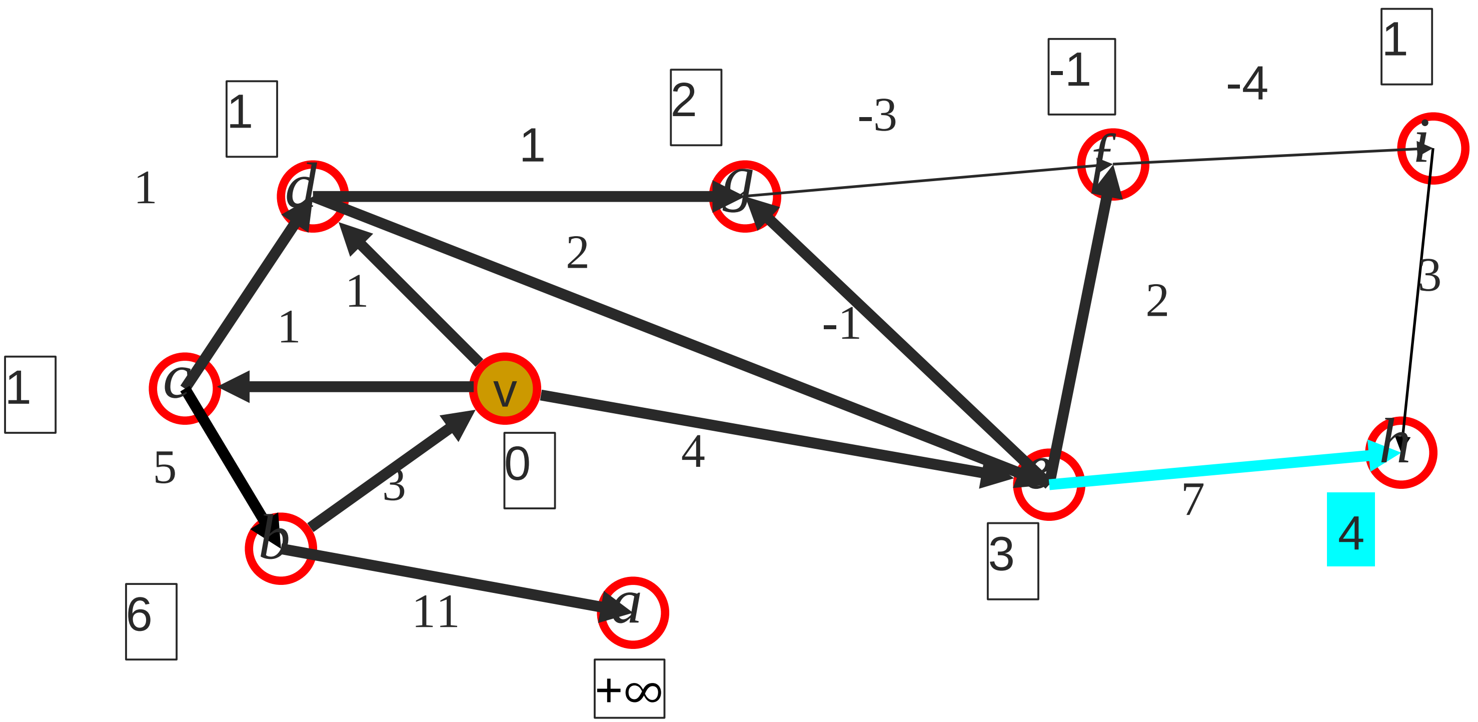
i=2



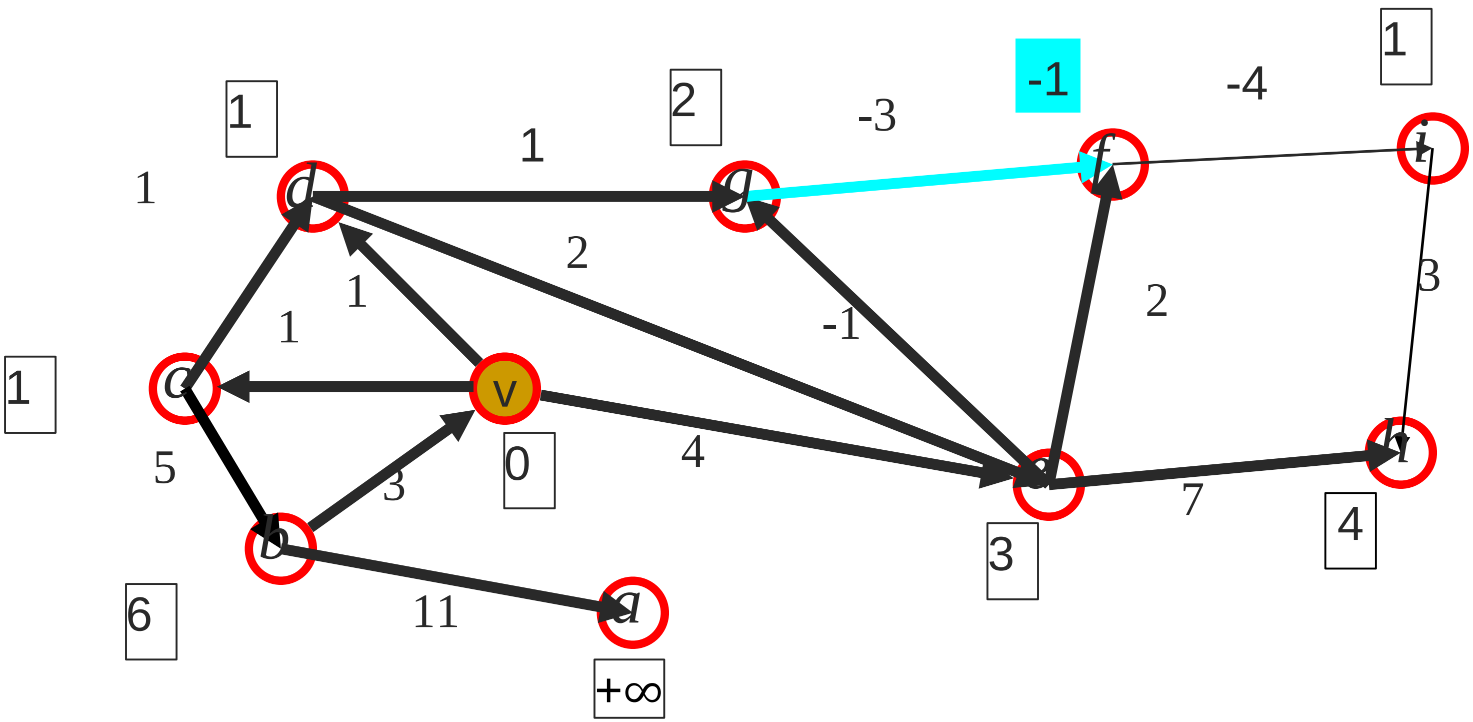
i=2



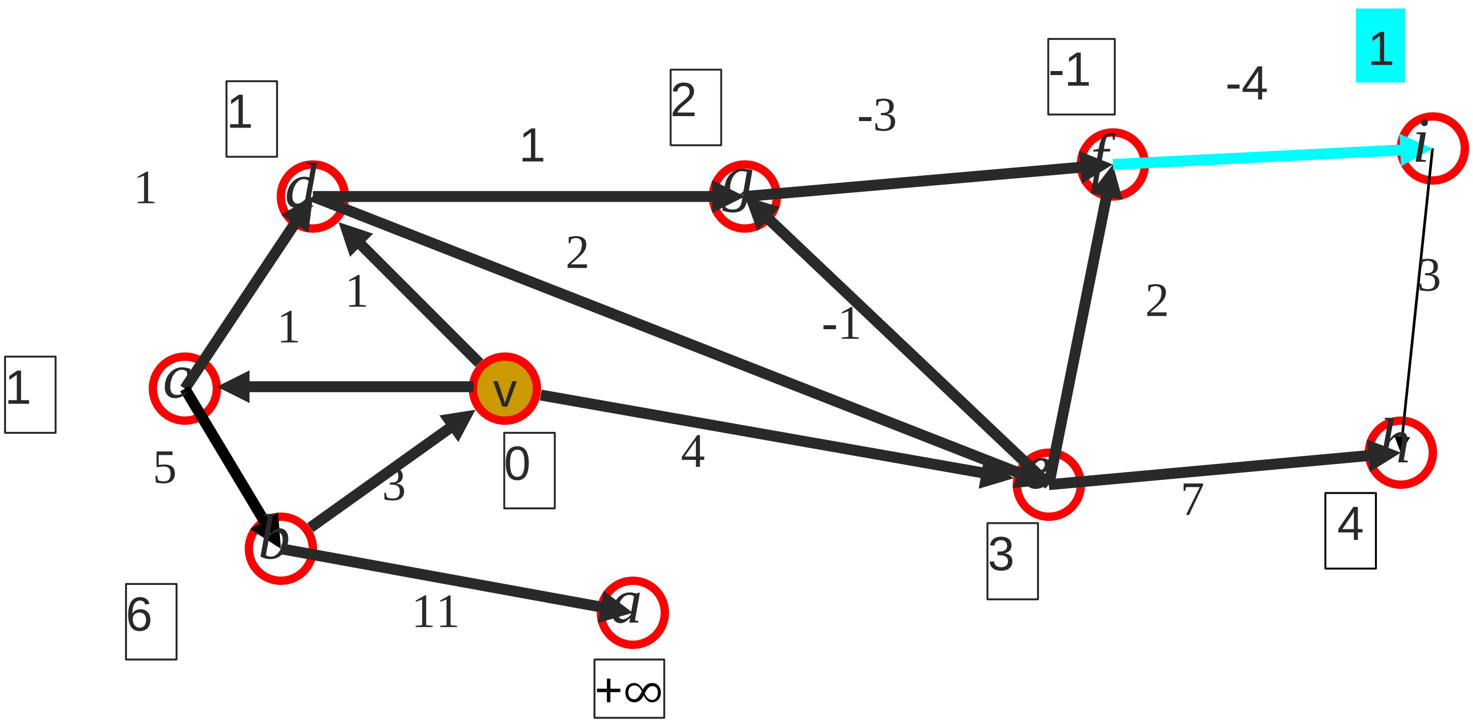
i=2



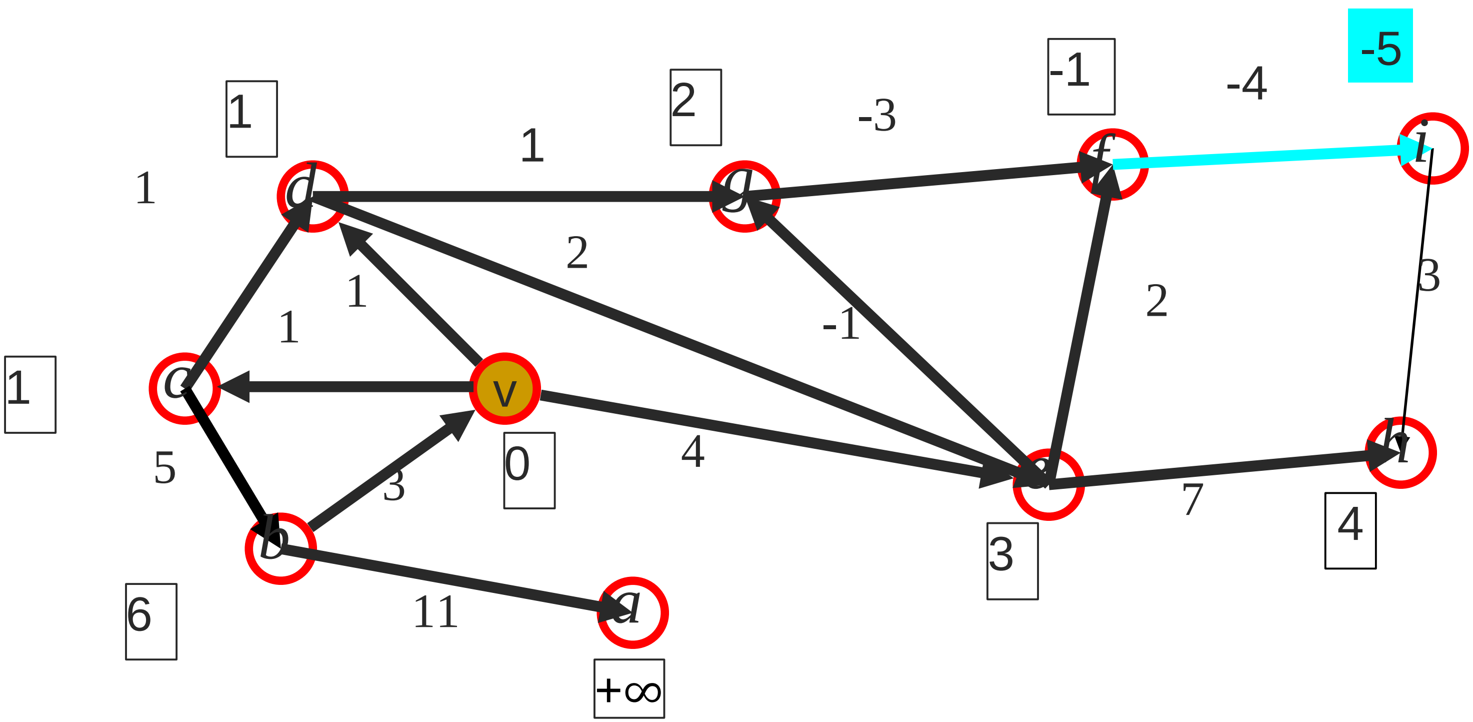
i=2



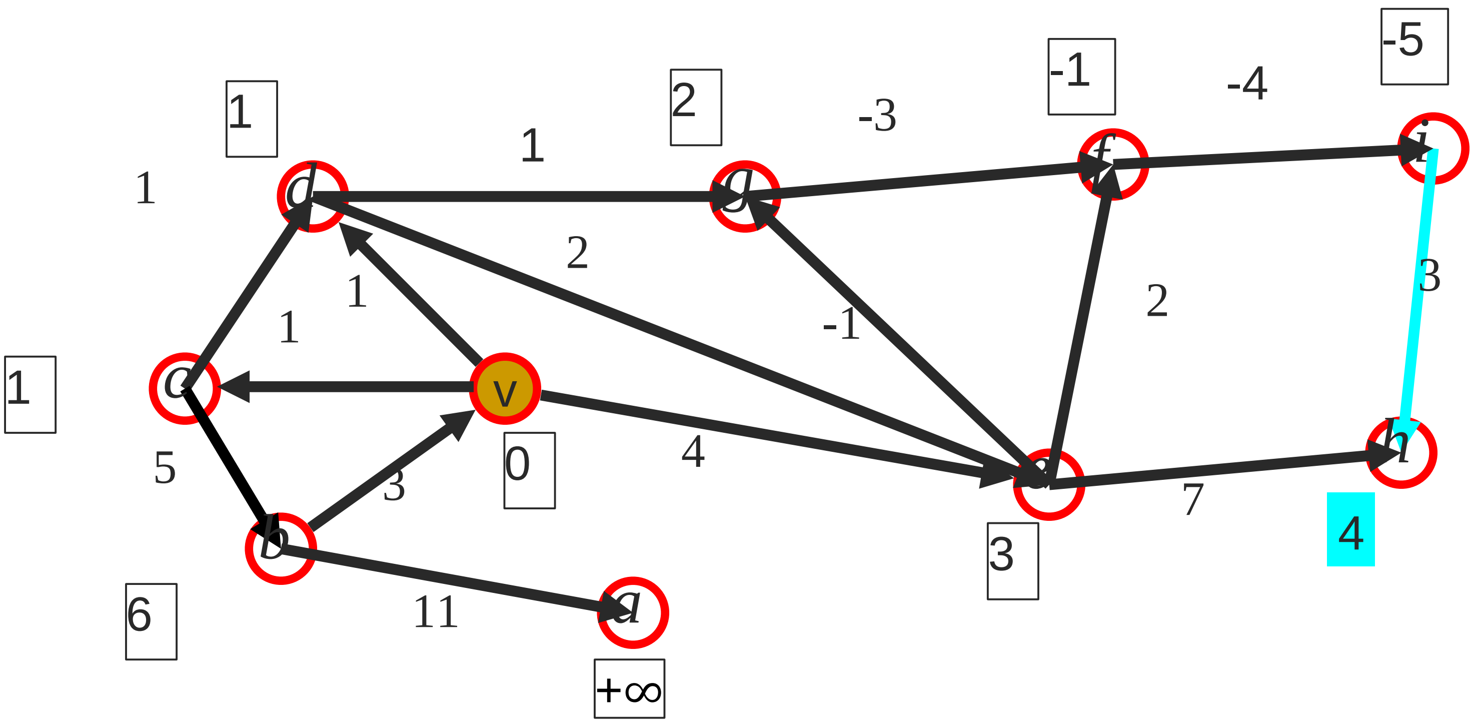
i=2



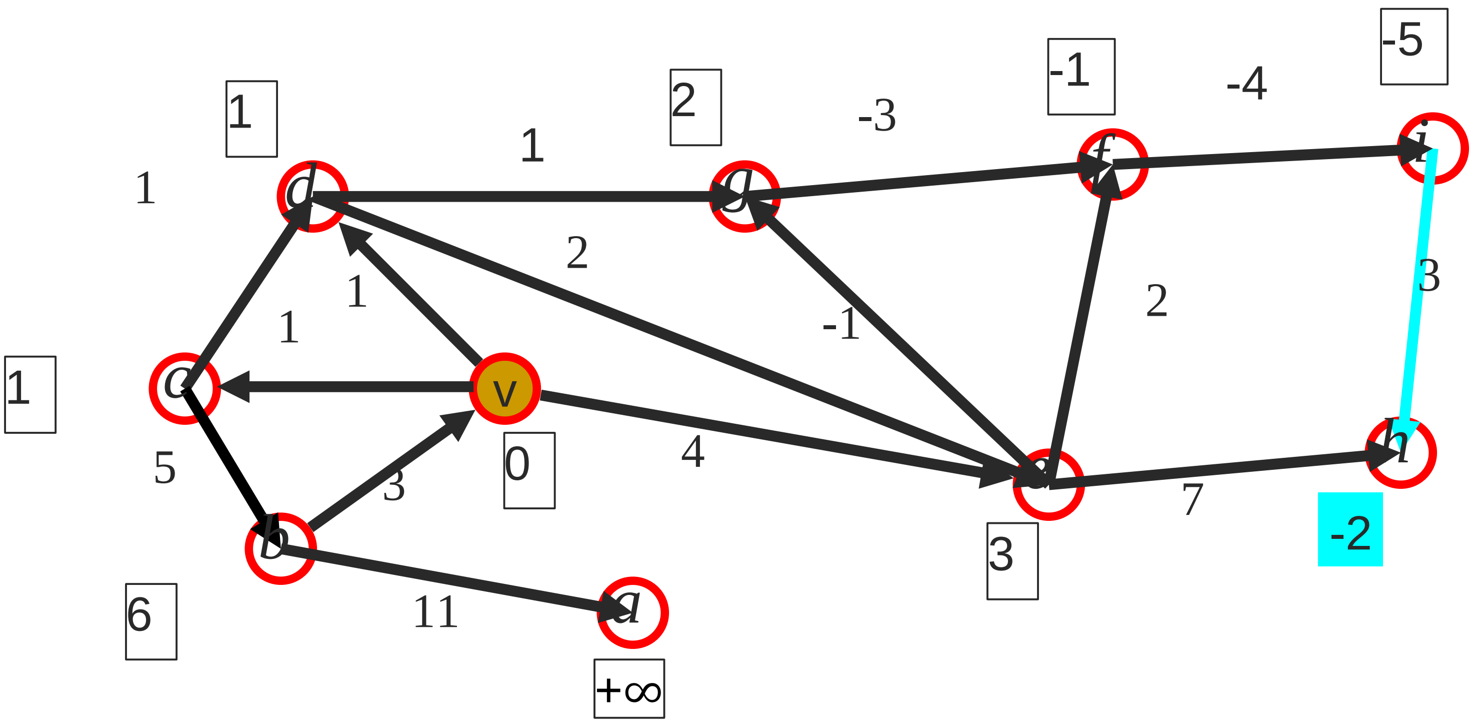
i=2



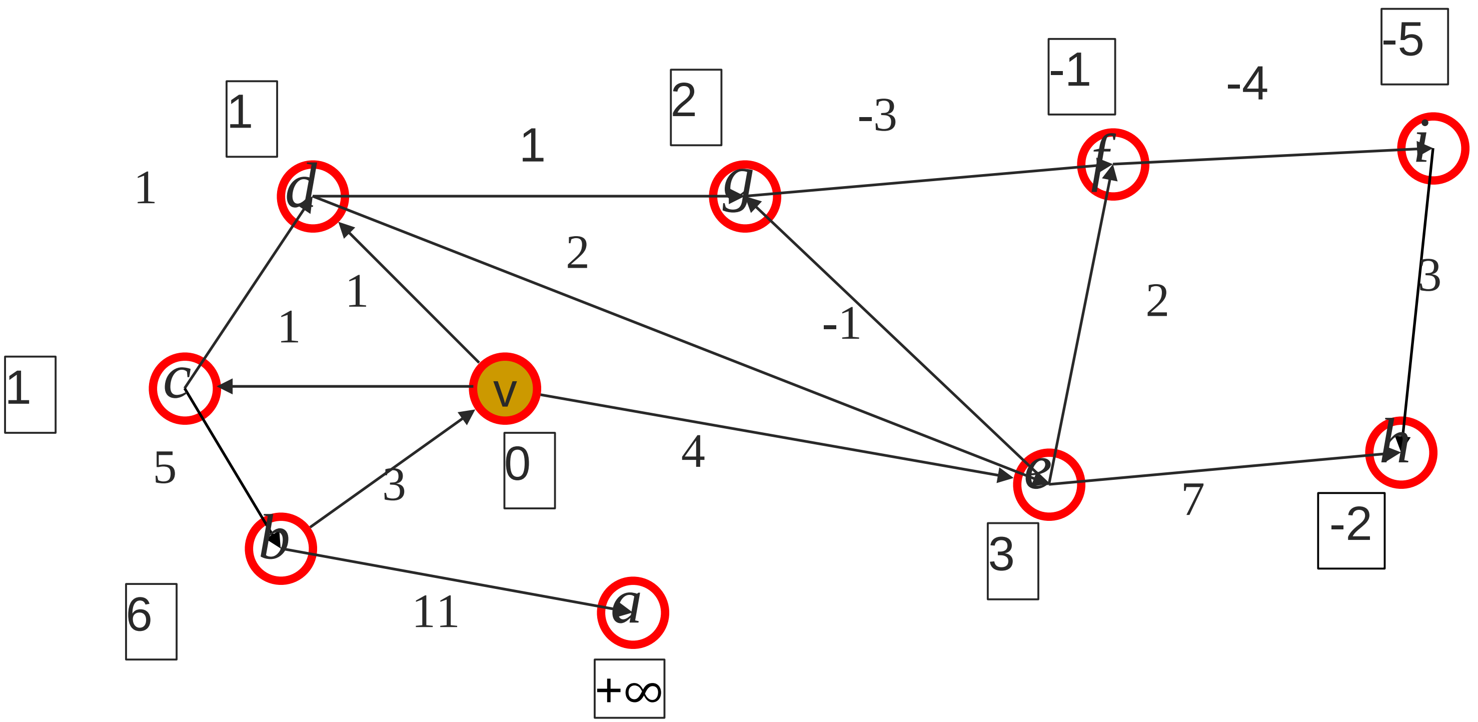
i=2



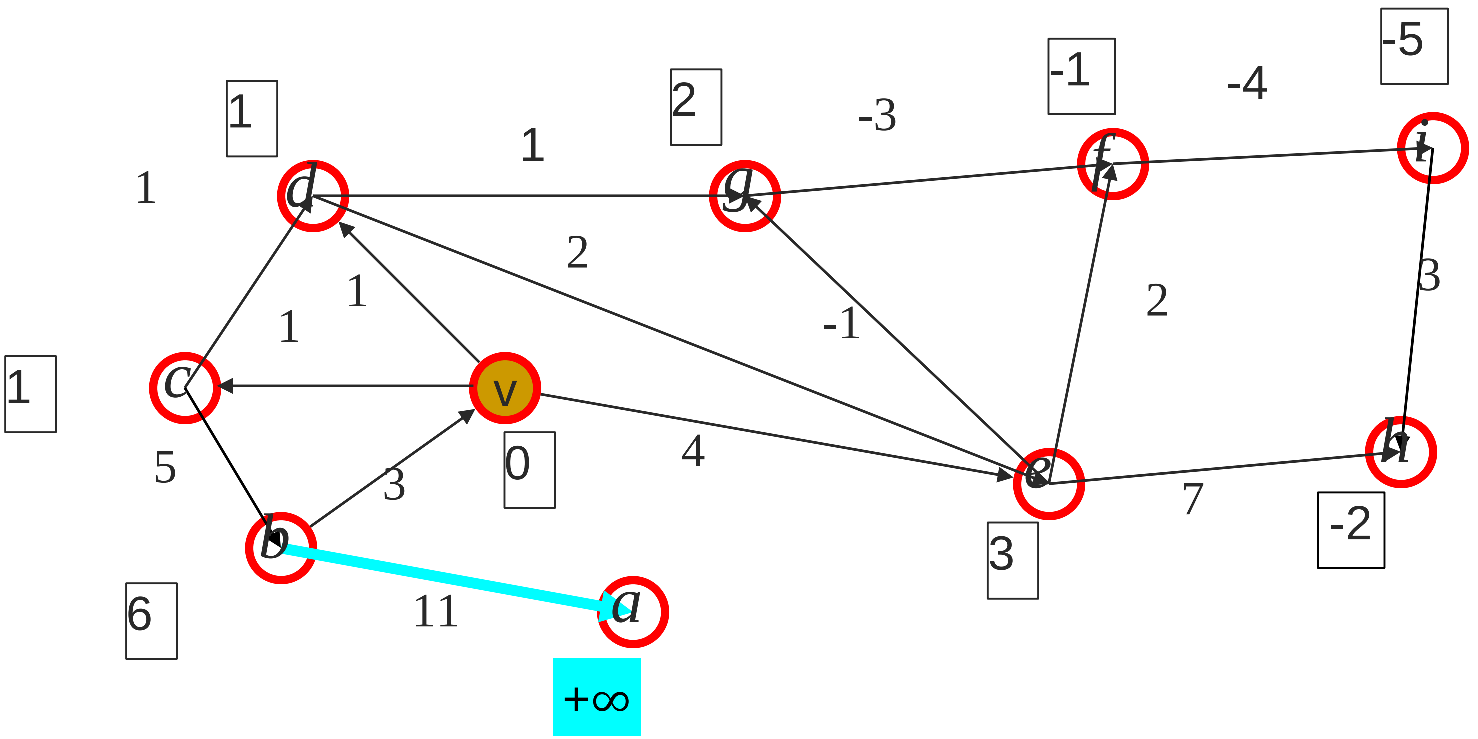
i=2



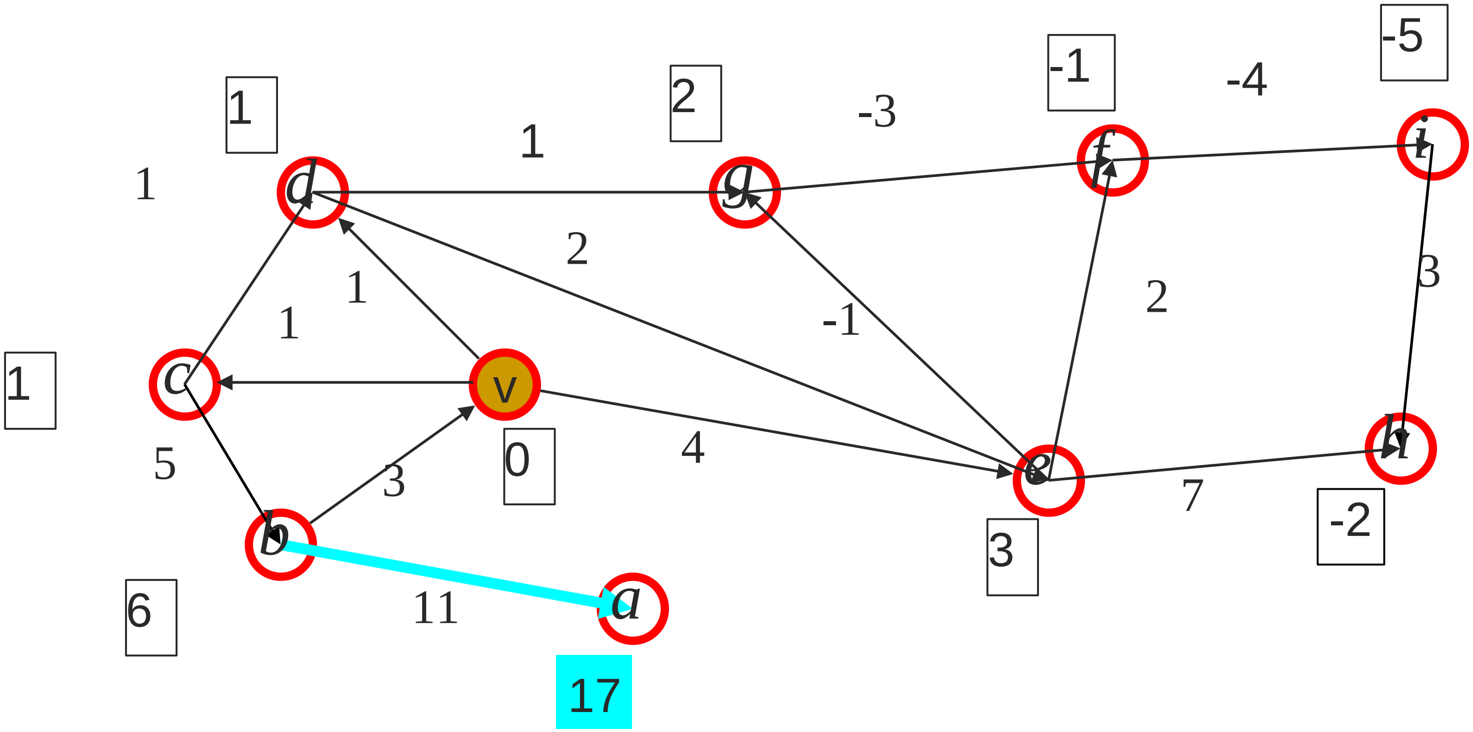
i=2



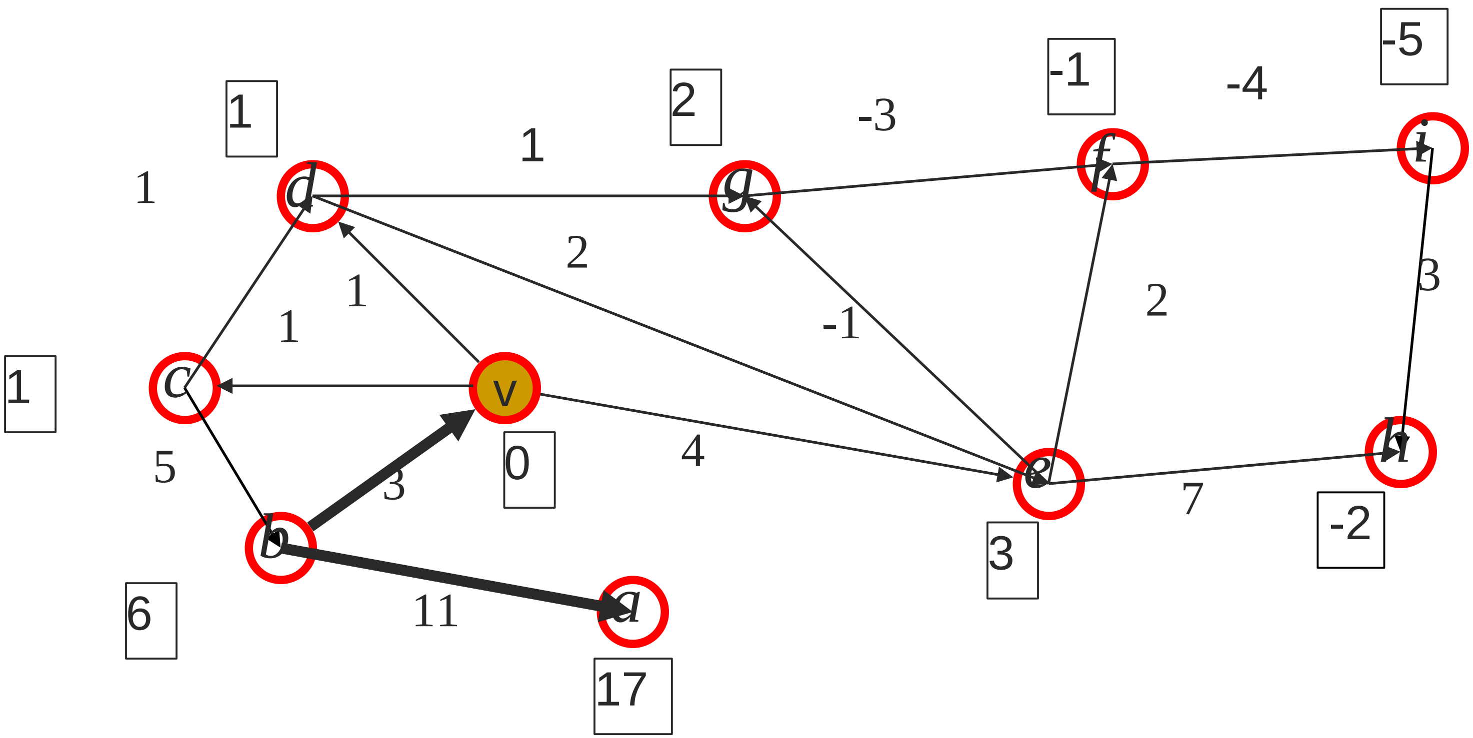
i=3



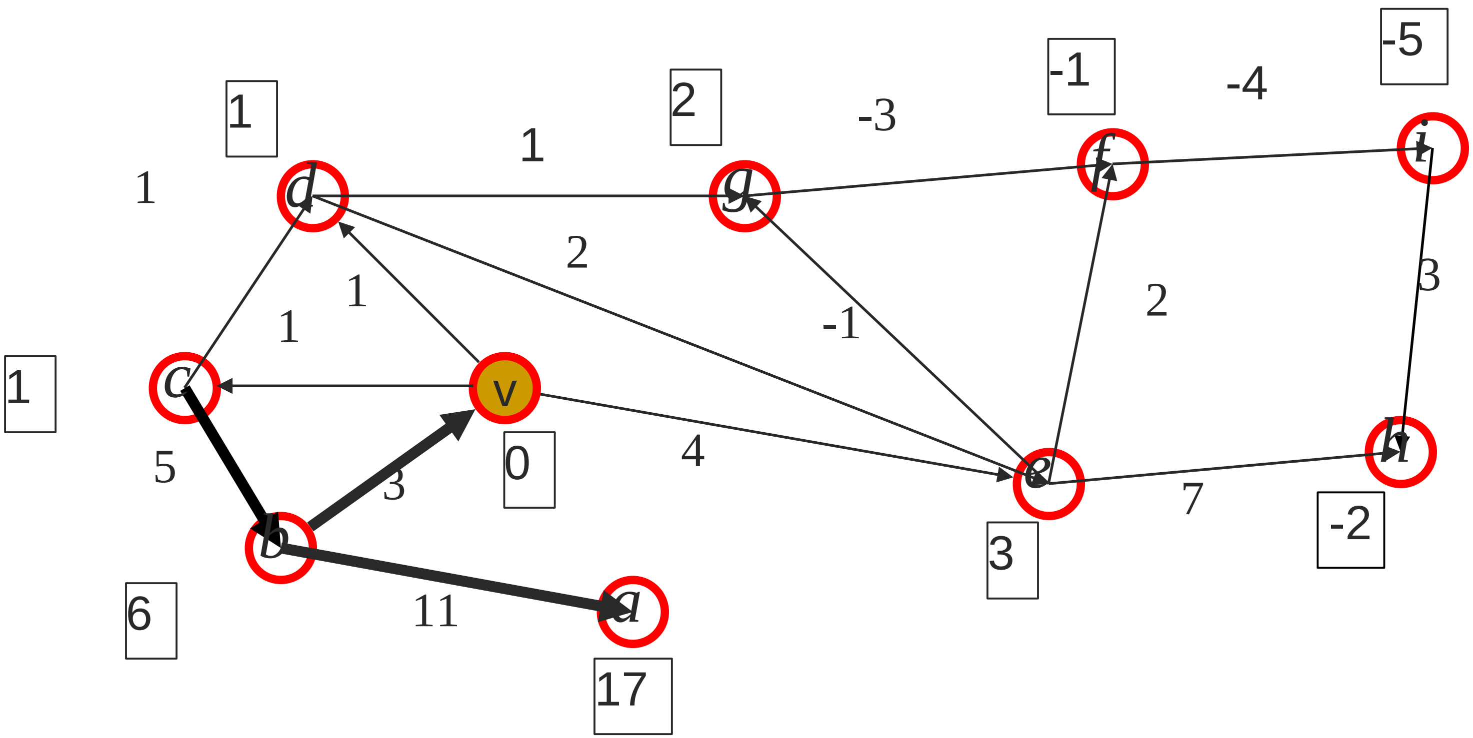
i=3



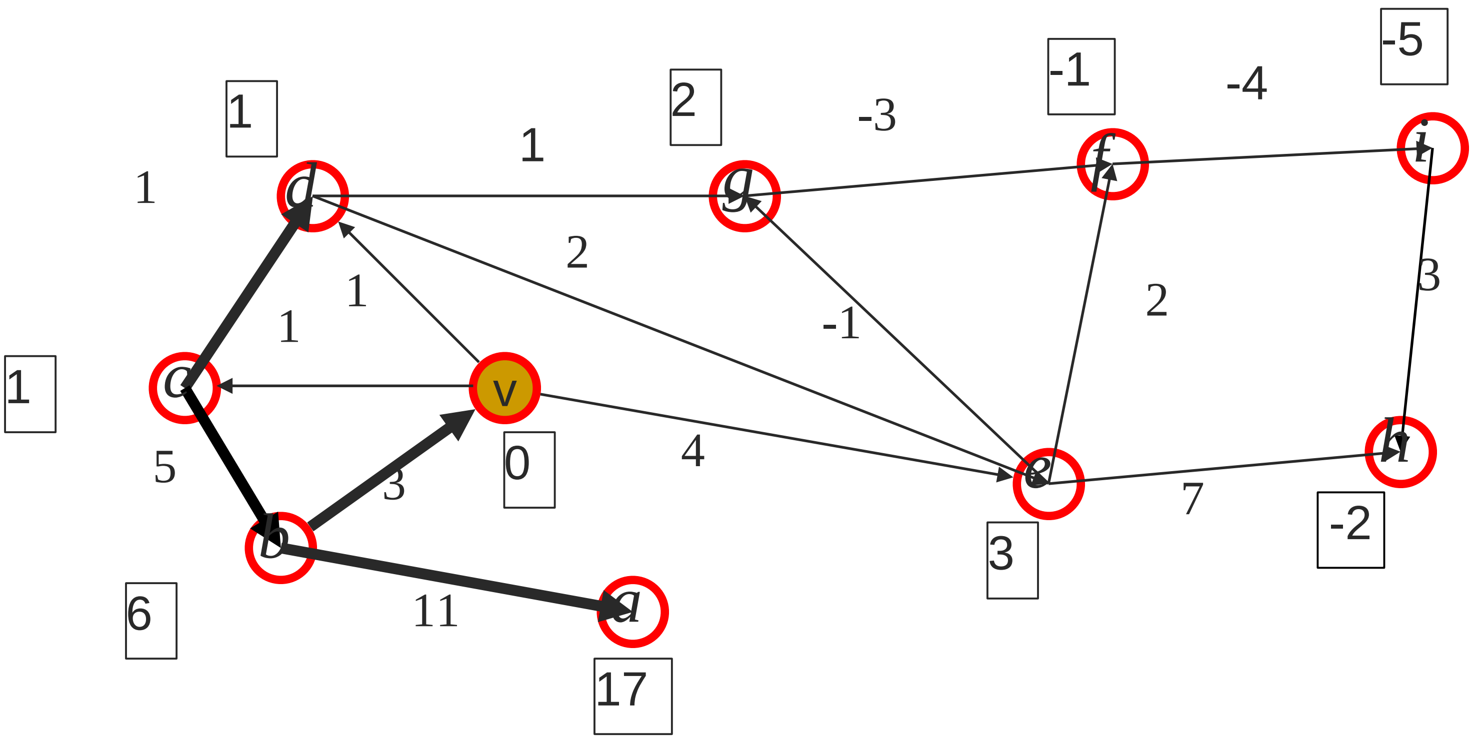
i=3



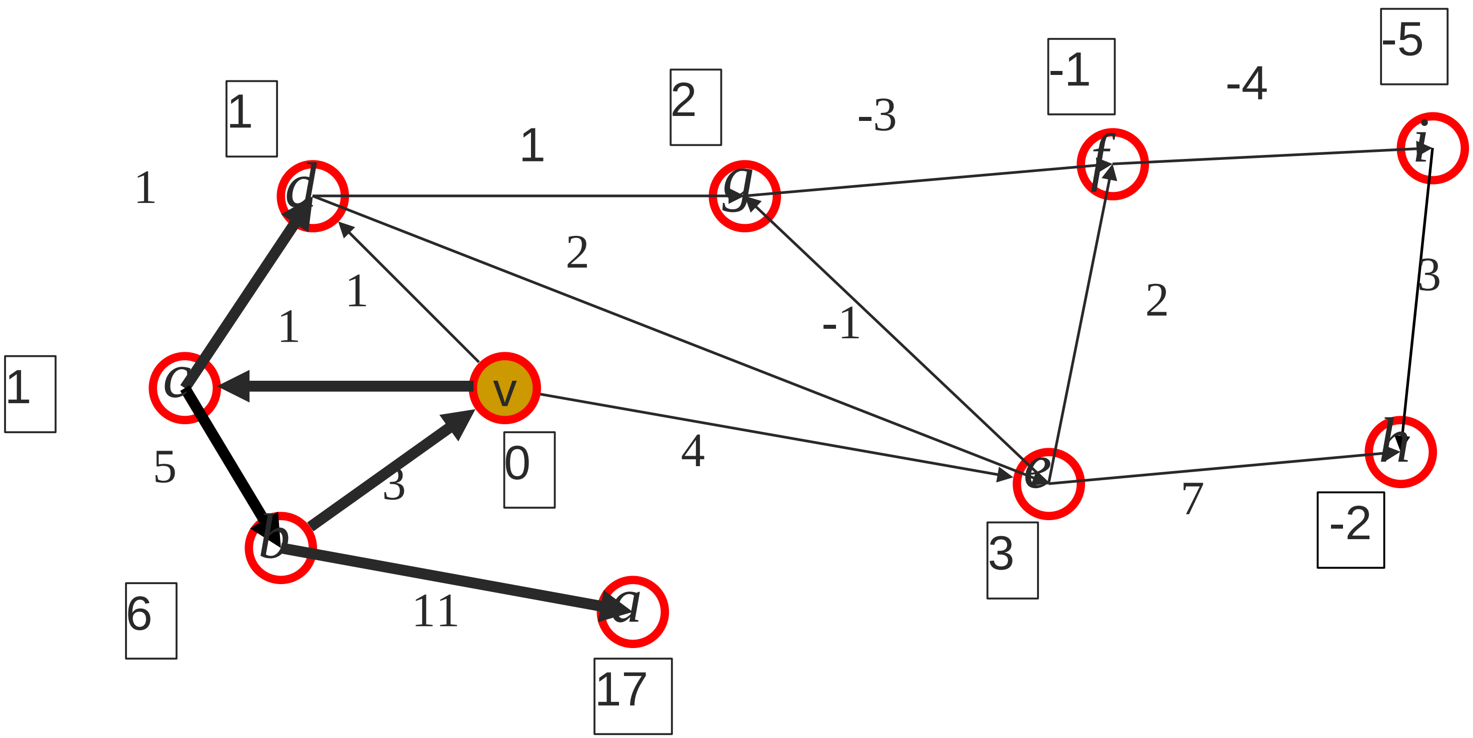
i=3



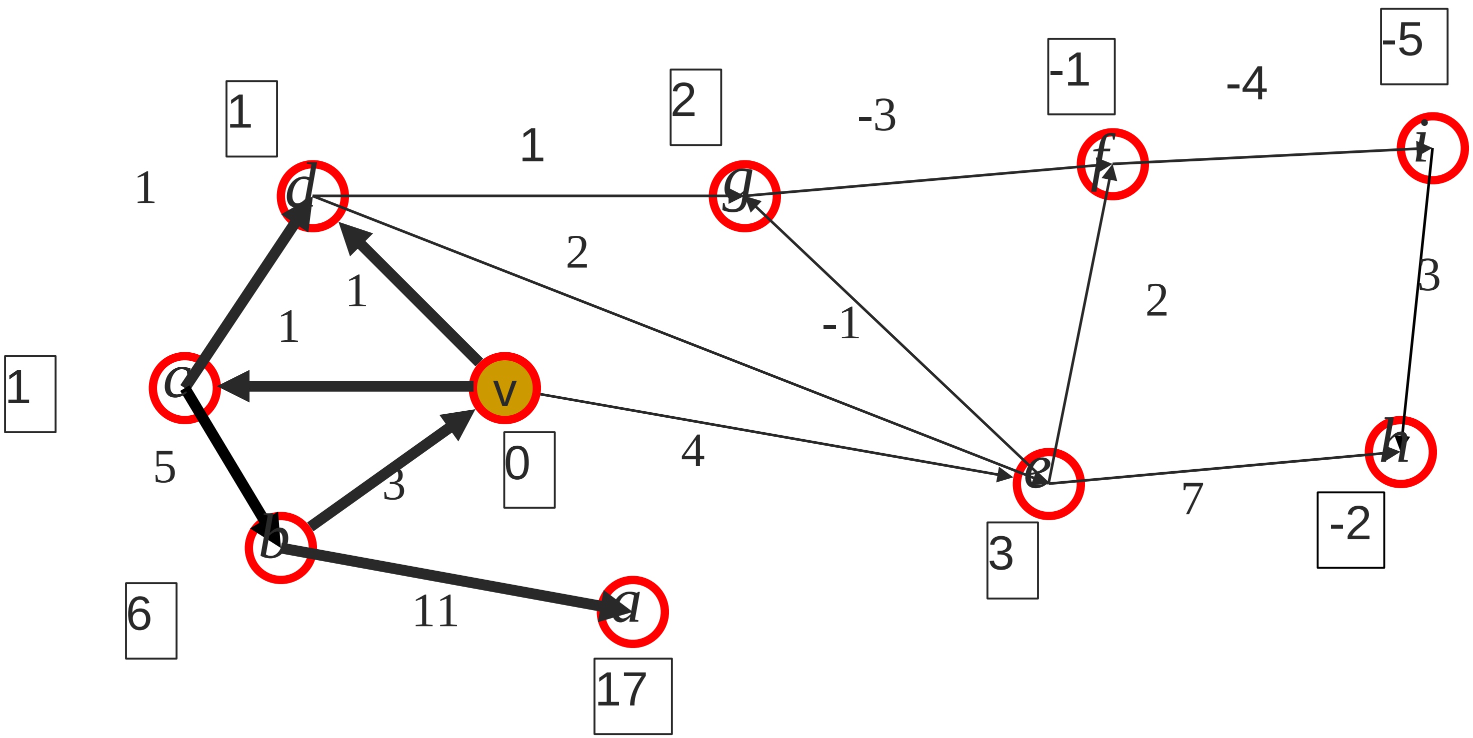
i=3



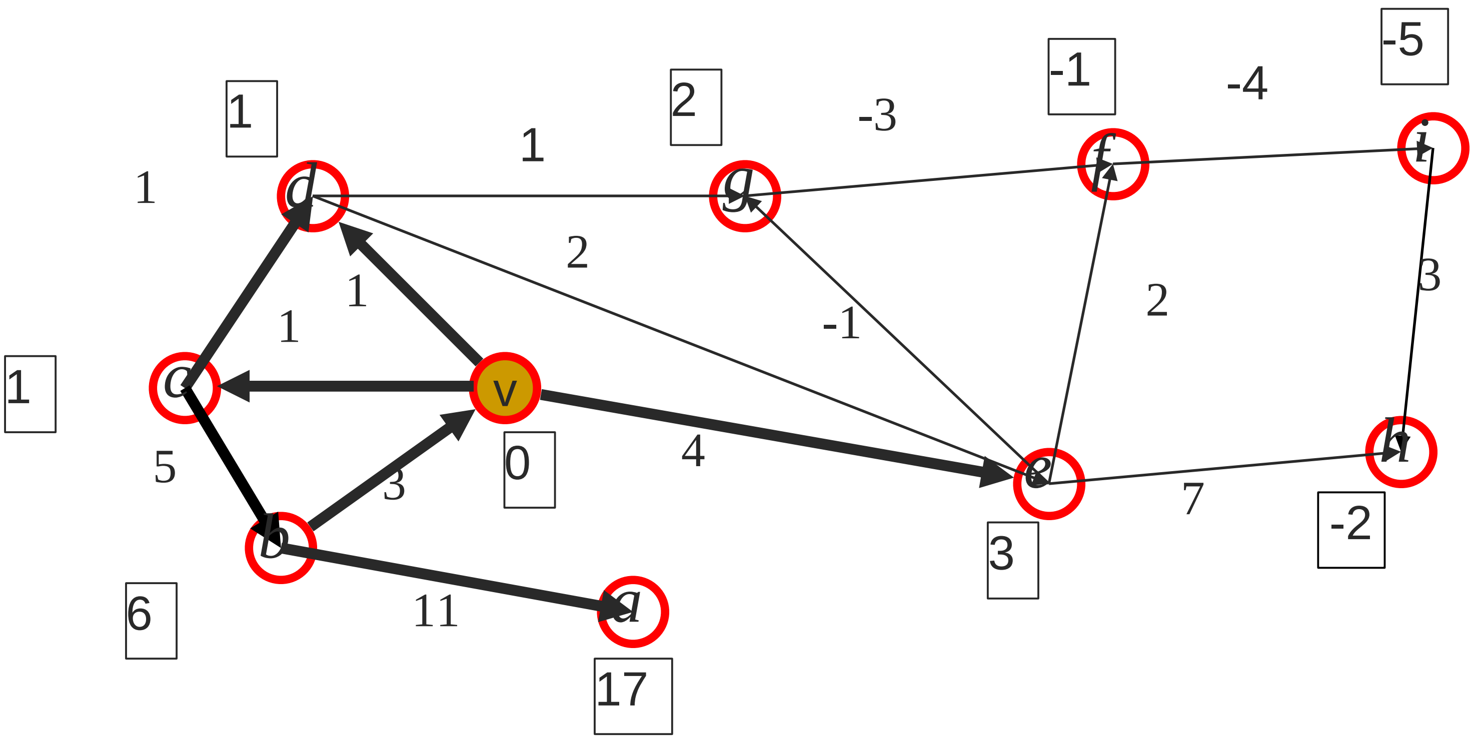
i=3



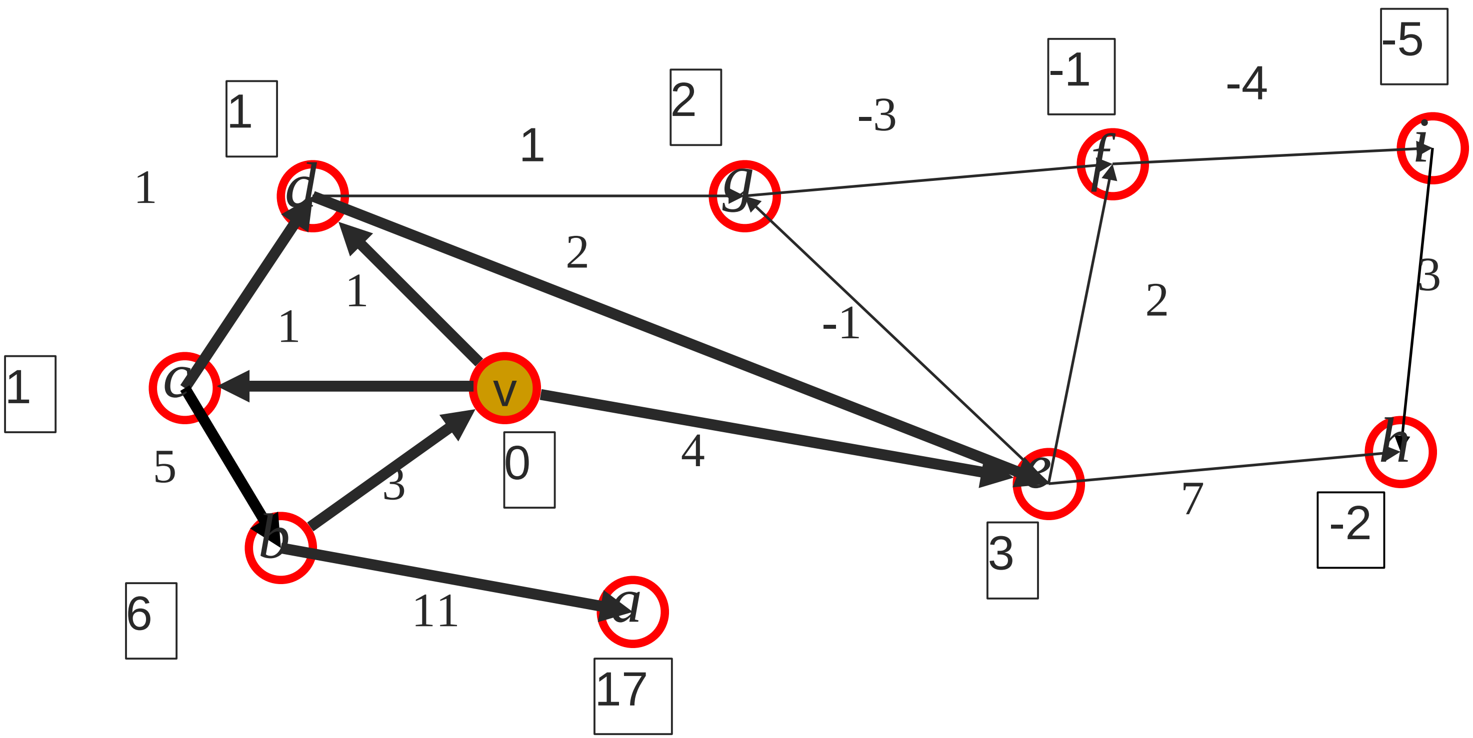
i=3



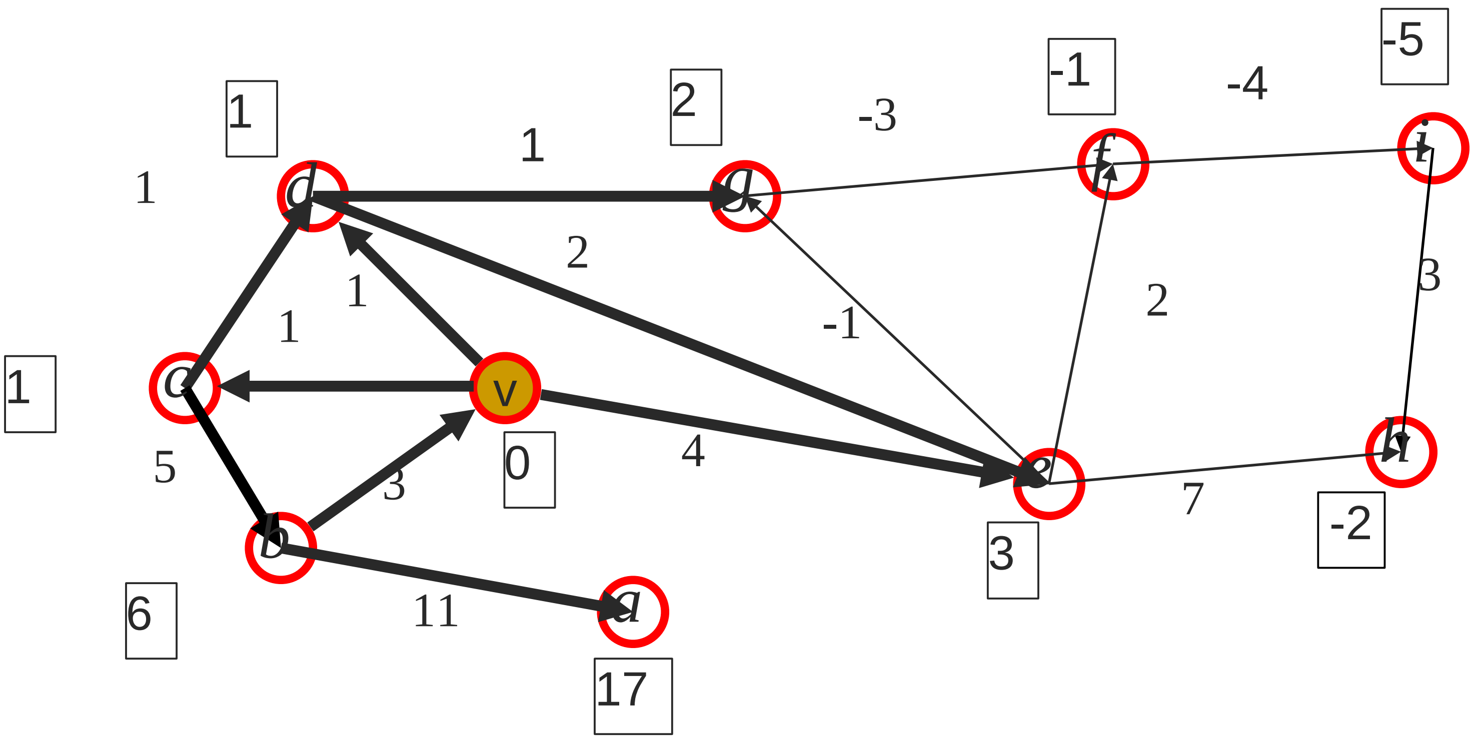
i=3



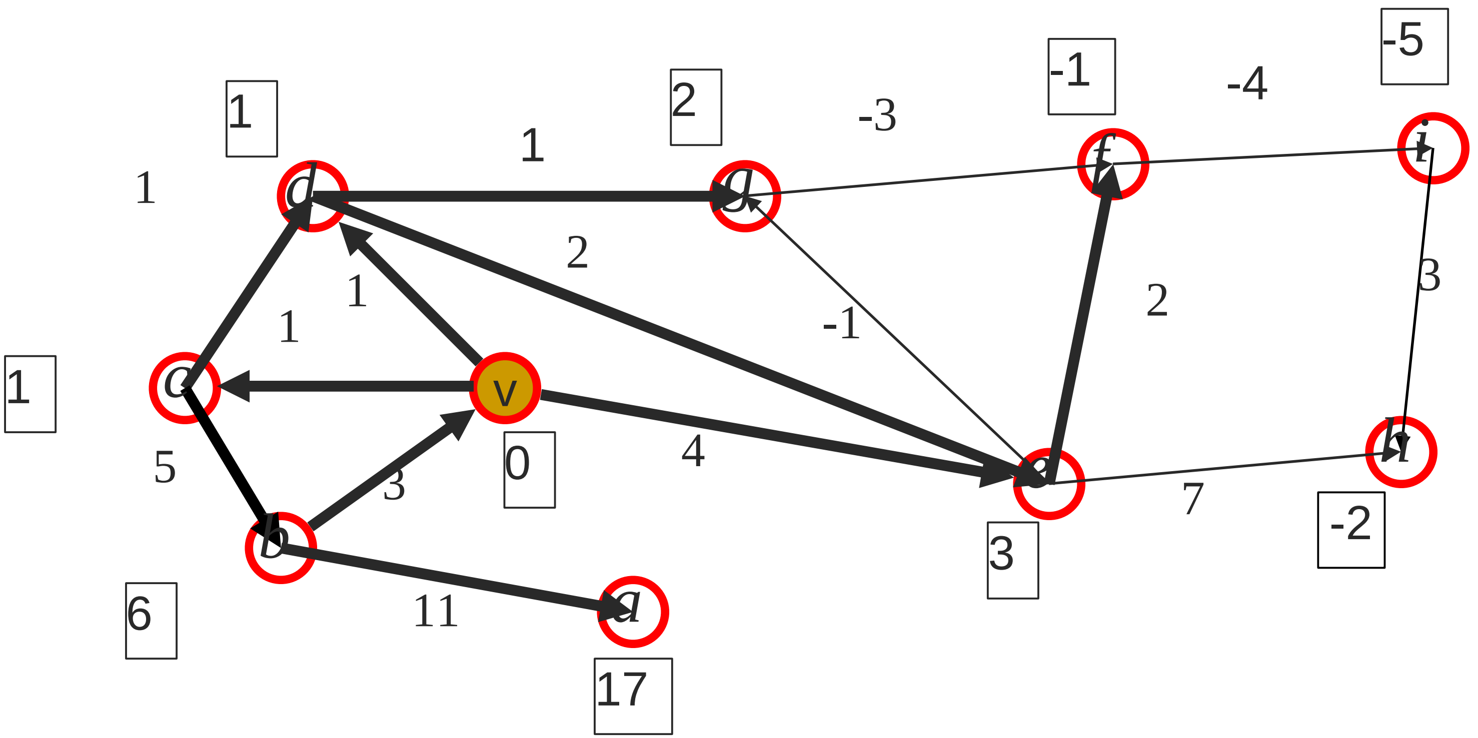
i=3



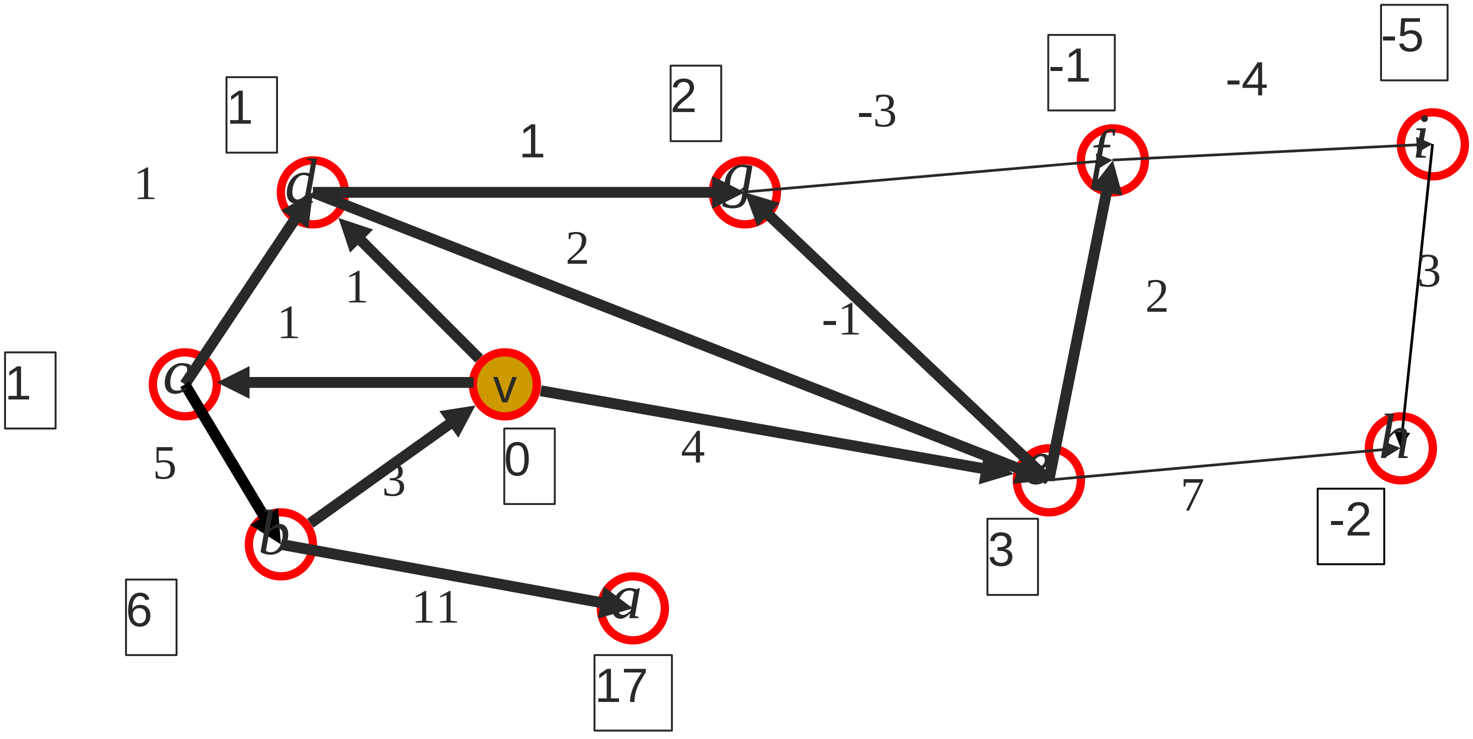
i=3



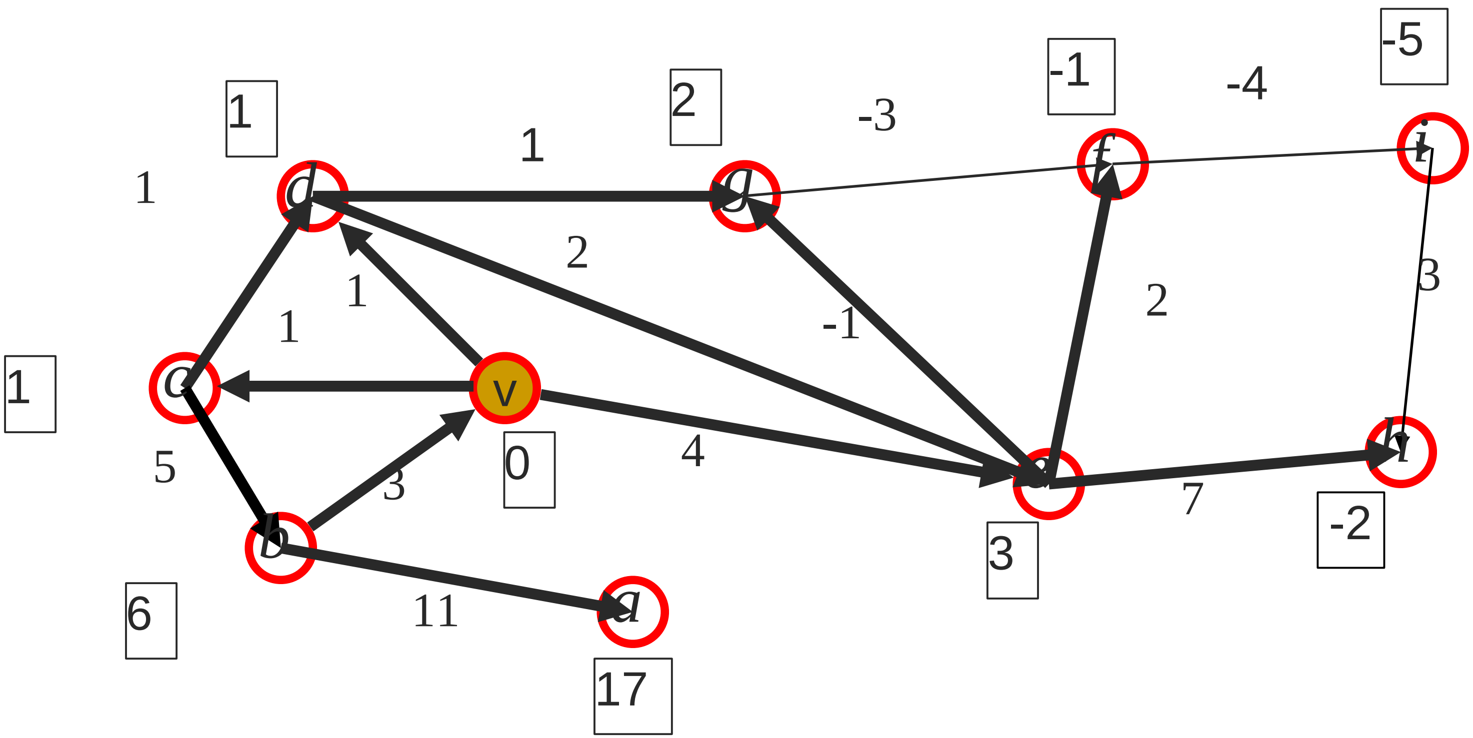
i=3



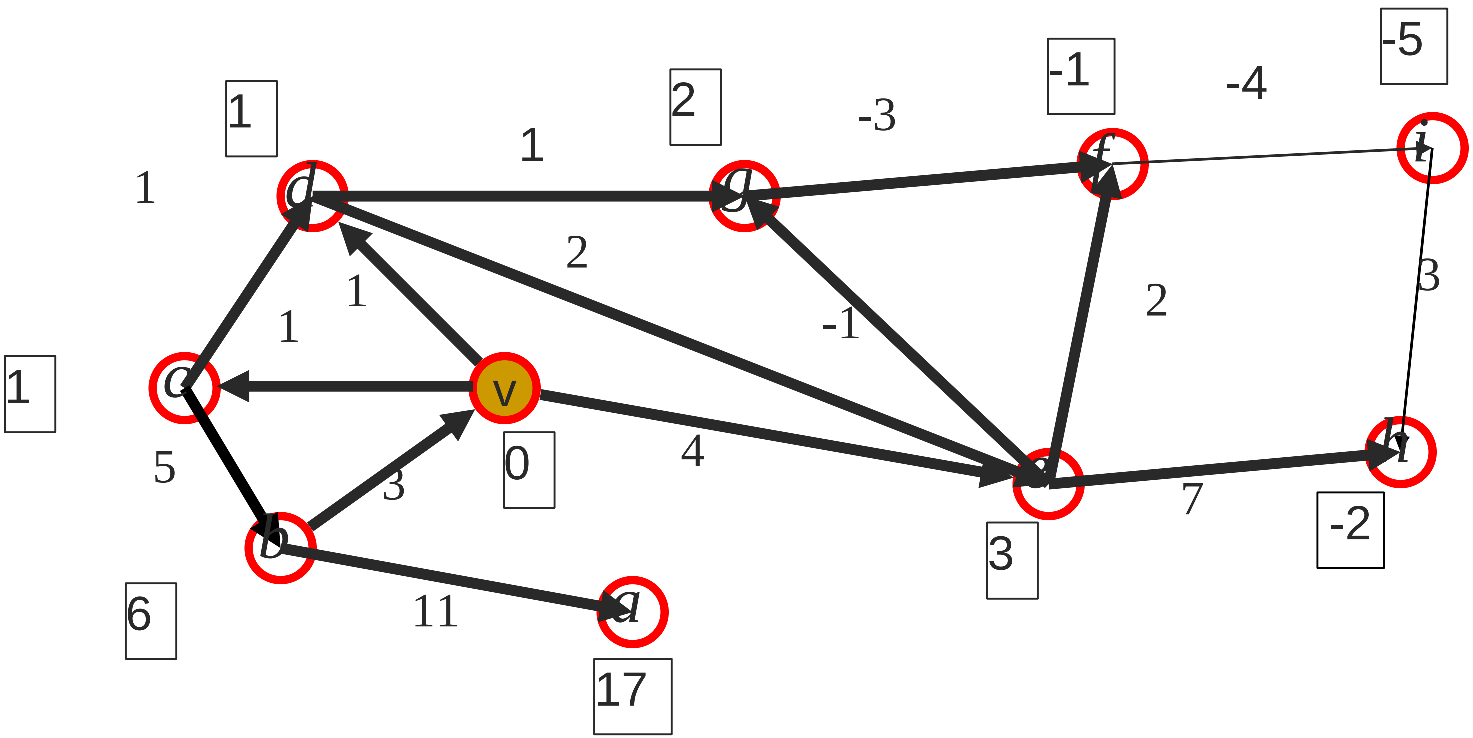
i=3



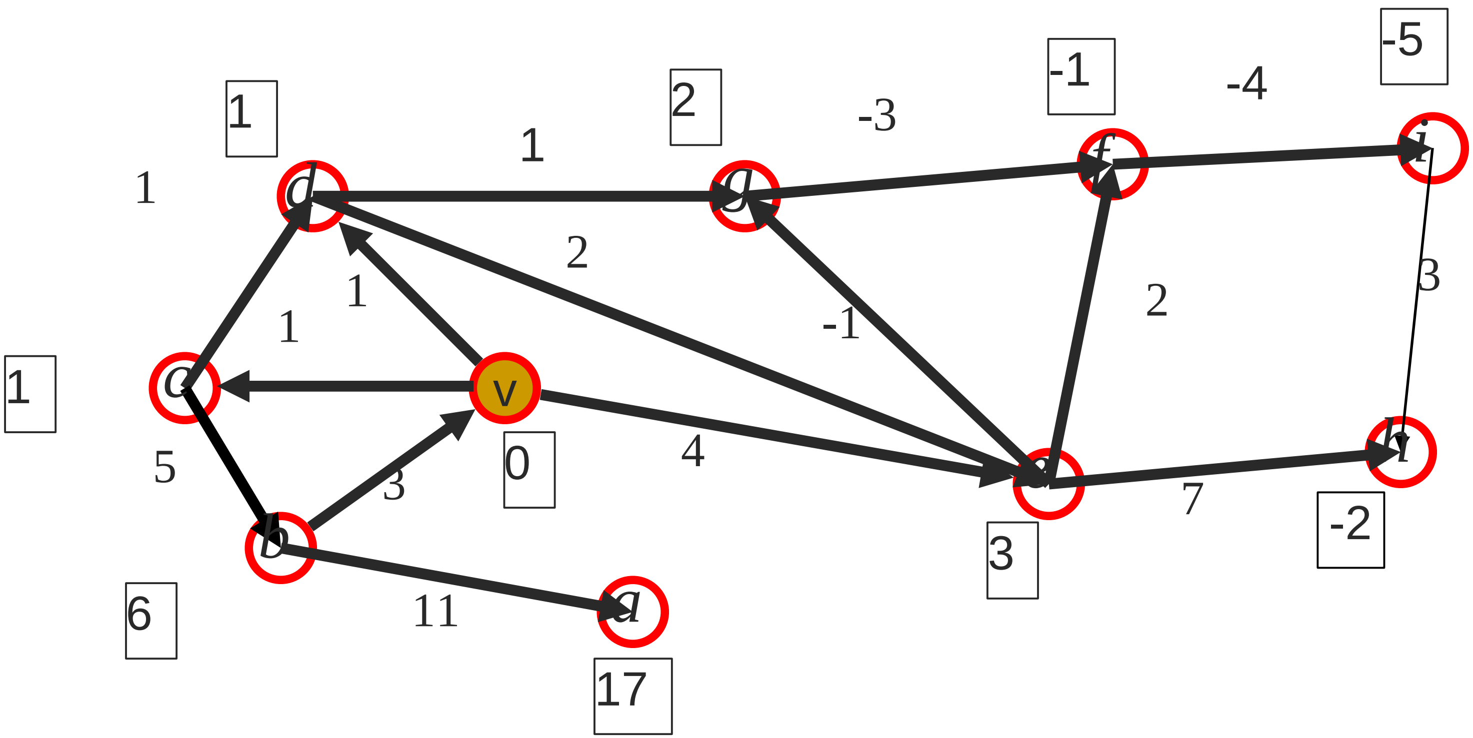
i=3



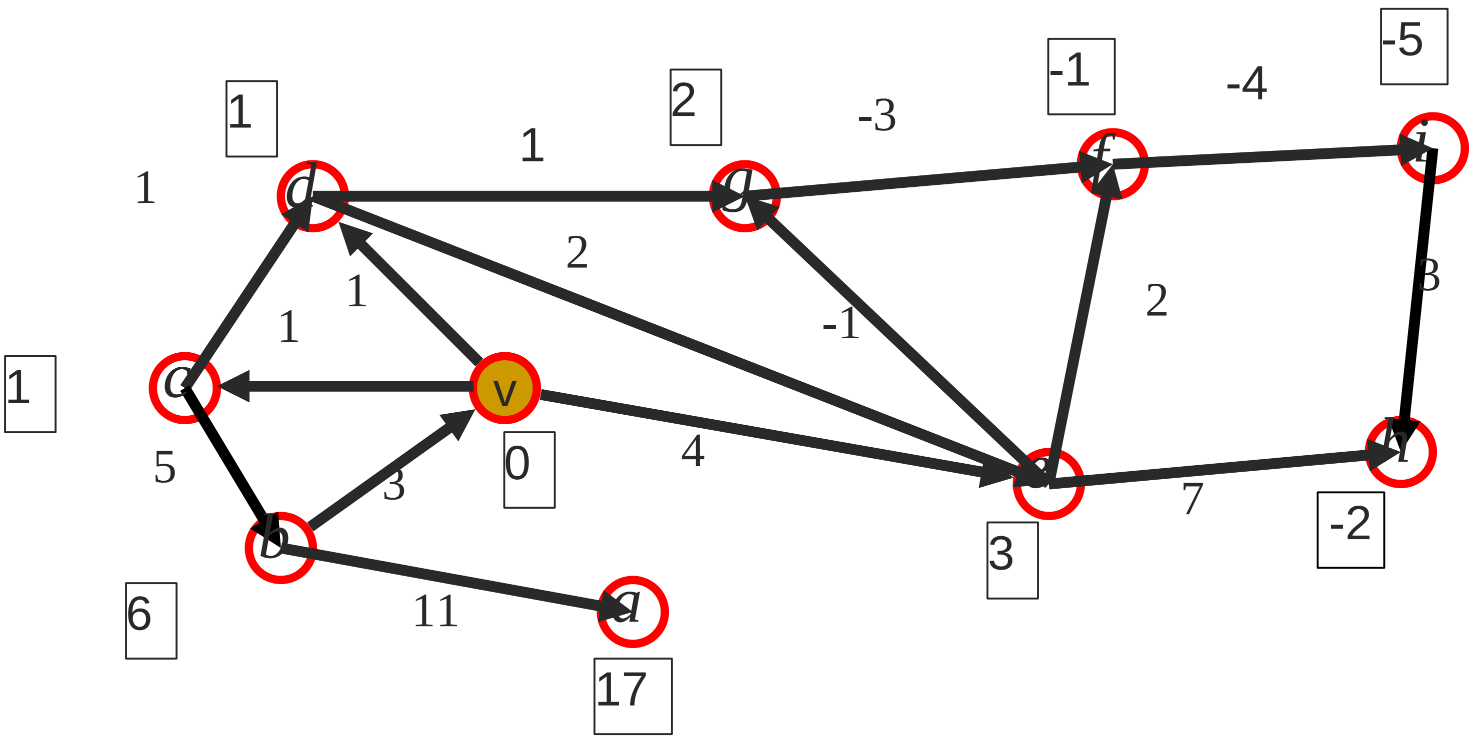
i=3



i=3



i=3



Algorithm continues until i=n-1

- In this example: $i = 9$, no more changes starting at $i = 4$

Running Time

Algorithm Bellman-Ford(G,v)

```
D[v]←0
for each vertex u ≠ v of G do
   D[u]← +∞
for i ←1 to n-1 do
   for each edge (u,z) in G do
      if D[u]+w((u,z)) < D[z] then
            D[z]←D[u]+w((u,z))
if there are no edges left with potential
   relaxation operations then
   return D
else
   return "G contains a negative cycle"
```

performs n-1 times a relaxation of every edge in the graph

Running time of Bellman-Ford algorithm

- $O(nm)$

Shortest Paths in directed <u>acyclic</u> graphs

- Can we do faster than Bellman-Ford?

All-pairs shortest paths

- For graphs with nonnegative edges
  - Run Dijkstra for each vertex (as a source).
  - n times $O(m \log n)$ is: $O(n\, m \log n)$
- For digraphs with negative edges
  - Run Bellman-Ford for each vertex (as a source).
  - n times $O(n\, m)$ is: $O(n^2\, m)$
- Use programming Dynamic Programming