Algorithm 1 BFS

```
procedure BFS(G, s)
    initialize an empty queue Q
    mark s as visited
    enqueue s into Q
    while Q is not empty do
        v ← dequeue from Q
        process(v)
        for each neighbor u of v in G do
            if u is not visited then
                mark u as visited
                enqueue u into Q
            end if
        end for
    end while
end procedure
```

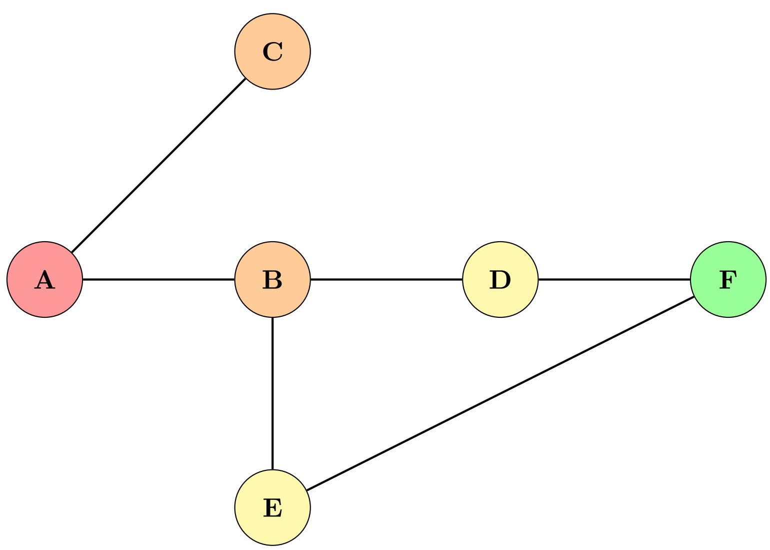
1 Shortest Path on an unweighted graph

For an unweighted graph, we know that BFS is a shortest path by the following proof

Note: This is a proof by induction. Given an unweighted graph, let us start from a node v_0 and for us to reach to any node v_i, bfs will reach it in the shortest distance. That is given any path v_0 to v_i with distance d, there must have existed a shortest path of distance $d-1$ from v_0 to v_j such that v_j to v_i is distance 1.

1. **Base Case:** For us to go from v_0 to v_0, it will only be a distance of 0. Bfs satisfies this.

2. **Induction step:** We know that the path from v_0 to v_j is the shortest path of distance d, then all unvisited neighbors of v_j must be $d+1$.



Different colors mean different distances.

2 Dijkstra

The most common modification for bfs is expanding it's properties of shortest path to a weighted (positive graph).

Algorithm 2 BFS

```
procedure BFS(G, s)
    initialize an empty priority queue Q
    mark s as visited
    enqueue s into Q
    while Q is not empty do
        v ← dequeue from Q
        process(v)
        for each neighbor u of v in G do
            if u is not visited then
                mark u as visited
                enqueue u into Q
            end if
        end for
    end while
end procedure
```

2.1 Minheap

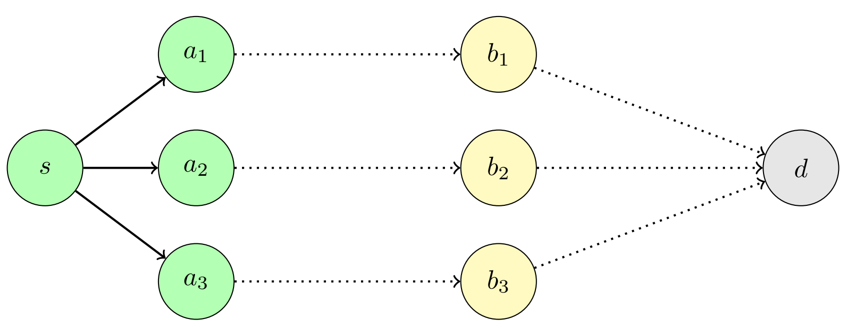
In c++, you can define a min heap with the following by telling it the type, the data structure to store it in(mainly a vector), and the comparator.

```
priority_queue<pair<int, int>,
    vector<pair<int, int>>,
    greater<pair<int, int>>
> pq;
```

2.2 Proof of correctness

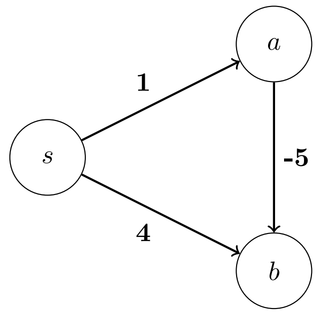
The proof of correctness for dijkstra is about the same as the proof of correctness for BFS. We can have the induction case that

1. Assume that the shortest distance from v_0 to v_i is distance d_i for some i. Then, if v_j is unvisited, the distance from v_0 to v_j is going to be the minimum from going from something already visited to v_j.



2.3 Failures on non-negative graphs

When we talk about dijkstra, we often say it fails on non-negative graphs. However, why? Looking from the proof above, we know that distance must be strictly monotonically increasing. That is if i had to visit d, i must be going through some a_i and some b_j. The distance of a_i to b_j and finally to d must be always increasing. Dijkstra is **greedy**.



It fails in the above example because the min dist from s to b is -4, but Dijkstra says its 4.

2.4 Applications and non-applications

1. **Example** Graph network problems. Bunch of routers, find the minimum distance. Example is Open Shortest Path used in routers. Actual application of dijkstra.

2. **Non-Example** Topology based graphs. If you are on a map, and there are mountains. Going up a mountain incurs some cost(postive edge) but going down returns some costs(negative edge).

3 Problems:

1. 2019 Gold https://usaco.org/index.php?page=viewproblem2&cpid=969