```
    for(auto &j: comp[i]) cout << j << ", ";
    cout << endl;
  }
  return;
}
```

Topological sort: (Kahn's Algorithm)
=> only applicable for **acyclic** graph

```
void topoSortBFS(int v, vector<vector<int>> &adj)
{
  queue<int> nodes;
  vector<int> inDeg(v + 1, 0);
  vector<int> ans;
  for (int i = 1; i <= v; i++) {
    for (int j : adj[i]) inDeg[j]++;
  }

  for (int i = 1; i <= v; i++) {
    if (inDeg[i] == 0) nodes.push(i);
  }

  while (!nodes.empty()) {
    int n = nodes.front();
    ans.push_back(n);
    nodes.pop();

    for (int i : adj[n]) {
      inDeg[i]--;
      if (inDeg[i] == 0) nodes.push(i);
    }
  }

  // If Topological Sort does not exist then the vector size will be less than the number of vertex size
  if(ans.size() < v) cout << "Topological Sort does not exist for this graph :)" << endl;
  else {
    for (auto &i: ans) cout << i << " ";
    cout << endl;
  }
}
```

Dijkstra: Used to find the shortest path between nodes in a graph with non-negative edge weights.
O((V + E) * Log(V))

```
const ll INF = LLONG_MAX;
vector<vector<pair<ll, ll>>> g;

void dijkstra(ll s, vector<ll> &d, vector<ll> &p)
{
  ll n = g.size();
  d.assign(n + 1, INF);
  p.assign(n + 1, -1);
  d[s] = 0;
  priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<pair<ll, ll>>> q;
  q.push({0, s});

  while (!q.empty())
  {
    auto [d_v, v] = q.top();
    q.pop();
    if (d_v != d[v]) continue;
    for (auto &[to, len] : g[v])
    {
      if (d[v] + len < d[to])
      {
        d[to] = d[v] + len;
        p[to] = v;
        q.push({d[to], to});
      }
    }
  }
}
vector<ll> restore_path(ll s, ll dest, vector<ll> const &p)
{
  vector<ll> path;
  for (auto v = dest; v != s && p[v] != -1; v = p[v])
                path.push_back(v);
  path.push_back(s);

  reverse(path.begin(), path.end());
  return path;
}
```

KMP :

```
vector<int> ConstructLPSarray(string pattern) // time complexity(O(n))
{
  int n = pattern.length();
  vector<int> lps(n);
  int j = 0;
  for (int i = 1; i < n;)
  {
    if (pattern[i] == pattern[j])
      lps[i] = j + 1, j++, i++;
    else
    {
      if (j != 0)   j = lps[j - 1];
      else  lps[i] = j, i++;
    }
  }
  return lps;
```