# Graph: Disjoint-set, MST, Shortest-path

## Graph representation

1. Adj matrix
    a. Directed
    b. Weighted
2. Adj list
    a. Directed
    b. Weighted

### Practice problems:

1. Print all out-going edges of a given vertex
2. Print the degree of each vertex for an undirected graph
3. Find the min degree of an undirected graph

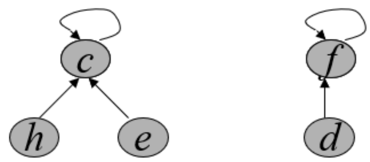
```
int* findDegree (int noOfVertices)
{
    // Traverse through row of ver and count
    int degree[noOfVertices];
    for (int i=0; i<noOfVertices; i++)
        for(int j=0; j<noOfVertices ; j++)
            if (adjacencyMatrix[i][j] == 1)
                degree[i]++;

    return degree;
}
```

```
int findMinDegree (int noOfVertices)
{
    int degree[noOfVertices];
    int minDegree = INF;
    for (int i=0; i<noOfVertices; i++)
        for(int j=0; j<noOfVertices ; j++)
            if (adjacencyMatrix[i][j] == 1)
                degree[i]++;
        if(degree[i] <= minDegree )
            minDegree = degree[i];
    return minDegree;
}
```

## Disjoint-set

- Make-Set(x) – Creates a new set {x} where x is it's only element (and therefore it is the representative of the set).
- Union(x, y) – Merges the set where x belongs with the set where y belongs. One of the elements of the merged set becomes the representative.
- Find(x) – Returns the representative of the set containing x.

## Rooted tree implementation (plain)




Set $\{c, h, e\}$  Set $\{f, d\}$

| | | |
|---|---|---|
| MAKE-SET($x$)<br>$p[x] \leftarrow x$ | UNION($x$, $y$)<br>$a$ = FIND-SET($x$)<br>$b$ = FIND-SET($y$))<br>$print(a, b)$<br>$p[a] = b$ | FIND-SET($x$)<br>**if** $x \neq p[x]$<br>  **return** FIND-SET**($p[x]$)**<br>**return** $p[x]$ |

## Rooted tree implementation (with union-by-rank and path compression heuristic):

| | | |
|---|---|---|
| MAKE-SET($x$)<br>$p[x] \leftarrow x$<br>$rank[x] \leftarrow 0$ | UNION($x$, $y$)<br>LINK(FIND-SET($x$), FIND-SET($y$))<br><br>LINK($x$, $y$)<br>**if** $rank[x] > rank[y]$<br>**then** $p[y] \leftarrow x$<br>**else** $p[x] \leftarrow y$<br>  **if** $rank[x] = rank[y]$<br>  **then** $rank[y]$++ | FIND-SET($x$)<br>**if** $x \neq p[x]$<br>  **then** $p[x] \leftarrow$**FIND-SET($p[x]$)**<br>**return** $p[x]$ |

## Practice problems:

1. Given a graph, find how many connected components are there? Print each connected-component. [Hint: use disjoint set data structure]

```
CONNECTED_COMPONENTS(G)
  for each vertex v in V[G] do
      MAKE_SET (v)

  for each edge (u, v) in E[G] do
      if FIND_SET(u) != FIND_SET(v) then
          UNION(u, v)


SAME_COMPONENT(u, v)
  if FIND_SET(u) == FIND_SET(v) then
      return TRUE
  else return FALSE
```

2. Describe a data structure that supports the following operations:
    a. find(x) – returns the representative of x
    b. union(x, y) – unifies the groups of x and y
    c. min(x) – returns the minimal element in the group of x

# MST

```
Kruskal()
{
   T = ∅;
   for each v ∈ V
      MakeSet(v);
   sort E into nondecreasing order by weight w
   for each (u,v) ∈ E (in sorted order)
      if FindSet(u) ≠ FindSet(v)
         T = T ∪ {{u,v}};
         Union(FindSet(u), FindSet(v));
}
```

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```

## Shortest path

```
BellmanFord()
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0;

    for i=1 to |V|-1
        for each edge (u,v) ∈ E
            Relax(u,v,w);

    for each edge (u,v) ∈ E
        if (d[v] > d[u] + w(u,v))
            return "no solution";
```

```
Relax(u,v,w): if (d[v] > d[u]+w(u,v))
                   then d[v]=d[u]+w(u,v)
```

```
Dijkstra(G)
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0; S = ∅; Q = V;

   while (Q ≠ ∅)
      u = ExtractMin(Q);
      S = S U {u};

      for each v ∈ u->Adj[]
         if (d[v] > d[u]+w(u,v))
            d[v] = d[u]+w(u,v);
```
X