Analysis of Graphs using Greedy Approach

Bishnu Tiwari Gx- 09

Enroll. No.- 510518009

Ph. No- 9593048305

Email id- bishnutiwari963@gmail.com

Objectives:

- Understand graphic matroid and how greedy algorithms can be used to find spanning trees for the graphs
- Learn about the disjoint set data structure and how they can be used to find connected components of a graph
- Understand Kruskal's algorithm and Prim's algorithm; implement them on graphs to find minimum spanning trees/forests
- Study the different order of addition of edges of the spanning tree in both the algorithms

Graphic Matroids:

A matroid is an ordered pair M = (S, I) satisfying the following conditions :

1. S is a finite nonempty set
2. I is a nonempty family of subsets of S, called the independent subsets of S, which is hereditary and satisfies the exchange property

A graphic matroid can be defined as an ordered pair $M_G = (S_G, I_G)$ defined in terms of a given undirected graph G = (V, E) as follows :

- The set S_G is defined to be E, the set of edges of G
- If A is a subset of E, then a belongs to I_G if and only if A is acyclic. That is, a set of edges A is independent if and only if the subgraph $G_A = (V, A)$ forms a forest

Matroids exhibit the greedy choice property as well as the optimal substructure property, therefore the application of greedy algorithms on them can help us achieve optimal solutions.

Greedy Algorithm for Weighted Matroid:

It can be observed that the determination of an optimal solution for a problem can be obtained conveniently using greedy algorithms by mapping the problem to the problem of determining the maximum weight independent subset in a weighted matroid.

The following procedure defines a greedy approach to find the maximal weight independent subset of a weighted matroid:

```
GREEDY(M, w)
    A ← Φ
    sort S[M] into monotonically decreasing order by weight w
    for each x Є S[M], taken in monotonically decreasing order by weight w(x)
            do if A U {x} Є l[M]
                    then  A ← A U {x}
            return A
```

At each step, the algorithm picks the element that has the maximum weight, and that will lead to the maximal independent subset. It works by taking advantage of the hereditary property and exchange property of a matroid. It ultimately leads to an optimal solution for the presented problem.

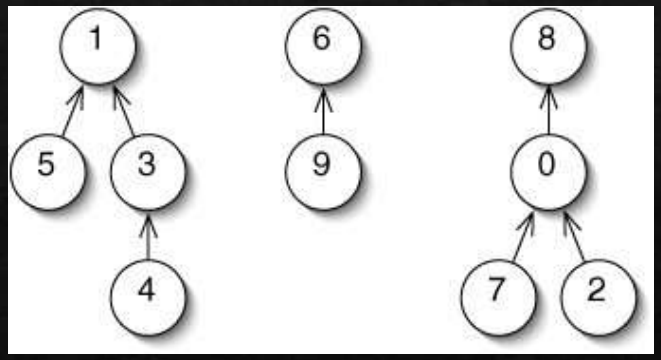
In the following slides, we will map the problem of obtaining minimum spanning tree for a connected graph.

Disjoint Set Forest:

- Disjoint set forest is a faster implementation of disjoint set data structure, where the sets are represented by rooted trees, with each node containing one member and each tree representing one set. In this structure, each member points only to its parent. The root of each tree contains the representative and is its own parent.

An example of a disjoint set forest:

The different sets of the elements are represented as different trees. Each node points to its parents. The representative elements serve as the roots of the trees and point to themselves.



The implementation of this representation is improved by adopting two heuristics:

- Union by rank : While merging any two sets, the height of the root is considered, and the smaller tree is merged with the larger tree. The rank of the root denotes the height of the tree
- Path compression : Every time we look for the position of an element in the forest, the parent of the element is changed to the representative element of the tree, thus looking for its representation next time will take O(1) time.

Implementing a disjoint set forest:

The following pseudo codes describe the procedures that help implement a disjoint set forest along with the heuristic of union by rank and path compression:

- MAKE-SET (x)
  p[x] ← x
  rank[x] ← 0
- UNION (x, y)
  LINK (FIND-SET (x), FIND-SET (y))
- FIND-SET (x)
  if x ≠ p[x]
  then p[x] ← FIND-SET (p[x])
  return p[x]

- LINK (x, y)
  if rank[x] > rank[y]
  then p[y] ← x
  else p[x] ← y
  if rank[x] = rank[y]
  then rank[y] ← rank[y]+1

The make set operation initializes the parent of each element as itself, and initializes their ranks as zero. The union operations help in making the disjoint set forest.

Connected components:

- The connected components of a graph refer to the equivalence classes of vertices under the “is reachable from” relation. A set of vertices is said to be connected if every vertex of the set is reachable from every other vertex of the set. An undirected graph is said to be connected if every vertex of the graph is reachable from every other vertex.
- The following procedure describes a way to determine the connected components of a given undirected graph G = (V, E) having no cycles, using the disjoint set forest structure :

```
CONNECTED-COMPONENTS:
    for each vertex v Є V[G]
        do MAKE-SET (v)
    for each edge (u, v) Є E[G]
        do if FIND-SET (u) ≠ FIND-SET (v)
            then UNION (u, v)
```

The procedure finds acyclic connected components of the provided graph. The separate trees of a disjoint set forest represent the different connected components of the graph.

The Minimum Spanning Tree problem:

A connected component of a graph that has all the vertices and no cycles is termed as a spanning tree of the graph. For a graphic matroid, it refers to the largest independent subset of the graph, that contains all the vertices.

- Consider a graph that has weights allotted to its edges, all positive real weights. Now, of all the spanning trees of the graph, the spanning tree that has the minimum weight, i.e., it has the edges with the minimum weights is termed as the minimum spanning tree of the graph.
- In order to obtain the minimum spanning tree, it can be mapped to a weighted matroid. The weight function of the matroid can be modified to $w'(e) = w_o - w(e)$, such that the algorithm of finding the maximal independent subset of the matroid gives us the minimum spanning tree. Here, w_o refers to the largest of all weights of the edges.

This problem can be solved by greedy algorithms, as at every point, we can make a greedy choice, that is the edge with minimum weight and that would be a part of the final minimum spanning tree. Making this choice at every point, for |V|-1 such choices, we will find the minimum spanning tree of the graph.

We will explore two algorithms to implement greedy approach to find minimum spanning trees of provided undirected weighted graphs in the following slides.

Greedy Algorithms for Minimum Spanning trees:

A generic approach to find a minimum spanning tree can be presented as:

```
GENERIC-MST (G, w)
    while A does not form a spanning tree
            do find an edge (u, v) that is safe for A
                    A ← A U {(u, v)}
            return A
```

The following two algorithms apply the greedy strategy to find the minimum spanning tree of a given graph:

- Kruskal's Algorithm : It implements the disjoint set forest data structure to arrive at the minimum spanning tree/forest
- Prim's Algorithm : It uses a priority queue to find the minimum spanning tree

Both these algorithms differ in the process of determining the safe edge to be added to the set. In Kruskal's algorithm, the safe edge chosen is always a least-weight edge in the graph that connects two distinct components. In Prim's algorithm, the minimum spanning tree is made by choosing a least-weighted edge that connects a vertex in the tree made so far to a vertex not yet in the tree

Kruskal's Algorithm:

- The approach:

Kruskal's algorithm is based directly on the generic minimum spanning tree algorithm. It finds a safe edge to add to the growing forest by finding , of all the edges connecting any two tree in the forest, an edge (u, v) of least weight.

This approach is greedy as at each step, it adds to the forest an edge of least possible weight.

The pseudo code for this approach is as follows:

```
MST-KRUSKAL (G, w)
    A ← Ø
    for each vertex v Є V[G]
        do MAKE-SET (v)
    sort the edges E in nondecreasing order by weight
    for each edge (u, v) Є E taken in nondecreasing
order by weight
        do if FIND-SET (u) ≠ FIND-SET (v)
            then A ← A U {(u, v)}
                UNION (u, v)
    return A
```

The running time of the algorithm is O(E lg E). Observing that $|E| < |V|^2$, we have, lg |E| = O(lg V), therefore the running time for the algorithm can be restated as O(E lg V).

Prim's Algorithm:

- The approach:

Prim's algorithm is a special case of the generic minimum spanning tree algorithm. Here, the tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V.

At each step, a light vertex is added to the tree A that connects A to an isolated vertex from among the vertices not yet in the tree, but will be included in the final minimum spanning tree obtained.

This strategy is greedy since the tree is augmented at each step with an edge that contributes the minimum amount possible to the tree's weight.

Here, Q represents a priority queue that contains all the vertices that aren't a part of the minimum spanning tree yet. The p array represents the parent nodes, and the key array stores the keys which help in the working of the priority queue. Adj represents the adjacency list of each vertex.

The pseudo code for this approach is as follows:

```
MST-PRIM (G, w, r)
    for each u Є V[G]
        do key[u] ← ∞
            p[u] ← NIL
    key[r] ← 0
    Q ← V[G]
    while Q ≠ Ø
        do EXTRACT-MIN (Q)
            for each v Є Adj[u]
                do if v Є Q and w (u, v) < key[v]
                    then p[v] ← u
                        key[v] ← w(u, v)
```

The running time for the algorithm depends on the way the priority queue is implemented. Upon analysis, the total time for the Prim's algorithm comes out to be O(V lg V + E lg V) = O(E lg V), which is asymptotically the same as that for the implementation of Kruskal's algorithm.

Experimental Results:

The following slides contain the application of the discussed methods on graph datasets to understand how they function

The Datasets:

- Toy dataset 1: https://docs.google.com/document/d/1yiEnKaJBd8ftOI8Gpp5gPXbRfXPp1axdATepX6dc-_U/edit?usp=sharing. The dataset has 30 vertices, 40 edges, and is a completely connected, positive weighted, undirected graph.
- Toy dataset 2: https://docs.google.com/document/d/1bT3nre1Q_5RQH5AHKfvNKetqrjB1m1HpxZi9pQ30RfQ/edit?usp=sharing. The dataset is an undirected, positively weighted graph with 30 vertices, 38 edges, and 2 connected components.
- Dataset 3: http://konect.uni-koblenz.de/networks/moreno_beach. This dataset explores contact between windsurfers. It has 43 vertices, 336 edges, and is a completely connected network.
- Dataset 4: http://konect.uni-koblenz.de/networks/moreno_lesmis. This dataset contains text analysis. It explores the co-occurences of characters in the novel Les Misérables. The graph has 77 vertices and 254 edges, and is a completely connected network.
- Dataset 5: http://konect.uni-koblenz.de/networks/moreno_train. This network explores interconnections between terrorists expected to be linked to a train bombing incident. It has 64 vertices and 243 edges and is a completely connected network.

Applying Algorithms on datasets:

- For Toy dataset 1, a single connected graph was obtained.

```
set 1:
vertices: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
edges: 40
1 2 12      26 20 1
1 29 9      20 14 29
1 9 1       14 7 2
1 15 5      22 30 1
1 23 3      30 11 2
1 10 8      11 4 3
2 8 19      11 5 4
2 9 13      4 5 5
2 3 4       4 13 6
2 17 14     5 6 7
2 16 7      6 13 8
10 15 7     13 21 9
10 18 9     13 19 10
16 23 15    21 28 11
23 17 17    21 27 12
23 24 25    27 28 13
24 17 19    27 19 14
17 12 3     19 25 15
17 18 23    29 30 27
18 26 7     12 22 17
```

These are the 40 edges making up the connected component

- The Minimum Spanning Tree was obtained using Kruskal's and Prim's algorithms. The order of addition of edges is as follows:
- The number of edges obtained equals |V|-1, thus proving that the graph obtained is a spanning tree.
- We can see that the same value for the weight of the minimum spanning tree was obtained using both the algorithms, thus showing how both the algorithms produce accurate results.
- It can also be observed how the Kruskal's Algorithm chooses edges based on the weights. But the Prim's algorithm can be seen to keep track of the tree developed so far, and every edge added is from a vertex already in the tree to a vertex not in the tree yet.

```
Kruskal's Algorithm
Number of Edges: 29
26 20 1
1 9 1
22 30 1
14 7 2
30 11 2
1 23 3
17 12 3
11 4 3
2 3 4
11 5 4
1 15 5
4 13 6
2 16 7
18 26 7
10 15 7
5 6 7
1 29 9
13 21 9
10 18 9
13 19 10
21 28 11
1 2 12
21 27 12
2 17 14
19 25 15
12 22 17
2 8 19
24 17 19
20 14 29
Weight of the Minimum Spanning Tree: 248
```

```
Prim's Algorithm
Number of Edges: 29
1 9 1
1 23 3
1 15 5
15 10 7
1 29 9
10 18 9
18 26 7
26 20 1
1 2 12
2 3 4
2 16 7
2 17 14
17 12 3
12 22 17
22 30 1
30 11 2
11 4 3
11 5 4
4 13 6
5 6 7
13 21 9
13 19 10
21 28 11
21 27 12
19 25 15
17 24 19
2 8 19
20 14 29
14 7 2
Weight of the Minimum Spanning Tree: 248
```

- For Toy dataset 2, two separate connected components were obtained using the disjoint set forest data structure:

```
Set 1:
number of vertices: 18
the vertices: 1 2 3 7 8 9 10 12 14 15 16 17 18 20 23 24 26 29
edges: 23
1 2 12
1 29 9
1 9 1
1 15 5
1 23 3
1 10 8
2 8 19
2 9 13
2 3 4
2 17 14
2 16 7
10 15 7
10 18 9
16 23 15
23 17 17
23 24 25
24 17 19
17 12 3
17 18 23
18 26 7
26 20 1
20 14 29
14 7 2
```

```
Set 2:
vertices: 12
the vertices: 4 5 6 11 13 19 21 22 25 27 28 30
edges: 15
22 30 1
30 11 2
11 4 3
11 5 4
4 5 5
4 13 6
5 6 7
6 13 8
13 21 9
13 19 10
21 28 11
21 27 12
27 28 13
27 19 14
19 25 15
```

- In this case, there are two connected components. Applying Kruskal's algorithm on the graph will give us a minimum spanning forest rather than a tree, with the number of edges = |V| - (no. of connected components)

- Prim's algorithm cannot be applied for the given graph as it is not fully connected, and Prim's algorithm works only for fully connected graphs. This is because the algorithm selects an arbitrary vertex and grows from there to the minimum spanning tree, assuming at every step that the vertex lies in the resulting MST.

```
Minimum Spanning Forest using Kruskal's algoritm:
vertices: 30
the vertices: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28 29 30
edges: 28
26 20 1
1 9 1
22 30 1
14 7 2
30 11 2
1 23 3
17 12 3
11 4 3
2 3 4
11 5 4
1 15 5
4 13 6
2 16 7
18 26 7
10 15 7
5 6 7
1 29 9
13 21 9
10 18 9
13 19 10
21 28 11
21 27 12
1 2 12
2 17 14
19 25 15
2 8 19
24 17 19
20 14 29

The weight of the Minimum Spanning Forest: 231
```

- We will now apply Kruskal's and Prim's algorithm on the largest connected components for the given graph, i.e., the graph with 18 vertices.
- The difference in the sequence of addition of edges in the minimum spanning tree will be observed.
- We can see that the same value for the weight of the minimum spanning tree was obtained using both the algorithms, thus showing how both the algorithms produce accurate results for this component of the graph.
- The difference in the working of Kruskal's and Prim's algorithm is evident.

```
Kruskal's Algorithm:

number of vertices: 18
the vertices: 1 2 3 7 8 9 10 12 14 15
16 17 18 20 23 24 26 29

edges: 17
1 9 1
26 20 1
14 7 2
1 23 3
17 12 3
2 3 4
1 15 5
10 15 7
18 26 7
2 16 7
10 18 9
1 29 9
1 2 12
2 17 14
24 17 19
2 8 19
20 14 29

The weight of the M.S.T is: 151
```

```
Prim's Algorithm:

number of vertices: 18
the vertices: 1 2 3 7 8 9 10 12 14 15
16 17 18 20 23 24 26 29

edges: 17
1 9 1
1 23 3
1 15 5
15 10 7
1 29 9
10 18 9
18 26 7
26 20 1
1 2 12
2 3 4
2 16 7
2 17 14
17 12 3
2 8 19
17 24 19
20 14 29
14 7 2

The weight of the M.S.T is: 151
```

- As the rest of the datasets are too large for the results to be presented in the slides, links have been provided to the documents containing the results:
- For dataset 3,
  - The determination of connected components returned just one set, showing the graph is a fully connected one. The connected component can be found here: https://docs.google.com/document/d/1yv0u61yOH_Y9uL7qetXbKlXL-ZoiyjK8Qnnx7gMowRs/edit?usp=sharing
  - The result of application of the algorithms for determining the minimum spanning tree is as follows: https://docs.google.com/document/d/1Xo3T8j2YY2Sy0h9zNJmO3Uv3HWMJNp4B4FkkykOHOOQ/edit?usp=sharing
- For dataset 4,
  - It was again found to be one set, the edges and vertices of the component can be found here: https://docs.google.com/document/d/1N7-czMTfhc9OqzSFfHOcBkf2BiSkME2mAOgFLmmarGU/edit?usp=sharing
  - The order of edges added by the algorithms can be found here: https://docs.google.com/document/d/1FBJo--7Xsi9QvCZVuMiegBsHz4ORQ0xFhHHalLAmt10/edit?usp=sharing
- For dataset 5,
  - This graph also turned out to be a fully connected graph. The detains can be found here: https://docs.google.com/document/d/1eI_NLmClcIOlCh2TAfHs3vDOstjWFtkw_fIY-4nXivM/edit?usp=sharing
  - The comparison of the two algorithms can be found here: https://docs.google.com/document/d/1Z0VnomUey5KjR135kJj6mgvNB1Gr4Sp0XYN44AimwBM/edit?usp=sharing

Conclusion:

In this assignment we have explored what are matroids and how they can be useful in the application of greedy algorithms. Then, we have seen how graphs can be mapped to matroids, and how the problem of finding the minimum spanning tree of a given graph can be mapped to matroids. This mapping enables us to apply greedy approaches to solve the problem.

Then we go on to explore two popular greedy algorithms, namely Kruskal's algorithm and Prim's algorithm, to solve the problem of finding the minimum spanning tree. Both these methods employ greedy strategy to arrive at the solution. While Kruskal's algorithm employs disjoint set forest, Prim's algorithm uses a priority queue. The two approaches differ, and that is explored by applying them on different datasets. The order in which the edges are added differ in both the algorithms.

It can be observed that while Kruskal's algorithm chooses edges arbitrarily based on minimum weights, Prims algorithm chooses a vertex, and divides the graph into two sets, one that will constitute the MST, and the other that is still to be added to the tree. This means that for Prim's algorithm, the graph will have to be a single connected component, but Kruskal's algorithm can also give a minimum spanning forest. All these differences can be observed in the results obtained by applying the algorithms on the graphs in this assignment.

The complexity of both the algorithms depends on the internal data structure used. Adopting the most efficient implementation of the internal data structures enables both the algorithms to work with the same complexity of O(E lg V). This completes our analysis of application of greedy strategy on graphs.

Thank You!