CSCE 629
Analysis of Algorithms
Project Report
Due Date : 2nd December, 2014

Submitted By:
Deep Desai
124001412

Implementation:

Algorithms:

1) Max Bandwidth path using Dijkstra's algorithm

```
a)  for each vertex v = 1 to n do
        status[v] = unseen
b)  status[s] = intree
    d[s] = 0
    dad[s] = -1
c)  for each edge[s,v] do
        status[v] = fringe
        dad[v] = s
        d[v] = wt(s,v)
d)  while there are fringes do
        let v be the fringe with max(d[v])
        // For array it goes through all the d[v]
        //To find out the max for heap it does delMax()
        status[v] = intree
        for each edge[v,t] do
            if status[t] == unseen
            then    status[t] = fringe
                    dad[t] = v
                    d[t] = min(d[v], wt(v,t))
            else if status[t] == fringe and d[t] < min(d[v], wt(v,t))
            then    d[t] = min(d[v], wt(v,t))
                    dad[t] = v
```

2) Max Bandwidth path using Kruskal's Algorithm

```
a)  Sort all edges in decreasing order
b)  T = ϕ
c)  for each edge e_i = [v_i, w_i] do
        r1 = find(v_i)
        r2 = find(w_i)
        if r1 != r2
        then    T = T + e_i
                union(r1, r2)
d)  return T
```

3) Union-Find

Union(s1, s2)

```
a)  if rank[s1] > rank[s2]
    then    dad[s2] = s1
            rank[s1] += rank[s2]
    else
    then    dad[s1] = s2
            rank[s2] += rank[s1]
```

```
Find(i)
a)      s = stack
b)      while (i != dad[i]) do
                s.push(i)
                i = dad[i]
c)      while (s not empty) do
                dad[s.pop()] = i
d)      return i
```

I have implemented this project in JAVA.

The flow of the program is as given below:

MainProgram -> GraphGenerator -> DoStuffForSparse, DoStuffForDense {5 times each}
DoStuffForSparse and DoStuffForDense functions take an EdgeWeightedGraph as an argument, then it adds a path from s to t that goes through all vertices in the graph and then makes instances of MaxBandwidthPathHeapDijkstras, MaxBandwidthPathArrayDijkstras and MaxBandwidthKruskal class one by one.

Implementation of Graph:

class EdgeWeightedGraph (class for Graph, each graph is an instance of this class)
class Edge (class for Edges, each edge is an instance of this class)
class SET (each list in the adjacency list is an instance of this class)
Edge weights are assigned random weight from 0 to 9999

Implementation of making Graph:

```
—>      for i = 0 to V do          //V = no. of vertices
                make integer array arrlist[V]
                for j = 0 to V do
                        if degree[j] < density and j != i
                        then    add j to arrlist
                left = density - degree[i]          //density = 6 or 20%(V)
                n = sizeof(arrlist)
                for k = 0 to n or left > 0 do //till left > 0 or whole arrlist (if n < left)
                        find random number r in between 0 to n-k-1
                        swap(arrlist[r], arrlist[n-k-1])
                        w = arrlist[n-k-1]
                        add edge [i, w] with random weight
                        degree[i]++
                        degree[w]++
```

Implementation of Heap:

I have used a private class Node which has variables vertex and value, value is the value of edge weight and the vertex is the corresponding vertex. I then make an array of instances of node class and implement heap using this array. I also keep a vertex to index mapping for example vi[v] = i means the index of vertex v in the heap is i, I do this to make update O(1).

Time Complexity and Performance of Algorithms:

1. Time Complexity

   A. MaxBandwidthPath using Dijkstra's Algorithm(array implementation) : **$O(n^2+m)$**
   The inner loop to find max runs n times and the outer loop can also run maximum of (n-1) times. Hence, $O(n^2)$. The inner for loop runs for total of m times in whole program. Hence, O(m). Thus overall time complexity is $O(n^2+m)$

   B. MaxBandwidthPath using Dijkstra's Algorithm(heap implementation) : **O(mlogn + nlogn)**
   The inner for loop in whole program runs for total number of edges, m, and the insert in the heap takes logn times. Hence, O(mlogn). The outer loop can run maximum of (n-1) times and the delete in heap takes O(nlogn). Hence O(nlogn). Thus overall time complexity comes to O(nlogn + mlogn)

   C. MaxBandwidthPath using Kruskal Algorithm : **O(mlog*n + mlogm)**
   Sorting m edges takes time O(mlogm) and m find operations with path compression in the for loop takes O(mlog*n) time. Hence, O(mlogm + mlog*n)

2. Performance

   A. Sparse Graph

   Kruskal < Dijkstra's(Heap) < Dijkstra's(Array)

   This is quite obvious because in sparse graph m is less, so kruskal's time complexity becomes O(mlog*n), Dijkstra's(heap) time complexity becomes O(nlogn) and Dijkstra's(array) time complexity becomes $O(n^2)$
   $O(mlog^*n) < O(nlogn) < O(n^2)$ {for n = 5000 and m = 15000}

   B. Dense Graph

   Dijkstra's(Heap) < Dijkstra's(Array) < Kruskal

   In dense graph the value of m is very high compared to n, so kruskal's time complexity becomes O(mlogm), Dijkstra's(heap) time complexity becomes O(mlogn) and Dijkstra's(array) time complexity becomes $O(n^2)$
   $O(mlogn) < O(n^2) < O(mlogm)$ {for n = 5000 and m = 2500000}

Future Improvements:

The swap operations in the Heap and HeapSort can be reduced which can improve time taken for the algorithm. Merge sort can be used instead of heap sort because Merge sort on arrays has considerably better data cache performance, often outperforming heapsort on modern desktop computers because merge sort frequently accesses contiguous memory locations (good locality of reference); heapsort references are spread throughout the heap.

Output:

```
deepdesai:~ deep$ java -jar ~/CSCE629.jar
Making Sparse Graph : 1
**********
For 2434 to 46
        KRUSKAL :- Time: 39ms Bandwidth: 7465
        DIJKSTRAS-HEAP :- Time: 101ms Bandwidth: 7465
        DIJKSTRAS-ARRAY :- Time: 201ms Bandwidth: 7465
For 4229 to 3535
        KRUSKAL :- Time: 19ms Bandwidth: 7314
        DIJKSTRAS-HEAP :- Time: 124ms Bandwidth: 7314
        DIJKSTRAS-ARRAY :- Time: 157ms Bandwidth: 7314
For 2204 to 57
        KRUSKAL :- Time: 15ms Bandwidth: 7500
        DIJKSTRAS-HEAP :- Time: 91ms Bandwidth: 7500
        DIJKSTRAS-ARRAY :- Time: 109ms Bandwidth: 7500
For 2405 to 1245
        KRUSKAL :- Time: 15ms Bandwidth: 7775
        DIJKSTRAS-HEAP :- Time: 77ms Bandwidth: 7775
        DIJKSTRAS-ARRAY :- Time: 120ms Bandwidth: 7775
For 1434 to 1243
        KRUSKAL :- Time: 24ms Bandwidth: 7724
        DIJKSTRAS-HEAP :- Time: 77ms Bandwidth: 7724
        DIJKSTRAS-ARRAY :- Time: 104ms Bandwidth: 7724
Making Sparse Graph : 2
**********
For 4312 to 2420
        KRUSKAL :- Time: 10ms Bandwidth: 7079
        DIJKSTRAS-HEAP :- Time: 28ms Bandwidth: 7079
        DIJKSTRAS-ARRAY :- Time: 98ms Bandwidth: 7079
For 640 to 1671
        KRUSKAL :- Time: 14ms Bandwidth: 6478
        DIJKSTRAS-HEAP :- Time: 28ms Bandwidth: 6478
        DIJKSTRAS-ARRAY :- Time: 82ms Bandwidth: 6478
For 2899 to 1413
        KRUSKAL :- Time: 16ms Bandwidth: 7952
        DIJKSTRAS-HEAP :- Time: 30ms Bandwidth: 7952
        DIJKSTRAS-ARRAY :- Time: 104ms Bandwidth: 7952
For 4191 to 447
        KRUSKAL :- Time: 8ms Bandwidth: 7403
        DIJKSTRAS-HEAP :- Time: 28ms Bandwidth: 7403
        DIJKSTRAS-ARRAY :- Time: 94ms Bandwidth: 7403
For 3341 to 3743
        KRUSKAL :- Time: 44ms Bandwidth: 6007
        DIJKSTRAS-HEAP :- Time: 40ms Bandwidth: 6007
        DIJKSTRAS-ARRAY :- Time: 68ms Bandwidth: 6007
Making Sparse Graph : 3
**********
```

For 1376 to 855
KRUSKAL :- Time: 10ms Bandwidth: 5468
DIJKSTRAS-HEAP :- Time: 35ms Bandwidth: 5468
DIJKSTRAS-ARRAY :- Time: 95ms Bandwidth: 5468
For 4781 to 3200
KRUSKAL :- Time: 7ms Bandwidth: 7425
DIJKSTRAS-HEAP :- Time: 35ms Bandwidth: 7425
DIJKSTRAS-ARRAY :- Time: 101ms Bandwidth: 7425
For 4589 to 4672
KRUSKAL :- Time: 11ms Bandwidth: 5749
DIJKSTRAS-HEAP :- Time: 38ms Bandwidth: 5749
DIJKSTRAS-ARRAY :- Time: 100ms Bandwidth: 5749
For 3689 to 450
KRUSKAL :- Time: 10ms Bandwidth: 7540
DIJKSTRAS-HEAP :- Time: 34ms Bandwidth: 7540
DIJKSTRAS-ARRAY :- Time: 112ms Bandwidth: 7540
For 4546 to 1179
KRUSKAL :- Time: 14ms Bandwidth: 8133
DIJKSTRAS-HEAP :- Time: 39ms Bandwidth: 8133
DIJKSTRAS-ARRAY :- Time: 117ms Bandwidth: 8133
Making Sparse Graph : 4
****
For 1349 to 3925
KRUSKAL :- Time: 7ms Bandwidth: 6127
DIJKSTRAS-HEAP :- Time: 29ms Bandwidth: 6127
DIJKSTRAS-ARRAY :- Time: 66ms Bandwidth: 6127
For 1549 to 3866
KRUSKAL :- Time: 8ms Bandwidth: 6721
DIJKSTRAS-HEAP :- Time: 30ms Bandwidth: 6721
DIJKSTRAS-ARRAY :- Time: 102ms Bandwidth: 6721
For 1181 to 2642
KRUSKAL :- Time: 8ms Bandwidth: 7882
DIJKSTRAS-HEAP :- Time: 29ms Bandwidth: 7882
DIJKSTRAS-ARRAY :- Time: 110ms Bandwidth: 7882
For 586 to 780
KRUSKAL :- Time: 7ms Bandwidth: 6812
DIJKSTRAS-HEAP :- Time: 27ms Bandwidth: 6812
DIJKSTRAS-ARRAY :- Time: 89ms Bandwidth: 6812
For 4990 to 4465
KRUSKAL :- Time: 9ms Bandwidth: 7245
DIJKSTRAS-HEAP :- Time: 30ms Bandwidth: 7245
DIJKSTRAS-ARRAY :- Time: 77ms Bandwidth: 7245
Making Sparse Graph : 5
****
For 369 to 4804
KRUSKAL :- Time: 13ms Bandwidth: 6801
DIJKSTRAS-HEAP :- Time: 36ms Bandwidth: 6801
DIJKSTRAS-ARRAY :- Time: 121ms Bandwidth: 6801
For 3299 to 4275

KRUSKAL :- Time: 12ms Bandwidth: 8031
DIJKSTRAS-HEAP :- Time: 35ms Bandwidth: 8031
DIJKSTRAS-ARRAY :- Time: 122ms Bandwidth: 8031
For 2591 to 4088
KRUSKAL :- Time: 9ms Bandwidth: 5381
DIJKSTRAS-HEAP :- Time: 30ms Bandwidth: 5381
DIJKSTRAS-ARRAY :- Time: 96ms Bandwidth: 5381
For 3484 to 4237
KRUSKAL :- Time: 10ms Bandwidth: 8227
DIJKSTRAS-HEAP :- Time: 31ms Bandwidth: 8227
DIJKSTRAS-ARRAY :- Time: 106ms Bandwidth: 8227
For 275 to 1896
KRUSKAL :- Time: 8ms Bandwidth: 8300
DIJKSTRAS-HEAP :- Time: 32ms Bandwidth: 8300
DIJKSTRAS-ARRAY :- Time: 126ms Bandwidth: 8300
Making Dense Graph : 1
****
For 4589 to 3050
DIJKSTRAS-HEAP :- Time: 533ms Bandwidth: 9981
DIJKSTRAS-ARRAY :- Time: 545ms Bandwidth: 9981
KRUSKAL :- Time: 5569ms Bandwidth: 9981
For 1344 to 4968
DIJKSTRAS-HEAP :- Time: 538ms Bandwidth: 9982
DIJKSTRAS-ARRAY :- Time: 532ms Bandwidth: 9982
KRUSKAL :- Time: 6801ms Bandwidth: 9982
For 860 to 4462
DIJKSTRAS-HEAP :- Time: 557ms Bandwidth: 9981
DIJKSTRAS-ARRAY :- Time: 642ms Bandwidth: 9981
KRUSKAL :- Time: 7672ms Bandwidth: 9981
For 358 to 4144
DIJKSTRAS-HEAP :- Time: 542ms Bandwidth: 9986
DIJKSTRAS-ARRAY :- Time: 543ms Bandwidth: 9986
KRUSKAL :- Time: 7280ms Bandwidth: 9986
For 2838 to 4720
DIJKSTRAS-HEAP :- Time: 552ms Bandwidth: 9983
DIJKSTRAS-ARRAY :- Time: 632ms Bandwidth: 9983
KRUSKAL :- Time: 7368ms Bandwidth: 9983
Making Dense Graph : 2
****
For 241 to 3018
DIJKSTRAS-HEAP :- Time: 512ms Bandwidth: 9971
DIJKSTRAS-ARRAY :- Time: 550ms Bandwidth: 9971
KRUSKAL :- Time: 7551ms Bandwidth: 9971
For 4333 to 2976
DIJKSTRAS-HEAP :- Time: 500ms Bandwidth: 9984
DIJKSTRAS-ARRAY :- Time: 552ms Bandwidth: 9984
KRUSKAL :- Time: 7398ms Bandwidth: 9984
For 1188 to 2530
DIJKSTRAS-HEAP :- Time: 611ms Bandwidth: 9982

DIJKSTRAS-ARRAY :- Time: 554ms Bandwidth: 9982
KRUSKAL :- Time: 6785ms Bandwidth: 9982
For 1915 to 294
DIJKSTRAS-HEAP :- Time: 511ms Bandwidth: 9983
DIJKSTRAS-ARRAY :- Time: 551ms Bandwidth: 9983
KRUSKAL :- Time: 7298ms Bandwidth: 9983
For 108 to 3262
DIJKSTRAS-HEAP :- Time: 519ms Bandwidth: 9987
DIJKSTRAS-ARRAY :- Time: 535ms Bandwidth: 9987
KRUSKAL :- Time: 7344ms Bandwidth: 9987
Making Dense Graph : 3
****
For 3285 to 1144
DIJKSTRAS-HEAP :- Time: 517ms Bandwidth: 9984
DIJKSTRAS-ARRAY :- Time: 558ms Bandwidth: 9984
KRUSKAL :- Time: 7216ms Bandwidth: 9984
For 3029 to 4214
DIJKSTRAS-HEAP :- Time: 481ms Bandwidth: 9986
DIJKSTRAS-ARRAY :- Time: 536ms Bandwidth: 9986
KRUSKAL :- Time: 7049ms Bandwidth: 9986
For 3014 to 4970
DIJKSTRAS-HEAP :- Time: 510ms Bandwidth: 9985
DIJKSTRAS-ARRAY :- Time: 555ms Bandwidth: 9985
KRUSKAL :- Time: 7046ms Bandwidth: 9985
For 692 to 4695
DIJKSTRAS-HEAP :- Time: 509ms Bandwidth: 9973
DIJKSTRAS-ARRAY :- Time: 525ms Bandwidth: 9973
KRUSKAL :- Time: 6971ms Bandwidth: 9973
For 3167 to 3964
DIJKSTRAS-HEAP :- Time: 511ms Bandwidth: 9985
DIJKSTRAS-ARRAY :- Time: 667ms Bandwidth: 9985
KRUSKAL :- Time: 6973ms Bandwidth: 9985
Making Dense Graph : 4
****
For 3919 to 3055
DIJKSTRAS-HEAP :- Time: 511ms Bandwidth: 9981
DIJKSTRAS-ARRAY :- Time: 520ms Bandwidth: 9981
KRUSKAL :- Time: 7391ms Bandwidth: 9981
For 927 to 1961
DIJKSTRAS-HEAP :- Time: 528ms Bandwidth: 9987
DIJKSTRAS-ARRAY :- Time: 560ms Bandwidth: 9987
KRUSKAL :- Time: 6780ms Bandwidth: 9987
For 3876 to 1701
DIJKSTRAS-HEAP :- Time: 487ms Bandwidth: 9975
DIJKSTRAS-ARRAY :- Time: 509ms Bandwidth: 9975
KRUSKAL :- Time: 6998ms Bandwidth: 9975
For 1088 to 3789
DIJKSTRAS-HEAP :- Time: 510ms Bandwidth: 9987
DIJKSTRAS-ARRAY :- Time: 549ms Bandwidth: 9987

KRUSKAL :- Time: 6966ms Bandwidth: 9987
For 1331 to 1965
DIJKSTRAS-HEAP :- Time: 502ms Bandwidth: 9983
DIJKSTRAS-ARRAY :- Time: 563ms Bandwidth: 9983
KRUSKAL :- Time: 6868ms Bandwidth: 9983
Making Dense Graph : 5
****
For 3115 to 3766
DIJKSTRAS-HEAP :- Time: 484ms Bandwidth: 9980
DIJKSTRAS-ARRAY :- Time: 496ms Bandwidth: 9980
KRUSKAL :- Time: 7158ms Bandwidth: 9980
For 4495 to 1311
DIJKSTRAS-HEAP :- Time: 507ms Bandwidth: 9974
DIJKSTRAS-ARRAY :- Time: 502ms Bandwidth: 9974
KRUSKAL :- Time: 6884ms Bandwidth: 9974
For 4754 to 4885
DIJKSTRAS-HEAP :- Time: 484ms Bandwidth: 9980
DIJKSTRAS-ARRAY :- Time: 514ms Bandwidth: 9980
KRUSKAL :- Time: 8622ms Bandwidth: 9980
For 121 to 3670
DIJKSTRAS-HEAP :- Time: 517ms Bandwidth: 9984
DIJKSTRAS-ARRAY :- Time: 551ms Bandwidth: 9984
KRUSKAL :- Time: 6907ms Bandwidth: 9984
For 911 to 2245
DIJKSTRAS-HEAP :- Time: 499ms Bandwidth: 9985
DIJKSTRAS-ARRAY :- Time: 562ms Bandwidth: 9985
KRUSKAL :- Time: 7478ms Bandwidth: 9985