CZ2101 Example Class 2
Dijkstra's Algorithm

Yi Hao, Sahithya, Joel and Saori

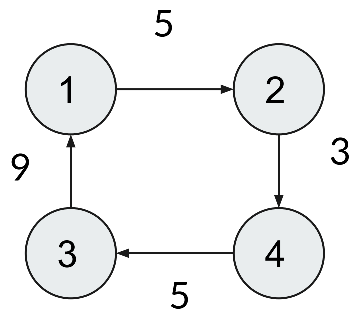
Table of contents

Dijkstra’s Algorithm

- Find the shortest paths between **source vertex** and **other <u>connected</u> vertices** in a graph
- Works on non-negative edge weights only
- Implemented via Adjacency Matrix and Adjacency List

Dijkstra's Algorithm

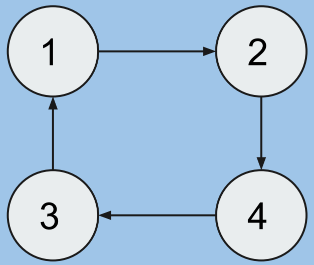
Iteration	Vertex 1 (Source)	Vertex 2	Vertex 3	Vertex 4
1	**0**	∞	∞	∞
2	**0**	**5**	∞	∞
3	**0**	**5**	**5+3=8**	∞
4	**0**	**5**	**8**	**8+9=17**



Input Graphs for Dijkstra’s Algorithm

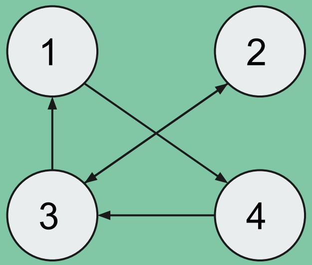
Best

One Path



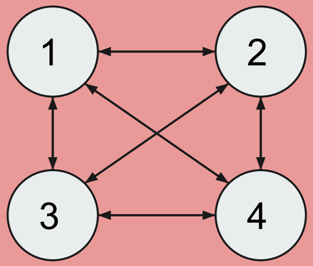
Average

Random



Worst

Complete



Adjacency Matrix/Priority Queue (Array) Implementation

Pseudocode

```
void dijkstra(int adj_matrix[][], int d[], int pq[]){

    while S has vertices with unfinalized distances:

        int u = minDistance(d, pq, vertices);

        set u to be finalized;

        for each v adjacent to u:

            if(d[v] > d[u] + adj_matrix[u][v]){

                set d[v] = d[u] + adj_matrix[u][v]

            }

    return;

}
```

```
int minDistance(int d[], int pq[],int sz){
    int lowest = ∞, min_index=0;
    for(int i=0; i<sz; i++){
        if(pq[i] == 0 && d[i] <= lowest){
            lowest = d[i];
            min_index= i;
        }
    }
    return min_index;
}
```

Time & Space Complexity (Theoretical)

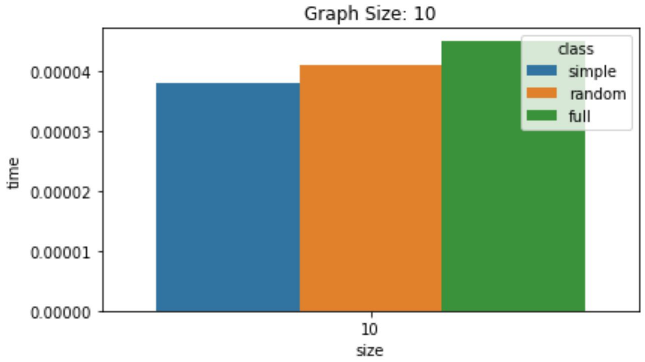
	Best/Average/Worst
Array Priority Queue/ minDistance()	$\theta(V)$
Set U to be finalized	$O(1)$
Loop over each v adjacent to u	$\theta(V)$
Total	$\theta(V) * (2\theta(V) + 1) = \theta(V^2)$

Adjacency Matrix: Space Complexity: O(V^2)

Time Complexity (Empirical)

At n = 10,

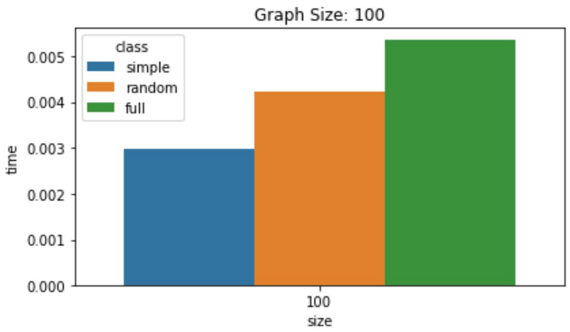
	Best (One Path)	Average (Random)	Worst (Complete)
Time Taken (in s)	0.000038	0.000041	0.000045



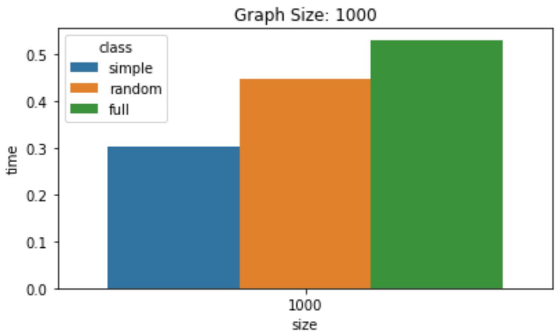
Time Complexity (Empirical)

No.of vertices

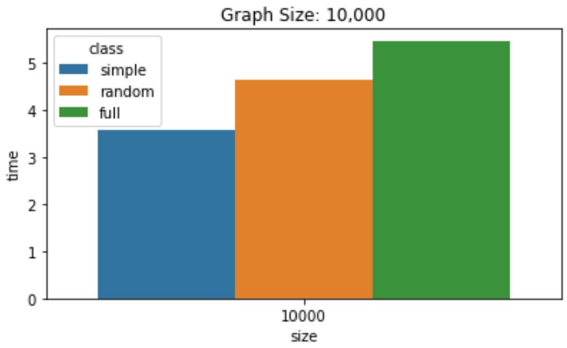
At V = 100



At V = 1000



At V = 10,000



Adjacency List/Priority Queue (MinHeap) Implementation

Pseudocode

```
void dijkstra(Graph G,int src)
{
     for(each vertex v in G)
    {
         d[v] = infinity;
    }
     d[s] = 0;
     enqueue all nodes in priority queue Q;
     while(Q != NULL)
    {
         u = extractMin(Q);
         for(each vertex v adjacent to u)
        {
              if(d[u] + w[u,v] < d[v])
             {
                  d[v] = d[u] + w[u,v];
                  decreaseKey(Q,v,d[v]);
             }
        }
    }
}
```

Time & Space complexity (theoretical)

Average case:

$Number\ of\ vertices * Cost\ of\ extracting\ \min\ Vertex + Number\ of\ edges * Cost\ of\ decreasing\ weight$

$= O(V * \log V) + O(E * \log V)$

$= O[(V + E) \log V]$

Worst case: $E = O(V^2)$

$O(V * \log V) + O(V^2 * \log V)$

$= O(V^2 \log V)$

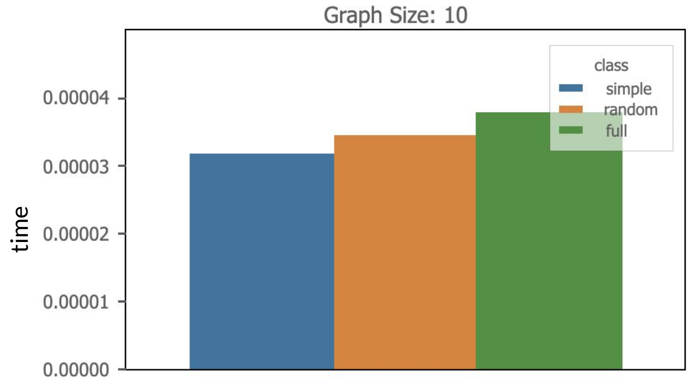
$= O(E * \log V)$

Space Complexity: O(V+E)

Time Complexity (Empirical)

At V = 10,

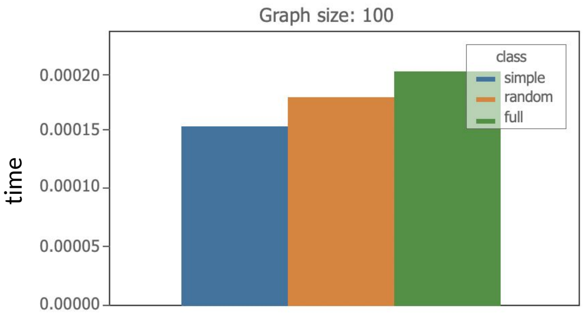
	Best (One Path)	Average (Random)	Worst (Complete)
Time Taken (in s)	0.0000329	0.0000364	0.0000388



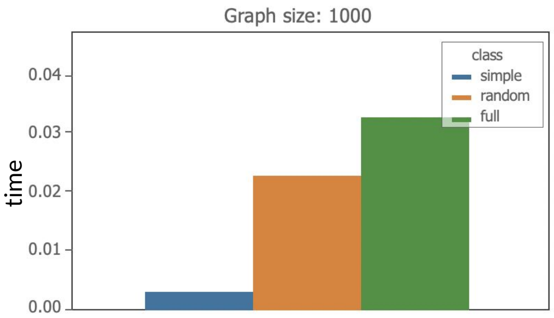
Time Complexity (Empirical)

No.of vertices

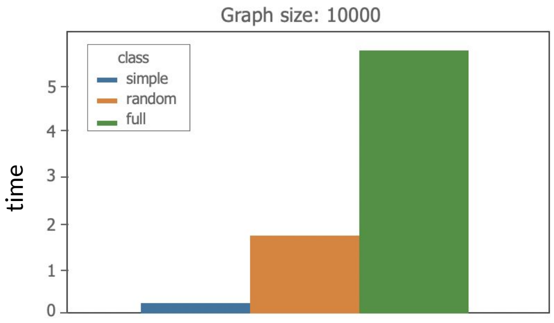
V = 100



V = 1000



V = 10 000



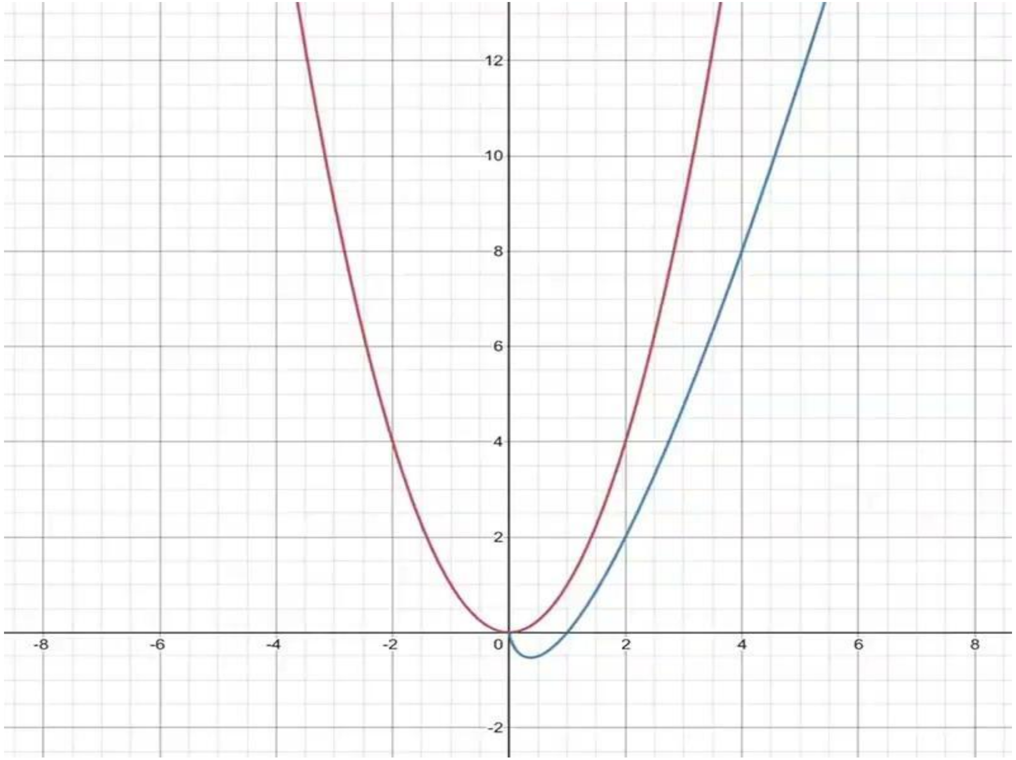
Comparison between Graph Implementations

List VS Matrix

Theoretical Comparison

$$O(E\log V + V\log V) - O(V^2) < 0$$
$$E\log V + V\log v \ < V^2$$
$$E\log V < V^2 - V\log V$$
$$E < \frac{\mathrm{V}^2 - \mathrm{V}\log V}{\log V}$$
$$E < \frac{V^2}{\log V} - V$$

Finding at which range of edges a graph implementation of adjacency list will perform better than adjacency matrix



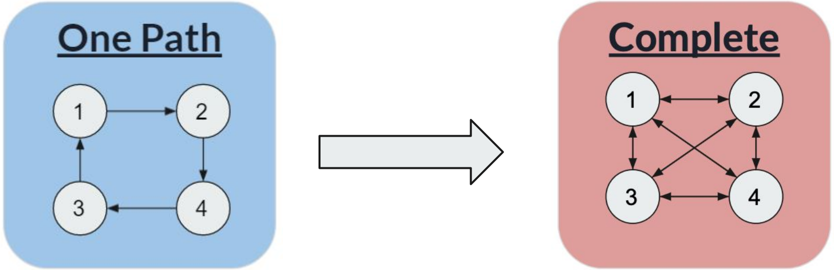
1 x^2

2 $(x+E)\log_2(x)$

3 $E = 7.58$

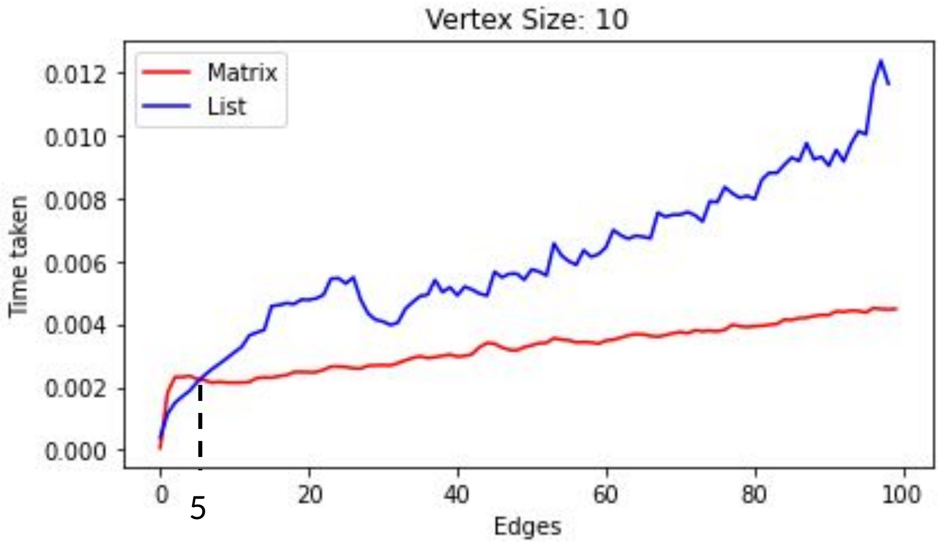
0 10

Empirical comparison

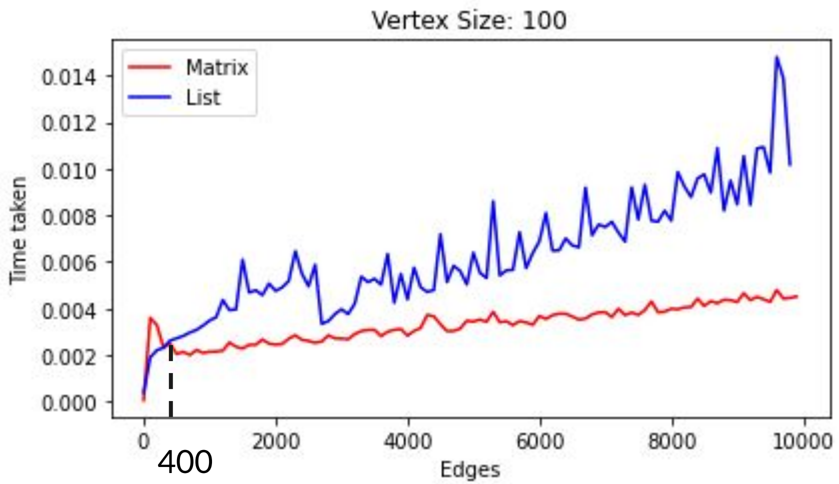


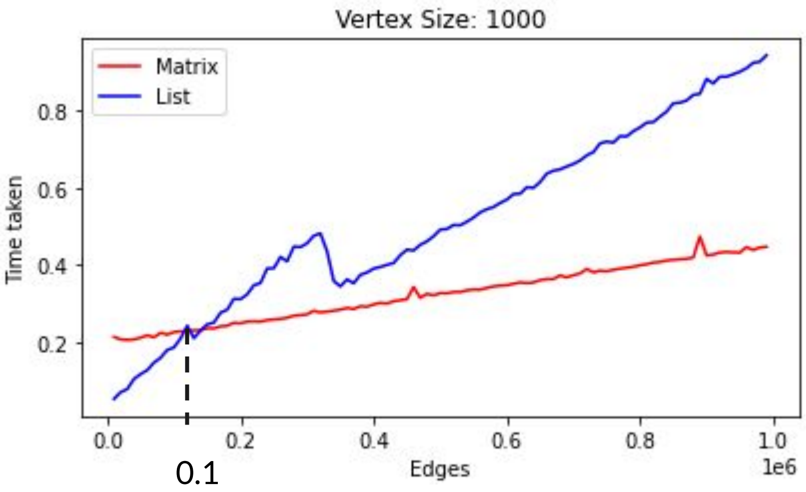
Sparse Graph:
List outperforms Matrix

Dense Graph:
Matrix outperforms List



No.of vertices





Theoretical Vs Empirical

Edge / Vertices	V = 10	V = 100	V = 1000
Theoretical Edge No.	20	1405	99,843
Empirical Edge No.	5	400	100,000

Factors for difference :
1) Constant multiplier to time complexity
2) Memory space
3) Computer system

Conclusion

Graph Type / Priority	Memory Space	Runtime
Dense	Adjacency List / Adjacency Matrix	Adjacency Matrix
Sparse	Adjacency List	Adjacency List

Comparison

Adjacency Matrix add edges easier than Adjacency List

Compare DENSE vs SPARSE graphs

Theoretical comparison

- Adjacency list will run faster than adjacency matrix up to a certain edge number
- Better to represent denser graph as adjacency matrix as it has larger number of edges