Algorithms and Data Structures

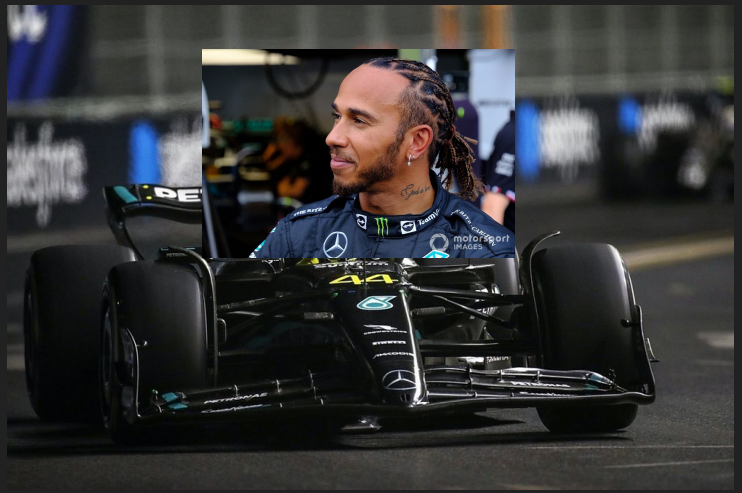
Exercise Session 10

https://n.ethz.ch/~ahmala/and

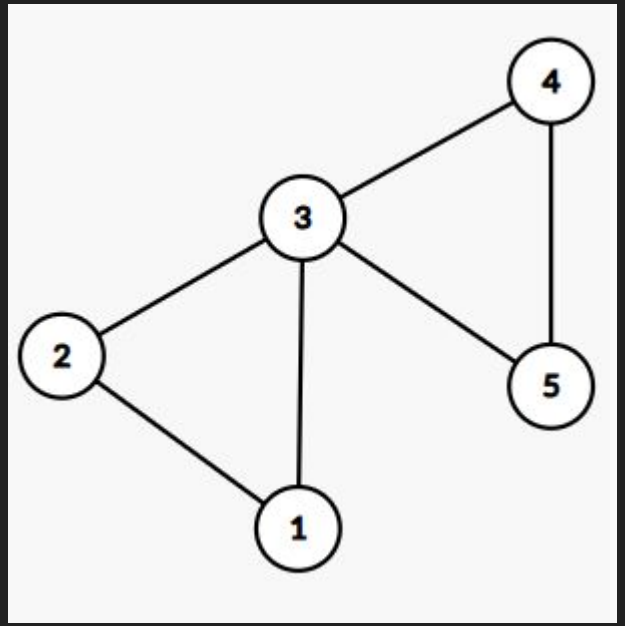
Quiz

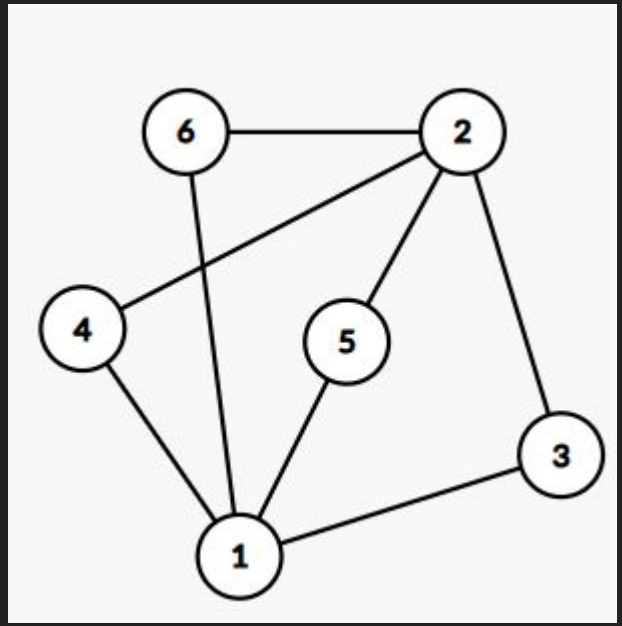
Is it Correct?
Eulerian ----> Hamiltonian

---->



Eulerian ----> Hamiltonian





Bipartite GraphK_{2,4}

BFS

- Start at the source vertex.
- Mark it as visited and add it to the queue.
- While the queue is not empty:
  - Remove a vertex from the front of the queue.
  - For each unvisited neighbor of this vertex:
    - Mark it as visited.
    - Add it to the queue.

Task-1

Find the length of the shortest path from vertex A to vertex B in an undirected graph

Task-2

Find the length of the shortest cycle in the undirected graph

Start a BFS from each vertex. As soon as we try to go from the current vertex back to the source vertex, we have found the shortest cycle containing the source vertex. At this point we can stop the BFS, and start a new BFS from the next vertex. From all such cycles (at most one from each BFS) choose the shortest.

Task-3

Find all the edges that lie on any shortest path between vertices A and B.

- We run two breadth-first searches: one from A and one from B.
- Let d_A[] be the array containing shortest distances obtained from the first BFS (from A) and d_B[] be the array containing shortest distances obtained from the second BFS (from B).
- Now for every edge (u,v), it is easy to check whether that edge lies on any shortest path between A and B:
    - the criterion is the condition **d_A[u] + 1 + d_B[v] = d_A[B]** .

Bellman-Ford Algorithm

- Create an array of distances d[1...n].
- Initially, d[s] = 0 (for the source vertex s) and all other distances are set to infinity ∞.
- Repeat n-1 times:
  - Iterate through all edges of the graph:
    - For the edge (a,b) with weight c:
      - Update d[b] using the value d[a]+c if this results in a smaller value for d[b] -- Essentially, we improve the shortest distance to vertex b using the edge (a,b) and the current shortest distance to vertex a.

Bellman-Ford Algorithm | Core Idea

After the execution of i-th phase, the Bellman-Ford algorithm correctly finds all shortest paths whose number of edges does not exceed i.

Presence of Negative Cycles for Bellman Ford

Dijkstra in Dense Graphs

Dijkstra in Dense Graphs (m = n*n)

Dijkstra's algorithm performs n iterations(visits each vertex once).

On each iteration it selects an unmarked vertex v with the lowest value $d[v]$, marks it and checks all the edges (v, to) attempting to improve the value $d[to]$.

Dijkstra in Dense Graphs (m = n*n)

Vertex search requires O(n)

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

Dijkstra in Dense Graphs (m = n*n)

Vertex search requires O(n)

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

each improvement can be performed in O(1)

Dijkstra in Dense Graphs (m = n*n)

Vertex search requires O(n)

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

$2*|E| = \Sigma \deg(v)$

each improvement can be performed in O(1)

Total Runtime = O(n*n + 2*m*1) = O(n^2 + m) = O(n^2)

Dijkstra in Sparse Graphs (m << n*n) + Min Heap

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

Dijkstra in Sparse Graphs (m << n*n) + Min Heap

Vertex search requires O(log(n)) -- Min Heap

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

Dijkstra in Sparse Graphs (m << n*n) + Min Heap

Vertex search requires O(log(n))

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

each improvement can be performed in O(log(n)) -- Min Heap

Dijkstra in Sparse Graphs (m << n*n) + Min Heap

Vertex search requires O(log(n))

Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

each improvement can be performed in O(log(n))

Total Runtime = O(n*log(n) + m*log(n)) = O((n+m) * log(n))

Dijkstra in Sparse Graphs (m << n*n) + Fibonacci Heap

Vertex search requires O(log(n))

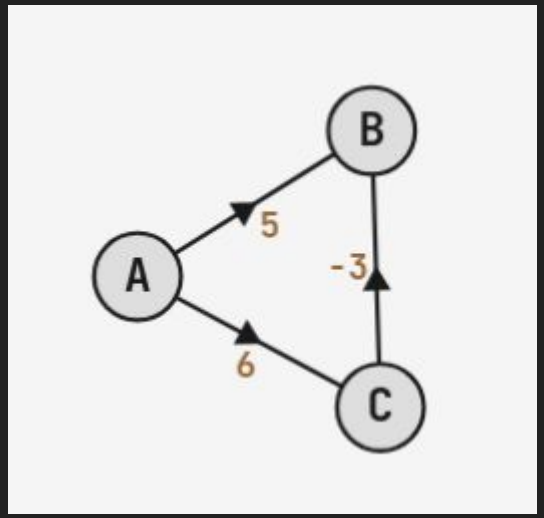
Dijkstra's algorithm performs n iterations.

On each iteration it selects an unmarked vertex v with the lowest value d[v], marks it and checks all the edges (v, to) attempting to improve the value d[to].

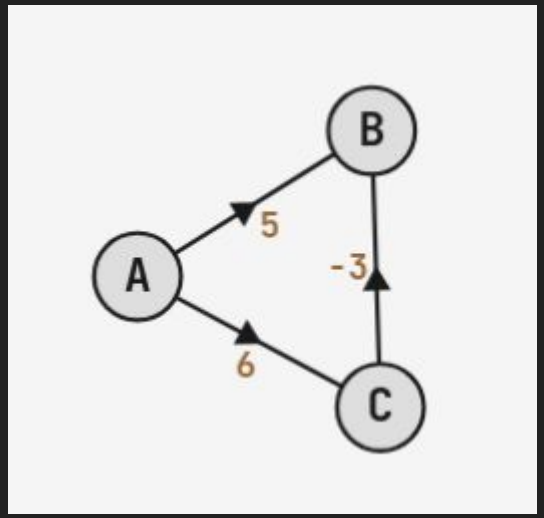
each improvement can be performed in O(1)

Total Runtime = O(n*log(n) + m*1) = O(n* log(n) + m)

Presence of Negative Weights for Dijkstra



Presence of Negative Weights for Dijkstra



Ex-Sheet 9

Peer Grading

Exercise 9.4

Peer Grading is Done With Former Groups!

New Groups for Sheets 10-12:

https://docs.google.com/spreadsheets/d/1wT5EDLl264fv-LHMnnFxhYlZIin3abMi4_XOmvPo2DA/edit?gid=1698819273#gid=1698819273