COMP9312 Shortest Path and Subgraph Matching

Outline

- Shortest Path
- Subgraph Matching

Shortest Path

Main idea

- Find a path between two vertices u and v where the sum of the weights of edges is minimized.

Shortest Path

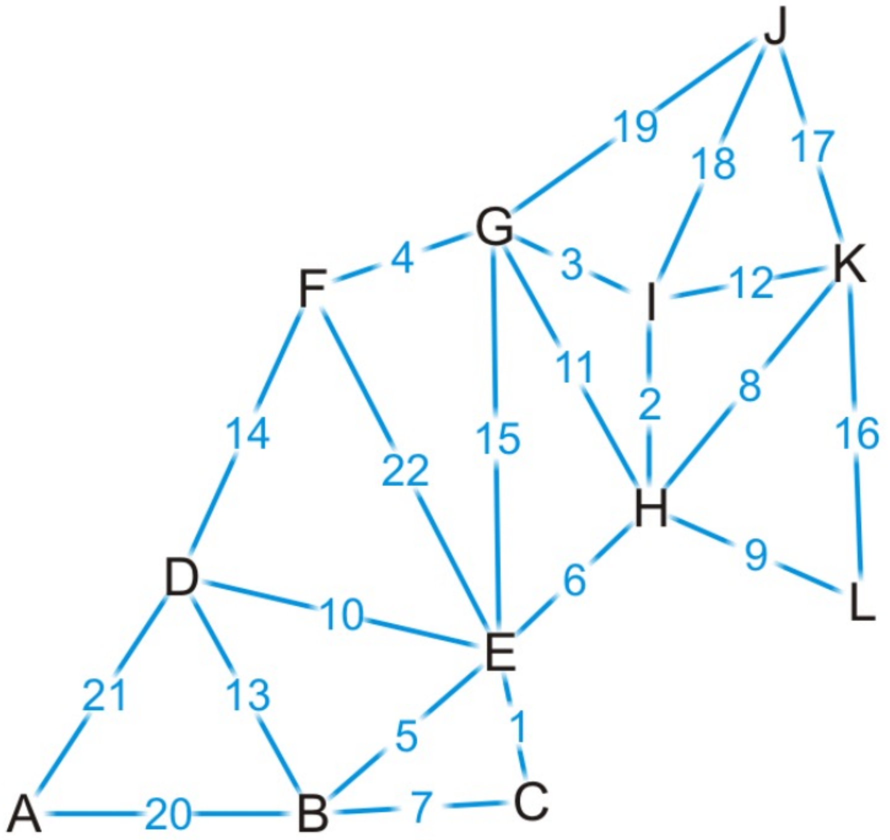
- **Single source shortest path:**
  - Dijkstra’s algorithm
  - A* algorithm

- **All pair shortest path:**
  - Floyd-Warshall algorithm

Dijkstra's algorithm

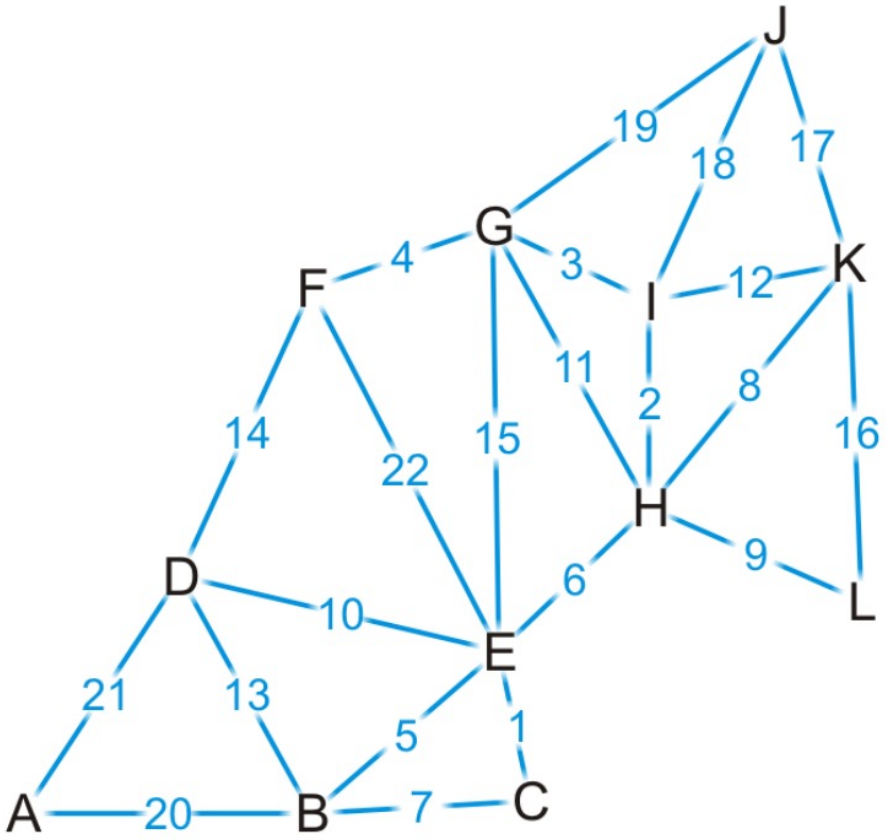
- **Implementation**
  Initialize an array of distances to infinity
  Initialize an array of previous vertices
  While still have unvisited vertex:
      Find unvisited vertex v that has a minimum distance
      mark vertex v as having been visited
      For every unvisited adjacent vertex w
          if the distance[v]+ weight of (v, w) < distance[w]:
              update the shortest distance of w
              record v as the previous pointer

Dijkstra’s algorithm



Vertex	Visited	Distance	Previous
A	F	∞	Ø
B	F	∞	Ø
C	F	∞	Ø
D	F	∞	Ø
E	F	∞	Ø
F	F	∞	Ø
G	F	∞	Ø
H	F	∞	Ø
I	F	∞	Ø
J	F	∞	Ø
K	F	0	Ø
L	F	∞	Ø

Dijkstra’s algorithm



Vertex	Visited	Distance	Previous
A	T	39	B
B	T	19	E
C	T	15	E
D	T	24	E
E	T	14	H
F	T	17	G
G	T	13	I
H	T	8	K
I	T	10	H
J	T	17	K
K	T	0	Ø
L	T	16	K

Dijkstra’s algorithm

- **Complexity Analysis**
  - Initialization: $O(|V|)$
  - Iteration through the table: $O(|V|)$
  - For each iteration, we must check all the neighbors of vertex v
  - With an adj matrix, the runtime is $O(|V| + |V|)) = O(|V|^2)$
  - With an adj list, the runtime is $O(|V|^2 + |E|) = O(|V|^2)$ as $|E| = O(|V|^2)$

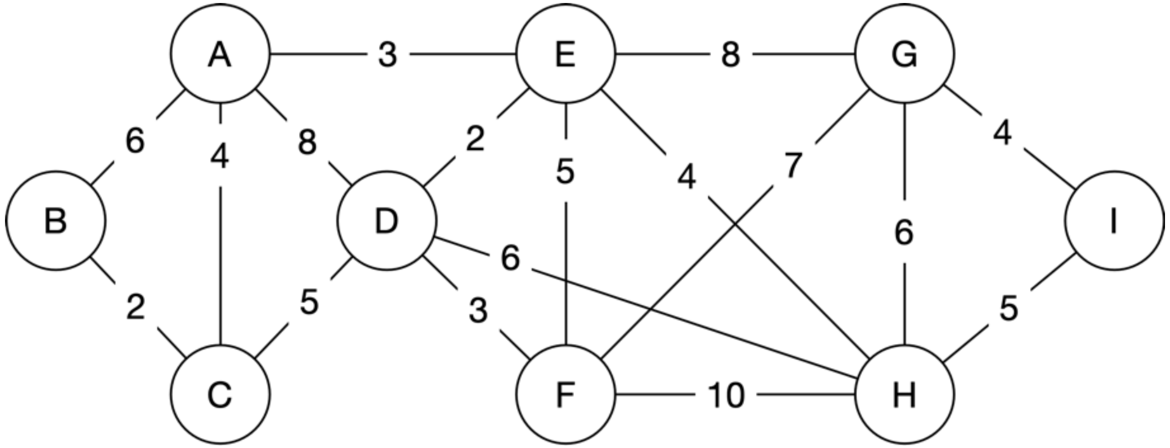
Dijkstra’s algorithm

- **Min-heap-based optimization**
  - Initialization: $O(|V|)$
  - For each iteration, find the unvisited closest vertex v takes $O(1)$, maintain the heap $O(log(|V|))$, so in total: $O(|V|log(|V|))$
  - $O(E)$ updates on the shortest distance of all the neighbors, and each update in heap takes $O(log(|V|))$, so in total: $O(|E|log(|V|))$
  - Thus, the total runtime is $O(|E|log(|V|))$

Dijkstra’s algorithm

- **Exercise:**
  - In the example graph G, find the shortest path of
    - <A,H>
    - <C, I>

Now, you have 3 minutes to do Ex1.2.

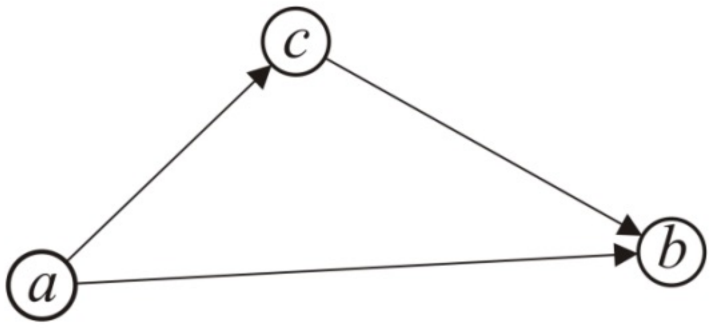


A* algorithm

- **Requirement:**
  The distance satisfy the triangle inequality, that is, the distance between a and b is less than the distance from a to c plus the distance from c to b.

  - Admissible Heuristics h
    - $h(u,v) \leq d(u,v)$
    - The heuristic is optimistic or lower bound on the distance

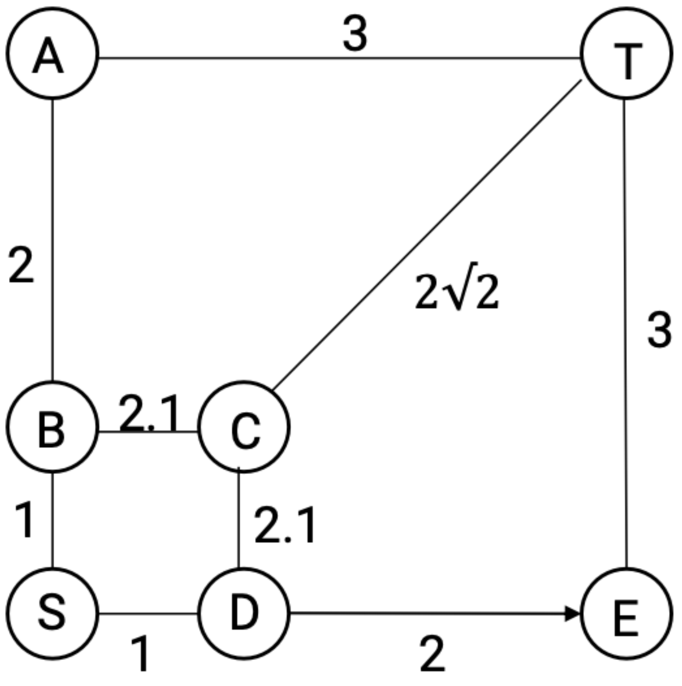
    When the heuristic is admissible, then it is guaranteed to return the shortest path.



A* algorithm

- **Implementation:**
  Mark each vertex as unvisited
  Starts with an array containing only the initial vertex
  The value of any vertex v in the array is the weight $w(v)$
  While not reach destination vertex z
  get the vertex u with the least weight
  mark the vertex u as visited
  For each unvisited adjacent vertex v
  If $w(v) = d(a, u) + d(u, v) + h(v, z)$ is less than the current $w(v)$
  update the path and weight of v

A* algorithm



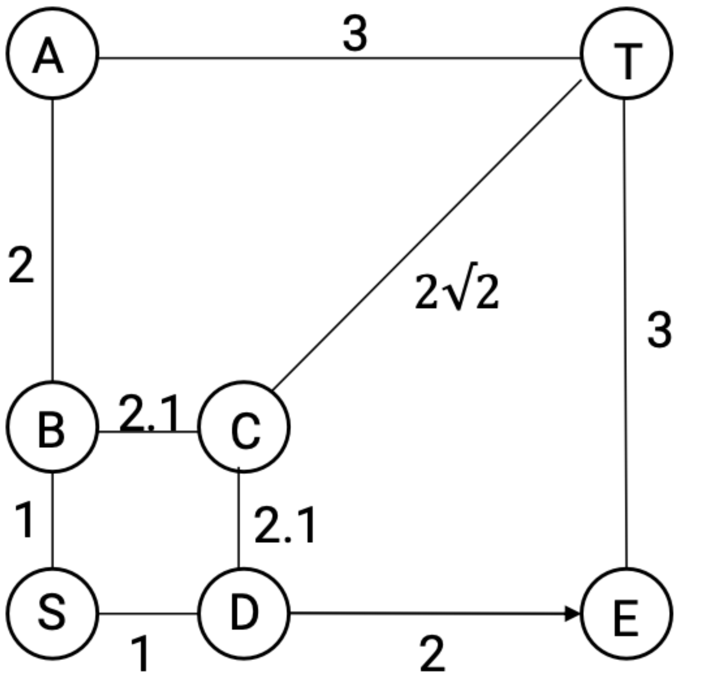
A = (0, 3)
B = (0, 1)
C = (1, 1)
D = (1, 0)
E = (3, 0)
S = (0, 0)
T = (3, 3)

Heuristic: $d(u,v) = \sqrt{(u_x - v_x)^2 + (u_y - u_y)^2}$

- Find the shortest path from S to T

Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	∞	3	∞	Ø
B	F	∞	$\sqrt{13}$	∞	Ø
C	F	∞	$2\sqrt{2}$	∞	Ø
D	F	∞	$\sqrt{13}$	∞	Ø
E	F	∞	3	∞	Ø
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	F	∞	0	∞	Ø

A* algorithm



A = (0, 3)
B = (0, 1)
C = (1, 1)
D = (1, 0)
E = (3, 0)
S = (0, 0)
T = (3, 3)

Heuristic: $d(u,v) = \sqrt{(u_x - v_x)^2 + (u_y - u_y)^2}$

- Find the shortest path from S to T

Vertex	Visited	Distance	Heuristic	Total	Previous
A	F	3	3	6	B
B	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
C	T	3.1	$2\sqrt{2}$	$3.1+2\sqrt{2}$	B
D	T	1	$\sqrt{13}$	$1+\sqrt{13}$	S
E	F	3	3	6	D
S	T	0	$3\sqrt{2}$	$3\sqrt{2}$	Ø
T	T	$3.1+2\sqrt{2}$	0	$3.1+2\sqrt{2}$	C

Floyd Warshall algorithm

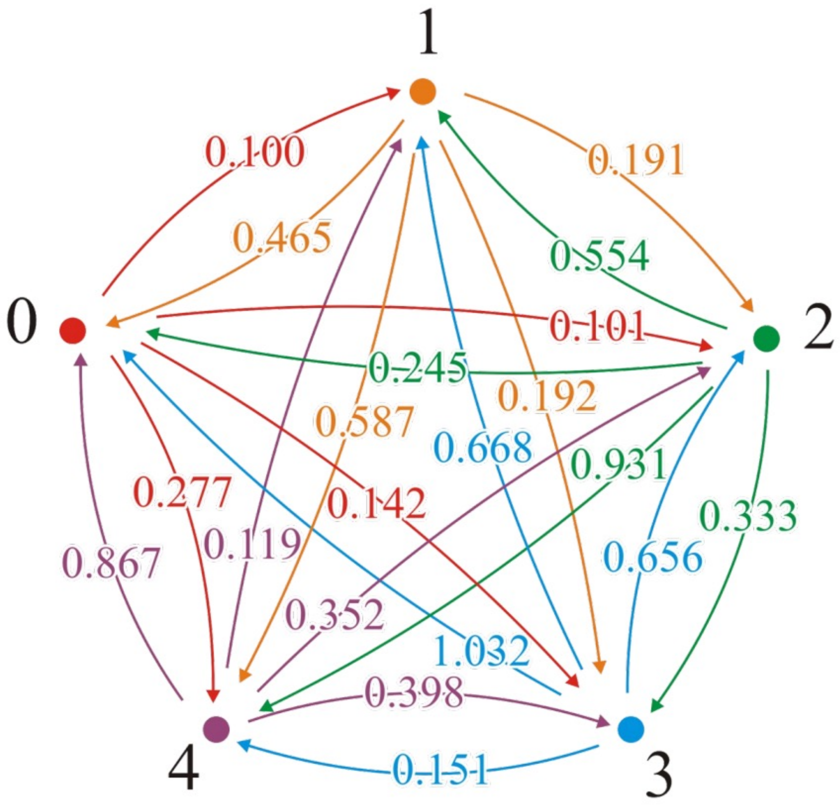
- **Implementation**

```
double d[num_vertices][num_vertices];

// Initialize the matrix d:  Theta(|V|^2)
//   ...

// Run Floyd-Warshall
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            d[i][j] = min( d[i][j], d[i][k] + d[k][j] );
        }
    }
}
```

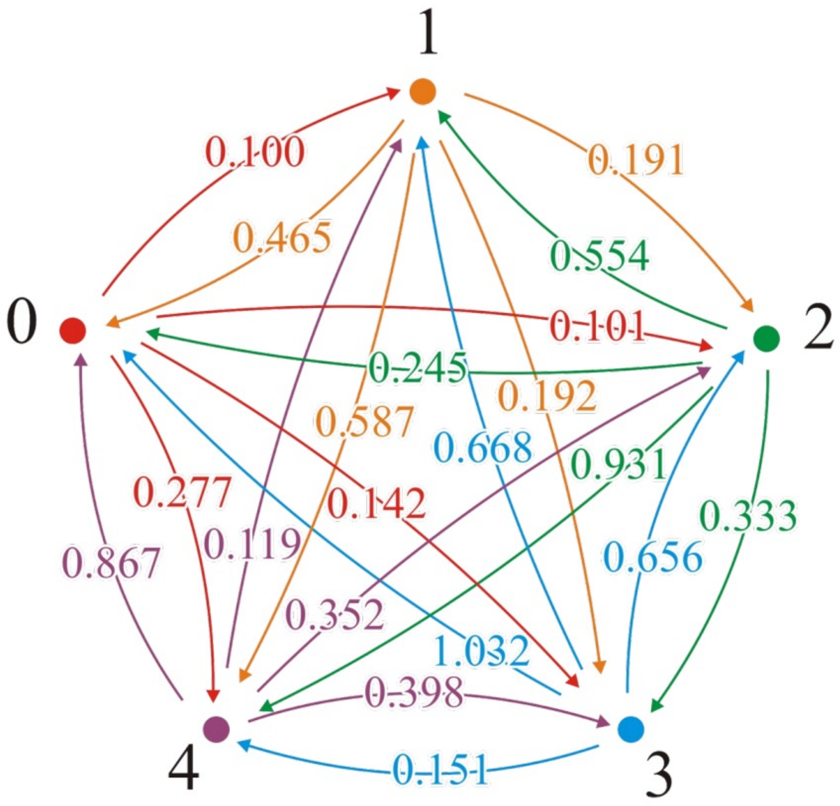
Floyd Warshall algorithm



- **Initialization**

$$\begin{pmatrix} 0 & 0.100 & 0.101 & 0.142 & 0.277 \\ 0.465 & 0 & 0.191 & 0.192 & 0.587 \\ 0.245 & 0.554 & 0 & 0.333 & 0.931 \\ 1.032 & 0.668 & 0.656 & 0 & 0.151 \\ 0.867 & 0.119 & 0.352 & 0.398 & 0 \end{pmatrix}$$

Floyd Warshall algorithm



- **Shortest distance**

0	0.100	0.101	0.142	0.277
0.436	0	0.191	0.192	0.343
0.245	0.345	0	0.333	0.484
0.706	0.270	0.461	0	0.151
0.555	0.119	0.310	0.311	0

Floyd Warshall algorithm

- **Shortest path**

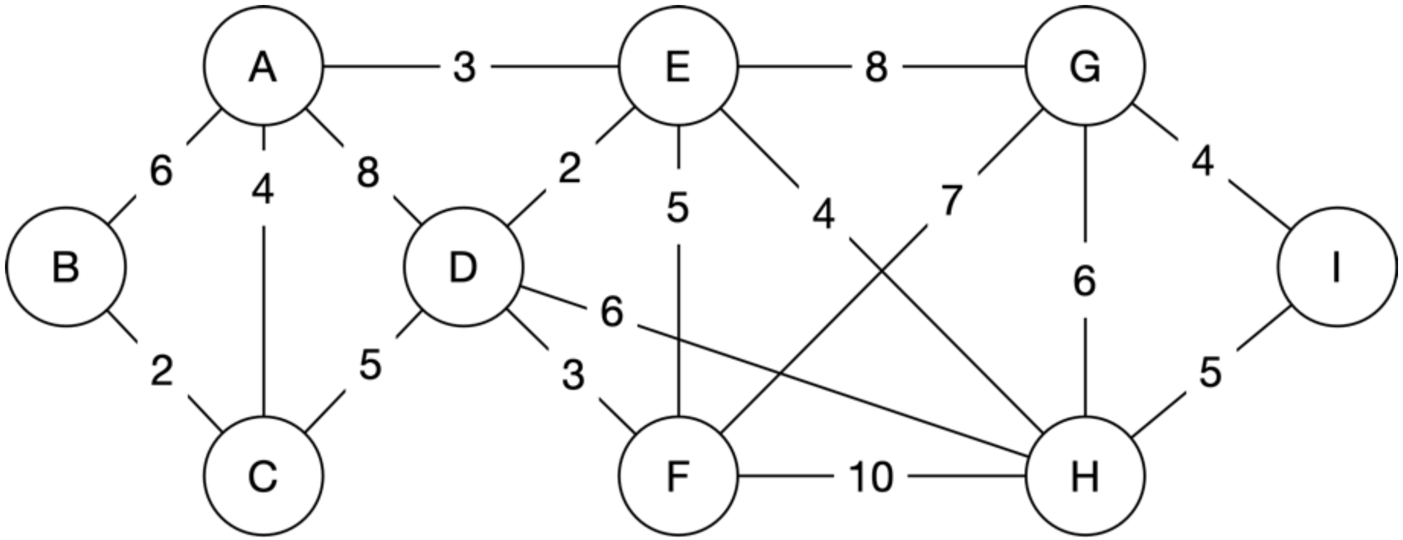
```
unsigned int p[num_vertices][num_vertices];

// Initialize the matrix p:  O(|V|^2)
//   ...

// Run Floyd-Warshall
for ( int k = 0; k < num_vertices; ++k ) {
    for ( int i = 0; i < num_vertices; ++i ) {
        for ( int j = 0; j < num_vertices; ++j ) {
            if ( d[i][j] > d[i][k] + d[k][j] ) {
                p[i][j] = p[i][k];
                d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

Floyd Warshall algorithm

- **Exercise**
  Find the shortest distance of all pair of vertices in the example graph G



Now, you have 5 minutes to do Ex1.4.

Graph Homomorphism

- **Definition**
  Two graphs G and H are homomorphic if there exists a function function $f: V_G \to V_H$ between vertices of the graph such that if $\{a, b\}$ is an edge in G then $\{f(a), f(b)\}$ is an edge in H.

  Two graphs G and H are isomorphic if there exists a **bijective** function $f: V_G \to V_H$ between vertices of the graph such that if $\{a, b\}$ is an edge in G then $\{f(a), f(b)\}$ is an edge in H.

Graph Homomorphism

- **Exercise**
  Are the graphs in Figure 1 and Figure 2 isomorphic? If so, demonstrate an isomorphism between the set of vertices.

Now, you have 5 minutes to do Ex2.2.

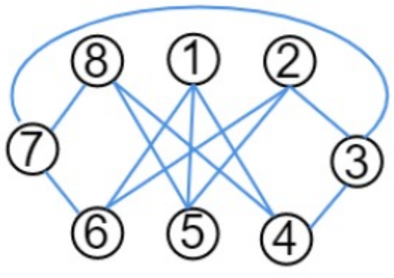


Figure 1

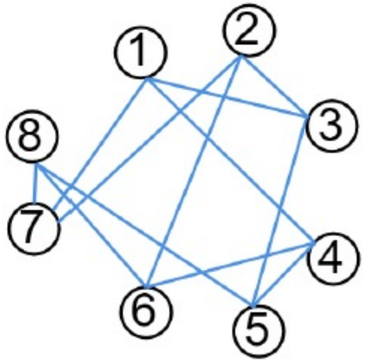
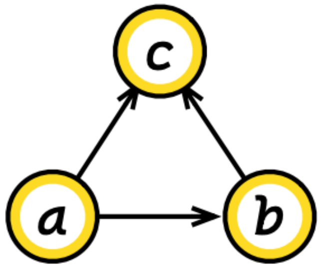


Figure 2

Triangle Counting

- **Implementation**
  - Priority: determined by degree
  - Orientation technique: Map undirected edge into directed edge. The direction is decided by the priority of the endpoint in the vertex-ordering, i.e., u->v if u has a higher priority than v.



Triangle Counting

- **Implementation**
  - **Compact Forward (CF) algorithm**

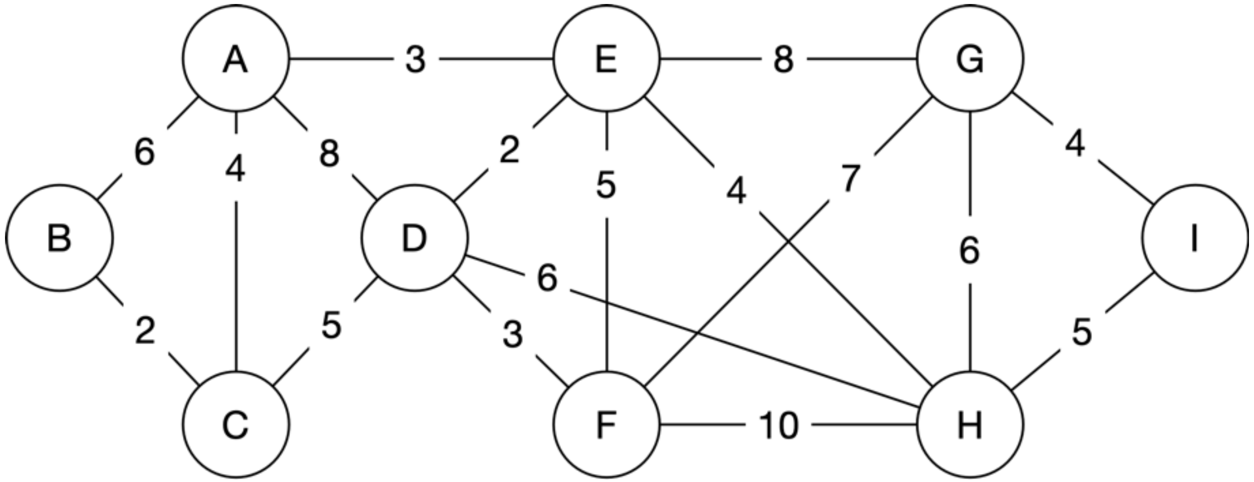
Algorithm 1: CF(G)

Input : G : an undirected graph
Output : All triangles in G

$\boldsymbol{G} \leftarrow$ Orientation graph of G based on degree-order;
for each vertex $u \in \boldsymbol{G}$ **do**
  for each out-going neighbor v **do**
    $T \leftarrow N^+(u) \cap N^+(v)$;
    for each vertex $w \in T$ **do**
      Output the triangle (u, v, w);

Triangle Counting

- **Exercise**
  - List all the triangles in the example graph G



Triangle Counting

- **Complexity analysis**

  We need to check common neighbors in the adj list.
  - If the adj list is sorted, the time complexity of CF algorithm is: $O\left(\sum_{(u,v)\in E} deg^{+}(u) * deg^{+}(v)\right)$
  - If the adj list is sorted, the time complexity of CF algorithm is: $O\left(\sum_{(u,v)\in E} deg^{+}(u) + deg^{+}(v)\right)$

Triangle Counting

- **Complexity analysis**
  - Suppose a hash table has been built for each vertex based on the out-going neighbors in the oriented graph. We can choose the vertex with larger number of neighbors as the hash table for intersection with $O(\min(deg^+(u), deg^+(v)))$ cost.

    The complexity of CF algorithm: $O\left(\sum_{(u,v)\in E} \min(deg^+(u), deg^+(v))\right) = O\left(\sum_{(u,v)\in E} \min(deg(u), deg(v))\right) = O(\alpha \cdot m) = O(m^3)$

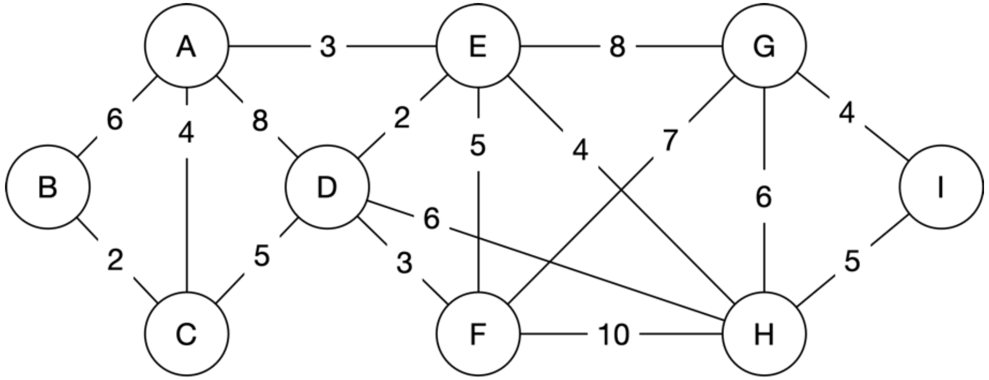
Triangle Counting

- **Complexity analysis**
  - Build a hash table for each vertex requires large space cost for big graphs. We can build a hash table on the fly.

    The complexity of CF algorithm: $O\left(\sum_{(u,v)\in E'} deg^+(v)\right) = O\left(\sum_{(u,v)\in E'} deg(v)\right) = O\left(\sum_{(u,v)\in E} \min(\deg(u), \deg(v))\right) = O(\alpha \cdot m) = O(m^{1.5})$

Triangle Counting

- **Exercise**
  - Load the graph via class ‘SimpleGraph’ in tutorial_6.py
  - Implement CF algorithm to list all the triangles in the graph G



Now, you have 15 minutes to do Ex2.5.

Q & A