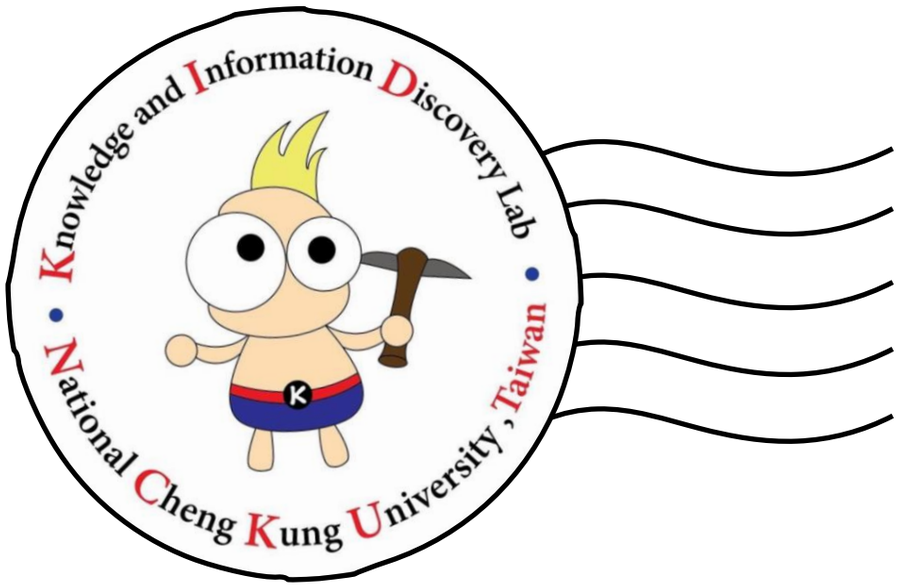


Influence Maximization

Midterm Document

Knowledge and Information Discovery Lab
National Cheng Kung University, Taiwan

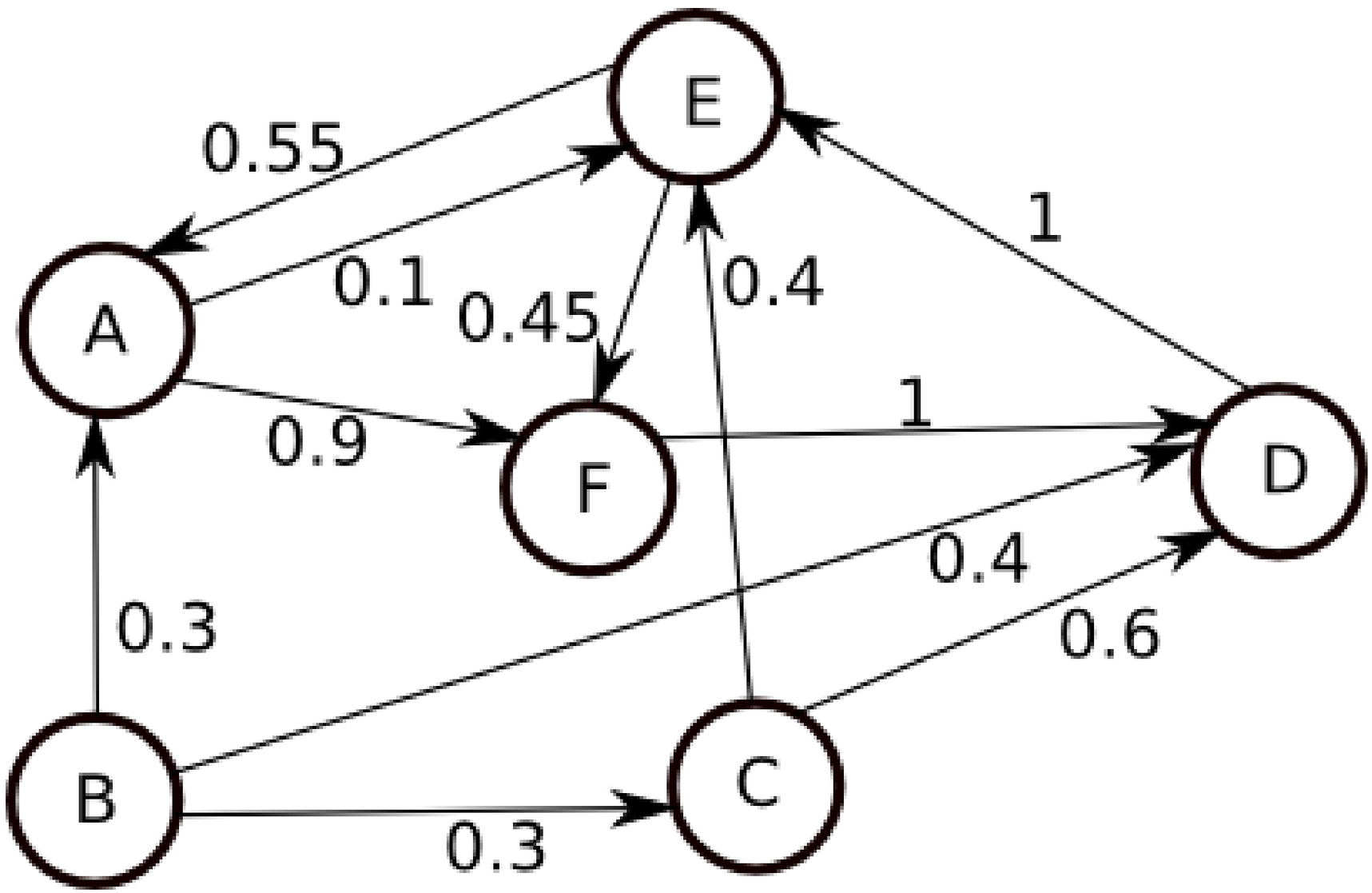
Outline

- Social Network
- Diffusion Model
- Task : Influence Maximization
- DirectedGraph Class
- Linear Threshold Model
- seedSelection Algorithm
- Midterm Notice

Social Media

Social Network

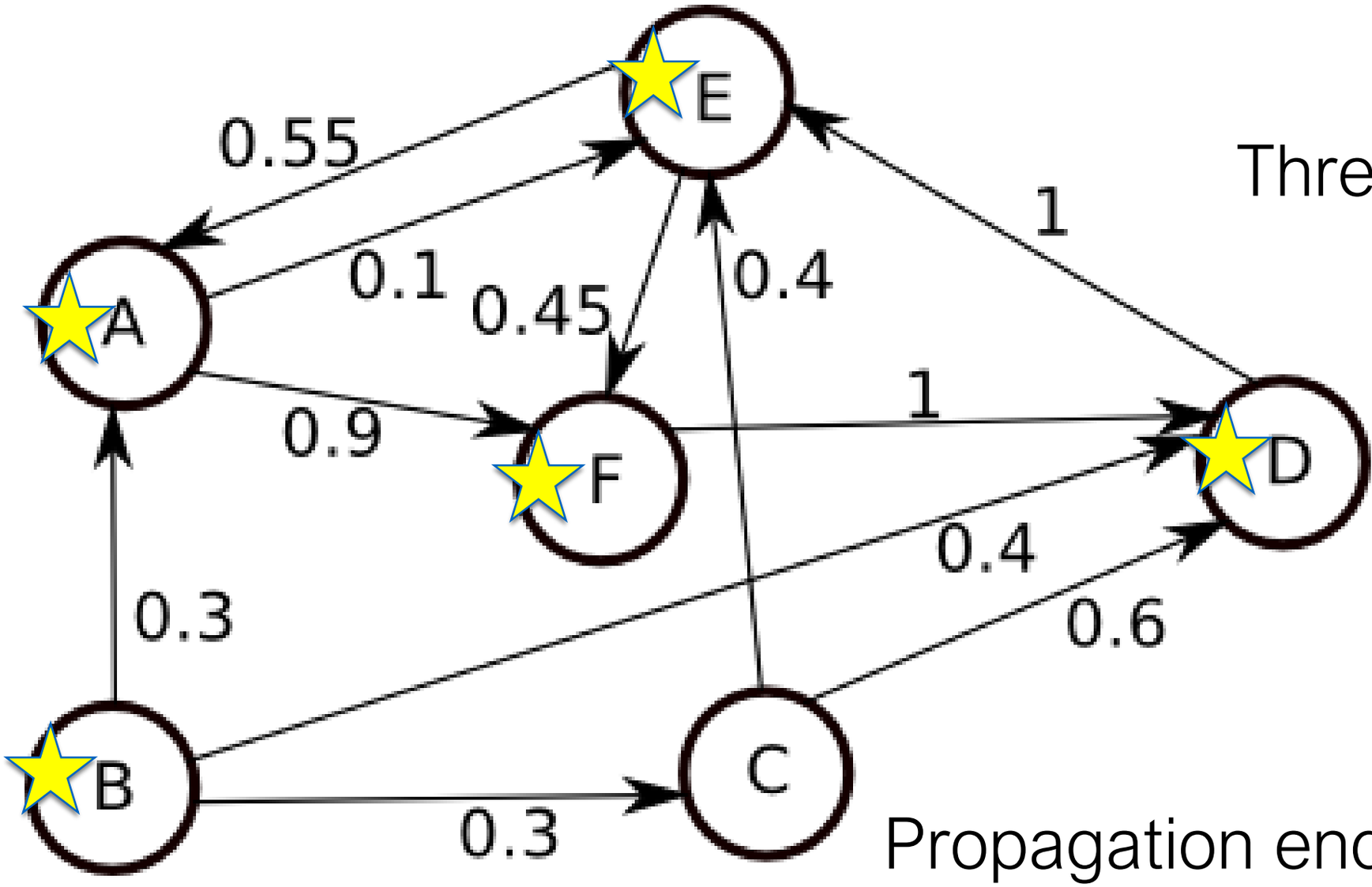
Graph



Diffusion Model : LT

- **Active Node** : Propagate influence through its out edges.
- **Activated Threshold** : For a non-active node in the network, if its total received influence is larger than its own activated threshold, then it becomes an active node in the next round.
- **End of Propagation** : When there is no non-active node turn into active node, the propagation ends.

Diffusion Process



Threshold : 0.6

Propagation ends in round **3**

Influence Maximization

- **Given** : A social network and number of initial active users.
- **Task** : Choose a group of influential people as initial active users.
- **Target** : Maximize the number of active users in the social network when propagation ends.

DirectedGraph Class

- #include “graph.h”
- All information about the network is stored in a DirectedGraph class.
- Retrieve the information you want via accessing the member functions belong to the class.
- Just use “.” operator, or “->” operator if it is a pointer.

Access Class Members

```
DirectedGraph G = DirectedGraph();
G.addEdge(1, 2, 0.7);
DirectedGraph *G_pointer = &G;
G_pointer->getEdgeThreshold(1, 2);
```

Edge struct

- ***int*** from_node

  Describe which user this Edge directs from

- ***int*** to_node

  Describe which user this Edge directs to

- ***double*** influence

  Describe the influence weight from user from_node to user to_node

Node struct

- ***int*** node

  Describe which user this Node belongs to

- ***double*** threshold

  Describe the threshold influence that will activate this user

- ***vector<Edge*>*** in_edges

  Contain Edge pointers which direct to this user

- ***vector<Edge*>*** out_edges

  Contain Edge pointers which this user directs to

Access Struct members

```
Edge edge = { 1, 2, 0.7 };

edge.threshold; // 0.7
```

```
Edge *edge_pointer = &edge;

edge_pointer->threshold; // 0.7
```

getNodeNumber

- *int* getNodeNumber(void)

Return number of total users in this network

getEdgeNumber

- *int* getEdgeNumber (void)

Return number of total connections between users in this network

addNode

- ***void*** addNode (int node, double threshold)

Insert a new Node into this network with given user and its threshold value

setNode

- ***void*** setNode (int node, double threshold)

Update a user's threshold with given value

addEdge

- ***void*** addEdge (int from_node, int to_node, double influence)

Insert a new Edge into this network with two given users and influence weight

setEdge

- ***void*** setEdge (int from_node, int to_node, double influence)

  Update an edge with two given users and influence weight

deleteNode

- ***void*** deleteNode (int node)

Delete a user from the network

deleteEdge

- ***void*** deleteEdge (int from_node, int to_node)

Delete a connection between two given users in this network

isNodeExist

- ***bool*** isNodeExist(int node)

Check if a given user exists in this network

isEdgeExist

- ***bool*** isEdgeExist(int from_node, int to_node)

Check if a given directed connection exists between two users in this network

getNodeThreshold

- ***double*** getNodeThreshold(int node)

Return a given user's threshold weight

getEdgeInfluence

- ***int*** getEdgeInfluence(int from_node, int to_node)

Return the influence weight from user from_node to user to_node

getNodeInNeighbors

- *vector<int>* getNodeInNeighbors(int node)

Return the users which have a connection directs to this given user

getNodeOutNeighbors

- ***vector<int>*** getNodeOutNeighbors(int node)

Return the users which this given user has a connection directs to

getAllNodes

- ***vector<int>*** getAllNodes(void)

Return all users in this network

getAllEdges

- ***vector<pair<int, int>>*** getAllEdges(void)

Return all connections between users in this network

Linear Threshold Model

- #include “LT.h”
- Diffusion model we use to run the propagation
- Includes three diffusion function with different purpose

diffuse_one_round

- ***vector<int>*** diffuse_one_round (DirectedGraph* G, unordered_set<int>& seeds)

  Return all active users in the network after one round of propagation with given initial active users

diffuse_all

- ***vector<int>*** diffuse_all (DirectedGraph* G, unordered_set<int> seeds)

Return all active users in the network when propagation ends, with given initial active users

diffuse_k_rounds

- ***vector<int>*** diffuse_k_rounds (DirectedGraph* G, unordered_set<int> seeds , int rounds)

  Return all active users in the network after k rounds of propagation with given initial active users

seedSelection Algorithm

- Your seedSelection function prototype must looks **exactly the same** as below !!!

- ***unordered_set<int>*** seedSelection (DirectedGraph G, unsigned int numberOfSeeds)

  Return users you choose as initial active users in the network to run the diffusion model

Ranking Criteria

- Active Rate
  - (Number of active users when propagation ends) / (Number of total users in the network)
- E.g.
  - Number of active users when propagation ends = 8,000
  - Number of total users in the network = 10,000
  - Active Rate = 80.0 %

Notice

- **Do not** cout anything in your seedSelection function
- **Do not** use system(“pause”) or cin
- Size of your return unordered_set **must be the same** as the given **numberOfSeeds**

Restriction

- **Time limitation** : For each dataset, your seedSelection function must return the result in **less than 12 minutes**
- **Memory limitation** : For each dataset, the **maximum memory** can be used is **2GB**
- **Submission** : At most **5** times a day
- **Blocking** : Can not submit a new file till the old one finishes judging

Q & A

If you find any bug in online judge system, please notify TA via email ASAP.