EE599 Discussion – Lab10

Xuan Zuo

04/01/2019

Lab 10

- In this lab, you will practice graph related algorithms.

- Part I - Graph valid tree

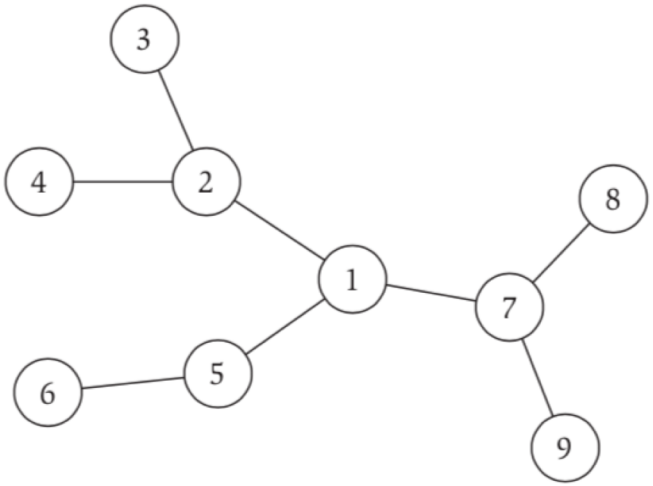
  This part is mandatory.

- Part II – Find shortest path using Dynamic Programming
- Part III – Find the maximum clique of a graph

- Parts II and III are extra credit, i.e., skipping them would not negatively impact your grade.

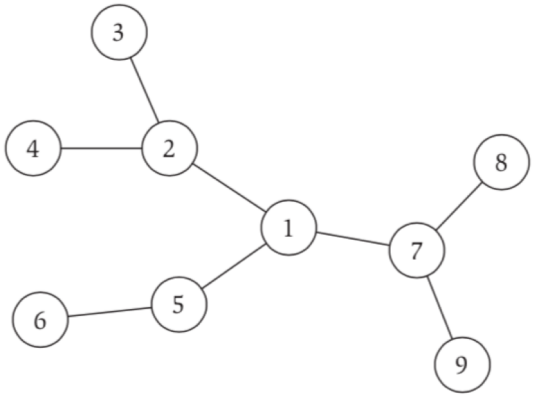
Background

- A set of nodes and edges
  • Each edge shows a relation between the corresponding nodes
- G = (V, E) comprising
  A set V of vertices, nodes or points together
  A set E of edges, arcs or lines,

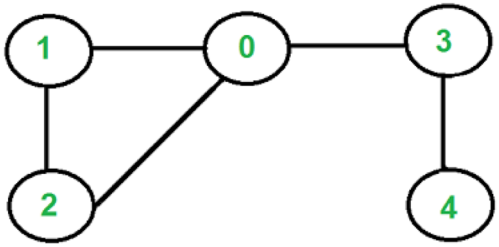


Part I

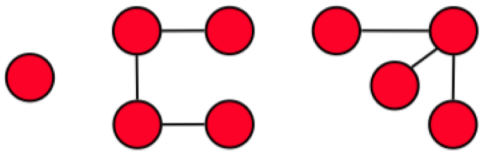
- An undirected graph is tree if it has following properties.
  1) There is no cycle.
  2) The graph is connected.



Tree



Not tree, there is a cycle



Not tree, Forest

Part 1

- **How to detect cycle in an undirected graph?**

- We can either use BFS or DFS.
- E.g. we do a DFS traversal of the given graph. For every visited vertex ‘v’, if there is an adjacent ‘u’ such that u is already visited and u is not parent of v, then there is a cycle in graph. If we don’t find such an adjacent for any vertex, we say that there is no cycle.

Part 1

- **How to check for connectivity?**

- Since the graph is undirected, we can start BFS or DFS from any vertex and check if all vertices are reachable or not. If all vertices are reachable, then graph is connected, otherwise not.

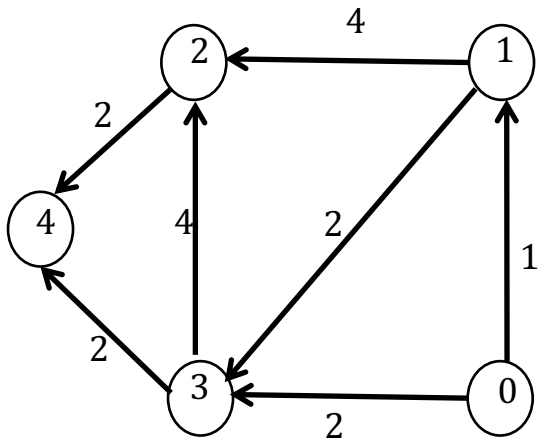
Part 1

- **Task:**
- Given n nodes labeled from 0 to n-1 and a list of undirected edges (each edge is a pair of nodes), check whether these edges make up a valid tree.
- **“./a.out input1.txt”**

  5

  0,1, 0,2, 0,3, 1,4
- Print “Graph a valid tree” if the given graph is a valid tree. Otherwise, print “Graph is not a valid tree”.
- Write a Python code that automatically generates 10 randomly generated undirected graph samples. Run c++ code in python.

Part 2 - Bellman-ford algorithm



A graph with n node/vertex:

- **Step 1: Initialization:**

d_0**(0) = 0.**

d_0**(x) = infinite for all** $x \neq 0$

predecessor[x] = null

- **Step 2:**

For i from 1 to n-1

- $s_0(x) = \min_{v \in V:(v,x)\ is\ an\ edge} \{d_0(v) + c(v,x)\}$ **for all** $x \neq 0$
- $d_0(x) = s_0(x)$ **for all** $x \neq 0$
- **Predecessor[x] = v**

/* for each edge (v, x) in Graph with cost c(v ,x) in edges:

if d_0**(v) +** $c(v,x)$ **<** d_0**(x)**

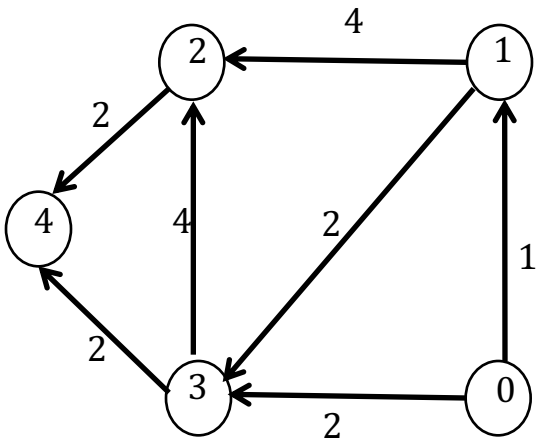
d_0**(x)** $= d_0\ (v) + c(v,x)$

predecessor[x] =v */

- **Return** $d_0(x)$ **for all x, predecessor[x] for all** $x \neq 0$

Part 2

Iterations	$d_0(0)$	$d_0(1)$	$d_0(2)$	$d_0(3)$	$d_0(4)$
0(Initialization)	0	Infinity	Infinity	Infinity	Infinity
1	0	1(pre=0)	Infinity	2(pre=0)	Infinity
2	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)
3	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)
4	0	1(pre=0)	5(pre=1)	2(pre=0)	4(pre=3)



Input:
0,1,*,2,*
,0,4,2,
,,0,*,2
,,4,0,2
,,*,*,0

Output:
0,1,5,2,4
0
0→1
0→1→2
0→3
0→3→4

Part 2

- **Implement Bellman-Ford algorithm in C++.**
  - **Read a graph described in an external input file.**
  - **Input file name should be an input parameter of your complied program**
    - ***"./a.out input2.txt"***
  - **Find the shortest paths from node 0 to all the other nodes**
  - **Output the shortest paths to a file.**

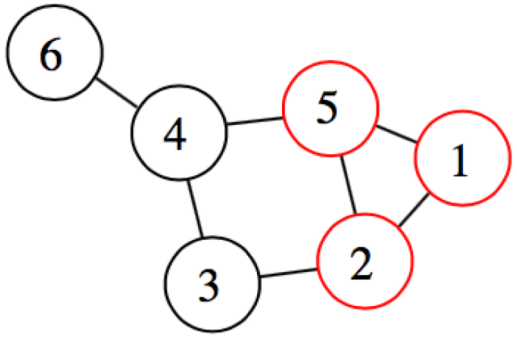
Input file:
0,1,*,2,*
,0,4,2,
,,0,*,2
,,4,0,2
,,*,*,0

Output file:
0,1,5,2,4
0
0→1
0→1→2
0→3
0→3→4

If there is a negative loop:
Output: Negative Loop Detected

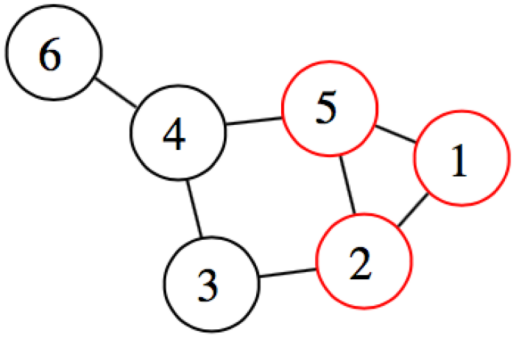
Part 3

- The clique problem has many real-world applications. E.g., consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends.
- **Given an undirected graph G = (V, E), a clique S is a subset of V such that for any two elements u, v ∈ S, (u, v) ∈ E.**



Part 3

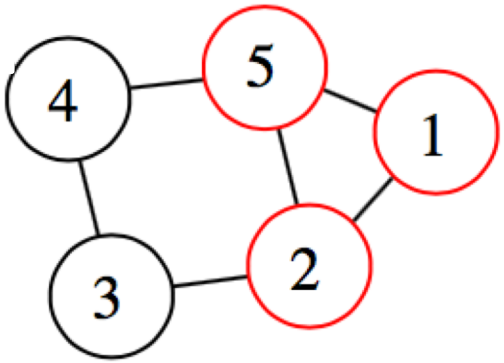
- A **maximum clique** of a graph, G, is a clique, such that there is no clique with more vertices.
- The clique number ω(G) of a graph G is the number of vertices in a maximum clique in G.
- Finding the maximum clique in a graph is an NP-hard problem, called the maximum clique problem (MCP).
- A **maximal clique** is a clique that is not included in a larger clique.
- Listing all maximal cliques in the graph (recursive) and returning the largest one. Runtime $O(3^{n/3}) = O(1.4422^n)$.



Part 3

- **Tasks:**
- **Find the maximum clique in a given graph.**
- **“./a.out input3.txt”**
- **Input file:**

  5
  0 1 0 0 1
  1 0 1 0 1
  0 1 0 1 0
  0 0 1 0 1
  1 1 0 1 0



- **Output file:**

  Clique number (3): 1 2 5

Submission

- **Due date: Monday, April 8th.**
- **Please push all files you wrote into the GitHub repo.**
- **https://classroom.github.com/a/6lklQVTz**
- **Don’t forget to submit a readme file:**
- **Your readme file should be named as readme_<STUDENT-ID>.txt, please replace <STUDENT-ID> with your own 10-digit USC-ID.**