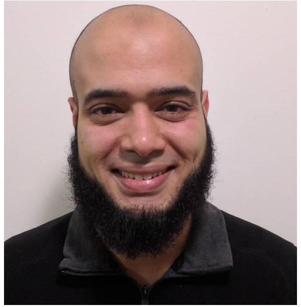
Algorithms

Graph Representation Homework 2

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

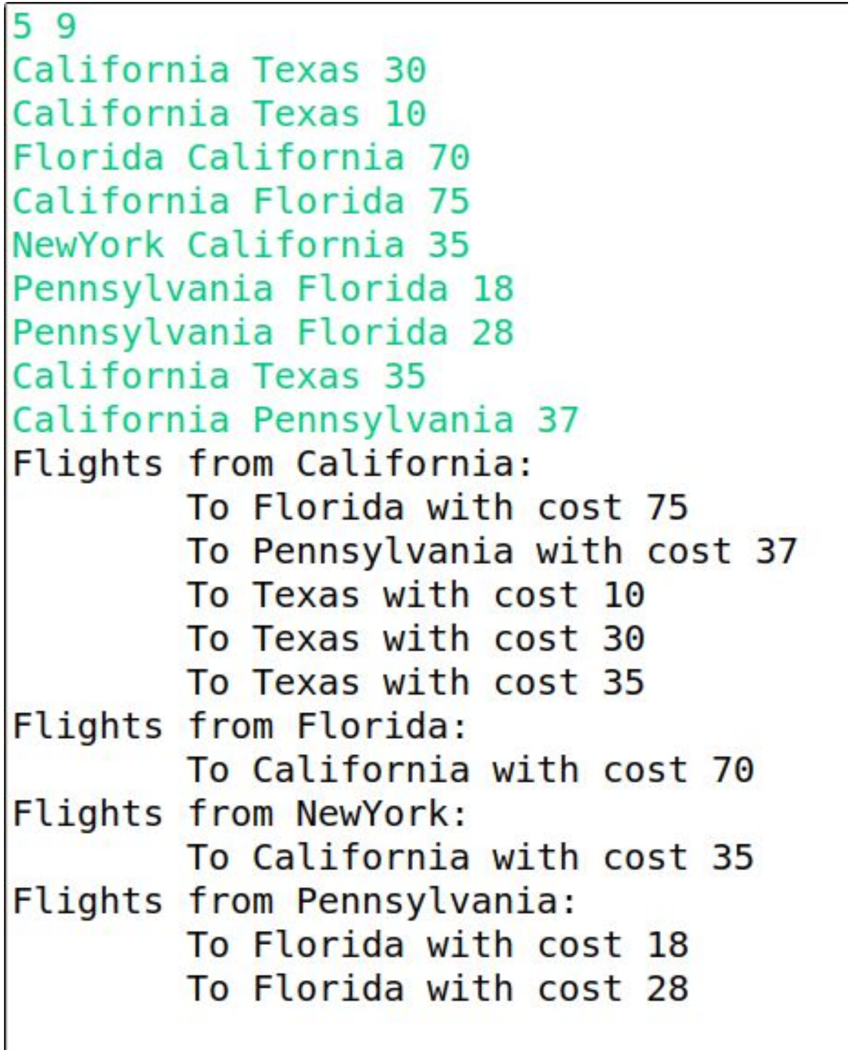


Problem #1: Adjacency-based repr for flights v1

- In the airports, there are many flights (from, to, cost)
  - Where from and to are strings (no spaces) and cost is an integer value
- Represent the **directed graph** based on **adjacency** style
  - There are multiple edges
- Your print function must do the following:
  - Flights **from** are printed sorted (alphabetical order)
  - For each **from** airport: print the **to** cities based on
    - alphabetical order
    - If there is a tie, the one with smaller cost first
  - See the sample

Problem #1: Adjacency-based repr for flights v1

- 5 airports and 9 flights
- **From Airport** is ordered
    - Florida, NewYork, Pennsylvania
- California To list is ordered by name
    - Florida, Pennsylvania, Texas
- Trips to Texas
    - Ordered by cost



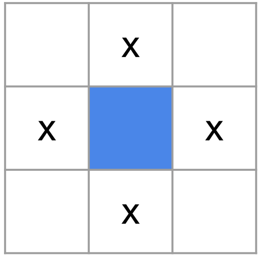
```
5 9
California Texas 30
California Texas 10
Florida California 70
California Florida 75
NewYork California 35
Pennsylvania Florida 18
Pennsylvania Florida 28
California Texas 35
California Pennsylvania 37
Flights from California:
        To Florida with cost 75
        To Pennsylvania with cost 37
        To Texas with cost 10
        To Texas with cost 30
        To Texas with cost 35
Flights from Florida:
        To California with cost 70
Flights from NewYork:
        To California with cost 35
Flights from Pennsylvania:
        To Florida with cost 18
        To Florida with cost 28
```

Problem #2: Adjacency-based repr for flights v2

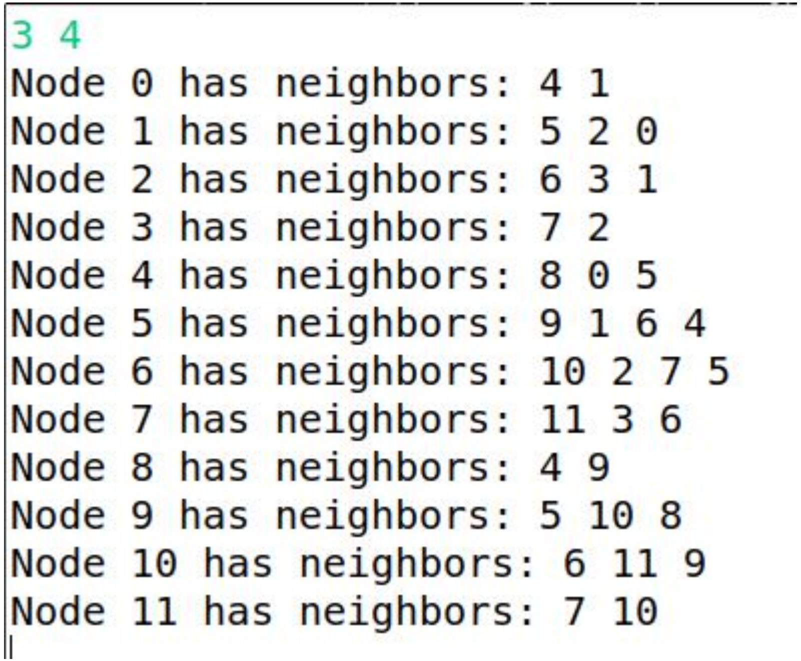
- The solution provided to the previous problem is interesting, but has a great drawback
  - If we have N standard graph algorithms, we will rewrite them all to work with the new representation
- Find a way to **reduce** this problem’s requirements (working on strings mainly) to the normal adjacency list representation. In this way, the implemented algorithms can be used as they are
  - Edge: int from, to, weight;
  - typedef vector<edge> GRAPH;
  - void add_edge(GRAPH &graph, int from, int to, int cost)
- For simplicity: Printing can be print in any order

Problem #3: Image as a graph

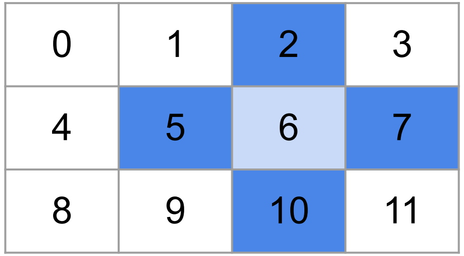
- In the **Image Processing** domain, we may need to represent the image as a graph. Assume, the image is represented originally as a rectangle RxC. This means we have R*C nodes.
- What about edges? The surrounding cells are your neighbours nodes.
  - Let’s use the **4 neighbours** cell around a cell as its neighbours to build edges
- We can **flatten** a 2D matrix so that we have an index for each cell [0, R*C-1]
- Design a program that reads 2 integers (Rows and Cols)
  - Rows, Cols >= 1
- Task: Create a graph and print it
- Make *proper graph choices*



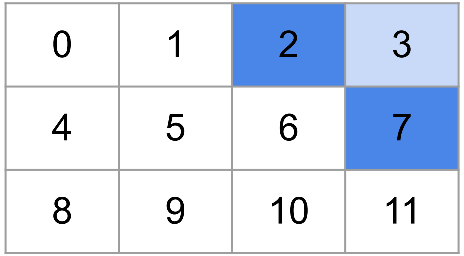
Problem #3: Image as a graph



```
3 4
Node 0 has neighbors: 4 1
Node 1 has neighbors: 5 2 0
Node 2 has neighbors: 6 3 1
Node 3 has neighbors: 7 2
Node 4 has neighbors: 8 0 5
Node 5 has neighbors: 9 1 6 4
Node 6 has neighbors: 10 2 7 5
Node 7 has neighbors: 11 3 6
Node 8 has neighbors: 4 9
Node 9 has neighbors: 5 10 8
Node 10 has neighbors: 6 11 9
Node 11 has neighbors: 7 10
```



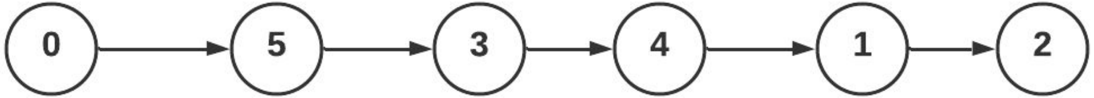
0	1	2	3
4	5	6	7
8	9	10	11



0	1	2	3
4	5	6	7
8	9	10	11

Problem #4: Print Chains

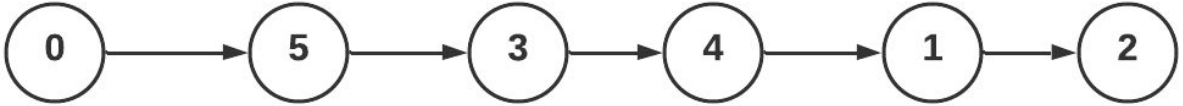
- Write a program that reads a **directed unweighted** graph
  - As we did. Read N nodes and M edges then read the M edges (from, to)
  - The graph represents a **chain**: a sequence of vertices from one vertex to another using the edges. The **length** of a chain is the **number of edges**. *A simple rooted tree.*
    - [0 ⇒ 5 ⇒ 3 ⇒ 4 ⇒ 1 ⇒ 2]

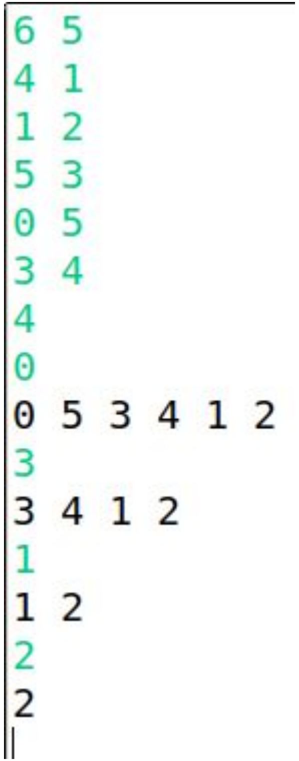


- Then read integer Q, for number of queries, then read Q integers. Each query is a node number; we want to list the path start from it until the last possible node
- Implement void print_chain(GRAPH &graph, int from)
  - It should be a simple recursive function

Example

- We read the graph, which is the chain below
  - The chain is: [0, 5, 3, 4, 1, 2]
- Then 4 queries
  - Node(0): the path until the end is the full chain
  - Node(3): Starting from 3 we can move to [4, 1, 2]
  - Node(2): Is the end node of the chain (leaf node)

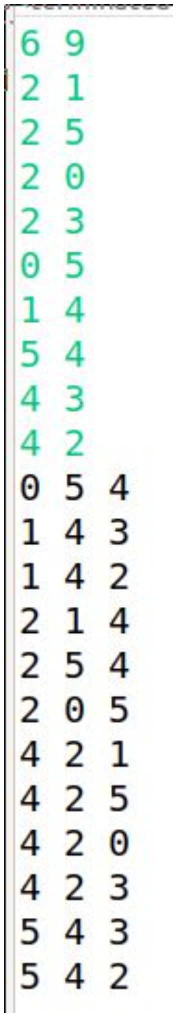




```
6 5
4 1
1 2
5 3
0 5
3 4
4
0
0 5 3 4 1 2
3
3 4 1 2
1
1 2
2
2
```

Problem #5: Print Paths of length 2

- Read a **directed** graph (as usual), and print all paths of length 2
    - Here a graph of 6 nodes and 9 edges
- 0 5 4 is 2-edges path: (0, 5) and (5, 4)
- void print_paths_len_2(GRAPH &graph)
    - Implement an iterative function



```
6 9
2 1
2 5
2 0
2 3
0 5
1 4
5 4
4 3
4 2
0 5 4
1 4 3
1 4 2
2 1 4
2 5 4
2 0 5
4 2 1
4 2 5
4 2 0
4 2 3
5 4 3
5 4 2
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."