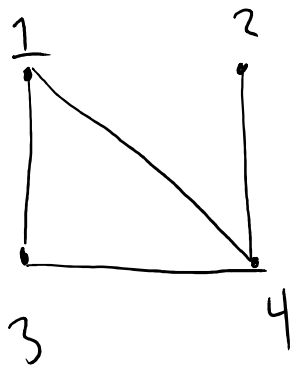
S. KIMMEL

Ways to Represent Graphs in Computer

Adjacency Matrix

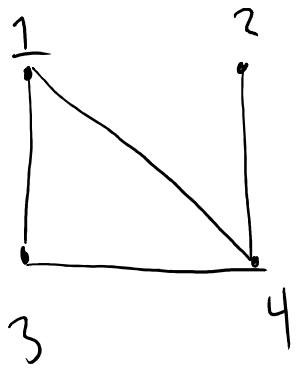
	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	1
4	1	1	1	0



S. KIMMEL

Adjacency List

Vertex	Adjacent Vertices
1	3, 4
2	4
3	1, 4
4	1, 2, 3



Store as an array of lists

ex: $A[4] = (1, 2, 3)$

$A[3, 2] = 4$ ← 4 is the 2nd vertex connected to vertex 3

[Plicker Question]

O(1) operations

Adjacency Matrix

$A[i, j]$ ← edge $(i, j) \in E$?

rows(A) ← # vertices

Adjacency List

$A[i, j]$ ← j^{th} neighbor of i

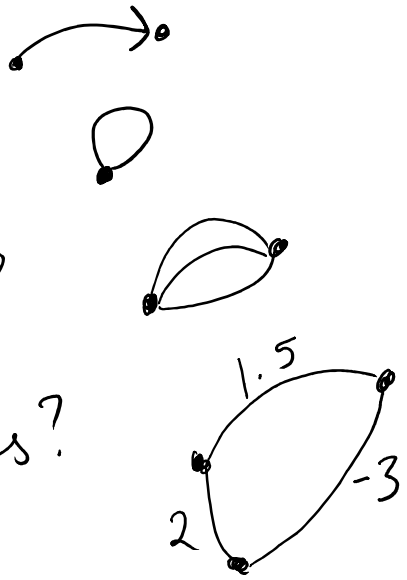
length(A) ← # vertices

length(A[i]) ← degree of i

S. KIMMEL

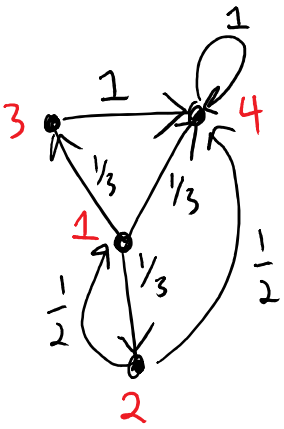
How would you represent a

- directed graph?
- graph with self-loops?
- graph with multi edges?
- graph with weighted edges?



Using Adjacency Matrix / Adjacency List?

Give representations of this graph using both approaches:



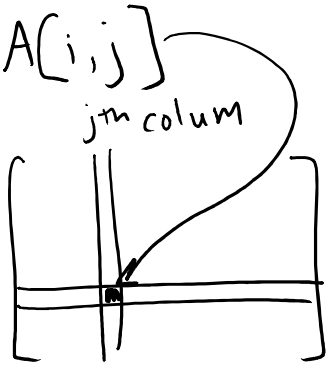
	1	2	3	4
1	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
2	$\frac{1}{2}$	0	0	$\frac{1}{2}$
3	0	0	0	1
4	0	0	0	1

V	List
1	$(2, \frac{1}{3}), (3, \frac{1}{3}), (4, \frac{1}{3})$
2	$(1, \frac{1}{2}), (4, \frac{1}{2})$
3	$(4, 1)$
4	$(4, 1)$

S.KIMMEL

Representing Adjacency Matrices + Lists in Computer

- Matrix
  - → List of Lists / Array
  - → In O(1) time can learn A[i,j]



ex:

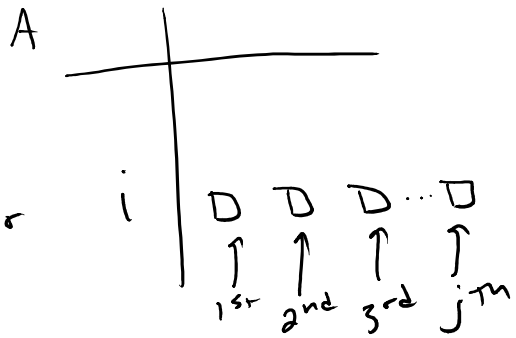
	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	1	0	0	1
4	1	1	1	0

A[2,3] = 0

- List
  - → List of Lists
  - → In O(1) time can learn

A[i,j]

↑ jth neighbor of i

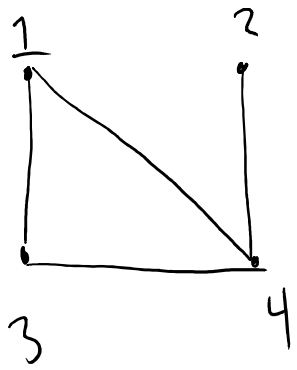


  - → In O(1) time can learn length(A[i])

S.KIMMEL

Adjacency List

Vertex	Adjacent Vertices
1	3, 4
2	4
3	1, 4
4	1, 2, 3



ex: A[4] = (1, 2, 3)

A[3, 2] = 4

4 is the 2nd vertex connected to vertex 3

[Plicker Question]

S. KIMMEL

Write pseudocode to learn degree of vertex v in an unweighted graph, and give big-O bound on time complexity.

- Adjacency Matrix

Input: vertex v, adj. matrix A for $G=(V,E)$
Output: degree of v

```
deg = 0
for i ∈ V        alternate: for i=1 to |V|
    deg = deg + A[i,v]    alt: if (A[i,v]=1): deg++
return deg.
```

Time complexity: $O(|V|)$

- Adjacency List

Input: vertex v, adj. list A for $G=(V,E)$
Output: degree of v

```
return A[v].length    Alterante: length(A[v])
```

Time Complexity: $O(1)$ ← Much faster for this problem