15.1.1
Graph notation and represetation

Notation and Convention

Notation

An edge in an undirected graphs is an unordered pair of nodes and hence it is a set. Conventionally we use uv for $\{u, v\}$ when it is clear from the context that the graph is undirected.

1. u and v are the end points of an edge $\{u, v\}$
2. Multi-graphs allow
   1. loops which are edges with the same node appearing as both end points
   2. multi-edges: different edges between same pairs of nodes
3. In this class we will assume that a graph is a simple graph unless explicitly stated otherwise.

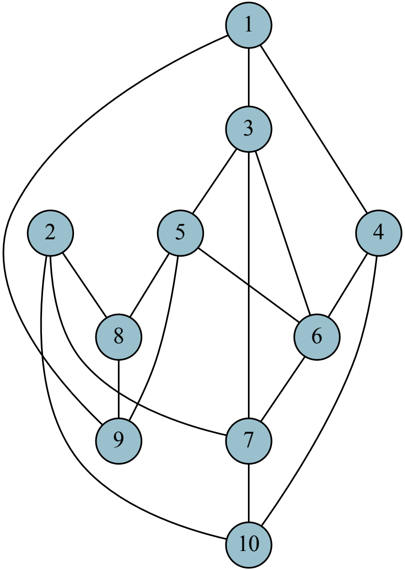
Graph Representation I

Adjacency Matrix

Represent $G = (V, E)$ with n vertices and m edges using a $n \times n$ adjacency matrix A where

1. $A[i, j] = A[j, i] = 1$ if $\{i, j\} \in E$ and $A[i, j] = A[j, i] = 0$ if $\{i, j\} \notin E$.
2. Advantage: can check if $\{i, j\} \in E$ in $O(1)$ time
3. Disadvantage: needs $\Omega(n^2)$ space even when $m \ll n^2$

Graph adjacency matrix example [10 vertices]



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

Graph Representation II

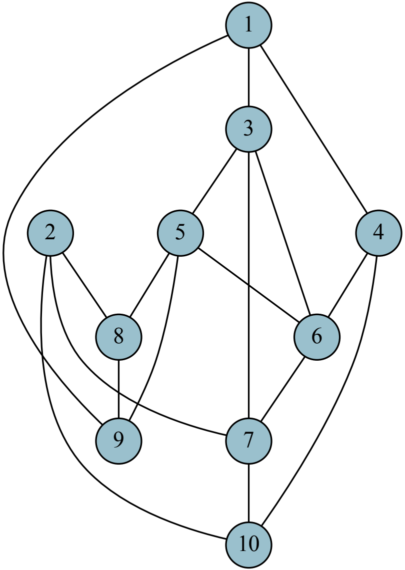
Adjacency Lists

Represent $G = (V, E)$ with n vertices and m edges using adjacency lists:

1. For each $u \in V$, $\text{Adj}(u) = \{v \mid \{u, v\} \in E\}$, that is neighbors of u. Sometimes $\text{Adj}(u)$ is the list of edges incident to u.
2. Advantage: space is $O(m + n)$
3. Disadvantage: cannot "easily" determine in $O(1)$ time whether $\{i, j\} \in E$
    1. By sorting each list, one can achieve $O(\log n)$ time
    2. By hashing "appropriately", one can achieve $O(1)$ time

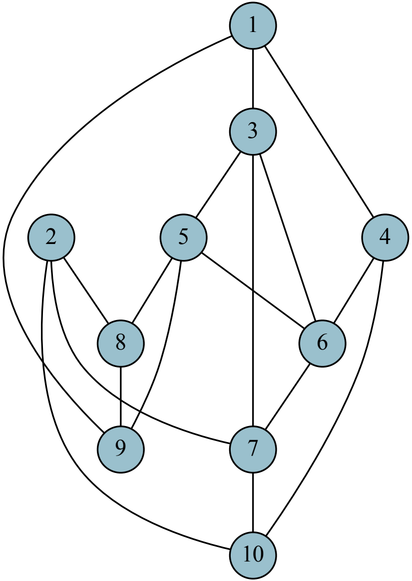
Note: In this class we will assume that by default, graphs are represented using plain vanilla (unsorted) adjacency lists.

Graph adjacency list example [10 vertices]



vertex	adjacency list
1	3, 4, 9
2	7, 8, 10
3	1, 5, 6, 7
4	1, 6, 10
5	3, 6, 8, 9
6	3, 4, 5, 7
7	2, 3, 6, 10
8	2, 5, 9
9	1, 5, 8
10	2, 4, 7

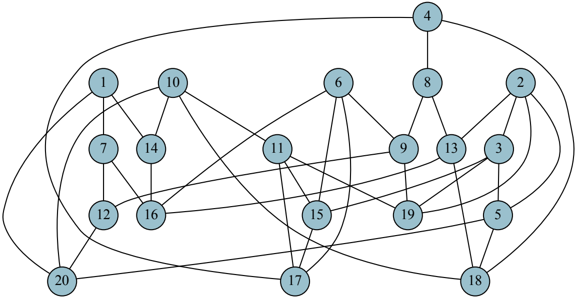
Graph adjacency matrix+list example [10 vertices]



vertex	adjacency list
1	3, 4, 9
2	7, 8, 10
3	1, 5, 6, 7
4	1, 6, 10
5	3, 6, 8, 9
6	3, 4, 5, 7
7	2, 3, 6, 10
8	2, 5, 9
9	1, 5, 8
10	2, 4, 7

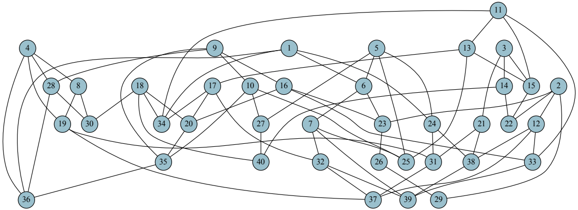
	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	0	0	0	1	0
2	0	0	0	0	0	0	1	1	0	1
3	1	0	0	0	1	1	1	0	0	0
4	1	0	0	0	0	1	0	0	0	1
5	0	0	1	0	0	1	0	1	1	0
6	0	0	1	1	1	0	1	0	0	0
7	0	1	1	0	0	1	0	0	0	1
8	0	1	0	0	1	0	0	0	1	0
9	1	0	0	0	1	0	0	1	0	0
10	0	1	0	1	0	0	1	0	0	0

Graph adjacency matrix example [20 vertices]



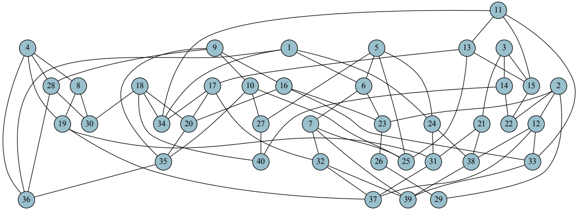
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1
2	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
3	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0
5	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
8	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1
11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0
12	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1
13	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
14	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
15	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0
17	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0
18	0	0	0	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
19	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
20	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Graph adjacency matrix example [40 vertices]



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0
6	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0
14	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
15	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
19	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
21	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
24	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
27	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
30	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0
32	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
34	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
36	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
37	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
39	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0

Graph adjacency list example [40 vertices]

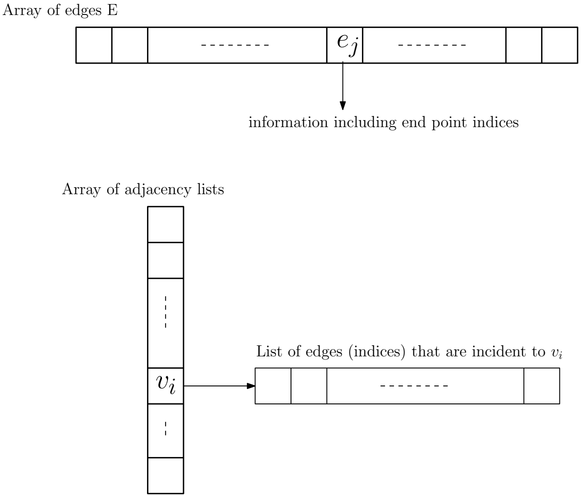


vertex	adjacency list
1	6, 24, 34, 36
2	12, 22, 23, 29
3	14, 15, 21
4	8, 19, 28, 36
5	6, 24, 25, 27
6	1, 5, 7, 23
7	6, 25, 32, 39
8	4, 19, 30
9	10, 16, 28, 35
10	9, 25, 27, 35
11	13, 15, 33, 34
12	2, 33, 37, 38
13	11, 15, 17, 25
14	3, 22, 40
15	3, 11, 13, 22
16	9, 20, 23, 33
17	13, 20, 32, 34
18	20, 30, 34, 40
19	4, 8, 31, 37
20	16, 17, 18, 35
21	3, 31, 38
22	2, 14, 15
23	2, 6, 16, 26
24	1, 5, 31, 38
25	5, 7, 10, 13
26	23, 29
27	5, 10, 40
28	4, 9, 30, 36
29	2, 26
30	8, 18, 28
31	19, 21, 24, 37
32	7, 17, 37, 39
33	11, 12, 16, 39
34	1, 11, 17, 18
35	9, 10, 20, 36
36	1, 4, 28, 35
37	12, 19, 31, 32
38	12, 21, 24, 39
39	7, 32, 33, 38
40	14, 18, 27

A Concrete Representation

- Assume vertices are numbered arbitrarily as $\{1, 2, \ldots, n\}$.
- Edges are numbered arbitrarily as $\{1, 2, \ldots, m\}$.
- Edges stored in an array/list of size m. $E[j]$ is jth edge with info on end points which are integers in range 1 to n.
- Array Adj of size n for adjacency lists. $Adj[i]$ points to adjacency list of vertex i. $Adj[i]$ is a list of edge indices in range 1 to m.

A Concrete Representation



A Concrete Representation: Advantages

- Edges are explicitly represented/numbered. Scanning/processing all edges easy to do.
- Representation easily supports multigraphs including self-loops.
- Explicit numbering of vertices and edges allows use of arrays: $O(1)$-time operations are easy to understand.
- Can also implement via pointer based lists for certain dynamic graph settings.

THE END

...

(for now)