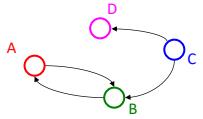
Graphs: Adjacency Matrix

- Assign each node a number from 0 to $|V| - 1$
- A $|V|$ by $|V|$ matrix `M` (2-D array) of Booleans
- `M[v][u]==true` means there is an edge from `v` to `u`



To

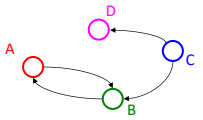
From	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F

Adjacency Matrix: Properties

- Running time to:
  - Get a vertex's out-bound edges:
  - Get a vertex's in-bound edges:
  - Decide if some edge exists:
  - Insert an edge:
  - Delete an edge:
- Space requirements:
- Better for Sparse or Dense Graphs?

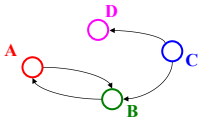
To

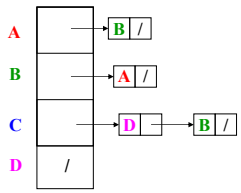
From	A	B	C	D
A	F	T	F	F
B	T	F	F	F
C	F	T	F	T
D	F	F	F	F



Graphs: Adjacency List

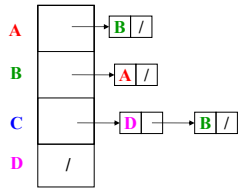
- Assign each node a number from 0 to $|V| - 1$
- An array `arr` of length $|V|$ where `arr[i]` stores a (linked) list of all adjacent vertices

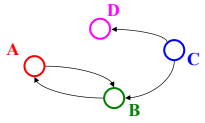




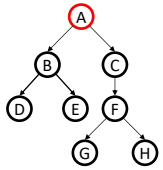
Adjacency List: Properties

- Running time to:
  - Get a vertex's out-bound edges:
  - Get a vertex's in-bound edges:
  - Decide if some edge exists:
  - Insert an edge:
  - Delete an edge:
- Space requirements:
- Better for Sparse or Dense Graphs?





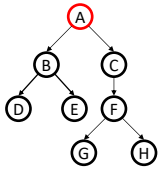
Traversal: Iterative DFS (Less common)



Order Processed:

```
IterativeDFS(Node src) {
  s = new Stack()
  s.push(src)
  mark src as visited
  while(s is not empty) {
      v = s.pop() // and "process"
      for each node u adjacent to v
      if(u is not marked)
            mark u as visited
            s.push(u)
      }
  }
}
```

Traversal: BFS (Soln.)



Order Processed:

```
BFS(Node src) {
  s = new Queue()
  s.enqueue(src)
  mark src as visited
  while(s is not empty) {
      v = s.dequeue() // and "process"
      for each node u adjacent to v
      if(u is not marked)
            mark u as visited
            s.enqueue(u)
      }
  }
}
```

Traversal: BFS Shortest Path Example

What is the shortest path from Seattle to Austin?