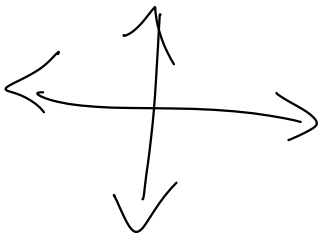
What's a graph?

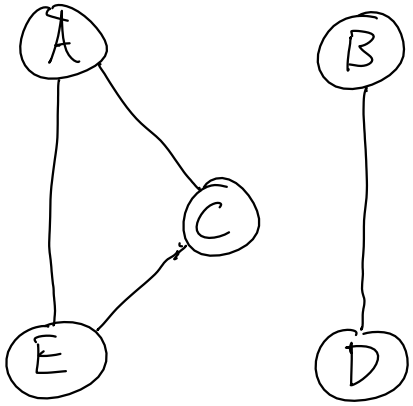


No

A graph is a set of vertices V and a set of edges E.

Each edge in E is a combination of source vertex, target vertex, weight, label

Graphs are good at expressing relationships.

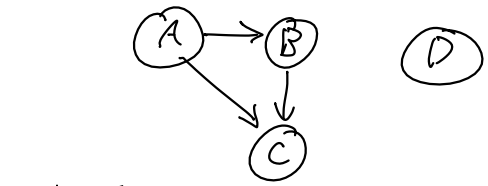


Vertices: groups of people

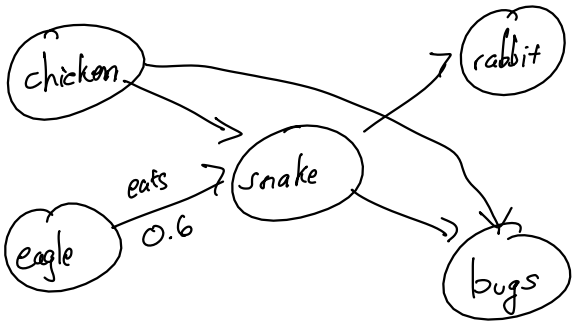
edges: same person in both groups

Graph ADT

```
void insertEdge(V source, V destination, W weight, E label)
void insertVertex(V vertex)
void removeVertex(V vertex)
void removeEdge(V source, V destination)
vector<V> getVertices()
bool containsVertex(V vertex)
bool containsEdge(V source, V destination)
vector<Edge<V,E,W>> getEdges()
vector<Edge<...>> getOutgoingEdges(V source)
vector<Edge<...>> getIncomingEdges(V destination)
Edge<V,E,W> getEdge(V source, V destination)
vector<V> getNeighbors(V vertex)
```



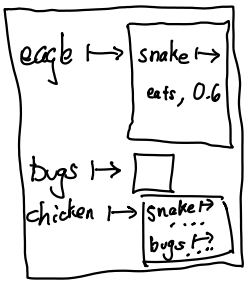

```
template <typename V, typename E, typename W>
class Edge {
    V src;
    V dest;
    E lbl;
    W weight;
};
```



What implementation should we use?

#1 `HashTable<V, HashTable<V, Edge<V,E,W>>>`

(source: first V; destination: second V)



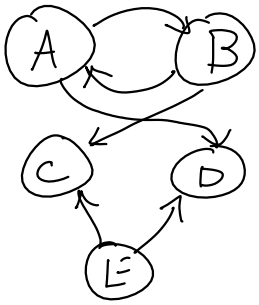
Adjacency List —— Adjacency Dictionary

#2 `pair<vector<V>, vector<Edge<V,E,W>>>`

#3 `bool[][], Dictionary<V, int>`

Adjacency Matrix

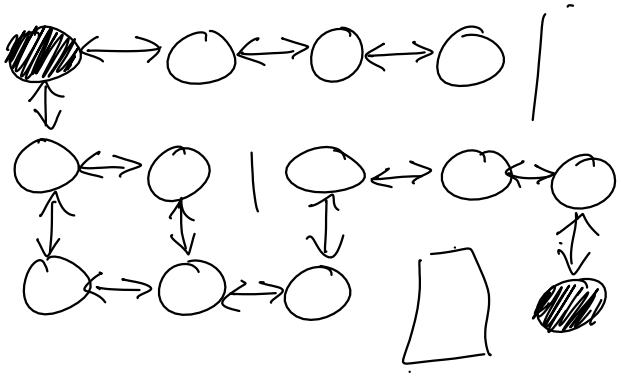
	A	B	C	D	E
A		✓		✓	
B	✓			✓	
C					
D					✓
E					✓

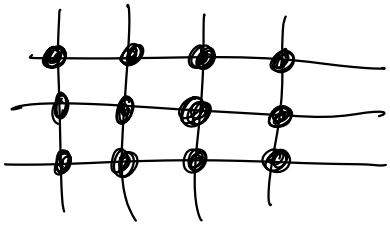


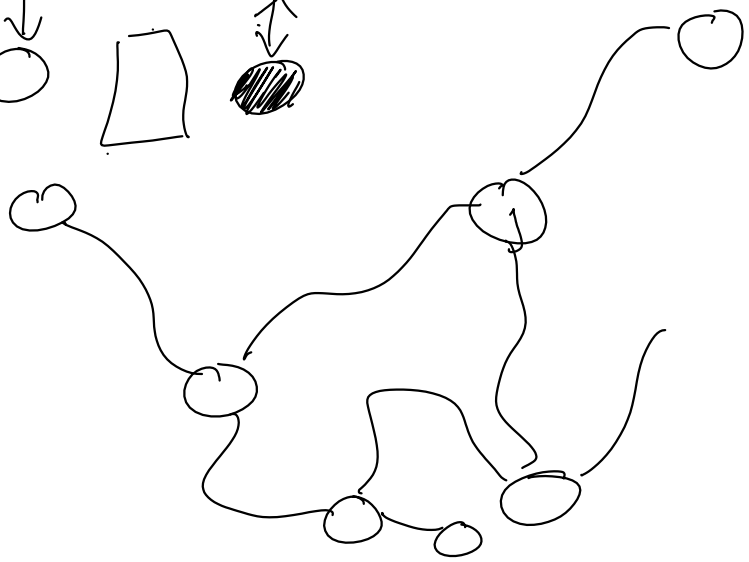
#4 `HashTable<V, pair<HashTable<V,Edge...>, HashTable<V,Edge...>>>`

Graph ADT applications

* any path from V_1 to V_2 ?
* shortest path ?
* cheapest path ?







```
is Path DFS ( Graph graph, V source, V dest )
    Stack<V> stack
    Set<V> visited
    stack.push(source)
    visited.add(source)
    while stack is not empty
        V current ← stack.pop()
        if current is dest:
            return true
        for each neighbor of current in graph:
            if visited does not contain neighbor
                stack.push(neighbor)
                visited.add(neighbor)
    return false
```