Algorithms and Data Structures II

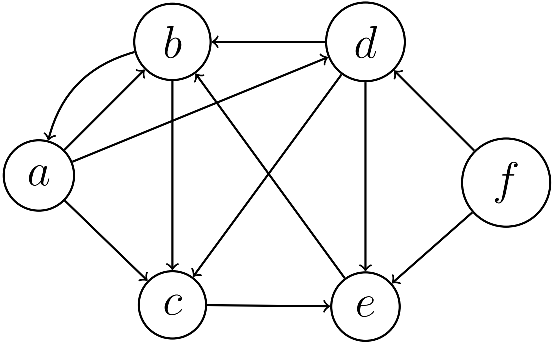
Lecture 3:

Graphs, Definitions and Representations

https://elms.u-aizu.ac.jp

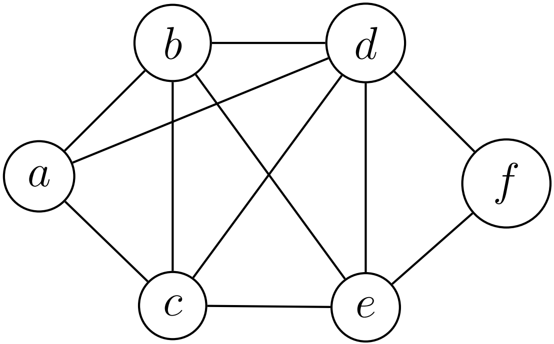
Directed Graphs

- A graph $G(V, E)$ consists of a set of vertices (nodes) V and a set of edges E. If the edges are ordered pairs (v, w) of vertices then the graph is directed (edges are also called arcs)



Undirected Graphs

- If the edges are unordered pairs (sets) of distinct vertices (also denoted by (v, w)) then the graph is undirected.



Graph Representations

- In a directed graph $G(V, E)$, if (v, w) is an edge in E then we say vertex w is adjacent to v. We also say edge (v, w) is from v to w. The number of vertices adjacent to v is the (out-) degree of v.
- In an undirected graph $G(V, E)$, (w, v) and (v, w) are the same edge. w is adjacent to v if (v, w) is in E. The degree of a vertex is the number of vertices adjacent to it. We say the edge (v, w) is incident on v.

Path

- A path in a graph is a sequence of edges of the form $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$. The path is from v_1 to v_n of length $n - 1$.
- As a special case, a single vertex denotes a path of length 0 from itself to itself.

Simple, Cycle

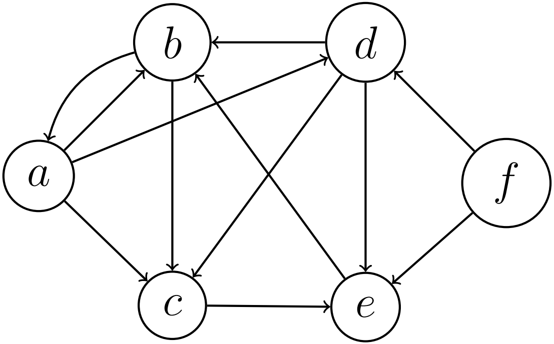
- A path is simple if all edges and all vertices on the path, except possibly the first and the last vertices, are distinct.
- A cycle is a simple path of length at least 1 which begins and ends at the same vertex. In an undirected graph, a cycle must be of length at least 3.

Adjacency Matrix

- One of the common representations for a graph $G(V, E)$ is the adjacency matrix, a $|V| \times |V|$ matrix A of 0's and 1's, where $A[i, j] = 1$ iff there is an edge from vertex i to vertex j. The adjacency matrix requires $O(|V|^2)$ memory space for graph $G(V, E)$.

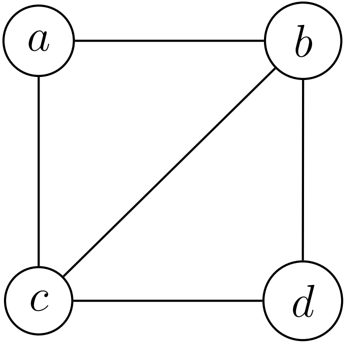
Graphs and adjacency matrices

	a	b	c	d	e	f
a	0	1	1	1	0	0
b	1	0	1	0	0	0
c	0	0	0	0	1	0
d	0	1	1	0	1	0
e	0	1	0	0	0	0
f	0	0	0	1	1	0



Graphs and adjacency matrices

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0



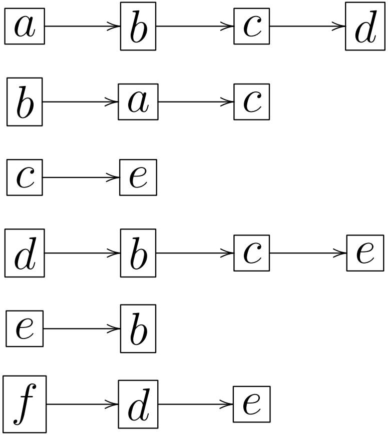
Adjacency List

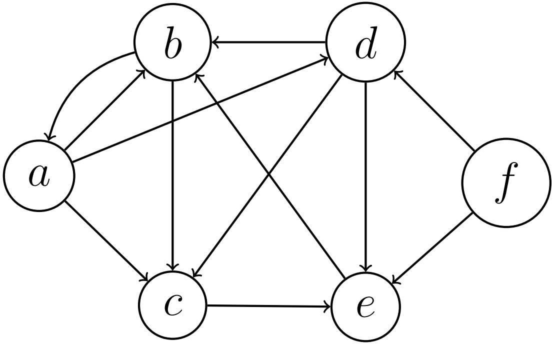
- Another possible representation for a graph is adjacency list.
- An adjacent list for a vertex is a list of all vertices adjacent to it. A graph can be represented by $|V|$ adjacency lists, one for each vertex.

Representation of Adjacency Lists

- Adjacency lists require $O(|V| + |E|)$ memory space for graph $G(V, E)$. Adjacency lists are often used for sparse graph $G(V, E)$ with $|E| \ll |V|^2$.
- Note that $|E| \leq |V|(|V| - 1)$ for directed graph and $|E| \leq (|V|(|V| - 1))/2$ for undirected graph.

Graphs and adjacency lists





Search Process

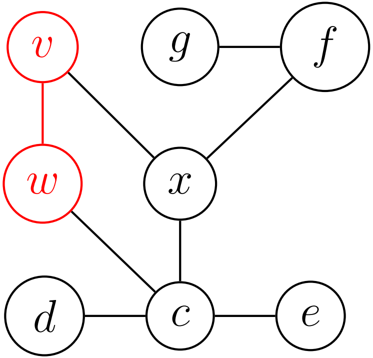
Once the representation of a graph G is established, the types(directed or undirected) of graph is not matter on the next search processes, DFS & BFS: Just process on adjacency matrix or list!

Depth-first Search

- Depth-first search (DFS) is a natural way to visit every vertex and check every edge in the graph systematically.
- It has applications for many graph problems, such as checking the connectivity, finding the connected components or cycles, and so on, in graphs.

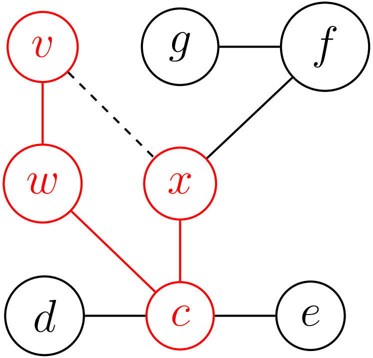
First step of DFS

- DFS visits graph $G(V, E)$ in the following way. Select a vertex v and visit v. (v is also called the root of the DFS search tree.) Then select any edge (v, w) incident on v and visit w.



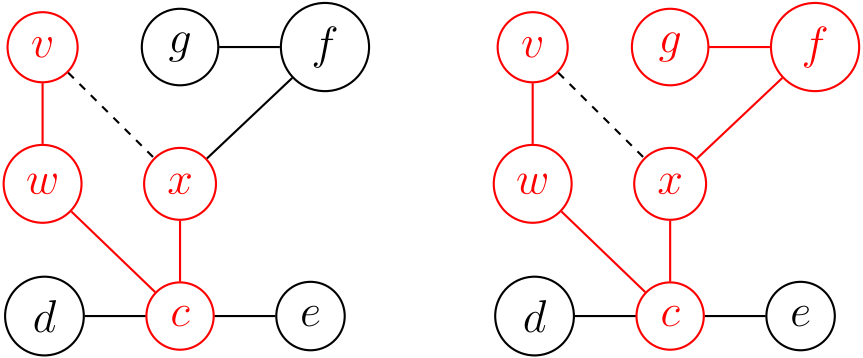
Second step of DFS

- In general, suppose x is the most recently visited vertex. The search is continued by selecting some unexplored edge (x, y) incident to x.



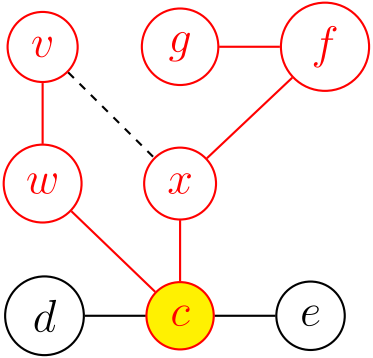
Next steps of DFS

- If y has been previously visited, then we find another new edge incident on x. If y has not been previously visited, then we visit y and begin the search anew starting at vertex y.



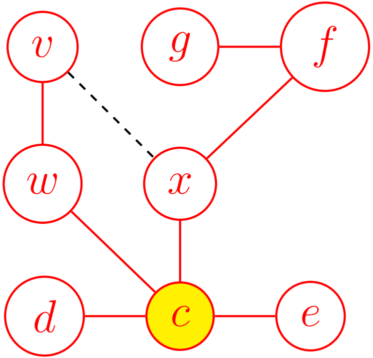
Returning to c

- After completing the search through all paths beginning at the y, the search returns to x, the vertex from which y was first reached.



finaly...

- The process of selecting unexplored edges incident on x is continued until the list of these edges is exhausted.



DFS

- This method is called depth-first search since we continue searching in the forward (deeper) direction as long as possible.

DFS Representation

- The following is a recursive procedure written in C that realizes DFS for graphs represented by adjacency matrices.
- In the procedure $val[|V|+1]$ is used to indicate if a vertex has been visited. Initially $val[|V|+1]$ is set to all zero, so $val[k]=0$, indicating vertex k has not been visited yet.

DFS code

- If vertex k is the i-th visited vertex, then $val[k]$ is set to i, $1 \leq i \leq |V|$.

```
DFS(int k){
  int t;
  val[k] = ++i;
  for (t=1; t <= |V|; t++)
    if (A[k][t] != 0 && val[t] == 0) DFS(t);
}
```

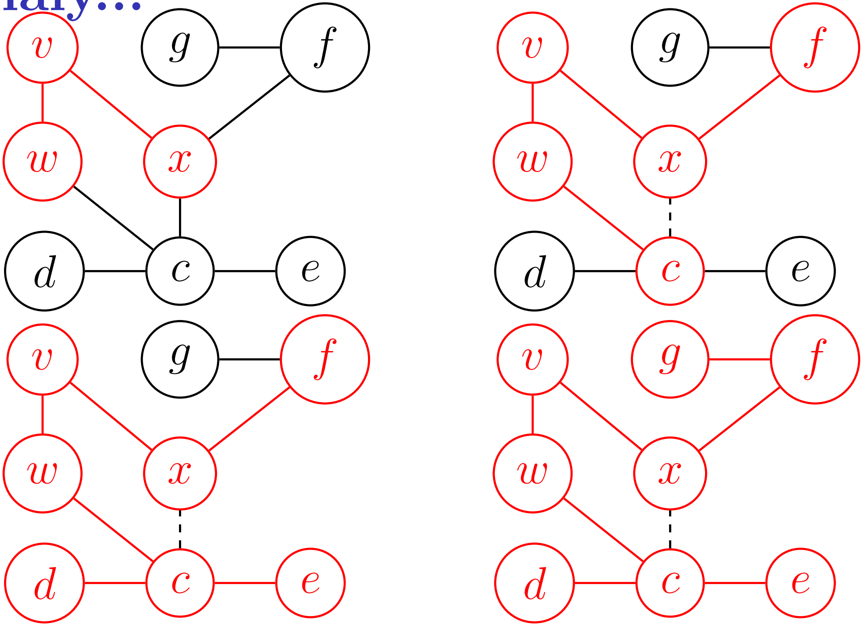
Numbering nodes by visiting order or reversed order

- Numbering the nodes by visiting order (or somehow) such as
  $v = 1, w = 2, c = 3, \ldots, g = 6, d = 7, e = 8$
  or
  $v = 8, w = 7, c = 6, \ldots, g = 3, d = 2, e = 1$
  is sometimes useful in Applications for finding strongly connected components or finding articulation points, etc.

Breadth-first Search

- Another classical graph-traversal algorithm is breadth-first search (BFS).
- BFS searches the graph as follows:
  1. Select a vertex v and visit v.
  2. For all edges (v, w), visit the vertices w and put w into a first-in first-out (FIFO) queue.
  3. After w's are visited for all (v, w), we select a vertex w from the FIFO queue, visit all unvisited vertices x for (w, x), and put x into the queue.
  4. Repeat the above process until all the vertices in the graph are visited.

finaly...



BFS

- BFS visits all successors of a visited vertex before visiting any successors of any of those successors.
- This is in contradistinction to the DFS which visits the successors of a visited vertex before visiting any of its “brothers”.

DFS & BFS

- Whereas DFS tends to create very long, narrow trees, BFS tends to create wide, short trees.
- Exercise Problem 2 gives a C program for BFS.

Connectivity of Graphs

- Let $G(V, E)$ be a directed graph. Two vertices u and v of G is strongly connected if there is a path from u to v and a path from v to u.
- A strongly connected component of G is a maximal subgraph of G whose vertices are all strongly connected with each other.

Finding strongly connected components

- The problem of finding the strongly connected components of G can be done by DFS as follows:
  1. The first step is to perform a DFS on G.
  2. The next step is to reverse all edges of G creating an inverse graph G_r, taking transpose of adjacent matrix of G.
  3. Finally, a DFS on G_r is performed, beginning at the vertex with the lowest label given in the DFS on G .

Finding strongly connected components 2

- If this search does not visit all the vertices of G_r, the unvisited vertex with the lowest label is chosen and the search is resumed there, carrying on in this way until all vertices of G_r have been visited.

connected, k-connected

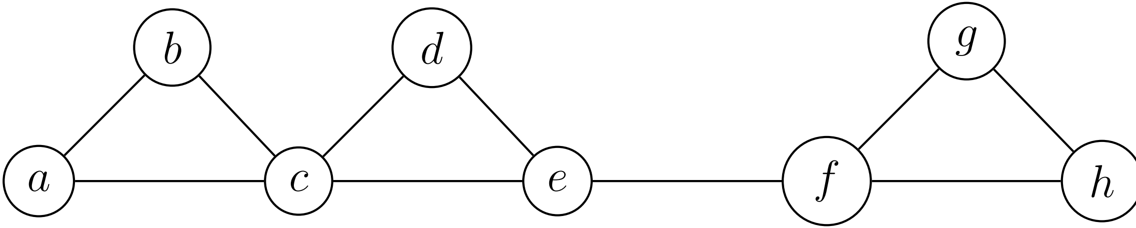
- An undirected graph G is called connected if there is a path between any pair of vertices in G.
- G is called k-connected if the removal of any $k-1$ vertices leaves the remaining subgraph connected.
  - 1-connected is connected;
  - 2-connected (biconnected) means that one vertex failure can be tolerated.

articulation point

- If a graph is connected but not biconnected, it has articulation points: vertices whose removal would disconnect the graph.

Example of articulation point

- For example, the articulation points of the graph are c, e, and f. Articulation points can be found by DFS.



Finding articulation point

- For example, the articulation points of the graph are c, e, and f. Articulation points can be found by DFS.