9.5 Graph Connectivity

For an undirected graph

- **Connected vertices**: Vertices \( x \) and \( y \) are connected if there is a path from \( x \) to \( y \).
  - Having a walk from \( x \) to \( y \) is sufficient: A walk can always be truncated to a path.)
- **Connected graph**: Every pair of distinct vertices is connected. (**Disconnected graph** otherwise.)
- **Connected component**: A connected subgraph of maximal size (can't add any more vertices and keep it connected).
- **Isolated vertex**: Is not connected to any other vertex.

**Vertex connectivity**
- Graph \( G \) is **\( k \)-vertex-connected**
  - iff (\( V \) set of \( k-1 \) vertices, removing [and their edges] them leaves \( G \) connected)
  - iff (\( \exists \) set of \( k-1 \) vertices whose removal disconnects \( G \))
  - (If \( \exists \) \( k-1 \) vertices whose removal disconnects \( G \), then \( G \) is not **\( k \)-vertex-connected**)
- **Vertex connectivity**: \( \kappa(G) \)
  - = largest \( k \) such that \( G \) is \( k \)-vertex-connected
  - = minimum number of vertices whose removal guarantees disconnection of the graph

**Minimum degree of \( G \)**: \( \delta(G) \) = the min of all the vertex degrees of \( G \)
- Note: \( \kappa(G) \leq \delta(G) \)

**Edge connectivity**: \( k \)-edge-connectivity and **edge connectivity** \( \lambda(G) \) are similar to vertex connectivity (except when you remove an edge, you leave the vertices behind). \( \lambda(G) \leq \delta(G) \).

**Questions**

1. Draw a graph with vertices \( V = \{ 1, 2, \ldots, 10 \} \), edges \( E = \{ \{ 1, 2 \}, \{ 2, 3 \}, \{ 5, 7 \}, \{ 6, 8 \}, \{ 6, 9 \}, \{ 6, 10 \}, \{ 9, 10 \} \} \). What are its connected components? Any isolated vertices?

2. Is a cyclic graph 1-vertex-connected? 2-vertex-connected? What is \( \delta(a \) cyclic graph)? \( \kappa(a \) cyclic graph)?
Illinois Institute of Technology

Activity 22

3. What are vertex and edge connectivity of graph to right? (Assume there’s a vertex at each intersection.)

4. (Ex 9.5.2) What are vertex and edge connectivity of graph to right?

9.6 Graph Coloring

- A graph coloring is a map of vertices to values called colors. If #colors = k, it’s a \textit{k-coloring}
  - Colors can be any values, not just Red, Blue, etc.
  - In a valid graph coloring, the endpoints of each edge have different colors.
  - $X(G) =$ the chromatic number of graph $G =$ the smallest $k$ such that there is a valid $k$-coloring of $G$.

Questions

5. How many colors does $K_n$ = the clique (fully connected graph) with $n$ vertices require? The \textbf{clique number} $\omega(G)$ is the largest $n$ such that $K_n$ is a subgraph of $G$. What is the relationship between $\omega(G)$ and $X(G)$?
6. How many colors does \( C_3 \) (the cycle with 3 vertices) require? \( C_4 \)? \( C_5 \)? Generalize this observation: \( X(C_n) = ?? \).

7. How many colors does the bipartite graph \( K_{m,n} \) need?

8. Give valid colorings for these graphs:

9. One example of graph coloring is with scheduling conflicts: You have a vertex for each event and an edge between each pair of vertices that interfere with each other. What does \( X(G) \) represent?
**Greedy Coloring Algorithm**
- Determining $X(G)$ is hard in general. The greedy coloring algorithm uses a sequence of locally-optimal decisions to color a graph. Can do a pretty reasonable job.
- For graph with $n$ vertices, let $\{1, 2, \ldots, n\}$ be the set of colors (we may not use them all).
- Order the vertices in any arbitrary order. For each vertex $v$ in order:
  - Take the set of colors assigned to the neighbors of $v$, find the lowest color not in the set, and assign it to $v$.
  - (E.g., if the neighbors’ colors are $\{1, 3\}$, then $v$ gets color 2. If no neighbors are colored yet, $v$ gets color 1.)

Questions
9. Color this graph using the greedy coloring algorithm

![](image)

10. Number the vertices of this graph left-to-right, top-to-bottom and apply the greedy coloring algorithm; what do you get?

![](image)

11. Number the vertices of this graph clockwise starting at the top (outer vertices first, then inner vertices) and apply the greedy coloring algorithm; what do you get?

![](image)