Graphs

• (p 186) **Graphs** $G = (V,E)$
  • set $V$ of **vertices**, each with a unique name
    • Note: book calls vertices as **nodes**
  • set $E$ of **edges** between vertices, each encoded as tuple of 2 vertices as in $(u,v)$

• Edges may be **directed** (from $u$ to $v$) or **undirected**
  • Undirected edge eqvt to pair of directed edges

• Example of undirected graph:

  ![Graph](image)

  $G =$

  \[
  (1,2,3,4)((1,2),(2,3),(3,1),(1,4))
  \]

• **Labeled graph**: each edge has a “name”

• **Weighted graph**: each edge has numerical value
  • E.g. in graph of $V$=cities, weight on $(u,v)$ is distance from $u$ to $v$
Terminology about Graphs (see P. 12)

- **Outdegree** of a vertex u: # of edges leaving it
  - i.e. $|\{(u, v)\}|$ for some v
- **Indegree** of a vertex v: # of edges entering it
  - i.e. $|\{(u, v)\}|$ for some u
  - For undirected graph outdegree(u) = indegree(u) = $\text{degree}(u)$
- **K-regular**: every vertex has degree k (p. 13)
- **Subgraph**: subsets $V'$ and $E'$ of V and E where all edges in $E'$ are between vertices in $V'$
- **Path of length k**: from u to v if a set of k edges in G $(u_i,v_i), 1 \leq i \leq k$, where $u_1 = u, v_i = u_{i+1}$, and $v_k = v$
- **Simple path**: no vertices are repeated
- **Shortest path**: between u & v is simple path of shortest length
- **Diameter**: longest shortest path between any 2 vertices
- **Hamiltonian Path**: goes thru every vertex once
- **Cycle**: a path exists from u back to u
- **Connected** iff every vertex can be reached from every other vertex by some path
- **Strongly connected** iff a directed path from each vertex to every other
- Tree: graph with no simple paths: Has root & leaves
- **Vertex Cover of size k:** subset of k vertices where every edge touches at least one of them (p. 312)
- **K-Clique:** subset of k vertices where there is an edge between every pair in it.
- Two graphs G & H are **isomorphic** if vertices of one can be reordered so that graphs are identical
- **Spanning Tree:** subgraph that forms a tree that includes all vertices, but with minimum # of edges
- **Flow Network:** directed weighted graph where:
  - Each edge can carry a “flow” (a number)
  - Each edge has a “capacity” (max possible flow value)
  - For each vertex, \( \sum \text{ incoming flow} = \sum \text{ outgoing flow} \)
    - Except for some **source** that generates out flow
    - And some **sink** which has no outgoing flow
Graph Data Structures

- Assume $|V| = N$, $|E| = M$

- **Adjacency Matrix**: $N \times N$ Boolean matrix $A$ where
  - $A[u,v] = 1$ if $(u,v)$ in $E$
  - $A[u,v] = 0$ otherwise
  - If graph is weighted, $A[u,v] = \text{weight on } (u,v)$

- **CSR (Compressed Sparse Row)**: 3 vectors
  - $A$: $M$ vector of weights (one per edge)
  - $JA$: $M$ vector of vertex indecies
  - $IA$: $N+1$ vector of indices into $A$, $JA$
    - $IA[u] = \text{index into } A, JA \text{ for } 1^{st} \text{ edge from } u$
    - $JA[IA[u]] \text{ thru } JA[IA[u+1]-1] \text{ are the } v's \text{ for edges } (u,v)$
    - Matching elements in $A$ are weights

- Lots of variations
Spanning Trees & BFS

- Common problem: starting at some vertex \( u \), find tree that reaches as many vertices as possible
  - If all vertices reachable, then a “spanning tree”
- Common algorithm: **BFS (Breadth First Search)**
  - **Frontier**: set of all vertices that have been reached “for 1\(^{st}\) time”
    - At start, just \( u \)
  - Also each vertex can be marked as “touched” or not
    - At start only \( u \) so marked
  - For each vertex in current frontier:
    - Follow each outgoing edge
    - If other vertex is not touched:
      - Mark as touched
      - Add to new frontier
    - When frontier empty, swap with new frontier and repeat
- Option: when first touched, mark vertex with “level”
  - Level = # of edges from root \( u \)
  - At end, final level = diameter of reached subgraph
- Clearly polynomial time
• Basis for the **GRAPH500** benchmark
  • [www.graph500.org](http://www.graph500.org)
  • Search graphs with up to trillions of vertices
  • Literally thousands of different implementations on different computers, esp. parallel
  • Established by an **ND quad-domer**

• **Beamer’s Algorithm:**
  • When frontier too large, instead explore all non-touched
  • If any of them have an edge to current touched, mark them as touched
  • Can reduce time by 10X
Maximum Flow Problems

• **Max flow**: given a flow network, find largest flow from source to sink where no edge exceeds its capacity

• **Ford-Fulkerson Algorithm**
  - Define:
    - \( c(u,v) = \) capacity of edge \((u, v)\)
    - \( f(u,v) = \) flow on edge \((u,v)\)
    - **Residual Network** \( G_f(V,E_f) = \) network with capacity
      \[ c_f(u,v) = c(u,v) - f(u,v) \] (“residual flow”)

**Output** Compute a flow \( f \) from \( s \) to \( t \) of maximum value

1. \( f(u, v) \leftarrow 0 \) for all edges \((u, v)\)
2. While there is a path \( p \) from \( s \) to \( t \) in \( G_f \), such that \( c_f(u,v) > 0 \) for all edges \((u, v) \in p\):
   1. Find \( c_f(p) = \min \{ c_f(u,v) : (u,v) \in p \} \)
   2. For each edge \((u,v) \in p\)
      1. \( f(u,v) \leftarrow f(u,v) + c_f(p) \) (Send flow along the path)
      2. \( f(v,u) \leftarrow f(v,u) - c_f(p) \) (The flow might be “returned” later)

- "\( \leftarrow \)" denotes assignment. For instance, "largest \( \leftarrow \) item" means that the value of \( \text{largest} \) changes to the value of \( \text{item} \).
- "return" terminates the algorithm and outputs the following value.

- “Finding paths” can use BFS
- Each new path chosen is called “augmenting path”
Bipartite Graphs

- References:
  - https://en.wikipedia.org/wiki/Bipartite_graph

- Bipartite Graph:
  - 2 disjoint sets of vertices $U$ and $V$, called “parts”
  - Every edge connects vertex from $U$ with one from $V$

- Matching: subset of edges where no two edges share an endpoint
  - Maximal Matching: edge set is largest possible matching
  - Perfect Matching: $|U| = |V| = |\text{matching set}|$

- Examples of Bipartite Graphs:
  - Athletes and Teams they played with
  - Actors and Movies they acted in
  - Trains and Stations
  - Social networks
  - All graphs that are trees
  - Graphs that form single cycles with even # vertices
  - Graphs that can be written “on a 2D plane” without edges crossing
• Properties
  • No odd cycles
  • 2-colorable

• **Kőnig's theorem**: # of edges in maximum matching = # of vertices in a minimum vertex cover

• **O(sqrt(V)E) algorithm**: **Hopcroft-Karp**
  • Similar to Ford-Fulkerson

**Input**: Bipartite graph $G(U \cup V, E)$
**Output**: Matching $M \subseteq E$

$M \leftarrow \emptyset$

repeat
  \[ P \leftarrow \{P_1, P_2, \ldots, P_k\} \] maximal set of vertex-disjoint shortest augmenting paths
  \[ M \leftarrow M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) \]
until $P = \emptyset$

• $P_k$ = augmenting path of length $k$
• “Free vertex” – does not appear in current matching $M$

Use BFS to partition vertices into layers
  • Current free vertices from $U$ the 1$^{st}$ layer
  • Alternate between matched and unmatched vertices
  • Ends at level $k$ when $\geq 1$ free vertices in $V$ are reached
  • Define free vertices $v$ from $V$ into a set $F$
    • A shortest augmenting path (of length $k$)
  • Use this to augment matching set $M$