Elementary Graph Algorithms

Dr. G P Gupta

Graphs

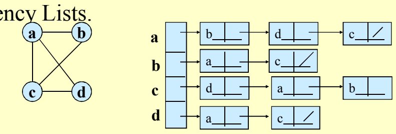
- *Graph G* = (*V*, *E*)
  - » *V* = set of vertices
  - » *E* = set of edges $\subseteq (V \times V)$
- Types of graphs
  - » Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops.)
  - » Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed.
  - » Weighted: each edge has an associated weight, given by a weight function $w : E \rightarrow \mathbf{R}$.
  - » Dense: $|E| \approx |V|^2$.
  - » Sparse: $|E| << |V|^2$.
- $|E| = O(|V|^2)$

Graphs

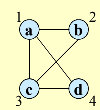
- If $(u, v) \in E$, then vertex v is adjacent to vertex u.
- Adjacency relationship is:
  - » Symmetric if *G* is undirected.
  - » Not necessarily so if *G* is directed.
- If *G* is connected:
  - » There is a path between every pair of vertices.
  - » $|E| \geq |V| - 1$.
  - » Furthermore, if $|E| = |V| - 1$, then *G* is a tree.
- Other definitions in Appendix B (B.4 and B.5) as needed.

Representation of Graphs

- Two standard ways.
  - » Adjacency Lists.



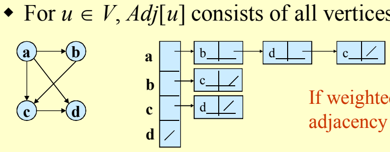
  - » Adjacency Matrix.



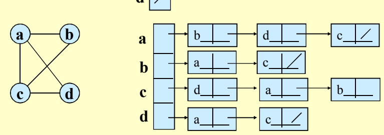
	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

Adjacency Lists

- Consists of an array *Adj* of $|V|$ lists.
- One list per vertex.
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u.



If weighted, store weights also in adjacency lists.



Storage Requirement

- For directed graphs:
  - » Sum of lengths of all adj. lists is

$$\sum_{v \in V} \text{out-degree}(v) = |E|$$

No. of edges leaving *v*

  - » Total storage: $\Theta(V+E)$
- For undirected graphs:
  - » Sum of lengths of all adj. lists is

$$\sum_{v \in V} \text{degree}(v) = 2|E|$$

No. of edges incident on *v*. Edge (*u*,*v*) is incident on vertices *u* and *v*.

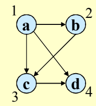
  - » Total storage: $\Theta(V+E)$

Pros and Cons: adj list

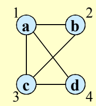
- Pros
  - » Space-efficient, when a graph is sparse.
  - » Can be modified to support many graph variants.
- Cons
  - » Determining if an edge $(u,v) \in G$ is not efficient.
    - Have to search in u's adjacency list. $\Theta(\text{degree}(u))$ time.
    - $\Theta(V)$ in the worst case.

Adjacency Matrix

- $|V| \times |V|$ matrix A.
- Number vertices from 1 to $|V|$ in some arbitrary manner.
- A is then given by: $$A[i,j] = a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

$A = A^{T}$ for undirected graphs.

Space and Time

- **Space:** $\Theta(V^2)$.
  - » Not memory efficient for large graphs.
- **Time:** to list all vertices adjacent to u: $\Theta(V)$.
- **Time:** to determine if $(u, v) \in E$: $\Theta(1)$.
- Can store weights instead of bits for weighted graph.

Graph-searching Algorithms

- Searching a graph:
  - » Systematically follow the edges of a graph to visit the vertices of the graph.
- Used to discover the structure of a graph.
- Standard graph-searching algorithms.
  - » Breadth-first Search (BFS).
  - » Depth-first Search (DFS).

Breadth-first Search

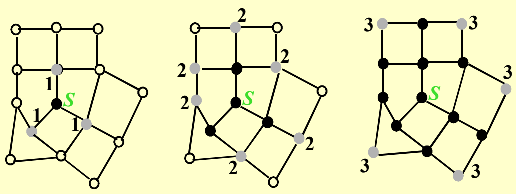
- **Input:** Graph $G = (V, E)$, either directed or undirected, and ***source vertex*** $s \in V$.
- **Output:**
  - » $d[v]$ = distance (smallest # of edges, or shortest path) from s to v, for all $v \in V$. $d[v] = \infty$ if v is not reachable from s.
  - » $\pi[v] = u$ such that (u, v) is last edge on shortest path $s \leadsto v$.
    - u is v's predecessor.
  - » Builds breadth-first tree with root s that contains all reachable vertices.

Definitions:
Path between vertices u and v: Sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $u=v_1$ and $v=v_k$, and $(v_i,v_{i+1}) \in E$, for all $1 \le i \le k$-1. Error!
Length of the path: Number of edges in the path.
Path is simple if no vertex is repeated.

Breadth-first Search

- Expands the frontier between discovered and undiscovered vertices uniformly across the breadth of the frontier.
  - » A vertex is "discovered" the first time it is encountered during the search.
  - » A vertex is "finished" if all vertices adjacent to it have been discovered.
- Colors the vertices to keep track of progress.
  - » White – Undiscovered.
  - » Gray – Discovered but not finished.
  - » Black – Finished.
    - Colors are required only to reason about the algorithm. Can be implemented without colors.

BFS for Shortest Paths



● Finished ● Discovered ○ Undiscovered

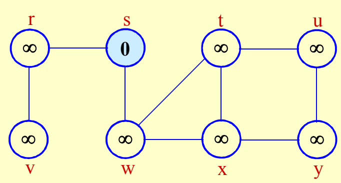
```
BFS(G,s)
for each vertex u in V[G] – {s}
    do color[u] ← white
       d[u] ← ∝
       π[u] ← nil
color[s] ← gray
d[s] ← 0
π[s] ← nil
Q ← Φ
enqueue(Q,s)
while Q ≠ Φ
    do u ← dequeue(Q)
       for each v in Adj[u]
           do if color[v] = white
                 then color[v] ← gray
                      d[v] ← d[u] + 1
                      π[v] ← u
                      enqueue(Q,v)
       color[u] ← black
```

white: undiscovered
gray: discovered
black: finished

Q: a queue of discovered vertices
color[*v*]: color of v
d[*v*]: distance from s to v
π[*u*]: predecessor of v

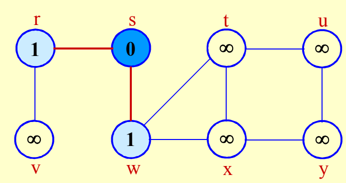
Example: animation.

Example (BFS)



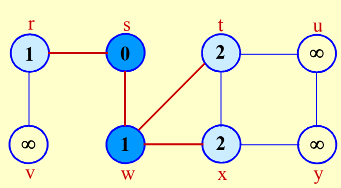
Q: s
0

Example (BFS)



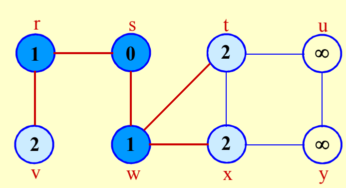
Q: w r
1 1

Example (BFS)



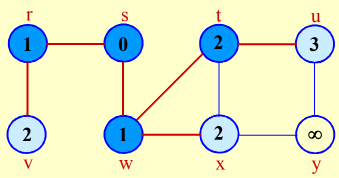
Q: r t x
1 2 2

Example (BFS)



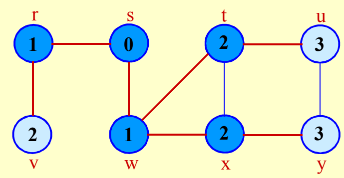
Q: t x v
2 2 2

Example (BFS)



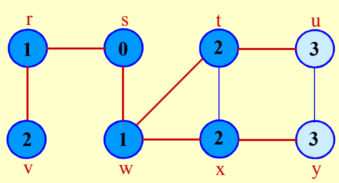
Q: x v u
2 2 3

Example (BFS)



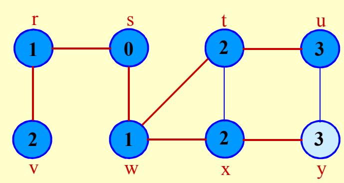
Q: v u y
2 3 3

Example (BFS)



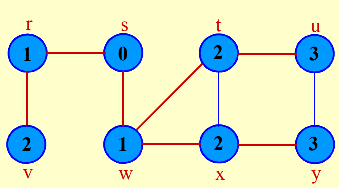
Q: u y
3 3

Example (BFS)



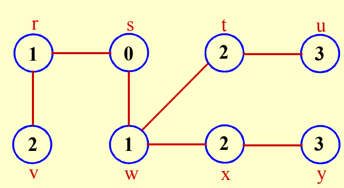
Q: y
3

Example (BFS)



Q: ∅

Example (BFS)



BF Tree

Analysis of BFS

- Initialization takes $O(V)$.
- Traversal Loop
  - » After initialization, each vertex is enqueued and dequeued at most once, and each operation takes $O(1)$. So, total time for queuing is $O(V)$.
  - » The adjacency list of each vertex is scanned at most once. The sum of lengths of all adjacency lists is $\Theta(E)$.
- Summing up over all vertices => total running time of BFS is $O(V+E)$, linear in the size of the adjacency list representation of graph.
- Correctness Proof
  - » We omit for BFS and DFS.
  - » Will do for later algorithms.

Breadth-first Tree

- For a graph $G = (V, E)$ with source s, the **predecessor subgraph** of G is $G_\pi = (V_\pi, E_\pi)$ where
  - » $V_\pi=\{v\in V : \pi[v] \neq \text{NIL}\}\cup\{s\}$
  - » $E_\pi=\{(\pi[v],v)\in E : v \in V_\pi - \{s\}\}$
- The predecessor subgraph G_π is a **breadth-first tree** if:
  - » V_π consists of the vertices reachable from s and
  - » for all $v\in V_\pi$, there is a unique simple path from s to v in G_π that is also a shortest path from s to v in G.
- The edges in E_π are called **tree edges**. $|E_\pi| = |V_\pi| - 1$.