Graph Algorithms

Tracy Bu FoA 2024 Sem 1

What's on today

- Adjacency matrix and list
- BFS and DFS
-

Graphs

- You’ve see it already – Trees are technically graphs
- large number of practical problems we can model as graph problems
  - network design
  - flow design
  - planning
  - scheduling
  - route finding
- You will see it again in later subjects

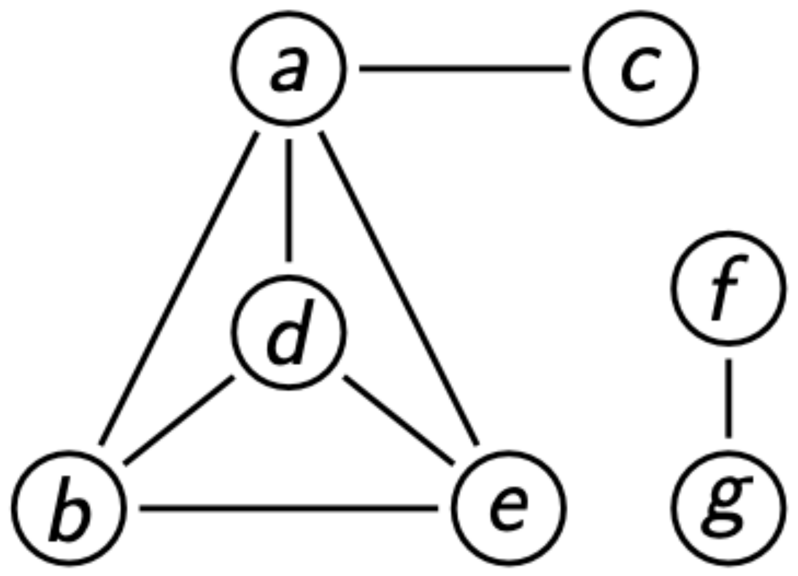
Graph, mathematically

Jargon warning

- Connected vs unconnected
- Directed vs undirected
- Node/vertex, edges, degrees
- Path(length)/Simple path/Cy cle
- Cyclic/Acyclic/DAG
- Dense/Sparse

Adjacency Matrix and List

Which one to use?



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

The adjacency matrix for the graph.

$a \to b \to c \to d \to e$
$b \to a \to d \to e$
$c \to a$
$d \to a \to b \to e$
$e \to a \to b \to d$
$f \to g$
$g \to f$

The adjacency list representation.

(Assuming lists are kept in sorted order.)

Searching, on Graphs

Or, Graph traversal

- Exhaustive search
- Mark as we go
- Breadth-first search (BFS)
  - exploring all neighbouring nodes before we go deeper
- Depth-first search (DFS)
  - Go all the way until we hit a dead end

DFS

- Based on back tracking
- Stack
- Time complexity?
  - Adj list vs matrix?

```
function DFS(⟨V, E⟩)
    mark each node in V with 0
    count ← 0
    for each v in V do
        if v is marked 0 then
            DfsExplore(v)

function DfsExplore(v)
    count ← count + 1
    mark v with count
    for each edge (v, w) do                    ▷ w is v's neighbour
        if w is marked with 0 then
            DfsExplore(w)
```

This works both for directed and undirected graphs.

DFS

- Time complexity?
  - Using an adjacency matrix, we need to consider $adj[v, w]$ for each w in V.
  - Here the complexity of graph traversal is $\Theta(|V|^2)$.
  - Using adjacency lists, for each v, we traverse the list $adj[v]$.
  - In this case, the complexity of traversal is $\Theta(|V| + |E|)$.

function $\text{DFS}(\langle V, E \rangle)$
  mark each node in V with 0
  $count \leftarrow 0$
  for each v in V **do**
    if v is marked 0 **then**
      $\text{DfsExplore}(v)$

function $\text{DfsExplore}(v)$
  $count \leftarrow count + 1$
  mark v with $count$
  for each edge (v, w) **do** $\triangleright$ w is v's neighbour
    if w is marked with 0 **then**
      $\text{DfsExplore}(w)$

This works both for directed and undirected graphs.

DFS

Applications

- Connected
- Cyclic

BFS

- Queue
- No backtrack needed
- Same complexity as DFS

function $\mathrm{BFS}(\langle V, E\rangle)$
  mark each node in V with 0
  $count \leftarrow 0$, $init$(queue) ▷ create an empty queue
  for each v in V **do**
    if v is marked 0 **then**
      $count \leftarrow count + 1$
      mark v with $count$
      $inject(queue, v)$ ▷ queue containing just v
      while $queue$ is non-empty **do**
        $u \leftarrow eject(queue)$ ▷ dequeues u
        for each edge (u, w) adjacent to u **do**
          if w is marked with 0 **then**
            $count \leftarrow count + 1$
            mark w with $count$
            $inject(queue, w)$ ▷ enqueues w

Topology sort

MORE SORTING!!!

- We use DFS:
  - Perform DFS and note the order in which nodes are popped off the stack.
  - List the nodes in the reverse of that order
- Why is topology sort useful?
  - Dependency Resolution
  - Task Scheduling
  - Course Prerequisites
  - Build Systems

Graph, Greedy and DP

It's all coming back together :))

- Shortest path problem (TSP), again
- Assume graph is connected
- Now think how you can implement it
- Time complexity?

Floyd’s algorithm

```
// Let dist be a 2D array of size V x V (where V is the number of vertices in the graph)
// Initialize the dist matrix with the given weights of the edges
// If there is no edge between vertex i and j, set dist[i][j] to infinity
// Set dist[i][i] to 0 for all vertices i

for k from 0 to V-1 do // Iterates over each vertex as an intermediate vertex.
    for i from 0 to V-1 do // Iterates over each starting vertex.
        for j from 0 to V-1 do // Iterates over each ending vertex.
            if dist[i][j] > dist[i][k] + dist[k][j] then
                dist[i][j] = dist[i][k] + dist[k][j]

// The dist array now contains the shortest paths between all pairs of vertices
```

- O(V^3) time
- DP: using the shortest paths between intermediate vertices to update the shortest paths between all pairs of vertices.

Dijkstra

Dike-struh

function $\textsc{Dijkstra}(\langle V, E\rangle, v_0)$
    for each $v \in V$ **do**
        $\mathit{dist}[v] \leftarrow \infty$
        $\mathit{prev}[v] \leftarrow \mathit{nil}$
    $\mathit{dist}[v_0] \leftarrow 0$
    $Q \leftarrow \textsc{InitPriorityQueue}(V)$ ▷ priorities are distances
    while Q is non-empty **do**
        $u \leftarrow \textsc{EjectMin}(Q)$
        for each $(u, w) \in E$ **do**
            if u in Q and $\mathit{dist}[u] + \mathit{weight}(u, w) < \mathit{dist}[w]$ **then**
                $\mathit{dist}[w] \leftarrow \mathit{dist}[u] + \mathit{weight}(u, w)$
                $\mathit{prev}[w] \leftarrow u$
                $\textsc{Update}(Q, w, \mathit{dist}[w])$ ▷ rearranges priority queue

Summary

- Graphs are super useful
- A lot more cool algorithms – Algorithms and Complexity (COMP90038)
- A lot of resource on line
- More on the proof heavy side – Graph Theory (MAST30011)
- If you are really really really keen on math heavy graph related topics – Group Theory and Linear Algebra (MAST20022)