LECTURE #13

Final Exam: Regular class time (6pm - 9pm)

SSSP:

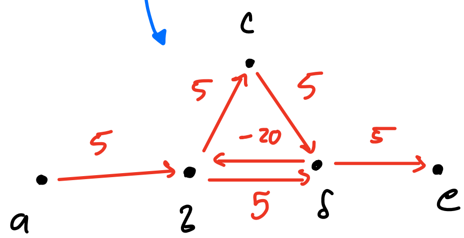
Dijkstra's : ← $O(E\log(V))$

- No negative weight edges "A weighted version of BFS"
- Uses Priority Queue. Regular binary heap

Bellman-Ford : ← $O(V^3)$

- Allows negative weight edges
- Detects negative weight cycles

Negative weight cycle:
∴ No shortest path from a to e



DIFFERENCE: HOW THEY CALL RELAX. Dijk = Strategy

Bellman = Brute Force

Common SSSP Operations (Used in both Dijkstras & Bellmans)

Initialization (G, S) { ← graph, start/source

δ = distance

p = parent/predecessor

for each vertex $v \in V$ {

// flag value (make it a pointer to int or pointer to null

```
        δ[v] = ∞          // ...  ...(infinity))
        p[v] = ∅          // null/no parent
    }
    δ[s] = 0         // distance to start node is zero
}


// called on an edge...
// used to improve best known path to a node...
Relax (u, v) {
    if (δ[v] > δ[u] + weight(u, v)) {
        δ[v] = δ[u] + weight(u, v);
        p[v] = u;
    }
}

Dijkstra's (G, s) {
    Initialization(G, s);
    for each vertex u ∈ V {
        PQ.insert(u)        // Min PQ on δ[u]
    }
    while (PQ.notEmpty()) {
        u = PQ.extractMin();
        for each vertex v ∈ Adj[u] {
            Relax(u, v);
        }
    }
}
```

Bellman-Ford (G, s) {

```
Bellman-Ford (G, s) {
    Initialization (G, s);
    for (int i = 0; i < |V|-1; i++) {      // cardinality of V
        for each edge (u, v) ∈ E {
            Relax (u, v);
        }
    }

    for each edge (u, v) ∈ E {
        if (δ[v] > δ[u] + weight (u, v)) {
            return ERROR/FALSE;
        }
    }

    return TRUE;
}
```

APSP

Floyd-Warshall: $O(V^3)$

- Dynamic Programming:

- optimal substructure
- works w/ overlapping subproblems
  "pieces of solution re used"

- Memoization
- works by considering intermediate nodes by the name or ID of the node!!!

Fundamental observation that explains how this works...

Consider nodes $1 \ldots k$ (from nodes $1 \ldots n$, where $k \le n$)

Look at a shortest path from node i to node j...

Q: Is node k on this path???

Case 1: No, k is not on the path. Then the shortest path from i to j, considering nodes $1 \ldots k$ is the same as the shortest path from i to j considering nodes $1 \ldots k-1$

Case 2: Yes, k is on the path. Then the shortest path from $i \to j$, $1 \ldots k$, is the sum of sp $i \to k$, $1 \ldots k-1$ plus sp $k \cdot j$, $1 \ldots k-1$

Base Case:

- $k = 0$
- No intermediate nodes allowed
- Basically just the adjacency list...
- Adjacent nodes only (not reachable)

Simple Idea:

"3d array" for distance

"3d array" for parent / predecessor

Min requirement for each:

- one 2D array for each

Compromise

- two 2D arrays each
  - current
  - previous

Calculate Distances Only:

$\delta_{ij}^{(k)}$ = total weight for shortest (known) path from i to j with all intermediate nodes in the range $1 \ldots k$

$K = 0$ base case

$\delta_{ij}^{(0)}$ = weight(i,j) weight of edge (i, j) or $\boxed{INF}$ if no edge

$$\delta_{ij}^{(k)} = \min\left(\delta_{ij}^{(k-1)},\ \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)}\right)$$

If we need to also remember the paths, then we need a separate array...

Π $\qquad$ P

base case:

No edge (i,j) exists...

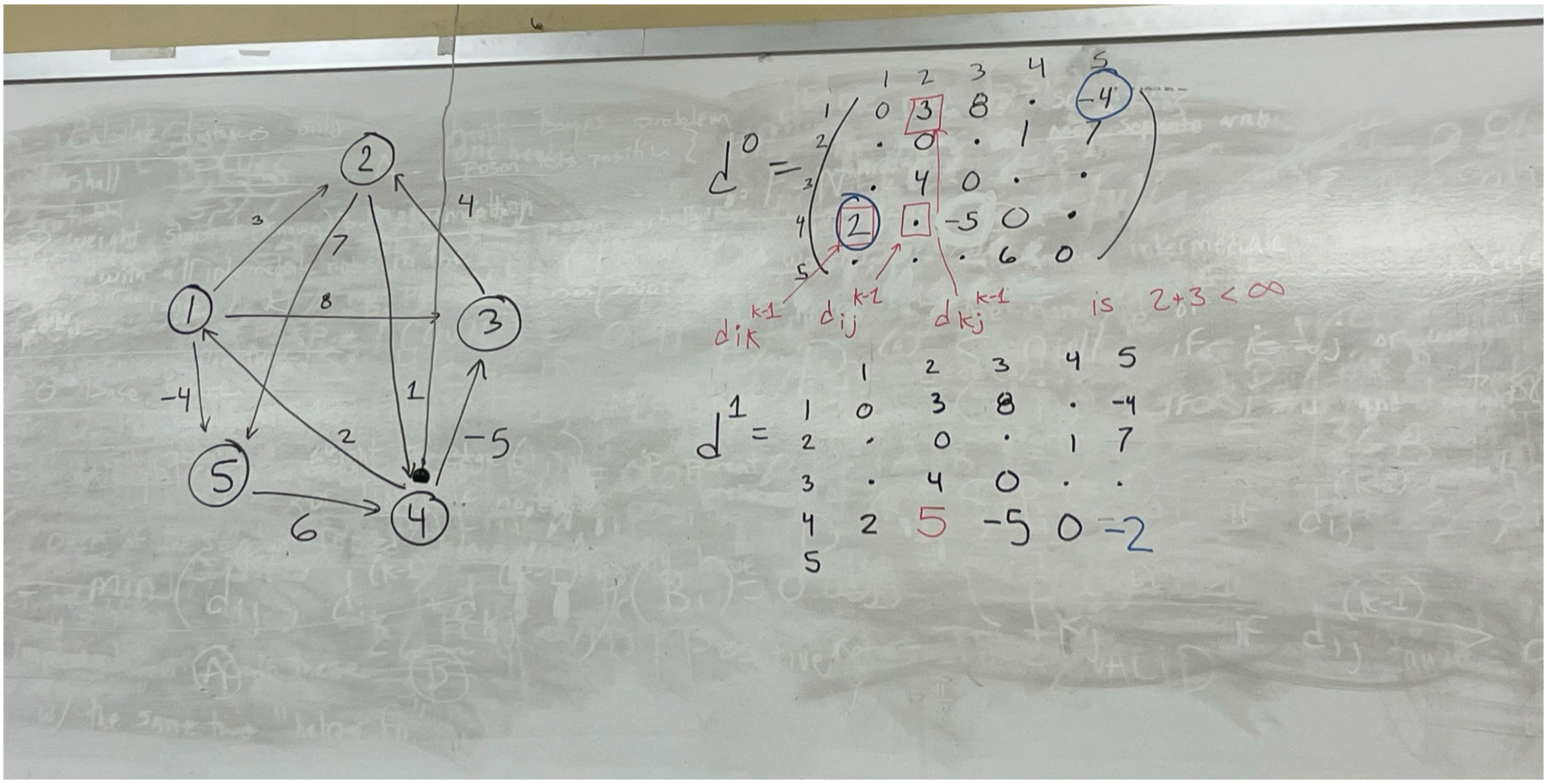
$$P_{ij}^{(0)} = \begin{cases} \text{null} & \text{if } i == j \text{ or } \underbrace{\text{weight } (i,j) == \infty} \\ i & \text{if } i \neq j \ \& \text{ weight } (i,j) < \infty \end{cases}$$

$$P_{ij}^{(k)} = \begin{cases} P_{ij}^{(k-1)} & \text{if } \delta_{ij}^{(k-1)} \leq \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)} \quad \text{A} \\ P_{kj}^{(k-1)} & \text{if } \delta_{ij}^{(k-1)} > \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)} \quad \text{B} \end{cases}$$

```
Floyd-Warshall(G) {
    n = |V|
    δ^0 = G.adjacencies          // for k = 0
    for k = 1 to n
        for i = 1 to n
            for j = 1 to n
                δ_ij^(k) = Min(δ_ij^(k-1), δ_ik^(k-1) + δ_kj^(k-1))

    return δ^n
}
```

5



FINAL EXAM INFO!

- Know $O(n)$ of everything... showing base performance is one. show best & worst O's...

- Could be a queue, linked list, stack, etc...
  coding question...

- BFS or DFS