Dijkstra's Algorithm — Coding Interview Notes (Light Theme)

General Pattern Template

```
from math import inf
from heapq import *

distances = [inf] * n
distances[source] = 0
heap = [(0, source)]

while heap:
    curr_dist, node = heappop(heap)
    if curr_dist > distances[node]:
        continue

    for nei, weight in graph[node]:
        dist = curr_dist + weight
        if dist < distances[nei]:
            distances[nei] = dist
            heappush(heap, (dist, nei))
```

Concept:
Dijkstra's Algorithm computes the shortest path from a single source node to all other nodes in a weighted graph (with non-negative edge weights). It uses a **min-heap (priority queue)** to always process the next closest node efficiently.

When to use: Weighted graphs (non-negative weights) where shortest paths from one source to all others are needed.

Time Complexity: O((V + E) log V) **Space Complexity:** O(V + E)

Key Ideas

1. Initialize all distances to infinity, except the source node (0).
2. Use a min-heap to repeatedly extract the node with the smallest distance.
3. Relax all outgoing edges: if a shorter path is found, update and push to heap.
4. Skip nodes already processed with a better distance (lazy deletion).
5. Stops naturally once all reachable nodes are finalized (heap empty).

Example 1: Dijkstra on a Weighted Graph (Adjacency List)

Goal: Compute shortest paths from a source in a small weighted graph.
Approach: Use adjacency list representation and priority queue.

```
from heapq import *
from math import inf

def dijkstra(n, graph, source):
    distances = [inf] * n
    distances[source] = 0
    heap = [(0, source)]

    while heap:
        curr_dist, node = heappop(heap)
        if curr_dist > distances[node]:
            continue
        for nei, weight in graph[node]:
            dist = curr_dist + weight
            if dist < distances[nei]:
                distances[nei] = dist
                heappush(heap, (dist, nei))
    return distances

# Example
graph = {
    0: [(1, 4), (2, 1)],
    1: [(3, 1)],
    2: [(1, 2), (3, 5)],
    3: []
}
print(dijkstra(4, graph, 0))  # Output: [0, 3, 1, 4]
```

Example 2: Reconstruct the Shortest Path

Goal: Return not only the shortest distances but also the actual shortest path from source to target.
Approach: Maintain a parent array to reconstruct the path.

```
def dijkstra_with_path(n, graph, source, target):
    from heapq import *
    from math import inf

    distances = [inf] * n
    parent = [-1] * n
    distances[source] = 0
    heap = [(0, source)]

    while heap:
        curr_dist, node = heappop(heap)
        if curr_dist > distances[node]:
            continue
        for nei, weight in graph[node]:
            dist = curr_dist + weight
            if dist < distances[nei]:
```

```
                distances[nei] = dist
                parent[nei] = node
                heappush(heap, (dist, nei))

    # reconstruct path
    path = []
    if distances[target] < inf:
        curr = target
        while curr != -1:
            path.append(curr)
            curr = parent[curr]
        path.reverse()
    return distances[target], path

# Example
graph = {
    0: [(1, 4), (2, 1)],
    1: [(3, 1)],
    2: [(1, 2), (3, 5)],
    3: []
}
print(dijkstra_with_path(4, graph, 0, 3))  # Output: (4, [0, 2, 1, 3])
```

Example 3: Dijkstra on a Weighted Grid

Goal: Find minimum cost path in a 2D grid where each cell has a cost.
Approach: Treat each cell as a graph node; move up/down/left/right if valid.

```
from heapq import *
from math import inf

def dijkstra_grid(grid):
    m, n = len(grid), len(grid[0])
    dist = [[inf]*n for _ in range(m)]
    dist[0][0] = grid[0][0]
    heap = [(grid[0][0], 0, 0)]
    dirs = [(1,0), (-1,0), (0,1), (0,-1)]

    while heap:
        d, r, c = heappop(heap)
        if d > dist[r][c]:
            continue
        for dr, dc in dirs:
            nr, nc = r+dr, c+dc
            if 0 <= nr < m and 0 <= nc < n:
                nd = d + grid[nr][nc]
                if nd < dist[nr][nc]:
                    dist[nr][nc] = nd
                    heappush(heap, (nd, nr, nc))
    return dist[m-1][n-1]

# Example
```

```
grid = [[1,3,1],[1,5,1],[4,2,1]]
print(dijkstra_grid(grid))  # Output: 7
```

Summary Table

ProblemGraph TypeKey Data StructureGoalComplexity Basic shortest pathWeighted (non-negative)Min-heapAll-pairs distance from sourceO((V+E)logV) Path reconstructionWeighted directed/undirectedMin-heap + parent arrayShortest path traceO((V+E)logV) Grid shortest pathWeighted grid2D heap-based searchMin cost cell-to-cellO(MN log(MN))