Problem Statement

There are n cities connected by a number of flights. Each flight is represented as an array flights where flights[i] = [from_i, to_i, price_i] indicates a flight from city from_i to city to_i with a cost of price_i. Given three integers src, dst, and k, the goal is to find the cheapest price from src to dst with at most k stops. If there is no such route, return -1.

Input: n = 3, flights = [[0, 1, 100], [1, 2, 100], [0, 2, 500]], src = 0, dst = 2, k = 1

Output: 200

Algorithm

To tackle this problem, we can employ a modified version of the Dijkstra's algorithm which is typically used for finding the shortest path in terms of cost. Here, we need to consider the additional constraint of the number of stops.

Approach

1. **Graph Representation**:
   - Represent the flights using an adjacency list where each city points to its neighboring cities along with the cost of the flight.
2. **Priority Queue**:
   - Use a priority queue to explore paths in the order of increasing cost. This ensures that once the destination city is reached, it is done with the minimum possible cost.
3. **Algorithm Steps**:
   - Initialize the priority queue with the source city, a cost of 0 (starting cost), and -1 stops (initially).
   - Dequeue from the priority queue and explore each city along with its current accumulated cost and stops taken.
   - If the dequeued city is the destination city, return the accumulated cost as the cheapest price found.
   - If the number of stops taken is less than k, explore all neighboring cities:
     - Calculate the cost to reach each neighboring city by adding the cost of the flight to the current accumulated cost.
     - Enqueue these neighbors with updated costs and increment the number of stops by 1.
   - Continue this process until either the priority queue is empty (indicating no more paths to explore) or the destination city is reached.
4. **Termination**:
   - If no path is found within k stops, return -1 indicating that no valid route exists.

Solution Code

Here is the Python implementation of the described approach:

```
from collections import defaultdict
import heapq

class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, k: int) -> int:
        # Step 1: Create a graph from the flights data
        graph = defaultdict(list)
        for u, v, price in flights:
            graph[u].append((v, price))

        # Step 2: Initialize the priority queue and the min_cost dictionary
        pq = [(0, src, 0)]  # (cost, current_city, stops)
        min_cost = defaultdict(lambda: float('inf'))
        min_cost[(src, 0)] = 0

        # Step 3: Process the queue
        while pq:
            cost, current_city, stops = heapq.heappop(pq)

            if current_city == dst:
                return cost

            if stops < k:
                for neighbor, price in graph[current_city]:
                    new_cost = cost + price
                    if new_cost < min_cost[(neighbor, stops + 1)]:
                        min_cost[(neighbor, stops + 1)] = new_cost
                        heapq.heappush(pq, (new_cost, neighbor, stops + 1))

        return -1
```

Alternative Approach: Bellman-Ford Algorithm

1. **Algorithm Steps**:
    - Initialize an array dist with size n to store the minimum cost to reach each city from src. Set dist[src] to 0 and all other entries to ∞.
    - Relax all edges $n-1$ times:
        - For each flight [u, v, w] in flights, if dist[u] + w < dist[v], update dist[v] to dist[u] + w.
    - After $n-1$ iterations, dist[dst] contains the minimum cost to reach dst from src with at most k stops or -1 if no such path exists.

2. **Source Code**:

```
class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, k: int) -> int:
        # Step 1: Initialize distances array with infinity and set source distance to 0
        inf = float('inf')
        dist = [inf] * n
        dist[src] = 0

        # Step 2: Relax edges for k + 1 times
        for _ in range(k + 1):
            # Create a copy of dist array for the current iteration
            current_dist = dist[:]

            # Relax all edges (u, v, w)
            for u, v, w in flights:
                if dist[u] != inf and dist[u] + w < current_dist[v]:
                    current_dist[v] = dist[u] + w

            # Update dist array for the next iteration
            dist = current_dist

        # Step 3: Return the shortest distance to dst or -1 if not reachable
        return dist[dst] if dist[dst] != inf else -1
```

Both approaches provide a method to solve the problem of finding the cheapest flight with at most k stops, leveraging different algorithms suited for the task.