(a)

Subproblems

Define $D(u, k)$ to be the minimum total weight of any path from the source s to u that uses at most k edges. Obviously, the final answer is the $\min_{u \in V} D(u, K) + p(u)$.

Recursive Formula

For base case, we have $D(s, 0) = 0$ as it uses 0 edge to reach s itself, and $D(u, 0) = \infty$ for $u \neq s$ as there are no way to reach $u \neq s$ using 0 edge.

For recurrence case, we have

$$D(u, k) = \min\{D(u, k-1), \min_{(v \to u) \in E} D(v, k-1) + w(v \to u)\}$$

It should be obvious that we could take an edge from all possible edge that go to vertex u, and the cost is the weight from s to v plus the weight of $v \to u$, the new cost is the minimum of all possibility.

Evaluation Order

Notice that $D(u, k)$ depends on $D(v, k-1)$, so we need to evaluate $k \in [1, K]$ in increasing order for every $u \in V$ (the order for which we evaluate vertices doesn't matter). Notice we need store a $P(u, k)$ that records the parent of the vertex which gives the minimum $D(u, k)$.

Algorithms

Optimal Value

Declare 2D arrays $D[u, k]$ and $P[u, k]$ for which $u \in V$ and $k \in [1, K]$. Initialize $D[s, 0] = 0$ and $D[u, *] = \infty$ for $u \neq s$.

For $k : 1 \to K$

- For each $u \in V$
  - $D[u, k] = D[u, k-1]$
  - $P[u, k] = P[u, k-1]$
- For each $(u \to v) \in E$
  - If $D[v, k-1] + w(v \to u) < D[v, k]$
    - $D[u, k] = D[v, k-1] + w(v \to u)$
    - $P[u, k] = v$

Return $\min_{u \in V} D[u, K] + p(u)$ with $\tilde{v}$ that makes the minimum

Optimal Solution

Define $\mathsf{Reconstruct}(u, k)$:

- Let $\mathsf{Path} = []$
- While $k \geq 0$ or $u \neq s$:
    - Append u to Path
    - Set $u = P[u, k]$
    - Set $k = k - 1$
- Append s to Path
- Reverse Path
- Return Path

Return $\mathsf{Reconstruct}(\tilde{v}, K)$

Complexity

It should be obvious that we run for all vertex and for each $e \in E$, so the time complexity is $O(mK)$, and that we store $D[u, k]$ so the space used is $O(nK)$

(b)

We could divide this problem into two halves where the most edge used is $M \approx K/2$. Then we run a forward DP on first M edges using only $O(n)$ spaces (two rows, one previous layer and one current layers). This gives a cost vector F such that for every vertex u, F[u] is the best cost from s to u in M steps. Run a similar DP on the *reversed graph* to compute for each vertex u the minimum cost B[u] to get from u to some destination v *plus* its penalty p(v). For every vertex u, F[u] + B[u] gives a candidate overall cost for a path that "splits" at u. Choose the vertex u that minimizes F[u] + B[u]. This vertex is the "midpoint" on an optimal path. Once the midpoint is determined, recursively reconstruct the first half (from s to u in M steps) and the second half (from u to the final vertex using K–M steps).

Analysis

- **Time:** At each level we do two DP's (forward and backward) costing O(mM) and O(m(K–M)) respectively, so about O(mK) work per level. With O(log K) levels we get O(mK log K) overall.
- **Space:** At any time we store only O(n) values in each DP array; with O(log K) recursion depth the space is O(n log K).