Problem Variations for Single Source Shortest Path Algorithms

Extended Practice Problems for CSE 208

December 2025

Introduction

This document contains various problem variations based on the two core algorithms:

- **Problem 1 Type:** Modified Dijkstra's Algorithm (with state tracking)
- **Problem 2 Type:** Bellman-Ford Algorithm (with negative cycle detection)

These variations extend the original problems by adding constraints, multiple states, or additional objectives while maintaining the core algorithmic approach.

Contents

1 Problem 1 Variations: Modified Dijkstra's Algorithm

1.1 Multiple Coupons

Problem 1.1: K Discount Coupons

Problem Statement:
You are given a directed graph with n nodes and m edges. You want to travel from node 1 to node n. You have k discount coupons ($k \leq 5$), where each coupon can be used to halve the cost of one edge (cost c becomes $\lfloor c/2 \rfloor$).
Find the minimum cost to reach from node 1 to node n.
Input:

- First line: n, m, k
- Next m lines: a, b, c (directed edge from a to b with cost c)

Output: Minimum cost from node 1 to node n.
Constraints:

- $2 \leq n \leq 10^5$
- $1 \leq m \leq 10^5$
- $1 \leq k \leq 5$
- $1 \leq c \leq 10^9$

Algorithm Hint: Use Dijkstra with state $(node, coupons\_used)$. Total states: $O(n \times k)$.

1.2 Different Discount Types

Problem 1.2: Multiple Discount Types

Problem Statement:
You have three types of discount coupons:

- Type A: Reduces cost to $\lfloor c/2 \rfloor$ (you have k_1 of these)
- Type B: Reduces cost by fixed amount D (you have k_2 of these)
- Type C: Reduces cost to $\lfloor c/3 \rfloor$ (you have k_3 of these)

Find the minimum cost from node 1 to node n using optimal coupon allocation.
Input:

- First line: n, m, k_1, k_2, k_3, D
- Next m lines: a, b, c

Output: Minimum cost.
Algorithm Hint: State: $(node, coupons\_A\_used, coupons\_B\_used, coupons\_C\_used)$.

1.3 Conditional Coupon Usage

Problem 1.3: Threshold-Based Discount

Problem Statement:
You have one discount coupon that can only be used on edges with cost $\geq T$. When used, it reduces the cost to $\lfloor c/2 \rfloor$.
Find the minimum cost from node 1 to node n.
Input:

- First line: n, m, T
- Next m lines: a, b, c

Output: Minimum cost.
Variation: Coupon can only be used in first k edges of the path, or only on edges within a specific subset of nodes.

1.4 Airport-Based Discounts

Problem 1.4: Special Airport Discounts

Problem Statement:
Certain airports (given in set S) offer a special deal: if you arrive at one of these airports, your next outgoing flight gets a 50% discount.
Find the minimum cost from node 1 to node n.
Input:

- First line: n, m, $|S|$
- Second line: $|S|$ space-separated integers (special airports)
- Next m lines: a, b, c

Output: Minimum cost.
Algorithm Hint: State: $(node, has\_discount\_for\_next\_flight)$.

1.5 Fuel Constraints

Problem 1.5: Minimum Cost with Fuel Limit

Problem Statement:
You start with F units of fuel. Each edge (a, b, c) costs c dollars and consumes f_{ab} units of fuel. Certain airports allow refueling (add R units of fuel for free).
Find the minimum cost to reach node n from node 1 without running out of fuel.
Input:

- First line: n, m, F, R
- Second line: $|S|$ followed by $|S|$ refueling stations
- Next m lines: a, b, c, f (cost and fuel consumption)

Output: Minimum cost, or -1 if impossible.
Algorithm Hint: State: $(node, remaining\_fuel)$. Discretize fuel into reasonable buckets.

1.6 Time-Cost Trade-off

Problem 1.6: Minimum Cost with Time Constraint

Problem Statement:
Each edge has both cost c and time t. You must reach node n from node 1 within time limit T.
Find the minimum cost subject to the time constraint.
Input:

- First line: n, m, T
- Next m lines: a, b, c, t (cost and time)

Output: Minimum cost, or -1 if impossible to reach within time T.
Algorithm Hint: State: $(node, time\_used)$. Use Dijkstra prioritizing cost.

1.7 Mandatory Stops

Problem 1.7: Path with Required Visits

Problem Statement:
You must visit at least one airport from a given set S before reaching node n.
Find the minimum cost from node 1 to node n with this constraint.
Input:

- First line: n, m, $|S|$
- Second line: $|S|$ required airports
- Next m lines: a, b, c

Output: Minimum cost.
Variation: Must visit exactly k specific nodes, or must visit nodes in specific order.
Algorithm Hint: State: $(node, visited\_any\_required)$ or use bitmask for multiple required nodes.

1.8 Two-Currency System

Problem 1.8: Dual Currency Travel

Problem Statement:
Each edge costs (c_1, c_2) units of currency A and B respectively. You start with (M_1, M_2) units. Some airports allow currency exchange at rate r (1 unit of A = r units of B). Find if you can reach node n from node 1, and if yes, what's the lexicographically smallest remaining currency pair.
Input:

- First line: n, m, M_1, M_2, r
- Second line: $|E|$ exchange airports
- Next m lines: a, b, c_1, c_2

Output: YES/NO, and if YES, remaining (A, B).
Algorithm Hint: State: $(node, currency\_A, currency\_B, can\_exchange)$.

1.9 Minimum Stops Problem

Problem 1.9: Minimum Cost with Hop Limit

Problem Statement:
Find the minimum cost from node 1 to node n using at most K edges.
Input:

- First line: n, m, K
- Next m lines: a, b, c

Output: Minimum cost with at most K hops, or -1 if impossible.
Algorithm Hint: State: $(node, edges\_used)$. This is actually a variant that can be solved with modified Dijkstra or BFS-based approaches.

1.10 Path with Maximum Edge Weight Limit

Problem 1.10: Minimax Path Problem

Problem Statement:
Find a path from node 1 to node n such that the maximum edge weight in the path is minimized. If there are multiple such paths, choose the one with minimum total cost.
Input:

- First line: n, m
- Next m lines: a, b, c

Output: Two integers: (1) minimum possible maximum edge weight, (2) minimum total cost among all paths achieving this maximum edge weight.
Algorithm Hint: Use modified Dijkstra prioritizing maximum edge first, then total cost.

2 Problem 2 Variations: Bellman-Ford & Negative Cycles

2.1 Shortest Negative Cycle

Problem 2.1: Most Negative Cycle

Problem Statement:
Given a directed graph, find the negative cycle with the minimum sum (most negative sum).
Input:

- First line: n, m
- Next m lines: a, b, c

Output:

- If no negative cycle exists: -1
- Otherwise: First line contains the sum of the most negative cycle, second line contains the cycle vertices in order

Algorithm Hint: Run Bellman-Ford from all nodes, track minimum cycle sum.

2.2 Count Negative Cycles

Problem 2.2: Count All Negative Cycles

Problem Statement:
Count the total number of distinct negative cycles in the graph. Two cycles are considered the same if they contain the same set of edges.
Input:

- First line: n, m
- Next m lines: a, b, c

Output: Number of distinct negative cycles.
Constraints: $n \leq 20$ (since this is a hard problem)
Algorithm Hint: Use DFS/backtracking with cycle hashing to avoid duplicates.

2.3 Remove Minimum Edges

Problem 2.3: Eliminate Negative Cycles

Problem Statement:
Find the minimum number of edges to remove from the graph such that no negative cycle remains.
Input:

- First line: n, m
- Next m lines: a, b, c

Output: Minimum number of edges to remove.
Bonus: Print which edges to remove.
Algorithm Hint: This is NP-hard in general, but can be solved with backtracking for small graphs or greedy heuristics.

2.4 K-th Shortest Path

Problem 2.4: K-th Shortest Path Detection

Problem Statement:
Find the k-th shortest path from node s to node t. If the graph contains negative cycles reachable from s, print "NEGATIVE CYCLE". If fewer than k paths exist, print "IMPOSSIBLE".
Input:

- First line: n, m, s, t, k
- Next m lines: a, b, c

Output: Cost of k-th shortest path, or "NEGATIVE CYCLE", or "IMPOSSIBLE".
Algorithm Hint: Use Yen's algorithm or modified Dijkstra with priority queue of paths, but first check for negative cycles.

2.5 Negative Cycle with Constraints

Problem 2.5: Negative Cycle Through Node

Problem Statement:
Given a node v, determine if there exists a negative cycle that passes through node v. If yes, print the cycle.
Input:

- First line: n, m, v
- Next m lines: a, b, c

Output:

- -1 if no such cycle exists
- Otherwise, print the cycle vertices including v

Algorithm Hint: Run Bellman-Ford, then check if v is reachable from any node in a negative cycle and vice versa.

2.6 Shortest Path After K Relaxations

Problem 2.6: Distance After K Iterations

Problem Statement:
Find the shortest path distance from node 1 to all other nodes after exactly k iterations of edge relaxation (as in Bellman-Ford).
Input:

- First line: n, m, k
- Next m lines: a, b, c

Output: For each node i from 1 to n, print the distance after k relaxations (or "INF" if unreachable).
Algorithm Hint: Standard Bellman-Ford but stop after exactly k iterations.

2.7 Conditional Edge Activation

Problem 2.7: Dynamic Edge Activation

Problem Statement:
Some edges are initially inactive. An edge (u, v) activates only after you visit a specific "activation node" a_e. Determine if activating edges in some order can create a negative cycle.
Input:

- First line: n, m
- Next m lines: a, b, c, act (edge from a to b with cost c, activates after visiting node act; if $act = 0$, edge is always active)

Output: YES if a negative cycle can be created, NO otherwise. If YES, print the sequence of nodes to visit and the resulting cycle.
Algorithm Hint: State-space search with state $(node, activated\_edges\_bitmask)$.

2.8 Safe Path Detection

Problem 2.8: Shortest Path Avoiding Negative Cycles

Problem Statement:
Find the shortest path from s to t that doesn't pass through any negative cycle. If all paths pass through negative cycles, print "UNSAFE".
Input:

- First line: n, m, s, t
- Next m lines: a, b, c

Output: Shortest safe path cost, or "UNSAFE".
Algorithm Hint: Find all nodes involved in or reachable from negative cycles, then run Dijkstra on the subgraph excluding these nodes.

2.9 Multi-Source Negative Cycle Detection

Problem 2.9: Negative Cycles from Multiple Sources

Problem Statement:
Given k source nodes, determine which sources can reach a negative cycle. For each such source, print one negative cycle reachable from it.
Input:

- First line: n, m, k
- Second line: k source nodes
- Next m lines: a, b, c

Output:

- First line: number of sources that can reach negative cycles
- For each such source: source node, followed by a negative cycle reachable from it

Algorithm Hint: Run Bellman-Ford from each source, track reachability to negative cycle nodes.

2.10 Minimum Cost to Break Cycles

Problem 2.10: Cheapest Edge Removal

Problem Statement:
Each edge has both a weight w (for cycle detection) and a removal cost r. Find the minimum total removal cost to eliminate all negative cycles.
Input:

- First line: n, m
- Next m lines: a, b, w, r (weight and removal cost)

Output: Minimum total removal cost, and which edges to remove.
Algorithm Hint: Iteratively find negative cycles and greedily remove the edge with minimum removal cost that breaks the cycle.

3 Combined & Advanced Variations

3.1 Shortest Path with Edge Removal

Problem 3.1: Remove Edges to Minimize Path

Problem Statement:
You can remove at most k edges from the graph. Find the minimum cost path from node 1 to node n after optimal edge removal. Ensure no negative cycles exist after removal.
Input:

- First line: n, m, k
- Next m lines: a, b, c

Output: Minimum path cost after removing at most k edges, or "IMPOSSIBLE" if negative cycles cannot be avoided.
Algorithm Hint: Try all $\binom{m}{k}$ combinations for small k, or use heuristics.

3.2 Dynamic Graph Shortest Path

Problem 3.2: Shortest Path with Updates

Problem Statement:
Initially, you're given a graph. Answer q queries where each query either:

- Type 1: Add edge (a, b, c)
- Type 2: Remove edge (a, b)
- Type 3: Report shortest path from s to t, or "NEGATIVE CYCLE" if one exists

Input:

- First line: n, m, q
- Next m lines: initial edges
- Next q lines: queries

Output: Answer for each Type 3 query.
Algorithm Hint: After each update, re-run Bellman-Ford or use dynamic algorithms.

3.3 Layered Graph Shortest Path

Problem 3.3: Multi-Layer Graph

Problem Statement:
The graph has L layers $(0, 1, \dots, L-1)$. Regular edges exist within each layer. Special "teleport" edges allow moving between layers with different costs. Start at node 1 in layer 0, reach node n in layer $L-1$.
Input:

- First line: n, m, L, t (nodes, edges per layer, layers, teleport edges)
- Next $m \times L$ lines: edges within each layer
- Next t lines: teleport edges $(a, layer_1, b, layer_2, cost)$

Output: Minimum cost to reach node n at layer $L-1$ from node 1 at layer 0.
Algorithm Hint: Construct a graph with $n \times L$ nodes and run Dijkstra.

3.4 Arbitrage Detection

Problem 3.4: Currency Arbitrage

Problem Statement:
Given n currencies and exchange rates between them, determine if there's an arbitrage opportunity (a cycle where you end up with more money than you started with).
Input:

- First line: n, m (currencies and exchange pairs)
- Next m lines: a, b, $rate$ (1 unit of currency $a = rate$ units of currency b)

Output: YES if arbitrage exists (print the cycle), NO otherwise.
Algorithm Hint: Convert to negative cycle detection: use $-\log(rate)$ as edge weights.

3.5 Path Counting with Constraints

Problem 3.5: Count Paths Below Cost Threshold

Problem Statement:
Count the number of distinct paths from node 1 to node n with total cost $\leq C$.
Input:

- First line: n, m, C
- Next m lines: a, b, c

Output: Number of paths with cost $\leq C$ modulo $10^9 + 7$.
Constraints: $n \leq 15$ (for DP with bitmask)
Algorithm Hint: DP with state $(current\_node, visited\_bitmask, cost)$.

4 Exam Preparation Tips

4.1 Key Concepts to Master

1. **State Representation:** Most variations involve tracking additional state beyond just the current node. Practice identifying what state needs to be tracked.
2. **Modified Dijkstra:**
   - State: $(node, additional\_info)$
   - Complexity: $O((V \times S)\log(V \times S) + E \times S)$ where S is the number of possible states per node
   - Use priority queue with state tuple
3. **Bellman-Ford Cycle Detection:**
   - Run $n - 1$ relaxations
   - On n-th iteration, any node that relaxes is in or reachable from a negative cycle
   - To find the cycle: trace back using parent pointers
4. **Graph Construction:**
   - Many problems require building a transformed graph
   - Example: for k coupons, create graph with $n \times (k + 1)$ nodes
   - Edges connect (u, i) to (v, i) with full cost and (u, i) to $(v, i + 1)$ with discounted cost
5. **Handling Multiple Constraints:**
   - Encode all constraints in the state
   - Use bit-masking for visiting subset of nodes
   - Be careful with state explosion (keep states $\leq 10^6$)

4.2 Common Pitfalls

- **Integer Overflow:** Use `long long` in C++ for costs up to 10^9
- **Infinite Loops:** In negative cycle detection, ensure proper termination
- **Cycle Printing:** Make sure to print cycles in correct order
- **State Space:** Don't create too many states (keep product of state dimensions reasonable)
- **Initialization:** Initialize distances correctly (INF for unreachable, 0 for source)

4.3 Time Complexity Analysis

Algorithm	**Complexity**
Dijkstra (basic)	$O((V + E)\log V)$
Dijkstra with S states	$O((V \cdot S + E \cdot S)\log(V \cdot S))$
Bellman-Ford	$O(V \cdot E)$
Bellman-Ford with cycle extraction	$O(V \cdot E + V)$

Table 1: Algorithm Complexities

4.4 Problem-Solving Strategy

1. **Identify the base algorithm:** Is it a shortest path problem (Dijkstra) or cycle detection (Bellman-Ford)?
2. **Determine the state:** What information do you need to track at each node?
3. **Check state space size:** Will your state representation fit in memory and time?
4. **Model transitions:** How do you move from one state to another?
5. **Handle edge cases:** Unreachable nodes, impossible scenarios, empty graphs
6. **Optimize if needed:** Use appropriate data structures (priority queue, adjacency list)
7. **Test with examples:** Verify your solution with the given test cases

4.5 Implementation Checklist

Before submitting, ensure:

- ☐ Correct input parsing (especially for multiple test cases)
- ☐ Proper initialization of distance arrays
- ☐ Handling of INF values and overflow
- ☐ Correct state transitions
- ☐ Output format matches exactly (spacing, newlines)
- ☐ Edge cases tested (single node, disconnected graph, etc.)

5 Sample Solutions Outline

5.1 Problem 1.1: Multiple Coupons Solution Sketch

```
State: (node, coupons_used)
Graph: Create nodes (u, k) for 0 <= k <= K

Algorithm:
1. Initialize dist[1][0] = 0, all others = INF
2. Use priority queue with (cost, node, coupons_used)
3. For each edge (u, v, c):
   - Try without coupon: relax(v, k) from (u, k)
   - Try with coupon: relax(v, k+1) from (u, k) if k < K
4. Answer: min(dist[n][k]) for all k
```

5.2 Problem 2.1: Most Negative Cycle Solution Sketch

```
Algorithm:
1. Run Bellman-Ford for n-1 iterations
2. On n-th iteration, mark all nodes that can be relaxed
3. For each marked node, trace back to find cycle
4. Track minimum sum cycle found
5. Return the cycle with most negative sum
```

6 Additional Practice Resources

6.1 Online Judges

- **Codeforces:** Search for "shortest path" and "negative cycle" problems
- **AtCoder:** Graph section has excellent shortest path problems
- **LeetCode:** "Cheapest Flights Within K Stops", "Network Delay Time"
- **CSES Problem Set:** Flight Discount, Flight Routes, Cycle Finding

6.2 Related Topics to Study

- Floyd-Warshall (all-pairs shortest paths)
- Johnson's Algorithm (negative edges with all-pairs)
- A* Search (heuristic-guided shortest path)
- Shortest Path Faster Algorithm (SPFA)
- Dynamic programming on graphs

7 Conclusion

These variations should give you comprehensive practice for your online exam. The key is to:

1. Recognize the base algorithm needed
2. Identify what additional state to track
3. Implement carefully with proper bounds checking
4. Test thoroughly with edge cases

Remember that most variations are just the standard algorithms with modified state spaces or additional constraints. Once you master the pattern of state expansion, you can tackle any variation thrown at you.

Good luck with your exam!

For questions or clarifications, contact your instructor.