Faire Coding Interview Preparation

Funnel Analysis Problem

Alex Yang

Prepared: November 23, 2024

Contents

1 Interview Overview

1.1 Problem Type

- **Category:** Data Processing / State Machine
- **Difficulty:** Medium
- **Source:** Confirmed Faire Interview Problem
- **Key Focus:** Algorithm correctness + comprehensive testing

Critical Point

Faire's Testing Emphasis:
Faire places exceptional emphasis on testing. Your solution must include:

- Comprehensive pytest test cases
- Edge case coverage (repeated steps, out-of-order events, drop-offs)
- Clear test descriptions
- High test coverage mindset

2 Problem Statement

2.1 Overview

Given multiple funnels (sequences of steps) and user events, determine how many distinct users reached each step of each funnel **IN ORDER**.

2.2 Input Format

Input 1 - Funnels (CSV format):

```
funnel_name,step_1,step_2,...,step_n
```

Examples:

- `checkout_process,view_product,add_to_cart,enter_payment_info,complete_order`
- `three_clicks_then_add,click_product,click_product,click_product,add_to_cart`

Key Insight

Funnels may contain **REPEATED** steps (same step name multiple times). This is a critical requirement that trips up many candidates.

Input 2 - User Events (CSV format):

```
user_id,timestamp,event_name
```

- `user_id`: integer user identifier

- `timestamp`: integer (guaranteed to be sorted chronologically)
- `event_name`: step name or irrelevant event

Output Format:

```
funnel_name,step_1(count_1),step_2(count_2),...,step_n(count_n)
```

2.3 Rules

1. Events are processed in timestamp order for each user
2. User advances to next step only when event matches current required step
3. Repeated steps require separate matching events
4. Each funnel is evaluated independently
5. Out-of-order events are ignored (don't advance the funnel)

2.4 Example

Funnels:

```
checkout_process,view_product,add_to_cart,enter_payment_info,complete_order
three_clicks_then_add,click_product,click_product,click_product,add_to_cart
```

Events:

```
4,1200,view_product       5,1250,view_product
4,1210,add_to_cart        5,1260,add_to_cart
4,1300,enter_payment_info 5,1270,enter_payment_info
                          5,1280,complete_order
6,2000,click_product      7,3000,click_product
6,2005,click_product      7,3005,click_product
6,2010,click_product
6,2015,add_to_cart
```

Output:

```
checkout_process,view_product(2),add_to_cart(2),enter_payment_info(2),complete_order(1)
three_clicks_then_add,click_product(2),click_product(2),click_product(1),add_to_cart(1)
```

Explanation:

- `checkout_process`: Users 4 and 5 both reach view_product, add_to_cart, enter_payment_info. Only user 5 completes the entire funnel.
- `three_clicks_then_add`: User 6 does 3 click_product events then add_to_cart (completes all 4 steps). User 7 only reaches the first 2 clicks.

2.5 Constraints

- $1 \leq \text{funnels} \leq 100$
- $1 \leq \text{events} \leq 10^5$
- Funnel steps may repeat
- Events are globally sorted by timestamp

3 Optimal Solution

3.1 Algorithm Overview

The optimal solution uses a **state machine approach** for each user in each funnel:

1. **Parse funnels**: Extract funnel name and list of steps
2. **Group events by user**: Create user → events mapping
3. **Track progress**: For each funnel, maintain sets of users who reached each step
4. **Process events**: For each user, maintain current step index and advance when matching event occurs
5. **Format output**: Convert step-user sets to counts

3.2 Key Insights

Key Insight

State Machine Pattern:

- Each user maintains their own position (state) in each funnel
- Users can only advance sequentially through funnel steps
- Repeated step names are handled naturally by step *index* tracking
- Using sets ensures each user is counted once per step

3.3 Common Mistakes to Avoid

Critical Point

Don't make these errors:

1. **Using step name as key**: Won't work for repeated steps like `click,click,click`
2. **Not using sets**: Will count users multiple times per step
3. **Incorrect state management**: Not maintaining separate state per user per funnel
4. **Off-by-one errors**: Starting index at 1 instead of 0
5. **Sorting per user**: Events are already globally sorted (don't re-sort)

3.4 Time & Space Complexity

- **Time**: $O(F \times U \times E_u)$ where:
  - F = number of funnels
  - U = number of users

  - E_u = average events per user

- **Space**: $O(F \times S \times U)$ where S = average steps per funnel
- **Optimization**: For given constraints (10^5 events), this is optimal

3.5 Python Implementation

```python
from typing import List
from collections import defaultdict


class Solution:
    def compute_funnel_counts(
        self,
        funnels: List[str],
        events: List[str]
    ) -> List[str]:
        """
        Compute, for each funnel, how many distinct users reach each step in order.

        :param funnels: List of strings, each formatted:
                        "funnel_name,step_1,step_2,...,step_n"
        :param events:  List of strings, each formatted:
                        "user_id,timestamp,event_name"
        :return: List of strings, one per funnel, formatted:
                 "funnel_name,step_1(count_1),step_2(count_2),...,step_n(count_n)"
        """
        # Parse funnels into (name, steps_list) tuples
        parsed_funnels = []
        for funnel_str in funnels:
            parts = funnel_str.split(',')
            funnel_name = parts[0]
            steps = parts[1:]  # May contain duplicate step names
            parsed_funnels.append((funnel_name, steps))

        # Parse and group events by user_id
        # user_id -> [(timestamp, event_name), ...]
        user_events = defaultdict(list)
        for event_str in events:
            parts = event_str.split(',')
            user_id = int(parts[0])
            timestamp = int(parts[1])
            event_name = parts[2]
            user_events[user_id].append((timestamp, event_name))

        # Events are already sorted globally by timestamp (guaranteed)

        # Process each funnel independently
        results = []
        for funnel_name, steps in parsed_funnels:
            # Track which users reached each step index
            # step_users[i] = set of user_ids who reached step i
            step_users = [set() for _ in range(len(steps))]

            # For each user, track their current position in this funnel
            for user_id, events_list in user_events.items():
                current_step_idx = 0  # User starts before step 0

                # Process user's events in timestamp order
                for timestamp, event_name in events_list:
                    # Check if this event matches the next required step
                    if current_step_idx < len(steps) and \
                       event_name == steps[current_step_idx]:
```

```
                        # User reached this step
                        step_users[current_step_idx].add(user_id)
                        current_step_idx += 1

            # Format output for this funnel
            output_parts = [funnel_name]
            for i, step_name in enumerate(steps):
                count = len(step_users[i])
                output_parts.append(f"{step_name}({count})")

            results.append(','.join(output_parts))

        return results
```

4 Comprehensive Test Suite

4.1 Testing Philosophy at Faire

Testing Focus

Why Faire emphasizes testing:

- E-commerce platform requires high reliability
- Bugs in analytics/funnel tracking can lead to incorrect business decisions
- Test coverage demonstrates engineering maturity
- Well-tested code is maintainable code

What to test:

1. **Happy path**: Basic functionality works
2. **Edge cases**: Empty states, single elements, boundaries
3. **Repeated steps**: The tricky part of this problem
4. **Out-of-order events**: Events that don't advance funnel
5. **Drop-offs**: Users leaving at various stages
6. **Multiple users**: Concurrent progress through funnels

4.2 Complete Test Implementation

```
import pytest
from typing import List


class TestFunnelAnalysis:
    """
    Comprehensive test suite for Funnel Analysis problem.

    Faire Interview Focus: Tests demonstrate understanding of:
    - Edge cases and boundary conditions
    - Repeated step handling
    - State machine correctness
    - Multiple user scenarios
    """

    def test_basic_example(self):
        """Test the main example from problem description."""
        funnels = [
            "checkout_process,view_product,add_to_cart,enter_payment_info,
    complete_order",
            "new_user_signup,visit_landing_page,click_signup,complete_profile,
    first_purchase",
            "three_clicks_then_add,click_product,click_product,click_product,
    add_to_cart",
```

```
        ]

        events = [
            "1,1000,visit_landing_page",
            "1,1005,click_signup",
            "1,1030,complete_profile",
            "2,1001,visit_landing_page",
            "2,1010,click_signup",
            "2,1020,complete_profile",
            "3,1002,visit_landing_page",
            "3,1100,click_signup",
            "4,1200,view_product",
            "4,1210,add_to_cart",
            "4,1300,enter_payment_info",
            "5,1250,view_product",
            "5,1260,add_to_cart",
            "5,1270,enter_payment_info",
            "5,1280,complete_order",
            "6,2000,click_product",
            "6,2005,click_product",
            "6,2010,click_product",
            "6,2015,add_to_cart",
            "7,3000,click_product",
            "7,3005,click_product",
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)

        expected = [
            "checkout_process,view_product(2),add_to_cart(2),enter_payment_info(2)
    ,complete_order(1)",
            "new_user_signup,visit_landing_page(3),click_signup(3),
    complete_profile(2),first_purchase(0)",
            "three_clicks_then_add,click_product(2),click_product(2),click_product
    (1),add_to_cart(1)",
        ]

        assert set(result) == set(expected)


    def test_single_user_complete_funnel(self):
        """Test single user completing entire funnel - happy path."""
        funnels = ["simple,a,b,c"]
        events = [
            "1,100,a",
            "1,200,b",
            "1,300,c"
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["simple,a(1),b(1),c(1)"]

        assert set(result) == set(expected)


    def test_user_drops_off_middle(self):
        """Test user dropping off at middle step."""
```

```
        funnels = ["dropout,step1,step2,step3"]
        events = [
            "1,100,step1",
            "1,200,step2",
            # User 1 stops here
            "2,100,step1",
            # User 2 stops after step1
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["dropout,step1(2),step2(1),step3(0)"]

        assert set(result) == set(expected)


    def test_out_of_order_events_ignored(self):
        """
        Test events not matching funnel order are ignored.
        Critical: Out-of-order events should NOT advance the funnel.
        """
        funnels = ["ordered,first,second,third"]
        events = [
            "1,100,second",  # Out of order - should be ignored
            "1,200,first",   # Now starts the funnel
            "1,300,second",  # Now in order
            "1,400,third"    # Completes
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["ordered,first(1),second(1),third(1)"]

        assert set(result) == set(expected)


    def test_repeated_steps_double_click(self):
        """
        Test funnel with repeated step names - the tricky case.
        This is what differentiates index-based vs name-based tracking.
        """
        funnels = ["double_click,click,click,purchase"]
        events = [
            "1,100,click",
            "1,200,click",
            "1,300,purchase",
            "2,100,click",     # Only one click
            "2,200,purchase"   # Doesn't count - missing second click
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["double_click,click(2),click(1),purchase(1)"]

        assert set(result) == set(expected)


    def test_repeated_steps_triple_click(self):
        """Test funnel requiring three identical events."""
```

```
        funnels = ["triple,click,click,click"]
        events = [
            "1,100,click",
            "1,200,click",
            "1,300,click",
            "2,100,click",
            "2,200,click",
            # User 2 only has 2 clicks
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["triple,click(2),click(2),click(1)"]

        assert set(result) == set(expected)


    def test_multiple_users_different_progress(self):
        """Test multiple users progressing at different rates."""
        funnels = ["test,a,b,c"]
        events = [
            "1,100,a",
            "1,200,b",
            "1,300,c",
            "2,500,a",
            "2,600,b"   # User 2 stops at b
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        # Both users reach a and b, only user 1 reaches c
        expected = ["test,a(2),b(2),c(1)"]

        assert set(result) == set(expected)


    def test_no_users_complete_any_step(self):
        """Test when no users reach any funnel steps."""
        funnels = ["empty,step1,step2"]
        events = [
            "1,100,unrelated_event",
            "2,200,other_event"
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["empty,step1(0),step2(0)"]

        assert set(result) == set(expected)


    def test_all_users_drop_at_first_step(self):
        """Test all users drop off immediately after first step."""
        funnels = ["early_exit,start,middle,end"]
        events = [
            "1,100,start",
            "2,200,start",
            "3,300,start"
        ]
```

```
196
197         sol = Solution()
198         result = sol.compute_funnel_counts(funnels, events)
199         expected = ["early_exit,start(3),middle(0),end(0)"]
200
201         assert set(result) == set(expected)
202
203
204     def test_multiple_funnels_same_events(self):
205         """
206         Test user progressing through multiple funnels simultaneously.
207         Same events can advance user in multiple independent funnels.
208         """
209         funnels = [
210             "funnel_a,x,y,z",
211             "funnel_b,x,y,z"
212         ]
213         events = [
214             "1,100,x",
215             "1,200,y",
216             "1,300,z"
217         ]
218
219         sol = Solution()
220         result = sol.compute_funnel_counts(funnels, events)
221         expected = [
222             "funnel_a,x(1),y(1),z(1)",
223             "funnel_b,x(1),y(1),z(1)"
224         ]
225
226         assert set(result) == set(expected)
227
228
229     def test_extra_events_after_completion(self):
230         """
231         Test that events after funnel completion don't affect counts.
232         Once a user completes, additional matching events are ignored.
233         """
234         funnels = ["complete,a,b"]
235         events = [
236             "1,100,a",
237             "1,200,b",
238             "1,300,a",  # Repeating 'a' after completion
239             "1,400,b"   # Repeating 'b' after completion
240         ]
241
242         sol = Solution()
243         result = sol.compute_funnel_counts(funnels, events)
244         expected = ["complete,a(1),b(1)"]
245
246         assert set(result) == set(expected)
247
248
249     def test_interleaved_events_multiple_users(self):
250         """Test multiple users with interleaved events."""
251         funnels = ["interleaved,step1,step2,step3"]
252         events = [
253             "1,100,step1",
254             "2,101,step1",
```

```
            "1,102,step2",
            "2,103,step2",
            "1,104,step3",
            # User 2 doesn't complete
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["interleaved,step1(2),step2(2),step3(1)"]

        assert set(result) == set(expected)


    def test_single_step_funnel(self):
        """Test funnel with only one step - boundary case."""
        funnels = ["single,only_step"]
        events = [
            "1,100,only_step",
            "2,200,only_step",
            "3,300,only_step"
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["single,only_step(3)"]

        assert set(result) == set(expected)


    def test_many_irrelevant_events(self):
        """
        Test filtering when most events are irrelevant.
        Performance consideration: should efficiently skip non-matching events.
        """
        funnels = ["signal,start,end"]
        events = [
            "1,100,noise1",
            "1,101,noise2",
            "1,102,start",   # Signal
            "1,103,noise3",
            "1,104,noise4",
            "1,105,end",     # Signal
            "1,106,noise5"
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["signal,start(1),end(1)"]

        assert set(result) == set(expected)


    def test_user_restarts_funnel(self):
        """
        Test user who appears to restart but funnel doesn't reset.
        Important: Once started, continue from current position.
        """
        funnels = ["no_restart,a,b,c"]
        events = [
```

```
            "1,100,a",
            "1,200,b",
            "1,300,a",  # Not a restart - stays at position after b
            "1,400,c"   # Completes from where they were
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["no_restart,a(1),b(1),c(1)"]

        assert set(result) == set(expected)


    def test_empty_funnel_list(self):
        """Test edge case: no funnels defined."""
        funnels = []
        events = [
            "1,100,some_event"
        ]

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = []

        assert result == expected


    def test_empty_events_list(self):
        """Test edge case: no events occurred."""
        funnels = ["empty,a,b,c"]
        events = []

        sol = Solution()
        result = sol.compute_funnel_counts(funnels, events)
        expected = ["empty,a(0),b(0),c(0)"]

        assert set(result) == set(expected)


# Run with: pytest -v faire_interview_prep_11.23_tests.py
# For coverage: pytest --cov=solution --cov-report=html
```

5 Edge Cases Deep Dive

5.1 Critical Edge Cases

1. **Repeated Steps**
   - Funnel: `click,click,click`
   - Requires 3 separate click events
   - Must use index-based tracking, not name-based
2. **Out-of-Order Events**
   - Events not matching current step are skipped
   - State machine doesn't go backwards
   - Example: `second,first,second,third` → `first,second,third`
3. **Empty Completion**
   - All users drop off before any step
   - Output: `funnel,step1(0),step2(0),...`
4. **Post-Completion Events**
   - Events after funnel completion are ignored
   - User counted once per step, not multiple times
5. **Multiple Funnels**
   - Same events can advance user in multiple independent funnels
   - Each funnel maintains separate state per user
6. **Boundary Conditions**
   - Empty funnel list: return empty result
   - Empty events list: all steps show count(0)
   - Single-step funnel: valid and should work

5.2 Common Implementation Bugs

⚠ Critical Point

Watch out for these bugs:

1. **Using step name as dictionary key**
   - Won't work: `step_users[step_name].add(user)`
   - Correct: `step_users[step_index].add(user)`
2. **Not resetting user state per funnel**
   - Each user needs independent state for each funnel
   - Don't share state across funnels
3. **Sorting events unnecessarily**
   - Events are already globally sorted (given)
   - Don't waste time re-sorting
4. **String formatting errors**
   - Output: `step(count)` not `step (count)` or `step:count`
   - Use `f"{step_name}({count})"` for correctness

6 Interview Execution Strategy

6.1 Clarifying Questions to Ask

Action Item

Start with these questions:

1. Are events guaranteed to be sorted by timestamp? *(Yes)*
2. Can a funnel have repeated step names? *(Yes - critical!)*
3. Should we count each user only once per step? *(Yes)*
4. Can users be in multiple funnels simultaneously? *(Yes)*
5. What happens if a user performs events out of order? *(Ignored)*
6. Are user IDs always integers? *(Yes)*
7. What's the scale? *(10^5 events)*

6.2 Implementation Approach

Time Management (30-35 minutes for coding):

1. **5 min**: Clarify requirements, discuss approach
2. **15 min**: Implement core solution
3. **10 min**: Write basic test cases
4. **5 min**: Walk through edge cases, add more tests

Incremental Testing Strategy:

1. Start with simplest test: 1 user, 1 funnel, complete path
2. Add complexity: multiple users
3. Test critical feature: repeated steps
4. Test edge case: out-of-order events
5. Test boundary: empty states

6.3 What to Verbalize During Interview

Action Item

Communicate your thinking:

- "I'm using a state machine approach where each user maintains position in each funnel"
- "The key insight is using step *index* not step *name* to handle repeated steps"
- "I'm using sets to ensure each user is counted once per step"
- "Events are already sorted, so I don't need to re-sort"
- "Let me write a test for repeated steps since that's the tricky part"

7 Optimization Discussion

7.1 Current Solution Analysis

Time Complexity: $O(F \times U \times E_u)$

- F = number of funnels (≤ 100)
- U = number of unique users
- E_u = average events per user
- For 10^5 total events: very manageable

Space Complexity: $O(F \times S \times U + E)$

- $F \times S \times U$ for step-user sets
- E for storing events grouped by user

7.2 Alternative Approaches (and why they're not better)

1. **Pre-filter events by funnel steps**
   - Idea: Only process events that match some funnel step
   - Why not better: Still need to track state, adds overhead
2. **Use hash maps for step tracking**
   - Idea: `dict[step_name] -> set(users)`
   - Why not better: Doesn't work for repeated steps
3. **Process events in single pass without grouping**
   - Idea: Maintain user state dictionary during event iteration
   - Why not better: Same complexity, harder to implement correctly

7.3 When to Optimize

Key Insight

Don't over-optimize:

- Current solution is already optimal for given constraints
- Premature optimization wastes interview time
- Focus on correctness and testing first
- Only optimize if interviewer asks or constraints change

8 Summary & Key Takeaways

8.1 Algorithm Checklist

Action Item

Before submitting solution, verify:

- [x] Parse funnels correctly (name, steps list)
- [x] Group events by user_id
- [x] Use step INDEX not step NAME for tracking
- [x] Use sets for user counting (not lists)
- [x] Process events in order (don't re-sort)
- [x] Format output correctly: `funnel,step(count),...`
- [x] Handle repeated steps correctly
- [x] Ignore out-of-order events

8.2 Testing Checklist

Testing Focus

Minimum test coverage:

- [x] Happy path (complete funnel)
- [x] Repeated steps (e.g., click,click,click)
- [x] Out-of-order events
- [x] User drop-offs
- [x] Multiple users
- [x] Edge cases (empty funnels/events, single step)

8.3 Interview Success Factors

1. **Clarify requirements**: Ask about repeated steps
2. **Explain approach**: State machine pattern
3. **Implement incrementally**: Start simple, add complexity
4. **Test thoroughly**: Faire values comprehensive testing
5. **Handle edge cases**: Repeated steps, out-of-order, empty states
6. **Communicate clearly**: Verbalize your thought process

8.4 Final Preparation

Action Item

Practice plan:

1. Implement solution from scratch (no looking): 20 minutes
2. Write all test cases: 15 minutes
3. Run tests and debug: 10 minutes
4. Explain algorithm to someone: 5 minutes
5. **Total practice time: 50 minutes**

Do this 2-3 times before interview for mastery.