📚 CATEGORY 1 — 10 EXTREME SPARSE TEST CASES

Test cases: T00–T09

What they contain (concrete link to T00–T09)

- Many {} empty rows (e.g. T00, T01, T03, T08, T09)
- Rows that have one small value surrounded by empties:
  - e.g. {{},{},{1}}, {{2},{},{3}} in **T00**
- Long stretches of empties with occasional numbers or zeros:
  - e.g. **T03**, **T07**, **T08**
- Matrices where multiplication can *technically* proceed but most structure is missing

Where this is useful in real life

These patterns appear anywhere you have **very incomplete or sparse data**:

- **Financial feeds**
  - Some fields or instruments don't update every tick
  - You get many "empty" or default rows with only a handful of meaningful entries
  - Example: order book levels 2–10 missing, only top-of-book present
- **IoT / sensor networks**
  - Battery-saving modes → many timestamps with missing values
  - Some sensors sleep or drop out completely
- **Healthcare wearables**
  - Heart rate recorded regularly, but temperature or SpO_2 missing at many points
- **User analytics / clickstreams**
  - Many users with almost no events (very sparse rows), a few with lots

Why your resilient algorithm is relevant here

Your rules:

- Skip empty rows safely
- Stop individual cell calculations when there's no more usable data

- Still produce **partial but meaningful results** from what *is* present

That's exactly what robust systems do when dealing with sparse logs or sparse matrices.

📚 CATEGORY 2 — 10 EXTREME IMBALANCE TEST CASES

Test cases: T10–T19

What they contain (linked to actual tests)

- **Very wide vs very tall matrices**:
  - e.g. T10: 1 row with many columns, next matrix has many short rows
  - T11, T13, T14, T16, T18: tall or wide shapes with serious mismatch
- Strong asymmetry:
  - One side has long rows (many columns)
  - Other has many small rows (many rows)

These are the "1×N × N×1 but jagged" style cases.

Where this is useful in real life

- **Machine Learning / NLP**
  - Variable-length token sequences (sentences of different lengths)
  - Combined with fixed-size embeddings or projection matrices
  - Your logic is similar to handling **ragged sequences** and still computing attention/aggregation where possible.
- **Trading / Market Data**
  - One stream: a single record with many features (e.g. 40 columns)
  - Another stream: many short rows (e.g. trades per second) with fewer features
  - You're effectively multiplying "very wide" snapshots with "very tall" event streams.
- **Log / telemetry aggregation**
  - Huge log lines with dozens of fields vs "summary rows" with only a few
  - Structured vs semi-structured records

- **Joining different data sources**
    - A table with 50 columns × a table with 3 columns
    - Incomplete dimension alignment but you still want best-effort metrics

Why your algorithm is relevant here

Your rules:

- Discard **surplus row elements** in A when B is shorter
- Discard **surplus column elements** in B when A is shorter
- Stop cells only when there is truly **insufficient** data to continue

This is exactly what a **resilient ETL pipeline** or **data-fusion engine** does when joining very mismatched sources.

📚 CATEGORY 3 — 10 NULL-HEAVY / MISSING & NEGATIVE TEST CASES

Test cases: T20–T29

What they contain

- Explicit null values all over the rows:
    - e.g. T20, T21, T22, T23, T25, T26, T27, T28, T29
- Mixture of:
    - valid integers
    - null
    - sometimes negative values

e.g.
{{null,-1,2},{3,null,-4}},
{{null,10,null},{-5,null,5}}, etc.

Where this is useful in real life

- **Data warehouses / BI**
    - Left joins often produce nulls for columns with no match
    - Many analytics systems must handle null × value correctly
- **Insurance / banking models**
    - Missing income, missing credit score, missing address fields

    - A scoring engine must decide how to treat nulls vs zeros
- **Medical datasets**
    - Missing lab results or vitals for certain visits
    - Need to compute risk scores from partially missing records
- **ETL / Data cleaning tools**
    - Complex pipelines where nulls pop up after schema evolution

Why your algorithm is relevant here

Your rules:

- Treat null as "no contribution" but **do not crash**
- Allow a row with some nulls to continue multiplying its other values
- Combine with negatives (gains/losses) in a sane way

That's **exactly** how Spark, Pandas, and SQL analytics often reason about null-containing data.

📚 CATEGORY 4 — 10 LARGE NUMERIC / STRESS TEST CASES

Test cases: T30–T39

What they contain

- Very large magnitudes:
    - 1000000000, -2000000000, 300000000, etc. (e.g. T30, T31, T33, T38)
- Big positive/negative combinations:
    - e.g. T31, T32, T34, T36, T39
- Often combined with irregular shapes

Where this is used in real life

- **Finance / risk systems**
    - Large monetary values, aggregated positions, P&L rolls
    - Stress-testing logic for overflow or precision loss
- **Scientific computing**
    - Large coefficients in models (climate, physics, energy simulations)

    - High magnitude matrix operations
- **Big simulations / gaming engines**
    - Transformation matrices with large scaling factors
    - Accumulated transforms across multiple frames
- **Machine learning weight matrices**
    - Large initialized or accumulated weights or gradients

Why your algorithm is relevant here

Your resilient logic must:

- Handle large multiplies without logical failure
- Maintain correct sign and accumulation even if shape is jagged
- Not “give up” because the numbers are big and structure weird

These tests mirror how your algorithm behaves under **real-world numeric load**, not just toy integers.

📚 CATEGORY 5 — 60 MIXED REALISTIC JAGGED CASES

Test cases: T40–T99
(these are the “original-style” AI-designed ones)

These are more heterogeneous and many of them fall into **multiple** of the earlier categories at once. They’re meant to feel like real messy input rather than clean lab experiments.

What they contain

- Multi-matrix chains (3–6 matrices each), e.g. T40–T44, T60–T69
- Mixture of:
    - jagged rows
    - 0’s
    - occasional nulls
    - some empty rows {}
    - negative values in later ones (e.g. T98, T99)

For example:

- **T40–T49**:
  Classic jagged multi-step chains similar to your earlier test styles
- **T50–T59**:
  Mixture of:
    - short & long rows
    - partially empty rows
    - some nulls
    - structured irregularities
- **T60–T69**:
  Deeper chains, changing shape at each step, including rows suddenly expanding or collapsing
- **T70–T79**:
  Heavily **0-focused**, exploring how your code treats complete zeros vs blanks vs nulls
- **T80–T89**:
  More subtle mixes:
  zeros + nulls + partial rows, but not as extreme as the first 40
- **T90–T99**:
  "End boss" style: combined null + 0 + negatives + semi-random jaggedness

Where this is useful in real life

These 60 cases behave like **real production inputs**, not synthetic stress-only patterns.

They match:

- **Log pipelines**
    - Mixed clean and dirty records
    - Some fully populated, some barely filled
- **Multi-source data fusion**
    - Combine 4–6 different systems, each with its own quirks
    - Some sources are consistent, others chaotic
- **Incrementally evolving schemas**
    - As systems evolve, older rows adhere to older formats and look "short"
    - Newer rows have more fields (longer rows)

- **Real ML feature engineering pipelines**
    - Many optional features
    - Some models look at only some columns
    - Some rows are missing entire feature groups

Why your algorithm is relevant here

Your rules handle all of this by:

- Continuing when it can
- Stopping only when truly impossible (not just inconvenient)
- Distinguishing null, zero, and "missing row"
- Surviving long chains of transformations without collapsing

This is exactly the kind of robustness you need in **production data systems**.

🧩 How to Think About It Altogether

You now have:

- **T00–T09** → extreme sparse stress
- **T10–T19** → extreme imbalance
- **T20–T29** → null & missing-heavy with negatives
- **T30–T39** → large-number stress tests
- **T40–T99** → realistic industry-style mixtures (often touching multiple categories at once)