
Booby Trap Treasure Hunt

Problem Statement

- Convex Hull Trick inspiration
- input constraints
- cases fail for Naive DP
- greedy, ordered, mixed
- hard

Sample Input 1

```
3
1 10
3 3
5 0
```

Sample Output 1

```
45
```

Sample Input 2

```
3
1 10
10 5
11 0
```

Sample Output 2

```
110
```

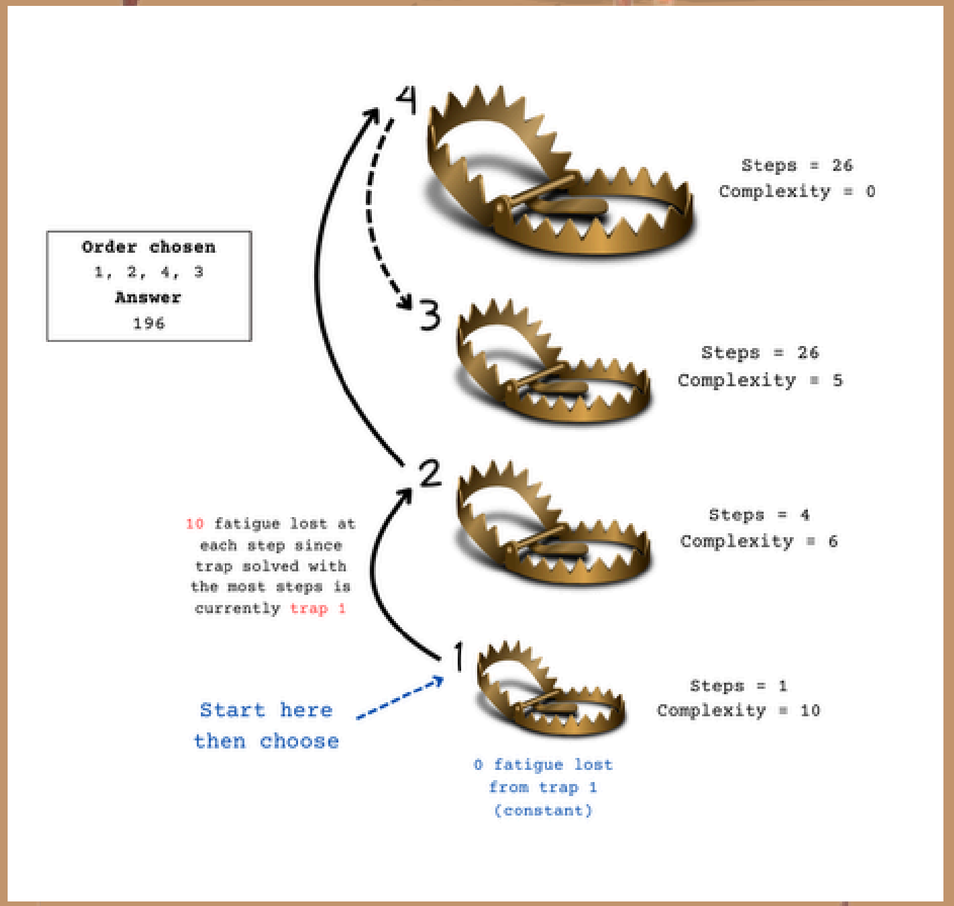
Sample Input 3

```
4
1 10
4 6
25 5
26 0
```

Sample Output 3

```
196
```

n	Slowest Accepted Algorithm
≤ 10	$O(n!), O(n^6)$
≤ 15	$O(2^n \times n^2)$
≤ 20	$O(2^n), O(n^5)$
≤ 50	$O(n^4)$
$\leq 10^2$	$O(n^3)$
$\leq 10^3$	$O(n^2)$
$\leq 10^5$	$O(n \log_2 n)$
$\leq 10^6$	$O(n), O(\log_2 n), O(1)$


Order chosen
1, 2, 4, 3
Answer
196
4
Steps = 26
Complexity = 0
3
Steps = 26
Complexity = 5
2
Steps = 4
Complexity = 6
10 fatigue lost at each step since trap solved with the most steps is currently trap 1
1
Steps = 1
Complexity = 10
Start here then choose
0 fatigue lost from trap 1 (constant)

Optimal Solution - Convex Hull Trick

$dp_i = \min_{j<i} (steps_i \times complexity_j + dp_j)$ → $y = mx + b$

Algorithm Idea:

- Maintain “hull”
- Add lines, delete useless lines
- query optimal dp[i]

Optimal Solution - Convex Hull Trick

- dp[]
- deque()

Line(m, c)

int getintersectx():

$m_1x + b_1 = m_2x + b_2 \Rightarrow x = (b_2 - b_1) / (m_1 - m_2)$

int eval()

helper functions:

bool is_bad(l1, l2, l3)

void add_line(line)

int query(x)

Inefficient Solution - Naive DP

- Time complexity: $O(n^2)$
- Space complexity: $O(n)$
- Fails for large input
- Seconds longer

```python
# Time complexity: O(n^2)
n = int(input())
steps = []
complexities = []
for i in range(n):
    s, c = map(int, input().split())
    steps.append(s)
    complexities.append(c)

dp = [float('inf')] * n
dp[0] = 0

for i in range(1, n):
    for j in range(i):
        cost = dp[j] + (complexities[j] * steps[i])
        dp[i] = min(dp[i], cost)
print(int(dp[n - 1]))
```

Wrong Solution - Greedy

- Time complexity: O(1)
- Space complexity: O(n)
- Always goes for last trap (smallest complexity)
- Plain wrong

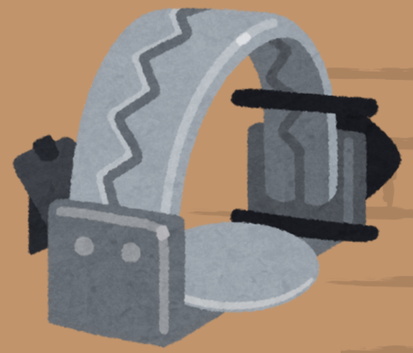
```python
from sys import stdin
# Greedy approach is wrong
n = int(input())
steps = []
complexities = []
for i in range(n):
    s, c = map(int, input().split())
    steps.append(s)
    complexities.append(c)

print(steps[n - 1] * complexities[0])
```

Challenges

$steps_1 = 1$, $steps_1 < steps_i < steps_n$ and
$complexity_1 > complexity_i > complexity_n$
$complexity_n = 0$.

Constraints

- 1 <= N <= 10^5
- steps are strictly increasing
- complexities are strictly decreasing
- $steps_1 = 1$
- $complexity_N = 0$



Input validator

- Use pointers to keep track of previous values for steps and complexity

```
try:
    n_line = sys.stdin.readline()
    print(repr(n_line)) # useful for debugging to see where we have read
    assert re.match('^[1-9][0-9]*\n$', n_line) # note: no leading zeros
    n = int(n_line)
    assert 1 <= n <= 100000

    prev_steps = prev_complexity = None
    for i in range(1, n+1):
        case_line = sys.stdin.readline()

        print(f"current line: {i + 1}")
        print(f"Validating line format {repr(case_line)} against regex...")
        assert re.match('^(0|[1-9][0-9]*) (0|[1-9][0-9]*)\n$', case_line)
        print("Done validating line format against regex..")
        # parse the line
```

Test Generator

- Use of pointers
- Challenge:
  - Random value generation

Example:

- N = 5
- $steps_1$ = 0
- $step_2$ randomly assigned max_int

Solution:

- adjust lower/ upper bound of possible values

Correctness

- Manual tests
- Validate against solution

Edge case / malformed input handling

- Regex to check input format
- additionnal logic to check constraints

```
N vs lines:
    N > lines
    N < lines
    N = lines

complexity:
    curr > prev
    curr < prev
    curr = prev
    complexity_n = 0
    complexity_n != 0

steps:
    curr > prev
    curr < prev
    curr = prev
    steps_1 = 0
    steps_1 != 0

input:
    lines don't contain expected input type (int)
    too many ints / line, not enough ints / line
```

Presentation

- dungeon theme
- CHT, different contexts
- LLM for teaching and coding
- good collaboration
- improvements: dynamic CHT, Li Chao Tree