2주차 풀이 읽기자료

문제 조건에 따른 상태 선택

문제	저장한 상태	갱신 근거	비용
중복 없는 최장 구간	집합, 왼쪽 끝	중복 제거 후 삽입	평균 O(n) 시간
목표 합의 최단 구간	합, 왼쪽 끝	양수와 유효 길이 기록	O(n) 시간
Min Stack	깊이별 값과 최소	이전 최소의 복원	조회 O(1)
후위 표기 계산	부분식 값의 스택	오른쪽부터 꺼내 결합	O(t) 시간

시간을 계산할 때 포인터 이동, 집합 연산, 리스트 재할당을 구분합니다. 결과가 맞는 예제와 함께 상태가 깨지는 반례도 설명합니다.

3 · 단계별 힌트

중복 없는 최장 연속 부분문자열의 길이를 반환합니다. 공백도 세며 빈 입력은 0입니다.

LeetCode 3: Longest Substring Without Repeating Characters · Medium

- H1. 현재 구간의 모든 문자가 서로 다른가요?
- H2. 중복 한 개를 없애려면 왼쪽을 몇 번 옮겨야 하나요?
- H3. 중복 문자가 구간에서 사라질 때까지 왼쪽 문자를 제거하고 새 문자를 추가합니다.

`text="abba c"`

right, 문자	left 이전	지우는 문자	삽입 후 구간	best
0, a	0	없음	a	1
1, b	0	없음	ab	2
2, b	0	a, b	b	2
3, a	2	없음	ba	2
4, 공백	2	없음	ba␣	3
5, c	2	없음	ba␣c	4

3 · 정당성과 비용

추가 직후 집합은 중복 없는 현재 구간과 같습니다. 오른쪽 끝을 고정할 때 가능한 가장 왼쪽 경계를 유지하므로 그 끝의 최장 구간을 검사합니다. 두 포인터는 각각 최대 n번 이동합니다.

right는 n번, left도 뒤로 가지 않아 최대 n번 움직입니다. 겉보기 중첩 반복을 n×n으로 세지 않고 전체 이동을 합하면 최대 2n입니다. set 연산이 기대 O(1)이므로 기대 시간 O(n), 동시에 든 문자 수가 최대 min(n,a)이므로 보조 공간 O(min(n,a))입니다. a는 문자 종류 수입니다.

3 · 반례와 검증

“abba”에서 한 번만 제거하면 b가 남아 불변식이 깨집니다. 빈 문자열은 0, 공백 한 개는 1입니다.

```
text="abba c" → 4 ("ba c")
```

3 · 완성 구현

```
def longest_unique(text):
    active, left, best = set(), 0, 0
    for right, char in enumerate(text):
        while char in active:
            active.remove(text[left])
            left += 1
        active.add(char)
        best = max(best, right - left + 1)
    return best
```

중복이 사라질 때까지 old left의 문자를 제거한 뒤 left를 옮깁니다. abba의 두 번째 b에서 a 하나만 지우면 여전히 b가 남으므로 if 한 번으로 끝낼 수 없습니다. 새 문자를 넣고 나서 길이를 계산합니다.

209 · 단계별 힌트

양수 배열에서 합이 양수 target 이상인 연속 구간의 최소 길이를 반환합니다. 그런 구간이 없으면 0입니다.

LeetCode 209: Minimum Size Subarray Sum · Medium

- H1. 합을 유지하면 매 구간을 다시 더할 필요가 있나요?
- H2. 왼쪽 값을 제거하면 합은 어느 방향으로 변하나요?
- H3. 오른쪽 값을 합에 넣고 조건을 만족하는 동안 길이를 기록하며 왼쪽을 줄입니다.

`target=6, nums=[1,2,4,2]`

동작	구간	total	best
1, 2까지 추가	[1,2]	3	미발견
4 추가, 길이 기록	[1,2,4]	7	3
1 제거, 길이 기록	[2,4]	6	2
2 제거, 축소 종료	[4]	4	2
마지막 2 추가, 기록	[4,2]	6	2

209 · 정당성과 비용

양수이므로 왼쪽을 줄이면 합이 감소합니다. 유효한 구간을 버리기 전에 길이를 기록하고 더 짧은 후보를 검사합니다. 이미 버린 왼쪽 경계는 이후 더 긴 구간만 만들므로 개선 후보가 아닙니다.

원소마다 합에 한 번 더하고 최대 한 번 뺍니다. 양 끝 이동의 합이 2n 이하이므로 전체 O(n) 시간입니다. 구간을 복사하지 않고 total, left, right, best만 저장하므로 보조 공간 O(1)입니다. 입력이 n칸인 것과 추가 작업 공간은 별도로 셉니다.

209 · 반례와 검증

음수 확장 반례 target=3, [1,-1,3]: 일반 축소는 길이 1을 놓칩니다. 이는 공식 양수 조건 밖의 예입니다.

```
target=6, nums=[1,2,4,2] → 2
```

209 · 완성 구현

```
def min_subarray(target, nums):
    total = left = 0
    best = len(nums) + 1
    for right, value in enumerate(nums):
        total += value
        while total >= target:
            best = min(best, right - left + 1)
            total -= nums[left]
            left += 1
    return 0 if best > len(nums) else best
```

right의 값을 먼저 더합니다. 목표 합을 만족하는 동안 현재 길이를 기록하고 왼쪽 값을 빼서 더 짧은 구간을 시험합니다. best=n+1은 아직 답을 못 찾았다는 표시이며, 가능한 구간 길이는 최대 n입니다.

155 · 단계별 힌트

push는 값을 넣고 pop은 맨 위 값을 제거합니다. top은 맨 위 값, getMin은 남은 전체 값의 최소를 반환하며, 빈 스택에는 조회·제거를 호출하지 않습니다.

LeetCode 155: Min Stack · Medium

- H1. 값을 제거한 뒤 이전 최소를 어떻게 복원하나요?
- H2. 최소가 중복되면 한 개를 지워도 되나요?
- H3. 각 깊이에 (값, 그 깊이까지의 최소)를 저장합니다. self.entries는 인스턴스마다 따로 생성합니다.

`push(2), push(2), push(5), pop(), pop()`

연산	저장한 (값, 최솟값)	getMin 결과
push(2)	[(2,2)]	2
push(2)	[(2,2),(2,2)]	2
push(5)	[(2,2),(2,2),(5,2)]	2
pop()	[(2,2),(2,2)]	2
pop()	[(2,2)]	2

155 · 정당성과 비용

빈 상태에서 시작해 새 최소를 min(value, 이전 최소)로 저장하면 귀납적으로 각 깊이의 최소가 맞습니다. pop은 마지막 쌍만 없애므로 이전 최소가 복원됩니다.

깊이 d에서 getMin과 top은 마지막 튜플 한 칸만 읽어 O(1)입니다. push·pop은 list 재할당을 포함한 분할상환 O(1)입니다. 각 깊이마다 값 두 개를 저장하므로 2d개 값, 즉 O(d) 공간입니다. getMin마다 전체 min을 계산하면 한 질의가 O(d)로 바뀝니다.

155 · 반례와 검증

min(entries)를 호출마다 계산하면 O(d)입니다. 최소값 하나만 저장하면 최소를 pop한 뒤 이전 값을 잃습니다.

```
push(2),push(2),push(5),pop(),pop(),getMin() → 2
```

155 · 완성 구현

```
class MinStack:
    def __init__(self):
        self.entries = []

    def push(self, val):
        minimum = min(val, self.entries[-1][1]) if self.entries else val
        self.entries.append((val, minimum))

    def pop(self):
        self.entries.pop()

    def top(self):
        return self.entries[-1][0]

    def getMin(self):
        return self.entries[-1][1]
```

각 튜플은 (현재 값, 이 깊이까지의 최솟값)입니다. pop은 마지막 튜플만 없애 이전 깊이의 최솟값을 다시 드러냅니다. getMin은 전체를 훑는 min 연산 없이 마지막 튜플의 두 번째 칸만 읽습니다.

entries[-1][1]에서 −1은 마지막 튜플, 1은 그 튜플의 두 번째 값입니다. 예를 들어 [(4,4),(7,4)]이면 top은 7, getMin은 4입니다. 빈 리스트에서는 조건식의 else 쪽 val만 평가하므로 없는 마지막 항목을 읽지 않습니다.

150 · 단계별 힌트

유효한 후위 표기 문자열 토큰을 계산합니다. 나눗셈은 0 방향으로 절삭하며, 0으로 나누는 입력은 없습니다.

LeetCode 150: Evaluate Reverse Polish Notation · Medium

- H1. 먼저 꺼낸 값은 왼쪽인가요, 오른쪽인가요?
- H2. -5/2의 절삭 방향은 무엇인가요?
- H3. 숫자는 push, 연산자는 right와 left를 순서대로 pop한 뒤 결과를 push합니다. 나눗셈은 절댓값 정수 몫에 부호를 붙입니다.

`tokens=["9","4","-","-2","/"]`

토큰	계산 전 스택	수행하는 연산	계산 후 스택
9	[]	push(9)	[9]
4	[9]	push(4)	[9,4]
-	[9,4]	left=9, right=4	[5]
-2	[5]	push(-2)	[5,-2]
/	[5,-2]	5 / -2, 0 방향 절삭	[-2]

150 · 정당성과 비용

스택은 지금까지 끝난 부분식의 값을 왼쪽부터 보존합니다. 연산자는 마지막 두 부분식을 하나로 줄입니다. 유효한 식이 끝나면 전체 식 값 하나가 남습니다.

t개 토큰 각각에서 최대 두 번 pop, 한 번 append, 산술 연산 한 번을 수행하여 O(t) 시간입니다. 문제의 중간 값은 32비트 범위라 산술을 O(1)로 셉니다. 아직 결합되지 않은 피연산자가 많이 쌓이면 스택 크기가 O(t)이므로 보조 공간도 O(t)입니다. 마지막 한 값만 남는다는 사실이 실행 중 공간을 줄이지는 않습니다.

150 · 반례와 검증

5//-2는 -3이지만 0 방향 절삭은 -2입니다. left-right를 right-left로 바꾸면 뺄셈이 달라집니다.

`tokens=["9","4","-","-2","/"] → -2`

`(left < 0) != (right < 0)`는 두 부호가 다른지 검사합니다. 5와 −2에서는 False != True가 True여서 −2, −5와 −2에서는 True != True가 False여서 +2입니다. 연산자 토큰 `"-"`와 숫자 토큰 `"-2"`는 서로 다른 문자열이므로 후자는 int로 변환됩니다.

150 · 완성 구현

```
def rpn(tokens):
    stack = []
    for token in tokens:
        if token not in {"+", "-", "*", "/"}:
            stack.append(int(token))
            continue
        right, left = stack.pop(), stack.pop()
        if token == "+":
            value = left + right
        elif token == "-":
            value = left - right
        elif token == "*":
            value = left * right
        else:
            quotient = abs(left) // abs(right)
            value = -quotient if (left < 0) != (right < 0) else quotient
        stack.append(value)
    return stack[0]
```

연산자를 만나면 먼저 꺼낸 값이 오른쪽 피연산자입니다. 두 번째 값이 왼쪽이며 계산 결과 하나를 다시 넣습니다. 나눗셈은 절댓값끼리 나눈 몫에 부호를 붙여 0 방향 절삭을 구현합니다.

3차시 확인 답

1. 마지막 고정 폭 구간의 합은 6입니다. right=3에서 index=1을 빼고 index=3을 더합니다.
2. “abba”의 두 번째 b에서 a만 지우면 이전 b가 남습니다. 집합은 그 구간의 중복 b를 표현하지 못합니다.
3. 두 포인터 모두 앞으로만 최대 n칸 움직입니다. 왼쪽 포인터를 처음으로 되돌리는 반복이 없습니다.
4. 양수 전용 규칙은 3을 반환하지만 실제 최소 길이는 1입니다. -1을 빼면 합이 커져서 축소를 멈추는 근거가 깨집니다.

4차시 확인 답

1. 최솟값은 2입니다. 두 번째 쌍을 제거해도 첫 번째 (2,2)가 남습니다.
2. 한 객체에 2, 다른 객체에 7을 넣습니다. top은 각각 2와 7, 두 번째 객체의 최소는 7이어야 합니다. 상태를 공유하면 이 조건이 깨집니다.
3. 첫 pop은 오른쪽 피연산자 4를 반환합니다. 계산은 9-4입니다.
4. Python의 내림 나눗셈은 -3, 0 방향 절삭은 -2입니다.
5. d개 항목 전체를 찾으면 O(d)가 되어, 위치로 바로 읽는 O(1) 조회보다 비용이 커집니다.

실행과 참고자료

```
cd code
python demo_03.py
python demo_04.py
python verify.py
```

완성 코드는 `solutions.py`, 독립 실습은 `student_week_02.py`에 있습니다. LeetCode에서는 공식 메서드 서명에 맞춰 함수를 연결합니다.

Erik Demaine and Srini Devadas, Introduction to Algorithms, MIT 6.006 (Fall 2011).