[효율성 비교] 시간복잡도와 메모리 복잡도, 구현했던 price format 비교해보기

구현 어려움 정도	개념
구현 완료	☑
기간	@November 29, 2024 → December 1, 2024
정리 완료	☑

시간 복잡도 (Time Complexity)

함수가 실행되는 동안 처리하는 연산의 양을 측정합니다. 예를 들어, 배열을 순회하는 것과 같은 작업은 그 길이에 비례해서 시간이 걸립니다. 처리해야 하는 데이터가 많을 때, 시간 복잡도가 높은 알고리즘은 성능에 문제가 될 수 있습니다.

메모리 복잡도 (Space Complexity)

알고리즘이 실행되는 동안 얼마나 많은 메모리를 사용하는지 측정합니다. 불필요하게 많은 데이터를 생성하거나 유지하면 메모리 낭비가 발생할 수 있습니다.

기존에 아래 링크의 블로그를 참고해서 정규식을 활용해 숫자 금액 단위 형식을 만들려고 했습니다.

참고: https://7942yongdae.tistory.com/193

블로그 코드

```
const moneyFormat = (value) => {
  const numbers = [
    numbering(value % 10000000000000000000, 10000000000000000),
    numbering(value % 10000000000000000, 1000000000000),
    numbering(value % 1000000000000, 100000000),
    numbering(value % 100000000, 10000),
    value % 10000
  ]

  return setUnitText(numbers)
          .filter(number => !!number)
          .join(' ');
}

const setUnitText = (numbers) => {
  const unit = ['원', '만', '억', '조', '경'];
  return numbers.map((number, index) => !!number ? numberFormat(number) + unit[(unit.length - 1) - index] :
}

const numbering = (value, division) => {
  const result = Math.floor(value / division);
  return result === 0 ? null : (result % division);
}

const NUMBER_FORMAT_REGX = /\B(?=(\d{3})+(?!\d))/g;

const numberFormat = (value) => {
  return value.toString().replace(NUMBER_FORMAT_REGX, ',');
}
```

숫자를 여러 구간으로 나누고, 각 구간에 대해 `numbering`, `setUnitText` 등의 함수를 호출하는 방법입니다.

시간 복잡도

- `numbers` 배열을 생성하고, `setUnitText`를 호출
- `numbers` 배열을 만들기 위해 `numbering` 함수가 각각 5번 호출 → 이 부분에서 `O(1)` 시간이 소요
- 그 다음 `setUnitText`에서 `map`을 돌며 `numberFormat`을 호출
- `setUnitText`에서 `numbers.length` 만큼 순회하므로, 이 부분에서 `O(n)` 시간이 소요
- `numberFormat`은 `String.replace()`를 사용해 숫자를 포맷하는데, 이 작업은 문자열의 길이에 비례하여 시간이 소요

메모리 복잡도

- `numbers` 배열은 5개의 요소를 가지며, `setUnitText`와 `numberFormat`에서 추가적인 임시 배열을 생성하고 결과를 반환
- 메모리 사용량은 상대적으로 적지만, `numbering`에서 추가적인 변수를 생성하므로 `O(1)`

적용한 코드

```
export const formatMarketCapPrice = (price: number, currency: string) => {
  const MILLION = 1000000;
  const BILLION = 1000000000;
  const TRILLION = 1000000000000;

  if (currency === 'USD') {
    const millionPrice = price / MILLION;
    return `$${millionPrice.toFixed(2)} million`;
  }

  if (price >= TRILLION) {
    const trillionPrice = price / TRILLION;
    return `${trillionPrice.toFixed(2)}조`;
  } else if (price >= BILLION) {
    const billionPrice = price / BILLION;
    return `${billionPrice.toFixed(2)}억`;
  } else if (price >= MILLION) {
    const millionPrice = price / MILLION;
    return `${millionPrice.toFixed(2)}백만`;
  }

  return price.toLocaleString();
};
```

시간 복잡도

이 코드는 여러 `if` 문을 순차적으로 평가합니다. 가격 값에 따라 한 번만 비교하고 그에 맞는 단위로 결과를 반환하므로 시간 복잡도는 `O(1)`입니다.

메모리 복잡도

코드에서 생성하는 변수들이 적고, 비교 연산이 주로 이루어지므로 `O(1)`입니다.

알고리즘의 성능을 측정 및 비교 가능한 사이트, 도구들

- JSPerf
  - Create Test 버튼을 클릭하고, 비교하고자 하는 코드를 입력
  - https://jsperf.com/
- Benchmark.js
  - 라이브러리로써 코드에 직접 통합하여 사용 가능
  - https://benchmarkjs.com/

- Google Chrome Developer Tools (Performance Tab)
  - Chrome 개발자 도구의 Performance 탭을 사용하여 자바스크립트 코드의 성능을 분석 가능
  - 코드의 실행 시간, 메모리 사용량 등을 측정
- Node.js에서 실행
  - `console.time()`과 `console.timeEnd()`는 코드 실행 시간 측정을 위한 간단한 방법

```
console.time("test");
// 테스트할 내 코드 작성
console.timeEnd("test");
```

참고

https://ledgku.tistory.com/33

https://noahlogs.tistory.com/27