**Ch. 3 자료구조 요약**

**03-1. 배열과 리스트 그리고 벡터**

**03-2. 구간 합**

**03-3. 투 포인터**

**03-4. 슬라이딩 윈도우**

**03-5. 스택과 큐**

**03-1. 배열과 리스트 그리고 벡터**

**배열:** 메모리의 연속적인 공간에 값이 채워져 있는 형태의 자료구조

* 특징

1. 인덱스를 사용하여 값에 바로 접근할 수 있다.
2. 삽입/삭제 연산이 어렵다. 배열의 특정 지점에 변화가 있을 시 해당 지점 주변에 있는 값을 이동시키는 과정이 필요하기 때문이다.
3. 선언 시 크기를 지정할 수 있다. 그러나 한 번 선언하면 그 크기를 변경할 수 없다.

**리스트:** 값과 포인터를 묶은 노드를 포인터로 연결한 자료구조

* 특징

1. 인덱스가 없으므로 값에 접근하는 속도가 느리다. -> Head 포인터부터 순차적으로 접근해야 한다.
2. 삽입/삭제 연산이 쉽다. 포인터로 연결되어 있기 때문에 포인터 간의 연결 관계를 새롭게 정의하기만 하면 되기 때문이다.
3. 선언 시 크기를 별도로 지정하지 않아도 된다. -> 크기가 가변적인 데이터를 다룰 때 용이하다.
4. 포인터를 저장할 공간이 필요하기 때문에 배열에 비해 구조가 복잡하다.

**벡터:** C++ 표준 라이브러리(STL)에 있는 자료구조 컨테이너 중 하나로, 동적 배열의 형태를 갖는다.

* 특징

1. 동적으로 원소를 추가할 수 있다.
2. 마지막 위치에 있는 데이터의 삽입/삭제 연산은 문제가 없지만, 중간 위치에 있는 데이터의 삽입/삭제 연산에 대해선 배열과 같은 메커니즘으로 동작한다.
3. 인덱스를 이용하여 요소에 접근할 수 있다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

벡터 사용법

**03-2. 구간 합**

**개요:** 합 배열을 이용하여 시간 복잡도를 더 줄이기 위해 사용하는 특수한 목적의 알고리즘이다.

**구간 합의 핵심 이론**

* 원시 배열 A에 대한 합 배열 S 정의:

// (S[i]) = (A[0]부터 A[i]까지의 합)

S[i] = A[0] + A[1] + A[2] + … + A[i – 1] + A[i]

* 합 배열 S를 만드는 공식

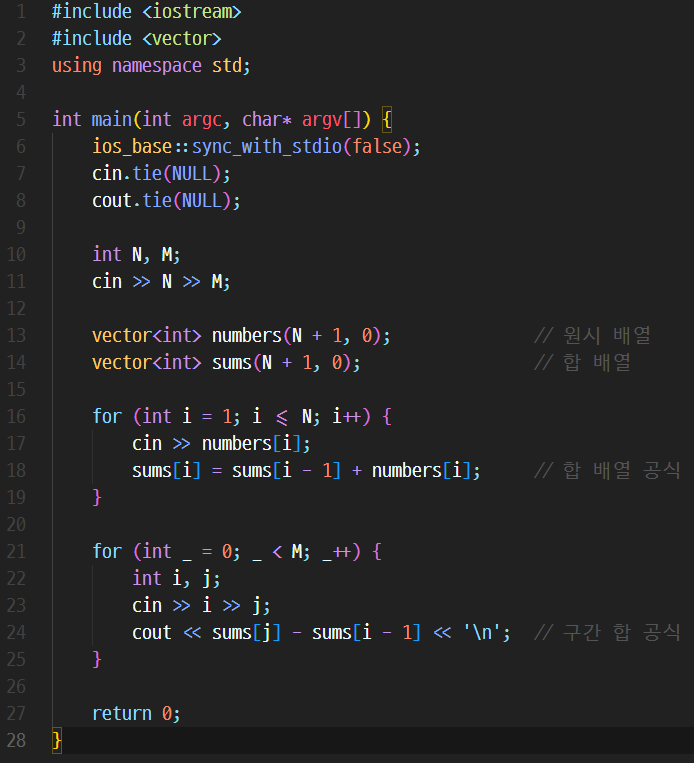
// 이전에 구한 결과를 활용하여 시간 복잡도 절약

S[i] = S[i – 1] + A[i]

* 구간 합을 구하는 공식

// i에서 j까지의 구간 합을 구한다고 가정

(part sum) = (S[j] – S[i – 1])



구간 합 공식의 활용 - 1차원 배열

구간 합의 핵심 이론을 활용하여 원시 배열에 데이터를 저장하고, 원시 배열의 데이터들로 합 배열을 채우는 것이 우선이다. 이때 (합 배열)[i] = (합 배열)[i – 1] + (원시 배열)[i] 공식을 사용한다. 이후 (i ~ j의 구간 합) = (합 배열)[j] – (합 배열)[i – 1] 공식을 사용하여 테스트 케이스마다의 구간 합을 O(1)의 시간 복잡도로 산출해 낸다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

구간 합 공식의 활용 - 2차원 배열

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

2차원 배열의 구간 합을 구할 땐 전혀 다른 공식이 유도된다. 위와 같은 2차원 배열에서 빨간색으로 표시한 구간의 합을 구한다고 생각해 보자.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | V |

일단 합 배열의 V 표시한 지점에 해당하는 빨간색 면적을 추출해야 한다.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | X |  | V |

그리고 먼저 X 표시한 지점에 해당하는 면적을 제거해야 한다.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  | X |
|  |  |  |  |
|  |  |  | V |

다음으로 위 그림에서 X 표시한 지점에 해당하는 면적을 제거해야 한다.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | O |  | X |
|  |  |  |  |
|  | X |  | V |

이렇게 두 면적을 제거하면 O 표시한 지점에 해당하는 파란색의 면적이 두 번 제거된다. 그러므로 마지막으로는 O 표시한 지점에 해당하는 면적을 추가해야 한다.

따라서, {(r1, c1)부터 ~ (r2, c2)까지의 구간 합} = {(r2, c2)까지의 구간 합} – {(r2, c1 – 1)까지의 구간 합} – {(r1 – 1, c2)까지의 구간 합} + {(r1 – 1, c1 – 1)까지의 구간 합}이라는 2차원 배열의 구간 합 공식이 유도된다.



**구간 합 공식의 활용 – 나머지 합**

다시 1차원 배열의 구간 합 공식을 활용하는 문제로 넘어왔다. 그러나 어떤 조건을 만족하는 모든 구간을 구하라는 조건이 붙었다. 배열의 최대 길이는 106이기 때문에 만약 이 문제를 O(N2)의 시간 복잡도로 해결하려 한다면 시간 제한을 위반할 것이다. 따라서 합 배열을 유도한 뒤 모든 경우의 수를 탐색하지 않고 메모이제이션 기법을 활용해야 한다.

우선 두 가지 경우의 수를 생각해볼 수 있다.

1. 합 배열의 원소 자체(인덱스 0부터 i(i > 0)까지의 합)가 M으로 나누어 떨어지는 경우
2. 0이 아닌 인덱스 i부터 j(j > i)까지의 합(S[j] – S[i - 1])이 M으로 나누어 떨어지는 경우

모듈러 연산의 성질에 의해 합 배열에서 M으로 나눴을 때의 나머지가 같은 원소끼리 빼면 M으로 나눴을 때의 나머지가 0이 된다. 이 두 원소의 인덱스를 각각 i, j라고 하면 (S[j] – S[i]) % M == 0이 성립하고, 이것은 i + 1부터 j까지의 구간 합과 같다.

첫 번째 경우의 수는 단순히 합 배열 자체에서 M으로 나눈 나머지가 0인 원소를 셈으로써 구할 수 있다.

두 번째 경우의 수는 0 ~ M – 1의 인덱스 범위를 갖는 별도의 카운팅 배열을 정의하여, 각각의 인덱스를 M으로 나눴을 때의 나머지로 생각하면 구할 수 있다. 같은 나머지를 갖는 원소가 2개 이상이라면 Combination을 활용하여 원소들 중 2개를 선택하는 경우의 수를 구하면 된다. 각각의 원소는 합 배열에서 M으로 나누었을 때 나머지가 같은 인덱스를 의미하므로, 두 원소를 i, j(i < j)라고 할 때 매번 S[j] – S[i]를 적용하여 i + 1 ~ j의 구간 합을 셌다고 가정하는 것이다.

**03-3. 투 포인터**

**개요:** 2개의 포인터로 알고리즘의 시간 복잡도를 최적화한다.