**REPORT**

로고이(가) 표시된 사진

자동 생성된 설명

**과목명 : 자료구조 심화학습**

**과제내용 : 1. Dynamic Programming의 개념**

**2. 피보나치 알고리즘 (분할정복, DP 비교)**

**3. DP 알고리즘 예시**

**교수명 : 심종익 교수님**

**학 년 : 3학년**

**학 번 : 201901366**

**성 명 : 서희준**

**제출일 : 23.07.30.(일)**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**<목차>**

1. **Dynamic programming 개념**
2. **피보나치 알고리즘 ( 분할정복과 DP 비교)**
3. **Dynamic Programming 예제 프로그램**
4. **Dynamic Programming ( 동적 계획법 ) 의 개념**

다이나믹 프로그래밍은 특정 문제를 해결하는데 있어서, 최적화하는 기법으로 시간 복잡도 ‘다항 시간’에 문제를 해결할 수 있는 방법이다. ‘지수 복잡도’를 가지는 무차별 대입 방법과 비교했을 때 훨씬 좋은 성능을 가진다. 특히 중복으로 호출되는 재귀함수나, 반복적으로 실행되는 명령문에는 동적 계획법을 적용하면 큰 이점을 가져올 수 있다. 문제의 크기에 따라 빠른 연산 속도와 큰 메모리 공간을 요구할 수 있는데, 이러한 요구사항을 최대한 효율적인 방법으로 해결한다.

다이나믹 프로그래밍은 큰 문제를 작은 문제로 나누어 푸는 문제에서 한 번 계산한 부분을 다시 계산하지 않도록 하는 최적화 기법으로, ‘동적 계획법’으로도 불린다. 따라서, 분할 과정 시 중복되는 부분이 있는 때 동적 계획법을 문제에 적용할 수 있다. 예를 들어, 분할 정복 알고리즘을 사용해 큰 문제를 더 작은 부분 문제로 분할하여 해결하려고 할 때 그 과정을 재귀함수로 구현하면서 중복 호출되는 재귀함수가 있을 수 있다. 이는 공간적인 측면뿐만 아니라, 시간적인 측면에서도 손해이다. 따라서, 중복되는 하위 문제를 먼저 계산한 재귀함수에서 저장해두고, 같은 재귀함수가 호출되었을 때 저장했던 값을 사용하는 방식으로 알고리즘을 최적화한다. 즉, 동적 계획법의 목적은 계산 과정을 최적화해 효율적인 시간, 공간의 이점을 가져오는 것이다.

이러한 동적 계획법은 두가지 방법으로 알고리즘을 최적화할 수 있다. 첫번째로는 탑다운(메모이제이션)방식으로 큰 문제를 해결하기 위해 작은 문제를 호출하는 방식이다. 큰 문제에서 하향식으로 작은 문제로 이어지기 때문에 점화식을 이해하기 쉽다는 장점이 있다. 주로 재귀적인 방식으로, 큰 문제를 분할한다. 여기서 메모제이션은 이전에 계산한 값을 일시적으로 저장하는 넓은 개념의 의미이다. 이전에 계산된 값은 저장만 되고, 동적 계획법에 사용되지 않을 수 있기 때문에 동적 계획법과 의미적인 차이가 있다. 즉, 메모이제이션은 동적 계획법의 핵심이 되는 기술이다. 두번째로는 바텀업(타뷸레이션)방식으로 가장 하위 문제에서 문제의 해를 계산하며 가장 상위 문제의 해를 찾는 방식이다. 하향식의 탑다운과 반대로, 상향식 풀이과정이다. 재귀적 방식을 사용하지 않기 때문에 함수 호출로 인한 시간과 메로리 사용량을 줄일 수 있다는 장점이 있다.

다음은 분할 정복과 동적계획법을 비교하며, 하향식과 상향식으로 알고리즘을 최적화한다. 이 두가지 방식을 피보나치 수열을 구하는 알고리즘에 적요해 설명한다. 처음은 탑다운 방식이다.

1. **피보나치 알고리즘 비교 ( 분할정복 : 동적 계획법 )**

**텍스트, 폰트, 친필, 번호이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 폰트, 라인, 도표이(가) 표시된 사진

자동 생성된 설명**

피보나치 수열은 전 항과 전전 항을 합해서 만드는 수의 열이다.

1. **분할 정복**

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**<결과>**

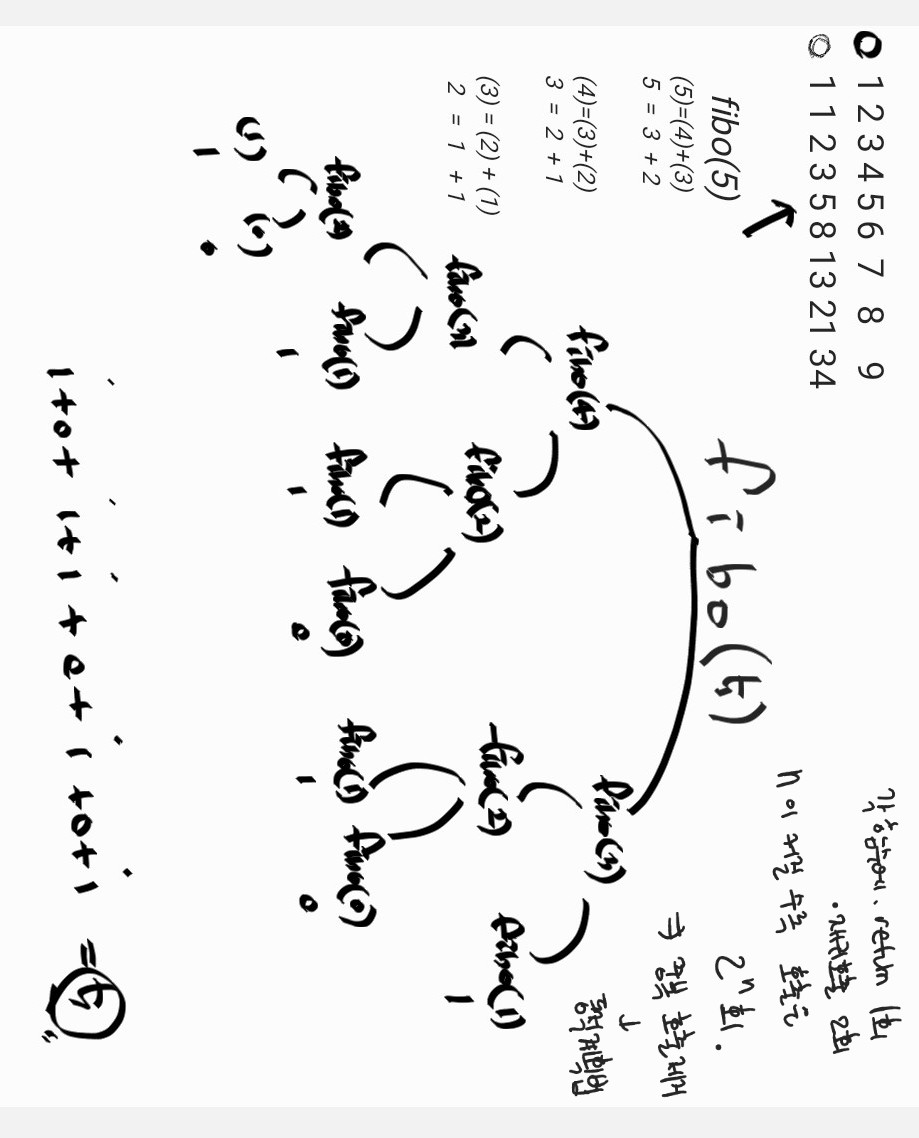
**텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명**

다음은 피보나치 수열을 수학 점화식을 분할 정복 알고리즘으로 나타낸 것이다. 수열의 알고 싶은 항의 위치를 입력하면, 해당 항의 수를 반환하는 알고리즘이다. 메인 함수는 다음과 같이, 위치를 입력받고 함수를 호출하여, 값을 출력한다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

****

위 그림은 피보나치 수열을 분할 정복 알고리즘으로 구현했을 때 도식화한 것이다. 수열의 위치를 입력하면, 해당 위치의 수를 반환하는 알고리즘이다. 메인 함수는 다음과 피보나치 수열의 기본적인 원래는 an = an-1 + an-2 이다. 따라서 a5 = a4 + a3 라는 것을 쉽게 알 수 있다. 이것은 또 a4 = a3 + a2  , a3 = a2 + a1 로 볼 수 있다. 이러한 방식으로 계속 수열의 위치에 대한 값을 분할하며, a1 =1 이라는 것을 이용해 그 합을 전체 해로 반환한다. 문제를 분할하는 과정에서 알 수 있지만, 재귀함수가 한번 호출될 때마다 두개의 또 다른 재귀함수를 호출하기 때문에 시간 복잡도는 O(2n)이 된다. 여기서 비효율적인 부분은 같은 값을 가지는 재귀 함수를 중복으로 호출한다는 것이다. a4 = a3 + a2  , a3 = a2 + a1 부분만 보더라도, 겹치는 부분이 많다는 것을 알 수 있다. 이러한 분할 정복은 오히려 문제해결의 효율을 떨어트릴 수 있다. 이 문제는 동적 계획법과 반복문을 이용하면, 성능을 계선할 수 있다.

1. **동적 계획법**
2. **Memoization(top-down)**

이전에 구한 결과를 중간 중간에 메모리 공간에 저장한다. 이후 계산 과정에서 같은 식을 호출하면, 저장했던 결과를 그대로 가져와 중복된 부분을 최소화한다. 값을 중간에 저장한다는 측면에서 캐싱이라고도 한다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위 코드에서 볼 수 있다시피, 가장 큰 문제인 fiboTopDown(6)에서부터 fiboTopDown(5), fiboTopDown(4)로 분할하며 작은 문제의 해를 구한다. 이때, 동적계획법을 적용해, 중간의 계산한 값을 memoization에 저장한다. 이후 다시 재귀함수가 호출 되었을 때, 해당되는 항이 memoization에 저장되어 있으면 저장된 값을 사용한다. 중복된 값을 대신 반환하면, 해당 재귀함수에서 파생되는 더 이상의 재귀함수는 없기 때문에 시간 복잡도가 개선된다.

아래는 위 알고리즘의 도식이다.

원, 도표, 그림, 디자인이(가) 표시된 사진

자동 생성된 설명

fiboTopDown(6) -> fiboTopDown(5) -> fiboTopDown(4) -> fiboTopDown(3) -> fiboTopDown(2) -> fiboTopDown(1) -> fiboTopDown(2) 의 순서를 거쳐 8이라는 값을 반환하는 것이다. 동적 계획법을 피보나치 알고리즘에 적용했을 때 시간 복잡도는 O(n)이다. 이는 분할 정복의 O(2n)에 비해 상당히 개선된 성능을 보인다. 한번 구한 결과를 재사용함으로써, 추가적인 재귀함수 호출이 발생하지 않기 때문이다.

1. **Tabulation(bottom-top)**

상향식 방식은 앞서 설명한것과 같이 재귀 호출을 하지 않기 때문에 시간과 메모리 사용량을 줄이는 방법이다. 반복문으로 알고리즘을 구현한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

DP[6] 를 구하기 위해서는 DP[0], DP[1], DP[2] ... DP[6]의 순서대로 값을 구하고, 계산된 값을 저장한다. 이때 중복된 계산을 방지하기 위해 계산된 값을 토대로 다음 값을 빠르게 구해나가는 bottom - up 형태를 취한다.

1. **동적 프로그래밍 (동적 계획법) 예제 프로그램 : 막대 자르기 ( cutting the rod )**

이 문제는 주어진 길이를 가진 막대기를 여러 개의 작은 막대기로 자르는 최적의 방법을 찾는 프로그램이다. 막대기를 자르는 각각의 조각은 정해진 가격으로 판매되며, 막대기의 최대 가치(최대 수익)를 얻을 수 있는 자르기 방법을 찾는 것이 목적이다. 예를 들어, 길이가 8인 막대기가 주어졌고 다음과 같은 가격표가 있다고 가정하자.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

이 경우, 막대기를 최적으로 자르는 방법은 길이가 2인 막대기 4개로 자른다. 즉, 길이가 2인 막대기 4개를 판매하여 최대 가치 20을 얻을 수 있다. 이처럼 "막대기 자르기" 문제를 동적 프로그래밍을 활용하여 각 길이별 최대 가치를 계산하고, 최종적으로 주어진 길이의 최대 가치를 찾아낸다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

또는 가격이 위에 표와 같다면, 길이가 8인 막대를 각 각 1로 잘라야 24라는 최대 가치를 만들 수 있다.

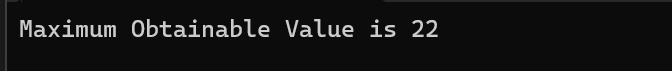
막대 자르기를 동적 계획법으로 구현할 때 bottom-up 상향식으로 문제를 해결한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

메인 함수에서 “int arr[] = {1, 5, 8, 9, 10, 17, 17, 20};”와 같이 주어진 막대기 조각의 가격 정보를 배열 arr[]에 저장한다. int size = sizeof(arr) / sizeof(arr[0]);: 배열 arr[]의 길이를 계산하여 cutRod()함수로 전달하는데, 여기서의 size가 주어진 막대의 길이이다. 여기서는 8의 길이를 가지고 있다. 이후 cutRod() 함수를 호출하여 주어진 가격 정보를 기반으로 최대 가격을 계산하고 출력한다.

**<결과>**



텍스트, 스크린샷, 폰트, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1의 길이를 가진 막대 가격을 3으로 변경하면, 앞서 설명한 바와 같이 8의 길이를 모두 1의 길로 나누는 것이 효율적이므로, 24라는 가치를 가지게 될 것이다.

**<결과>**

텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

동적 계획법을 적용한 막대 자르기 문제는 막대 길이가 1일때부터 상향식으로 최대 가치를 구한다. 예를 들어, 막대길이가 1일때의 최대가치가 3이라고 가정한다. 2의 막대길이를 구할때는 1과 1로 나누거나 나누지 않는 2와 0의 경우의 수가 있다. 이때 막대 길이가 1의 최대 가치는 이전에 3으로 계산하고, 저장하였기 때문에 저장된 값을 사용하면 중복된 계산을 피할 수 있다. 2을 “1” 두개로 나눴을 때 3+3이라는 값을 가지고, 2를 2와 0으로 나눴을 때 5라는 값을 가진다. 2라는 막대 길이는 1과 1로 나눈 것이 더 큰 가격으로 매겨지기 때문에 “2”크기의 막대 길이의 최대 가격을 6으로 저장한다. 이후 3에 대한 최대 길이를 찾는 경우의 수 중 2 와 1로 3을 나눌 때 2에 대한 최대 값이 이미 저장되어 있기 때문에 3과 0으로 나눈 가격과 바로 비교가 가능한다. 이런식의 상향식 동적 계획법을 적용한 알고리즘이 cutRod()함수이다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

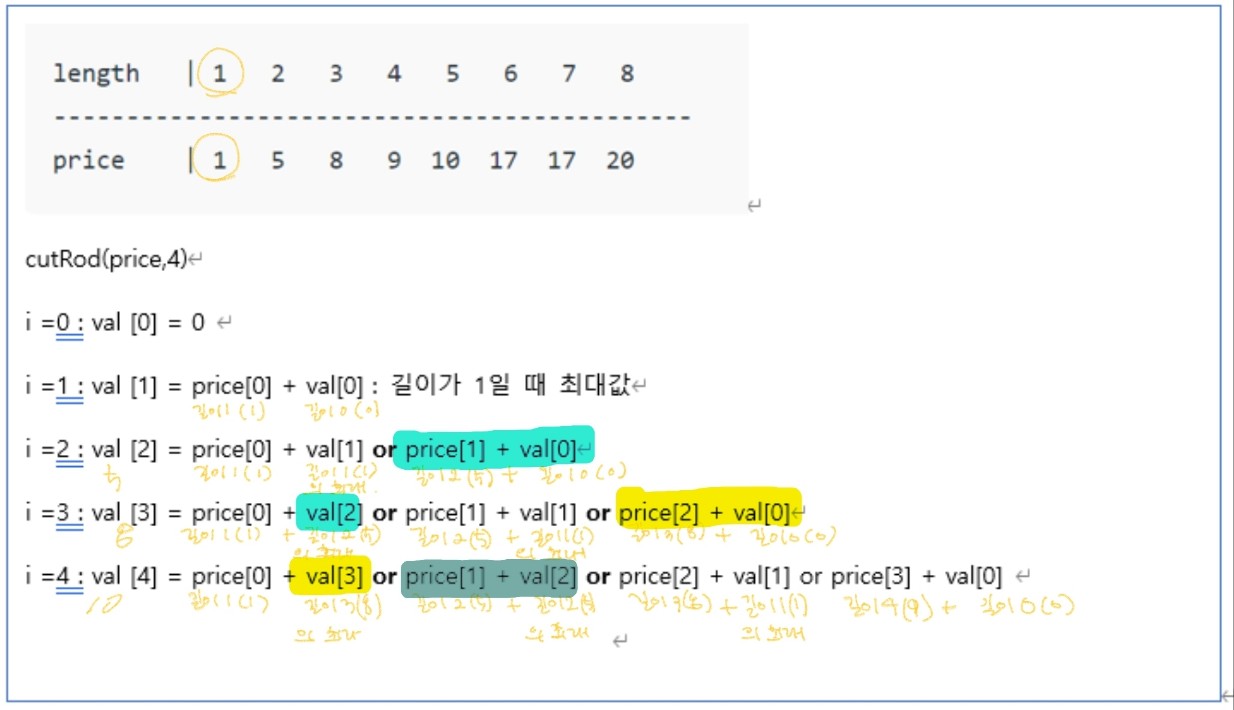
자동 생성된 설명

먼저 각 막대 길이에 대한 최대 값을 저장하기 위해 malloc동적할당을 이용한다. 이는 에러처리를 포함하여 define매크로로 정의했다. 이때 n+1의 크기를 가지는 이유는 val[0]= 0을 포함해야 하기 때문이다. 길이가 0인 막대기는 길이가 없기 때문에 가격을 0으로 매긴다. 예를 들어 3의 최대 가격을 구할 때 나누지 않는 경우 즉, 3과 0으로 나누는 경우로 인해 val[0]= 0을 포함해야 한다.

Line 38 : 길이 1부터 n까지의 막대기에 대해 최대 가격을 계산하기 위해 반복문을 실행한다. Bottom-up방식으로 재귀함수를 이용하는 방식과 달리 시간과 메모리 공간을 절약할 수 있다. 반목문 내에서 특정 막대 길이에 대한 최대 가격을 구할 때 int max\_val = -1 과 같이 max\_val을 -1로 초기화한다. 이때 val[0]과 0의 값을 가지기 때문에 이후에 비교할 때 처음 들어오는 값이 무조건 최대 값이 되게 하기 위함이다.

Line 42 : max\_val = max(max\_val, price[j] + val[i - j - 1]); 길이 i인 막대기를 자를 때, 조각의 길이가 j인 경우의 가격을 계산한다. price[j]는 조각의 가격이고, val[i - j - 1]은 남은 길이 (i - j - 1)에 대한 최대 가격을 의미한다. 따라서 price[j] + val[i - j - 1]은 j길이의 조각으로 나눌 때 얻을 수 있는 가격이다. 이후 val[i] = max\_val; 과 같이 길이 i인 막대기에 대한 최대 가격을 max\_val로 저장한다.

반복문이 모두 종료되면, 동적으로 할당 받은 공간을 반환하고, 길이 n인 막대기에 대한 최대 가격을 메인함수로 반환한다.



텍스트, 시계, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



**<참고자료(출처)>**

1. Complete Tutorial on Dynamic Programming (DP) Algorithm. (16, Jun, 2023). Retrieved from

https://www.geeksforgeeks.org/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/.

1. [알고리즘] 다이나믹 프로그래밍 (Dynamic Programming) . (2021).

https://velog.io/@kimdukbae/%EB%8B%A4%EC%9D%B4%EB%82%98%EB%AF%B9-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-Dynamic-Programming.