**Computer Algorithm**

**Chapter 11**

**Dynamic Programming**

**[실습] 동적계획법 (Dynamic Programming)**

**실습 목표**

* **동적계획법을 이해한다.**
* **동적계획법을 활용하여 주어진 문제를 작은 문제로 분할한 후 문제들 간의 관계를 정리하고 이를 통해 작은 문제를 해결하고 그 결과를 활용하여 큰 결과를 해결하는 방식으로 최종적인 문제의 답을 구한다.**

**요구사항**

**실습과제 1) (100점)**

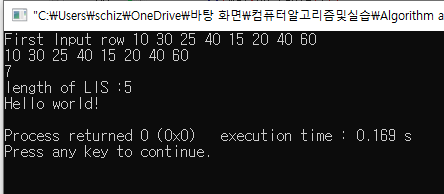
**1-1. (30점)**

주어진 수열에서 수열의 각 값들의 순서를 바꾸지 않은 상태에서 숫자들이 계속 증가하는 모습을 보여주는 부분수열을 추출하려 한다 있다. 이 경우 최대 길이를 가지는 부분수열의 길이를 출력하라.

입력값은 아래와 같이 형식의 파일(input.txt)로 주어지며 입력 파일 내의 숫자들은 공백으로 구분된다. 출력 형식은 아래와 같다.

입력 : 10 30 25 40 15 20 40 60

출력 : 5



**Hint)**

위 예시에서 입력 수열의 순서를 바꾸지 않으며 숫자가 계속 증가하는 모습의 부분수열 종류는 매우 많으며 그중 일부를 나타내면 다음과 같다.

10 25 40 60

30 40 60

10 15 20 60

…

그 중에 가장 긴 길이의 숫자가 증가하는 모습의 부분수열은 아래와 같다.

10 15 20 40 60

따라서 출력값은 5가 된다.

이를 풀기 위해서는 먼저 문제를 세부 문제들로 분할하고 분할된 문제들과 원 문제와의 관계를 도출해야 한다.

수열 의 i번째 숫자까지의 최대 부분수열의 길이를 라고 하고 i보다 작은 임의의 수 j에 대하여 j번째까지 수열에서의 최대 부분수열의 길이를 라고 가정할 경우 와 간에는 다음과 같은 관계가 성립한다.

위 식은 i보다 작은 숫자 j에 대해 A[i]가 A[j]의 값보다 큰 경우의 모든 j에 대한 dp(j) 에 1을 더한 값 중 가장 큰 값을 dp(i)로 한다는 의미이다. 위 식은 만약 i보다 작은 임의의 숫자 j에 대하여 A[i]의 값이 A[j] 값보다 작다면 해당 하는 j에 대한 dp(j)는 dp(i) 계산에 사용하지 않도록 한다는 의미도 포함한다.

예를 들면 위 예제의 수열에서 로 계산된다. 다른 5보다 작은 임의의 수 j에 대하여 를 계산에 포함하지 않은 이유는 A[5] = 20인데 A[0]과 A[4]를 제외하면 다 A[5]보다 큰 값을 가지기 때문이다.

위 식의 j값은 항상 i보다 작은 값을 가진다는 특성을 활용하여 는 반복문을 통하여 구하도록 한다. 또한 수열 A에 대하여 의 초기값은 1로 설정하는 것도 계산을 위해 필요하다.

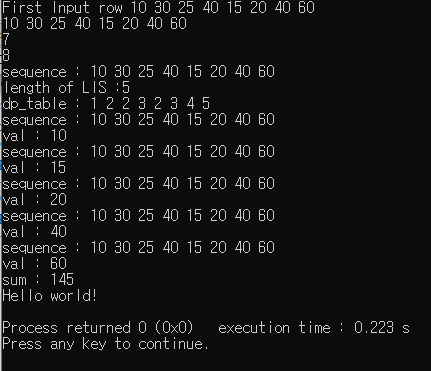
**1-2. (40점)**

주어진 수열에서 수열의 각 값들의 순서를 바꾸지 않은 상태에서 숫자들이 계속 증가하는 모습을 보여주는 부분수열을 추출할 수 있다. 이 경우 최대 길이를 가지는 부분수열의 수열의 합을 출력하라.

입력값은 아래와 같이 형식의 파일(input.txt)로 주어지며 입력 파일 내의 숫자들은 공백으로 구분된다. 출력 형식은 아래와 같다.

입력 : 10 30 25 40 15 20 40 60

출력 : 145



**Hint)** 1-1 예제의 구현 결과를 활용하여 구현한다. 여기서 최대 길이를 가지는 부분수열의 길이 를 구하는 계산식은 1-1 예제와 같으나 여기에 부분수열의 합을 구할 수 있는 계산식을 추가하여 구현을 완료하도록 한다.

**1-3. (30점)**

두 문자열의 차이를 구하는 방법은 다음과 같은 최적 부분구조(optimal substructure)로 정의할 수 있다.

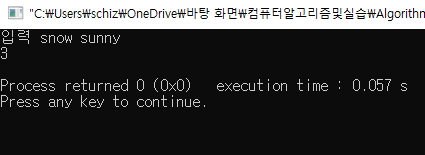
여기서 는 문자열 의 번째까지의 부분 문자열과 문자열 의 번째까지의 부분 문자열 간의 차이를 의미한다.

위 식을 활용하여 두 문자열의 차이 (edit distance)를 구하여라.

입력 값은 두 문자열이며 출력 값은 두 문자의 거리를 의미한다.

입력: snow sunny

출력: 3



**배경지식**

동적계획법(Dynamic Programming)은 큰 문제를 여러개의 작은 문제들로 쪼갠 후, 이 작은 문제들을 해결하여 이 결과들을 통해 궁극적으로 큰 문제를 해결해 가는 방식을 의미한다.

아래는 동적계획법과 재귀를 활용하여 피보나치 수열을 만든 예이다.

재귀를 활용한 예:

int fib(int n)

{

**if** (n <= 1)

return n;

return fib(n-1) + fib(n-2)

}

재귀는 재귀호출은 기본적으로 시스템에 대한 부하가 크며 또는 위 방법은 같은 함수를 여러 차례 호출하는 단점을 보여 시스템의 낭비가 예상되므로 동적계획법을 사용하여 알고리즘의 실행 성능을 높일 필요가 있다.

동적계획법은 세부문제를 해결함에 따라 큰 문제를 해결하도록 하는 방법으로 최적 부분구조(Optimal Substructure)라고 하는 우선 큰 문제와 세부문제와의 관계를 설정한다. 최적 부분구조는 어떤 문제의 해결책을 작은 문제의 정답으로부터 설계되는 구조를 말한다. 예를 들면 피보나치 수열은 다음과 같이 정의되는 데,

여기서 은 과 의 값을 통해 구하고 있음를 알 수 있다. 은 보다 큰 매개변수에 대한 값을 가지므로 큰 문제와 세부문제와의 관계로 보는데 무방하고 따라서 위 식은 최적 부분구조라고 볼 수 있다.

어떤 문제에 대하여 최적 부분구조를 얻었다면 이를 통해 동적계획법을 구현할 수 있다. 아래는 동적계획법으로 구현한 예시코드이다.

fib[0] = 0;

fib[1] = 1;

for(i = 2; i <= n; i++) {

fib[n] = fib[n-1] + fib[n-2]

}

return fib[n]

동적계획법은 주어진 문제를 풀 때 이용했던 결과값들을 저장해두고 다음계산에 이용하기 때문에 속도를 더 향상시킬 수 있다. 5번째 피보나치 수를 구한다고 가정했을 때, 재귀함수를 이용하게되면 fib(2) + fib(3) => 연산에서 fib(3)을 할때 다시 fib(2)를 계산하게되지만 동적계획법은 이전에 값들을 이용하여 축적되기때문에 연산이 더 빠르다.

동적계획법을 구현하기 위해서는 큰 문제와 상대적으로 작은 규모의 문제들간의 관계를 정의해야 하는데, 이부분만 해결되면 나머지는 반복문을 사용하여 구현을 할 수 있으므로 쉽게 구현될 수 있다.

**제출방법**

* 보고서 작성방법: 실습문제 번호별로 결과가 나온 화면의 내용을

캡쳐하여 보고서에 붙여 놓는다.

* 소스코드의 파일이름에 연습문제 번호를 붙이는 것을 잊지 않는다. 예) ex-1.c, ex-2.c
* 결과 보고서에 이름과 작성 날짜를 기입하는 것을 잊지 않는다. 예) 김웅섭\_2020\_09\_01.doc
* 실행결과를 보고서에 작성하여 소스코드와 함께 제출한다.
* 제출 마감 : e-class 제출 마감시간까지