C++ for Coders and Data Structures

Lecture Notes by idebtor@gmail.com, Handong Global University

본 PSet 은 저의 강의 경험과 학생들의 의견 및 Stanford CS106 과 Harvard CS50 같은 강의에서 수집된 자료를 토대로 작성되었습니다.
본 PSet 에 문제점이나 질문 혹은 의견이 있다면, 저의 이메일(idebtor@gmail.com)로 알려 주시면 강의 개선에 많은 도움이 되겠습니다.

Note: 과제 요구사항은 마지막 3 페이지에 해당합니다.

점화 관계

점화 관계는 값의 시퀀스 또는 다차원 배열을 재귀적으로 정의하는 식이다. 하나 이상의 초기 항이 주어지면: 각 원소의 이후 원소는 이전 원소들의 함수로 표현된다.

점화 관계를 해결하는데 여러가지 방법이 존재한다.
"*telescoping*" 방법을 사용한다. 이 방법은 아래와 같이 흘러간다.

- 점화 관계식을 더 작은 n 값으로 푼다. 새 식의 왼편은 이전 식의 우편과 같고, 초기 조건이 구해질 때까지 반복한다.
- 관계식의 좌편과 우편을 더한다. 양쪽에 같은 항들을 제거한다.
- 우편에 남은 항들을 더한다. 그 결과가 시퀀스의 일반 공식이다.

점화 관계의 **일반적인** 해결 방법은 "*unfolding*"이다. 초기 조건에 도달할 때까지, 반복해서 재귀적으로 교체한다.

유용한 수식:

1 + 2 + 3 + … + N = N(N+1)/2
1 + 2 + 4 + 8 + … + 2^n = 2^{n+1} – 1

Example 1:

선형 탐색의 시간 복잡도는..

$T(0) = c_0$ **// 하나인 배열의 시간 복잡도는 1 이다.**
$T(n) = T(n-1) + c$ **// n 개의 원소의 시간 복잡도는 (n-1)개의 원소 + 1 개의 원소의 시간 복잡도와 같다.**

n 개의 원소를 탐색하는 비용을 1 개의 원소를 탐색하는데 사용하는 비용, 더하기 n – 1 개의 원소를 탐색하는데 사용하는 비용이다. 상수 c_0와 c는 1 로 정의한다. 식은 아래와 같다:

$$T(0) = 1$$
$$T(n) = T(n-1) + 1$$

Telescoping:
T(n) = T(n – 1) + 1
T(n – 1) = T(n – 2) + 1
T(n – 2) = T(n – 3) + 1
…
T(1) = T(0) + 1
왼편과 우편을 더한다:
T(n) + T(n – 1) + T(n – 2) + … + T(1) = T(n – 1) + T(n – 2) + … + T(0) + 1 + 1 + … + 1
양쪽에 같은 항들을 제거한다:
T(n) = T(0) + 1 + 1 + … + 1

n 이 몇개 남아 있나요**?**

$T(n) = T(0) + n$

$= 1 + n$

그리하여서**, T(n) is O(n).**

Unfolding:

$$
\begin{aligned}
T(n) &= T(n-1) + 1 \\
&= T(n-2) + 2 \\
&= T(n-3) + 3 \\
&\dots \\
&= T(n-n) + n = T(0) + n \\
T(n) &= 1 + n \\
T(n) &= O(n)
\end{aligned}
$$

Example 2:

다음 시퀀스: 1, 3, 6, 10, 15, 21, 28, 36, ... 각 항은 이전 항과 새로운 항의 위치를 더한 값과 같다. 예) 10 은 4 번째 위치에 있고 이전 항은 6 이다. 10 = 6 + 4. 이 시퀀스의 일반 공식을 구한다.

$a_1 = 1$

$a_n = a_n\text{-}1 + n$, $n = 2, 3, 4, \dots$

Telescoping:

$a_n = a_{n-1} + n$

$a_{n-1} = a_{n-2} + (n-1)$

$a_{n-2} = a_{n-3} + (n-2)$

....

$a_4 = a_3 + 4$

$a_3 = a_2 + 3$

$a_2 = a_1 + 2$

왼편과 우편을 더한다:

$a_n + a_{n-1} + a_{n-2} + a_{n-3} + \dots + a_4 + a_3 + a_2 =$

$a_{n-1} + a_{n-2} + a_{n-3} + \dots + a_4 + a_3 + a_2 + a_1 + 2 + 3 + 4 + \dots + (n-1) + n$

양쪽에 같은 항들을 제거한다:

$a_n = a_1 + 2 + 3 + 4 + \dots + (n-1) + n$

초기 조건에 주어진 값으로 대체한다:

$a_n = 1 + 2 + 3 + 4 + \dots + (n-1) + n = n(n+1)/2$ (open form)

그리하여서 해당 시퀀스의 각 항의 일반 공식을 구했다:

$a_n = n(n+1)/2$ (closed form)

Example 3: findmax()

재귀적으로 배열의 가장 큰 원소를 찾는다.

알고리즘:

If 배열이 원소 하나만 포함하는 경우,

이 원소는 가장 큰 원소다.

Else,

a. 첫 n – 1 개의 원소를 포함한 배열의 가장 큰 원소를 찾는다.

b. n 번째 원소와 비교하여 더 큰 원소를 반환한다.

```
int findmax(int [] array, int size) {
    if (size == 1)
            return array[0];
    else {
            int max = findmax(array, size-1)
            if (max >= array[size-1])
                return max;
            else
                return array[size-1];
    }
}
```

다음과 같이 정의한다.

- N 의 크기를 가진 배열의 가장 큰 원소를 찾는데 걸리는 시간은, N – 1 의 크기를 가진 배열 더하기 비교하는 부분과 같다.
- 비교하는 부분의 시간 복잡도는 배열의 크기와 상관이 없으므로 상수로 여겨지고, 1 을 사용한다.
- 배열의 크기가 1 일 때 걸리는 시간은 상수이다 (동일하게 1 이 사용된다).

그리하여서, 다음과 같은 점화 관계식을 구한다.

T(1) = 1 // **1 의 크기를 가진 배열의 시간 복잡도는 상수 1 이다.**
T(N) = T(N-1) + 1

Telescoping:
T(N) = T(N-1) + 1
T(N-1) = T(N-2) + 1
T(N-2) = T(N-3) + 1
….
T(3) = T(2) +1
T(2) = T(1) + 1
T(1) = 1

왼편과 우편을 더하고, 같은 항들을 제거한다:
T(N) = 1 + 1 + … + 1 (Open form)

Telescoping 울 한 결과는 N 개의 원소를 가진다, 즉 1 이 N 개 존재한다:
T(N) = N (Closed form)
그리하여서, T(N) = O(n)

Notice: 여기서부터 과제가 시작된다. 다음의 페이지들을 스크린 캡처하여 제출한다.

Problem 1 – 삽입 정렬

점화 관계: N 개의 원소로 이루어진 배열을 정렬하는데 걸리는 시간은 N – 1 개의 원소를 가진 배열을 정렬하는데 걸리는 시간과, N – 1 번의 비교를 하는데 걸리는 시간의 합이다. 초기 조건: 1 개의 원소를 가진 배열을 정렬하는데 걸리는 시간은 상수 1 이다.

T(1) = 1

T(N) = T(N-1) + N-1

Next **telescoping** 을 해봅시다: 다음 조건들을 위해 새로운 점화 관계식을 작성한다: N-1, N-2, …, 2

T(N) = T(N-1) + N-1

T(N-1) = ______________________

T(N-2) = ______________________

……

T(2) = ______________________

Next 위에서 구한 식의 왼편과 우편을 더한다:

T(N) + T(N-1) + T(N-2) + T(N-3) + …. T(3) + T(2) =
T(N-1) + T(N-2) + T(N-3) + …. T(3) + T(2) + T(1) + ____

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(N) = T(1) + __________________________ (Open form)

T(N) = 1 + ____________________________ (Closed form)

그리하여서, 삽입 정렬의 시간 복잡도는:

T(N) = _______________________ (big O)

Problem 2

T(1) = 1

T(N) = T(N – 1) + 2 // 2 는 c 와 같은 상수이다

Telescoping:

T(N) = T(N-1) + 2

T(N-1) = __________________

T(N-2) = __________________

……

T(2) = ___________________

Next 위에서 구한 식의 왼편과 우편을 더한다:

T(N) + T(N-1) + ____ =

T(N-1) + ____

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(N) = T(1) + ____________________ (open form)

T(N) = 1 + ______________________ (closed form)

그리하여서, 큐 (queue)를 뒤집는 시간 복잡도는:

T(N) = ______________ (Big O)

Problem 3 - Power()

```
long power(long x,long n) {
  if(n==0)
    return1;
  else
    return x * power(x,n-1);
}
```

T(n) = n 크기의 문제를 해결하는데 걸리는 시간.
점화 관계는 재귀 프로그램들의 시간 복잡도를 구하는데 사용된다 – 점화 관계 그들 자신도 재귀다.

T(0) = 크기가 0 인 문제를 해결하는데 걸리는 시간.

- 초기 조건

T(n) = 크기가 n 인 문제를 해결하는데 걸리는 시간.

- 재귀 조건

T(0) = 1

T(n) = T(n – 1) + 1 // +1 은 상수다.

Telescoping 을 사용한 풀이:

T(n – 1)을 알면, T(n)을 풀 수 있다.

T(n) = T(n – 1) + 1

T(n – 1) = ____________________

T(n – 2) = ___________________

….

T(2) = __________________

T(1) = __________________

Next 위에서 구한 식의 왼편과 우편을 더한다:

T(n) +

마지막으로, 같은 항들을 제거하고 우편에 합을 간소하게 만든다:

T(n) = ________________ (Open form)

T(n) = ________________ (Closed form)

T(n) = _________________(Big O)

Problem 4 - Power()

```
long power(long x,long n) {
  if (n == 0) return 1;
  if (n == 1) return x;
  if ((n % 2) == 0)
   return power(x * x, n/2);
  else
   return power(x * x, n/2) * x;
}
```

T(0) = 1

T(1) = 1

T(n) = T(n / 2) + 1 // n 은 2 의 제곱 수이고, +1 은 상수다.

Unfolding 을 사용한 풀이:

T(0) = 1

T(1) = 1

T(n) = T(n/2) + 1

= ____________________ since T(n/2) = T(n/4) + 1

= ____________________ since T(n/4) = T(n/8) + 1

= _____________________

….

= _____________________ n, 2^k, k 으로 표현.

T($n/2^k$) 을 제거 하기를 원한다.

T(1) 에 도달하였을 때 해결을 한다.

$n/2^k = 1$

$n = 2^k$

log n = k

T(n) = _________________ (Open form) n, 2^k, k 으로 표현.

= _________________ (Open form)

= _________________ (Closed form)

그리하여서, T(n) = __________________ (Big O)

제출할 파일들

Piazza 폴더에 현재 파일의 마지막 3 페이지의 스크린 캡쳐를 제출한다.

제출 마감

11:55 PM

One thing I know, I was blind but now I see. John 9:25