[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.09.10 목요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

- 본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.
- 본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

Algorithm Design Example

- **Maximum Subsequence Sum (MSS) Problem**

  Given N (possibly negative) integers $A_0, A_1, \cdots, A_{N-1}$, find the maximum value of $\sum_{k=i}^{j} A_k$ for $0 \le i \le j \le N-1$. (For convenience, the maximum subsequence sum is 0 if all the integers are negative.)

- Example
  - (-2, 11, -4, 13, -5, -2) → MSS = 20

Maximum Subarray Problem
Maximum Positive Sum Subarray Problem

Figure 2.2 Running times of several algorithms for maximum subsequence sum (in seconds)

Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	$N = 10$	0.00103	0.00045	0.00066	0.00034
	$N = 100$	0.47015	0.01112	0.00486	0.00063
	$N = 1{,}000$	448.77	1.1233	0.05843	0.00333
	$N = 10{,}000$	NA	111.13	0.68631	0.03042
	$N = 100{,}000$	NA	NA	8.0113	0.29832

최대 부분 수열의 합 문제

길이 n인 정수의 수열 $a_0, a_1, a_2, \cdots, a_{n-1}$이 입력으로 주어져 있다.
여기서 부분 수열 $[i, j]$라는 것은 $a_i, a_{i+1}, a_{i+2}, \cdots, a_j$ 를 말한다.
본 문제는 주어진 수열의 부분 수열의 합, 즉 $\sum_{i \le k \le j} a_k$ 의 최대값을 구하는 문제이다. (이때 주어진 수열의 정수가 모두 음수이면 최대 부분 수열의 합은 0 이라고 간주한다)

예를 들어 다음과 같은 수열이 주어졌을 때,

$$+31, -41, +59, +26, -53, +58, +97, -93, -23, +84$$

최대 부분 수열은 $[2, 6]$이며 수열의 합은 187 이 된다.

이 문제는 최대 부분 수열의 합을 구하는 것이지만, 앞으로 소개할 알고리즘을 조금만 수정하면 최대 부분 수열도 쉽게 구할 수 있다.

Sequence
Subsequence
Length
Empty sequence/subsequence

GOODMORNING!

OODOR!

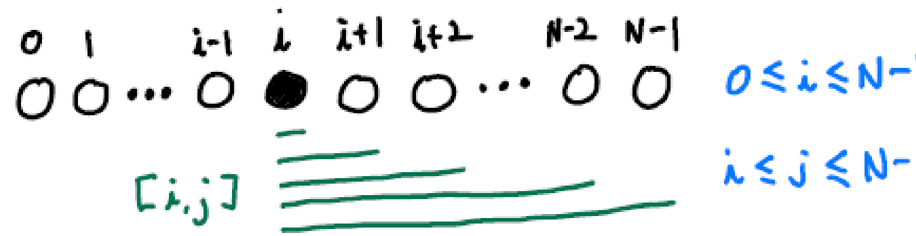
MORNIN

String
Substring
Length
Empty string/substring

Three Approaches for Max. Subsequence Sum Problem

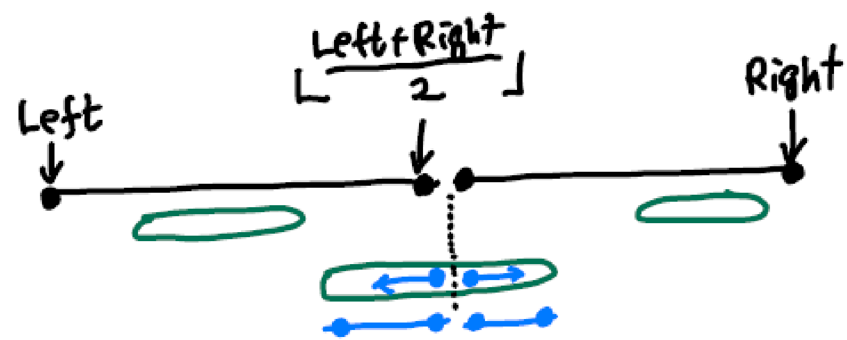
- Approach I: Simple Counting
  - Algorithms I & 2

$\boxed{O(N^3)}$ $\boxed{O(N^2)}$



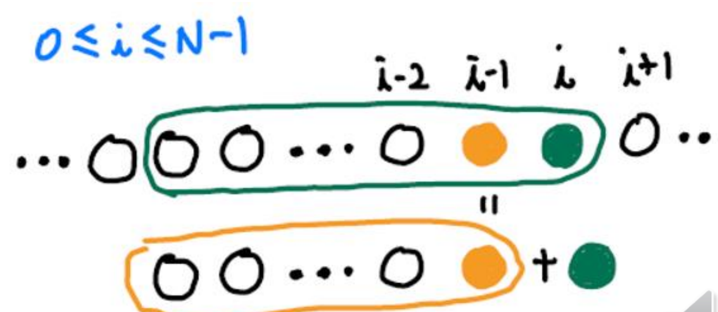
- Approach II: Divide and Conquer
  - Algorithm 3

$\boxed{O(N \log N)}$



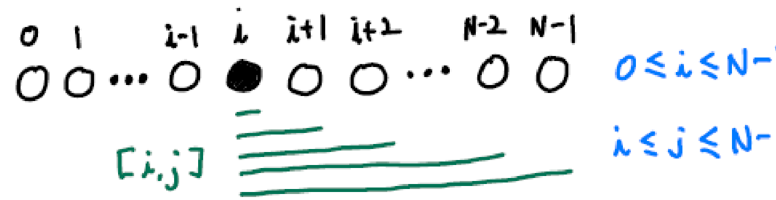
- Approach III: Dynamic Programming
  - Algorithm 4

$\boxed{O(N)}$



Maximum Subsequence Sum: **Algorithm 1**

- Strategy
  - Enumerate all possibilities one at a time.
  - No efficiency is considered, resulting in a lot of unnecessary computation!




```
int
MaxSubsequenceSum( const int A[ ], int N )
{
    int ThisSum, MaxSum, i, j, k;

    MaxSum = 0;
    for( i = 0; i < N; i++ )
        for( j = i; j < N; j++ )
        {
            ThisSum = 0;
            for( k = i; k <= j; k++ )
                ThisSum += A[ k ];

            if( ThisSum > MaxSum )
                MaxSum = ThisSum;
        }
    return MaxSum;
}
```

Is this for-loop OK for you?

$\boxed{O(N^3)}$

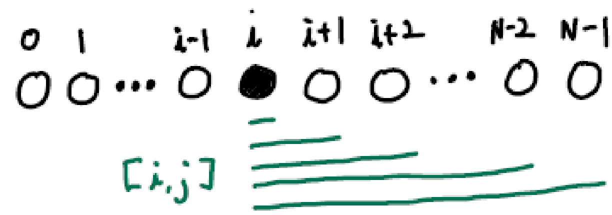
$$\sum_{i=0}^{N-1}\sum_{j=i}^{N-1}\sum_{k=i}^{j} 1 = \frac{N^3+3N^2+2N}{6}$$

$$\sum_{j=i}^{N-1}(j-i+1) = \frac{(N-i+1)(N-i)}{2}$$

$$\sum_{k=i}^{j} 1 = j-i+1$$

Maximum Subsequence Sum: **Algorithm 2**

- Strategy
  - Get rid of the inefficiency in the innermost for-loop.
    - Notice that $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k.$



$0 \le i \le N-1$

$i \le j \le N-1$

```
int
MaxSubSequenceSum( const int A[ ], int N )
{
    int ThisSum, MaxSum, i, j;

    MaxSum = 0;
    for( i = 0; i < N; i++ )
    {
        ThisSum = 0;
        for( j = i; j < N; j++ )
        {
            ThisSum += A[ j ];

            if( ThisSum > MaxSum )
                MaxSum = ThisSum;
        }
    }
    return MaxSum;
}
```

$$\sum_{i=0}^{N-1} \sum_{j=i}^{N-1} 1 = ???$$

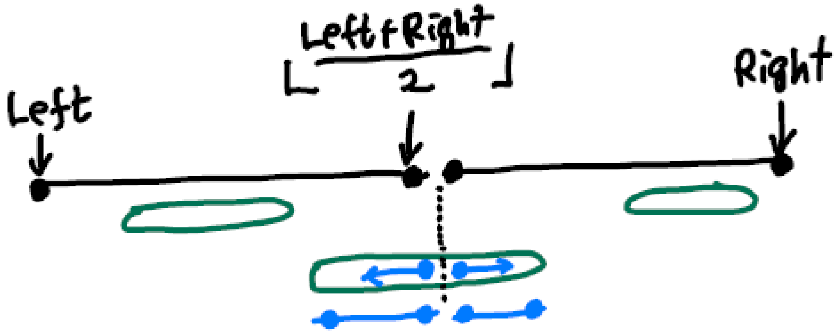
$\boxed{O(N^2)}$

Fibonacci number: $a_{n+2} = a_{n+1} + a_n (n \geq 0), a_0 = 0, a_1 = 1$

Maximum Subsequence Sum: **Algorithm 3**

- Strategy
  - Use the **Divide-and-Conquer** strategy.
    - The maximum subsequence sum can be in one of three places.

Cost: $T(n) = 2T(\frac{n}{2}) + cn,\ T(1) = d$



$\boxed{O(N \log N)}$ ← why?

```
static int
MaxSubSum( const int A[ ], int Left, int Right )
{
    int MaxLeftSum, MaxRightSum;
    int MaxLeftBorderSum, MaxRightBorderSum;
    int LeftBorderSum, RightBorderSum;
    int Center, i;

    if( Left == Right )  /* Base Case */
        if( A[ Left ] > 0 )
            return A[ Left ];
        else
            return 0;

    Center = ( Left + Right ) / 2;
    MaxLeftSum = MaxSubSum( A, Left, Center );
    MaxRightSum = MaxSubSum( A, Center + 1, Right );

    MaxLeftBorderSum = 0; LeftBorderSum = 0;
    for( i = Center; i >= Left; i-- )
    {
        LeftBorderSum += A[ i ];
        if( LeftBorderSum > MaxLeftBorderSum )
            MaxLeftBorderSum = LeftBorderSum;
    }

    MaxRightBorderSum = 0; RightBorderSum = 0;
    for( i = Center + 1; i <= Right; i++ )
    {
        RightBorderSum += A[ i ];
        if( RightBorderSum > MaxRightBorderSum )
            MaxRightBorderSum = RightBorderSum;
    }

    return Max3( MaxLeftSum, MaxRightSum,
                 MaxLeftBorderSum + MaxRightBorderSum );
}

int
MaxSubsequenceSum( const int A[ ], int N )
{
    return MaxSubSum( A, 0, N - 1 );
}
```

Kadane's algorithm

Maximum Subsequence Sum: **Algorithm 4**

```
if (ThisSum < 0)
    ThisSum = 0;
else if (ThisSum > maxSum)
    MaxSum = ThisSum;
```

- Strategy
  - Use the **Dynamic Programming** strategy.
    - Idea

$B[i]$: the sum of a maximum subsequence that ends at index i

$$\longrightarrow B[i] = \max\{\, B[i-1] + A[i],\, 0 \,\}$$

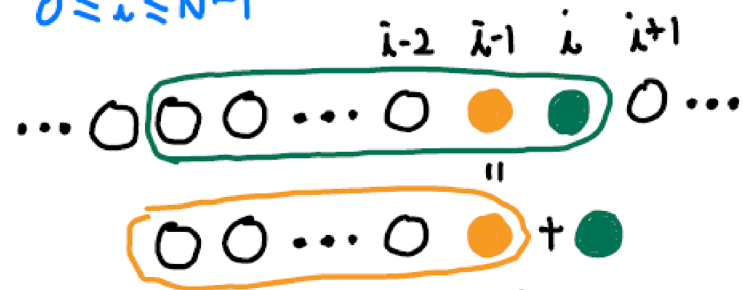
```
        int
        MaxSubsequenceSum( const int A[ ], int N )
        {
            int ThisSum, MaxSum, j;

/* 1*/      ThisSum = MaxSum = 0;
/* 2*/      for( j = 0; j < N; j++ )
            {
/* 3*/          ThisSum += A[ j ];

/* 4*/          if( ThisSum > MaxSum )
/* 5*/              MaxSum = ThisSum;
/* 6*/          else if( ThisSum < 0 )
/* 7*/              ThisSum = 0;
            }
/* 8*/      return MaxSum;
        }
```

$\boxed{O(N)}$

$0 \le i \le N-1$



optimal substructure
→ dynamic programming

- 만약에 sum이 음수라도 무방하고 1개 이상의 원소로 구성된 Subsequence (subarray)를 구하는 문제라면?

C Implementation

Maximum sum rectangle in a 2D matrix (DP-27) by GeeksforGeeks

```
int kadane(int* arr, int* start, int* finish, int n) {
    int sum = 0, maxSum = INT_MIN;
    *finish = -1;

    int local_start = 0;
    for (int i = 0; i < n; ++i) {
        sum += arr[i];
        if (sum < 0) {
            sum = 0; local_start = i+1;
        }
        else if (sum > maxSum) {
            maxSum = sum;
            *start = local_start; *finish = i;
        }
    }
    if (*finish != -1) return maxSum; // at least one non-negative number.

    // When all numbers in the array are negative
    maxSum = arr[0]; *start = *finish = 0;
    for (int i = 1; i < n; i++)  {
        if (arr[i] > maxSum) {
            maxSum = arr[i]; *start = *finish = i;
        }
    }
    return maxSum;
}
```

Empty subsequence를 허용하면 0을 리턴 (원래 문제)
Empty subsequence를 허용하지 않으면 음수 중 가장 큰 원소를 리턴