Lecture 11 : 동적계획법의 실전 활용 – 서열 정렬, 일정 계획

최근 들어 인간 유전체(genome) 서열이 밝혀짐으로써 긴 문자열을 데이터로 하는 일이 매우 늘어나고 있다. 또한, 빅데이터 시대를 맞이하며 text를 대상으로 하는 컴퓨터 작업이 많이 요구되고 있다. 이 모든 작업의 공통적인 특징은 단일 서열의 string에 관한 작업이다. 이들 문자열을 기반으로 하는 대부분 문제의 알고리즘은 동적계획법을 근간으로 한다. 왜 그것이 가능한가 하면 문자열은 그 순서가 정해져 있기 때문이다. 예를 들어 보자. 어떤 문자열 complexity와 이 단어의 문자(character)를 multi 집합으로 표시한 **{ c, o, m, p, l, e, x, i, t, y }**와는 전혀 다르기 때문이다. 따라서 우리가 주어진 문자열 S[N]에 대하여 S[N-1]까지의 데이터로 그 답을 알고 있다면 이것을 확장하여 S[N]의 답은 손쉽게 유추할 수 있기 때문이다.

특히 인간 유전체(문자열)을 대상으로 탐구해야 할 문제는 매우 많다. 이 구체적인 예를 들어가면서 설명해보자. 두 사람의 유전체 정보[1] Ga, Gb가 있다고 가정하자. 이 정보를 대상으로 우리가 해결해야 하는 문제의 예를 다음과 같다.

P1: Ga에서 반복되는 서열 중에서 가장 긴 것은 무엇인가 ?
P2: Ga와 Gb에 존재하는 가장 긴 공통서열은 무엇인가 ?
P3: Ga에서 단 한번만 나타나는 $k-$mer 중에서 가장 짧은 것은 ?
예) $aacaggtcagtactttgtgatacagaacgacatt$?

P4: Ga, Gb, Gc에 존재하는 공통서열은 무엇인가 ?
P5: Ga와 Gb에는 존재하고 Gc에는 없는 서열 중에서 가장 짧은 것은?

11.1 최장 공통서열 (LCS, Longest Common Subsequence)의 다양한 응용

유전체의 문제는 염색체의 문제로 치환되고 염색체의 문제는 결국 4개의 문자 {a, t, g, c}로 구성된 문자열(string)간의 다양한 특성 분석 문제로 귀착된다. 여러분은 먼저- subsequence과 substring의 차이를 이해해야 한다.

A_i = [c, o, m, p, u, t, e, r, s, c, i, e, n, c, t, i, s, t]
부순서, subsequence = '**cputt, cst, ttt, oeiei**
부문자열, substring = **compu, tisit, rsci, tersci**

---

1) 대부분 인간 염색체 하나의 DNA 서열의 길이는 200 mega bytes 이상이다. 인간은 이런 크기의 염색체를 무려 23개나 가지고 있다. 따라서 전체 정보는 giga 단위이다. 그러나 인간이 만든 어떤 text 창작물도 이 크기에 비교할 수 있는 것은 없다. 예를 들어 매우 큰 문서인 성경의 경우 그 글자수(characters)는 1.4 mega 정도에 불과하다. 따라서 이렇게 큰 단위의 문자열을 빠르게 처리하는 것은 매우 중요한 작업이며 이것이 생물정보학(bioinformatics)의 기본이다.

- LCS($\{A_i\},\{B_i\}$), 두 서열에서 공통 부순서 중에서 가장 긴 것
- LCS 문제의 formal (형식적인 정의)
    - input : two strings

        A = $a_1\ a_2\ a_3 \ldots a_{n-1},\ a_n$

        B = $b_1\ b_2\ b_3 \ldots b_{m-1}, b_m$
    - output : longest list of index such that $a_{c[i]} = b_{d[i]}$
- LCS의 다양한 응용의 예

    DNA, RNA, Protein 서열의 유사성(상동성 계산하기)

11.3 LCS를 동적계획법으로 해결하기

- $LCS[i][j]$ = the longest common sequence of A[1:i], B[1:j] 로 정의
- 우리는 $LCS[i][j]$를 구하는데 이 이전 단계의 답을 모두 알고 있다고 가정하다.

$LCS[i][j]$의 답은 4가지 경우로 완전히 구분된다.

case 1) A[i] == B[j] 이며 이 matching이 LCS에 들어가는 경우.

$LCS[i][j] = LCS[i-1][j-1] + 1$

case 2) A[i] 만 LCS에 포함되는 경우

$LCS[i][j] = LCS[i][j-1]$

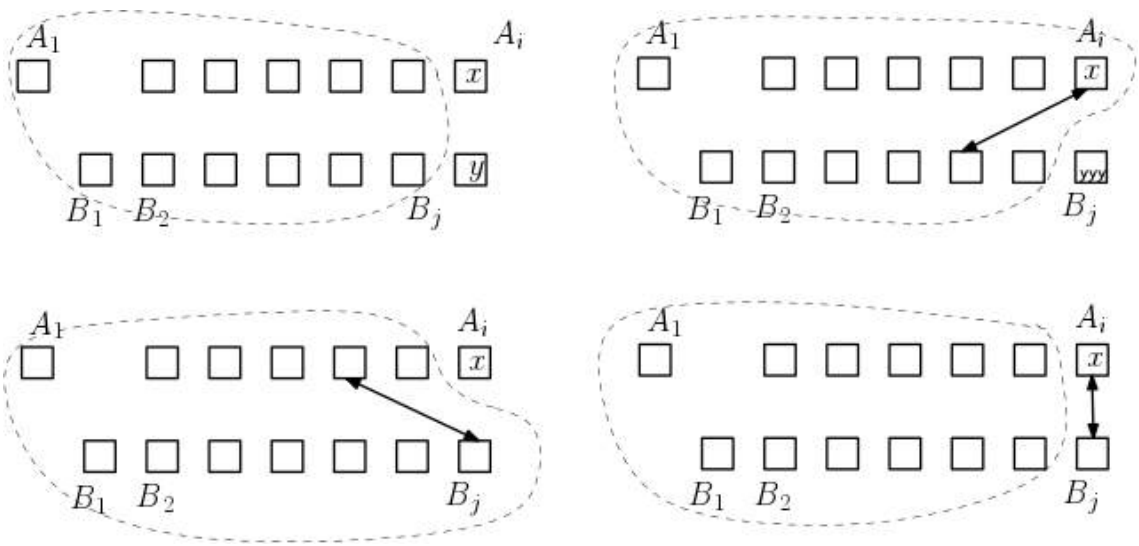
case 3) B[j] 만 LCS에 포함되는 경우

$LCS[i][j] = LCS[i-1][j]$

case 4) A[i], B[j] 모두 포함되지 않는 경우

$LCS[i][j] = LCS[i-1][j-1]$

따라서 답은 max { case1, case2, case3, case4 } 로 결정하면 된다.



	Φ	t	c	g	c	g	a	t	g	c	g	a	c
Φ	0	0	0	0	0	0	0	0	0	0	0	0	0
t	0												
a	0												
g	0												
g	0			K	L								
c	0			M									
c	0												
a	0												
t	0											W	V
g	0											U	

	Φ	g	g	t	c	a	t	t	g	c	g	a	a
Φ													
g													
c													
t													
a													
t													
g													?

Lecture 12 : 동적계획법[2]의 다양한 응용 (1)

이번 12장에서 다룰 문제는 매우 현실적인 문제로서 일종의 job scheduling에 관한 문제이다. 이 문제를 처음 접했을 때 대부분의 초심자는 어떻게 접근해야 할지 감이 잘 잡히지 않을 것이다. 매우 복잡하게 보이는 이 문제의 해결 핵심은 앞서 설명한 문제를 decompose하는 parameter를 어떻게 설정하는가 하는 것이다. N을 날짜 기준으로 해야 할 것인가, 5개의 요청(requet)로 해야 할 것인가, 아니면 개별 task기준으로 해야 할 것인가를 결정하는 것은 중요한다. 물론 어떤 기준을 선택하더라고 궁극적으로 답을 구할 수는 있지만 선택 기준에 따라서 간단하게(코딩 기준으로) 풀릴수도 있고 또는 매우 복잡하게 풀릴 수 있다. Dynamic programming 식이 복잡해지면 이것을 코드로 옮기는 과정에서 여러 오류가 발생하기 때문에 컴퓨터 과학적 입장에서는 다르게 보아야 한다.

12.1 프리 랜서의 일정계획 (Free Lance Scheduling)

day	1	2	3	4	5	6	7	8	9	10	11	12	13	14
일정	*A. 13*							*B=18*						
일정				*D=17*							*C=9*			
일정		*6*					*10*							
일정	*10*													
일정			*F=21*							*13*				
일정				*E=19*							*13*			

어떤 업자는 다음과 같은 공사 의뢰를 받았다. 그림에서 회식으로 표시된 부분이 공사가 진행 될 기간이며, 그 안에 들어있는 숫자가 해당 공사를 했을 때 받을 수 있는 비용이다. 즉 A를 1일부터 4일까지 4일간 하면 130만원을 받는다. (단위는 10만원).

그런데 같은 날 다른 공사를 동시에 할 수는 없다. 즉 A와 D는 동시에 진행될 수는 없다. 그리고 공사가 한번 시작되면 중간에 중지할 수 없다. 이 문제에서 어떤 공사를 중복없이 선택하면 가장 많은 돈을 받을 수 있는가를 결정ㅎ하는 것이다. 각 공사는 다음과 같이 $C_i(b_i, e_i, w_i)$ 표현한다. 여기에서

b_i는 시작 일(starting day), e_i는 끝나는 일, 그리고 w_i는 그 해당하는 공사 C_i를 마쳤을 때 받는 비용이다. 그리고 $O(i,j)$는 i번째 공사부터 j번째 공사 중에서 선택하여 우리가 올릴 수 있는 최대(Optimal) 이득을 나타낸다. 즉 $O(i,i)$는 항상 $O(i,i)=w_i$ 임으로 알 수 있다. 우리가 원하는 답은 전체 공사 n개 대하여 최적의

2) 2015년에 edit distance 를 빼고 추가한 내용. 관련 자료의 위치는 다음 사이트에서 확인할 수 있음
http://www.geeksforgeeks.org/dynamic-programming-set-28-minimum-insertions-to-form-a-palindrome/

이득을 말하는 것이므로 $O(1,n)$을 구하는 것이다. 이것을 Dynamic programming 으로 접근해보자. 우리는 모든 $O(i,j)$, $1 \le i,j \le n-1$ 의 결과를 알고 있다고 가정으로 이것을 이용해서 $O(1,n)$를 구하는 것이다. 자 어떻게 식을 세워야할까 ?

만일 $O(1,n)$의 답이 있다면 그 경우는 2가지다. 즉 C_n이 들어가는 경우(case A)와 C_n이 최적 선택에 들어가지 않는 경우(case B)다. 따라서 우리는 이 중에서 더 높은 비용을 보장하는 선택을 택하면 된다.

Case A) C_n이 반드시 들어가는 경우라면 일단 무조건 C_n을 넣은 다음에 이것과 겹치는 다른 일정을 모두 제거한다. C_n과 겹치는 일정을 모두 제거하고 남은 일정을 D라고 하면 그것의 O(D)를 구한 다음에 이 값이 w_n을 더하면 됩니다.

Case B) C_n이 포함되지 않는 경우를 고려하는 것은 더 쉽습니다. 그것은 $O(1,n-1)$이죠.

따라서 우리는 다음의 재귀식을 얻을 수 있습니다.

$$O(1,n) = \max\{\ O(D)+w_i\ ,\ O(1,n-1)\ \},$$

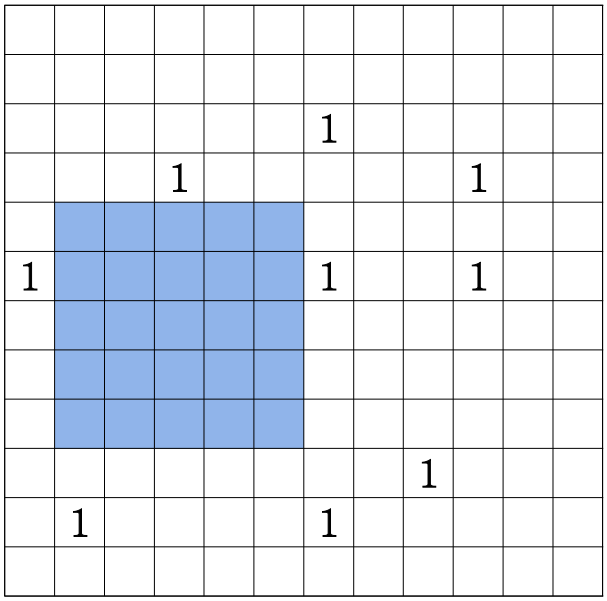
여기에서 D는 C_n과 겹치는 모든 C_*를 제거하고 남은 작업들로만 구성된 작업의 집합.

base 조건은 아까 설명한대로 $O(i,i) = w_i$ 입니다. 한개만 있을 경우에는 그것 밖에 없죠.

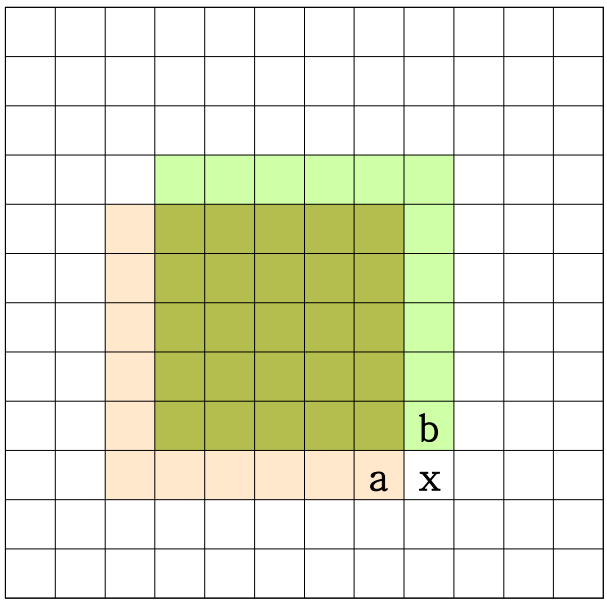
12.2 일정계획 (Job Scheduling) 연습문제용 데이터

day	1	2	3	4	5	6	7	8	9	10	11	12	13	14
일정	7								8					
일정	6					12								
일정					9						12			
일정		13						11						
일정					15									
일정			14							9				

12.3[3] $N \times N$ {0,1} 행렬(matrix)가 있다. 이것을 어떤 땅이라고 생각하자. 여기에서 1은 큰 돌과 같은 장애물을 의미라고 0은 일반 평지를 나타낸다. 우리는 이 지역에 정방형(square , $k \times k$ 크기의 평지를 찾아 여기에 돔 야구장(baseball park)을 세우려고 한다. 물론 그 평지의 k는 가장 큰 값이 되어야 한다. 이 문제를 풀어보시오. 아래 경우라면 색칠된 5×5 subsquare가 정답이다.



이 문제의 핵심은 찾아야 하는 공간이 양변의 길이가 같은 정방형(square)라는 것이다. 이 제한 조건을 이용해서 solution space의 크기를 확장해야 한다. 아래 그림에서 x를 오른쪽 아래 코너 점으로 하는 최적 정방형 공간을 찾는다고 생각해보자. 만일 우리가 아래 그림에서와 같이 a, b를 코너점으로 하는 최적 정방형 공간을 알고 있다면 이것을 이용해서 x점 기준의 최적 정방형을 어떻게 구할 것인지 생각해보자.



3) 2021년 원격강의를 위해서 추가함. 이 문제는 조금 어려운 문제다. 이 문제는 과제로 제시될 예정이다.

Lecture 13 : 동적계획법 알고리즘 개발시의 주의사항

앞서 분할정복법과 같이 동적계획법으로 접근할 때 주의해야 할 점이 있다. 어떤 개발자들의 경우 문제가 나오면 무턱대고 동적계획법을 적용하여 기본적인 solution을 만들고자 한다. 이 경우 2가지 문제를 생각해볼 수 있다. 하나는 잘못 적용하게 되면 최적의 값을 구하지 못할 수 있다. 즉 incorrect algorithm이 될 수 있다. 또는 최적의 해를 구할 수는 있지만, 일반적인 직관적인 알고리즘(straightforward)[4]으로 해결하는 것보다 복잡도나 성능 면에서 떨어진 알고리즘이 개발되기도 한다. 이 장에서는 이것을 살펴보고자 한다.

앞서도 언급하였지만, 동적계획법은 하나의 접근 방법이지 이 방법은 효율과는 무관하다. 즉 dynamic programming approach는 optimal 성능(공간과 시간)을 보장하는 알고리즘은 아니라는 것이다. 좀 쉽게 설명하자면 감이 전혀 잡히지 않는 문제에 대하여 그나마 정답을 시간과 관계없이 도출해줄 수 있다는 점에서 의미가 있다.

동적계획법 기반의 알고리즘들의 시간과 공간 복잡도는 대부분 사용하는 Table의 크기와 밀접한 연관이 있다. 만일 그 공간의 크기가 F(N)이고 하나의 cell을 채우는 데 걸리는 시간이 W(N)라고 한다면 전체 시간은 F(N)W(N)이 될 것이다. 따라서 만일 여러분이 동적계획법을 사용한다면 다른 것보다는 준비해야 할 Table의 전체 크기에 집중해야 한다. 어떤 경우 전체 Table은 필요치 않으면 S(N) 단계를 계산하는데 필요한 최소의 단계만을 저장 관리하면 된다. 예를 들어 Fibonacci 수열 계산의 경우에는 array F[N]크기의 공간이 아니라 array F[3]이면 충분히 가능하다. 물론 data move는 늘어나겠지만.

13.1 계산 문제를 동적계획법으로 해결하려고 할 때 주의해야 할 점.

a. 부분최적 구조[5]가 있는지 살펴봐야 한다. (최장거리 문제가 좋은 반례)
부분최적 구조 : 어떤 문제 전체 S의 답이 P라면 문제 일부분의 답은 역시 "답 P"의 일부분이다.

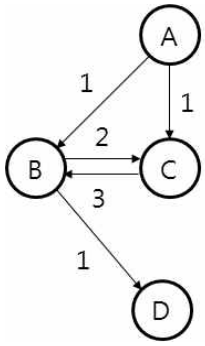
예) s에서 t까지의 최단거리(shortest path)가 P = $s \rightarrow a \rightarrow b \rightarrow c \rightarrow \cdots t$라고 하면 s에서 c까지의 shortest path는 P의 일부분인 $s \rightarrow a \rightarrow b \rightarrow c$ 이다.

최장거리 문제는 dynamic programming으로 해결할 수 없는 전형적인 경우이다.

---

4) 문제에서 주어진 대로 별다른 기교를 부리지 않고 프로그래밍하는 방법. 대표적인 방법이 앞으로 배울 greedy algorithm design이다.

5) suboptimal structure

A에서 D로의 최장 경로는 [A, C, B, D]=5 가 된다. 그러나 이 경로의 부분 경로인 A에서 C 로의 최장 경로는 [A, C]가 아니라 [A, B, C]이다. 이 문제에서 "**부분 최적의 원칙**"이 적용되지 않는다. 따라서 최장경로문제는 DP로 해결할 수 없다.	

b. Induction을 이용할 것인지, inclusion/exclusion을 이용할 것인지 생각

- induction : $P(k-1)$인 문제의 답을 알고 있을 때
  한 단계 위인 $P(k)$를 해결할 수 있는지 ?
- inclusion/exclusion
  어떤 문제의 답은 특정 사건(E)를 포함하는 경우와
  포함하지 않는 경우로 나눠, 각각을 계산해서 최적을 선택

c. 많은 문제를 풀어보면서 그 감을 익혀야 한다.

d. 관련 사이트를 열심히 살펴본다.

http://www.crazyforcode.com/dynamic-programming/

가장 많은 문제가 해설과 함께 제시되고 있다.

https://people.cs.clemson.edu/~bcdean/dp_practice/

Dynamic Programming Practice Problems

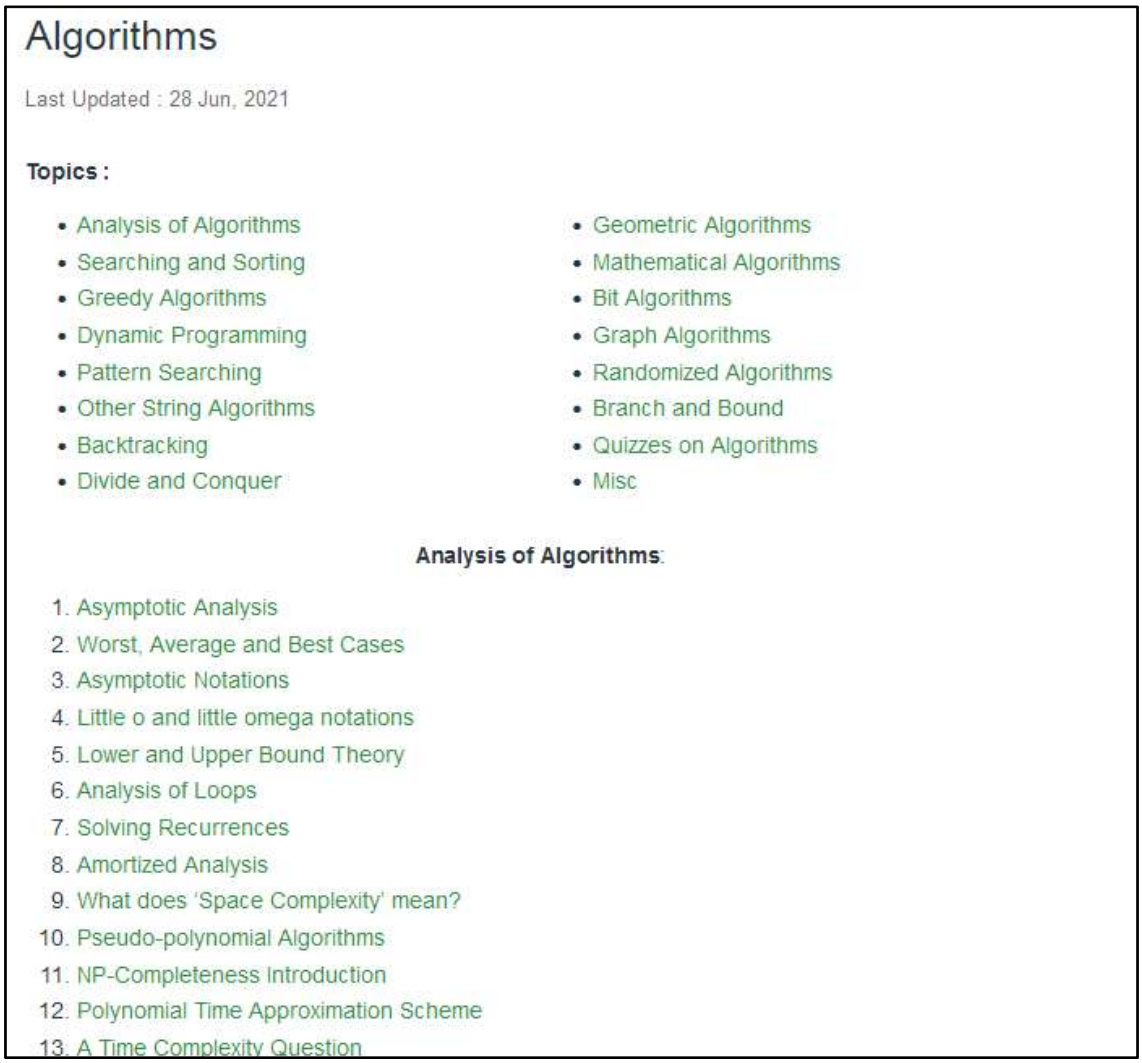
This site contains an old collection of practice dynamic programming problems and their animated solutions tha
serving as a TA for the undergraduate algorithms course at MIT. I am keeping it around since it seems to have a
web. Eventually, this animated material will be updated and incorporated into an algorithms textbook I am writi

To view the solution to one of the problems below, click on its title. To view the solutions, you'll need a machin
animations and which has audio output. I have also included a short review animation on how to solve the integ
of items allowed) using dynamic programming.

Problems:

1. Maximum Value Contiguous Subsequence. Given a sequence of n real numbers A(1) ... A(n), determine a
   which the sum of elements in the subsequence is maximized.
2. Making Change. You are given n types of coin denominations of values v(1) < v(2) < ... < v(n) (all intege
   make change for any amount of money C. Give an algorithm which makes change for an amount of mone
   problem set 4]
3. Longest Increasing Subsequence. Given a sequence of n real numbers A(1) ... A(n), determine a subseque
   maximum length in which the values in the subsequence form a strictly increasing sequence. [on problem
4. Box Stacking. You are given a set of n types of rectangular 3-D boxes, where the i^th box has height h(i),
   numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box o
   the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of cou
   functions as its base. It is also allowable to use multiple instances of the same type of box.
5. Building Bridges. Consider a 2-D map with a horizontal river passing through its center. There are n citie
   a(1) ... a(n) and n cities on the northern bank with x-coordinates b(1) ... b(n). You want to connect as man
   with bridges such that no two bridges cross. When connecting cities, you can only connect city i on the n
   (Note: this problem was incorrectly stated on the paper copies of the handout given in recitation.)
6. Integer Knapsack Problem (Duplicate Items Forbidden). This is the same problem as the example above,

이 사이트는 아래와 같이 문제 유형별 topics으로 상당히 잘 정리되어 있다.
http://www.geeksforgeeks.org/tag/dynamic-programming/



Algorithms

Last Updated : 28 Jun, 2021

Topics :

- Analysis of Algorithms
- Searching and Sorting
- Greedy Algorithms
- Dynamic Programming
- Pattern Searching
- Other String Algorithms
- Backtracking
- Divide and Conquer
- Geometric Algorithms
- Mathematical Algorithms
- Bit Algorithms
- Graph Algorithms
- Randomized Algorithms
- Branch and Bound
- Quizzes on Algorithms
- Misc

Analysis of Algorithms:

1. Asymptotic Analysis
2. Worst, Average and Best Cases
3. Asymptotic Notations
4. Little o and little omega notations
5. Lower and Upper Bound Theory
6. Analysis of Loops
7. Solving Recurrences
8. Amortized Analysis
9. What does 'Space Complexity' mean?
10. Pseudo-polynomial Algorithms
11. NP-Completeness Introduction
12. Polynomial Time Approximation Scheme
13. A Time Complexity Question

13.2 연쇄 행렬곱셈(Chained matrix Multiplication)에서
최적의 Matrix 계산순서 결정하기

N개의 행렬을 곱하는 문제가 있다. 이 문제를 행렬을 단순하기 주어진 순서대로, 예를 들어 왼쪽에서 오른쪽으로 가면서 서로 곱한다. 보통의 경우, 그런데 그 순서를 적절히 조절하여 전체 곱셈을 수를 획기적으로 줄일 수 있다. 왜냐하면 행렬 곱셈은 교환법칙은 성립하지 않지만 결합법칙은 성립하기 때문에 순서와 상관없이 그 최종 결과는 동일하기 때문이다. 따라서 그 곱셈의 순서를 잘 정하면 단위 원소끼리의 곱셈의 수를 크게 줄일 수 있다.

a. 5개의 행렬 A(3×5), B(5×8), C(8×7), D(7×2), E(2×6)을 곱하려고 한다.

b. 2개의 행렬 A(p*q), B(q*r)을 곱할 때 필요한 곱셈의 수

c. 3개의 행렬 A B C를 곱할 때 순서에 따라서 전체 곱셈의 수가 달라진다.

- **A (B C)**
- **(A B) C**

d. 위 5개의 행렬을 곱할 때 어떤 순서로 하면 좋을지 Dynamic Programming으로 해결하시오. 필요한 것은 무엇 ? i) Dynamic 식 ii) base 조건

e. M[i][j]는 Matrix M_i , M_{i+1}. . . M_j 을 곱할 때 필요한 최소의 곱셈의 횟수이다.

f. *j-i+1*= 1 인 경우를 생각해보자. M[1][1], M[2][2], M[3][3].... 은 무엇인가 ?
j-i+1= 2 인 경우를 생각해보자. M[1][2], M[2][3]... M[n-1][n] 은 무엇인가 ?

h. 이 최적 행렬계산 문제의 Dynamic Programming 식을 세우시오.
$M[i][j] = \min_{i \le k \le j-1} \{M[i][k] + M[k+1][j] + d_i \cdot d_k \cdot d_j\}$ 추가 곱셈

	A	B	C	D	E
A	0	*3*5*8* 120			
B		0	*5*8*7* 280		
C			0		
D				0	
E					0

13.3 두 스트링 X, Y의 최적 정렬(Optimal alignment)

두 스트링의 정렬은 두 스트링에 Gap 문자('-')를 넣어서 길이를 같도록 하는 작업으로서 단 그 평가함수의 결과가 최대가 되도록 해야하는 것이다.

match	(두 문자가 같을 경우 보상값을 받는다. 권장.) +3
mismatch	(두 문자가 다를 경우 벌칙값을 받는다.) **-1**
gap	(gap 문자가 포함될 경우 벌칙 값을 받는다.) **-2**

위 값에서 +는 reward(보상)이라고 하고 **- (minus)** 값을 penalty라고 한다. 각 column의 값을 모두 더한 값이 전체 alignment의 score이다.

우리는 ‘agtacg’와 ‘gcgacta’를 이용해서 여러 가능한 정렬의 평가 값을 계산해보자.

a	g	t	a	c	g				
g	c	g	a	c	t	a			

a	g	t	a	c	g				

제일 마지막 Cell에서 매칭으로 나타날 수 있는 모든 경우를 생각해보면 된다.

	X_i
	Y_i

	-
	Y_i

	X_i
	-

$$align(i,j) = \max \begin{cases} align(i-1, j-1) + match(i,j) \\ align(i \quad , j-1) + gap \\ align(i-1, j \quad) + gap \end{cases}$$

13.4 최적 이진탐색트리 (Optimal Binary Search Tree, OBST)

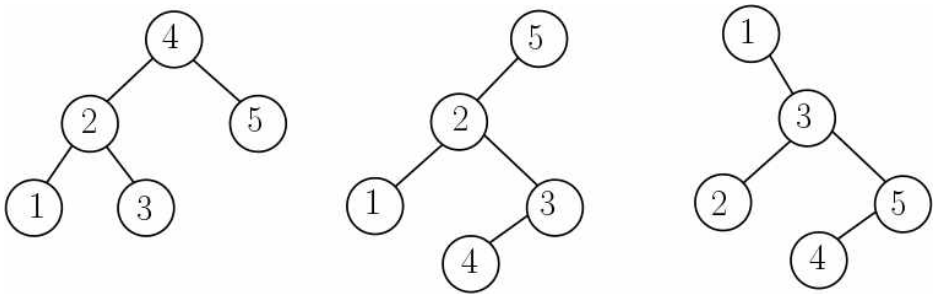
탐색트리가 "균일"하면 좋지만, 원소별 탐색빈도가 다르다면 문제가 달라진다. 예를 들어 몇 개의 reserved word 집합 S가 있고 특정 token이 이 중 어떤 reserved word인지를 찾으려고 한다. 이 문제를 해결하는 방법은 크게 3가지이다.

방법1) linear searching
방법2) hashing을 이용하는 방법
방법3) Binary search tree를 이용하는 방법

5개 keyword K1, K2, K3, K4, K5의 출연 빈도가 다음과 같을 경우 평균 탐색 복잡도를 구하시오.

Key	1	2	3	4	5
String	else	for	if	return	string
Frequence	0.3	0.1	0.05	0.05	0.5

다음 3가지 search tree의 평균비교 횟수를 구하시오.



다음 Table T[i][j]는 Keyword i부터 j까지의 keywords로 구성한 OBST의 비교 횟수를 나타낸다. 따라서 우리가 원하는 Entry는 T[1][5]를 구하는 것이다. 아래 Table을 모두 채우시오.

	1	2	3	4	5
1					
2					
3					
4					
5					

	1	2	3	4	5
1					
2					
3					
4					
5					

A working Sheet for OBST

Key	1	2	3	4	5
String	else	for	if	return	string
Frequence	0.3	0.1	0.1	0.1	0.4

	1	2	3	4	5
1	0.3				
2		0.1			
3			0.1		
4				0.1	
5					0.4

	1	2	3	4	5
1	0.3				
2		0.1			
3			0.1		
4				0.1	
5					0.4

	1	2	3	4	5
1	0.3				
2		0.1			
3			0.1		
4				0.1	
5					0.4

	1	2	3	4	5
1	0.3				
2		0.1			
3			0.1		
4				0.1	
5					0.4

13.5 전역 정렬(Global), 지역(Local) 정렬(Alignment) 실습

Match = +3, Mismatch = -1 , Gap = -1

S1 = agtccactta, S2 = acttgagtacat

아래 표를 채우시오.

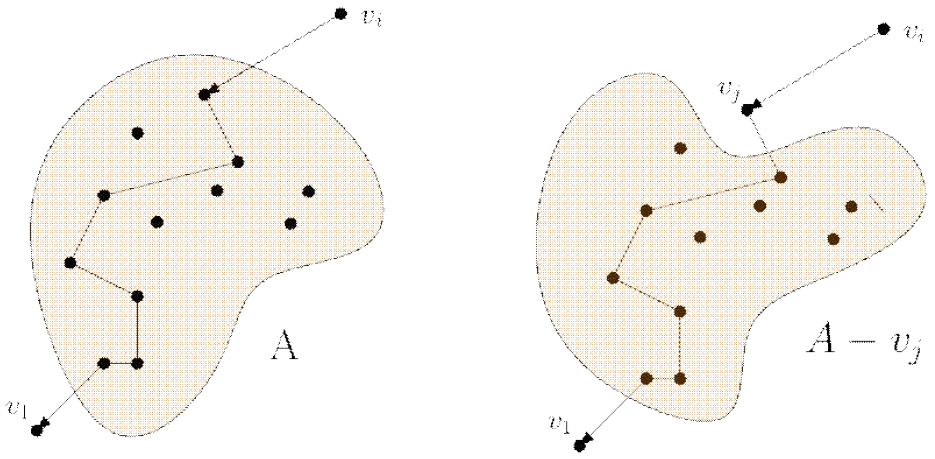
	Φ	a	g	t	c	c	a	c	t	t	a
Φ											
a											
c											
t											
t											
g											
a											
g											
t											
a											
c											
a											
t											

13.6 외판원(Traveling Salesman) 문제 해결을 위한 동적계획법[6)]

$D[i][A]$는 v_i에서 시작하여 집합 A에 있는 모든 정점을 1번만 거쳐서 v_1에 도착하는 path 중에서 가장 짧은 경로를 나타내기로 한다. 만일 A가 Null 집합이라고 한다면 이는 edge(i,1)의 길이를 의미할 것이다.

따라서 만일 TSP가 있다면 그것의 종류는 v_1에서 시작하여 v_i로 가서 $D[i][A]$로 되돌아오는 TSP 경로 중에서 최적인 것이 될 것이다. 따라서 전체 vertex 집합 S에서의 답, 즉 우리가 원하는 최종의 답은 다음과 같이 정리된다. 아래 식에서 W[i][j]는 edge (v_i, v_j)의 weight를 의미한다.

$$Optimal\ TSP(V) = \min\{W[1][j] + D[i][V - \{v_1, v_j\}]\} \qquad \text{for } 2 \le j \le n$$



$D[i][A]$ Optimal TSP route for A from v_i to v_1

위 식을 V의 모든 부분집합 A에 대하여 정리하면 다음과 같은 Dynamic 식이 찾아진다.

$$D[i][A] = \min\{W[i][j] + D[j][A - v_j]\} \text{ for } v_j \in A$$
$$D[i][\varnothing] = W[i][1]$$

위에서 2번째 식은 base 조건이다. 만일 |V|=50이라고 한다면 |V'|=49인 50개의 set에 대하여 모두 답을 구해야 하고, 더하여 |V''|=25라면 $\binom{50}{25}$개의 모든 부분집합에 대하여 답을 구해두어야만 |V'''|=26인 새로운 집합에 대하여 최적을 값을 구할 수 있어 현실적으로 $n > 40$인 경우 이 방법은 사용할 수 없다. 앞서에도 말을 했지만 Dynamic programming은 체계적인 해결방법을 제시해주는 것이지, 효율적인 알고리즘을 항상 보장하는 것은 아니다.

---

6) 중요합니다. 각종 배달(delivery) application에서 자주 응용되는 문제입니다. 물론 실전에서 $n > 50$인 경우는 다양한 heuristics이 사용됩니다. 예를 들어 Genetic algorithm이라든지.

k개의 지점으로 구성된 모든 부분집합 A에 대하여 $n-1-k$개의 도시를 시작점으로 고려해야하고 (위 그림에서 v_i). v_j에 해당하는 바로 다음 출발지를 고려해야하므로 그 실제 복잡도T(n)은 다음과 같이 계산된다.

$$T(n) = \sum (n-1-k)\, k \binom{n-1}{k}$$

이를 위하여 먼저 아래 식을 증명해야 한다.

$$\sum_{k=1}^{n} k \binom{n}{k} = n\, 2^{n-1}$$

이것을 증명하는 방법은 많으나 우리는 combinatorial한 방법으로 한다. 위 식은 n명의 사람에서 k명을 선발하고 그 중에서 특정인은 대표로 지정하는 방법의 수와 같다. n명에서 k명을 선택하는 경우의 수가 $\binom{n}{k}$이고 이 뽑힌 k명에서 각각을 대표로 선발할 수 있기 때문이다. 이를 다른 방법으로도 헤아릴 수 있다. 모인 사람 중에서 i 번이 대표가 되는 모든 경우의 수는 n명에서 i번을 제외한 모든 부분 집합에서 i번을 나중에 넣어 대표로 하는 방법이다. 따라서 $n-1$명으로 가능한 모든 부분집합(공집합 포함)의 갯수가 2^{n-1}이고 그렇게 대표로 선발할 수 있는 사람의 수가 n명이므로 그 값은 $n\, 2^{n-1}$이 되는 것은 자명하다.