9주차 : Knapsack Problem
& Matrix Chain Multiplication

알 고 리 즘

2015. 10. 29.

충남대학교 컴퓨터공학과 임베디드 시스템 연구실
TA 권진세

Overview

▶ 이번 주 실습 / 과제

1) Matrix Chain Multiplication 구현하기

2) Optimal Binary Search Tree 구현하기

Matrix-Chain Multiplication

▶ Matrix Chain Multiplication

행렬-체인 곱셈 = 최적 결합 법칙 찾기 이다.

$\mathbf{A}(p \times q)$, $\mathbf{B}(q \times r)$, $\mathbf{C}(r \times s)$의 3개의 행렬이 있다.

mult[(AB)C] = pqr + prs
mult[A(BC)] = qrs + pqs

만약 p = 5, q = 4, r = 6, s = 2 일 때

mult[(AB)C] = pqr + prs = 5*4*6 + 5*6*2 = 180
mult[A(BC)] = qrs + pqs = 4*6*2 + 5*4*2 = 88

결합 순서를 이용한 계산 비용 절약 알고리즘이다.

Matrix-Chain Multiplication

▶ Matrix Chain Multiplication

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

위 식은 행렬i와 행렬j의 곱의 계산 비용을 일반화한 식이다.

Ex) A1($p0 \times p1$), **A2**($p1 \times p2$), **A3**($p2 \times p3$)

(p0 = 5, p1 = 4, p2 = 6, p3 = 2)

1. A1과 A2의 계산 cost는

m[1,2] = m[1,1]+m[2,2]+p0*p1*p2 = 0+0+5*4*6 = 120

2. A1과 A3의 계산 cost는

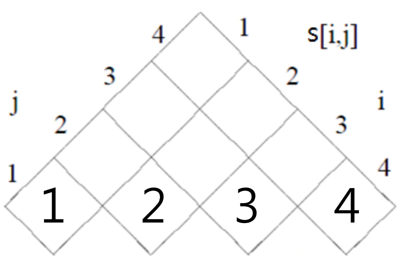
$$m[1,3] = \min \begin{cases} m[1,1]+m[2,3]+p0*p1*p3 = 0+m[2,3]+5*4*2 = 88 \\ m[1,2]+m[3,3]+p0*p2*p3 = 120+0+5*6*2 = 180 \end{cases}$$

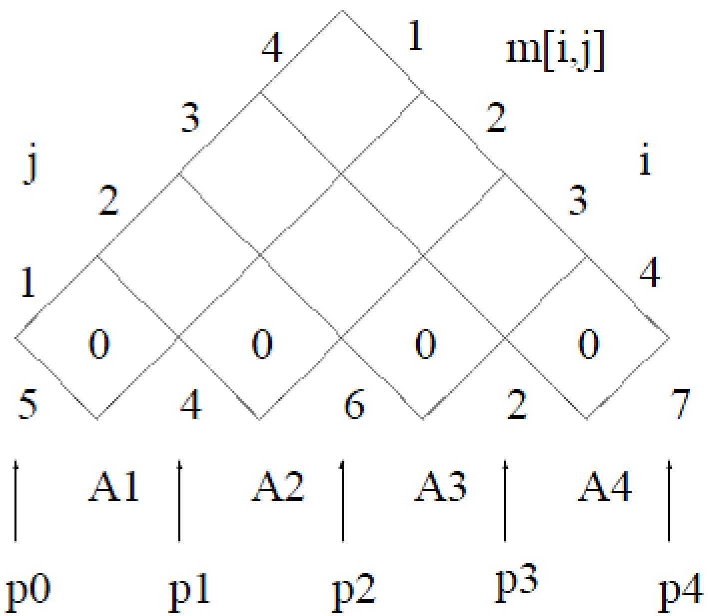
m[2,3] = m[2,2]+m[3,3]+p1*p2*p3=0+0+48=48

Matrix-Chain Multiplication

Example: Given a chain of four matrices A_1, A_2, A_3 and A_4, with $p_0 = 5$, $p_1 = 4$, $p_2 = 6$, $p_3 = 2$ and $p_4 = 7$. Find $m[1, 4]$.

S0: Initialization

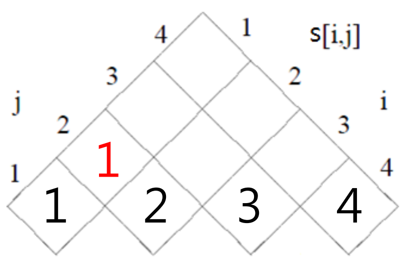


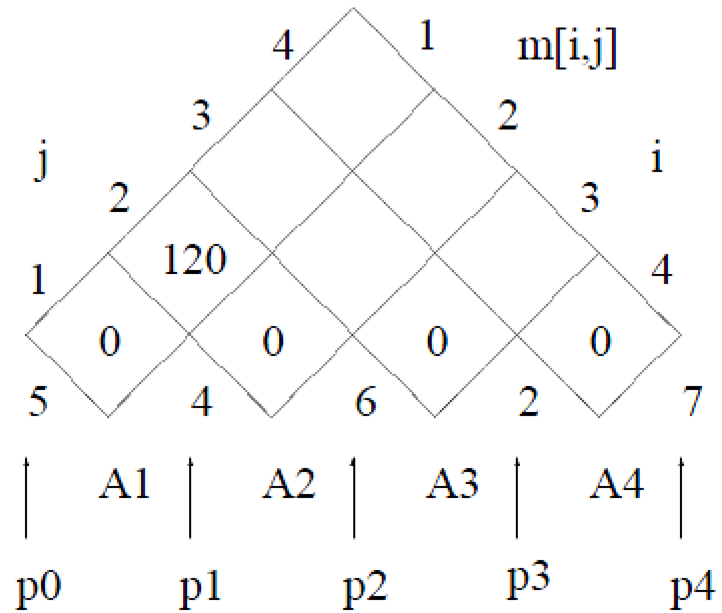


Matrix-Chain Multiplication

Stp 1: Computing $m[1,2]$ By definition

$$
\begin{aligned}
m[1,2] &= \min_{1 \le k < 2} (m[1,k] + m[k+1,2] + p_0 p_k p_2) \\
&= m[1,1] + m[2,2] + p_0 p_1 p_2 = 120.
\end{aligned}
$$

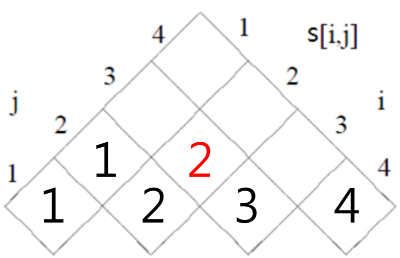


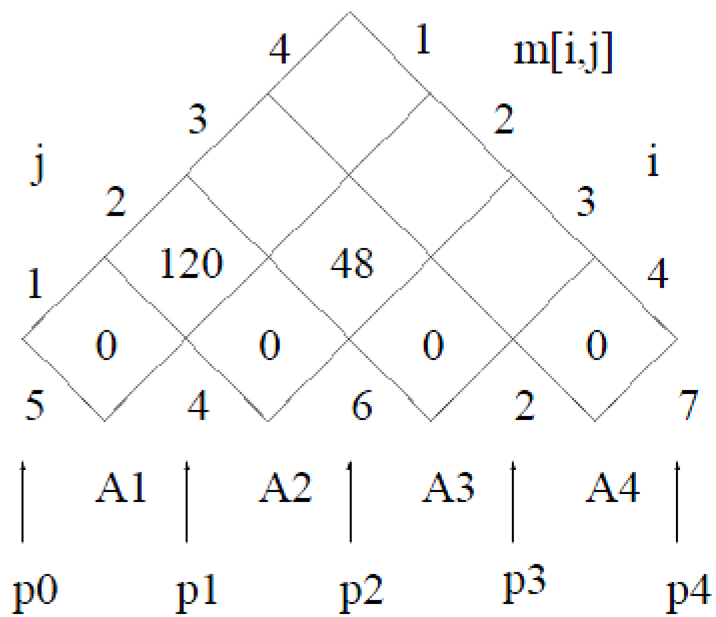


Matrix-Chain Multiplication

Stp 2: Computing $m[2,3]$ **By definition**

$$
\begin{aligned}
m[2,3] &= \min_{2\le k<3}(m[2,k]+m[k+1,3]+p_1p_kp_3) \\
&= m[2,2]+m[3,3]+p_1p_2p_3 = 48.
\end{aligned}
$$

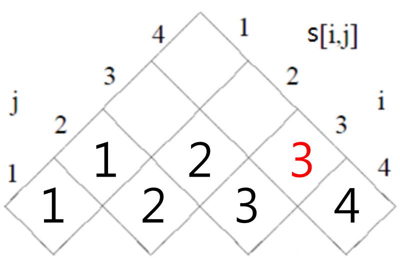


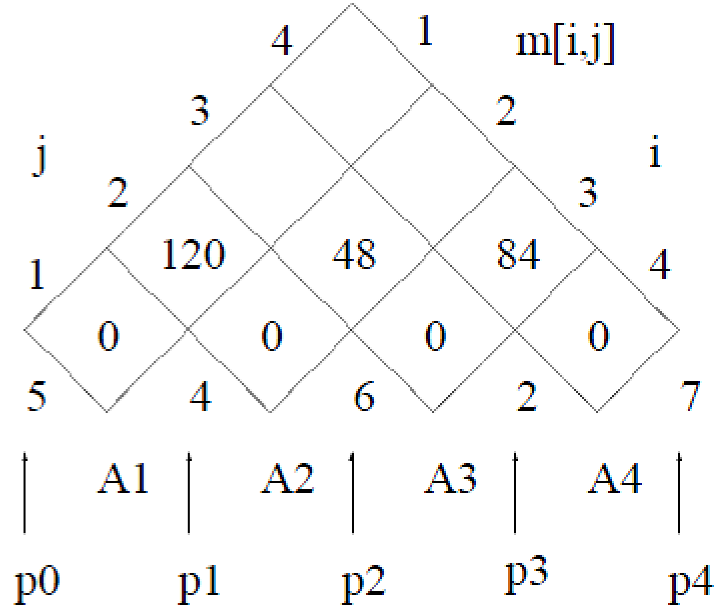


Matrix-Chain Multiplication

Stp3: Computing $m[3,4]$ By definition

$$\begin{aligned} m[3,4] &= \min_{3 \le k < 4} (m[3,k] + m[k+1,4] + p_2 p_k p_4) \\ &= m[3,3] + m[4,4] + p_2 p_3 p_4 = 84. \end{aligned}$$

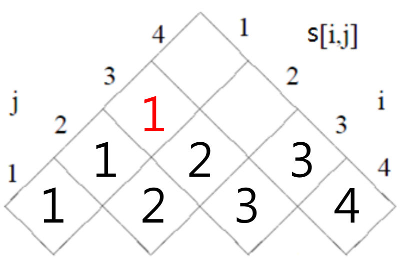


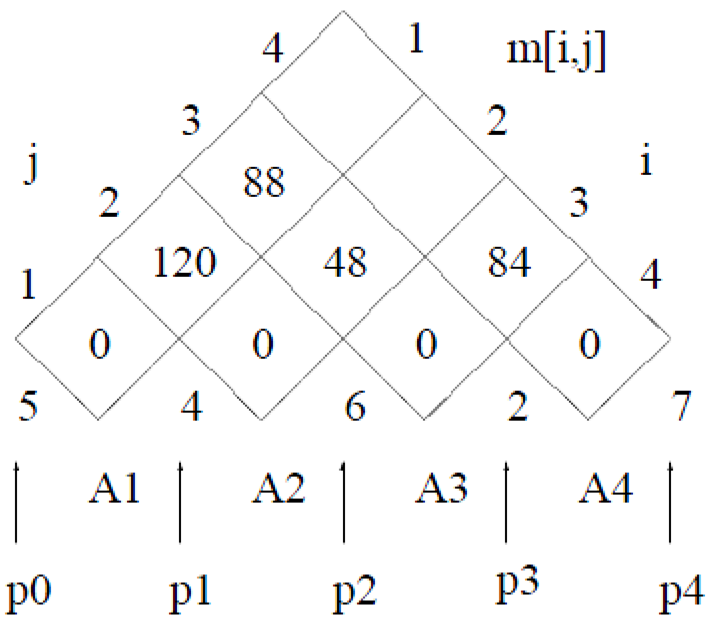


Matrix-Chain Multiplication

Stp4: Computing $m[1,3]$ By definition

$$
\begin{aligned}
m[1,3] &= \min_{1 \le k < 3} (m[1,k] + m[k+1,3] + p_0 p_k p_3) \\
&= \min \left\{ \begin{array}{l} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \end{array} \right\} \\
&= 88.
\end{aligned}
$$

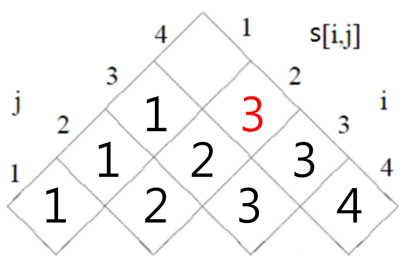


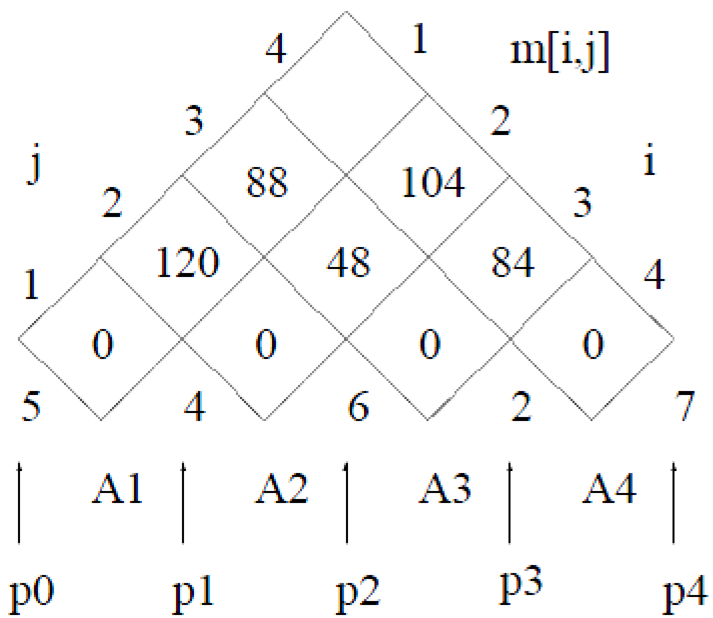


Matrix-Chain Multiplication

Stp5: Computing $m[2,4]$ By definition

$$\begin{aligned} m[2,4] &= \min_{2 \le k < 4} (m[2,k] + m[k+1,4] + p_1 p_k p_4) \\ &= \min \left\{ \begin{array}{l} m[2,2] + m[3,4] + p_1 p_2 p_4 \\ m[2,3] + m[4,4] + p_1 p_3 p_4 \end{array} \right\} \\ &= 104. \end{aligned}$$

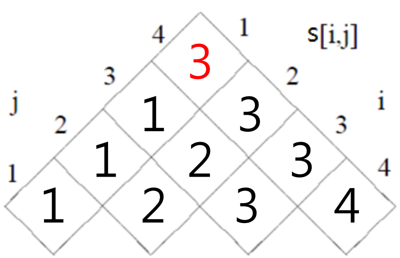


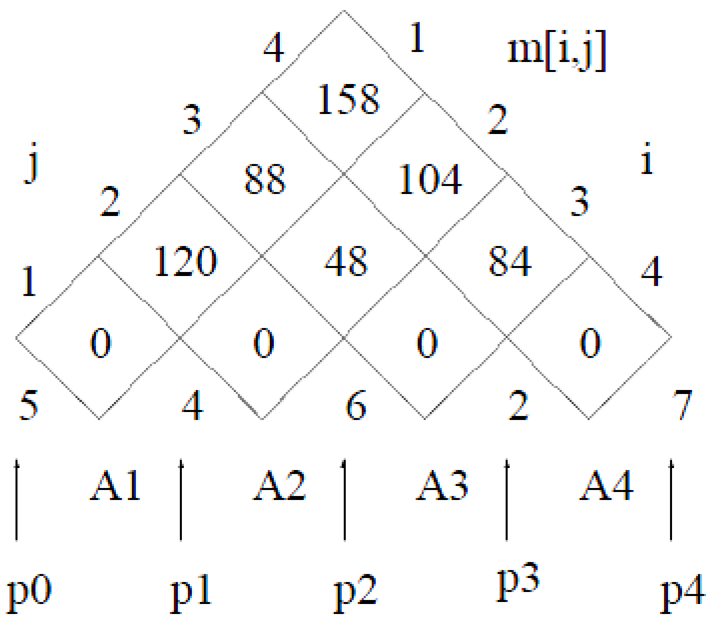


Matrix-Chain Multiplication

St6: Computing $m[1,4]$ By definition

$$\begin{aligned} m[1,4] &= \min_{1\le k<4}\left(m[1,k]+m[k+1,4]+p_0p_kp_4\right) \\ &= \min\left\{\begin{array}{l} m[1,1]+m[2,4]+p_0p_1p_4 \\ m[1,2]+m[3,4]+p_0p_2p_4 \\ m[1,3]+m[4,4]+p_0p_3p_4 \end{array}\right\} \\ &= 158. \end{aligned}$$





We are done!

Matrix-Chain Multiplication

▶ Pseudo Code

```
Matrix-Chain(p, n)
{   for (i = 1 to n) m[i, i] = 0;
    for (l = 2 to n)
    {
        for (i = 1 to n − l + 1)
        {
            j = i + l − 1;
            m[i, j] = ∞;
            for (k = i to j − 1)
            {
                q = m[i, k] + m[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j];
                if (q < m[i, j])
                {
                    m[i, j] = q;
                    s[i, j] = k;
                }
            }
        }
    }
    return m and s; (Optimum in m[1, n])
}
```

Matrix-Chain Multiplication

▶ Pseudo Code

```
PRINT-OPTIMAL-PARENS ( s, i , j )
    if  i = = j
        print  "A" & i
    else
        print "("
        PRINT-OPTIMAL-PARENS ( s,  i , s[ i,  j ] )
        PRINT-OPTIMAL-PARENS ( s, s[ i, j ] + 1,  j ] )
        print ")"
```

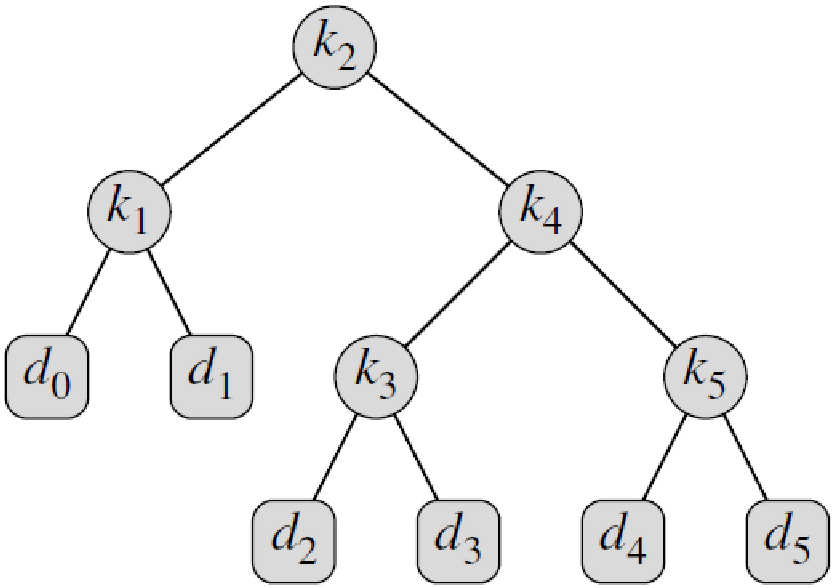
Overview

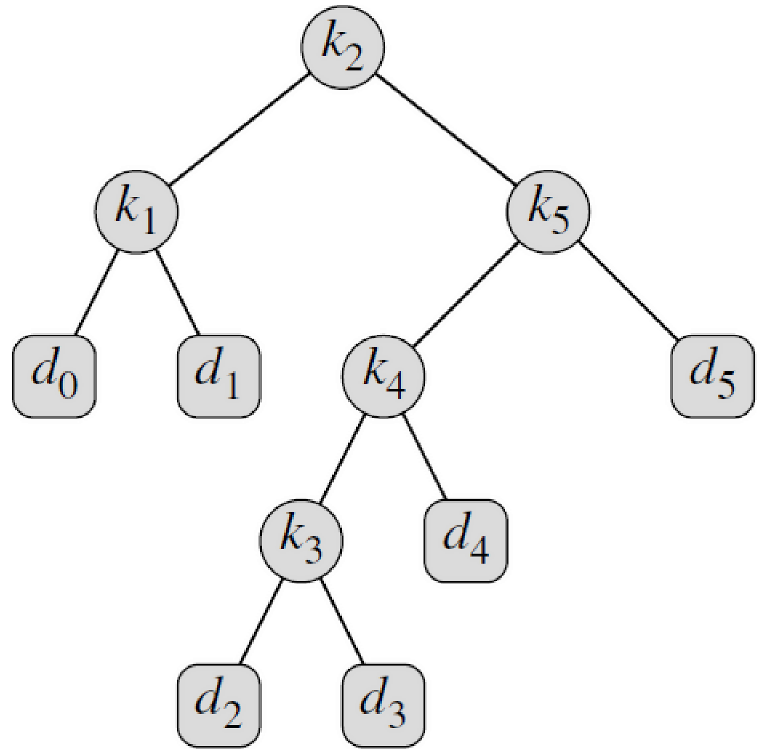
- **Optimal binary search tree**
  - 키(k_i)와 가상 키(d_i), 그리고 발생 빈도
  - $E(X)$: 기대 검색 비용 (Expected search cost)
  - $e(i,j)$와 $w(i,j)$, 그리고 $root(i,j)$

- **Practice & Homework**

  Optimal binary search tree with Dynamic programming

Optimal BST





i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

Optimal BST

$$\begin{aligned} \mathrm{E}[\text{search cost in } T] &= \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i \ , \end{aligned}$$

node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	2	0.10	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

Optimal BST

$$w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

$$w(i, j) = w(i, r-1) + p_r + w(r+1, j)$$

$$w[i, j] = w[i, j-1] + p_j + q_j$$

Optimal BST

$$e[i, j] = p_r + (e[i, r-1] + w(i, r-1)) + (e[r+1, j] + w(r+1, j))$$

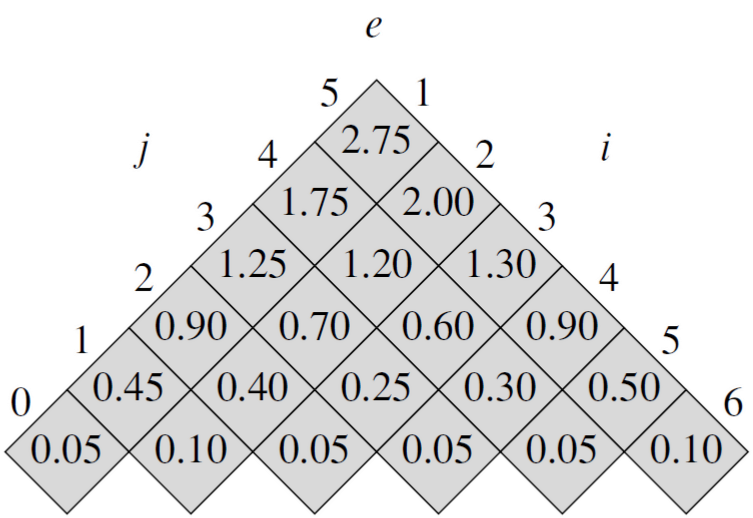
$$e[i, j] = e[i, r-1] + e[r+1, j] + w(i, j)$$

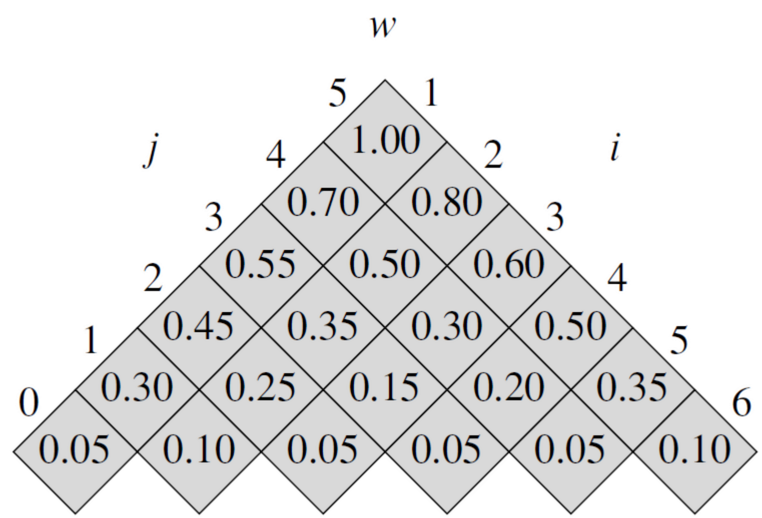
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j . \end{cases}$$

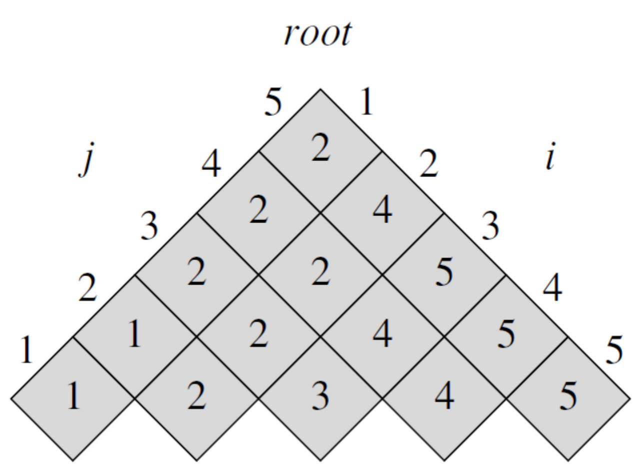
Optimal BST

```
OPTIMAL-BST(p, q, n)
for i ← 1 to n + 1
     do e[i, i − 1] ← q_{i−1}
        w[i, i − 1] ← q_{i−1}
for l ← 1 to n
     do for i ← 1 to n − l + 1
             do j ← i + l − 1
                e[i, j] ← ∞
                w[i, j] ← w[i, j − 1] + p_j + q_j
                for r ← i to j
                    do t ← e[i, r − 1] + e[r + 1, j] + w[i, j]
                       if t < e[i, j]
                          then e[i, j] ← t
                               root[i, j] ← r
return e and root
```

Optimal BST

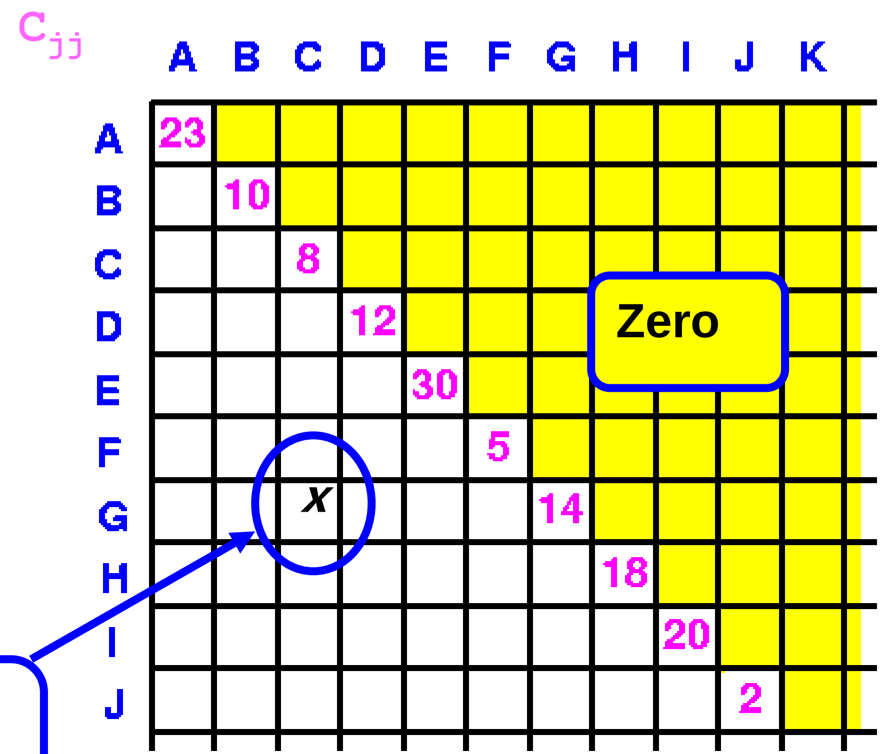

e
j
i
5 1
2.75
4 2
1.75 2.00
3 3
1.25 1.20 1.30
2 4
0.90 0.70 0.60 0.90
1 5
0.45 0.40 0.25 0.30 0.50
0 6
0.05 0.10 0.05 0.05 0.05 0.10


w
j
i
5 1
1.00
4 2
0.70 0.80
3 3
0.55 0.50 0.60
2 4
0.45 0.35 0.30 0.50
1 5
0.30 0.25 0.15 0.20 0.35
0 6
0.05 0.10 0.05 0.05 0.05 0.10


root
j
i
5 1
2
4 2
2 4
3 3
2 2 5
2 4
1 2 4 5
1 5
1 2 3 4 5

Optimal Binary Search Tree - Cost matrix

- Initialise
- Diagonal
  - `C[j,j]`
  - Costs of one-element 'trees'
- Below diagonal
  - `C[j,k]`
  - Costs of best tree `j` to `k`



Cost of best tree C-G

Optimal Binary Search Tree - Cost matrix

- Store the costs of the best two element trees
- Diagonal
  - `C[j,j]`
  - Costs of one-element 'trees'
- Below diagonal
  - `C[j-1,j]`
  - Costs of best 2-element trees `j-1` to `j`

$C_{j-1,j}$

	A	B	C	D	E	F	G	H	I	J	K
A	23										
B	43	10									
C		26	8								
D			28	12							
E				54	30						
F					40	5					
G						24	14				
H							46	18			
I								56	20		
J									24	2	

Optimal Binary Search Tree - Root matrix

- Store the roots of the best two element trees
- Diagonal
  - Roots of 1-element trees
- Below diagonal
  - `best[j-1,j]`
  - Root of best 2-element trees `j-1` to `j`

	A	B	C	D	E	F	G	H	I	J	K
A	0										
B	0	1									
C		1	2								
D			3	3							
E				4	4						
F					4	5					
G						6	6				
H							7	7			
I								8	8		
J									8	9	

Optimal Binary Search Tree - 3-element trees

- Now examine the 3-element trees

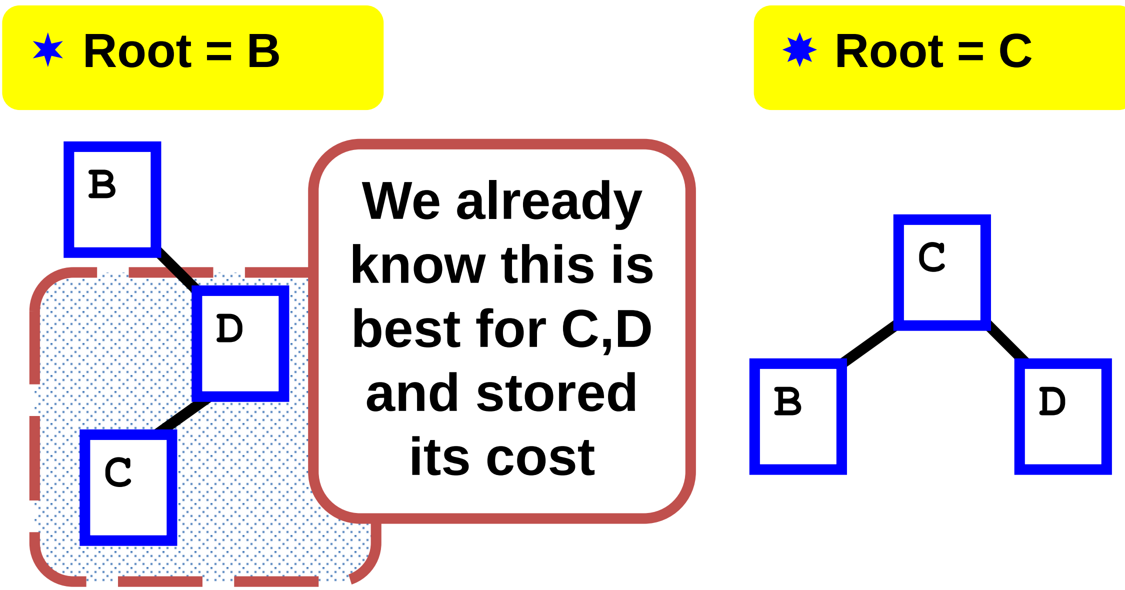
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	..
23	10	8	12	30	5	14	18	20	2	4	11	7	22	22	10	..

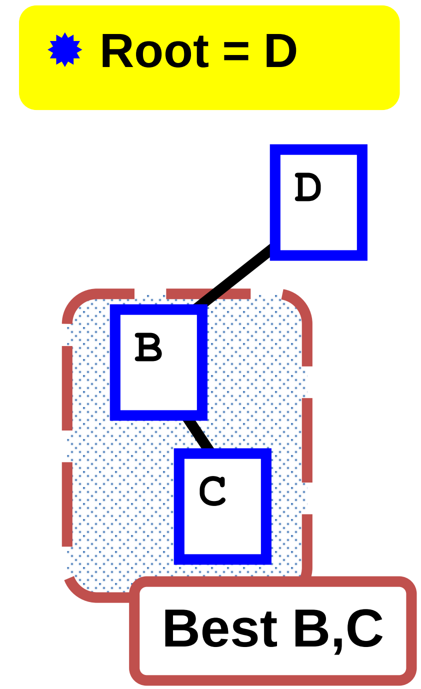
- Choose each in turn as the root
  - B with (C,D) to the right
  - C with B and D as children
  - D with (B,C) to the left
- Find best, store cost in `C[B,D]`
- Store root in `best[B,D]`

Next slide

Optimal Binary Search Tree - 3-element trees

- 3-element trees





- Find best, store cost in `C[B,D]`
- Store root in `best[B,D]`

Optimal Binary Search Tree - 3-element trees

- Similarly, update all `C[j-2,j]` and

	A	B	C	D	E	F	G	H	I	J	K
A	23										
B	43	10									
C	67	26	8								
D		52	28	12							
E			78	54	30						
F				64	40	5					
G					73	24	14				
H						60	46	18			
I							86	56	20		
J								60	24	2	

Costs

	A	B	C	D	E	F	G	H	I	J	K
A	0										
B	0	1									
C	0	1	2								
D		2	3	3							
E			4	4	4						
F				4	4	5					
G					4	6	6				
H						6	7	7			
I							7	8	8		
J								8	8	9	

Roots

Optimal Binary Search Trees - 4-trees

- Now the 4-element trees

  eg A-D

– Choose A as root	**Use 0 for left** **Best B-D is known**
– Choose B as root	**A-A is in C[0,0]** **Best C-D is known**
– Choose C as root	**A-B is in C[0,1]** **D is in C[3,3]**
– Choose D as root	**A-C is in C[0,2]** **Use 0 in C[4,3] for right**

Optimal Binary Search Trees

- Final cost will be in `C[0,n-1]`

	A	B	C	D	E	F	G	H	I	J
A	23									
B	43	10								
C	67	26	8							
D	104	52	28	12						
E	180	112	78	54	30					
F	195	122	88	64	40	5				
G	230	155	121	97	73	24	14			
H	284	209	175	151	125	60	46	18		
I	345	270	236	212	180	101	86	56	20	
J	353	278	244	220	186	107	92	60	24	2

Final cost

Optimal Binary Search Trees

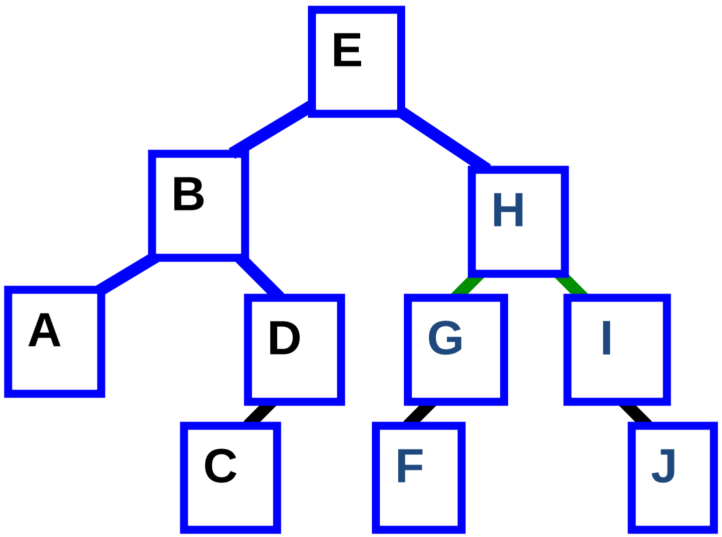
- Construct the search tree
- Root will be in
  `best[0,n-1]`
- If `r0 = best[0,n-1]`,
- Left subtree root is
  `best[0,r0-1]`,
  Right subtree root is
  `best[r0+1,n-1]`

	A	B	C	D	E	F	G	H	I	J
A	0									
B	0	1								
C	0	1	2							
D	1	2	3	3						
E	2	4	4	4	4					
F	2	4	4	4	4	5				
G	4	4	4	4	4	6	6			
H	4	4	4	4	6	6	7	7		
I	4	4	4	4	7	7	7	8	8	
J	4	4	4	4	7	7	7	8	8	9

Root = ‘E’

Optimal Binary Search Trees

- Construct the search tree



	A	B	C	D	E	F	G	H	I	J
A	0									
B	0	1								
C	0	1	2							
D	1	2	3	3						
E	2	4	4	4	4					
F	2	4	4	4	4	5				
G	4	4	4	4	4	6	6			
H	4	4	4	4	6	6	7	7		
I	4	4	4	4	7	7	7	8	8	
J	4	4	4	4	7	7	7	8	8	9

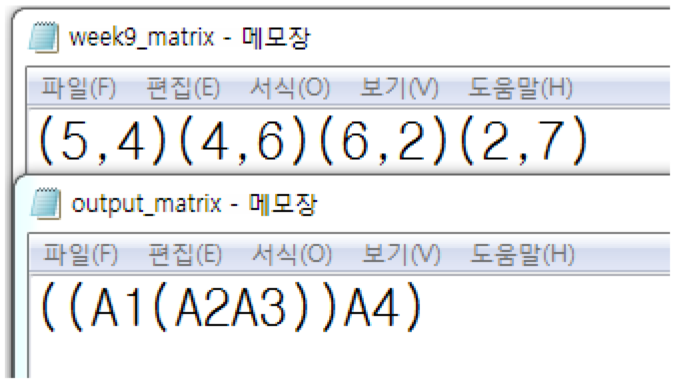
Practice / Homework

1. **Matrix Chain Multiplication구현**

과제 예시)

Input Data :

Output Data :



Practice / Homework

※ 그 외 실습 과제 수행 중 유의 사항

- 포함내용 : 코드만 제출
  - ※ 이번주 보고서 제출을 없습니다.
  - ※ 201500000_09_matrixchain.c
  - 201500000_09_optimalbst.c

  (두 개의 소스코드를 압축해서 보내세요)
- 제출이름 :
  - 메일 : [알고리즘00반]_201500000_홍길동_9주차
  - 파일 : 201500000_09.zip

- 제출기한 : 2015-11-12 18:00까지

- 메일주소 : kwonse@cnu.ac.kr

APPENDIX 1. File I/O

1. 파일 입출력 방법

```
FILE* fp;          //fp : input file pointer
FILE* fop;         //fop : input file pointer

//파일 이름은 "00_201500000_insertion.txt"
//입출력 파일은 *.c 소스파일과 같은 폴더에 있어야 한다.
fp = fopen(FILENAME,"rt");                //입력 파일 열기
fop = fopen(FILENAME2,"wt");              //출력 파일 열기

if (fp == NULL) {
        printf("**** Input File open error ****₩n");
        exit(1);
}  //파일 없을 경우 예외처리로 프로그램 종료
```

APPENDIX 1. File I/O (계속)

```
while(!feof(fp)){
} //파일의 끝날때 까지 반복

fscanf(fp,"%d, ", &변수);        // ex) 123, 456,
                                // int 값만 추출함 ‘, ’는 제외됨

fprintf(fop, "%d", 출력할 값);  // 파일 출력 시 사용

fclose(fp);
```

APPENDIX 2. 배열 넘기기

1. Main 함수의 배열을 주소로 넘겨서 다룰 때 !! 이중포인터 사용
   - 장점 : 메모리 절약, 리턴 불필요.

```
#include <stdio.h>
#include <stdlib.h>

void user_malloc(int** num);

void main(void){
    int *ptr;
    user_malloc(&ptr);      //포인터 변수 ptr의 주소를 인자로 보냄.
    printf("%d₩n", *ptr);  //출력 값은 10 이다.
    return 0;
}

void user_malloc(int** num){
    *num = (int*)malloc(sizeof(int));
    (*num)[0] = 10;
}
```

Q & A.

포인터로 받은 배열의 크기를 구하는 방법? (있다)

- Malloc 함수의 선언을 보면 void* malloc(size_t size)
- Size_t는 많은경우 unsinged long int로 되어있으므로
- 메모리를 할당 할 때 이 크기만큼 더 할당해서 할당 영역의 처음 부분에 길이의 값을 저장해 두고 있음.

- *(ptr - sizeof(size_t))
- **다음과 같은 함수로 만들어 사용 가능**

```
int sizeof_ar(int* S){
    int size;
    size = *(S - sizeof(int));
    return size;
}
```

APPENDIX 4. 중간 값 찾기

3개의 원소중에 중간 값을 찾는 방법

```
int iPivot;
int ptrCenter = (ptrLeft + ptrRight) / 2;
if(!(ptrLeft < ptrCenter ^ ptrCenter < ptrRight))
    iPivot = ptrCenter;
else if(!(ptrCenter < ptrLeft ^ ptrLeft < ptrRight))
    iPivot = ptrLeft;
else
    iPivot = ptrRight;
```

APPENDIX 5. quick sort lib func

Quick sort library function

#include <stdlib.h>

```
int compareX(const void* a, const void* b)
{
    d2_arr *p1 = (d2_arr *)a,  *p2 = (d2_arr *)b;
    return (p1->x - p2->x);
}
int compareY(const void* a, const void* b)
{
    d2_arr *p1 = (d2_arr *)a,  *p2 = (d2_arr *)b;
    return (p1->y - p2->y);
}
```

qsort(arr, arr size, element size, compare_위에참조)

APPENDIX 6. 각 자료형의 최대크기

Variant limits 헤더

#include <limits.h>

=>정수형 변수의 최대값을 전처리 매크로로 저장한 헤더

#include <float.h>

⇒ ex) double 사이즈의 최대 크기를 알고 싶을 때

⇒ printf("%lf",DBL_MAX);

APPENDIX 7. 2차원 배열 동적 할당

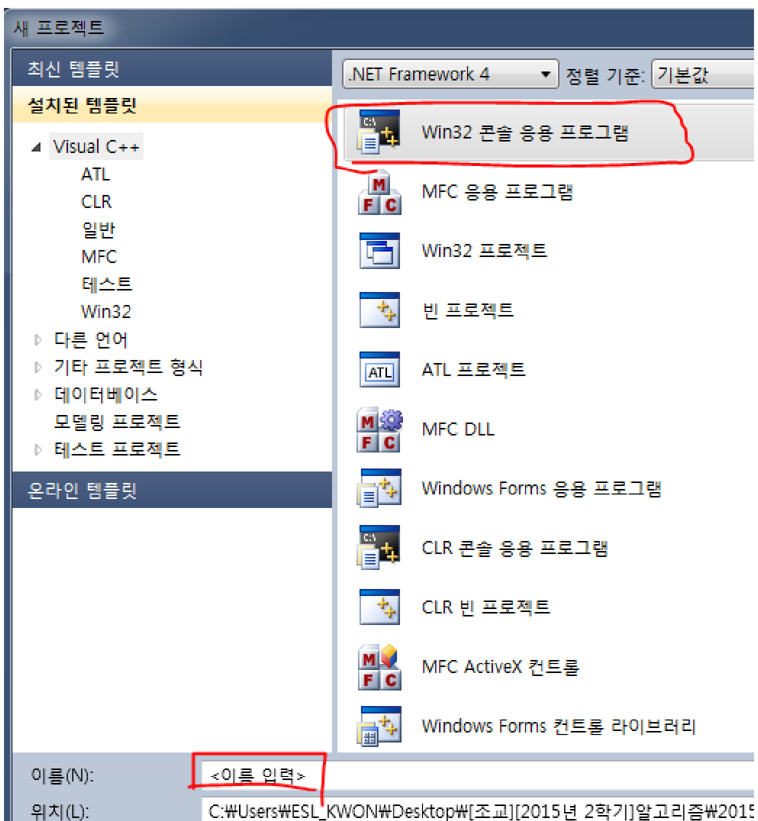
```
int input, i;

int **array = (int**)malloc(sizeof(int *)*input);

for(i=0; i<input; i++)

    array[i] = (int *)malloc(sizeof(int)*input);

//생성된 array[input][input] 을 사용

for(i=0; i<input; i++)

    free(array[i]);

free(array);
```

Trouble Shooting

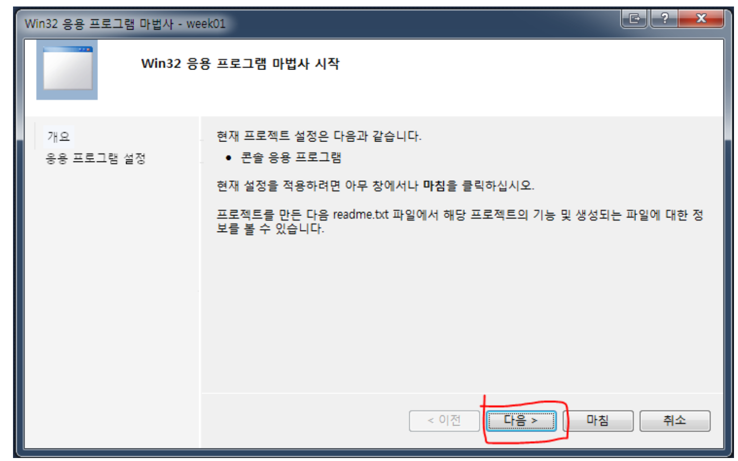
Visual Studio 2010 프로젝트 생성

1.

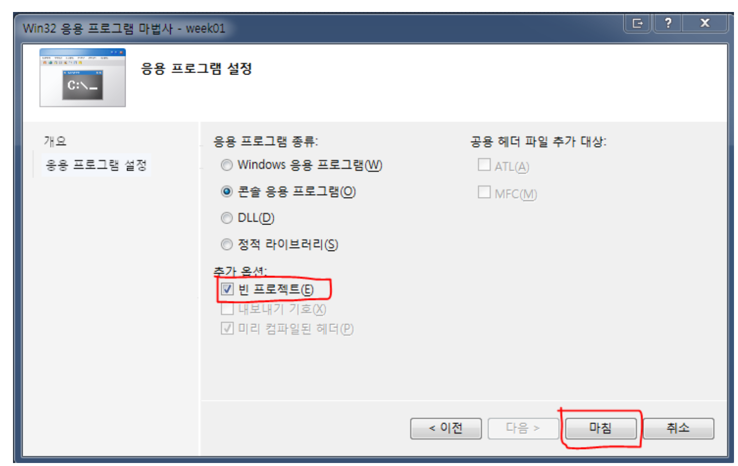
2.



3.



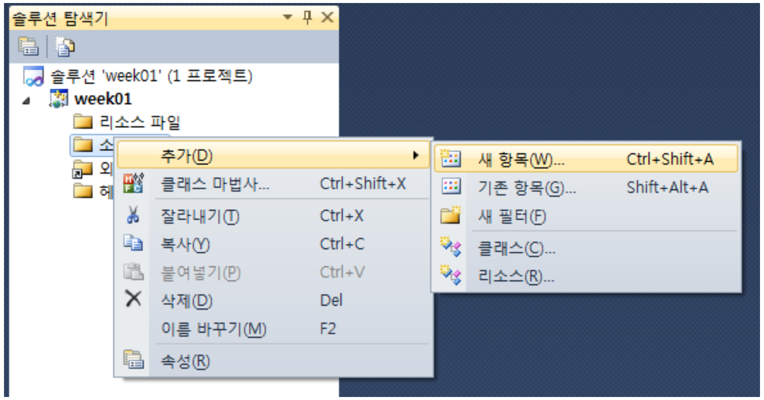
4.



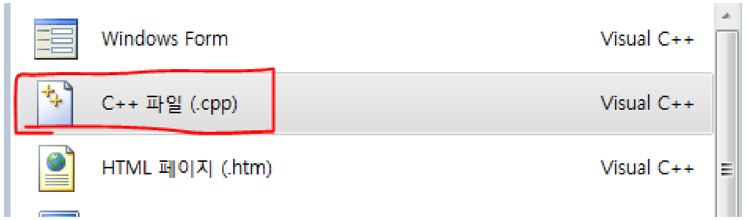
Trouble Shooting

Visual Studio 2010 프로젝트 생성

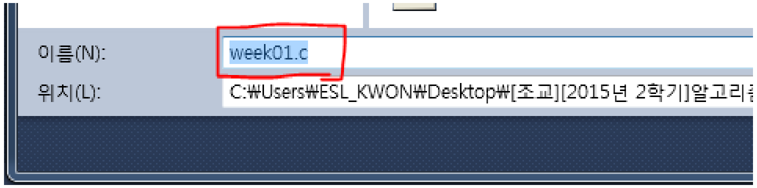
5.



6.



7.



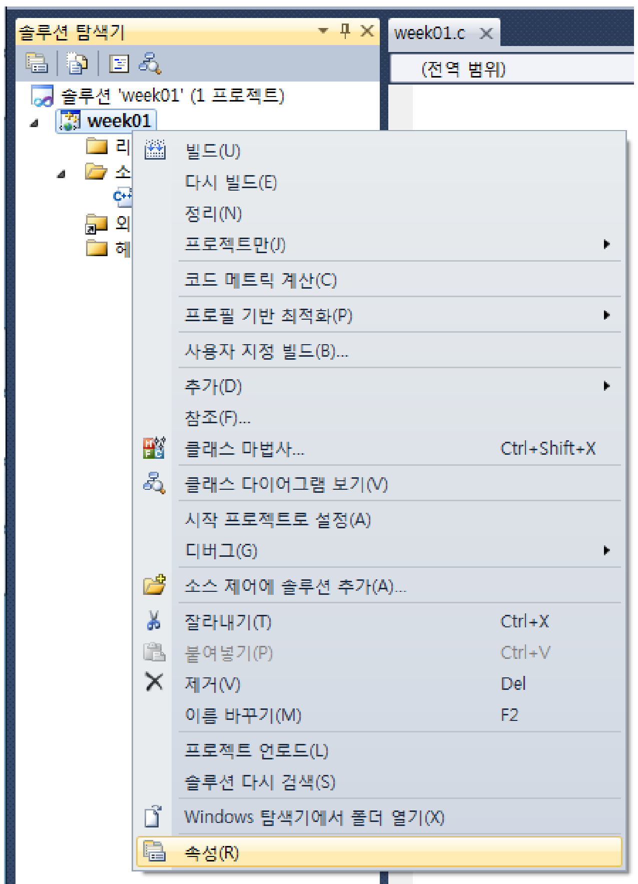
반드시 .c 로 이름 변경!!

Trouble Shooting

Visual Studio 2010 메니페스트 오류 해결

1>LINK : fatal error LNK1123: COFF로 변환하는 동안 오류가 발생했습니다. 파일이 잘못되었거나 손상되었습니다.
1>
1>빌드하지 못했습니다.

1.

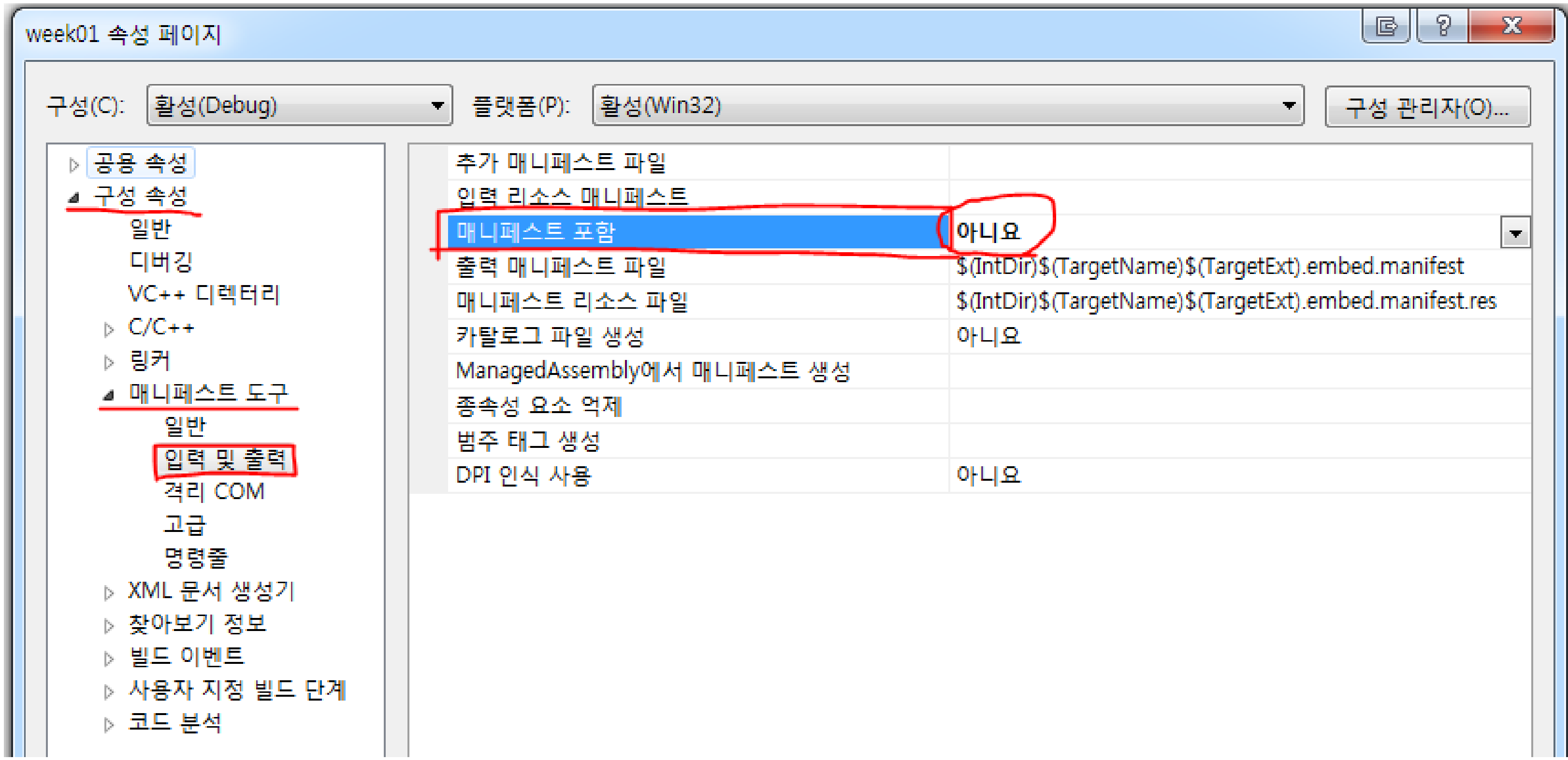


< 프로젝트 속성열기

Trouble Shooting

Visual Studio 2010 메니페스트 오류 해결

2.



구성 속성 -> 매니페스트 도구 -> 입력 및 출력 ->

메니페스트포함 : “아니요 "

Trouble Shooting

Visual Studio 2010 메니페스트 오류 해결
3. 매니페스트 문제 영구적 해결 방법

Visual Studio Service Pack 1 다운로드.
(>600MB 오래 걸림...)

https://www.microsoft.com/en-us/download/confirmation.aspx?id=23691