
알고리즘분석 실습 자료
5주차
Photo by Piotr Guzik on Unsplash

실습 소요 시간 100분

3장 동적계획 (Dynamic Programming)

실습프로그램

- ✓ 이항계수계산 (bin 재귀, bin2 배열)
- ✓ Floyd 알고리즘
- ✓ 연쇄행렬 최소곱셈 알고리즘

동적계획

- divide-and-conquer(분할정복식, 재귀) 알고리즘은 하향식(top-down) 해결법
  - ✓ 나누어진 부분들 사이에 서로 상관관계가 없는 문제를 해결하는데 적합
- 피보나찌 알고리즘은 나누어진 부분들이 서로 연관이 있음.
  - ✓ 같은 항 $f(i)$ 를 한 번 이상 → 비효율적
  - ✓ 분할정복식 방법은 적합하지 않음.
- 동적계획법(dynamic programming)은 상향식 해결법(bottom-up approach)
  - ✓ 분할정복식 방법과 마찬가지로 문제를 나눈 후에 나누어진 부분들을 먼저 푼다.
  - ✓ 인덱스를 효과적으로 설정하여 작은 문제들의 중복해결을 배제
  - ✓ 작은 문제 해결을 먼저 → 결과를 큰 문제의 해결로 확산
  - ✓ 개발 절차
    - (1) 재귀 관계식(recursive property) 정립
    - (2) 작은 사례를 먼저 해결하는 상향식 방법으로 진행

이항계수 구하기

- 이항계수(binomial coefficient) 공식

$$ {}_nC_k = \binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \le k \le n $$

- 계산량이 많은 $n!$이나 $k!$을 계산하지 않고 이항계수를 구하기 위해서 다음 식을 사용한다.(100! ?)

$$ \binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases} $$

알고리즘: 분할정복식 접근방법

- 문제: 이항계수를 계산한다.
- 입력: 음수가 아닌 정수 n과 k, 여기서 $k \le n$
- 출력: bin, $\binom{n}{k}$
- 알고리즘:

```
int bin(int n, int k) {
   if (k == 0 || n == k)
      return 1;
   else
      return bin(n-1,k-1) + bin(n-1,k)
}
```

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases}$$

알고리즘: 분할정복식 접근방법

- 문제: 이항계수를 계산한다.
- 입력: 음수가 아닌 정수 n과 k, 여기서 $k \le n$
- 출력: bin, $\binom{n}{k}$
- 알고리즘:

```
int bin(int n, int k) {
   if (k == 0 || n == k)
     return 1;
   else
     return bin(n-1,k-1) + bin(n-1,k)
}
```

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases}$$

[실습프로그램] bin, bin2 구현

```
def bin(n,k):

     재귀적 방법 구현

def bin2(n,k):


     배열을 이용한 구현


print(bin(10,5), bin2(10,5))
```

[실습프로그램]

- 큰 n값에 대해 다양한 k를 사용하여 bin, bin2의 수행시간을 비교한다.

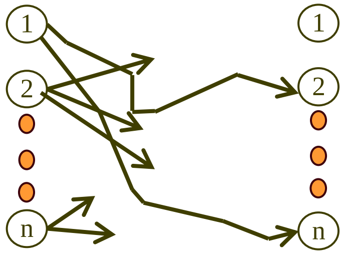
최단경로 문제
(all-pairs shortest paths problem)

- <u>보기</u> : 모든 도시에 대해, 한 도시에서 다른 도시로 갈 수 있는 가장 짧은 길을 찾는 문제

- <u>문제</u>: 가중치 포함, 방향성 그래프에서 최단경로 찾기

- <u>최적화문제(optimization problem)</u>
  - ✓ 주어진 문제에 대하여 하나 이상의 많은 해답이 존재할 때, 이 가운데에서 가장 최적인 해답(optimal solution)을 찾아야 하는 문제를 최적화문제(optimization problem)라고 한다.

- 최단경로 찾기 문제는 최적화문제에 속한다.



동적계획식 설계전략 - 자료구조

- 그래프의 인접행렬(adjacency matrix)식 표현: W

$$W[i][j] = \begin{cases} \text{이음선의 가중치}, & v_i\text{에서 } v_j\text{로 가는 이음선이 있는 경우} \\ \infty, & v_i\text{에서 } v_j\text{로 가는 이음선이 없는 경우} \\ 0, & i = j\text{인 경우} \end{cases}$$

- 그래프에서 최단경로의 길이의 표현:

$$D^{(k)}[i][j] = \text{집합}\{v_1, v_2, \ldots, v_k\}\text{의}$$

 정점들 만을 이용해서(이들을 모두 이용해야 하는 것은 아님. 일부만 이용 가능) v_i에서 v_j로 가는 최단경로의 길이

Floyd의 알고리즘 I

- 문제: 가중치 포함 그래프의 각 정점에서 다른 모든 정점까지의 최단 거리를 계산하라.
- 입력: 가중치 포함, 방향성 그래프 *W*와 그 그래프에서의 정점의 수 *n*.
- 출력: 최단거리의 길이가 포함된 배열 *D*

```
void floyd(int n, const number W[][],number D[][]) {
    int i, j, k;
    D = W;
    for(k=1; k <= n; k++)
      for(i=1; i <= n; i++)
        for(j=1; j <= n; j++)
          D[i][j] = minimum(D[i][j], D[i][k]+D[k][j]);
}
```

Floyd의 알고리즘 II

- 출력: 최단경로의 길이가 포함된 배열 D, 그리고 다음을 만족하는 배열 P.

$$P[i][j]=\begin{cases} v_i \text{에서 } v_j \text{ 까지 가는 최단경로의 중간에 놓여 있는 정점이 최소한} \\ \quad \text{하나는 있는 경우} \rightarrow \text{그 놓여 있는 정점 중에서 가장 큰 인덱스} \\ \text{최단경로의 중간에 놓여 있는 정점이 없는 경우} \rightarrow 0 \end{cases}$$

```
void floyd2(int n,const number W[][],number D[][],index P[][]) {
    index i, j, k;
    for(i=1; i <= n; i++)
        for(j=1; j <= n; j++)
                P[i][j] = 0;
    D = W;
    for(k=1; k<= n; k++)
        for(i=1; i <= n; i++)
            for(j=1; j<=n; j++)
                if (D[i][k] + D[k][j] < D[i][j]) {
                    P[i][j] = k;
                    D[i][j] = D[i][k] + D[k][j];
                }
}
```

최단경로의 출력

- 문제: 최단경로 상에 놓여 있는 정점을 출력.

```
void path(index q,r) {
 if (P[q][r] != 0) {
        path(q,P[q][r]);
        cout << " v" << P[q][r];
        path(P[q][r],r);
         }
 }
```

$W(n) \in \Theta(n)$

$P[i][j]$	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

- 위의 P를 가지고 path(5,3)을 구해 보시오.

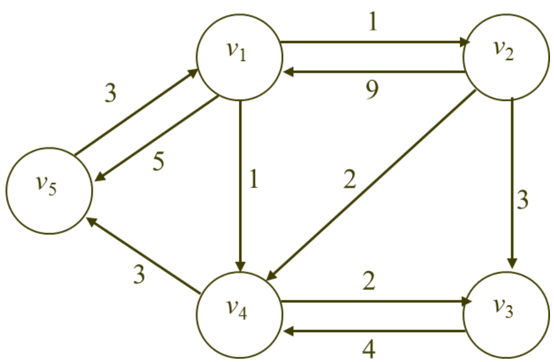
```
      path(5,3) = 4
          path(5,4) = 1
               path(5,1) = 0
               v1
               path(1,4) = 0
          v4
          path(4,3) = 0
```

결과: v1 v4.   즉, v_5에서 v_3으로 가는 최단경로 v_5, v_1, v_4, v_3 이다.

[실습프로그램] Floyd 알고리즘 구현

```
def allShortestPath(g,n):
# node number는 1부터 n


    구현




def printMatrix(d):
    n=len(d[0])
    for i in range(0,n):
        for j in range(0,n):
            print(d[i][j],end=" ")
        print()
```

```
inf=1000
g=[[0,1,inf, 1,5],
   [9,0,3,2,inf],
   [inf,inf,0,4,inf],
   [inf,inf,2,0,3],
   [3,inf,inf,inf,0]]
d, p = allShortestPath(g,5)
print()
printMatrix(d)
print()
printMatrix(p)
```



utility.py

```
def printMatrix(d):
    m = len(d)
    n=len(d[0])

    for i in range(0,m):
        for j in range(0,n):
            print(f'{d[i][j]:4d}',end=" ")
        print()

#print float matrix
def printMatrixF(d):
    n=len(d[0])
    for i in range(0,n):
        for j in range(0,n):
            print(f'{d[i][j]:5.2f}',end=" ")
        print()

def print_inOrder(root):
    if not root:
        return
    print_inOrder(root.l_child)
    print(root.data)
    print_inOrder(root.r_child)

def print_preOrder(root):
    if not root:
        return
    print(root.data)
    print_preOrder(root.l_child)
    print_preOrder(root.r_child)
```

```
def print_postOrder(root):
    if not root:
        return

    print_postOrder(root.l_child)
    print_postOrder(root.r_child)
    print(root.data)
```


Floyd 알고리즘 output

```
>>>
0 1 1000 1 5
9 0 3 2 1000
1000 1000 0 4 1000
1000 1000 2 0 3
3 1000 1000 1000 0

0 1 3 1 4
8 0 3 2 5
10 11 0 4 7
6 7 2 0 3
3 4 6 4 0

0 0 4 0 4
5 0 0 0 4
5 5 0 0 4
5 5 0 0 0
0 1 4 1 0
>>>
```

[실습프로그램] 두 노드 간의 최단경로를 출력하는 path 함수를 작성한다.

[실습프로그램]

```
from utility import *

def allShortestPath(g,n):


     구현
```

```
def path(p, q, r):
    if (p[q][r] !=0):

         구현
```

```
inf=1000
g=[[0,1,inf, 1,5],
   [9,0,3,2,inf],
   [inf,inf,0,4,inf],
   [inf,inf,2,0,3],
   [3,inf,inf,inf,0]]
d, p = allShortestPath(g,5)
print()
printMatrix(d)
print()
printMatrix(p)

path(p, 5, 3)
```

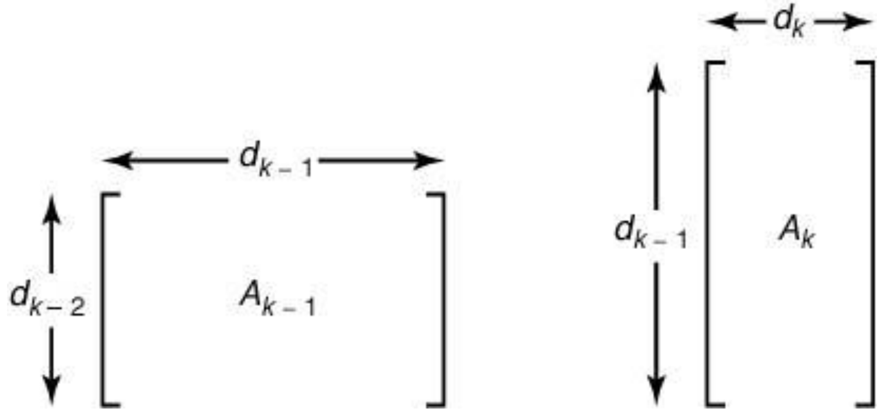
```
   0     1  1000    1     5
   9     0     3    2  1000
1000  1000     0    4  1000
1000  1000     2    0     3
   3  1000  1000 1000     0

   0    1    3    1    4
   8    0    3    2    5
  10   11    0    4    7
   6    7    2    0    3
   3    4    6    4    0

   0    0    4    0    4
   5    0    0    0    4
   5    5    0    0    4
   5    5    0    0    0
   0    1    4    1    0
v1 v4
>>>
```

연쇄 행렬곱셈(matrix-chain multiplication)

- $i \times j$ 행렬과 $j \times k$행렬을 곱하기 위해서는 $i \times j \times k$번 만큼의 곱셈이 필요.
- 연쇄적으로 행렬을 곱할 때, 어떤 행렬곱셈을 먼저 수행하느냐에 따라서 필요한 총 곱셈의 횟수가 달라짐.
- (예)
  - ✓ $A_1 \times A_2 \times A_3$.
  - ✓ A_1의 크기는 10×100, A_2의 크기 100×5, A_3의 크기 5×50.
  - ✓ $(A_1 \times A_2) \times A_3$ 곱셈의 총 횟수 7,500(=5,000+2,500)회
  - ✓ $A_1 \times (A_2 \times A_3)$ 곱셈의 총 횟수 75,000(=25,000+50,000)회
  - ✓ 따라서, 연쇄적으로 행렬을 곱할 때 곱셈의 횟수가 가장 적게 되는 최적의 순서를 결정하는 알고리즘을 개발하는 것이 목표.

연쇄 행렬곱셈 동적계획식 설계전략

- A_k의 크기는 $d_{k-1} \times d_k$: d_{k-1} : 행(row)의 수, d_k: 열(column)의 수
- A_1의 행의 수는 d_0.



$M[i][j] =$ $i \le j$일 때 A_i부터 A_j까지의 행렬을 곱하는데 필요한 기본적인 곱셈의 최소 횟수 $\quad 1 \le i \le j \le n$

$$= \begin{cases} minimum_{i \le k \le j-1}(M[i][k] + M[k+1][j] + d_{i-1}d_k d_j) & \text{if } i < j \\ M[i][i] = 0 \end{cases}$$

$$\underbrace{(A_i A_{i+1} \ldots A_k)}_{\text{C: } d_{i-1} \times d_k} \times \underbrace{(A_{k+1} \ldots. A_j)}_{\text{D: } d_k \times d_j}$$

$M[i][j] =$ $i \le j$일 때 A_i부터 A_j까지의 행렬을 곱하는데 필요한 기본적인 곱셈의 최소 횟수 $1 \le i \le j \le n$

$$= \begin{cases} minimum_{i \le k \le j-1}(M[i][k] + M[k+1][j] + d_{i-1}d_k d_j) & \text{if } i < j \\ M[i][i] = 0 \end{cases}$$

$$\underbrace{(A_i A_{i+1} \ldots A_k)}_{\text{C: } d_{i-1} \times d_k} \times \underbrace{(A_{k+1} \ldots . A_j)}_{\text{D: } d_k \times d_j}$$

최적 순서의 구축

- 최적 순서를 얻기 위해서는 $M[i][j]$를 계산할 때 최소값을 주는 k값을 $P[i][j]$에 기억한다.
- 예: $P[2][5] = 4$인 경우의 최적 순서는 $((A_2 A_3)A_4) A_5$이다. 구축한 P는 다음과 같다.

$P[i][j]$	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5

따라서, 최적 분해는 $(A_1((((A_2 A_3) A_4) A_5) A_6))$.

최소곱셈알고리즘

- 문제: n개의 행렬을 곱하는데 필요한 기본적인 곱셈의 횟수의 최소치를 결정하고, 그 최소치를 구하는 순서를 결정하라.
- 입력: 행렬의 개수 n, 배열 $d[i-1] \times d[i]$는 i번째 행렬의 규모를 나타낸다.
- 출력:
  - ✓ 기본적인 곱셈의 횟수의 최소치를 나타내는 *minmult*;
  - ✓ 최적의 순서를 구할 수 있는 배열 P, P 는 $1\ldots n-1$ by $1..n$. 여기서 $P[i][j]$는 행렬 i부터 j까지가 최적의 순서로 갈라지는 기점을 나타낸다.

$M[i][j]$	1	2	3	4	5	6
1	0	30	64	132	226	348
2		0	24	72	156	268
3			0	72	198	366
4				0	168	392
5					0	336
6						0

대각 2

대각 1

```
int minmult(int n, const int d[], index P[][]) {
    index i, j, k, diagonal;
    int M[1..n][1..n];

    for(i=1; i <= n; i++)
       M[i][i] = 0;
    for(diagonal = 1; diagonal <= n-1; diagonal++)
        for(i=1; i <= n-diagonal; i++) {
           j = i + diagonal;
           M[i][j]= minimum_{i<=k<=j-1}(M[i][k]+M[k+1][j]+
                        d[i-1]*d[k]*d[j]);
           P[i][j] = 최소치를 주는 k의 값
           }
    return M[1][n];
}
```

최적의 해를 주는 순서의 출력

- 문제: n개의 행렬을 곱하는 최적의 순서를 출력하시오.
- 입력: n과 P
- 출력: 최적의 순서

```
void order(index i, index j) {

    if (i == j)
          cout << "A" << i;
    else {
         k = P[i][j];
         cout << "(";
         order(i,k);
         order(k+1,j);
         cout << ")";
         }
}
```

- order(1,6)은 (A1((((A2A3)A4)A5)A6)) 을 출력
- $T(n) \in \Theta(n)$.

[실습프로그램] 연쇄행렬 최소곱셈 알고리즘 구현

A_1	A_2	A_3	A_4	A_5	A_6
5×2	2×3	3×4	4×6	6×7	7×8

```
import utility

def order(p,i,j):




     구현



d=[5,2,3,4,6,7,8]
n=len(d)-1

m=[[0 for j in range(1,n+2)] for i in range(1,n+2)]
p=[[0 for j in range(1,n+2)] for i in range(1,n+2)]





     구현



utility.printMatrix(m)
print()
utility.printMatrix(p)
order(p,1,6)
```

utility.py

```
def printMatrix(d):
    m = len(d)
    n=len(d[0])

    for i in range(0,m):
        for j in range(0,n):
            printf(f'{d[i][j]:4d}',end=" ")
        print()

#print float matrix
def printMatrixF(d):
    n=len(d[0])
    for i in range(0,n):
        for j in range(0,n):
            printf(f'{d[i][j]:5.2f}',end=" ")
        print()

def print_inOrder(root):
    if not root:
        return
    print_inOrder(root.l_child)
    print(root.data)
    print_inOrder(root.r_child)

def print_preOrder(root):
    if not root:
        return
    print(root.data)
    print_preOrder(root.l_child)
    print_preOrder(root.r_child)
```

```
def print_postOrder(root):
    if not root:
        return

    print_postOrder(root.l_child)
    print_postOrder(root.r_child)
    print(root.data)
```

연쇄행렬 곱셈 알고리즘 output

```
>>>
   0     0     0     0     0     0     0
   0     0    30    64   132   226   348
   0     0     0    24    72   156   268
   0     0     0     0    72   198   366
   0     0     0     0     0   168   392
   0     0     0     0     0     0   336
   0     0     0     0     0     0     0

   0     0     0     0     0     0     0
   0     0     1     1     1     1     1
   0     0     0     2     3     4     5
   0     0     0     0     3     4     5
   0     0     0     0     0     4     5
   0     0     0     0     0     0     5
   0     0     0     0     0     0     0
(A 1((((A 2A 3)A 4)A 5)A 6))
>>>
```

강의가 곧 시작됩니다.