숭실대학교 컴퓨터학부

알고리즘 2021 (나)

과제 4

이름	윤 주호
학번	20201866
출석 번호	225

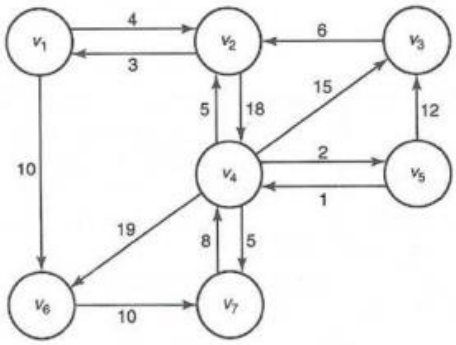
프로그램 개요

(1) 프로그램 설명

Floyd's Algorithm의 개념을 정확히 이해하고, 최단 경로를 출력해주는 알고리즘을 구현한다. 출력인 행렬 D와 P에 어떤 값이 저장되고 어떻게 변하는지 확인하며 알고리즘을 이해한다. Floyd's algorithm, Print shortest path 등.

(2) 과제 명세.

1. Use Floyd's algorithm for the Shortest Paths problem 2 (Algorithm 3.4) to construct the matrix D, which contains the lengths of the shortest paths, and the matrix P, which contains the highest indices of the intermediate vertices on the shortest paths, for the following graph. Show the actions step by step.



2. Use the Print Shortest Path algorithm (Algorithm 3.5) to find the shortest path from vertex v_7 to vertex v_3, in the graph of Exercise 5, using the matrix P found in that exercise. Show the actions step by step.

3. 위 그래프에서 v_1 => a_6, v_2 => a_5, v_3 => a_4, v_4 => a_3, v_5 => a_2, v_6 => a_1, v_7 => a_7로 바꾸어서 Shortest Path Problem 2 (Algorithm 3.4)를 다시 해결하도록 한다. Matrix D와 P를 구하시오.

4. Algorithm 3.5를 적용하여 노드 a_7에서 a_4로 가는 shortest path를 찾고, 그 결과를 문제 2에서 구한 결과와 비교하시오.

5. Create the optimal binary search tree for the following items, where the probability occurrence of each word is given in parentheses: CASE (.05), ELSE (.15), END (.05), IF (.35), OF (.05), THEN (.35).

<1 번과 2 번> Floyd's Algorithm.

(1) 소스 코드

```c
#include <stdio.h>
#define SIZE 7
#define Far 100

int W[][SIZE] = {                       //한 번에 갈 수 없는 경우는 Far로 설정, 100
        {0,4,Far,Far,Far,10,Far},
        {3,0,Far,18,Far,Far,Far},
        {Far,6,0,Far,Far,Far,Far},
        {Far,5,15,0,2,19,6},
        {Far,Far,12,1,0,Far,Far},
        {Far,Far,Far,Far,Far,0,10},
        {Far,Far,Far,8,Far,Far,0}
    };

int D[SIZE][SIZE];
int P[SIZE][SIZE];

void path(int q, int r) {               //최단 경로를 출력하는 path 함수
    if (P[q-1][r-1] != 0) {             //path(7,3)=6
        path(q, P[q-1][r-1]);           //path(7,6)=4
        printf("v%d -> ", P[q-1][r-1]); //path(7,4)=0
        path(P[q-1][r-1], r);           //따라서 7->6->4->3
    }
}

int main(void) {

int i, j, k;
for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
        P[i][j] = 0;                    //P 초기화
        D[i][j] = W[i][j];              //W 배열 값을 모두 D 배열에 저장.
    }
}


for (k = 0; k < SIZE; k++) {
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < SIZE; j++) {
            if (D[i][k] + D[k][j] < D[i][j]) {          //v(i+1)에서 v(j+1)까지 가는데 v(k+1)를 거쳐 가는게 빠른 경우
                P[i][j] = k+1;
                D[i][j] = D[i][k] + D[k][j];
                printf("k: %d, i: %d, j: %d  | D[i][j]: %d, D[i][k]: %d, D[k][j]: %d \n", k, i, j, D[i][j], D[i][k], D[k][j]);
            }
        }
    }
}

printf("\n D 결과 \n");                      //D결과 출력
for (int a = 0; a < SIZE; a++) {
    for (int b = 0; b < SIZE; b++) {
        printf("%d ", D[a][b]);
    }
    printf("\n");
}

printf("\n P 결과 \n");                      //P결과 출력
for (int a = 0; a < SIZE; a++) {
    for (int b = 0; b < SIZE; b++) {
        printf("%d ", P[a][b]);
    }
    printf("\n");
}

int start = 7;
int end = 3;
printf("\n v7에서 v3까지의 최단 경로 \n");
printf("v%d -> ", start);
path(start, end);
printf("v%d \n ", end);

return 0;
}
```

<1 번과 2 번> Floyd's Algorithm.

(2) 실행 결과

```
k: 0, i: 1, j: 5  | D[i][j]: 13, D[i][k]: 3, D[k][j]: 10
k: 1, i: 0, j: 3  | D[i][j]: 22, D[i][k]: 4, D[k][j]: 18
k: 1, i: 2, j: 0  | D[i][j]: 9, D[i][k]: 6, D[k][j]: 3
k: 1, i: 2, j: 3  | D[i][j]: 24, D[i][k]: 6, D[k][j]: 18
k: 1, i: 2, j: 5  | D[i][j]: 19, D[i][k]: 6, D[k][j]: 13
k: 1, i: 3, j: 0  | D[i][j]: 8, D[i][k]: 5, D[k][j]: 3
k: 1, i: 3, j: 5  | D[i][j]: 18, D[i][k]: 5, D[k][j]: 13
k: 2, i: 4, j: 0  | D[i][j]: 21, D[i][k]: 12, D[k][j]: 9
k: 2, i: 4, j: 1  | D[i][j]: 18, D[i][k]: 12, D[k][j]: 6
k: 2, i: 4, j: 5  | D[i][j]: 31, D[i][k]: 12, D[k][j]: 19
k: 3, i: 0, j: 2  | D[i][j]: 37, D[i][k]: 22, D[k][j]: 15
k: 3, i: 0, j: 4  | D[i][j]: 24, D[i][k]: 22, D[k][j]: 2
k: 3, i: 0, j: 6  | D[i][j]: 27, D[i][k]: 22, D[k][j]: 5
k: 3, i: 1, j: 2  | D[i][j]: 33, D[i][k]: 18, D[k][j]: 15
k: 3, i: 1, j: 4  | D[i][j]: 20, D[i][k]: 18, D[k][j]: 2
k: 3, i: 1, j: 6  | D[i][j]: 23, D[i][k]: 18, D[k][j]: 5
k: 3, i: 2, j: 4  | D[i][j]: 26, D[i][k]: 24, D[k][j]: 2
k: 3, i: 2, j: 6  | D[i][j]: 29, D[i][k]: 24, D[k][j]: 5
k: 3, i: 4, j: 0  | D[i][j]: 9, D[i][k]: 1, D[k][j]: 8
k: 3, i: 4, j: 1  | D[i][j]: 6, D[i][k]: 1, D[k][j]: 5
k: 3, i: 4, j: 5  | D[i][j]: 19, D[i][k]: 1, D[k][j]: 18
k: 3, i: 4, j: 6  | D[i][j]: 6, D[i][k]: 1, D[k][j]: 5
k: 3, i: 6, j: 0  | D[i][j]: 16, D[i][k]: 8, D[k][j]: 8
k: 3, i: 6, j: 1  | D[i][j]: 13, D[i][k]: 8, D[k][j]: 5
k: 3, i: 6, j: 2  | D[i][j]: 23, D[i][k]: 8, D[k][j]: 15
k: 3, i: 6, j: 4  | D[i][j]: 10, D[i][k]: 8, D[k][j]: 2
k: 3, i: 6, j: 5  | D[i][j]: 26, D[i][k]: 8, D[k][j]: 18
k: 4, i: 0, j: 2  | D[i][j]: 36, D[i][k]: 24, D[k][j]: 12
k: 4, i: 1, j: 2  | D[i][j]: 32, D[i][k]: 20, D[k][j]: 12
k: 4, i: 3, j: 2  | D[i][j]: 14, D[i][k]: 2, D[k][j]: 12
k: 4, i: 6, j: 2  | D[i][j]: 22, D[i][k]: 10, D[k][j]: 12
k: 5, i: 0, j: 6  | D[i][j]: 20, D[i][k]: 10, D[k][j]: 10
k: 6, i: 5, j: 0  | D[i][j]: 26, D[i][k]: 10, D[k][j]: 16
k: 6, i: 5, j: 1  | D[i][j]: 23, D[i][k]: 10, D[k][j]: 13
k: 6, i: 5, j: 2  | D[i][j]: 32, D[i][k]: 10, D[k][j]: 22
k: 6, i: 5, j: 3  | D[i][j]: 18, D[i][k]: 10, D[k][j]: 8
k: 6, i: 5, j: 4  | D[i][j]: 20, D[i][k]: 10, D[k][j]: 10

 D 결과
0 4 36 22 24 10 20
3 0 32 18 20 13 23
9 6 0 24 26 19 29
8 5 14 0 2 18 5
9 6 12 1 0 19 6
26 23 32 18 20 0 10
16 13 22 8 10 26 0

 P 결과
0 0 5 2 4 0 6
0 0 5 0 4 1 4
2 0 0 2 4 2 4
2 0 5 0 0 2 0
4 4 0 0 0 4 4
7 7 7 7 7 0 0
4 4 5 0 4 4 0

 v7에서 v3까지의 최단 경로
v7 -> v4 -> v5 -> v3
```

D와 P를 구성하는 step

완성된 D와 P 배열.

V7에서 v3까지 최단 경로

<3 번과 4 번> Floyd's Algorithm.

(1) 소스 코드

```c
#include <stdio.h>
#define SIZE 7
#define Far 100

int W[][SIZE] = {                       //한 번에 갈 수 없는 경우는 Far로 설정, 100
        {0,Far,Far,Far,Far,Far,10},
        {Far,0,1,12,Far,Far,Far},
        {19,2,0,15,6,Far,6},
        {Far,Far,Far,0,6,Far,Far},
        {Far,Far,18,Far,0,3,Far},
        {10,Far,Far,Far,4,0,Far},
        {Far,Far,8,Far,Far,Far,0}
};

int D[SIZE][SIZE];
int P[SIZE][SIZE];

void path(int q, int r) {               //최단 경로를 출력하는 path 함수
    if (P[q - 1][r - 1] != 0) {             //path(7,4) = 3
        path(q, P[q - 1][r - 1]);           //path(3,4) = 2
        printf("a%d -> ", P[q - 1][r - 1]);     //path(7,3) = 0
        path(P[q - 1][r - 1], r);           //따라서 7->3->2->4
    }
}

int main(void) {

    int i, j, k;
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < SIZE; j++) {
            P[i][j] = 0;                    //P 초기화
            D[i][j] = W[i][j];              //W 배열 값을 모두 D 배열에 저장.
        }
    }


    for (k = 0; k < SIZE; k++) {
        for (i = 0; i < SIZE; i++) {
            for (j = 0; j < SIZE; j++) {
                if (D[i][k] + D[k][j] < D[i][j]) {              //a(i+1)에서 a(j+1)까지 가는데 a(k+1)를 거쳐 가는게 빠른 경우
                    P[i][j] = k + 1;
                    D[i][j] = D[i][k] + D[k][j];
                    printf("k: %d, i: %d, j: %d  | D[i][j]: %d, D[i][k]: %d, D[k][j]: %d \n", k, i, j, D[i][j], D[i][k], D[k][j]);
                }
            }
        }
    }

    printf("\n D 결과 \n");                          //D결과 출력
    for (int a = 0; a < SIZE; a++) {
        for (int b = 0; b < SIZE; b++) {
            printf("%d ", D[a][b]);
        }
        printf("\n");
    }

    printf("\n P 결과 \n");                          //P결과 출력
    for (int a = 0; a < SIZE; a++) {
        for (int b = 0; b < SIZE; b++) {
            printf("%d ", P[a][b]);
        }
        printf("\n");
    }

    int start = 7;
    int end = 4;
    printf("\n a7에서 a4까지의 최단 경로 \n");
    printf("a%d -> ", start);
    path(start, end);
    printf("a%d \n ", end);

    return 0;
}
```

<3 번과 4 번> Floyd's Algorithm.

(2) 실행 결과

```
k: 0, i: 5, j: 6  | D[i][j]: 20, D[i][k]: 10, D[k][j]: 10
k: 1, i: 2, j: 3  | D[i][j]: 14, D[i][k]: 2, D[k][j]: 12
k: 2, i: 1, j: 0  | D[i][j]: 20, D[i][k]: 1, D[k][j]: 19
k: 2, i: 1, j: 4  | D[i][j]: 6, D[i][k]: 1, D[k][j]: 5
k: 2, i: 1, j: 6  | D[i][j]: 6, D[i][k]: 1, D[k][j]: 5
k: 2, i: 4, j: 0  | D[i][j]: 37, D[i][k]: 18, D[k][j]: 19
k: 2, i: 4, j: 1  | D[i][j]: 20, D[i][k]: 18, D[k][j]: 2
k: 2, i: 4, j: 3  | D[i][j]: 32, D[i][k]: 18, D[k][j]: 14
k: 2, i: 4, j: 6  | D[i][j]: 23, D[i][k]: 18, D[k][j]: 5
k: 2, i: 6, j: 0  | D[i][j]: 27, D[i][k]: 8, D[k][j]: 19
k: 2, i: 6, j: 1  | D[i][j]: 10, D[i][k]: 8, D[k][j]: 2
k: 2, i: 6, j: 3  | D[i][j]: 22, D[i][k]: 8, D[k][j]: 14
k: 2, i: 6, j: 4  | D[i][j]: 13, D[i][k]: 8, D[k][j]: 5
k: 4, i: 1, j: 5  | D[i][j]: 9, D[i][k]: 6, D[k][j]: 3
k: 4, i: 2, j: 5  | D[i][j]: 8, D[i][k]: 5, D[k][j]: 3
k: 4, i: 3, j: 0  | D[i][j]: 43, D[i][k]: 6, D[k][j]: 37
k: 4, i: 3, j: 1  | D[i][j]: 26, D[i][k]: 6, D[k][j]: 20
k: 4, i: 3, j: 2  | D[i][j]: 24, D[i][k]: 6, D[k][j]: 18
k: 4, i: 3, j: 5  | D[i][j]: 9, D[i][k]: 6, D[k][j]: 3
k: 4, i: 3, j: 6  | D[i][j]: 29, D[i][k]: 6, D[k][j]: 23
k: 4, i: 5, j: 1  | D[i][j]: 24, D[i][k]: 4, D[k][j]: 20
k: 4, i: 5, j: 2  | D[i][j]: 22, D[i][k]: 4, D[k][j]: 18
k: 4, i: 5, j: 3  | D[i][j]: 36, D[i][k]: 4, D[k][j]: 32
k: 4, i: 6, j: 5  | D[i][j]: 16, D[i][k]: 13, D[k][j]: 3
k: 5, i: 1, j: 0  | D[i][j]: 19, D[i][k]: 9, D[k][j]: 10
k: 5, i: 2, j: 0  | D[i][j]: 18, D[i][k]: 8, D[k][j]: 10
k: 5, i: 3, j: 0  | D[i][j]: 19, D[i][k]: 9, D[k][j]: 10
k: 5, i: 4, j: 0  | D[i][j]: 13, D[i][k]: 3, D[k][j]: 10
k: 5, i: 6, j: 0  | D[i][j]: 26, D[i][k]: 16, D[k][j]: 10
k: 6, i: 0, j: 1  | D[i][j]: 20, D[i][k]: 10, D[k][j]: 10
k: 6, i: 0, j: 2  | D[i][j]: 18, D[i][k]: 10, D[k][j]: 8
k: 6, i: 0, j: 3  | D[i][j]: 32, D[i][k]: 10, D[k][j]: 22
k: 6, i: 0, j: 4  | D[i][j]: 23, D[i][k]: 10, D[k][j]: 13
k: 6, i: 0, j: 5  | D[i][j]: 26, D[i][k]: 10, D[k][j]: 16

 D 결과
0 20 18 32 23 26 10
19 0 1 12 6 9 6
18 2 0 14 5 8 5
19 26 24 0 6 9 29
13 20 18 32 0 3 23
10 24 22 36 4 0 20
26 10 8 22 13 16 0

 P 결과
0 7 7 7 7 7 0
6 0 0 0 3 5 3
6 0 0 2 0 5 0
6 5 5 0 0 5 5
6 3 0 3 0 0 3
0 5 5 5 0 0 1
6 3 0 3 3 5 0

 a7에서 a4까지의 최단 경로
a7 -> a3 -> a2 -> a4
```

D와 P를 구성하는 step

완성된 D와 P 배열.

a7에서 a4까지 최단 경로

<2 번과 4 번> 비교하기.

2 번에서는 path 를 계산할 때, v7 – v4 – v5- v3

path(7,3)=5 | path(5,3)=0

path(7,5)=4 | path(4,5)=0

path(7,4)=0

path 함수의 두번째 path 가 0 이 나옴을 확인할 수 있다.

⇨ 좀 더 간단하다고 볼 수 있다.

4 번에서는 path 를 계산할 때, a7 - a3 - a2 - a4

path(7,4)=3 | path(3,4)=2

path(3,2)=0 | path(2,4)=0

…

path 함수의 두번째 path 가 0 이 나오지 않음을 확인할 수 있다.

⇨ 7 과 4 사이에 3 이 있고 또, 3 과 4 사이에 2 가 있다.

⇨ 좀 더 확인할 것이 많다고 할 수 있다.