8장. 응용예제 답

응용예제 11

미로를 탈출하라!

maze.c

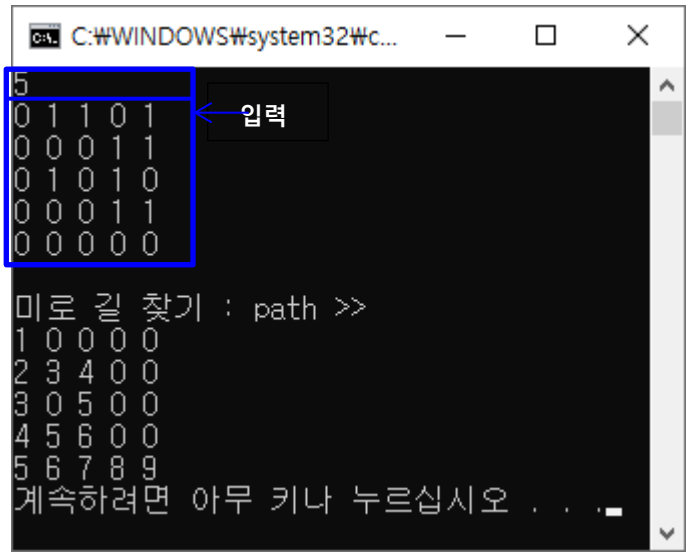
```
//응용예제11. 미로 길 찾기
#define _CRT_SECURE_NO_WARNINGS  //scanf 보안 경고 해제
#pragma warning(disable : 6031) //scanf의 반환값없음warning C6031을 해제하기 위해 추가.
#include<stdio.h>
#include <string.h>
#define MAX 30

typedef struct cell {  // 미로의 cell의 x,y좌표를 구조체로 정의
        int x, y;  //cell의 x,y좌표
} cell;

typedef cell element;     // 구조체cell을 연결 큐 원소(element)형으로 타입선언

typedef struct QNode {  // 연결 큐의 노드를 구조체로 정의
        element data;   //data필드는 미로 셀의 (x, y) 좌표가 됨
        struct QNode* link;
} QNode;

typedef struct {         // 연결 큐에서 사용하는 포인터 front와 rear를 구조체로 정의
        QNode* front, * rear;
} LQueueType;

int N; //미로 크기
int maze[MAX][MAX] = {0}; // 미로
int visited[MAX][MAX] = { 0 }; // 방문표시
int path[MAX][MAX] = {0}; // 시작점으로부터의 거리 표시
//미로 셀에서 이동할 수 있는 4가지 경우 {(1,0), (0,1), (-1,0), (0,-1)}
int dx[4] = { 1, 0, -1, 0 }; //미로 셀에서 이동할 수 있는 4가지 경우의 x 좌표 값
int dy[4] = { 0, 1, 0, -1 }; //미로 셀에서 이동할 수 있는 4가지 경우의 y 좌표 값
int cnt=1; //미로 셀의 방문 순서

LQueueType* createLinkedQueue(void);
int isLQEmpty(LQueueType* LQ);
void enLQueue(LQueueType* LQ, element item);
element deLQueue(LQueueType* LQ);
void maze_BFS(int x, int y);

int main(void) {
        int i, j;
        //1. 입력
        //1) 미로의 크기 N 입력
```

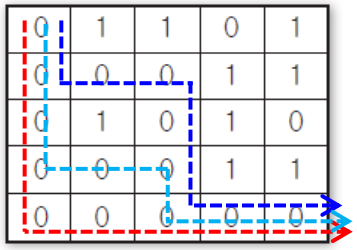
```
	scanf("%d", &N);
	//2) 미로 지도 입력 ( 0:열림, 1:막힘 )
	for (i = 0; i < N; i++)
		for (j = 0; j < N; j++) {
			scanf("%d", &maze[i][j]);
		}

	maze_BFS(0, 0); //BFS 기법으로 미로 길 찾기
	/*
	printf("₩nvisited : BFS 순서 >> ₩n");
	for (i = 0; i < N; i++) {
		for (j = 0; j < N; j++) {
			printf("%d ", visited[i][j]); //BFS 방문 순서 출력
		}	printf("₩n");
	}
	printf("₩n₩n미로 길 찾기 : BFS 거리 = %d ₩n", cnt);
	*/
	printf("₩n미로 길 찾기 : path >> ₩n");
	for (i = 0; i < N; i++) {
		for (j = 0; j < N; j++) {
			printf("%d ", path[i][j]);  //미로에 입구(1)부터 출구까지 path 출력
		}	printf("₩n");
	}

	printf("₩n미로 경로 길이: %d₩n₩n", path[N-1][N-1]);
	return 0;
}

void maze_BFS(int x, int y)
{
	int i = 0;
	cell current, next;
	QNode* tmp = NULL;
	LQueueType* maze_Q;			// 큐
	maze_Q = createLinkedQueue();	// 큐 생성

	current.x = x;  //시작 셀의 x 좌표 저장
	current.y = y;  //시작 셀의 y 좌표 저장
	path[current.x][current.y] = 1; //시작 셀은 길 찾기 순서 1

	visited[current.x][current.y] = cnt; // 미로 셀에 대한 방문 순서 저장
	enLQueue(maze_Q, current);		// 시작 위치를 큐에 삽입

	while (!isLQEmpty(maze_Q)) // 큐가 공백이 될때까지
	{
		current = deLQueue(maze_Q);

		for (i = 0; i < 4; i++)
		{
			next.x = current.x + dx[i];
			next.y = current.y + dy[i];

			if (next.x >= 0 && next.x < N && next.y >= 0 && next.y < N) // 미로의 범위
			{
```

```
                                // 이동할 셀이 존재하고 이전에 방문했던 적이 없는 경우
                                if (maze[next.x][next.y] == 0 && visited[next.x][next.y] == 0)
                                {
                                        path[next.x][next.y] = path[current.x][current.y] + 1;
                                        visited[next.x][next.y] = ++cnt;
                                        enLQueue(maze_Q, next);  //이동할 위치 next를 큐에 저장
                                }
                        }
                }
        }
}

//<<  큐 연산  >>
// 공백 연결 큐를 생성하는 연산
LQueueType* createLinkedQueue(void) {
        LQueueType* LQ;
        LQ = (LQueueType*)malloc(sizeof(LQueueType));
        LQ->front = NULL;
        LQ->rear = NULL;
        return LQ;
}

// 연결 큐가 공백 상태인지 검사하는 연산
int isLQEmpty(LQueueType* LQ) {
        if (LQ->front == NULL) {
                //printf(" Linked Queue is empty! ");
                return 1;
        }
        else return 0;
}

// 연결 큐의 rear에 원소를 삽입하는 연산
void enLQueue(LQueueType* LQ, element item) {
        QNode* newNode = (QNode*)malloc(sizeof(QNode));
        newNode->data = item;
        newNode->link = NULL;
        if (LQ->front == NULL) { // 현재 연결 큐가 공백 상태인 경우
                LQ->front = newNode;
                LQ->rear = newNode;
        }
        else {  // 현재 연결 큐가 공백 상태가 아닌 경우
                LQ->rear->link = newNode;
                LQ->rear = newNode;
        }
}

// 연결 큐에서 원소를 삭제하고 반환하는 연산
element deLQueue(LQueueType* LQ) {
        QNode* old = LQ->front;
        element item;
        if (isLQEmpty(LQ)) return;
        else {
                item = old->data;
                LQ->front = LQ->front->link;
                if (LQ->front == NULL)
                        LQ->rear = NULL;
                free(old);
```

```
                return item;
        }
}
```

[실행 화면]





응용예제 12

화장실이 급해요, 급해!

예제 8-5 (다익스트라 알고리즘)를 수정하여 작성

Version 1.

dijkstra.h (수정한 부분은 빨간색으로 표시)

```
#pragma once
#define TRUE  1
#define FALSE 0
#define INF  10000          // 무한대 값
#define MAX_VERTICES  50 // 그래프의 정점 개수

int distance[MAX_VERTICES];      // 시작 정점으로부터의 최단 경로 길이 저장
int S[MAX_VERTICES];                              // 정점의 집합 S
int nextVertex(int n);
int printStep(int step, int n);
void Dijkstra_shortestPath(int(*weight)[50], int start, int n);
```

dijkstra.c (수정한 부분은 빨간색으로 표시)

```
#include <stdio.h>
#include "dijkstra.h"

// 최소 거리를 갖는 다음 정점을 찾는 연산
int nextVertex(int n) {
        int i, min, minPos;
        min = INF;
        minPos = -1;
        for (i = 0; i < n; i++)
                if ((distance[i] < min) && !S[i]) {
                        min = distance[i];
                        minPos = i;
                }
        return minPos;
}

// 최단 경로 구하는 과정을 출력하는 연산
int printStep(int step, int end) {
        int i;
        printf("₩n %3d 단계 : S={", step);
        for (i = 0; i < end; i++)
                if (S[i] == TRUE)
                        printf("%3c", i + 65);

        if (step < 1) printf(" } ₩t₩t₩t");
        else if (step < 4) printf(" } ₩t₩t");
        else printf(" } ₩t");
        printf(" distance :[ ");
        for (i = 0; i < end; i++)
                if (distance[i] == INF)
```

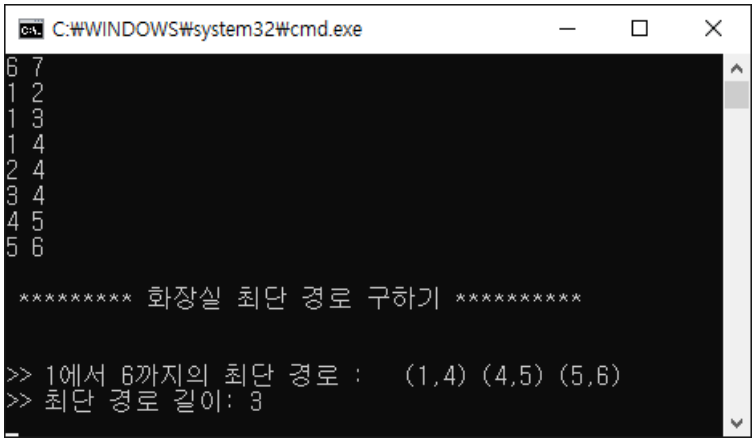
```
                        printf("%4c", '*');
                else printf("%4d", distance[i]);
        printf("%4c", ']');
        return ++step;
}
// [알고리즘 8-3] 구현
void Dijkstra_shortestPath(int (* weight)[50], int start, int n) {
        int i, u, w, step = 0;
        char path[MAX_VERTICES] = { 0 };  int p = 0; //최단 경로 저장용 변수

        for (i = 0; i < n; i++) {// 초기화
                distance[i] = weight[start][i];
                S[i] = FALSE;
        }

        S[start] = TRUE;                // 시작 정점을 집합 S에 추가
        distance[start] = 0;            // 시작 정점의 최단경로를 0으로 설정
        path[p++] = start;  //path[p++] = start + 65;

        //step = printStep(0, n);       // 0단계 상태를 출력 ////////////////////////

        for (i = 0; i < n - 1; i++) {
                u = nextVertex(n);              // 최단 경로를 만드는 다음 정점 u 찾기
                S[u] = TRUE;                    // 정점 u를 집합 S에 추가
                for (w = 0; w < n; w++)
                        if (!S[w])                      // 집합 S에 포함되지 않은 정점 중에서
                                if (distance[u] + weight[u][w] < distance[w]) {
                                        distance[w] = distance[u] + weight[u][w]; // 경로 길이 수정
                                        path[p++] = u;  //최단경로 정점 저장
                                }
                //step = printStep(step, n);    // 현재 단계 출력 ////////////////////////
        } path[p] = (n-1);

        //printf("₩n₩n>> %c에서 %c까지의 최단 경로 : ", start + 65, (n-1)+65);
        printf("₩n₩n>> %d에서 %d까지의 최단 경로 :  ", start+1, n);
        for (i = 0; i <= p; i++) {
                if(i != p)
                        //printf(" (%c,%c)", path[i]+65, path[i+1] + 65);  // 정점을 문자로 출력
                    printf(" (%d,%d)", path[i]+1, path[i+1]+1);  //정점을 숫자로 출력 (정점 A = 정점
1)
        }
        printf("₩n>> 최단 경로 길이: %d₩n", distance[n-1]);
}
```

findToilet.c

```
//응용예제12. 화장실 최단 경로 찾기
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "dijkstra.h"
#define MAX 50

typedef struct edge {
```

```
        int n1, n2;  //길이 있는 두 지점
} edge;

int main(void) {
        int i, j;
        int N, M;  //화장실 개수 N,  길의 개수 M
        edge road[MAX] = {0};
        int weight[MAX][MAX] = {0};

        //1. 입력: 첫째 줄 => 화장실 개수 N,  길의 개수 M
        scanf("%d %d", &N, &M);
        //2. 입력: 둘째 줄 ~ M+1번째 줄 => 길이 있는 두 지점
        for (i = 0; i < M; i++) {
                scanf("%d %d", &road[i].n1, &road[i].n2);
        }

        //3. 입력값을 이용하여 가중치 인접 행렬 만들기
        for (i = 0; i < M; i++) { //길이 있는 M개의 두 지점에 대해 1 저장
                weight[(road[i].n1)-1][(road[i].n2)-1] = 1;
                weight[(road[i].n2)-1][(road[i].n1)-1] = 1;
        }
        for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                        if ((i != j) && (weight[i][j] ==0))  //길이 없는 두 지점에 대해 INF 저장
                                weight[i][j] = INF;

        ////4. 완성된 가중치 인접 행렬 출력하여 확인하기
        //printf("₩n ********** 가중치 인접 행렬 **********₩n₩n");
        //for(i = 0; i < N; i++) {
        //      for(j = 0; j < N; j++) {
        //              if (weight[i][j] == INF)
        //                      printf("%4c", '*');
        //              else printf("%4d", weight[i][j]);
        //      }
        //      printf("₩n₩n");
        //}

        printf("₩n ********* 화장실 최단 경로 구하기 **********₩n");
        Dijkstra_shortestPath(weight, 0, N);
        getch();  return 0;
}
```

[실행화면]



Version 2. 작업 과정 확인용으로 출력문 추가 (Version 1. 에서 출력문 주석 해제)

dijkstra.c

```
#include <stdio.h>
#include "dijkstra.h"

// 최소 거리를 갖는 다음 정점을 찾는 연산
int nextVertex(int n) {
        int i, min, minPos;
        min = INF;
        minPos = -1;
        for (i = 0; i < n; i++)
                if ((distance[i] < min) && !S[i]) {
                        min = distance[i];
                        minPos = i;
                }
        return minPos;
}

// 최단 경로 구하는 과정을 출력하는 연산
int printStep(int step, int end) {
        int i;
        printf("Wn %3d 단계 : S={", step);
        for (i = 0; i < end; i++)
                if (S[i] == TRUE)
                        printf("%3c", i + 65);

        if (step < 1) printf(" } WtWtWt");
        else if (step < 4) printf(" } WtWt");
        else printf(" } Wt");
        printf(" distance :[ ");
        for (i = 0; i < end; i++)
                if (distance[i] == INF)
                        printf("%4c", '*');
                else printf("%4d", distance[i]);
        printf("%4c", ']');
        return ++step;
}
// [알고리즘 8-3] 구현
void Dijkstra_shortestPath(int (* weight)[50], int start, int n) {
        int i, u, w, step = 0;
        char path[MAX_VERTICES] = { 0 };  int p = 0; //최단 경로 저장용 변수

        for (i = 0; i < n; i++) {// 초기화
                distance[i] = weight[start][i];
                S[i] = FALSE;
        }

        S[start] = TRUE;                // 시작 정점을 집합 S에 추가
        distance[start] = 0;            // 시작 정점의 최단경로를 0으로 설정
        path[p++] = start;
        step = printStep(0, n);         // 0단계 상태를 출력 ////////////////////////

        for (i = 0; i < n - 1; i++) {
                u = nextVertex(n);              // 최단 경로를 만드는 다음 정점 u 찾기
```

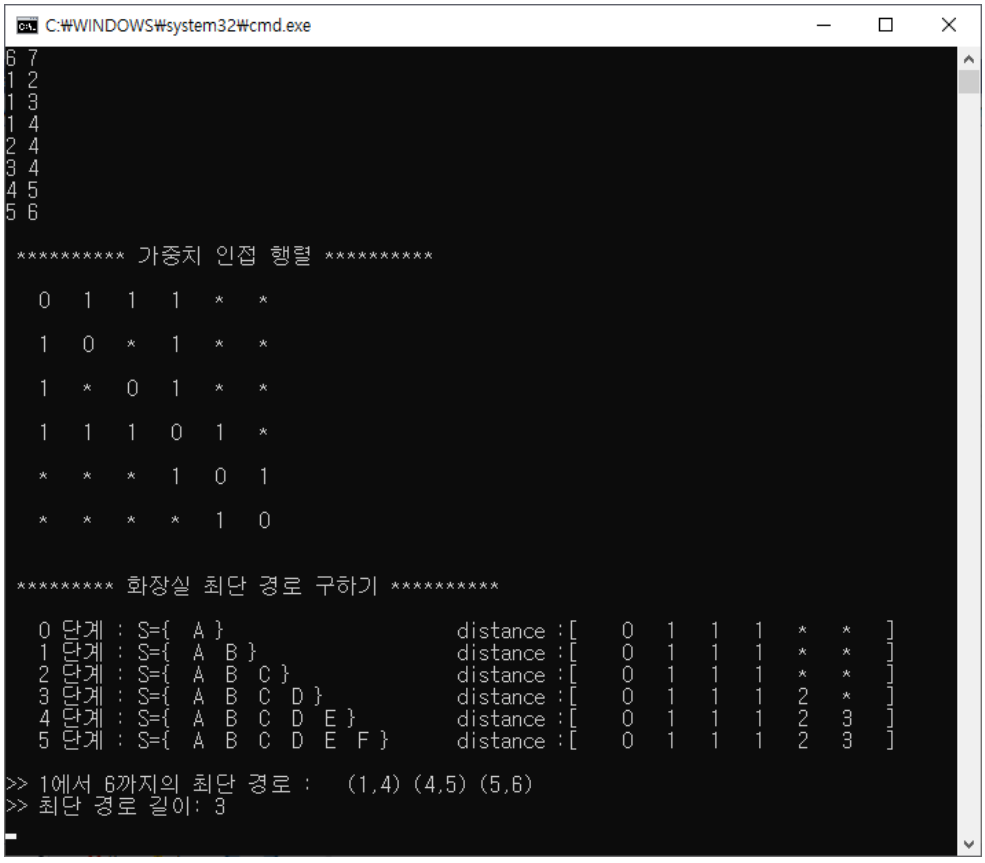
```
                S[u] = TRUE;                    // 정점 u를 집합 S에 추가
                for (w = 0; w < n; w++)
                        if (!S[w])                      // 집합 S에 포함되지 않은 정점 중에서
                                if (distance[u] + weight[u][w] < distance[w]) {
                                        distance[w] = distance[u] + weight[u][w]; // 경로 길이 수정

                                        path[p++] = u;  //최단경로 정점 저장
                                }
                step = printStep(step, n);      // 현재 단계 출력 ////////////////////////
        } path[p] = (n-1);

        //printf("₩n₩n>> %c에서 %c까지의 최단 경로 : ", start + 65, (n-1)+65);
        printf("₩n₩n>> %d에서 %d까지의 최단 경로 :  ", start+1, n);
        for (i = 0; i <= p; i++) {
                if(i != p)
                        //printf(" (%c,%c)", path[i]+65, path[i+1] + 65);  // 정점을 문자로 출력
                    printf(" (%d,%d)", path[i]+1, path[i+1]+1);  //정점을 숫자로 출력 (정점 A = 정점 1)
        }
        printf("₩n>> 최단 경로 길이: %d₩n", distance[n-1]);
}
```

findToilet.c

```
//응용예제12. 화장실 최단 경로 찾기
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "dijkstra.h"
#define MAX 50

typedef struct edge {
        int n1, n2;  //길이 있는 두 지점
} edge;

int main(void) {
        int i, j;
        int N, M;  //화장실 개수 N,  길의 개수 M
        edge road[MAX] = {0};
        int weight[MAX][MAX] = {0};

        //1. 입력: 첫째 줄 => 화장실 개수 N,  길의 개수 M
        scanf("%d %d", &N, &M);
        //2. 입력: 둘째 줄 ~ M+1번째 줄 => 길이 있는 두 지점
        for (i = 0; i < M; i++) {
                scanf("%d %d", &road[i].n1, &road[i].n2);
        }

        //3. 입력값을 이용하여 가중치 인접 행렬 만들기
        for (i = 0; i < M; i++) { //길이 있는 M개의 두 지점에 대해 1 저장
                weight[(road[i].n1)-1][(road[i].n2)-1] = 1;
                weight[(road[i].n2)-1][(road[i].n1)-1] = 1;
        }
        for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
```

```
                        if ((i != j) && (weight[i][j] ==0))  //길이 없는 두 지점에 대해 INF 저장
                                weight[i][j] = INF;

        //4. 완성된 가중치 인접 행렬 출력하여 확인하기
        printf("₩n ********** 가중치 인접 행렬 **********₩n₩n");
        for(i = 0; i < N; i++) {
                for(j = 0; j < N; j++) {
                        if (weight[i][j] == INF)
                                printf("%4c", '*');
                        else printf("%4d", weight[i][j]);
                }
                printf("₩n₩n");
        }

        printf("₩n ********* 화장실 최단 경로 구하기 **********₩n");
        Dijkstra_shortestPath(weight, 0, N);
        getch();  return 0;
}
```

[실행화면: version 2]



정점을 문자로 바꾸어 출력한 경우 :