**로고, 폰트, 텍스트, 상징이(가) 표시된 사진

자동 생성된 설명**

**[데이터구조2 리포트]**

**이름: 황지상 학번: 20224060**

**과제#4 인접행렬, 인접리스트를 이용한 그래프 표현과 BFS 구현**

1. **코드**

// 경고 메시지 무시 설정

#define \_CRT\_SECURE\_NO\_WARNINGS

// 필요한 라이브러리 포함

#include <stdio.h> // 표준 입력과 출력 함수 라이브러리 포함

#include <stdlib.h> // 표준 라이브러리 함수 라이브러리 포함

#include <time.h> // 시간 관련 함수 라이브러리 포함

// 큐의 최대 크기를 정의

#define MAX\_SIZE 10

// 논리적 참과 거짓을 나타내는 매크로 정의

#define TRUE 1

#define FALSE 0

// 정수를 element로 재정의

typedef int element;

// 큐 구조체 정의

typedef struct {

element queue[MAX\_SIZE]; // 요소를 저장하는 배열

int front, rear; // 큐의 전단과 후단을 나타내는 변수

} QueueType;

// 오류 메시지 출력 함수

void error(const char\* message) {

fprintf(stderr, "%s\n", message);

exit(1);

}

// 큐 초기화 함수

void queue\_init(QueueType\* q) {

q->front = q->rear = 0;

}

// 큐가 비어있는지 여부를 반환하는 함수

int is\_empty(QueueType\* q) {

return (q->front == q->rear);

}

// 큐가 가득 차 있는지 여부를 반환하는 함수

int is\_full(QueueType\* q) {

return ((q->rear + 1) % MAX\_SIZE == q->front);

}

// 큐에 요소를 추가하는 함수

void enqueue(QueueType\* q, element item) {

if (is\_full(q))

error("가득참");

q->rear = (q->rear + 1) % MAX\_SIZE;

q->queue[q->rear] = item;

}

// 큐에서 요소를 삭제하고 반환하는 함수

element dequeue(QueueType\* q) {

if (is\_empty(q))

error("공백상태");

q->front = (q->front + 1) % MAX\_SIZE;

return q->queue[q->front];

}

// 최대 정점 개수를 정의

#define MAX\_VERTICES 50

// 그래프 구조체 정의

typedef struct GraphType {

int n; // 정점의 개수를 나타내는 변수

int adj\_mat[MAX\_VERTICES][MAX\_VERTICES]; // 인접한 행렬을 나타내는 2차원 배열

} GraphType;

// 정점의 방문 여부를 저장하는 배열

int visited[MAX\_VERTICES];

// 그래프 초기화 함수

void graph\_init(GraphType\* g) {

int r, c;

g->n = 0;

// 인접 행렬의 모든 원소를 0으로 초기화

for (r = 0; r < MAX\_VERTICES; r++) {

for (c = 0; c < MAX\_VERTICES; c++) {

g->adj\_mat[r][c] = 0;

}

}

}

// 정점을 그래프에 추가하는 함수

void insert\_vertex(GraphType\* g, int v) {

if (((g->n) + 1) > MAX\_VERTICES) {

// 정점의 개수가 초과할 경우 에러 메시지 출력

fprintf(stderr, "그래프: 정점의 개수 초과");

return;

}

g->n++;

}

// 두 정점 사이에 간선을 삽입하는 함수

void insert\_edge(GraphType\* g, int start, int end) {

if (start >= g->n || end >= g->n) {

// 정점 번호가 유효 범위를 벗어나면 오류 메시지 출력

fprintf(stderr, "그래프: 정점의 번호 오류");

return;

}

g->adj\_mat[start][end] = 1; // start에서 end로 가는 간선을 표시

g->adj\_mat[end][start] = 1; // end에서 start로 가는 간선을 표시

}

// 너비 우선 탐색을 행하는 함수 (인접 행렬 사용)

void bfs\_mat(GraphType\* g, int v) {

int w;

QueueType q;

queue\_init(&q);

visited[v] = TRUE;

printf("%d 방문 ->", v + 1);

enqueue(&q, v);

while (!is\_empty(&q)) {

v = dequeue(&q);

for (w = 0; w < g->n; w++) {

if (g->adj\_mat[v][w] && !visited[w]) {

visited[w] = TRUE;

printf("%d 방문 -> ", w + 1);

enqueue(&q, w);

}

}

}

}

// 배열의 최대 크기를 정의

#define MAX 50

// 그래프 노드 구조체 정의

typedef struct GraphNode {

int vertex; // 정점 번호를 저장

struct GraphNode\* link; // 다음 정점을 연결하기 위한 포인터

} GraphNode;

// 인접 리스트 그래프 구조체 정의

typedef struct GraphType1 {

int n; // 정점의 개수를 저장

GraphNode\* adj\_list[MAX]; // 각 정점의 인접 리스트를 저장하는 배열

} GraphType1;

// 인접 리스트 그래프 초기화 함수

void init1(GraphType1\* g) {

int v;

g->n = 0; // 그래프의 정점 개수 초기화

for (v = 0; v < MAX; v++) {

g->adj\_list[v] = NULL; // 각 정점의 인접 리스트 초기화

}

}

// 정점을 인접 리스트에 추가하는 함수

void insert\_list\_vertex(GraphType1\* g, int v) {

if (((g->n) + 1) > MAX) {

fprintf(stderr, "그래프: 정점의 개수 초과"); // 정점 개수 초과 시 에러 출력

return;

}

g->n++; // 정점 개수 증가

}

// 간선을 인접 리스트에 추가하는 함수

void insert\_list\_edge(GraphType1\* g, int u, int v) {

GraphNode\* node;

if (u >= g->n || v >= g->n) {

fprintf(stderr, "그래프: 정점 번호 오류"); // 정점 번호 오류 시 에러 출력

return;

}

node = (GraphNode\*)malloc(sizeof(GraphNode)); // 새로운 노드 생성

node->vertex = v; // 노드에 정점 번호 저장

node->link = g->adj\_list[u]; // 노드를 인접 리스트에 연결

g->adj\_list[u] = node; // 정점 u의 인접 리스트의 시작을 새로운 노드로 갱신

}

// 인접 리스트를 출력하는 함수

void print\_adj\_list(GraphType1\* g) {

for (int i = 0; i < g->n; i++) {

GraphNode\* p = g->adj\_list[i]; // 현재 정점의 인접 리스트 시작을 가리키는 포인터

printf("정점 %d의 인접 리스트", i + 1); // 현재 정점 이름과 인접 리스트 시작을 출력

while (p != NULL) {

printf("-> %d ", p->vertex + 1); // 현재 인접 정점을 출력

p = p->link; // 다음 인접 정점으로 이동

}

printf("\n");

}

}

// 방문 여부를 저장하는 배열

int visited1[MAX\_VERTICES];

// 너비 우선 탐색을 행하는 함수 (인접 리스트 사용)

void bfs\_list(GraphType1\* g, int v) {

GraphNode\* w;

QueueType q;

queue\_init(&q);

visited1[v] = TRUE;

printf("%d 방문 -> ", v + 1);

enqueue(&q, v);

while (!is\_empty(&q)) {

v = dequeue(&q);

for (w = g->adj\_list[v]; w; w = w->link) {

if (!visited1[w->vertex]) {

visited1[w->vertex] = TRUE;

printf("%d 방문 -> ", w->vertex + 1);

enqueue(&q, w->vertex);

}

}

}

}

// 인접 행렬을 출력하는 함수

void print\_adj\_mat(GraphType\* g) {

for (int i = 0; i < g->n; i++) {

printf("%d ", i + 1); // 정점 레이블 출력

for (int j = 0; j < g->n; j++) {

printf("%d ", g->adj\_mat[i][j]); // 인접 행렬의 원소 출력

}

printf("\n");

}

}

int main(void) {

GraphType\* g;

GraphType1\* gg; // 인접 리스트를 사용할 그래프 포인터 선언

g = (GraphType\*)malloc(sizeof(GraphType));

gg = (GraphType1\*)malloc(sizeof(GraphType1)); // 인접 리스트 그래프 동적 할당

graph\_init(g);

init1(gg); // 인접 리스트 그래프 초기화 함수 호출

for (int i = 0; i < 8; i++) {

insert\_vertex(g, i); // 정점을 순차적으로 추가하고 정점 번호를 연속된 정수로 할당

}

insert\_edge(g, 0, 1);

insert\_edge(g, 1, 7);

insert\_edge(g, 7, 5);

insert\_edge(g, 7, 6);

insert\_edge(g, 5, 6);

insert\_edge(g, 0, 2);

insert\_edge(g, 2, 3);

insert\_edge(g, 2, 4);

insert\_edge(g, 3, 4);

printf("\n");

printf("인접 행렬 \n");

print\_adj\_mat(g);

printf("\n너비 우선 탐색 (인접 행렬)\n");

bfs\_mat(g, 0);

printf("\n");

printf("\n \n");

for (int i = 0; i < g->n; i++) {

insert\_list\_vertex(gg, i); // 인접 리스트 그래프에 정점 추가

}

for (int i = 0; i < g->n; i++) {

for (int j = 0; j < g->n; j++) {

if (g->adj\_mat[i][j] == 1) {

insert\_list\_edge(gg, i, j); // 인접 리스트 그래프에 무작위 간선 추가

}

}

}

printf("인접 리스트\n\n");

print\_adj\_list(gg); // 인접 리스트 그래프 출력

printf("\n\n너비 우선 탐색 (인접 리스트)\n");

bfs\_list(gg, 0);

free(g);

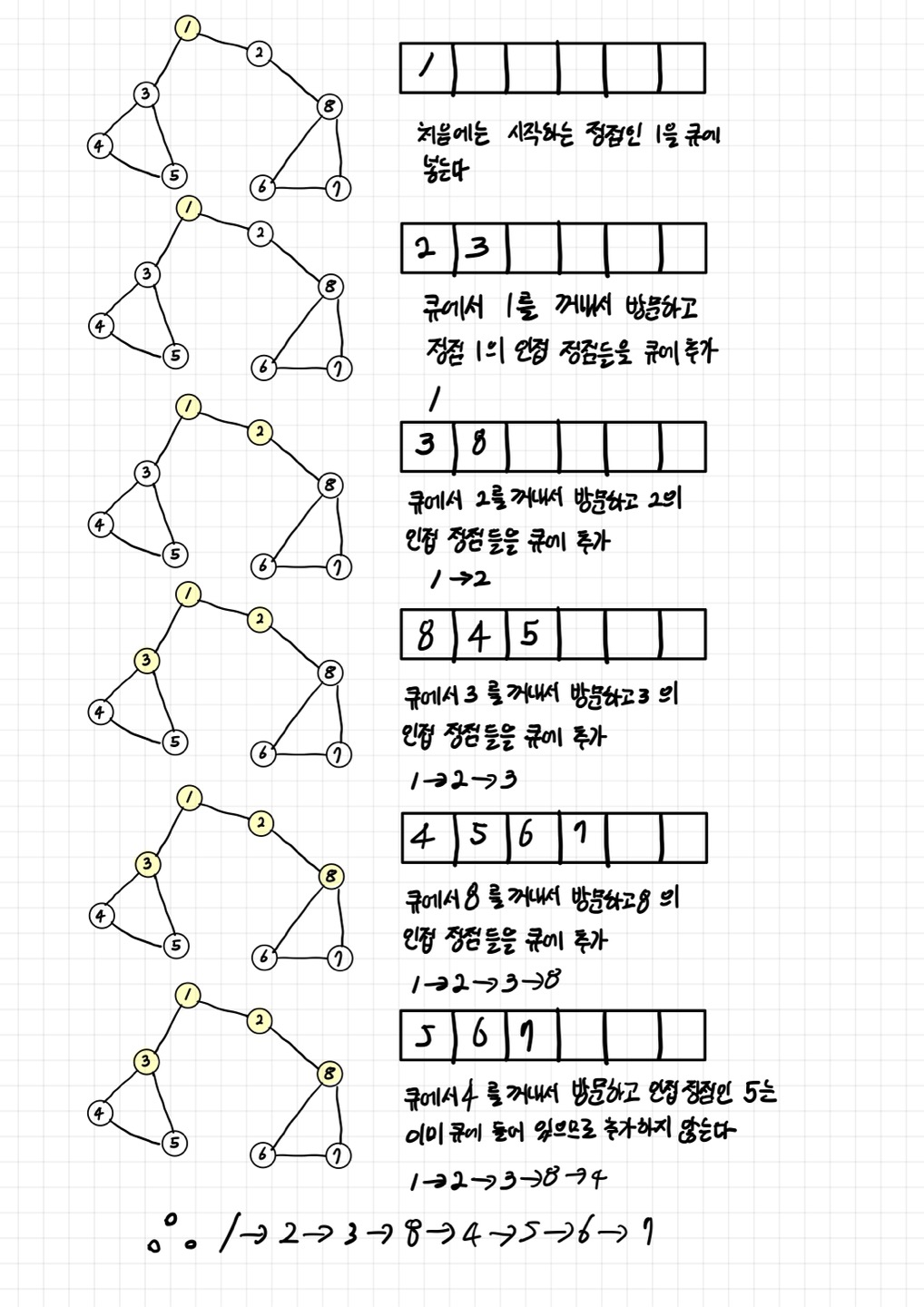
free(gg);

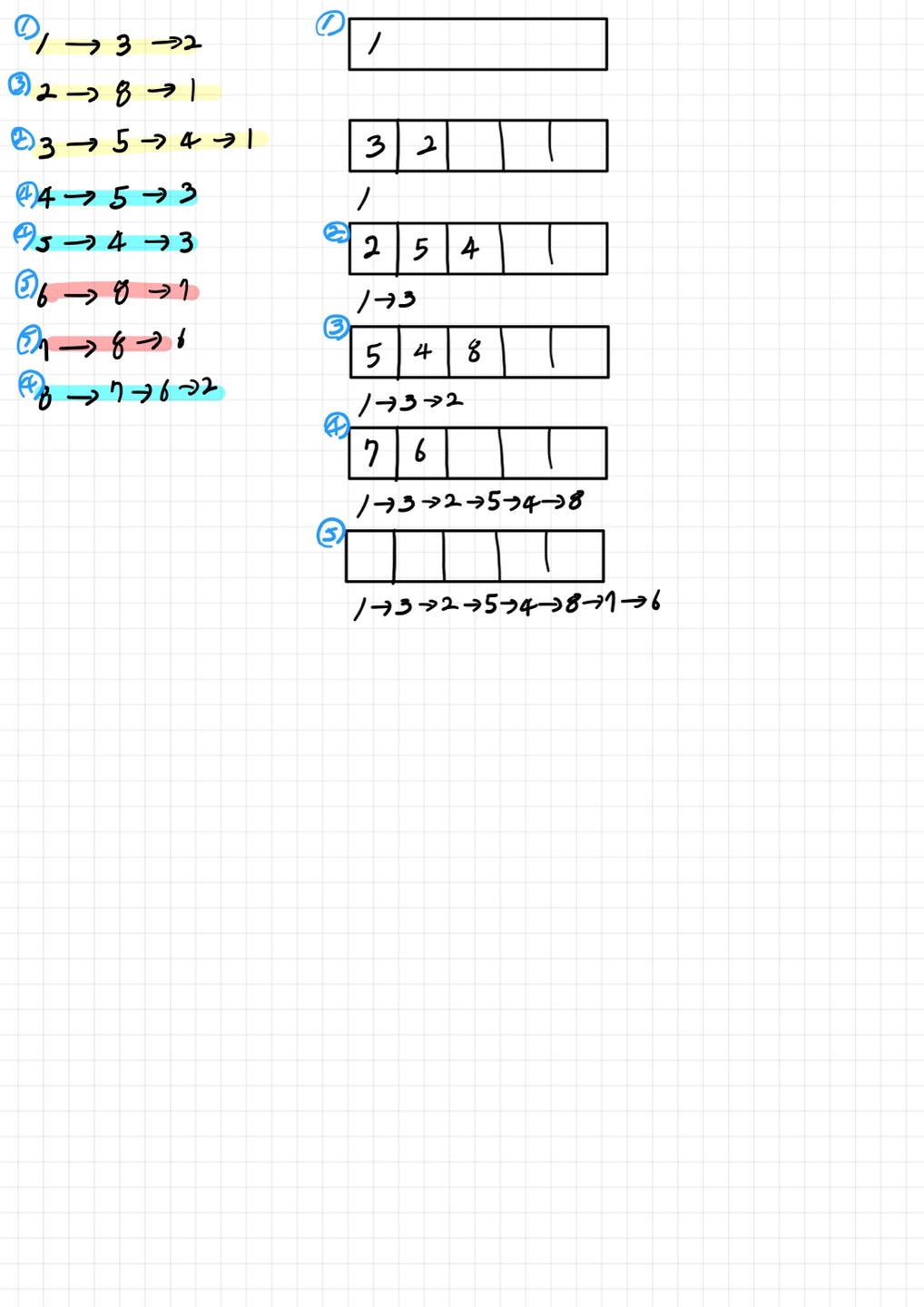
return 0;

}

1. **텍스트, 스크린샷, 디자인이(가) 표시된 사진

   자동 생성된 설명결과**

****

****