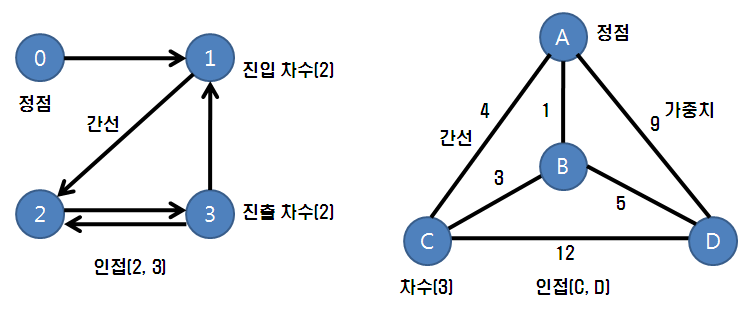
**———————————————————————————————————————————————————**

[1] 기초 문제

———————————————————————————————————————————————————

1. **다음 그림에서 찾을 수 있는 그래프의 용어들에 어떤 것이 있는지 나열하시오.**



1. **그래프 용어와 관련된 문제다. 빈 칸에 들어가는 단어를 맞춰보도록 한다.**
2. ① 정점 ② 간선 ③ 위치 ④ 노드
3. ① 간선 ② 정점 ③ 선 ④ 링크
4. ① 경로 ② 정점 ③ 순서 ④ 간선
5. **그래프를 사용하고 있는 곳을 세 가지씩만 나열해 보자.**
6. 지하철 노선도, 전기 회로, 수도 배관, 인맥, 도로 설계
7. 구글 맵, 네비게이션, 네트워크 트래픽, 지하철 요금 계산, 프로세스 자원 관리, 선수 과목

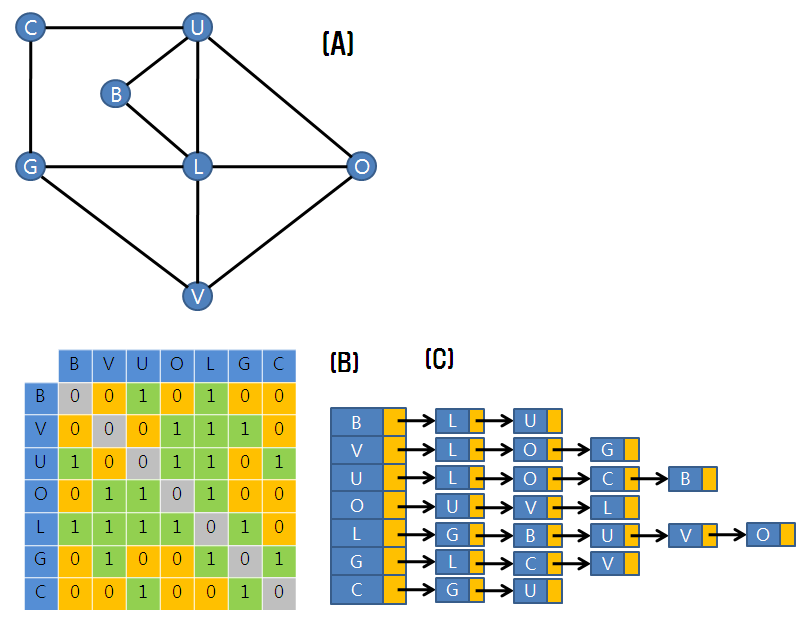
|  |
| --- |
| 해답  정답으로 간주할 수 있는 것들이 너무 많아서 이번 장에서 설명한 것만 실었습니다. |

1. **그래프에서 사용되는 두 가지 탐색 방법에 대해 비교해 본다.**

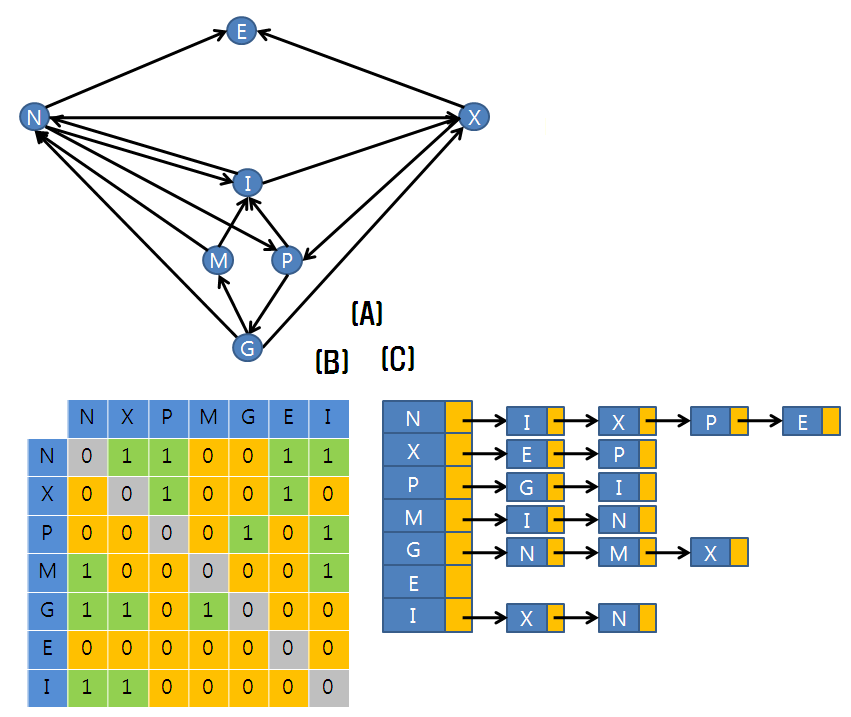
|  |  |  |
| --- | --- | --- |
|  | 깊이 우선 탐색 | 너비 우선 탐색 |
| 구현 | 재귀 함수 | 반복문 |
| 순서 | 아래쪽으로 먼저 진행 | 오른쪽으로 먼저 진행 |
| 추가 자료 구조 | 스택 | 큐 |

|  |
| --- |
| 해답  표에 대한 완전한 설명은 2장의 심화 학습에 있습니다. 설명이 너무 길어 기본적인 답안만 제공합니다. |

1. **무방향 그래프에 대해 다음과 같은 입력이 발생했다. 아래 질문에 답해보자.**



1. **방향 그래프에 대해 다음과 같은 입력이 발생했다. 아래 질문에 답해보자.**



1. **본문에 나왔던 인접 행렬에 대해 설명을 하는 곳이다. 중복되는 것 같아 인접 리스트는 싣지 않았다. 그래프를 배우려는 친구에게 설명한다는 생각으로 차분하게 기술해 보자.**

|  |
| --- |
| 해답  자세한 설명이 심화 학습과 인접 행렬 프로젝트에 있습니다. 설명이 너무 길어 답안은 생략했습니다. |

**———————————————————————————————————————————————————**

[2] 기본 실습 문제

———————————————————————————————————————————————————

1. **그래프에서 나올 수 있는 가벼운 코드들이다. 인접 리스트는 제외하고, 인접 행렬에 대해서만 구현하도록 한다.**

|  |
| --- |
| 해답  모두 그래프에 포함된 함수들이라서 프로젝트의 내용을 그대로 옮겼음을 알려드립니다. |

1. int isEmptyGraph(void)

{

return g\_count <= 0;

}

1. char getVertex(int index)

{

return g\_vertex[index];

}

1. void insertVertex(char name)

{

g\_vertex[g\_count] = name;

g\_count++;

}

1. int findVertex(char name)

{

int i;

for(i = 0; i < g\_count; i++)

{

if(g\_vertex[i] == name)

return i;

}

return -1;

}

1. **그래프에서 나올 수 있는 조금 무거운 코드들이다. 인접 리스트는 제외하고, 인접 행렬에 대해서만 구현하도록 한다.**

|  |
| --- |
| 해답  모두 그래프에 포함된 함수들이라서 프로젝트의 내용을 그대로 옮겼음을 알려드립니다. |

1. void printEdge(void)

{

int i, j;

printf("[정점+간선]\n");

for(i = 0; i < g\_count; i++)

{

printf("%c : ", g\_vertex[i]);

for(j = 0; j < g\_count; j++)

{

if(g\_matrix[i][j] == 1)

printf("(%c,%c) ", g\_vertex[i], g\_vertex[j]);

}

printf("\n");

}

printf("\n");

}

1. int deleteEdge(char v, char u)

{

int vi = findVertex(v),

ui = findVertex(u);

if(vi == -1 || ui == -1)

return 0;

g\_matrix[vi][ui] = g\_matrix[ui][vi] = 0;

return 1;

}

1. void printAdjacent(char name)

{

int i, find = findVertex(name);

if(find != -1)

{

for(i = 0; i < g\_count; i++)

{

if(g\_matrix[find][i] == 1)

printf("%c ", g\_vertex[i]);

}

printf("\n");

}

}

1. int deleteVertex(char name)

{

int i, find = findVertex(name);

if(find == -1)

return 0;

g\_vertex[find] = g\_vertex[--g\_count];

for(i = 0; i < g\_count; i++)

g\_matrix[i][find] = g\_matrix[find][i] = g\_matrix[i][g\_count];

g\_matrix[find][find] = 0;

return 1;

}

1. **저자는 여러분이 복잡한 코드까지 직접 구현하기를 바라지는 않는다. 대신 비어있는 곳의 코드를 채울 정도는 되어야 한다고 생각한다. 빈 칸에 들어가는 코드는 무엇인가? 두 함수 모두 인접 리스트 기반이다.**
   1. ① DepthFirstSearchReal(find, 0);

② g\_visit[index] = 1;

③ order = DepthFirstSearchReal(i, order+1);

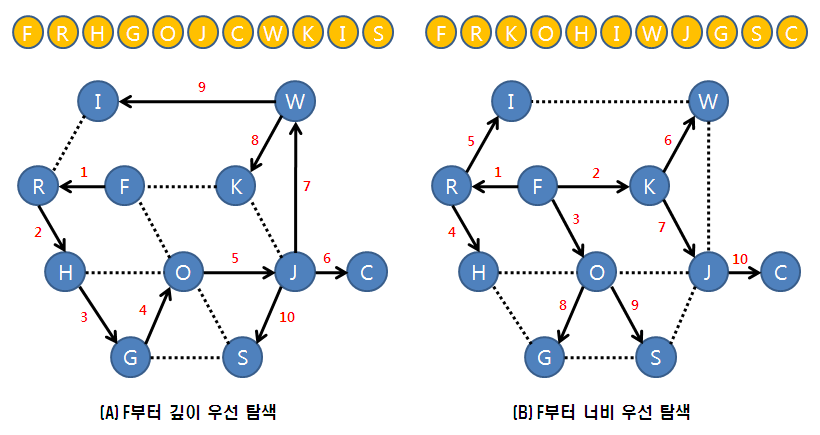
* 1. ① enQueue(find);

② while(!isEmptyQueue())

③ index = deQueue();

④ enQueue(i);

1. **11개의 정점과 17개의 간선으로 구성된 무방향 그래프가 있을 때, 아래 질문에 답해보자. 인접 행렬 방식으로 구현했고, 정점의 순서는 아래와 같다고 가정한다.**



**———————————————————————————————————————————————————**

[3] 응용 실습 문제

———————————————————————————————————————————————————

1. 인접 행렬에서 간선의 개수를 세는 함수입니다. 첫 번째는 행렬 전체에 대해 반복했고, 두 번째는 대칭성을 이용해서 절반만 반복합니다.

int edgeCountMaxtix(void)

{

int i, j, edge = 0;

for(i = 0; i < MAX\_VERTEX; i++)

for(j = 0; j < MAX\_VERTEX; j++)

{

if(g\_matrix[i][j] != 0)

edge++;

}

return edge/2;

}

int edgeCountMatrix(void)

{

int i, j, edge = 0;

for(i = 0; i < MAX\_VERTEX; i++)

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

edge += g\_matrix[i][j];

return edge;

}

인접 리스트는 모든 노드를 다 뒤져보는 방법밖에 없습니다.

int edgeCountList(void)

{

int i, edge = 0;

for(i = 0; i < MAX\_VERTEX; i++)

edge += countNode(g\_list+i);

return edge/2;

}

1. 깊이 우선으로 방문하는 함수입니다. 본문에서 만들었던 함수들을 그대로 사용했기 때문에 오히려 복잡해 보이는 부분이 있습니다. 정점이나 간선의 존재를 파악하는 부분은 함수를 호출할 필요가 없습니다. 찾으면 1을 반환하기 때문에 언제 재귀 함수를 멈출지 알 수 있습니다.

int printDepthFirstReal(char v, char u)

{

int i;

printf("%c ", v);

g\_visit[findVertex(v)] = 1;

if(v == u)

return 1;

for(i = 0; i < g\_count; i++)

{

if(g\_visit[i] == 0 && checkEdge(v, g\_list[i].data) == 1)

{

// 찾을 때만 반환합니다. 못 찾으면 나머지 반복문에서 찾아봅니다.

if(printDepthFirstReal(getVertex(i), u) == 1)

return 1;

}

}

return 0;

}

// 재귀 함수 시작을 위한 래퍼입니다.

int printDepthFirst(char v, char u)

{

if(findVertex(v) != -1 && findVertex(u) != -1)

{

int i;

for(i = 0; i < g\_count; i++)

g\_visit[i] = 0;

printDepthFirstReal(v, u);

return 1;

}

return 0;

}

너비 우선으로 방문하는 함수입니다. 본문에서처럼 int 큐를 사용했고, 기존 함수를 재사용했습니다. 너비 우선은 깊이 우선에 비해 어렵지 않습니다. 본문의 내용을 일부만 수정해서 구현할 수 있습니다.

int printWidthFirst(char v, char u)

{

int vi = findVertex(v),

ui = findVertex(u), i;

if(vi == -1 || ui == -1)

return 0;

for(i = 0; i < g\_count; i++)

g\_visit[i] = 0;

// 본문에는 없던 함수로 큐를 비웁니다. 이전 결과가 남아있을 수 있습니다.

clearQueue();

g\_visit[vi] = 1;

enQueue(vi);

while(!isEmptyQueue())

{

printf("%c ", v = g\_list[deQueue()].data);

if(v == u)

break;

for(i = 0; i < g\_count; i++)

{

if(g\_visit[i] == 0 && checkEdge(v, g\_list[i].data) == 1)

{

g\_visit[i] = 1;

enQueue(i);

}

}

}

return 1;

}

1. 본문과 똑같이 다중 파일 프로젝트로 처리했습니다. 이 코드의 핵심은 1부터 100까지의 합계를 계산하는 n(n+1)/2에 있습니다. 이 방법을 사용하면 삼각형 형태의 정적 배열을 만들 수 있습니다. 실제로는 1차원이지만, 함수를 통해 2차원처럼 사용할 수 있습니다.

본문에 나온 함수를 최대한 살리는 쪽으로 작업했는데, getIndex 함수 때문에 그렇습니다. 생각보다 함수들의 연관성이 강해서 없애지 못한 것도 있지만, g\_matrix 전역 배열과 연관 없는 함수가 많았습니다.

**// GraphExample.h**

#define MAX\_VERTEX 9 // 정점을 main 함수에서 사용하는 것과 똑같이 조절했습니다.

void init(void);

void insertVertex(char name);

int insertEdge(char v, char u);

char getVertex(int index);

int checkEdge(char v, char u);

int findVertex(char name);

void printMatrix(void);

void printAdjacent(char name);

**// GraphExample.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include "GraphMatrix.h"

// 배열 크기 : n(n+1)/2

enum { ARRAY\_SIZE = MAX\_VERTEX \* (MAX\_VERTEX+1) / 2 };

char g\_vertex[MAX\_VERTEX];

int g\_count;

int g\_matrix[ARRAY\_SIZE]; // 1차원 배열로 수정

// 2차원 배열에 접근할 때의 i와 j를 받아서 1차원 옵셋 계산

int getIndex(int vi, int ui) // 0부터 시작하기 때문에 +1을 사용하는 계산법을 재사용

{

return (vi < ui) ? (ui\*(ui+1))/2 + vi : (vi\*(vi+1))/2 + ui;

}

void init(void)

{

int i;

for(i = 0; i < MAX\_VERTEX; i++)

g\_vertex[i] = '\*';

for(i = 0; i < ARRAY\_SIZE; i++) // 1차원 배열이기 때문에 수정

g\_matrix[i] = 0;

g\_count = 0;

}

void insertVertex(char name)

{

g\_vertex[g\_count] = name;

g\_count++;

}

int insertEdge(char v, char u)

{

int vi = findVertex(v),

ui = findVertex(u);

if(vi == -1 || ui == -1)

return 0;

g\_matrix[getIndex(vi,ui)] = 1; // vi와 ui를 getIndex 함수의 매개 변수로 전달

return 1;

}

int checkEdge(char v, char u)

{

int vi = findVertex(v),

ui = findVertex(u);

return g\_matrix[getIndex(vi,ui)];

}

char getVertex(int index)

{

return g\_vertex[index];

}

int findVertex(char name)

{

int i;

for(i = 0; i < g\_count; i++)

{

if(g\_vertex[i] == name)

return i;

}

return -1;

}

void printMatrix(void)

{

int i, j;

printf("[정점+행렬]\n");

// 위쪽뚜껑(두줄)

printf(" | ");

for(i = 0; i < g\_count; i++)

printf("%c ", g\_vertex[i]);

printf("\n");

printf("--+");

for(i = 0; i < g\_count; i++)

printf("--");

printf("\n");

// 실제데이터

for(i = 0; i < g\_count; i++)

{

// 왼쪽뚜껑

printf("%c | ", g\_vertex[i]);

for(j = 0; j < g\_count; j++)

printf("%d ", g\_matrix[getIndex(i, j)]);

printf("\n");

}

printf("\n");

}

void printAdjacent(char name)

{

int i, find = findVertex(name);

if(find != -1)

{

printf("정점%c : ", name);

for(i = 0; i < g\_count; i++)

{

if(g\_matrix[getIndex(find,i)] == 1)

printf("%c ", g\_vertex[i]);

}

printf("\n");

}

}

**// GraphExampleMain.c**

#pragma warning(disable:4996)

#include <stdio.h>

#include <stdlib.h>

#include "GraphMatrix.h"

#define VERTEX\_COUNT MAX\_VERTEX // 정점의 개수를 똑같이 만들었음

#define EDGE\_COUNT 13

// main 함수는 본문과 동일합니다. 출력 결과 또한 본문과 똑같은 형식으로 맞췄습니다.

// getIndex 함수는 내부적으로만 사용하기 때문에, main 함수에서 달라질 것은 없습니다.

void main(void)

{

const char\* vertex = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int i;

char v, u, name;

init();

printf("[정점 추가]\n");

for(i = 0; i < VERTEX\_COUNT; )

{

name = vertex[rand()%26];

if(findVertex(name) == -1)

{

printf("%c ", name);

insertVertex(name);

i++;

}

}

printf("\n\n");

printf("[간선추가]\n");

for(i = 0; i < EDGE\_COUNT; )

{

v = getVertex(rand()%VERTEX\_COUNT);

u = getVertex(rand()%VERTEX\_COUNT);

if(v == u || checkEdge(v, u) == 1)

continue;

printf("(%c,%c) ", v, u);

if(i%10 == 9)

printf("\n");

insertEdge(v, u);

i++;

}

printf("\n\n");

printMatrix();

printAdjacent('P');

printAdjacent('A');

}

**// 출력 결과**

[정점 추가]

P H Q G U M E A Y

[간선 추가]

(Y,Q) (A,E) (A,M) (A,Y) (G,P) (P,E) (M,P) (U,A) (E,M) (Y,M)

(Q,P) (E,U) (Y,H)

[정점+행렬]

| P H Q G U M E A Y

--+------------------

P | 0 0 1 1 0 1 1 0 0

H | 0 0 0 0 0 0 0 0 1

Q | 1 0 0 0 0 0 0 0 1

G | 1 0 0 0 0 0 0 0 0

U | 0 0 0 0 0 0 1 1 0

M | 1 0 0 0 0 0 1 1 1

E | 1 0 0 0 1 1 0 1 0

A | 0 0 0 0 1 1 1 0 1

Y | 0 1 1 0 0 1 0 1 0

정점 P : Q G M E

정점 A : U M E Y