1. **소스코드 및 주석**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define MAX\_VERTEX 20

#define MAX\_EDGE 30

typedef struct {

int adjM[MAX\_VERTEX][MAX\_VERTEX];

int N;

} Graph;

void initGraph(Graph\* G, int N) {

G->N = N;

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

G->adjM[i][j] = 0;

}

void addEdge(Graph\* G, int u, int v) {

if (u >= 0 && u < G->N && v >= 0 && v < G->N) {

G->adjM[u][v] = 1;

G->adjM[v][u] = 1;

}

}

void generateRandomBipartiteGraph(Graph\* G) {

srand(time(NULL));

int leftCount = rand() % 6 + 5;

int rightCount = rand() % 6 + 5;

int vertexCount = leftCount + rightCount;

int edgeCount = rand() % 11 + 20;

initGraph(G, vertexCount);

int edgesAdded = 0;

while (edgesAdded < edgeCount) {

int u = rand() % leftCount;

int v = rand() % rightCount + leftCount;

if (G->adjM[u][v] == 0) {

addEdge(G, u, v);

edgesAdded++;

}

}

printf("Unweighted & Undirect graph를 생성합니다\n");

printf("(2-1와 2-2은 이분 그래프에서만 성립하므로, 이분 그래프로 생성합니다)");

printf("\n[Edge - 각 정점에 연결된 정점들]:\n");

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

printf("%d - (", i);

int first = 1;

for (int j = leftCount; j < G->N; j++) {

if (G->adjM[i][j]) {

if (!first) {

printf(", ");

}

printf("%d", j);

first = 0;

}

}

printf(")\n");

}

}

int vertexPrioritySelection(Graph\* G) {

int result = 0;

int visited[MAX\_VERTEX] = { 0 };

while (1) {

int maxEdges = -1;

int vertex = -1;

for (int i = 0; i < G->N; i++) {

if (visited[i]) continue;

int count = 0;

for (int j = 0; j < G->N; j++)

if (G->adjM[i][j] == 1) count++;

if (count > maxEdges) {

maxEdges = count;

vertex = i;

}

}

if (vertex == -1 || maxEdges == 0) break;

// 선택된 정점 처리

visited[vertex] = 1;

result++;

for (int i = 0; i < G->N; i++) {

G->adjM[vertex][i] = 0;

G->adjM[i][vertex] = 0;

}

}

return result;

}

int maximalMatching(Graph\* G) {

int result = 0;

int matched[MAX\_VERTEX] = { 0 };

for (int i = 0; i < G->N; i++) {

if (matched[i]) continue;

for (int j = i + 1; j < G->N; j++) {

if (G->adjM[i][j] == 1 && !matched[j]) {

matched[i] = matched[j] = 1;

result++;

break;

}

}

}

return result;

}

void copyGraph(Graph\* dest, Graph\* src) {

dest->N = src->N;

for (int i = 0; i < src->N; ++i)

for (int j = 0; j < src->N; ++j)

dest->adjM[i][j] = src->adjM[i][j];

}

int main() {

Graph G;

generateRandomBipartiteGraph(&G);

Graph G1, G2;

copyGraph(&G1, &G);

copyGraph(&G2, &G);

int vertexCount1 = vertexPrioritySelection(&G1);

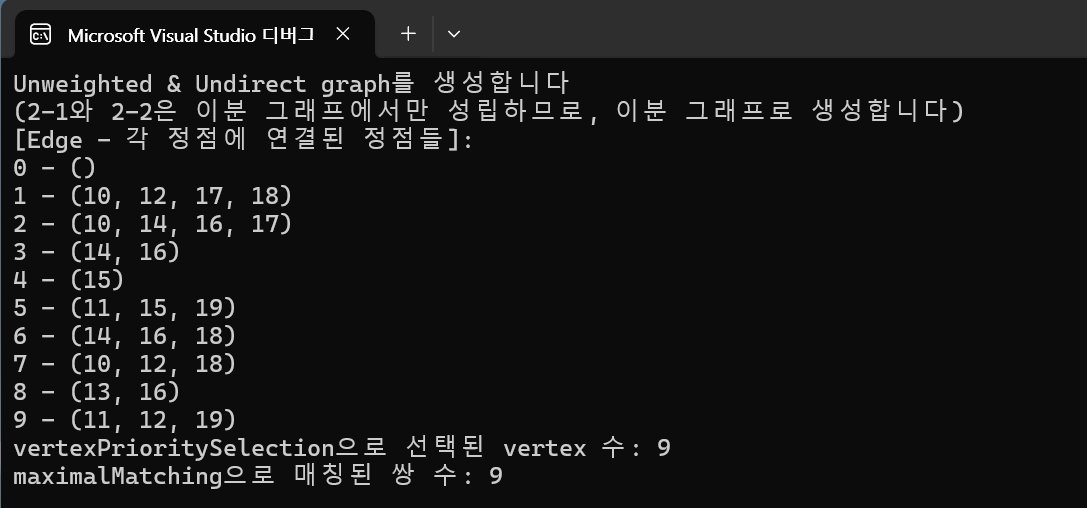
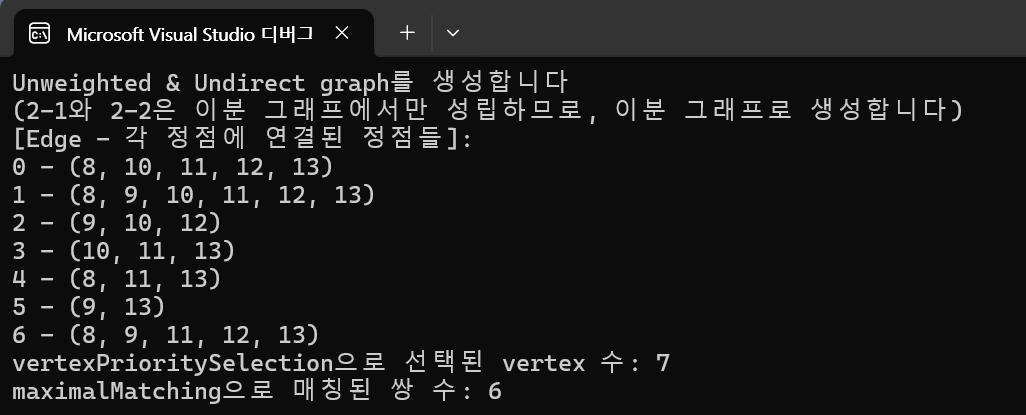
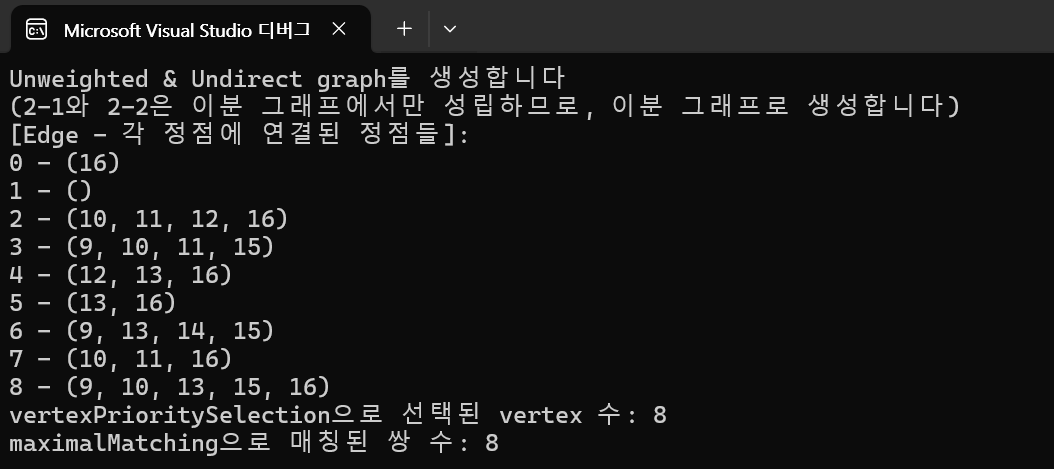
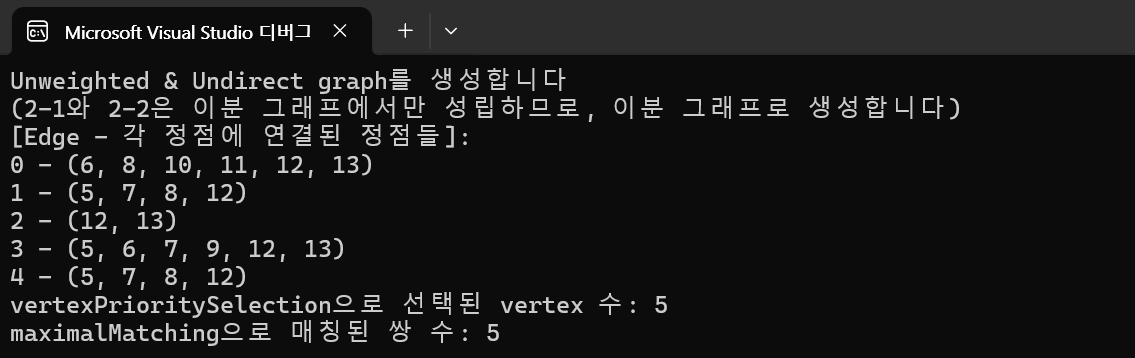
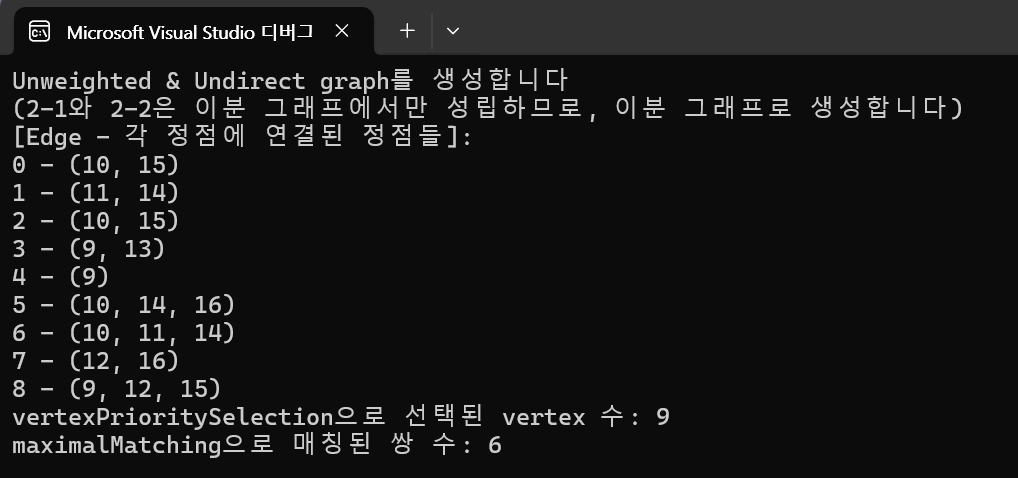
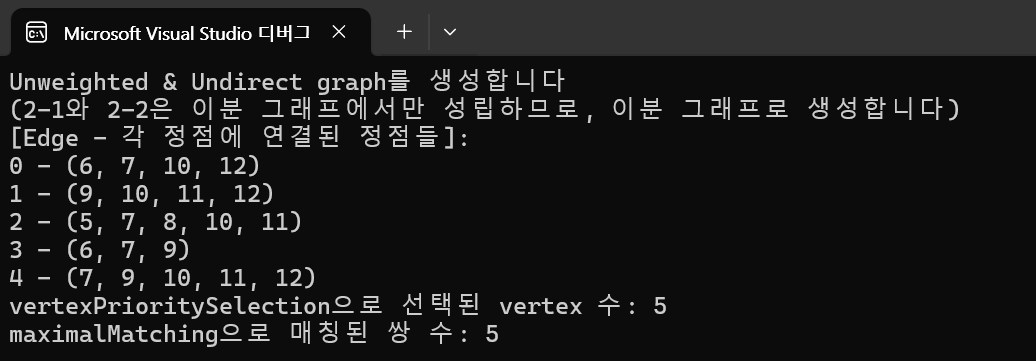
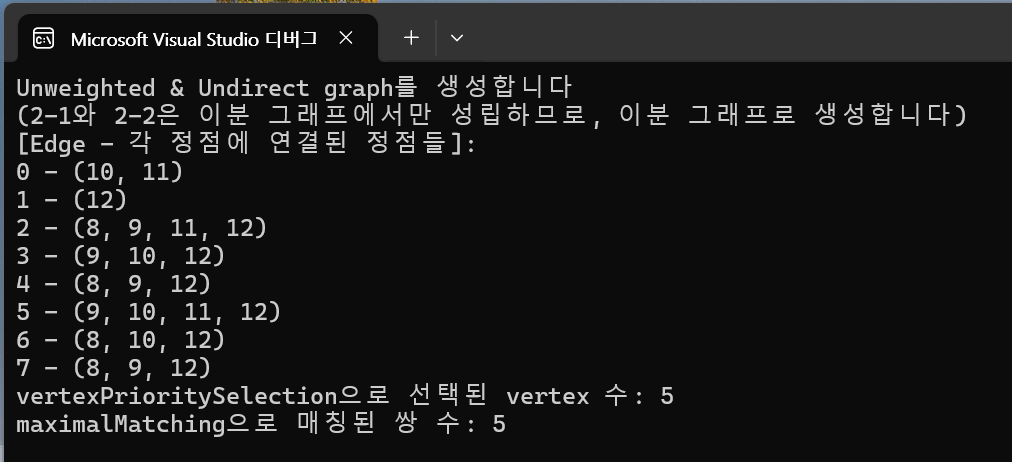
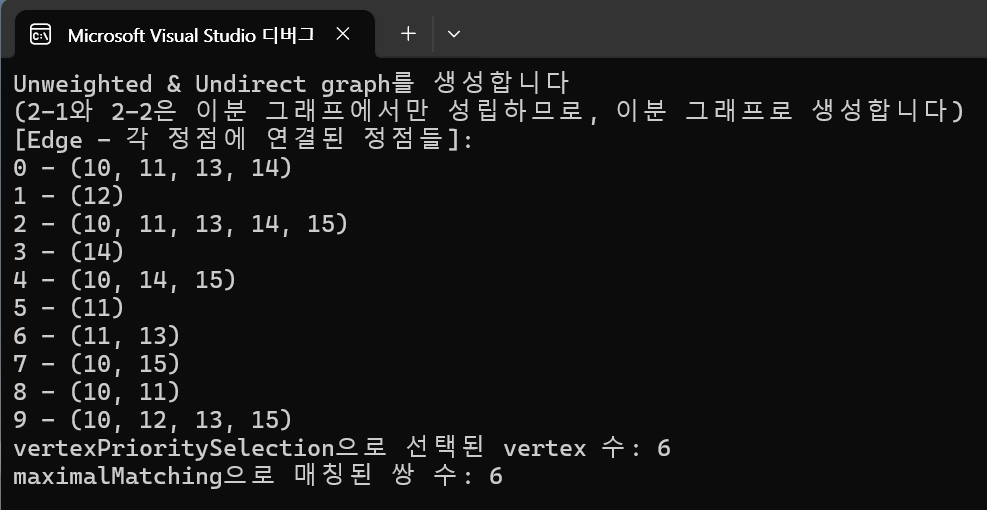
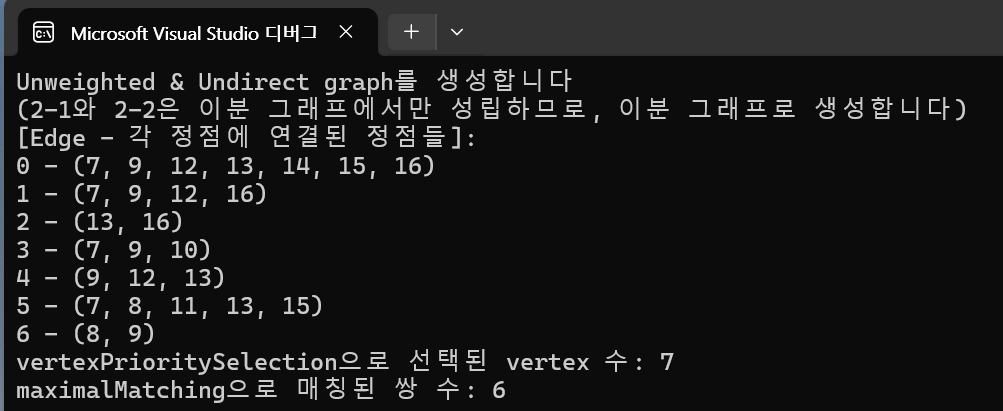
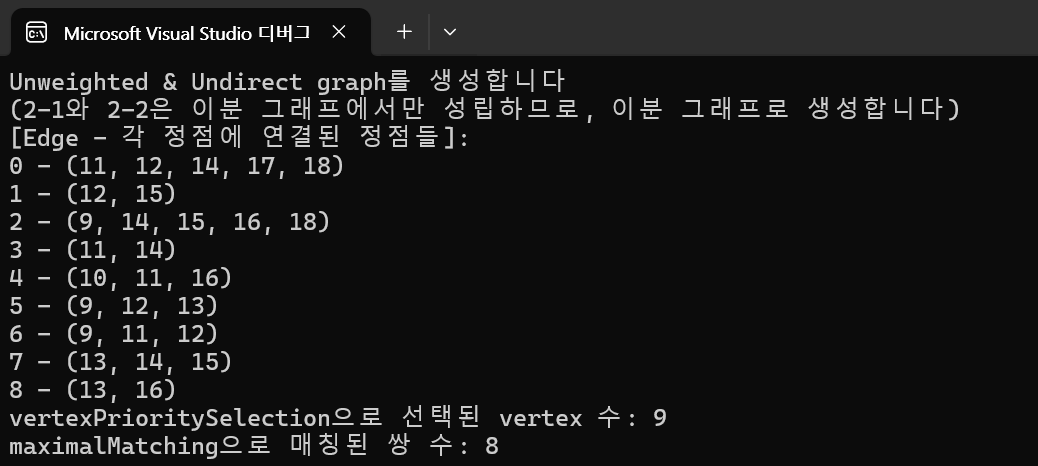
printf("vertexPrioritySelection으로 선택된 vertex 수: %d\n", vertexCount1);

int matchingCount = maximalMatching(&G2);

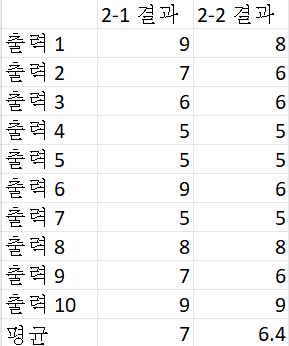
printf("maximalMatching으로 매칭된 쌍 수: %d\n", matchingCount);

return 0;

}

1. **출력 결과**

**최종 결과**



* 평균적으로 2-2 “Maximal Matching: disjoint edges(matching)” 방식이 더 우수하였습니다

1. **고찰**

* 평균적으로 2-1의 방식이 정답에 더 근사합니다
* 시간복잡도 역시 2-2가 더 적습니다
* 다만, 2-1의 방식이 직관적으로 이해하기 쉽다는 장점이 있습니다