**Graph**

#include <stdio.h>

#include <stdlib.h>

typedef struct arc{

struct arc\* next;

int weight;

struct vertex\* dest;

}Arc;

typedef struct vertex{

struct vertex\* next;//vertex들을 무작위 순서로 연결해놓은 것

int processed = 0;

Arc\* arc; //linked list형태로 연결된 vertex들을 표현함

void\* dataPtr;//vertex 이름

}Vertex;

typedef struct graph{

int count;

Vertex\* first;

int (\*compare)(void\* arg1, void\* arg2);

}Graph;

**//graph struct입니다. graph라는 큰 틀 안에 vertex와 arc 각각의 struct를 만들어 넣었고, 인접 리스트로 표현하였습니다.**

typedef struct node{

Vertex\* info;

struct node\* next;

}Node;

typedef struct head{

Node\* front, \*rear;

int count;

}Head;

**//위 두가지는 bfs를 위해서 만들었습니다. queue struct를 짜기 위해 필요한 head와 node입니다.**

Graph\* graphCreate(int (\*compare)(void\* a, void\* b)){

Graph\* a = (Graph\*)malloc(sizeof(Graph));

if (a == NULL)

return NULL;

a->compare = compare;

a->count = 0;

a->first = NULL;

return a;

}

**//graphCreate함수입니다. 저는 여기서 char를 vertex의 정보값으로 사용하였기에 compare 함수에다가 charcompare를 넣었습니다. Count는 0, first는 vertex를 향하는 것인데 현재 없으므로 NULL값으로 하였습니다.**

Head\* createqueue(){

Head\* a = (Head\*)malloc(sizeof(Head));

a->count = 0;

a->front = a->rear = NULL;

return a;

}

**//queue 생성 함수입니다.**

void enqueue(Head\* head, Vertex\* a){

Node\* newnode = (Node\*)malloc(sizeof(Node));

newnode->info = a;

newnode->next = NULL;

if (head->count == 0){

head->count++;

head->front = head->rear = newnode;

return;

}

head->count++;

head->rear->next = newnode;

head->rear = newnode;

return;

}

**// enqueue함수입니다.**

Vertex\* graphInsertVertex(Graph\* a, void\* info){

Vertex\* newvtx = (Vertex\*)malloc(sizeof(Vertex));

newvtx->arc = NULL;

newvtx->next = NULL;

newvtx->processed = 0;

newvtx->dataPtr = info;

a->count++;**// 새로운 vertex를 할당하고 a의 개수를 늘립니다.**

Vertex\* node = a->first;

if (node == NULL){

a->first = newvtx;

return newvtx;

}**//만약 vertex가 아무것도 없었을 경우**

while (node->next != NULL)

node = node->next;

node->next = newvtx;

return newvtx; **//vertex가 있었을 경우 vertex list의 끝부분에 삽입합니다.**

}

**//graphInsertVertex함수입니다. Void 포인터를 받아서, 그것을 vertex의 정보값으로 활용하여 그래프 a에 삽입합니다.**

void \_insertArc(Graph\* a, Vertex\* start, Vertex\* end, int weight){

if (a == NULL)

return;

Vertex\* find = a->first;

while (find != NULL){

if (start == find){ **//만약 vertex list에서 start vertex를 찾았을 때**

Arc\* a = find->arc;

Arc \*newnode = (Arc\*)malloc(sizeof(Arc));

newnode->dest = end;

newnode->weight = weight;

newnode->next = NULL;

**//end vertex를 dest로 하는 새로운 arc를 만들고**

if (a == NULL){

find->arc = newnode;

return;

}**//삽입 위치가 arc list의 가장 처음일 경우**

while (a->next != NULL)

a = a->next;

a->next = newnode;

return;

}**//삽입 위치가 arc list의 중간일경우**

find = find->next;

}

}

**//Arc삽입함수입니다. Directed graph일 경우 사용합니다.**

void \_removeArc(Graph\* a, Vertex\* start, Vertex\* end){

Vertex\* findvtx = a->first;

while (findvtx != NULL){

if (findvtx == start){//만약 시작 vertex를 찾았다면

Arc\* findarc = findvtx->arc;

Arc\* prefindarc = NULL;

for (; findarc != NULL; prefindarc = findarc, findarc = findarc->next){

if (findarc->dest == end){//만약 끝 vertex arc를 찾았다면

if (prefindarc == NULL){//arc list 첫번째일경우

findvtx->arc = findarc->next;

free(findarc);

return;

}

//첫번째 아닐경우

prefindarc->next = findarc->next;

free(findarc);

return;

}

}

}

findvtx = findvtx->next;

}

printf("There is no arc inside this Graph");

}

**//arc 제거함수입니다. Directed graph일 경우 사용합니다.**

void graphInsertArc(Graph\* a, Vertex\* start, Vertex\* end, int weight){

\_insertArc(a, start, end,weight);

\_insertArc(a, end, start,weight);

}

**//undirected graph일 때 arc를 insert하는 함수입니다. 위에서 구현한 directed graph용 함수를 2번 사용합니다.**

void graphRemoveArc(Graph\* a, Vertex\* start, Vertex\* end){

\_removeArc(a, start, end);

\_removeArc(a, end, start);

}

**//undirected graph일 때 arc를 remove하는 함수입니다. 위에서 구현한 directed graph용 함수를 2번 사용합니다.**

int charCompare(void\* a, void\* b){

if (\*(char\*)a > \*(char\*)b)

return 1;

else if (\*(char\*)a == \*(char\*)b)

return 0;

else

return -1;

}

**//charcompare 함수입니다. Graph struct에 집어넣습니다.**

void deleteVertex(Graph\* a, Vertex\* todelete){

if (a == NULL)

return;

Vertex\* prevtx = NULL;

Vertex\* nowvtx = a->first;

while(nowvtx != NULL){

if (todelete == nowvtx){

a->count--;

if (prevtx == NULL)//맨 앞이라면?

a->first = nowvtx->next;

else

prevtx->next = nowvtx->next;

Arc\* prearc;

Arc\* nowarc;

for (prearc = NULL, nowarc = nowvtx->arc; nowarc != NULL; prearc = nowarc, nowarc = nowarc->next){

if (nowarc == NULL)

break;//arc가 아무것도 없었다면?

else if (prearc == NULL){//맨 앞이라면?

free(nowarc);

break;

}

else

free(prearc);

}

if (prearc !=NULL)

free(prearc); //arc가 2개 이상이었을 때

free(nowvtx);

if (prevtx == NULL)//맨 앞 vtx

nowvtx = a->first;

else//아닐때

nowvtx = prevtx->next;

continue;

}//같은 vertex를 발견했다면

Arc\* prearc = NULL;

Arc\* nowarc = nowvtx->arc;

while (nowarc != NULL){

if (nowarc->dest == todelete){//지우고자 하는arc일때

if (prearc == NULL){//첫번째거이면

nowvtx->arc = nowarc->next;

free(nowarc);

nowarc = nowvtx->arc;

continue;

}

//첫번째가 아니면

prearc->next = nowarc->next;

free(nowarc);

nowarc = prearc->next;

continue;

}

prearc = nowarc;

nowarc = nowarc->next;

}

//같은 vertex가 아닐때는 arc를 지운다.

prevtx = nowvtx;

nowvtx = nowvtx->next;

}

}

**//이건 vertex에 arc가 있을 때도 vertex를 제거하는 함수입니다. 과제를 잘못 이해해서 만들었습니다.**

void graphDestroy(Graph\* a){

Vertex\* pDel = NULL;

Vertex\* pNext = NULL; **// vertex를 제거하기 위해 설정한 변수들입니다.**

for (pDel = a->first; pDel != NULL; pDel = pNext){

pNext = pDel->next;

Arc\* arcDel = NULL;

Arc\* arcNext = NULL; **//vertex 내에서 arc들을 free하기위해 설정했습니다.**

for (arcDel = pDel->arc; arcDel != NULL; arcDel = arcNext){

arcNext = arcDel->next;

free(arcDel);

}

free(pDel);

}

free(a);

}

**//graph를 destroy하는 함수입니다.**

void graphRemoveVertex(Graph\* a, Vertex\* todelete){

Vertex\* pPrev = NULL;

Vertex\* pDel = NULL;

**//지우고자 하는 Vertex와 그 전 vertex를 봅니다. 연결리스트이기 때문에 연결해주기 위해서입니다.**

for (pDel = a->first; pDel != NULL; pPrev = pDel, pDel = pDel->next){

if (todelete == pDel){ **//만약 지우고자 하는 Vertex가 맞을 경우**

a->count--;

if (pDel->arc != NULL){ **//arc가 있을 때는 지우지 않습니다.**

printf("It has an arc, so can't erase");

return;

}

if (pPrev == NULL){ **//list의 맨 앞에 있는 vertex일 경우**

a->first = pDel->next;

free(pDel);

}

else{**//list의 중간에 있는 vertex일 경우**

pPrev->next = pDel->next;

free(pDel);

}

return;

}

}

printf("No vertex inside here");

**//vertex가 존재하지 않을 경우 에러 메시지를 띄웁니다.**

}//Vertex가 arc를 가지고 있지 않을 경우만 지운다.

**//Vertex가 arc를 가지고 있지 않을 때만 지우는 graph Vertex remove함수입니다. 제거하기 위한 Vertex를 받아서 지웁니다.**

Vertex\* dequeue(Head\* a){

if (a->count == 0)

return NULL;

Node\* freenode;

if (a->count == 1){

freenode = a->front;

a->count--;

a->front = a->rear = NULL;

Vertex\* tmp = freenode->info;

free(freenode);

return tmp;

}

freenode = a->front;

a->front = a->front->next;

a->count--;

Vertex\* tmp = freenode->info;

free(freenode);

return tmp;

}

**//bfs를 위한 dequeue함수입니다. Vertex의 포인터를 받습니다.**

void dfs(Vertex\* a){

if (a->processed == 1){ // 방문한 적이 있으면

return;

}

a->processed = 1; **//현재 방문하고 있는 vertex processed를 바꾸고**

printf("%c", \*(char\*)a->dataPtr); **//출력합니다**

Arc\* nowarc = a->arc;

while (nowarc != NULL){ **//arc들을 전부 탐색해서**

if (nowarc->dest->processed == 0) **//만약 그중에 방문한 적이 없으면**

dfs (nowarc->dest); **//재귀로 돌립니다.**

nowarc = nowarc->next;

}

}

**//dfs 함수입니다.**

void bfs(Head\* a, Vertex\* start){

if (start->processed == 1) **//시작점을 방문했다면 그냥 종료합니다.**

return;

start->processed = 1;

enqueue(a, start); **//시작점을 enqueue합니다.**

while (a->count != 0){ **//queue에 아무것도 없어질때까지**

Vertex\* aa = dequeue(a); **//첫번째거를 dequeue하고 출력합니다.**

printf("%c", \*(char\*)aa->dataPtr);

Arc\* nowver = aa->arc;

while (nowver != NULL){ **//dequeue한 것의 모든 연결지점들을 processed를 바꾼 후 enqueue합니다.**

if (nowver->dest->processed != 1){

nowver->dest->processed = 1;

enqueue(a, nowver->dest);

}

nowver = nowver->next;

}

}

}

**//bfs 함수입니다.**

int isVertexIn(void\* a, Graph\* b){

Vertex\* nowver = NULL;

for (nowver = b->first; nowver != NULL; nowver = nowver->next){

if (b->compare(nowver->dataPtr, a) == 0)

return 1;

}

return 0;

}//그래프 b 안에 저 void\*를 가진 vertex가 존재하는가?

**//존재할 경우 1, 아니면 0을 출력합니다.**

Vertex\* vertexIn(void\* a, Graph\* b){

Vertex\* nowver = NULL;

for (nowver = b->first; nowver != NULL; nowver = nowver->next){

if (b->compare(nowver->dataPtr, a) == 0)

return nowver;

}

return NULL;

}//그래프 b 안에 저 void\*를 가진 vertex가 존재하는가?

**//존재할 경우 그 vertex pointer를, 아니면 NULL를 출력합니다.**

Graph\* PrimAlgorithm(Graph\* a){

Graph\* b = graphCreate((\*charCompare)); **//새로운 graph b를 만들고**

graphInsertVertex(b, a->first->dataPtr); **//주어진 그래프의 시작점을 b에 넣는다.**

Arc\* max = (Arc\*)malloc(sizeof(Arc));

max->weight = 99999999;

max->dest = NULL;

max->next = NULL;

**//가장 작은 arc를 판별하기 위해서, 임의의 max weight를 가진 arc를 만들었습니다.**

while (a->count != b->count){ **//b가 a에 있는 모든 vertex를 가질 경우 빠져나옵니다.**

Vertex\* nowvtx = NULL;

Vertex\* minvtx = NULL;

Arc\* nowarc = NULL;

Arc\* min = max;

for (nowvtx = a->first; nowvtx != NULL; nowvtx = nowvtx->next){

**//a의 모든 vertex 탐색**

if (isVertexIn(nowvtx->dataPtr, b) == 1){

//만약 이 vertex가 b 안에 있다면 이 vertex의 모든 arc를 살펴 가장 작은 것을 고릅니다.

for (nowarc = nowvtx->arc; nowarc != NULL; nowarc = nowarc->next){

if (nowarc->weight < min->weight && isVertexIn(nowarc->dest->dataPtr, b) !=1){

**//가장 작은 arc이나, 그 destination이 이미 b에 들어있는 경우 cycle이 형성되므로 그경우를 제외합니다.**

min = nowarc;

minvtx = nowvtx;

}

}

}

}

**//이렇게 min weight를 가진 arc와 양쪽의 vertex를 알고 있으니 b에 이 arc와 arc의 destination vertex를 각각 추가해줍니다.**

graphInsertArc(b, vertexIn(minvtx->dataPtr,b), graphInsertVertex(b, min->dest->dataPtr), min->weight);

}

return b;

}

**//Prim’s algorithm입니다. 우선 새로운 그래프를 하나 더 만들어서, 주어진 그래프 a를 가지고 minimum spanning tree를 b에 만들었습니다. b에 vertex를 넣고, 그 vertex에 연결된 모든 arc들을 a에서 탐색합니다. 그래서 제일 작은 것을 골라 b에 추가하는데, cycle이 형성될 경우는 제외합니다.(그 arc의 destination이 이미 b의 vertex 목록에 있을 경우)**

int main(){

Graph \* aa = graphCreate((\*charCompare));

Head\* queue = createqueue();

Vertex\* a = graphInsertVertex(aa, "A");

Vertex\* b = graphInsertVertex(aa, "B");

Vertex\* c = graphInsertVertex(aa, "C");

Vertex\* d = graphInsertVertex(aa, "D");

Vertex\* e = graphInsertVertex(aa, "E");

Vertex\* f = graphInsertVertex(aa, "F");

graphInsertArc(aa,a,b,6);

graphInsertArc(aa, a, c, 3);

graphInsertArc(aa, c, b, 2);

graphInsertArc(aa, d, b, 5);

graphInsertArc(aa, c,d, 3);

graphInsertArc(aa, c,e, 4);

graphInsertArc(aa, d,e, 2);

graphInsertArc(aa, d, f, 3);

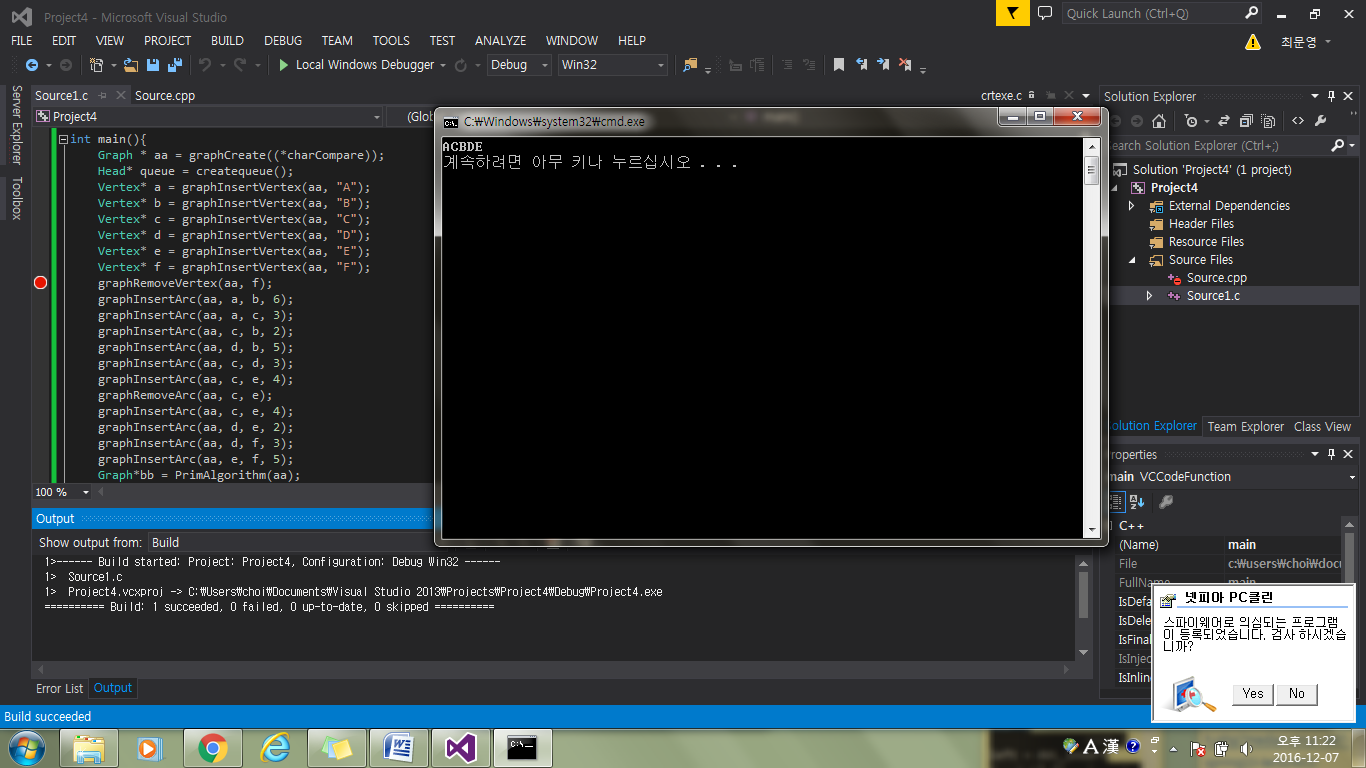
graphInsertArc(aa, e, f, 5);

Graph\*bb = PrimAlgorithm(aa);

bfs(queue, bb->first);

}

**//main에서 실행시켜 보았습니다. 모든 코드를 각자 돌려보았습니다. 아래는 전부 다 간단히 돌려본 콘솔창입니다.**

****