Breath-First Search Code

20150291 하현수

#include <iostream>

#include <vector>

#include <queue>

#include <stack>//각종 STL 자료구조를 인클루드합니다.

using namespace std;

class mynode

{

public :

int id;//알파벳순서대로 0~19의 id를 가지도록 하였습니다.

mynode\* parentnode;//부모노드입니다.

int gn;//시작지에서부터 해당노드에 가는데 까지 필요한 Cost입니다.

public :

mynode(int myid, mynode myparentnode)//mynode라는 클래스를 정의하였습니다.

{

id = myid;

parentnode = &myparentnode;

}

mynode()

{

}

};

void numtocity(int i)//숫자를 받아와 도시이름으로 변환해주는 함수입니다.

{

switch (i)

{

case 0:

printf("Arad ");

break;

case 1:

printf("Bucharest ");

break;

case 2:

printf("Craiova ");

break;

case 3:

printf("Dobreta ");

break;

case 4:

printf("Eforie ");

break;

case 5:

printf("Giurgiu ");

break;

case 6:

printf("Hirsova ");

break;

case 7:

printf("Iasi ");

break;

case 8:

printf("Lugoj ");

break;

case 9:

printf("Mehadia ");

break;

case 10:

printf("Neamt ");

break;

case 11:

printf("Oradea ");

break;

case 12:

printf("Pitesti ");

break;

case 13:

printf("RimnicuVilcea ");

break;

case 14:

printf("Sibiu ");

break;

case 15:

printf("Timisoara ");

break;

case 16:

printf("Urziceni ");

break;

case 17:

printf("Vaslui ");

break;

case 18:

printf("Yagaras ");

break;

case 19:

printf("Zerind ");

break;

}

}

void printmynode(queue<mynode> myopen,queue<mynode> myclose)//Open과 Close를 출력하는 함수입니다.

{

printf("Open : [ ");

while (!myopen.empty())

{

numtocity(myopen.front().id);

printf(" ");

myopen.pop();

}

printf("]\n");

printf("Close : [ ");

while (!myclose .empty())

{

numtocity(myclose.front().id);

myclose.pop();

}

printf("]\n");

}

int main()

{

int i;

int distance[20][20] =//도시간의 거리를 그래프형식으로 표현한 배열입니다. 0일경우 갈수 없다는 의미입니다.

{

//0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,140,118, 0, 0, 0, 75 },//0

{ 0, 0, 0, 0, 0,90, 0, 0, 0, 0, 0, 0, 101,0, 0, 0, 85, 0,211, 0 },//1

{ 0, 0, 0,120,0, 0, 0, 0, 0, 0, 0, 0,138,146,0, 0, 0, 0, 0, 0 },//2

{ 0, 0,120,0, 0, 0, 0, 0, 0,75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//3

{ 0, 0, 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0 },//4

{ 0,90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0 },//5

{ 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 98, 0, 0, 0 },//6

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 92, 0, 0 },//7

{ 0, 0, 0, 0, 0, 0, 0, 0, 0,70, 0, 0, 0, 0, 0,111,0, 0, 0, 0 },//8

{ 0, 0, 0,75, 0, 0, 0, 0,70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//9

{ 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//10

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,151,0, 0, 0, 0, 71 },//11

{ 0,101,138,0,0, 0, 0, 0, 0, 0, 0, 0, 0,97, 0, 0, 0, 0, 0, 0 },//12

{ 0, 0,146, 0,0, 0, 0, 0, 0, 0, 0, 0, 97, 0,80, 0, 0, 0, 0, 0 },//13

{ 140,0, 0, 0,0, 0, 0, 0, 0, 0, 0,151,0, 80, 0, 0, 0, 0,99, 0 },//14

{ 118,0, 0, 0,0, 0, 0, 0,111,0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 0 },//15

{ 0, 85, 0, 0,0, 0,98, 0, 0, 0, 0, 0, 0 , 0, 0, 0, 0,142, 0, 0 },//16

{ 0, 0, 0, 0, 0, 0, 0,92, 0, 0, 0, 0, 0, 0, 0, 0,142, 0, 0, 0 },//17

{ 0,211,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99,0, 0, 0, 0, 0 },//18

{ 75, 0, 0, 0,0, 0, 0, 0, 0, 0, 0,71, 0, 0 ,0, 0, 0, 0 ,0, 0 }//19

};

queue<mynode> open;

queue<mynode> close;

queue<mynode> tmpopen;

queue<mynode> tmpclose;//open과 close를 출력하면 기존의 open과 close가 손상되므로 임시 변수로 선언하였습니다.

mynode parent[1000];//부모노드를 저장하기 위한 공간입니다.

mynode tmpnode;

mynode nextnode;

mynode startnode;//임시 노드들입니다.

startnode.id = 15;//스타트노드인 Timisoara가 알파벳 15번쨰 순서이므로 15를 주고 시작합니다.

open.push(startnode);

bool check[20] = { false };//들렀던 곳인지 체크하기 위한 bool배열입니다.

bool find = false;//목표노드를 찾았는지 체크하기 위한 변수입니다.

int n = 0;

int plus = 0;

while (!open.empty())//distance 배열을 통해 BreadthFirst방식으로 큐에 open과 close를 삽입 삭제합니다.

{

printf("Iteration : %d\n", ++n);

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

tmpnode = open.front();

if (tmpnode.id == 1)//open노드에서 뽑은것이 Bucharest면 루프를 break합니다.

break;

check[tmpnode.id] = true;

open.pop();

close.push(tmpnode);

for (i = 0; i < 20; i++)

{

if (distance[tmpnode.id][i] != 0)//자식노드를 추가합니다.

{

if (check[i] == false)

{

nextnode.id = i;

parent[plus++] = tmpnode;

nextnode.parentnode = &parent[plus - 1];

nextnode.gn = parent[plus - 1].gn + distance[tmpnode.id][i];

open.push(nextnode);

check[i] = true;

if (i == 1)

find = true;//목표노드를 찾았는지 체크합니다.

}

}

}

}

printf("Iteration : Final\n");

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

stack<mynode> path;

while (open.front().id != 1)

{

open.pop();

}

tmpnode = open.front();

if (find = true)//목표노드를 찾았다면 경로를 출력합니다.

{

int cost = tmpnode.gn;

printf("Path : [ ");

while (tmpnode.id != 15)

{

path.push(tmpnode);

tmpnode = \*tmpnode.parentnode;

}

path.push(tmpnode);

while (!path.empty())

{

numtocity(path.top().id);

path.pop();

}

printf(" ]");

printf("\nCost : %d", cost);

}

else//경로 못찾았을시 메세지 출력

{

printf("경로찾기 실패!");

}

getchar();

}

Depth-First Search Source Code

#include <iostream>

#include <vector>

#include <deque>

#include <stack>

#include <deque>//자료구조들을 위한 include

using namespace std;

class mynode

{

public:

int id;//알파벳순서대로 0~19의 id를 가지도록 하였습니다.

mynode\* parentnode;//부모노드입니다.

int gn;//시작지에서부터 해당노드에 가는데 까지 필요한 Cost입니다.

public:

mynode(int myid, mynode myparentnode)//mynode라는 클래스를 정의하였습니다.

{

id = myid;

parentnode = &myparentnode;

}

mynode()

{

}

};

void numtocity(int i)//숫자를 받아와 도시이름으로 변환해주는 함수입니다.

{

switch (i)

{

case 0:

printf("Arad ");

break;

case 1:

printf("Bucharest ");

break;

case 2:

printf("Craiova ");

break;

case 3:

printf("Dobreta ");

break;

case 4:

printf("Eforie ");

break;

case 5:

printf("Giurgiu ");

break;

case 6:

printf("Hirsova ");

break;

case 7:

printf("Iasi ");

break;

case 8:

printf("Lugoj ");

break;

case 9:

printf("Mehadia ");

break;

case 10:

printf("Neamt ");

break;

case 11:

printf("Oradea ");

break;

case 12:

printf("Pitesti ");

break;

case 13:

printf("RimnicuVilcea ");

break;

case 14:

printf("Sibiu ");

break;

case 15:

printf("Timisoara ");

break;

case 16:

printf("Urziceni ");

break;

case 17:

printf("Vaslui ");

break;

case 18:

printf("Yagaras ");

break;

case 19:

printf("Zerind ");

break;

}

}

void printmynode(deque<mynode> myopen, deque<mynode> myclose)//Open과 Close를 출력하는 함수입니다.

{

printf("Open : [ ");

while (!myopen.empty())

{

numtocity(myopen.front().id);

printf(" ");

myopen.pop\_front();

}

printf("]\n");

printf("Close : [ ");

while (!myclose.empty())

{

numtocity(myclose.front().id);

myclose.pop\_front();

}

printf("]\n");

}

int main()

{

int i;

int distance[20][20] =//도시간의 거리를 그래프형식으로 표현한 배열입니다. 0일경우 갈수 없다는 의미입니다.

{

//0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,140,118, 0, 0, 0, 75 },//0

{ 0, 0, 0, 0, 0,90, 0, 0, 0, 0, 0, 0, 101,0, 0, 0, 85, 0,211, 0 },//1

{ 0, 0, 0,120,0, 0, 0, 0, 0, 0, 0, 0,138,146,0, 0, 0, 0, 0, 0 },//2

{ 0, 0,120,0, 0, 0, 0, 0, 0,75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//3

{ 0, 0, 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0 },//4

{ 0,90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0 },//5

{ 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 98, 0, 0, 0 },//6

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 92, 0, 0 },//7

{ 0, 0, 0, 0, 0, 0, 0, 0, 0,70, 0, 0, 0, 0, 0,111,0, 0, 0, 0 },//8

{ 0, 0, 0,75, 0, 0, 0, 0,70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//9

{ 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//10

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,151,0, 0, 0, 0, 71 },//11

{ 0,101,138,0,0, 0, 0, 0, 0, 0, 0, 0, 0,97, 0, 0, 0, 0, 0, 0 },//12

{ 0, 0,146, 0,0, 0, 0, 0, 0, 0, 0, 0, 97, 0,80, 0, 0, 0, 0, 0 },//13

{ 140,0, 0, 0,0, 0, 0, 0, 0, 0, 0,151,0, 80, 0, 0, 0, 0,99, 0 },//14

{ 118,0, 0, 0,0, 0, 0, 0,111,0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 0 },//15

{ 0, 85, 0, 0,0, 0,98, 0, 0, 0, 0, 0, 0 , 0, 0, 0, 0,142, 0, 0 },//16

{ 0, 0, 0, 0, 0, 0, 0,92, 0, 0, 0, 0, 0, 0, 0, 0,142, 0, 0, 0 },//17

{ 0,211,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99,0, 0, 0, 0, 0 },//18

{ 75, 0, 0, 0,0, 0, 0, 0, 0, 0, 0,71, 0, 0 ,0, 0, 0, 0 ,0, 0 }//19

};

int hn[20] = { 366,0,160,242,161,77,151,226,244,241,234,380,98,193,253,329,80,199,178,374 };

deque<mynode> open;

deque<mynode> close;

deque<mynode> tmpopen;

deque<mynode> tmpclose;//open과 close를 출력하면 기존의 open과 close가 손상되므로 임시 변수로 선언하였습니다.

mynode parent[1000];//부모노드를 저장하기 위한 공간입니다.

mynode tmpnode;

mynode nextnode;

mynode startnode;

startnode.id = 15;//스타트노드인 Timisoara가 알파벳 15번쨰 순서이므로 15를 주고 시작합니다.

open.push\_front(startnode);

bool check[20] = { false };//들렀던 곳인지 체크하기 위한 bool배열입니다.

bool find = false;//목표노드를 찾았는지 체크하기 위한 변수입니다.

int n = 0;

int plus = 0;

while (!open.empty())//distance 배열을 통해 depthFirst방식으로 deque에 open과 close를 삽입 삭제합니다.

{

printf("Iteration : %d\n", ++n);

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

tmpnode = open.front();

if (tmpnode.id == 1)//open노드에서 뽑은것이 Bucharest면 루프를 break합니다.

break;

check[tmpnode.id] = true;

open.pop\_front();

close.push\_front(tmpnode);

for (i = 19; i >= 0; i--)

{

if (distance[tmpnode.id][i] != 0)//자식노드를 추가합니다.

{

if (check[i] == false)

{

nextnode.id = i;

parent[plus++] = tmpnode;

nextnode.parentnode = &parent[plus - 1];

nextnode.gn = parent[plus - 1].gn + distance[tmpnode.id][i];

open.push\_front(nextnode);

check[i] = true;

if (i == 1)

find = true;//목표노드를 찾았는지 체크합니다.

}

}

}

}

printf("Iteration : Final\n");

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

stack<mynode> path;

while (open.front().id != 1)

{

open.pop\_front();

}

tmpnode = open.front();

if (find = true)//목표노드를 찾았다면 경로를 출력합니다.

{

int cost = tmpnode.gn;

printf("Path : [ ");

while (tmpnode.id != 15)

{

path.push(tmpnode);

tmpnode = \*tmpnode.parentnode;

}

path.push(tmpnode);

while (!path.empty())

{

numtocity(path.top().id);

path.pop();

}

printf(" ]");

printf("\nCost : %d", cost);

}

else//경로 못찾았을시 메세지 출력

{

printf("경로찾기 실패!");

}

getchar();

}

Hill Climbing Source Code

#include <iostream>

#include <vector>

#include <deque>

#include <stack>

#include <deque>//각종 STL 자료구조를 인클루드합니다.

#include <algorithm>//sort 함수가 담긴 헤더

using namespace std;

class mynode

{

public:

int id;//알파벳순서대로 0~19의 id를 가지도록 하였습니다.

mynode\* parentnode;//부모노드입니다.

int gn;//시작지에서부터 해당노드에 가는데 까지 필요한 Cost입니다.

int fn;//평가함수입니다.

int hn;//Bucharest까지의 직선거리입니다.

public:

mynode(int myid, mynode myparentnode)//mynode라는 클래스를 정의하였습니다.

{

id = myid;

parentnode = &myparentnode;

}

mynode()

{

}

bool operator <(const mynode &a) const//sort를 위한 연산자 오버로딩입니다.

{

return this->fn < a.fn;

}

};

void numtocity(int i)//숫자를 받아와 도시이름으로 변환해주는 함수입니다.

{

switch (i)

{

case 0:

printf("Arad ");

break;

case 1:

printf("Bucharest ");

break;

case 2:

printf("Craiova ");

break;

case 3:

printf("Dobreta ");

break;

case 4:

printf("Eforie ");

break;

case 5:

printf("Giurgiu ");

break;

case 6:

printf("Hirsova ");

break;

case 7:

printf("Iasi ");

break;

case 8:

printf("Lugoj ");

break;

case 9:

printf("Mehadia ");

break;

case 10:

printf("Neamt ");

break;

case 11:

printf("Oradea ");

break;

case 12:

printf("Pitesti ");

break;

case 13:

printf("RimnicuVilcea ");

break;

case 14:

printf("Sibiu ");

break;

case 15:

printf("Timisoara ");

break;

case 16:

printf("Urziceni ");

break;

case 17:

printf("Vaslui ");

break;

case 18:

printf("Yagaras ");

break;

case 19:

printf("Zerind ");

break;

}

}

void printmynode(deque<mynode> myopen, deque<mynode> myclose)//Open과 Close를 출력하는 함수입니다.

{

printf("Open : [ ");

while (!myopen.empty())

{

numtocity(myopen.front().id);

printf(" ");

myopen.pop\_front();

}

printf("]\n");

printf("Close : [ ");

while (!myclose.empty())

{

numtocity(myclose.front().id);

myclose.pop\_front();

}

printf("]\n");

}

int main()

{

int i;

int distance[20][20] =//도시간의 거리를 그래프형식으로 표현한 배열입니다. 0일경우 갈수 없다는 의미입니다.

{

//0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,140,118, 0, 0, 0, 75 },//0

{ 0, 0, 0, 0, 0,90, 0, 0, 0, 0, 0, 0, 101,0, 0, 0, 85, 0,211, 0 },//1

{ 0, 0, 0,120,0, 0, 0, 0, 0, 0, 0, 0,138,146,0, 0, 0, 0, 0, 0 },//2

{ 0, 0,120,0, 0, 0, 0, 0, 0,75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//3

{ 0, 0, 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0 },//4

{ 0,90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0 },//5

{ 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 98, 0, 0, 0 },//6

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 92, 0, 0 },//7

{ 0, 0, 0, 0, 0, 0, 0, 0, 0,70, 0, 0, 0, 0, 0,111,0, 0, 0, 0 },//8

{ 0, 0, 0,75, 0, 0, 0, 0,70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//9

{ 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//10

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,151,0, 0, 0, 0, 71 },//11

{ 0,101,138,0,0, 0, 0, 0, 0, 0, 0, 0, 0,97, 0, 0, 0, 0, 0, 0 },//12

{ 0, 0,146, 0,0, 0, 0, 0, 0, 0, 0, 0, 97, 0,80, 0, 0, 0, 0, 0 },//13

{ 140,0, 0, 0,0, 0, 0, 0, 0, 0, 0,151,0, 80, 0, 0, 0, 0,99, 0 },//14

{ 118,0, 0, 0,0, 0, 0, 0,111,0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 0 },//15

{ 0, 85, 0, 0,0, 0,98, 0, 0, 0, 0, 0, 0 , 0, 0, 0, 0,142, 0, 0 },//16

{ 0, 0, 0, 0, 0, 0, 0,92, 0, 0, 0, 0, 0, 0, 0, 0,142, 0, 0, 0 },//17

{ 0,211,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99,0, 0, 0, 0, 0 },//18

{ 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 71, 0, 0 ,0, 0, 0, 0 ,0, 0 }//19

};

int hnlist[20] = { 366,0,160,242,161,77,151,226,244,241,234,380,98,193,253,329,80,199,178,374 };

deque<mynode> open;

deque<mynode> close;

deque<mynode> tmpopen;

deque<mynode> tmpclose;//open과 close를 출력하면 기존의 open과 close가 손상되므로 임시 변수로 선언하였습니다.

mynode parent[1000];

mynode tmpnode;

mynode nextnode;

mynode startnode;//임시 노드들입니다.

startnode.id = 15;//스타트노드인 Timisoara가 알파벳 15번쨰 순서이므로 15를 주고 시작합니다.

open.push\_front(startnode);

bool check[20] = { false };//들렀던 곳인지 체크하기 위한 bool배열입니다.

bool find = false;//목표노드를 찾았는지 체크하기 위한 변수입니다.

int n = 0;

int plus = 0;

int nodenum =0;

deque<mynode>::iterator iter;//sort를 위한 반복자입니다.

while (!open.empty())//distance 배열을 통해 Hillclimbing방식으로 큐에 open과 close를 삽입 삭제합니다.

{

printf("\n");

printf("iteration : %d\n", ++n);

mynode tmpmy;

iter = open.begin();

tmpmy = \*iter;

for (i = 1; i < nodenum; i++)

{

//tmpmy = \*iter;

iter++;

//printf("id : %d\n", tmpmy.id);

}

sort(open.begin(), iter+1);//새로들어오게될 자식노드만 정렬합니다.(기존에 있던 노드는 원자리 유지)

nodenum = 0;

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

tmpnode = open.front();

if (tmpnode.id == 1)

break;

check[tmpnode.id] = true;

open.pop\_front();

close.push\_front(tmpnode);

for (i = 19; i >= 0; i--)

{

if (distance[tmpnode.id][i] != 0)//자식노드를 추가합니다.

{

if (check[i] == false)

{

nextnode.id = i;

parent[plus++] = tmpnode;

nextnode.parentnode = &parent[plus - 1];

nextnode.hn = hnlist[i];

nextnode.gn = parent[plus - 1].gn + distance[tmpnode.id][i];//부모의 gn에 distance를 더해 총 거리를 구합니다.

nextnode.fn = nextnode.hn + nextnode.gn;//평가함수fn에 값을 넣습니다.

open.push\_front(nextnode);

check[i] = true;//지나온곳이라고 표시합니다.

if (i == 1)

find = true;//목표노트를 찾았는지 체크합니다.

nodenum++;//총 몇개의 자식노드가 들어오는지를 세고, 이를 통해 새로 들어올 노드만 소팅합니다.

}

}

}

}

printf("\n");

printf("Iteration : Final\n");

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

stack<mynode> path;

while (open.front().id != 1)

{

open.pop\_front();

}

tmpnode = open.front();

if (find = true)

{

int cost = tmpnode.gn;

printf("\n");

printf("Path : [ ");

while (tmpnode.id != 15)

{

path.push(tmpnode);

tmpnode = \*tmpnode.parentnode;

}

path.push(tmpnode);

while (!path.empty())

{

numtocity(path.top().id);

path.pop();

}

printf(" ]");

printf("\nCost : %d", cost);

}

else

{

printf("경로찾기 실패!");

}

getchar();

}

Best-First Search Source Code :

#include <iostream>

#include <vector>

#include <deque>

#include <stack>

#include <deque>//각종 STL 자료구조를 인클루드합니다.

#include <algorithm>//sort 함수가 담긴 헤더

using namespace std;

class mynode

{

public:

int id;//알파벳순서대로 0~19의 id를 가지도록 하였습니다.

mynode\* parentnode;//부모노드입니다.

int gn;//시작지에서부터 해당노드에 가는데 까지 필요한 Cost입니다.

int fn;//평가함수입니다.

int hn;//Bucharest까지의 직선거리입니다.

public:

mynode(int myid, mynode myparentnode)//mynode라는 클래스를 정의하였습니다.

{

id = myid;

parentnode = &myparentnode;

}

mynode()

{

}

bool operator <(const mynode &a) const//sort를 위한 연산자 오버로딩입니다.

{

return this->fn < a.fn;

}

};

void numtocity(int i)//숫자를 받아와 도시이름으로 변환해주는 함수입니다.

{

switch (i)

{

case 0:

printf("Arad ");

break;

case 1:

printf("Bucharest ");

break;

case 2:

printf("Craiova ");

break;

case 3:

printf("Dobreta ");

break;

case 4:

printf("Eforie ");

break;

case 5:

printf("Giurgiu ");

break;

case 6:

printf("Hirsova ");

break;

case 7:

printf("Iasi ");

break;

case 8:

printf("Lugoj ");

break;

case 9:

printf("Mehadia ");

break;

case 10:

printf("Neamt ");

break;

case 11:

printf("Oradea ");

break;

case 12:

printf("Pitesti ");

break;

case 13:

printf("RimnicuVilcea ");

break;

case 14:

printf("Sibiu ");

break;

case 15:

printf("Timisoara ");

break;

case 16:

printf("Urziceni ");

break;

case 17:

printf("Vaslui ");

break;

case 18:

printf("Yagaras ");

break;

case 19:

printf("Zerind ");

break;

}

}

void printmynode(deque<mynode> myopen, deque<mynode> myclose)//Open과 Close를 출력하는 함수입니다.

{

printf("Open : [ ");

while (!myopen.empty())

{

numtocity(myopen.front().id);

printf(" ");

myopen.pop\_front();

}

printf("]\n");

printf("Close : [ ");

while (!myclose.empty())

{

numtocity(myclose.front().id);

myclose.pop\_front();

}

printf("]\n");

}

int main()

{

int i;

int distance[20][20] =//도시간의 거리를 그래프형식으로 표현한 배열입니다. 0일경우 갈수 없다는 의미입니다.

{

//0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,140,118, 0, 0, 0, 75 },//0

{ 0, 0, 0, 0, 0,90, 0, 0, 0, 0, 0, 0, 101,0, 0, 0, 85, 0,211, 0 },//1

{ 0, 0, 0,120,0, 0, 0, 0, 0, 0, 0, 0,138,146,0, 0, 0, 0, 0, 0 },//2

{ 0, 0,120,0, 0, 0, 0, 0, 0,75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//3

{ 0, 0, 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0 },//4

{ 0,90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,0, 0 },//5

{ 0, 0, 0, 0,86, 0, 0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 98, 0, 0, 0 },//6

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 92, 0, 0 },//7

{ 0, 0, 0, 0, 0, 0, 0, 0, 0,70, 0, 0, 0, 0, 0,111,0, 0, 0, 0 },//8

{ 0, 0, 0,75, 0, 0, 0, 0,70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//9

{ 0, 0, 0, 0, 0, 0, 0,87, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },//10

{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,151,0, 0, 0, 0, 71 },//11

{ 0,101,138,0,0, 0, 0, 0, 0, 0, 0, 0, 0,97, 0, 0, 0, 0, 0, 0 },//12

{ 0, 0,146, 0,0, 0, 0, 0, 0, 0, 0, 0, 97, 0,80, 0, 0, 0, 0, 0 },//13

{ 140,0, 0, 0,0, 0, 0, 0, 0, 0, 0,151,0, 80, 0, 0, 0, 0,99, 0 },//14

{ 118,0, 0, 0,0, 0, 0, 0,111,0, 0, 0, 0, 0 ,0, 0, 0, 0, 0, 0 },//15

{ 0, 85, 0, 0,0, 0,98, 0, 0, 0, 0, 0, 0 , 0, 0, 0, 0,142, 0, 0 },//16

{ 0, 0, 0, 0, 0, 0, 0,92, 0, 0, 0, 0, 0, 0, 0, 0,142, 0, 0, 0 },//17

{ 0,211,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99,0, 0, 0, 0, 0 },//18

{ 75, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 71, 0, 0 ,0, 0, 0, 0 ,0, 0 }//19

};

int hnlist[20] = { 366,0,160,242,161,77,151,226,244,241,234,380,98,193,253,329,80,199,178,374 };

deque<mynode> open;

deque<mynode> close;

deque<mynode> tmpopen;

deque<mynode> tmpclose;//open과 close를 출력하면 기존의 open과 close가 손상되므로 임시 변수로 선언하였습니다.

mynode parent[1000];

mynode tmpnode;

mynode nextnode;

mynode startnode;//임시 노드들입니다.

startnode.id = 15;//스타트노드인 Timisoara가 알파벳 15번쨰 순서이므로 15를 주고 시작합니다.

open.push\_front(startnode);

bool check[20] = { false };//들렀던 곳인지 체크하기 위한 bool배열입니다.

bool find = false;//목표노드를 찾았는지 체크하기 위한 변수입니다.

int n = 0;

int plus = 0;

int nodenum = 0;

deque<mynode>::iterator iter;//sort를 위한 반복자입니다.

while (!open.empty())//distance 배열을 통해 Best First방식으로 큐에 open과 close를 삽입 삭제합니다.

{

printf("\n");

printf("iteration : %d\n", ++n);

sort(open.begin(), open.end());//모든 노드를 매번 재정렬합니다.

nodenum = 0;

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

tmpnode = open.front();

if (tmpnode.id == 1)

{

break;

}

check[tmpnode.id] = true;

open.pop\_front();

close.push\_front(tmpnode);

for (i = 19; i >= 0; i--)

{

if (distance[tmpnode.id][i] != 0)//자식노드를 추가합니다.

{

if (check[i] == false)

{

nextnode.id = i;

parent[plus++] = tmpnode;

nextnode.parentnode = &parent[plus - 1];

nextnode.hn = hnlist[i];

nextnode.gn = parent[plus-1].gn + distance[tmpnode.id][i];

nextnode.fn = nextnode.hn + nextnode.gn;

open.push\_front(nextnode);

check[i] = true;//지나온곳이라고 표시합니다.

if (i == 1)

find = true;//목표노트를 찾았는지 체크합니다.

}

}

}

}

printf("\n");

printf("Iteration : Final\n");

tmpopen = open;

tmpclose = close;

printmynode(tmpopen, tmpclose);

stack<mynode> path;

while (open.front().id != 1)

{

open.pop\_front();

}

tmpnode = open.front();

if (find = true)

{

int cost = tmpnode.gn;

printf("\n");

printf("Path : [ ");

while (tmpnode.id != 15)

{

path.push(tmpnode);

tmpnode = \*tmpnode.parentnode;

}

path.push(tmpnode);

while (!path.empty())

{

numtocity(path.top().id);

path.pop();

}

printf(" ]");

printf("\nCost : %d", cost);

}

else

{

printf("경로찾기 실패!");

}

getchar();

}