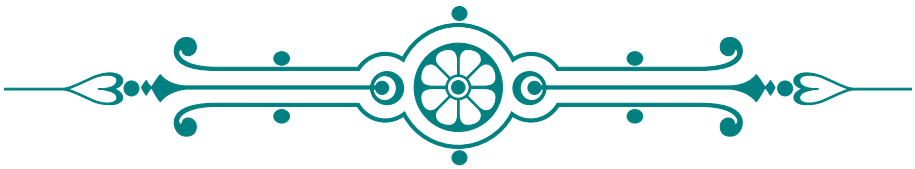


컴퓨터 알고리즘과 실습

학번 : 2016110056

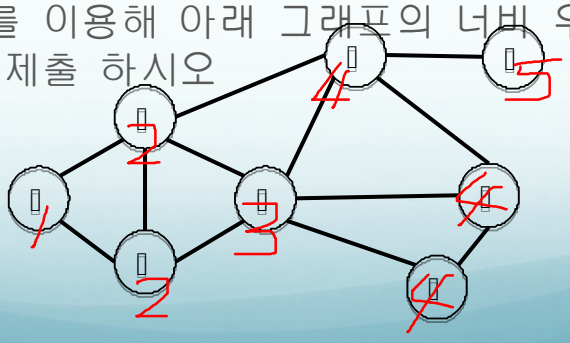
학과 : 불교학부

이름 : 박승원

날짜 : 2017년 5월 11일

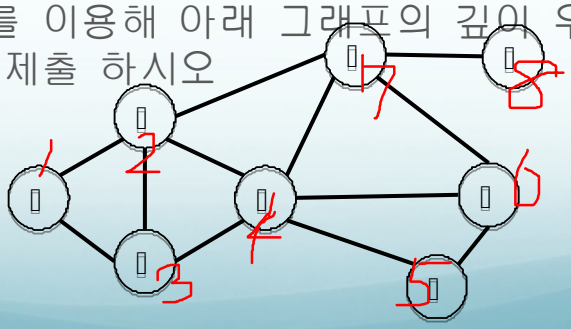
문제 □ □□너비우선 탐색

- 다음 그래프에 대해 너비 우선 탐색을 해보시오
  - 너비우선 탐색을 해보고 결과로 나오는 너비우선□탐색□□ 트리를 그리시오□□탐색은 노드 □로 부터 시작되고 탐색 결과로 나온 트리 각 노드 밑에 □로 부터의 탐색 거리를 적으시오□□
  - 너비 우선 탐색을 하고 탐색되는 순서대로 노드를 출력하는 코드를 구현하시오□
  - 작성한 코드를 이용해 아래 그래프의 너비 우선 탐색결과를 캡쳐해 제출 하시오



문제 □ □□깊이우선 탐색

- 다음 그래프에 대해 깊이 우선 탐색을 해보시오
  - 깊이우선 탐색을 해보고 결과로 나오는 깊이우선□탐색□□ 트리를 그리시오□□탐색은 노드 □로 부터 시작되고 탐색 결과로 나온 트리 각 노드 밑에 □로 부터의 탐색 거리를 적으시오□□
  - 깊이 우선 탐색을 하고 탐색되는 순서대로 노드를 출력하는 코드를 구현하시오□
  - 작성한 코드를 이용해 아래 그래프의 깊이 우선 탐색결과를 캡쳐해 제출 하시오



```
// 2016110056  박승원

#include"tgraph.h"
using namespace std;

int main()
{
    Graph<char> gr;
    const char vert[] = "svwxmkqe";
    for(int i=0; i<8; i++) gr.insert_vertex(vert[i]);
    const char edge[][3] = {"sv", "sw", "vw", "vx", "wx", "xk", "xm", "mk", "vq", "xq", "qe", "qm"
        };
    for(int i=0; i<sizeof(edge)/3; i++) {
        gr.insert_edge(edge[i][0], edge[i][1], 0);
        gr.insert_edge(edge[i][1], edge[i][0], 0);
    }
    gr.view();
    cout << endl << "너비 우선 탐색 결과 : ";
    gr.breadth();
    cout << endl << "깊이 우선 탐색 결과 : ";
    gr.depth();
    cout << endl;

}
```

```
//2016110056 박승원
#include"tgraph.h"
using namespace std;

int main()
{
    Graph<char> gr;
    const char vert[] = "svwxmkqe";
    for(int i=0; i<8; i++) gr.insert_vertex(vert[i]);
---------- 문제 1번 실행을 시작합니다. --------------------------
./1.x
s : <s,v>0 <s,w>0
v : <v,s>0 <v,w>0 <v,x>0 <v,q>0
w : <w,s>0 <w,v>0 <w,x>0
x : <x,v>0 <x,w>0 <x,k>0 <x,m>0 <x,q>0
m : <m,x>0 <m,k>0 <m,q>0
k : <k,x>0 <k,m>0
q : <q,v>0 <q,x>0 <q,e>0 <q,m>0
e : <e,q>0

너비 우선 탐색 결과 : s v w x q k m e
깊이 우선 탐색 결과 : s v w x k m q e
---------- 문제 1번 실행을 종료합니다. --------------------------
```

문제 3. Hamlitonian path VS Eulerian path

- 다음 spectrum 에 대한 SBH problem 을 hamiltonian path approach 와 eulerian path approach 를 이용하여 구해 보시오

S = {AGT, AAA, ACT, AAC, CTT, GTA, TTT, TAA}

- Construct the graph with 8 vertices corresponding to these 3-mers (Hamiltonian path approach) and find a Hamiltonian path (7 edges) which visits each vertex exactly once. Write the string you got from the Hamiltonian path (여러개일 경우 전부 적을 것).
- Construct the graph with 8 edges corresponding to these 3-mers (Eulerian path approach) and find an Eulerian path (8 edges) which visits each edge exactly once. Write the string you got from the Hamiltonian path (여러개일 경우 전부 적을 것).

```
// 2016110056   박승원
#include<string>
#include<set>
#include"tgraph.h"
using namespace std;

int main()
{
    const string vert[] = {"AGT", "AAA", "ACT", "AAC", "CTT", "GTA", "TTT", "TAA"};

    Graph<string> gr;
    for(int i=0; i<8; i++) gr.insert_vertex(vert[i]);
    for(int i=0; i<8; i++) for(int j=0; j<8; j++)
        if(i != j && vert[i][1] == vert[j][0] && vert[i][2] == vert[j][1])
            gr.insert_edge(vert[i], vert[j], 0);
    cout << "Hamiltonian" << endl;
    gr.view();
    gr.depth();

    Graph<string> euler;
```

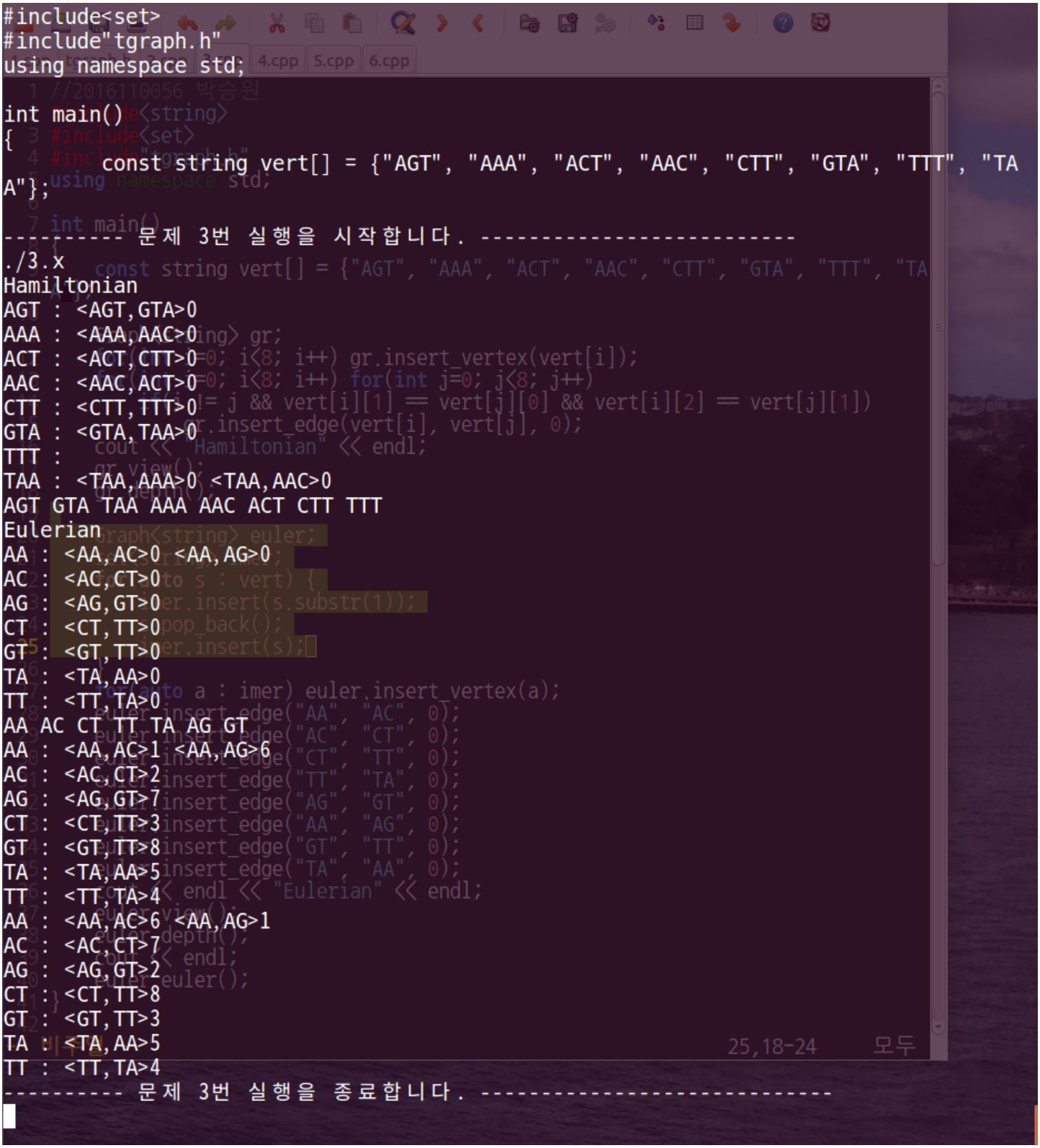
```
    set<string> imer;
    for(auto s : vert) {
        imer.insert(s.substr(1));
        s.pop_back();
        imer.insert(s);
    }
    for(auto a : imer) euler.insert_vertex(a);
    euler.insert_edge("AA", "AC", 0);
    euler.insert_edge("AC", "CT", 0);
    euler.insert_edge("CT", "TT", 0);
    euler.insert_edge("TT", "TA", 0);
    euler.insert_edge("AG", "GT", 0);
    euler.insert_edge("AA", "AG", 0);
    euler.insert_edge("GT", "TT", 0);
    euler.insert_edge("TA", "AA", 0);
    cout << endl << "Eulerian" << endl;
    euler.view();
    euler.depth();
    cout << endl;
    euler.euler();
}
```

프로그램에서 설정한대로 에지를 설정했을 경우, 해밀턴은 한 경로가 존재하고, 오일러에서는 결과에서 보여지듯이 두 가지의 경로가 존재한다.

해밀턴:AGTAAACTTT

오일러:AACTTAAGTT, AAGTTAACT

오일러에서 결과화면에 보이는 <AA,AC>뒤의 숫자가 순서이다. 프로그램에서 쓰인 그래프 헤더는 맨 마지막에 첨부하였다.



```
#include<set>
#include"tgraph.h"
using namespace std;

int main()
{
    const string vert[] = {"AGT", "AAA", "ACT", "AAC", "CTT", "GTA", "TTT", "TA
A"};

--------- 문제 3번 실행을 시작합니다. --------------------------
./3.x
Hamiltonian
AGT : <AGT,GTA>0
AAA : <AAA,AAC>0
ACT : <ACT,CTT>0
AAC : <AAC,ACT>0
CTT : <CTT,TTT>0
GTA : <GTA,TAA>0
TTT :
TAA : <TAA,AAA>0 <TAA,AAC>0
AGT GTA TAA AAA AAC ACT CTT TTT
Eulerian
AA : <AA,AC>0 <AA,AG>0
AC : <AC,CT>0
AG : <AG,GT>0
CT : <CT,TT>0
GT : <GT,TT>0
TA : <TA,AA>0
TT : <TT,TA>0
AA AC CT TT TA AG GT
AA : <AA,AC>1 <AA,AG>6
AC : <AC,CT>2
AG : <AG,GT>7
CT : <CT,TT>3
GT : <GT,TT>8
TA : <TA,AA>5
TT : <TT,TA>4
AA : <AA,AC>6 <AA,AG>1
AC : <AC,CT>7
AG : <AG,GT>2
CT : <CT,TT>8
GT : <GT,TT>3
TA : <TA,AA>5
TT : <TT,TA>4
25,18-24      모두
---------- 문제 3번 실행을 종료합니다. ----------------------------
```

```
void euler () {// for  directed  graph
    clearv () ;
    Vertex<T>* p;
    int edge_count = 0;
    for(p = root; p; p = p->vertex) {
        for(Edge<T>* e = p->edge; e; e = e->edge) {
            p->v++;//out grade
            e->vertex->v--;//in grade
            edge_count++;
        }
```

```
    }
    for(p = root; p; p = p->vertex) if(p->v == 1) break;// starting point
     euler_visit (p, 0, edge_count);
}

void euler_visit (Vertex<T>* p, int n, int edge_count) {
    if(n == edge_count) view(); // if all the edges are visited
    for(Edge<T>* e = p->edge; e; e = e->edge) {
        if(!e->v) {//if not visited
            e->v = 1;//mark visited
            e->weight = n+1;//use weight to show visit order
             euler_visit (e->vertex, n+1, edge_count);
            e->v = 0;//remove visited mark
        }
    }
}
```

문제 4. Re-sequencing problem

- Trivial Algorithm의 시간 복잡도
  - 3,000,000,000 length genome (N)
  - 300,000,000 reads to map (M)
  - Reads are of length 30 (L)
  - Number of mismatches allowed is 2 (D).
  - Each comparison of match vs. mismatch takes 1/1,000,000 seconds (t).
  - Total Time 을 수식 및 초, 년 단위로 구하시오.

30 *

3e+10 * 3e+09 / 1e+07 / 60 / 60 /24 /365

약800만년

문제 5. Indexing Algorithm

- Indexing Algorithm의 시간복잡도.
  - 3,000,000,000 length genome (N)
  - 300,000,000 reads to map (M)
  - Reads are of length 30 (L)
  - Number of mismatches allowed is 2 (D).
  - Each comparison of match vs. mismatch takes 1/1,000,000 seconds (t).
  - Total Time 을 수식 및 초, 년 단위로 구하시오.
  - L=45일때의 Total Time 을 수식 및 초, 시 단위로 구하시오 (다른 조건은 위와 같음).

3e+10/4**10 * 3e+9 /1e+7 /60 /60 /24 / 365

약 100일

3e+10/4**15 * 3e+9 /1e+7 /60 /60 /24

약 2.3시간

Listing 1: tgraph.h

```
#pragma once
#include<climits>
#include<cassert>
#include<fstream>
#include<iostream>
#include<map>
#include<vector>
#include<stack>
#define min(a, b) a < b ? a : b

template <typename T> class Vertex;

template<typename T> class Edge
{
public:
```

```
    int weight = 0;
    int v = 0; // for visit check or other uses like dijkstra route check
    struct Vertex<T>* vertex = nullptr ;
    struct Edge<T>* edge = nullptr;
};

template<typename T> class Vertex
{
public:
    T data;
    int v = 0; // for visit check or other uses like parity bit
    struct Edge<T>* edge = nullptr;
    struct Vertex<T>* vertex = nullptr ;
};

/// This class does not make its own data structure it just deals with pointers
///and allocate memory for data. Thus enhance interoperability with C style data
/// structure . It arange data like below. V->vertex always points to the next line ,
///E->edge always points to the next one, while E->vertex point to the other
///edge irregularly with direction .
/// V - E - E - E - E
/// V - E - E
/// V - E - E - E
template<typename T> class Graph
{
public:
    virtual ~Graph() {
        gfree(root);
        root = nullptr ;
    }

    void insert_vertex (T n) {
        root = insert(root, n);
    }

    void insert_edge (T a, T b, int weight) {
        Vertex<T> *va, *vb;
        for(Vertex<T>* p = root; p; p = p->vertex) {
            if(p->data == a) va = p;
            if(p->data == b) vb = p;
        }
        va->edge = insert(va->edge, vb, weight);
```

```
}

void read_file (std :: string file ) {
    T n1, n2;
    std :: ifstream f( file );
    int vc;
    f >> vc;
    for(int i=0; i<vc; i++) {
        f >> n1;
        insert_vertex (n1);
    }
    int wt;
    while(f >> n1 >> n2 >> wt) insert_edge(n1, n2, wt);
}

void view() {
    for(Vertex<T>* p = root; p; p = p->vertex) {
        std :: cout << p->data << " : ";
        for(Edge<T>* e = p->edge; e; e = e->edge)
            if (e->vertex) std :: cout << '<' << p->data << ',' << e->vertex->data << '>' << e
                ->weight << ' ';
        std :: cout << std::endl;
    }
}

Vertex<T>* data() {// this is to make compatible with C structure ,
    return root; // return root pointer , GraphV will access it .
}

void prim() {
    clearv () ;
    root->v = 1;//below for syntax is just to call n-1 times
    for(Vertex<T>* q = root->vertex; q; q = q->vertex) shortest(root);
}

void topology() {//check v to entry and print data
    clearv () ;
    for(Vertex<T>* q; q = find_entry (root) ;) {
        q->v = 1;
        std :: cout << q->data << " - ";
    }
}
```

```
int floyd(T a, T b) {
    clearv();
    std::map<T, std::map<T, int>> A;
    for(Vertex<T>* q = root; q; q = q->vertex)
        for(Vertex<T>* p = root; p; p = p->vertex)
            A[q->data][p->data] = INT_MAX / 2;
    for(Vertex<T>* q = root; q; q = q->vertex)
        for(Edge<T>* e = q->edge; e; e = e->edge)
            A[q->data][e->vertex->data] = e->weight;

    for(Vertex<T>* k = root; k; k = k->vertex)
        for(Vertex<T>* i = root; i; i = i->vertex)
            for(Vertex<T>* j = root; j; j = j->vertex)
                A[i->data][j->data] = min(A[i->data][j->data],
                        A[i->data][k->data] + A[k->data][j->data]);
    return A[a][b];
}

int dijkstra(T a, T b) {
    clearv();
    distance.clear();
    for(Vertex<T>* p = root; p; p = p->vertex) distance[p] = INT_MAX / 2;
    Vertex<T>* pa = find(root, a); assert(pa);
    Vertex<T>* pb = find(root, b); assert(pb);
    distance[pa] = 0;
    while(pb != find_closest());
    for(auto& a : waypoint[pb]) a->v = 1;
    return distance[pb];
}

void depth() {
    clearv();
    depth(root);
}

void breadth() {
    clearv();
    std::cout << root->data << ' ';
    root->v = 1;
    breadth(root);
}
```

```
void euler () { // for directed graph
    clearv () ;
    Vertex<T>* p;
    int edge_count = 0;
    for(p = root; p; p = p->vertex) {
        for(Edge<T>* e = p->edge; e; e = e->edge) {
            p->v++;//out grade
            e->vertex->v--;//in grade
            edge_count++;
        }
    }
    for(p = root; p; p = p->vertex) if(p->v == 1) break;// starting point
     euler_visit (p, 0, edge_count);
}

void euler_visit (Vertex<T>* p, int n, int edge_count) {
    if(n == edge_count) view(); // if all the edges are visited
    for(Edge<T>* e = p->edge; e; e = e->edge) {
        if(!e->v) {//if not visited
            e->v = 1;//mark visited
            e->weight = n+1;//use weight to show visit order
             euler_visit (e->vertex, n+1, edge_count);
            e->v = 0;//remove visited mark
        }
    }
}


void bridge () {
    clearv () ;
    std :: vector<Edge<T>*> v;
    for(Vertex<T>* p = root; p; p = p->vertex)
        for(Edge<T>* e = p->edge; e; e = e->edge)
            if( is_bridge (p, e)) v.push_back(e);
    for(auto& a : v) a->v = 1;
}

void greedy() {
    clearv () ;
    union_set . clear () ;
    int i = 1;
```

```
        for(Vertex<T>* p = root; p; p = p->vertex) union_set[p] = i++;
        std :: vector<Edge<T>*> v;
        for(Vertex<T>* p = root->vertex; p; p = p->vertex) v.push_back(find_greed () );
        clearv () ;
        for(auto& a : v) a->v = 1;
    }

protected:
    Vertex<T>* root = nullptr ;

    Vertex<T>* insert(Vertex<T>* p, T n) {// recursively insert a value 'n'
        if (!p) {
            p = new Vertex<T>;
            p->data = n;
            return p;
        }
        p->vertex = insert(p->vertex, n);
        return p;
    }

    Edge<T>* insert(Edge<T>* e, Vertex<T>* v, int weight) {
        if (!e) {// recursively insert edge in at the end of e pointint to v
            e = new Edge<T>;
            e->vertex = v;
            e->weight = weight;
            return e;
        }
        e->edge = insert(e->edge, v, weight);
        return e;
    }

    bool is_bridge (Vertex<T>* p, Edge<T>* eg) {//check all the bridges in the graph
        eg->v = 1;
        for(Edge<T>* e = eg->vertex->edge; e; e = e->edge)
            if(e->vertex == p) e->v = 1;
        depth(eg->vertex);
        bool r = !p->v;
        clearv () ;
        return r;
    }

private:
```

```
std :: map<Vertex<T>*, int> distance;
std :: map<Vertex<T>*, std::vector<Edge<T>*>> waypoint;
std :: map<Vertex<T>*, int> union_set;

Vertex<T>* find_closest () {// dijkstra
    int min = INT_MAX / 2;
    Vertex<T>* p = nullptr;
    for(auto& a : distance) if(min > a.second && !a.first->v) {
        p = a.first;
        min = a.second;
    }
    p->v = 1;
    for(Edge<T>* e = p->edge; e; e = e->edge) if(!e->vertex->v) {
        if( distance[e->vertex] > distance[p] + e->weight) {
            distance[e->vertex] = distance[p] + e->weight;
            waypoint[e->vertex] = waypoint[p];
            waypoint[e->vertex].push_back(e);
        }
    }
    // for(auto& a : distance) std :: cout << a.first->data << ' ' << a.second <<
    //   ' ' << a.first->v << std::endl;
    return p;
}

Vertex<T>* find(Vertex<T>* p, T n) {// find value 'n' and return the address
    for(Vertex<T>* v = p; v; v = v->vertex) if(v->data == n) return v;
}

void depth(Vertex<T>* p) {//depth search the graph
    assert(p);
    p->v = 1;
    std :: cout << p->data << ' ';
    for(Edge<T>* e = p->edge; e; e = e->edge) {
        if(!e->vertex->v && !e->v) {//!e->v is for bridge func
            e->v = 1;
            depth(e->vertex);
        }
    }
}

void breadth(Vertex<T>* p) {
    std :: vector<Vertex<T>*> q;
```

```
    for(Edge<T>* e = p->edge; e; e = e->edge) {
        if (!e->vertex->v) {
            q.push_back(e->vertex);
            e->v = 1;
            std :: cout << e->vertex->data << ' ';
            e->vertex->v = 1;
        }
    }
    for(auto& a : q) breadth(a);
}

void efree (Edge<T>* e) {
    if (!e) return;
    efree (e->edge);
    delete e;
}

void gfree (Vertex<T>* p) {
    if (!p) return;
    efree (p->edge);
    gfree (p->vertex);
    delete p;
}

void clearv () {
    for(Vertex<T>* p = root; p; p = p->vertex) {
        p->v = 0;
        for(Edge<T>* e = p->edge; e; e = e->edge) e->v = 0;
    }
}

void shortest (Vertex<T>* p) {//prim
    Edge<T>* me;
    int min = INT_MAX;
    for (; p; p = p->vertex) {
        if (p->v) {
            for(Edge<T>* e = p->edge; e; e = e->edge) {
                if (e->v) continue;
                if (e->weight < min && !e->vertex->v) {
                    min = e->weight;
                    me = e;
                }
```

```
            }
        }
    }
    me->v = 1;
    me->vertex->v = 1;
}

Vertex<T>* find_entry(Vertex<T>* p) {//return  NULL when no more,topology
    for(Vertex<T>* q = p; q;  q = q->vertex) {
        if(q->v != 1) {//1  or  2  or  0
            for(Edge<T>* e = q->edge; e; e = e->edge) {
                if (!e->vertex->v) e->vertex->v = 2;//entry marking
            }
        }
    }
    Vertex<T>* r = NULL;
    for(Vertex<T>* q = p; q;  q = q->vertex) {
        if(q->v == 2) q->v = 0;
        else  if(!q->v) r = q;
    }
    return r;
}

void unite(Vertex<T>* a, Vertex<T>* b) {//greed
    for(auto& u :  union_set)
        if(u.second == union_set[a])  u.second = union_set[b];
}

Edge<T>* find_greed() {
    int  min = INT_MAX / 2;
    Edge<T>* eg;
    Vertex<T>* vt;
    for(Vertex<T>* p = root;  p;  p = p->vertex) {
        for(Edge<T>* e = p->edge; e; e = e->edge) {
            if (!e->v && e->weight < min) {
                min = e->weight;
                eg = e;
                vt = p;
            }
        }
    }
    eg->v = 1;
```

```
        if(union_set[vt] != union_set[eg->vertex]) {
            unite(vt, eg->vertex);
            return eg;
        } else return find_greed();
    }
};
```