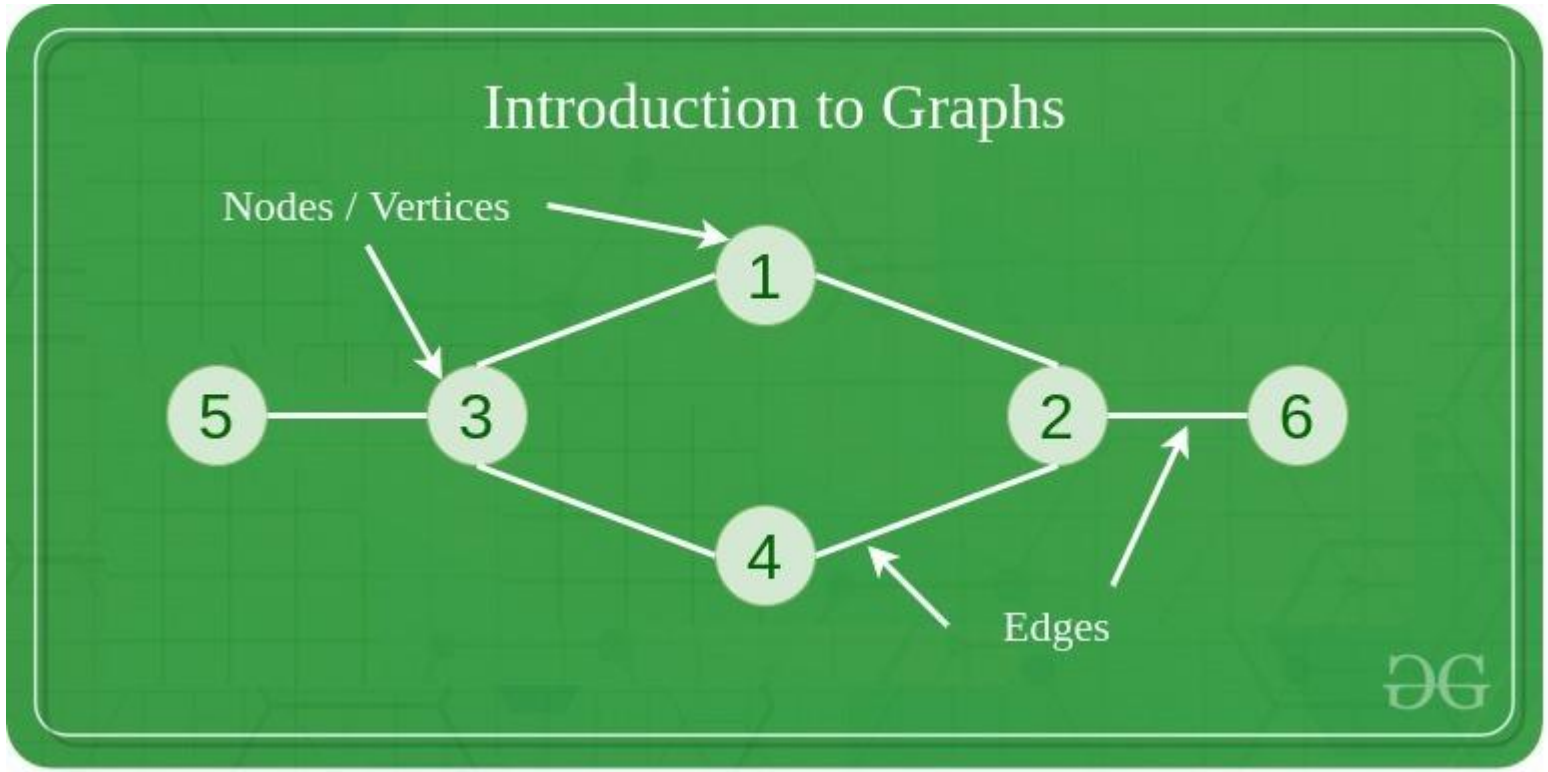
OOP & data struct

14. Graph

BY SOMSIN THONGKRAIRAT

Graph



https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Graph

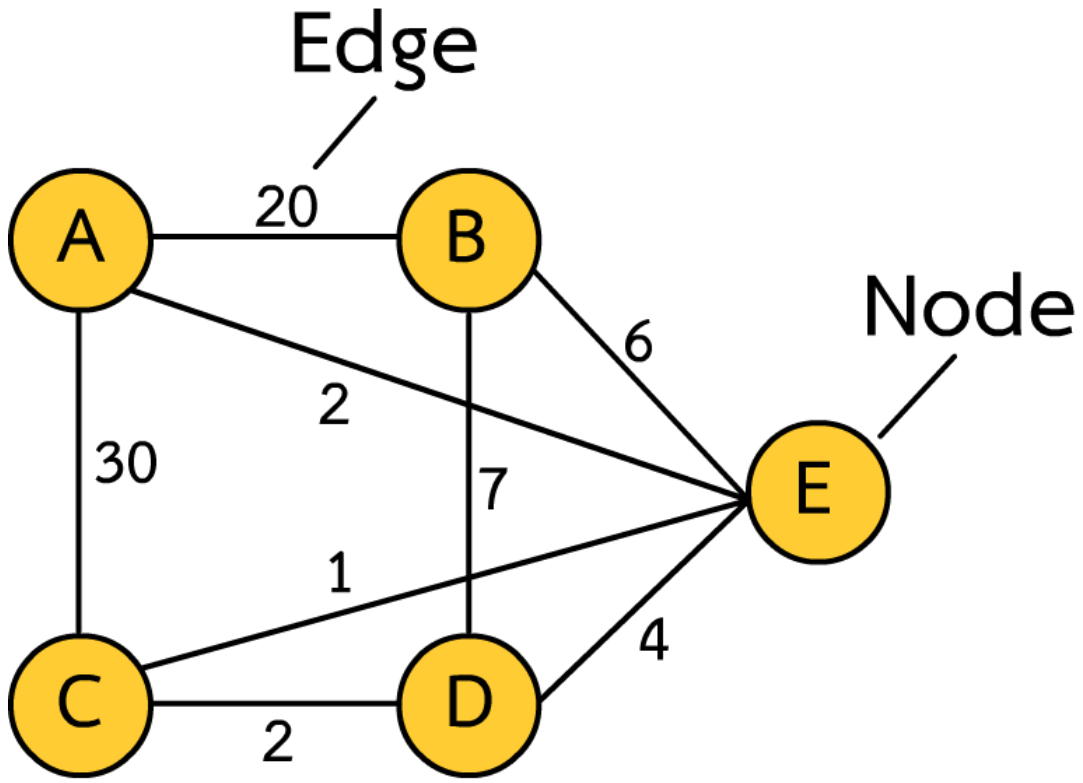
- fundamental non-linear data structure consisting of node(vertex) and path(edge)
- node(vertex) is an item or element in structure
- path(edge) is the information that how each node connected

- เป็น structure พื้นฐานแบบ non-linear ที่ประกอบไปด้วย node(vertex) และ path(edge)
- node(vertex) คือ item หรือ element ใน structure
- path(edge) คือข้อมูลที่บอกว่าแต่ละ node เชื่อต่อกันอย่างไร

Component of graph

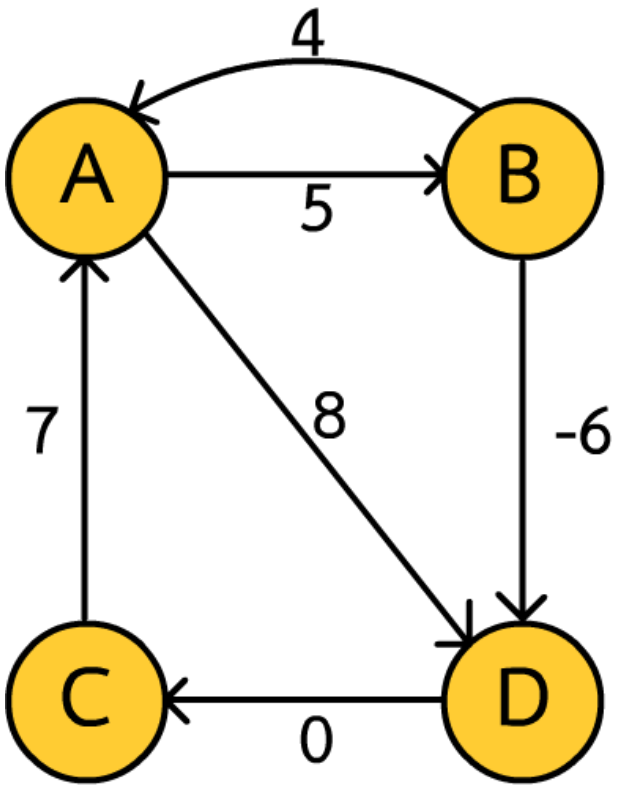
- Node contain data object
- Edge contain travel cost between node

- Node เก็บข้อมูลของแต่ละ object
- Edge เก็บ cost ของการเดินทางระหว่าง node



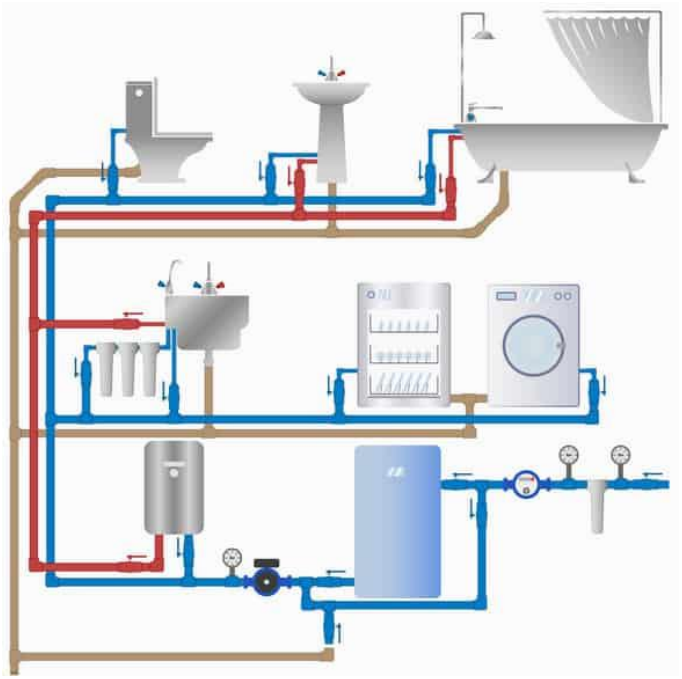
type of graph

- There are several type of graph classified by rule
- graph มีหลายชนิดจัดประเภทตามกฎ



- Direct / undirect
- Non-negative edge / floating point edge
- Cyclic graph / Acyclic graph

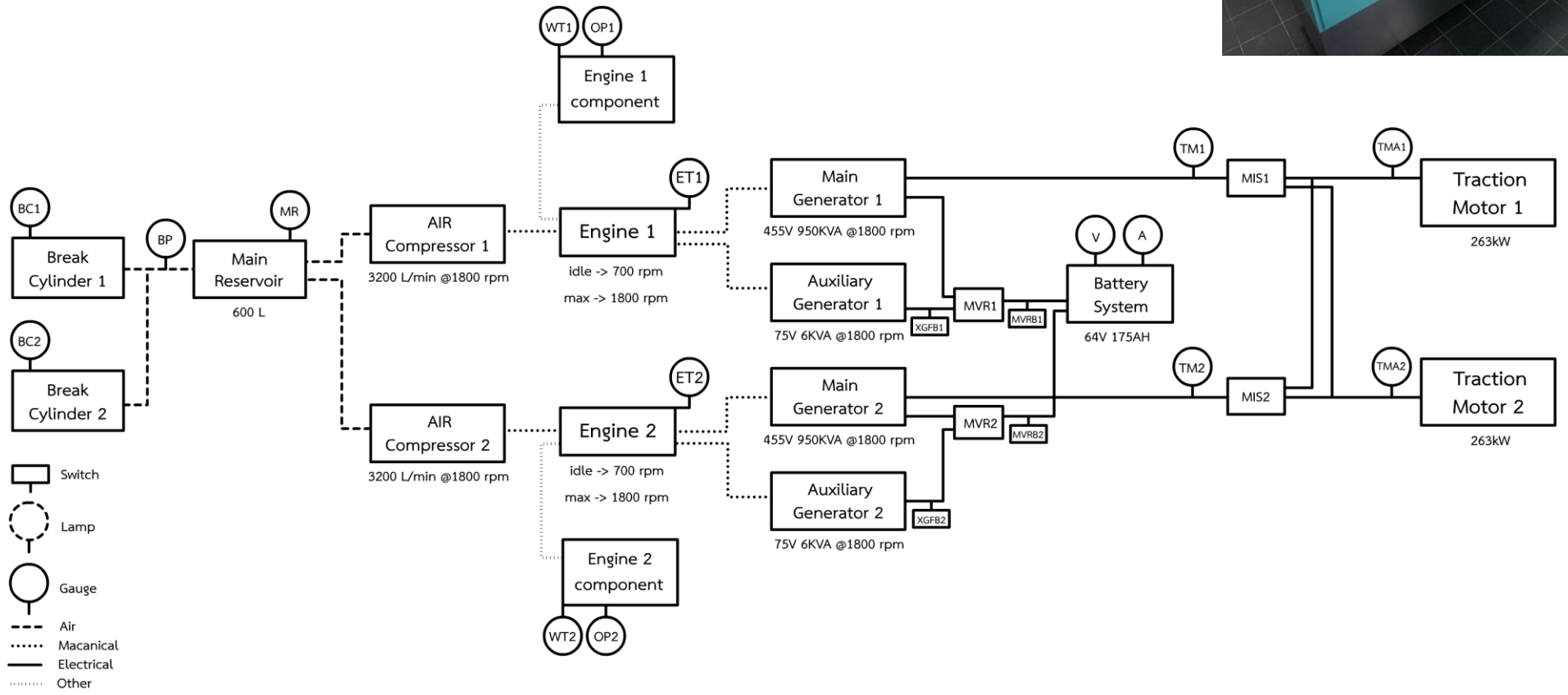
Depend on purpose



https://dreamcivil.com/systems-of-plumbing/

https://junkee.com/google-maps-has-been-tracking-your-every-move-and-theres-a-website-to-prove-it/39639

Depend on purpose

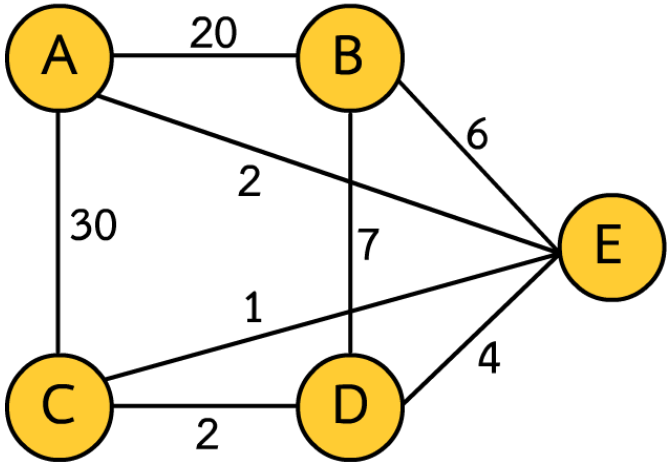


Our scope

- Non-negative edge undirected cyclicable graph

  - Edge can’t be negative cost
  - All edge can use any direction to travel

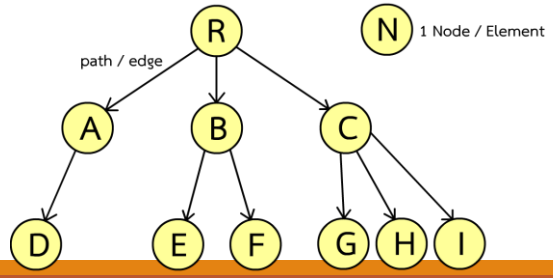
  - แต่ละ edge จะไม่มี cost ที่ติดลบ
  - ทุก edge ใช้เส้นทางใดก็ได้

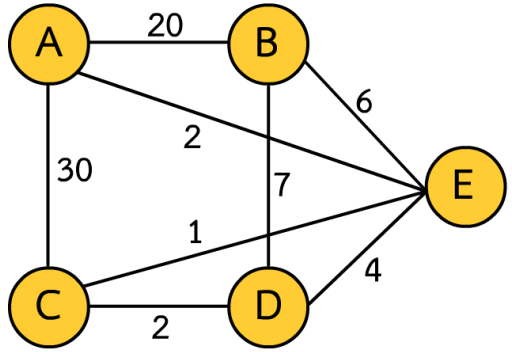


Before we start

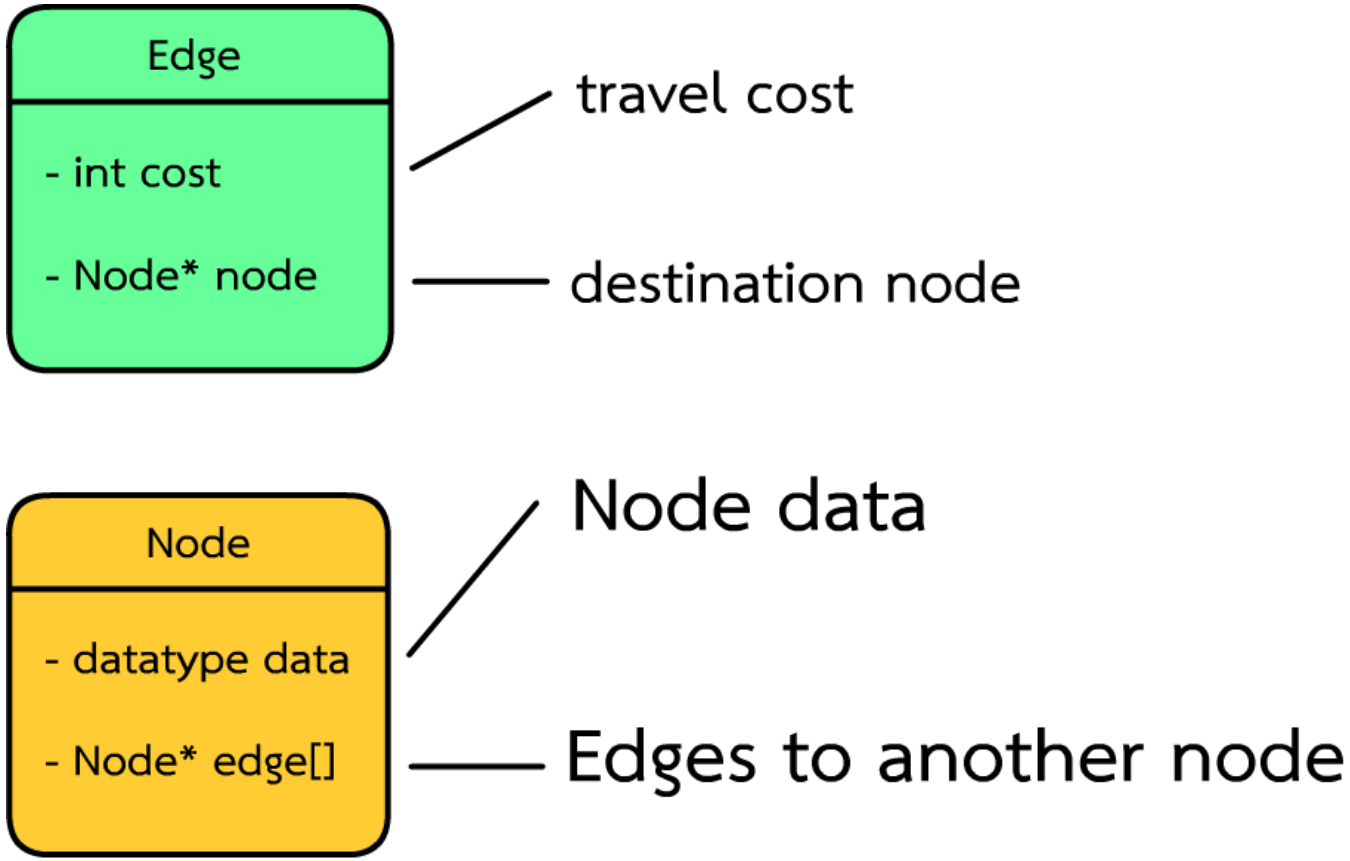
- Tree is subset of graphs with restricted rule, So tree always a graph but not all graph will be tree

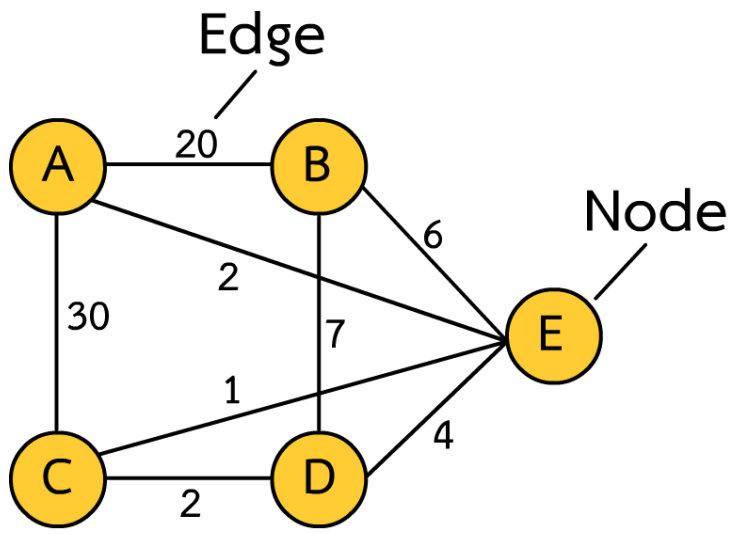
- Tree เป็น subset ของ graph ที่มีกฎของตัวเอง ดังนั้น tree ทุกตัวจะเป็น graph แต่ไม่ใช่ว่า graph ทุกตัวจะเป็น tree





Data structure object base graph (node & edge)

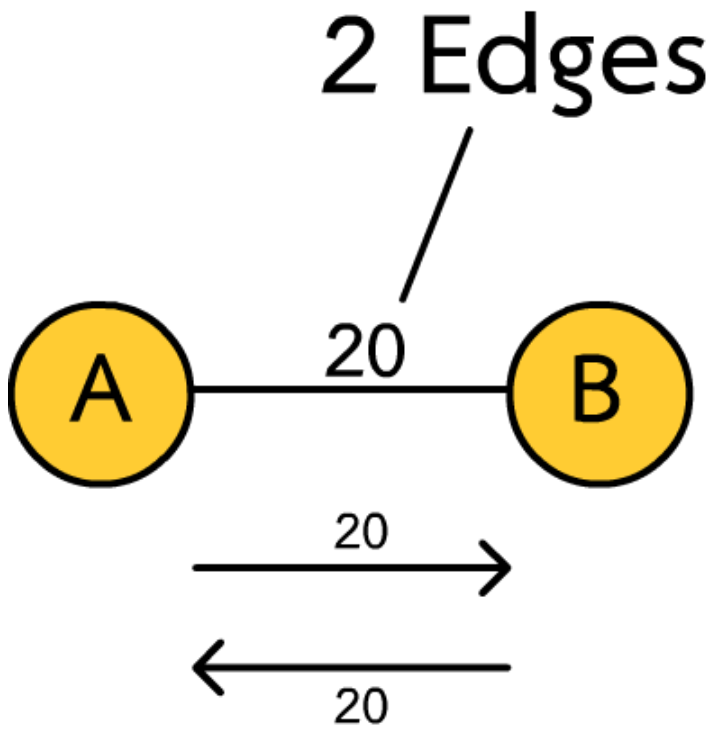




Edge (undirected graph)

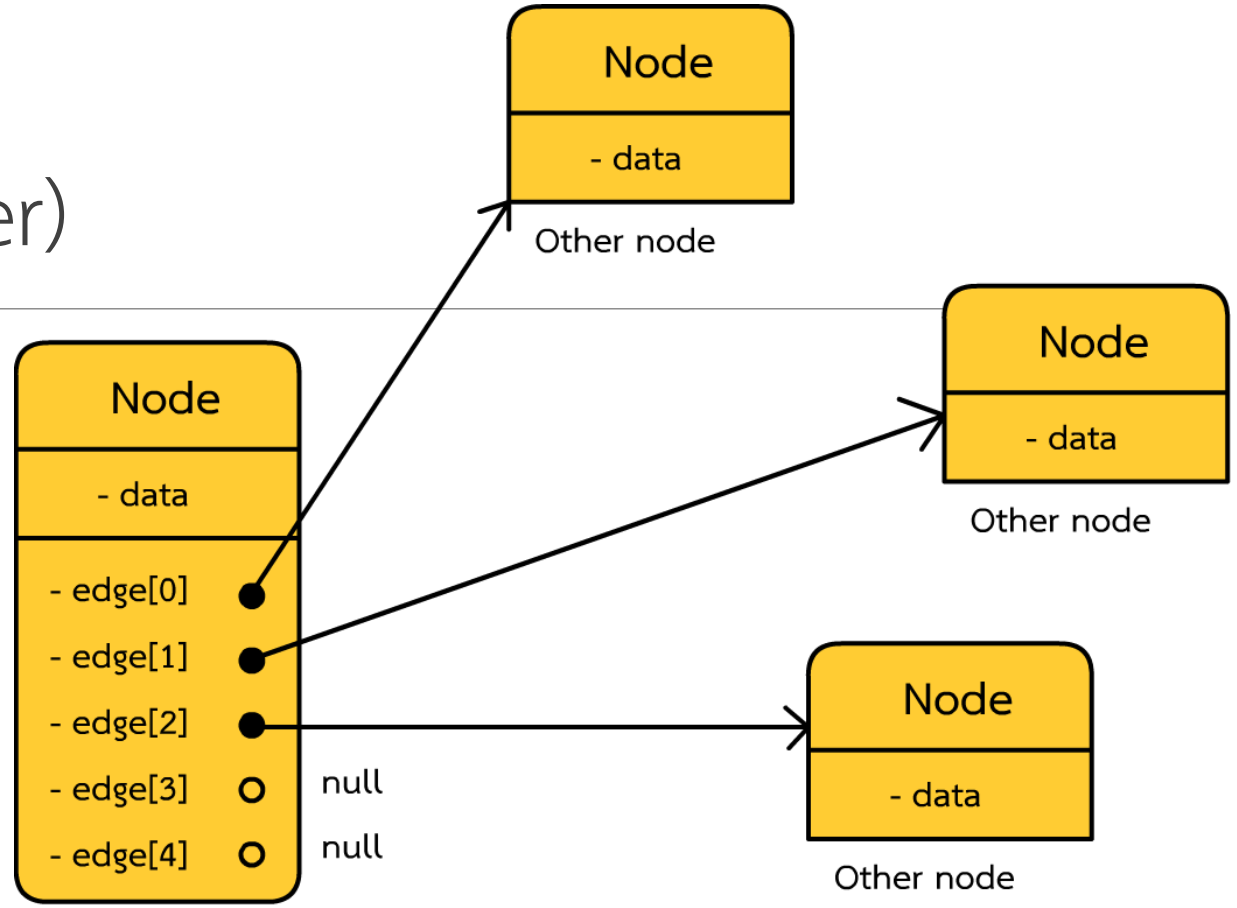
```
class Edge{
    public :
    int cost;
    Node* node;

    Edge(int _cost,Node* dest){
        cost = _cost;
        node = dest;
    }
};
```



Code (fixed node number)

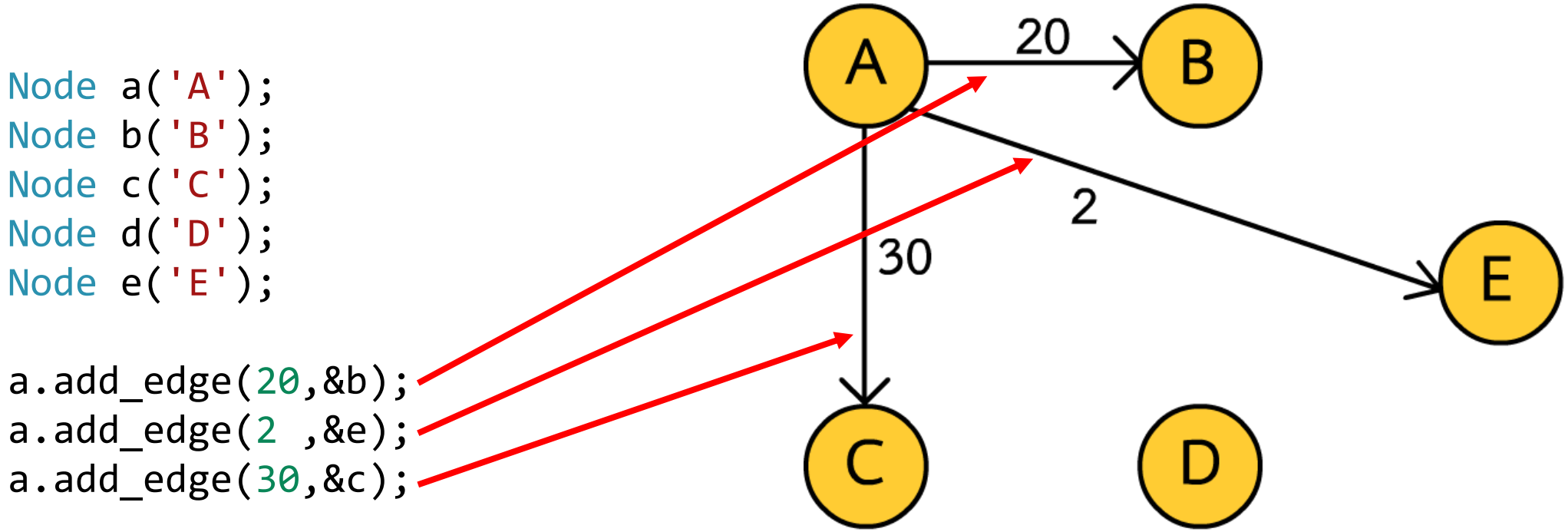
```
class Node{
public :
    char data;
    int edge_count;
    Edge* edges[5];
    Node(char item){
        data = item;
        edge_count = 0;
    }
    void add_edge(int cost,Node* n){
        edges[edge_count] = new Edge(cost,n);
        edge_count++;
    }
};
```



Build graph (each node)

```
Node a('A');
Node b('B');
Node c('C');
Node d('D');
Node e('E');

a.add_edge(20,&b);
a.add_edge(2 ,&e);
a.add_edge(30,&c);
```



Now only one way

Build graph

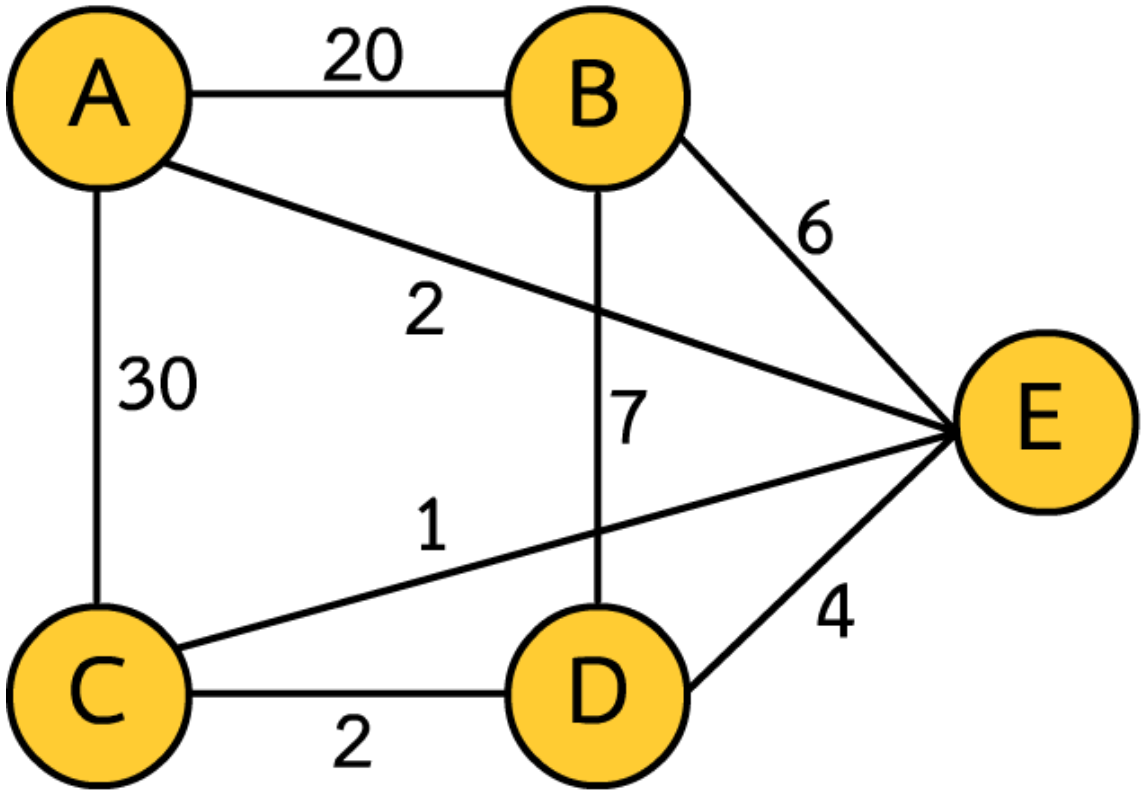
```
a.add_edge(20,&b);
a.add_edge(2 ,&e);
a.add_edge(30,&c);

b.add_edge(20,&a);
b.add_edge(7 ,&d);
b.add_edge(6 ,&e);

c.add_edge(30,&a);
c.add_edge(1 ,&e);
c.add_edge(2 ,&d);

d.add_edge(2 ,&c);
d.add_edge(7 ,&b);
d.add_edge(4 ,&e);

e.add_edge(6 ,&b);
e.add_edge(2 ,&a);
e.add_edge(1 ,&c);
e.add_edge(4 ,&d);
```



Print graph (node)

```
void print(){ // method in node
    cout << data << "-> ";
    for(int i=0;i<edge_count;i++){
        cout << "(" << edges[i]->cost <<","<< (edges[i]->node)->data << ")";
    }
    cout << endl;
}

a.print();
b.print();
c.print();
d.print();
e.print();
```

Result :

A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

C-> (30,A)(1,E)(2,D)

D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)

Result :

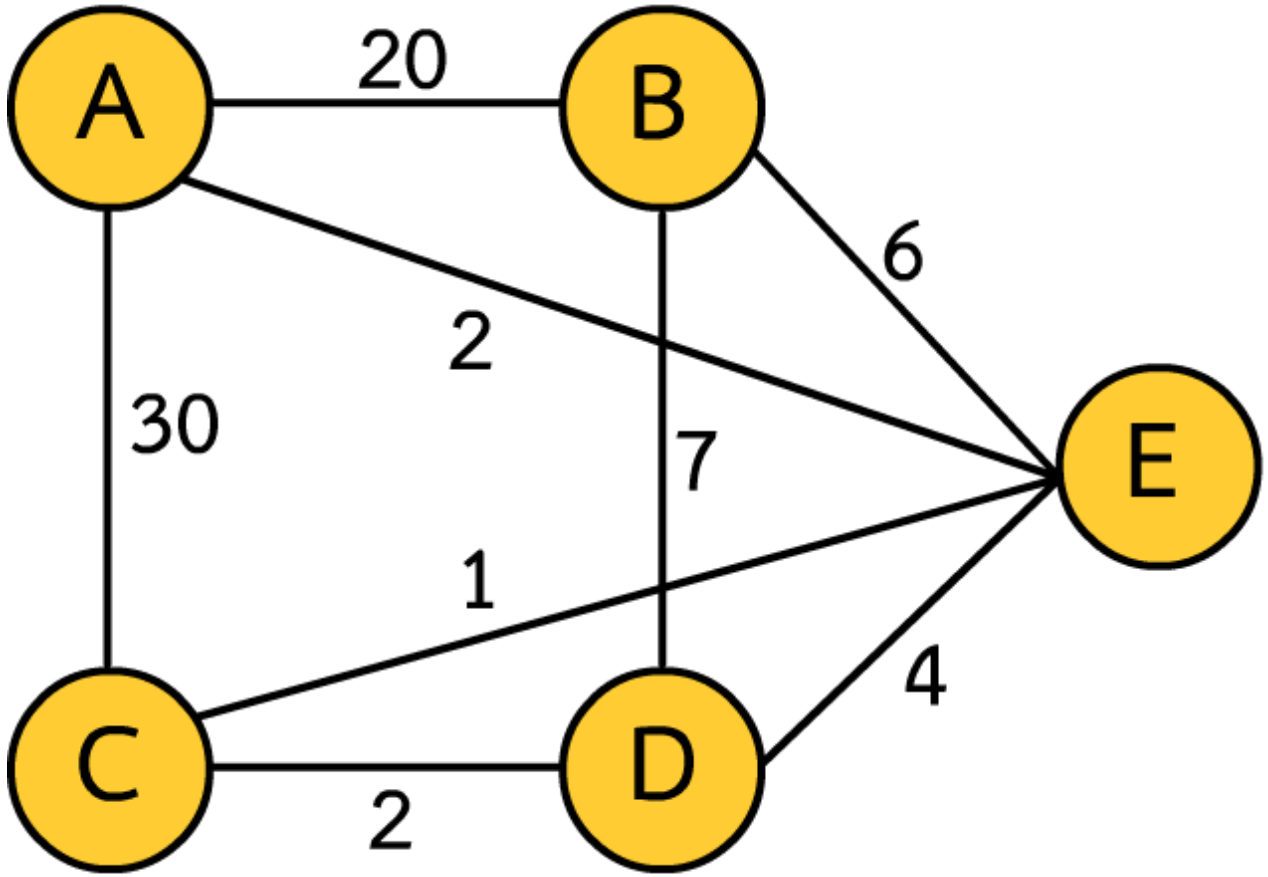
A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

C-> (30,A)(1,E)(2,D)

D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)



Access

- Can start from any node / สามารถเริ่มใช้จาก node ใดก็ได้

Result :

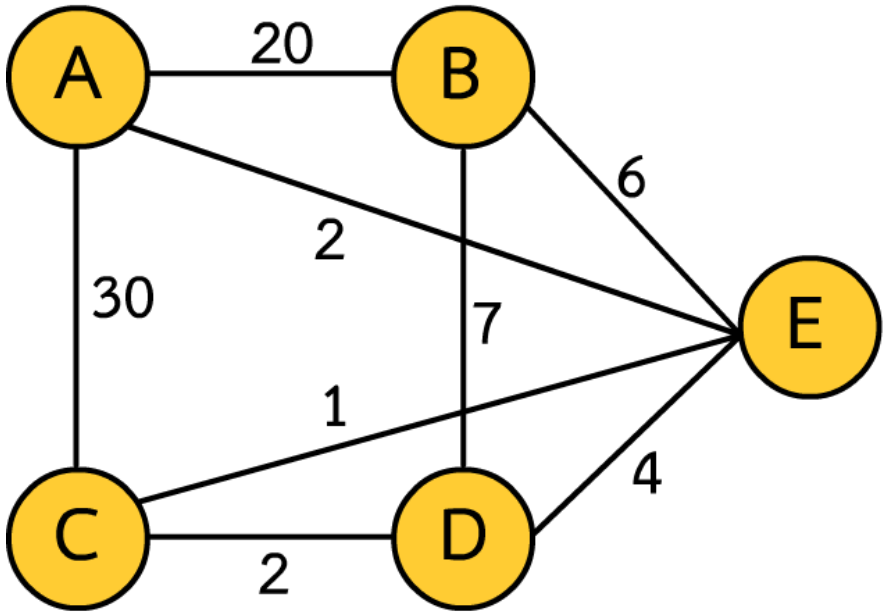
A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

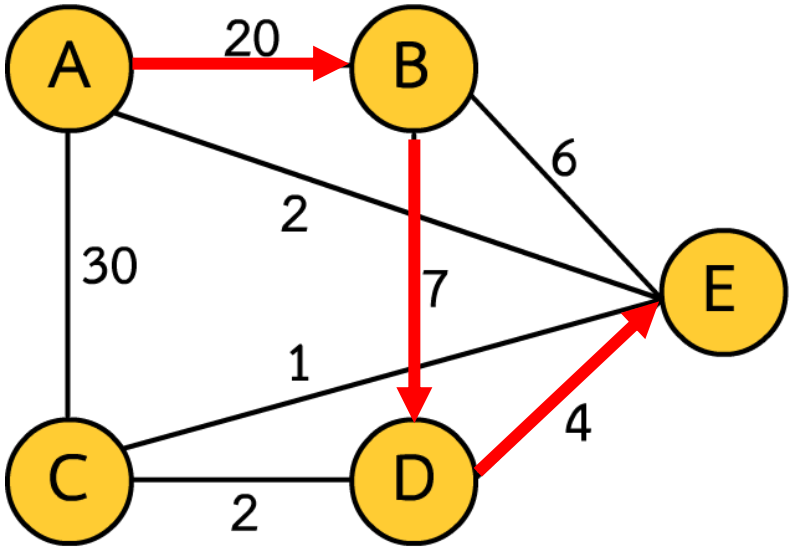
C-> (30,A)(1,E)(2,D)

D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)



example



A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

C-> (30,A)(1,E)(2,D)

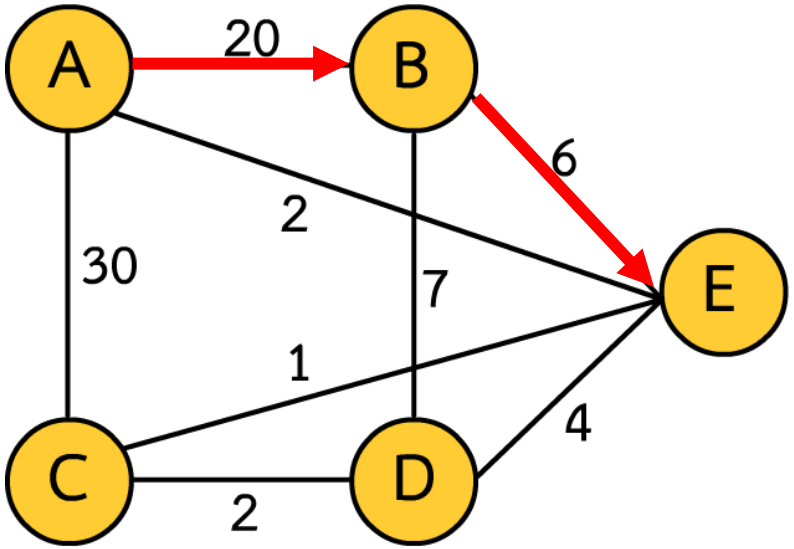
D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)

How much total cost for this path from A to E

```
cout << a.edges[0]->cost + b.edges[1]->cost + d.edges[2]->cost << endl; // 31
```

example



A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

C-> (30,A)(1,E)(2,D)

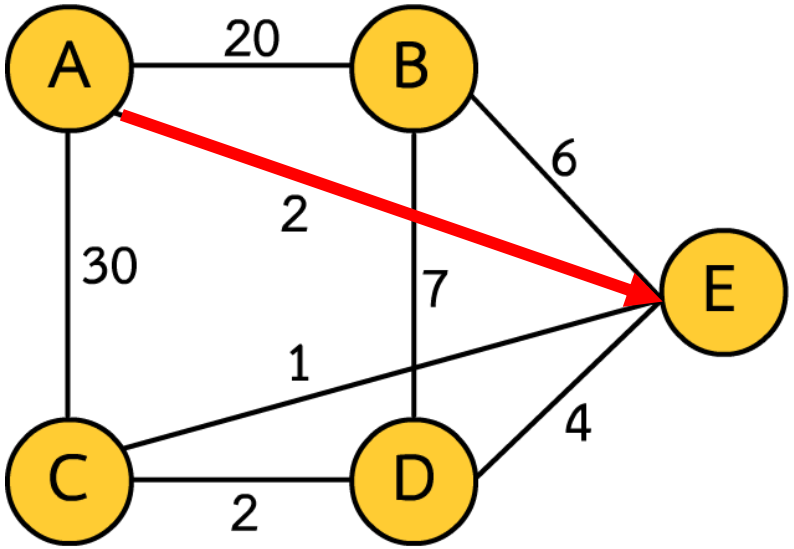
D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)

How much total cost for this path from A to E

```
cout << a.edges[0]->cost + b.edges[2]->cost << endl; // 26
```

example



A-> (20,B)(2,E)(30,C)

B-> (20,A)(7,D)(6,E)

C-> (30,A)(1,E)(2,D)

D-> (2,C)(7,B)(4,E)

E-> (6,B)(2,A)(1,C)(4,D)

How much total cost for this path from A to E

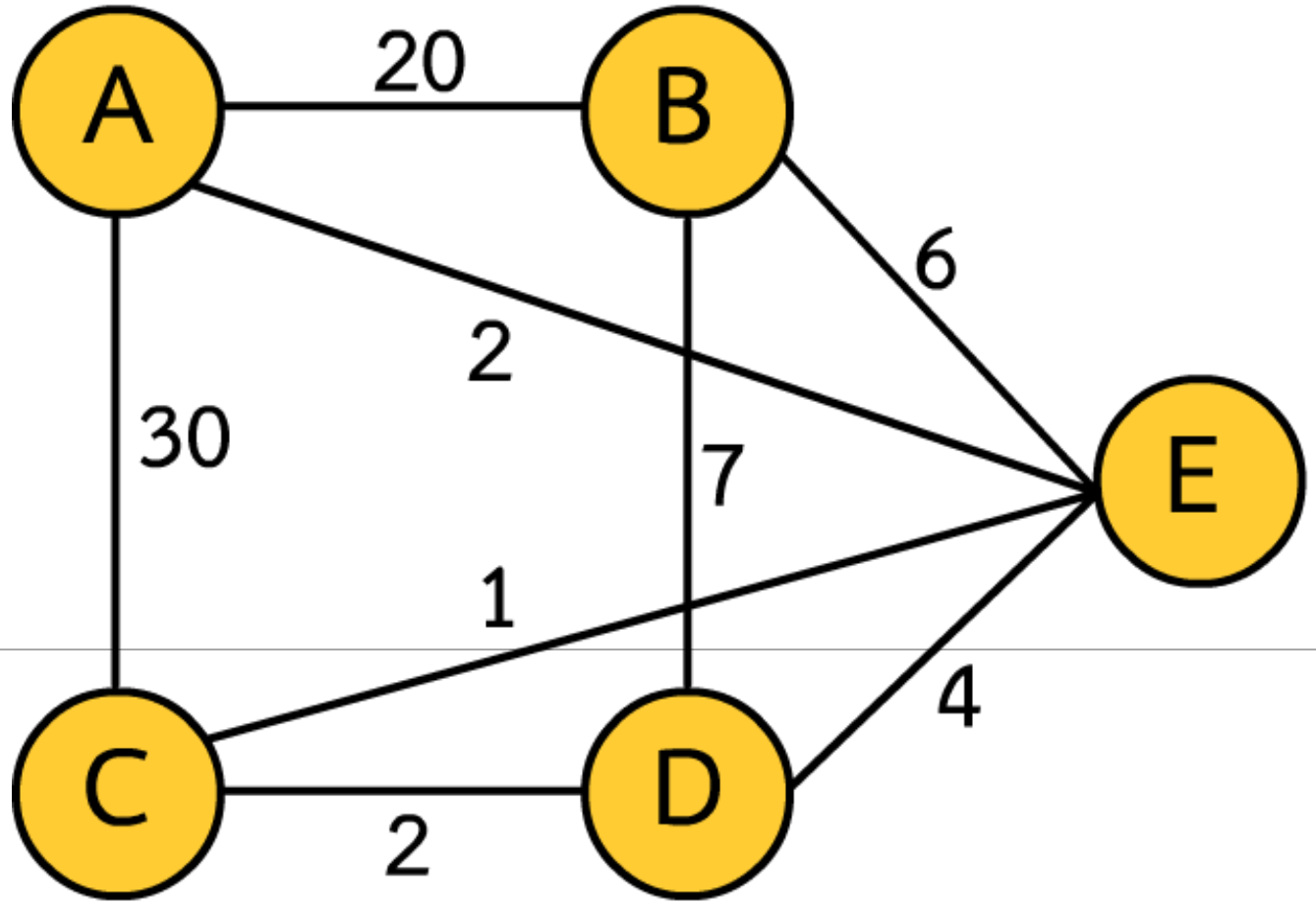
```
cout << a.edges[1]->cost << endl; // 2
```

End of graph structure

Happy?

Quiz

SHORTEST PATH?

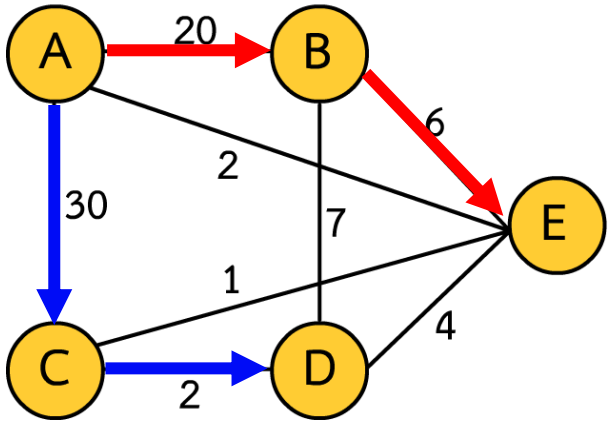


Why graph

- more flexible than tree
- more complex algorithm implementation***
- seem like realistic modelling (such as networking traffic)
- ยืดหยุ่นกว่า tree
- สามารถใช้ algorithm ที่ซับซ้อนได้มากกว่า***
- สามารถ modeling ให้เหมือนกันความเป็นจริงได้ เช่น networking การจราจร

Example program

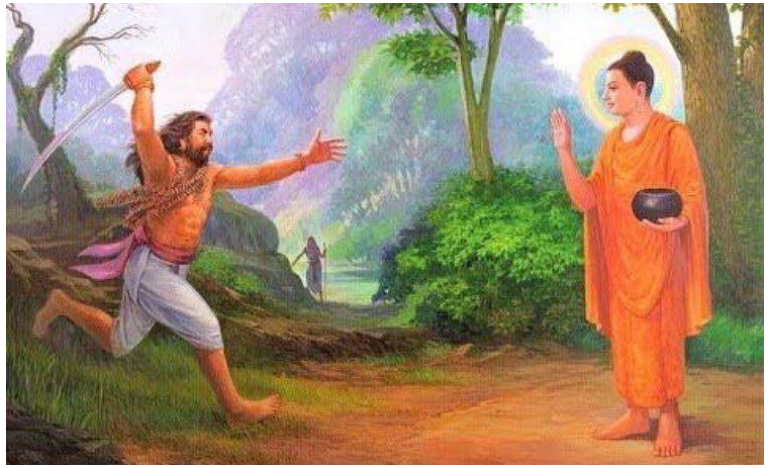
- second order visited
  - List all possibility node that can visit with in 2 move
  - แสดง node ทั้งหมดที่เป็นไปได้ ที่สามารถเข้าถึงได้ด้วยการเดิน 2 ครั้ง



Remark ***

- following algorithm is just a partial algorithm can’t use in real life application

- algorithm ต่อไปนี้เป็นเพียงส่วนหนึ่งของ algorithm เท่านั้นไม่สามารถนำไปใช้จริงได้



second order visited

```
void second_order_visited(Node* n){
    for(int i = 0;i<n->edge_count;i++){ // like print
        Node* node1 = (n->edges[i])->node;
        int cost1 = (n->edges[i])->cost;
        for(int j=0;j<node1->edge_count;j++){
            Node* node2 = (node1->edges[j])->node;
            int cost2 = cost1 + (node1->edges[j])->cost;
            cout << n->data << "->" << node1->data << "->" << node2->data
                 << " = " << cost2 << endl;
        }
    }
}
```

second order visited

```
second_order_visited(&a);
```

A->B->A = 40

A->B->D = 27

A->B->E = 26

A->E->B = 8

A->E->A = 4

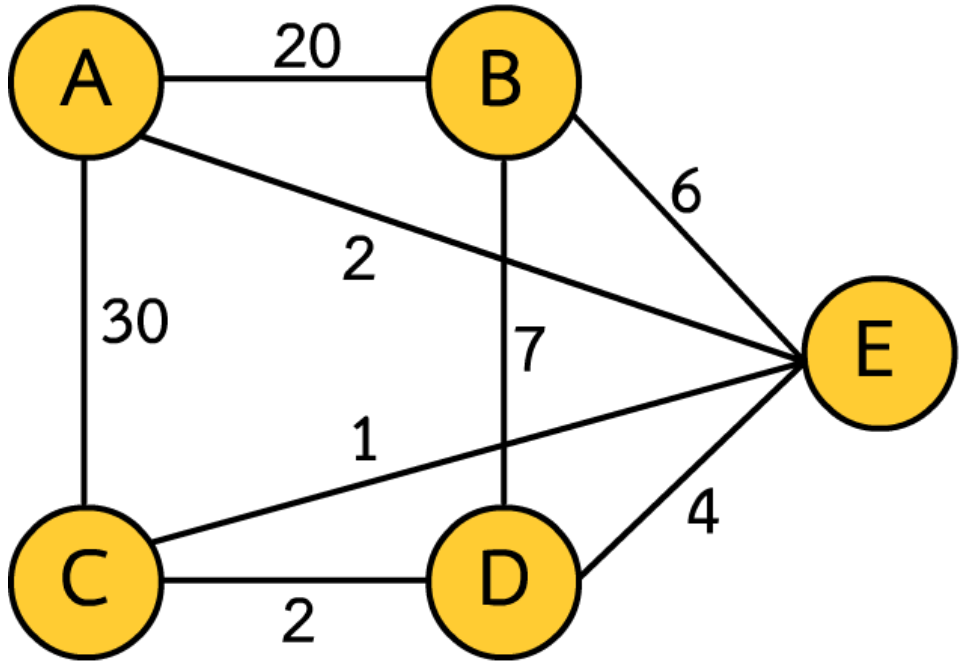
A->E->C = 3

A->E->D = 6

A->C->A = 60

A->C->E = 31

A->C->D = 32

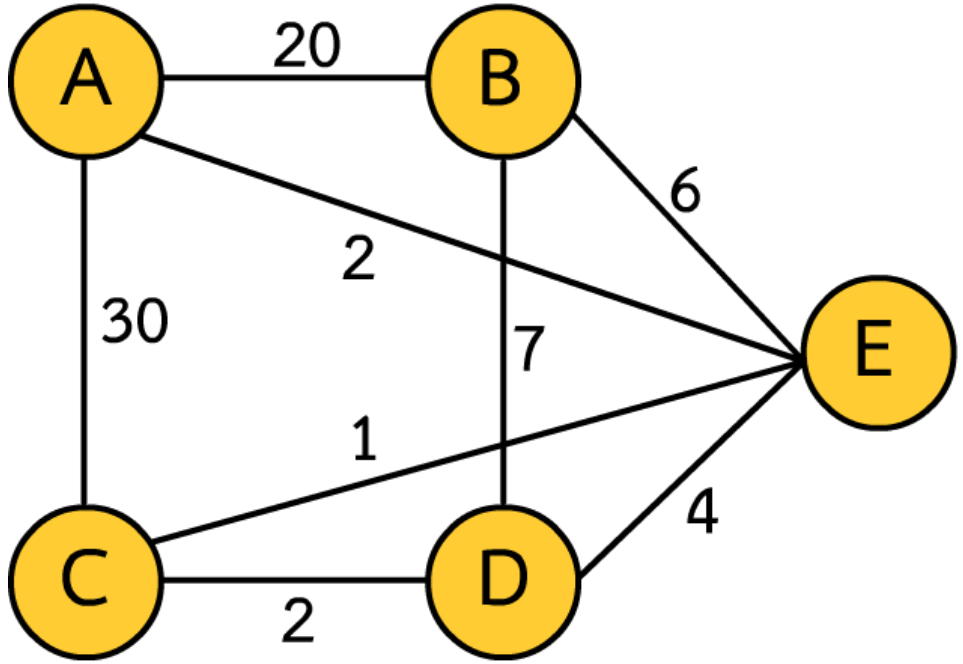


third order visited

```
void third_order_visited(Node* n){
    for(int i = 0;i<n->edge_count;i++){ // like print
        Node* node1 = (n->edges[i])->node;
        int cost1 = (n->edges[i])->cost;
        for(int j=0;j<node1->edge_count;j++){
            Node* node2 = (node1->edges[j])->node;
            int cost2 = cost1 + (node1->edges[j])->cost;
            for(int k=0;k< node2->edge_count;k++){
                Node* node3 = (node2->edges[k])->node;
                int cost3 = cost2 + (node2->edges[k])->cost;
                cout << n->data << "->" << node1->data << "->" << node2->data
                     << "->" << node3->data << " = " << cost3 << endl;
            }
        }
    }
}
```

third order visited

A->B->A->B = 60
A->B->A->E = 42
A->B->A->C = 70
A->B->D->C = 29
A->B->D->B = 34
A->B->D->E = 31
A->B->E->B = 32
A->B->E->A = 28
A->B->E->C = 27
A->B->E->D = 30
A->E->B->A = 28
A->E->B->D = 15
A->E->B->E = 14
A->E->A->B = 24
A->E->A->E = 6
A->E->A->C = 34
A->E->C->A = 33
A->E->C->E = 4
A->E->C->D = 5
A->E->D->C = 8
A->E->D->B = 13
A->E->D->E = 10
A->C->A->B = 80
A->C->A->E = 62
A->C->A->C = 90
A->C->E->B = 37
A->C->E->A = 33
A->C->E->C = 32
A->C->E->D = 35
A->C->D->C = 34
A->C->D->B = 39
A->C->D->E = 36



Forth? Fifth? Sixth?

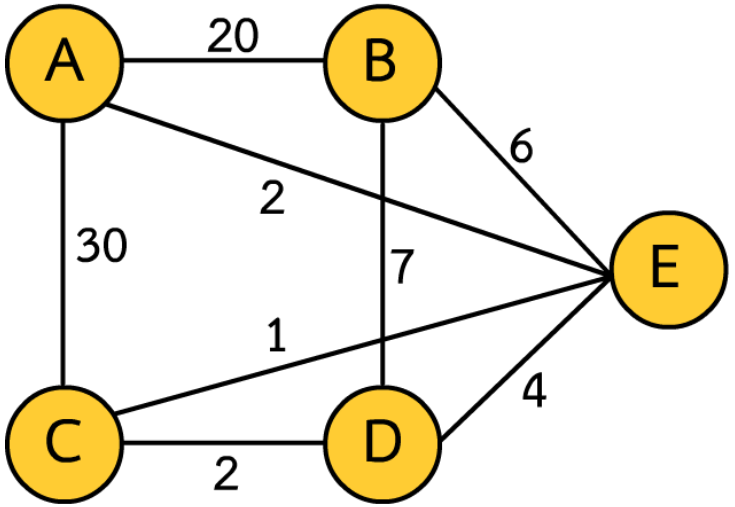
n order visited = graph traversal

- inorder
- postorder
- preorder

nearest neighbor graph

- find nearest node (in first order)

- หา node ที่อยู่ใกล้ที่สุด (ใน first order)



Who is nearest node of ‘B’ ?

ใครอยู่ใกล้ B ที่สุด ?

Code

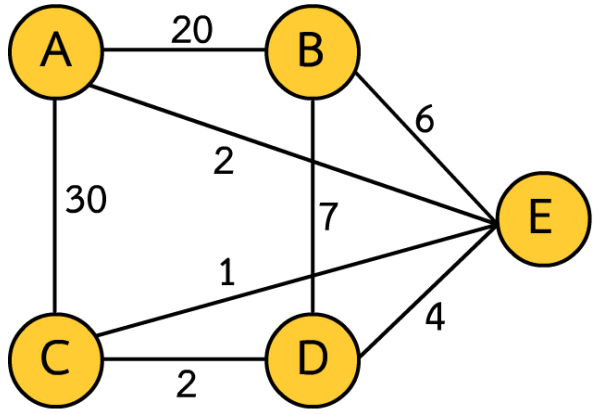
```
Node* nearest_node(Node* n){
    int nearest_idx = 0;
    for(int i=0;i<n->edge_count;i++){
        if(n->edges[i]->cost < n->edges[nearest_idx]->cost){
            nearest_idx = i;
        }
    }
    return n->edges[nearest_idx]->node;
}
```

*Like find minimum number in set

Code

```
cout << "nearest node of A is " << nearest_node(&a)->data << endl;
cout << "nearest node of B is " << nearest_node(&b)->data << endl;
cout << "nearest node of C is " << nearest_node(&c)->data << endl;
cout << "nearest node of D is " << nearest_node(&d)->data << endl;
cout << "nearest node of E is " << nearest_node(&e)->data << endl;
```

```
Result :
nearest node of A is E
nearest node of B is E
nearest node of C is E
nearest node of D is C
nearest node of E is C
```



question

- what is the shortest path between node x and y with exact 3 move

- เส้นทางใหนที่สั้นที่สุดระหว่าง node x และ y หากกำหนดให้ต้องเดินทั้งหมด 3 ครั้ง

Brute force algorithm

- travel to the z order visited and collect minimum cost
- ท่อง node โดยใช้วิธี z order visited และเก็บวิธีที่ cost น้อยที่สุด

What if sixth order Really !!?

Easy question (if third order)

- Traversal in third order visited method and collect minimum cost visited node

- ใช้วิธี third order visited และเก็บครั้งที่ใช้ cost น้อยที่สุด

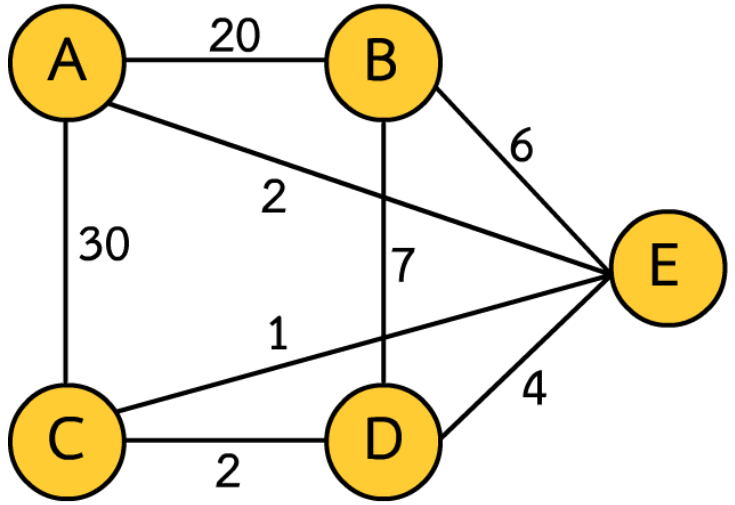
code

```
int third_shortest_path(Node* n, char target){
    int min_cost = -1;
    for(int i = 0;i<n->edge_count;i++){ // like print
        Node* node1 = (n->edges[i])->node;
        int cost1 = (n->edges[i])->cost;
        for(int j=0;j<node1->edge_count;j++){
            Node* node2 = (node1->edges[j])->node;
            int cost2 = cost1 + (node1->edges[j])->cost;
            for(int k=0;k< node2->edge_count;k++){
                Node* node3 = (node2->edges[k])->node;
                int cost3 = cost2 + (node2->edges[k])->cost;
                if(node3->data == target){
                    if(min_cost == -1 || cost3 < min_cost){
                        min_cost = cost3;
                    }
                }
            }
        }
    }
    return min_cost;
}
```

code

```
cout << "Shortest from A to E is cost " << third_shortest_path(&a, 'E') << endl;
cout << "Shortest from A to C is cost " << third_shortest_path(&a, 'C') << endl;
cout << "Shortest from B to C is cost " << third_shortest_path(&b, 'C') << endl;
cout << "Shortest from A to B is cost " << third_shortest_path(&a, 'B') << endl;
```

```
Shortest from A to E is cost 4
Shortest from A to C is cost 8
Shortest from B to C is cost 12
Shortest from A to B is cost 13
```



What is big theta for third shortest path

- n^3 !!??

Challenge question

- what is the shortest path between node x and y in z move

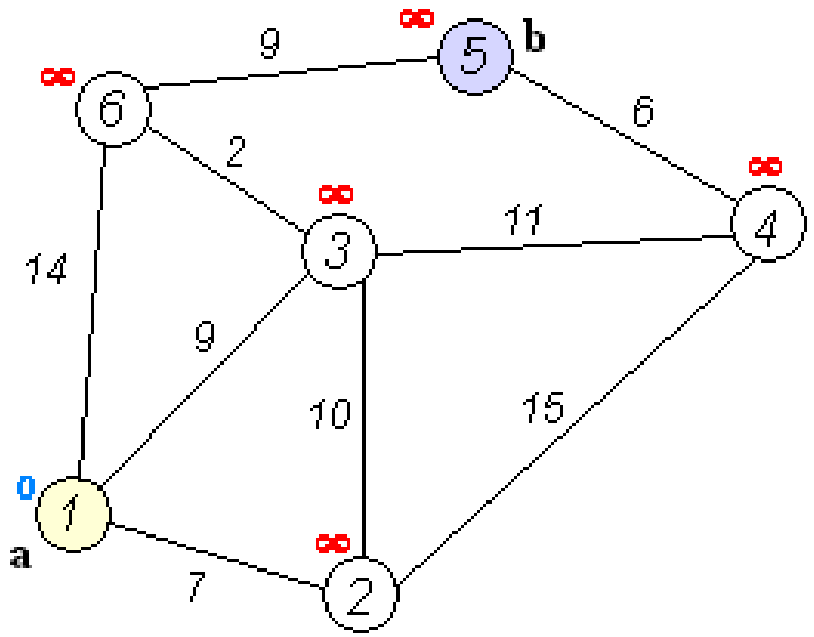
- เส้นทางใหนที่สั้นที่สุดระหว่าง node x และ y หากกำหนดให้เดินได้ z ครั้ง

What is big theta for n path

- n^n !!??

Dijkstra's algorithm

O(n log n) where n is the number of veritces



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Conclude

- Component of graph

- fundamental graph algorithm