Graph

LI Hao 李颢, Assoc. Prof. SPEIT & Dept. Automation of SEIEE

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

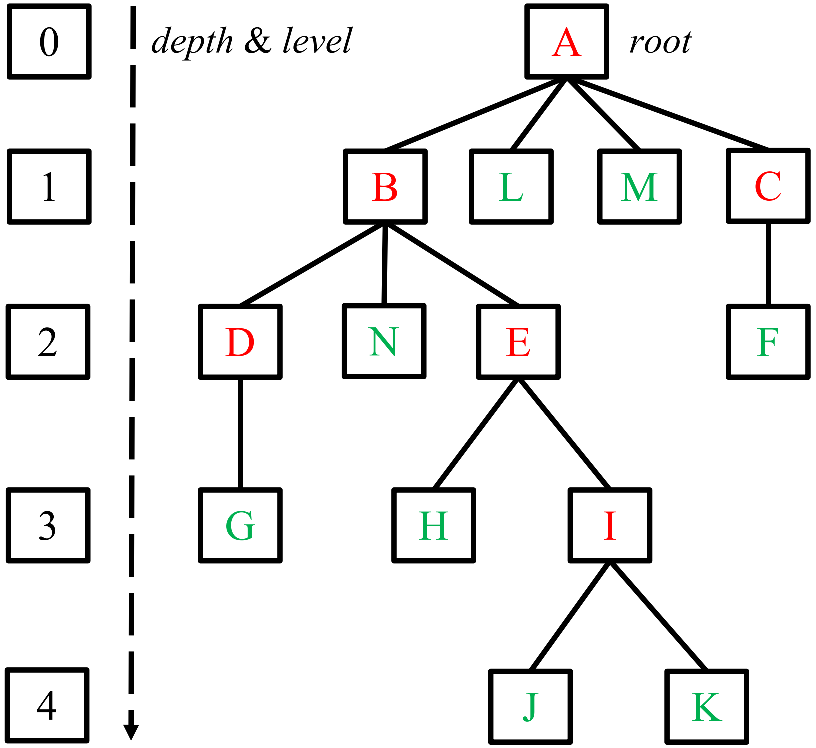
Graph

- **Applications**
  - model connectivity in computer networks & connected agents
  - represent a geographical map
  - model flow capacities in transportation networks
  - find an abstract path (i.e. a sequence of operations) from a starting condition to an objective condition
  - find an acceptable order for finishing tasks in a complex activity
  - model interactive relationships among components of control systems
  - model interactive relationships among social entities

Graph

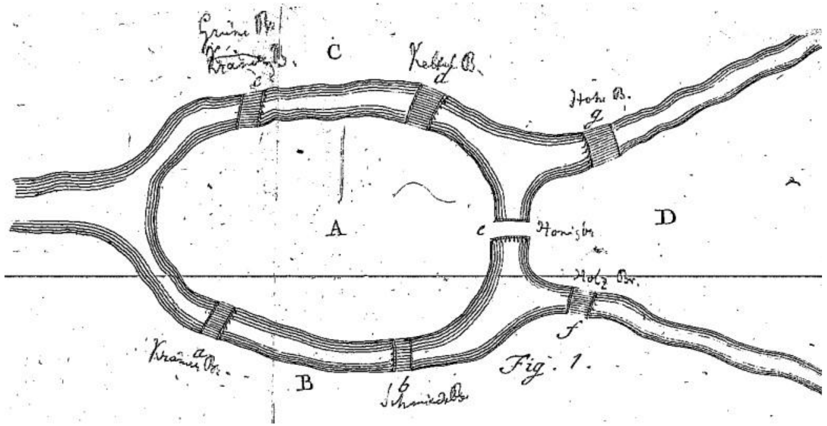
REVIEW

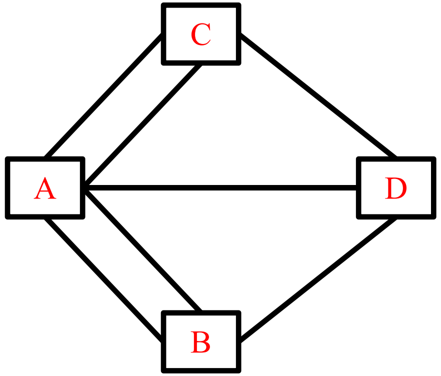
- **General tree - non-cyclic graph**
  - *node & edge*
  - *root*
  - arbitrary number of *subtrees*
  - *parent* & *children* (not limited to 2)
  - *out degree* - number of children
  - *left-most child* - arranged from left to right
  - *ancestor & descendant*
  - *path & length*
  - *depth* (cardinal) & *level* (ordinal)
  - *height* (largest depth+1)
  - *leaf node* & *internal node*



Graph

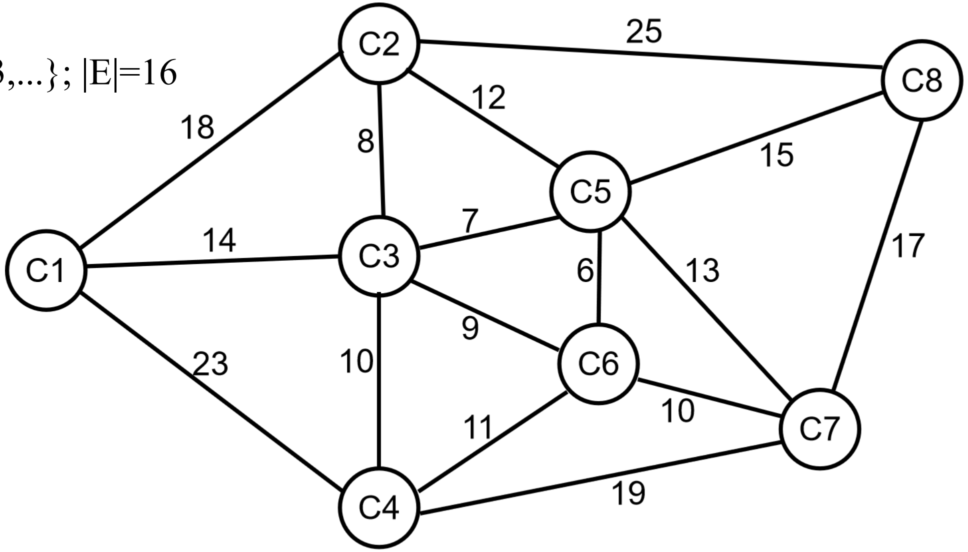
- **History**
  - *Leonhard **EULER***
    - "Solutio problematis ad geometriam situs pertinentis", 1736
      - says literally "{démêlement} (du) problème {à/pour} (la) geométrie (d'un) {site} {atteignant}"
      - means "solution of the problem (with regard) to the geometry of a connecting/connected region"
  - debut of *graph theory*
    - *seven bridges of Konigsberg* (Kaliningrad of Russia, hometown of *Emmanuel **Kant***)





Graph

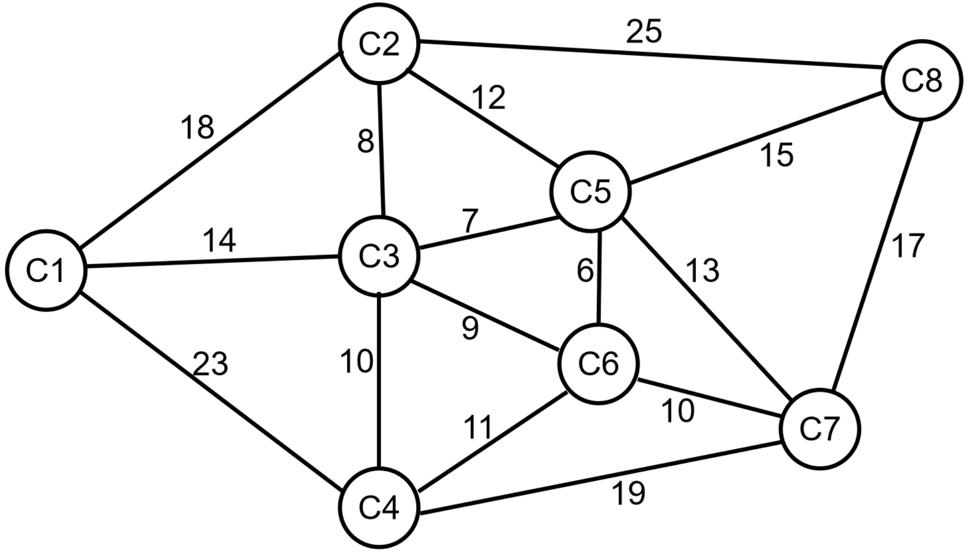
- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
    - V={C1,C2,C3,C4,C5,C6,C7,C8}; |V|=8
  - *edge* E (number of edges |E|)
    - E={C1C2,C1C3,C1C4,C2C3,...}; |E|=16
  - *in degree & out degree*
    - e.g. i.d.(C3)=5; o.d.(C8)=3
  - *edge weight*
    - e.g. |C3C5|=7
  - *path & length*
    - e.g. |C1C3C5C8|=36



Graph

	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	18	14	23	0	0	0	0
C2	18	0	8	0	12	0	0	25
C3	14	8	0	10	7	9	0	0
C4	23	0	10	0	0	11	19	0
C5	0	12	7	0	0	6	13	15
C6	0	0	9	11	6	0	10	0
C7	0	0	0	19	13	10	0	17
C8	0	25	0	0	15	0	17	0

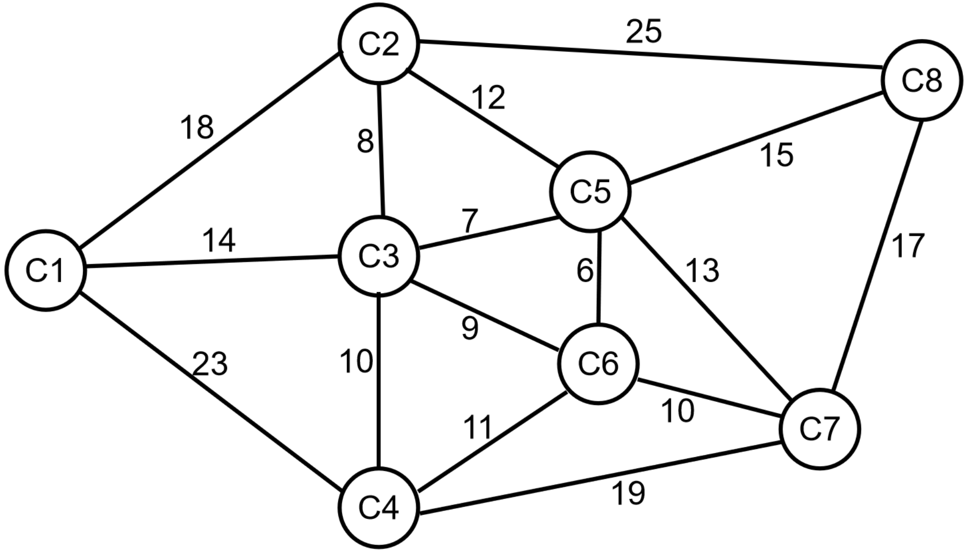
- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
  - *edge* E (number of edges |E|)
  - *in degree & out degree*
  - *edge weight*
  - *path & length*
  - *adjacent & neighbour*
    - **adjacency matrix**
    - adjacency list



	C1	C2	C3	C4	C5	C6	C7	C8
C1	0	18	14	23	0	0	0	0
C2	18	0	8	0	12	0	0	25
C3	14	8	0	10	7	9	0	0
C4	23	0	10	0	0	11	19	0
C5	0	12	7	0	0	6	13	15
C6	0	0	9	11	6	0	10	0
C7	0	0	0	19	13	10	0	17
C8	0	25	0	0	15	0	17	0

Graph

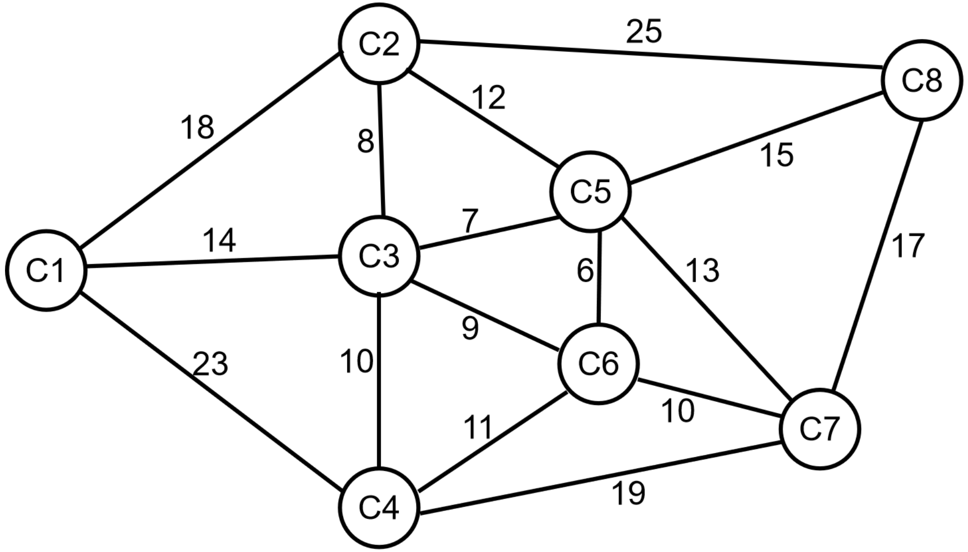
- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
  - *edge* E (number of edges |E|)
  - *in degree & out degree*
  - *edge weight*
  - *path & length*
  - *adjacent & neighbour*
    - **adjacency matrix**
      - representation cost: O(|V|^2)
    - adjacency list



Graph

- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
  - *edge* E (number of edges |E|)
  - *in degree & out degree*
  - *edge weight*
  - *path & length*
  - *adjacent & neighbour*
    - adjacency matrix
      - representation cost: O(|V|^2)
    - **adjacency list**

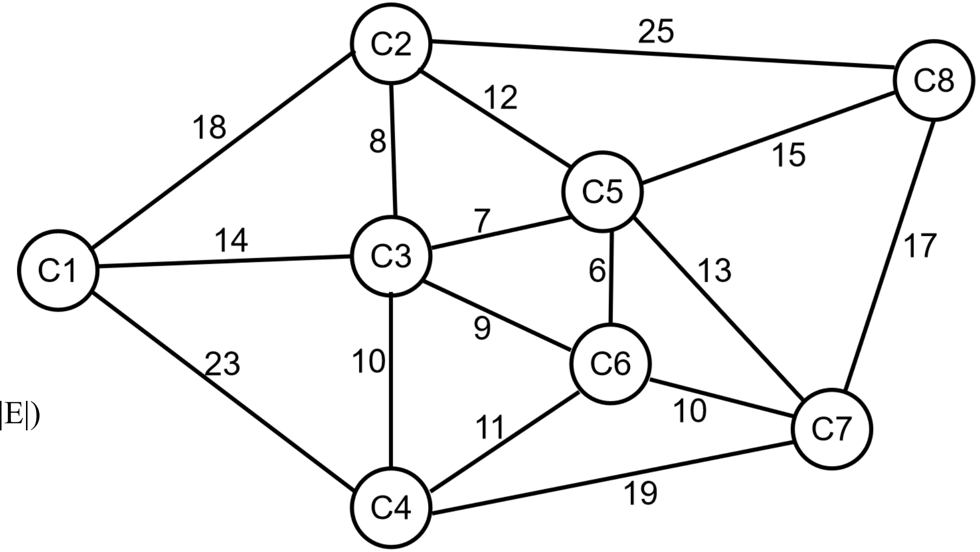
C1	C2	18	C3	14	C4	23				
C2	C1	18	C3	8	C5	12	C8	25		
C3	C1	14	C2	8	C4	10	C5	7	C6	9
C4	C1	23	C3	10	C6	11	C7	19		
C5	C2	12	C3	7	C6	6	C7	13	C8	15
C6	C3	9	C4	11	C5	6	C7	10		
C7	C4	19	C5	13	C6	10	C8	17		
C8	C2	25	C5	15	C7	17				



Graph

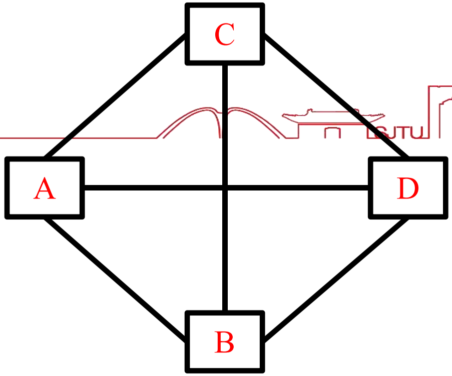
- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
  - *edge* E (number of edges |E|)
  - *in degree & out degree*
  - *edge weight*
  - *path & length*
  - *adjacent & neighbour*
    - adjacency matrix
      - representation cost: O(|V|^2)
    - **adjacency list**
      - representation cost: O(|V|+|E|)

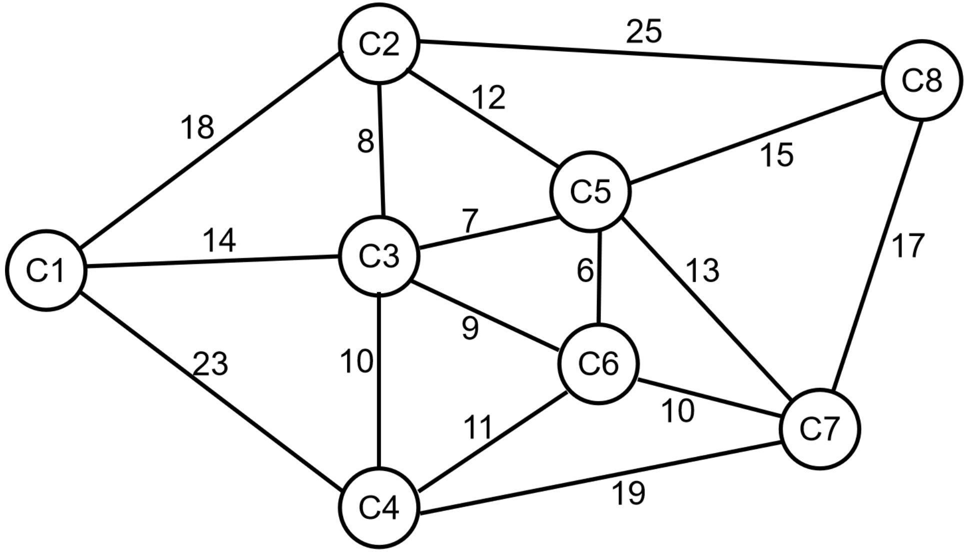
C1	C2	18	C3	14	C4	23				
C2	C1	18	C3	8	C5	12	C8	25		
C3	C1	14	C2	8	C4	10	C5	7	C6	9
C4	C1	23	C3	10	C6	11	C7	19		
C5	C2	12	C3	7	C6	6	C7	13	C8	15
C6	C3	9	C4	11	C5	6	C7	10		
C7	C4	19	C5	13	C6	10	C8	17		
C8	C2	25	C5	15	C7	17				



Graph

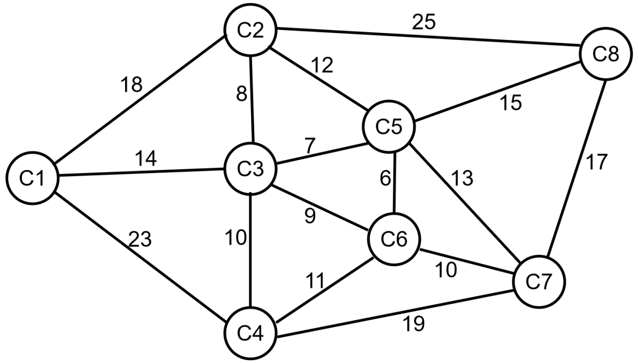
- **General graph**
  - *undirected & directed graph* G=(V,E)
  - *vertex* V (number of vertices |V|)
  - *edge* E (number of edges |E|)
  - *in degree & out degree*
  - *edge weight*
  - *path & length*
  - *adjacent & neighbour*
  - *complete*
  - *dense* e.g. $|E| \sim O(|V|^2)$
    - adjency matrix representation
  - *sparse* e.g. $|E| \sim O(|V|)$
    - adjency list representation

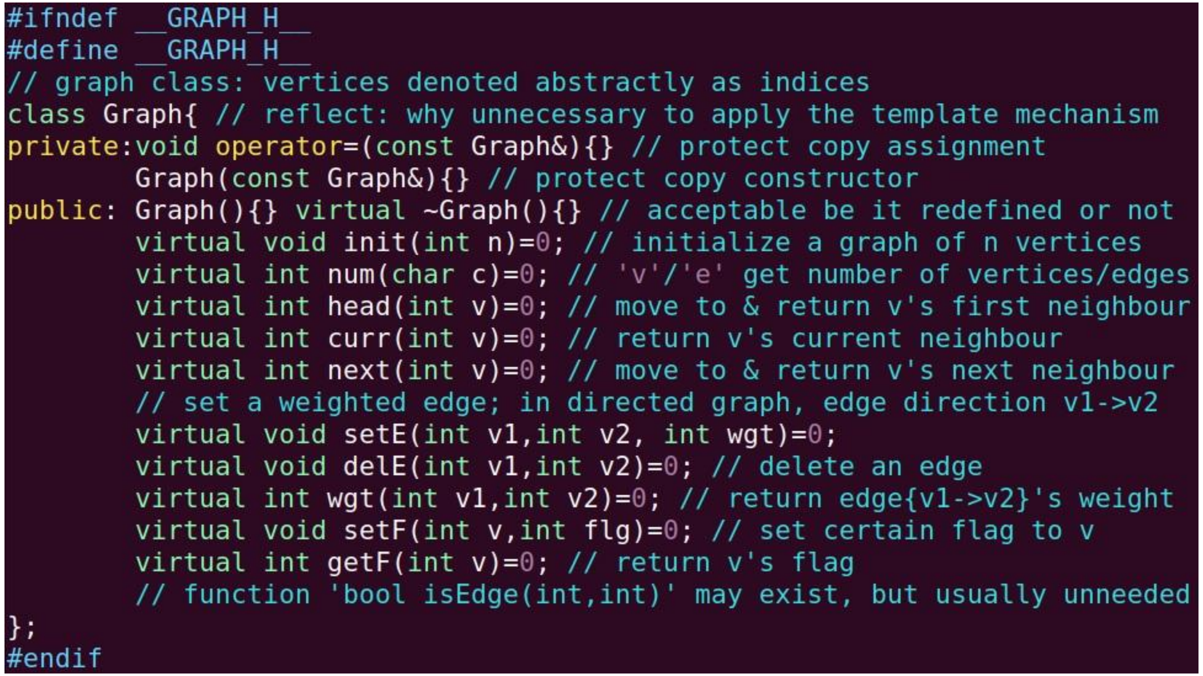




Graph

- **General graph**
  - *undirected & directed graph* G=(V,E)

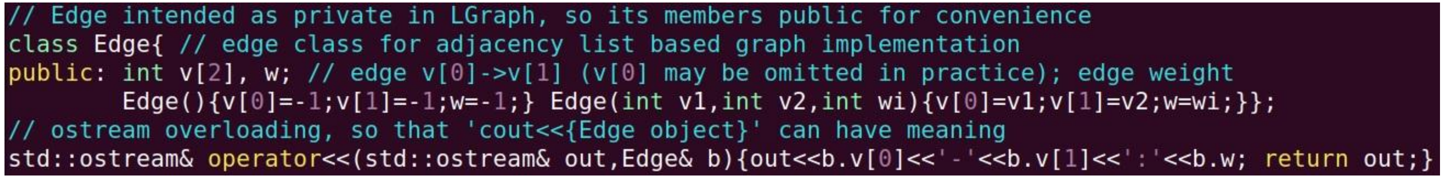




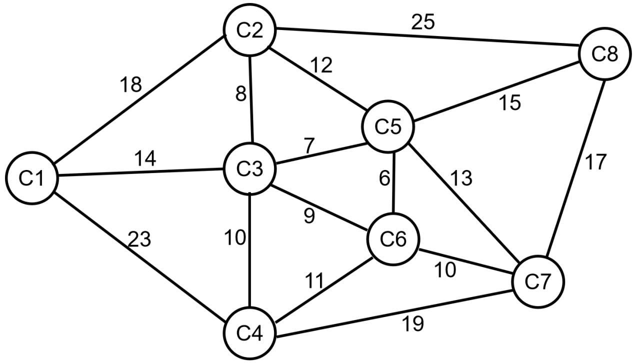
```
#ifndef __GRAPH_H__
#define __GRAPH_H__
// graph class: vertices denoted abstractly as indices
class Graph{ // reflect: why unnecessary to apply the template mechanism
private:void operator=(const Graph&){} // protect copy assignment
        Graph(const Graph&){} // protect copy constructor
public: Graph(){} virtual ~Graph(){} // acceptable be it redefined or not
        virtual void init(int n)=0; // initialize a graph of n vertices
        virtual int num(char c)=0; // 'v'/'e' get number of vertices/edges
        virtual int head(int v)=0; // move to & return v's first neighbour
        virtual int curr(int v)=0; // return v's current neighbour
        virtual int next(int v)=0; // move to & return v's next neighbour
        // set a weighted edge; in directed graph, edge direction v1->v2
        virtual void setE(int v1,int v2, int wgt)=0;
        virtual void delE(int v1,int v2)=0; // delete an edge
        virtual int wgt(int v1,int v2)=0; // return edge{v1->v2}'s weight
        virtual void setF(int v,int flg)=0; // set certain flag to v
        virtual int getF(int v)=0; // return v's flag
        // function 'bool isEdge(int,int)' may exist, but usually unneeded
};
#endif
```

Graph

- **General graph**
  - *undirected & directed graph* G=(V,E)



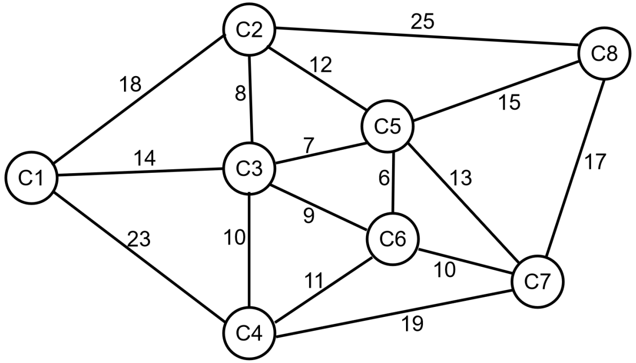
```
// Edge intended as private in LGraph, so its members public for convenience
class Edge{ // edge class for adjacency list based graph implementation
public: int v[2], w; // edge v[0]->v[1] (v[0] may be omitted in practice); edge weight
        Edge(){v[0]=-1;v[1]=-1;w=-1;} Edge(int v1,int v2,int wi){v[0]=v1;v[1]=v2;w=wi;}};
// ostream overloading, so that 'cout<<{Edge object}' can have meaning
std::ostream& operator<<(std::ostream& out,Edge& b){out<<b.v[0]<<'-'<<b.v[1]<<':'<<b.w; return out;}
```

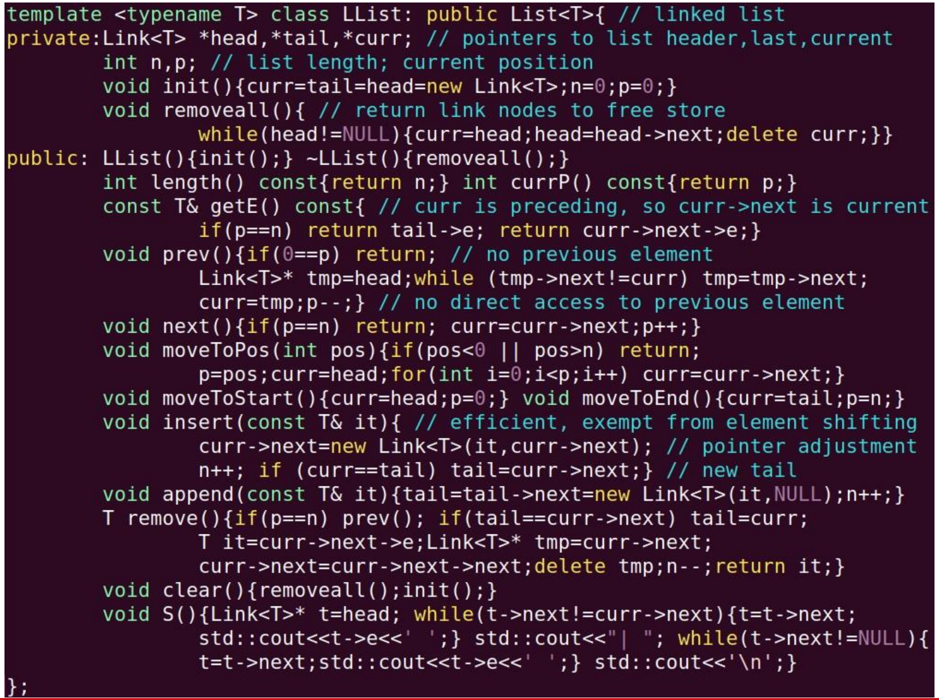


Graph

- **General graph**
  - *undirected & directed graph* G=(V,E)

LList2: add an internal variable *p* to trace current element position

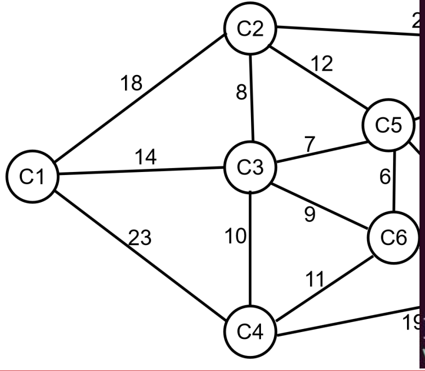


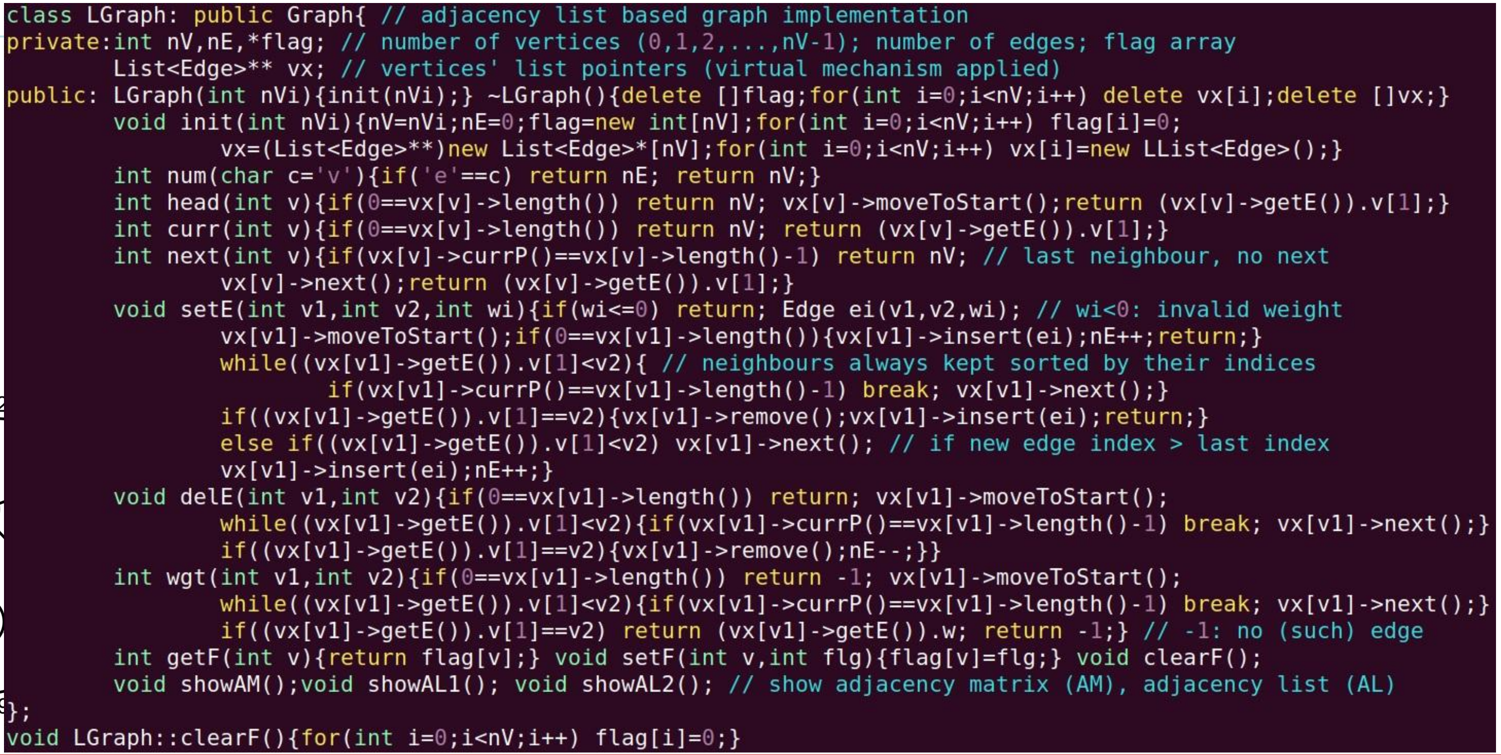


```
template <typename T> class LList: public List<T>{ // linked list
private:Link<T> *head,*tail,*curr; // pointers to list header,last,current
        int n,p; // list length; current position
        void init(){curr=tail=head=new Link<T>;n=0;p=0;}
        void removeall(){ // return link nodes to free store
                while(head!=NULL){curr=head;head=head->next;delete curr;}}
public: LList(){init();} ~LList(){removeall();}
        int length() const{return n;} int currP() const{return p;}
        const T& getE() const{ // curr is preceding, so curr->next is current
                if(p==n) return tail->e; return curr->next->e;}
        void prev(){if(0==p) return; // no previous element
                Link<T>* tmp=head;while (tmp->next!=curr) tmp=tmp->next;
                curr=tmp;p--;} // no direct access to previous element
        void next(){if(p==n) return; curr=curr->next;p++;}
        void moveToPos(int pos){if(pos<0 || pos>n) return;
                p=pos;curr=head;for(int i=0;i<p;i++) curr=curr->next;}
        void moveToStart(){curr=head;p=0;} void moveToEnd(){curr=tail;p=n;}
        void insert(const T& it){ // efficient, exempt from element shifting
                curr->next=new Link<T>(it,curr->next); // pointer adjustment
                n++; if (curr==tail) tail=curr->next;} // new tail
        void append(const T& it){tail=tail->next=new Link<T>(it,NULL);n++;}
        T remove(){if(p==n) prev(); if(tail==curr->next) tail=curr;
                T it=curr->next->e;Link<T>* tmp=curr->next;
                curr->next=curr->next->next;delete tmp;n--;return it;}
        void clear(){removeall();init();}
        void S(){Link<T>* t=head; while(t->next!=curr->next){t=t->next;
                std::cout<<t->e<<' ';} std::cout<<"| "; while(t->next!=NULL){
                t=t->next;std::cout<<t->e<<' ';} std::cout<<'\n';}
};
```

Graph

- **General graph**

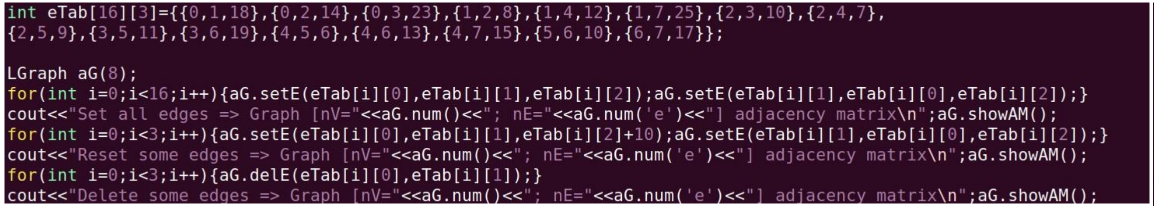




```
class LGraph: public Graph{ // adjacency list based graph implementation
private:int nV,nE,*flag; // number of vertices (0,1,2,...,nV-1); number of edges; flag array
        List<Edge>** vx; // vertices' list pointers (virtual mechanism applied)
public: LGraph(int nVi){init(nVi);} ~LGraph(){delete []flag;for(int i=0;i<nV;i++) delete vx[i];delete []vx;}
        void init(int nVi){nV=nVi;nE=0;flag=new int[nV];for(int i=0;i<nV;i++) flag[i]=0;
                vx=(List<Edge>**)new List<Edge>*[nV];for(int i=0;i<nV;i++) vx[i]=new LList<Edge>();}
        int num(char c='v'){if('e'==c) return nE; return nV;}
        int head(int v){if(0==vx[v]->length()) return nV; vx[v]->moveToStart();return (vx[v]->getE()).v[1];}
        int curr(int v){if(0==vx[v]->length()) return nV; return (vx[v]->getE()).v[1];}
        int next(int v){if(vx[v]->currP()==vx[v]->length()-1) return nV; // last neighbour, no next
                vx[v]->next();return (vx[v]->getE()).v[1];}
        void setE(int v1,int v2,int wi){if(wi<=0) return; Edge ei(v1,v2,wi); // wi<0: invalid weight
                vx[v1]->moveToStart();if(0==vx[v1]->length()){vx[v1]->insert(ei);nE++;return;}
                while((vx[v1]->getE()).v[1]<v2){ // neighbours always kept sorted by their indices
                        if(vx[v1]->currP()==vx[v1]->length()-1) break; vx[v1]->next();}
                if((vx[v1]->getE()).v[1]==v2){vx[v1]->remove();vx[v1]->insert(ei);return;}
                else if((vx[v1]->getE()).v[1]<v2) vx[v1]->next(); // if new edge index > last index
                vx[v1]->insert(ei);nE++;}
        void delE(int v1,int v2){if(0==vx[v1]->length()) return; vx[v1]->moveToStart();
                while((vx[v1]->getE()).v[1]<v2){if(vx[v1]->currP()==vx[v1]->length()-1) break; vx[v1]->next();}
                if((vx[v1]->getE()).v[1]==v2){vx[v1]->remove();nE--;}}
        int wgt(int v1,int v2){if(0==vx[v1]->length()) return -1; vx[v1]->moveToStart();
                while((vx[v1]->getE()).v[1]<v2){if(vx[v1]->currP()==vx[v1]->length()-1) break; vx[v1]->next();}
                if((vx[v1]->getE()).v[1]==v2) return (vx[v1]->getE()).w; return -1;} // -1: no (such) edge
        int getF(int v){return flag[v];} void setF(int v,int flg){flag[v]=flg;} void clearF();
        void showAM();void showAL1(); void showAL2(); // show adjacency matrix (AM), adjacency list (AL)
};
void LGraph::clearF(){for(int i=0;i<nV;i++) flag[i]=0;}
```

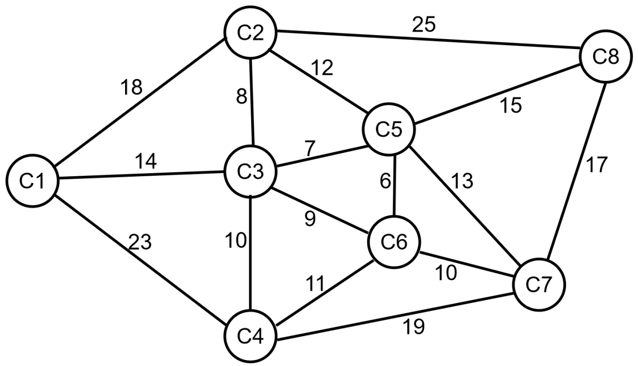
Graph

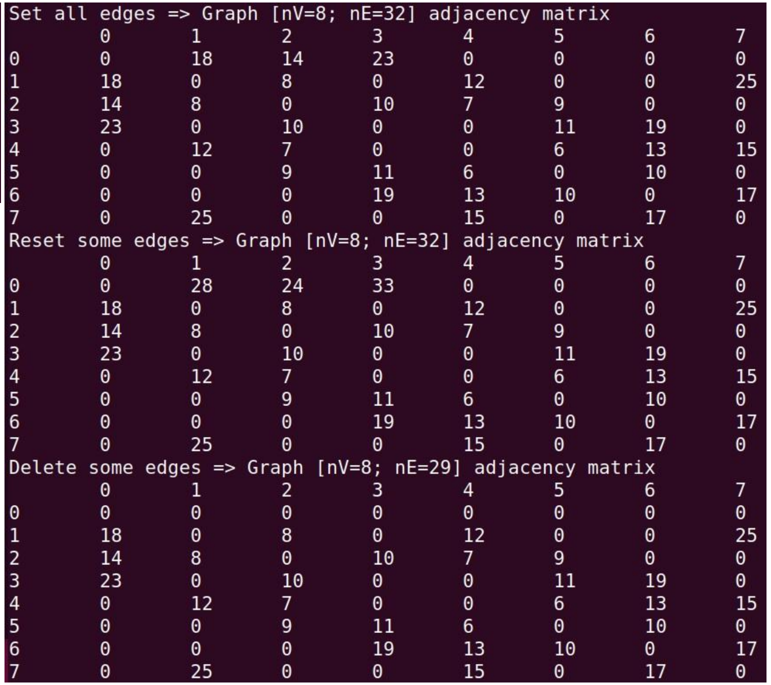
- **General graph**
  - *undirected & directed graph* G=(V,E)



```
int eTab[16][3]={{0,1,18},{0,2,14},{0,3,23},{1,2,8},{1,4,12},{1,7,25},{2,3,10},{2,4,7},
{2,5,9},{3,5,11},{3,6,19},{4,5,6},{4,6,13},{4,7,15},{5,6,10},{6,7,17}};

LGraph aG(8);
for(int i=0;i<16;i++){aG.setE(eTab[i][0],eTab[i][1],eTab[i][2]);aG.setE(eTab[i][1],eTab[i][0],eTab[i][2]);}
cout<<"Set all edges => Graph [nV="<<aG.num()<<"; nE="<<aG.num('e')<<"] adjacency matrix\n";aG.showAM();
for(int i=0;i<3;i++){aG.setE(eTab[i][0],eTab[i][1],eTab[i][2]+10);aG.setE(eTab[i][1],eTab[i][0],eTab[i][2]);}
cout<<"Reset some edges => Graph [nV="<<aG.num()<<"; nE="<<aG.num('e')<<"] adjacency matrix\n";aG.showAM();
for(int i=0;i<3;i++){aG.delE(eTab[i][0],eTab[i][1]);}
cout<<"Delete some edges => Graph [nV="<<aG.num()<<"; nE="<<aG.num('e')<<"] adjacency matrix\n";aG.showAM();
```

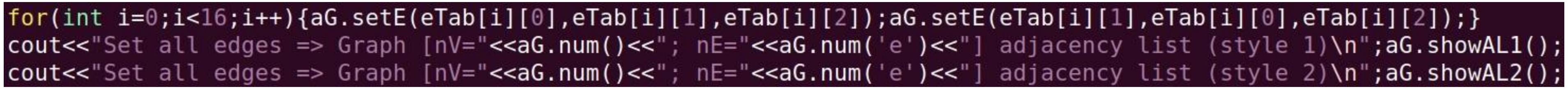




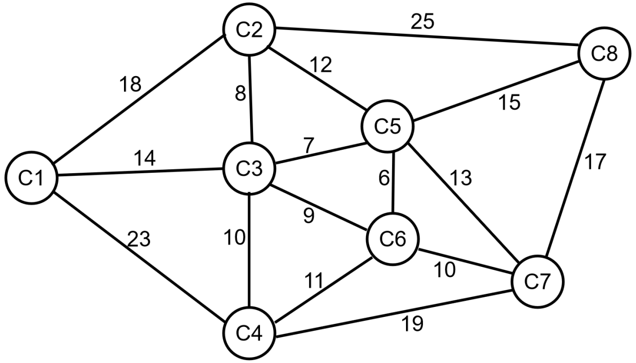
```
Set all edges => Graph [nV=8; nE=32] adjacency matrix
        0       1       2       3       4       5       6       7
0       0       18      14      23      0       0       0       0
1       18      0       8       0       12      0       0       25
2       14      8       0       10      7       9       0       0
3       23      0       10      0       0       11      19      0
4       0       12      7       0       0       6       13      15
5       0       0       9       11      6       0       10      0
6       0       0       0       19      13      10      0       17
7       0       25      0       0       15      0       17      0
Reset some edges => Graph [nV=8; nE=32] adjacency matrix
        0       1       2       3       4       5       6       7
0       0       28      24      33      0       0       0       0
1       18      0       8       0       12      0       0       25
2       14      8       0       10      7       9       0       0
3       23      0       10      0       0       11      19      0
4       0       12      7       0       0       6       13      15
5       0       0       9       11      6       0       10      0
6       0       0       0       19      13      10      0       17
7       0       25      0       0       15      0       17      0
Delete some edges => Graph [nV=8; nE=29] adjacency matrix
        0       1       2       3       4       5       6       7
0       0       0       0       0       0       0       0       0
1       18      0       8       0       12      0       0       25
2       14      8       0       10      7       9       0       0
3       23      0       10      0       0       11      19      0
4       0       12      7       0       0       6       13      15
5       0       0       9       11      6       0       10      0
6       0       0       0       19      13      10      0       17
7       0       25      0       0       15      0       17      0
```

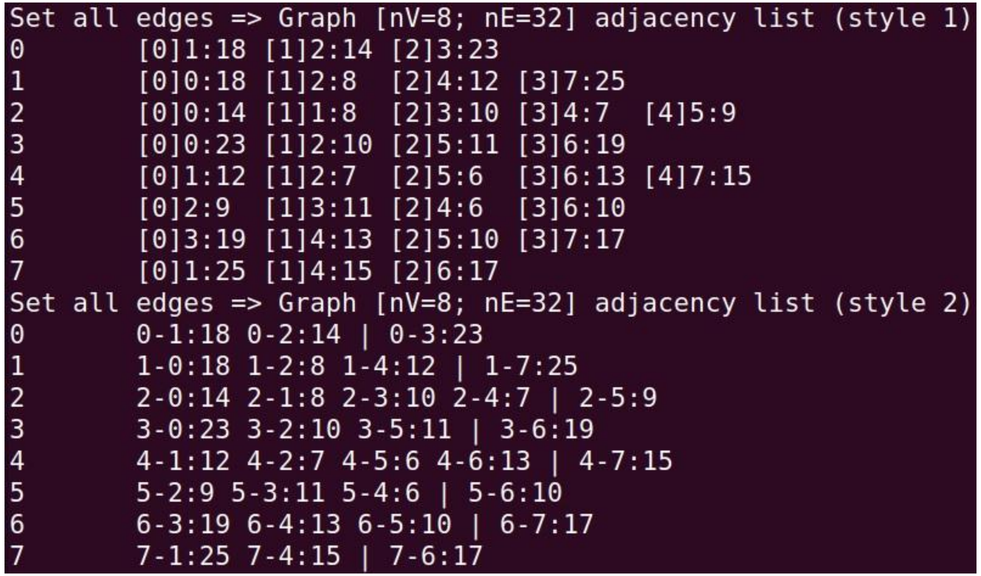
Graph

- **General graph**
  - *undirected & directed graph* G=(V,E)



```
for(int i=0;i<16;i++){aG.setE(eTab[i][0],eTab[i][1],eTab[i][2]);aG.setE(eTab[i][1],eTab[i][0],eTab[i][2]);}
cout<<"Set all edges => Graph [nV="<<aG.num()<<"; nE="<<aG.num('e')<<"] adjacency list (style 1)\n";aG.showAL1();
cout<<"Set all edges => Graph [nV="<<aG.num()<<"; nE="<<aG.num('e')<<"] adjacency list (style 2)\n";aG.showAL2();
```

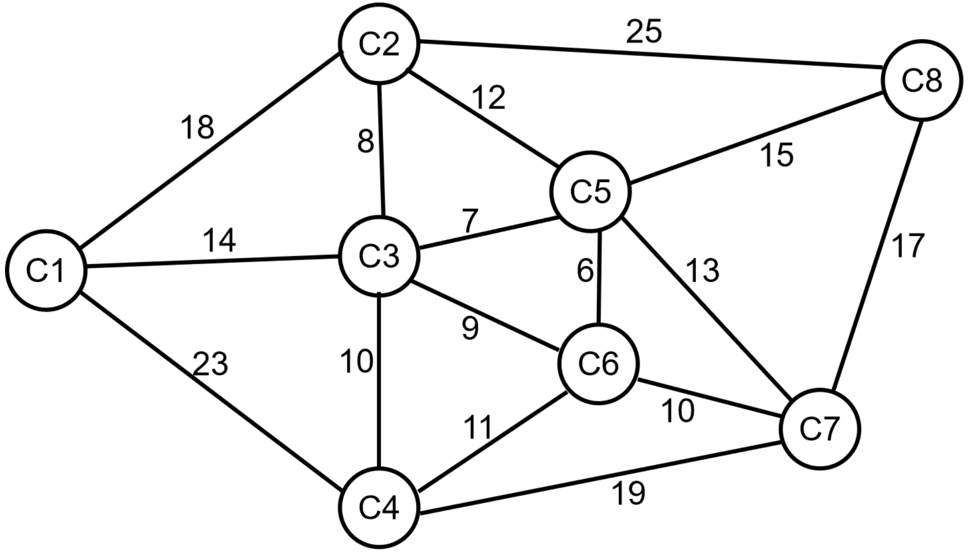




```
Set all edges => Graph [nV=8; nE=32] adjacency list (style 1)
0       [0]1:18 [1]2:14 [2]3:23
1       [0]0:18 [1]2:8  [2]4:12 [3]7:25
2       [0]0:14 [1]1:8  [2]3:10 [3]4:7  [4]5:9
3       [0]0:23 [1]2:10 [2]5:11 [3]6:19
4       [0]1:12 [1]2:7  [2]5:6  [3]6:13 [4]7:15
5       [0]2:9  [1]3:11 [2]4:6  [3]6:10
6       [0]3:19 [1]4:13 [2]5:10 [3]7:17
7       [0]1:25 [1]4:15 [2]6:17
Set all edges => Graph [nV=8; nE=32] adjacency list (style 2)
0       0-1:18 0-2:14 | 0-3:23
1       1-0:18 1-2:8 1-4:12 | 1-7:25
2       2-0:14 2-1:8 2-3:10 2-4:7 | 2-5:9
3       3-0:23 3-2:10 3-5:11 | 3-6:19
4       4-1:12 4-2:7 4-5:6 4-6:13 | 4-7:15
5       5-2:9 5-3:11 5-4:6 | 5-6:10
6       6-3:19 6-4:13 6-5:10 | 6-7:17
7       7-1:25 7-4:15 | 7-6:17
```

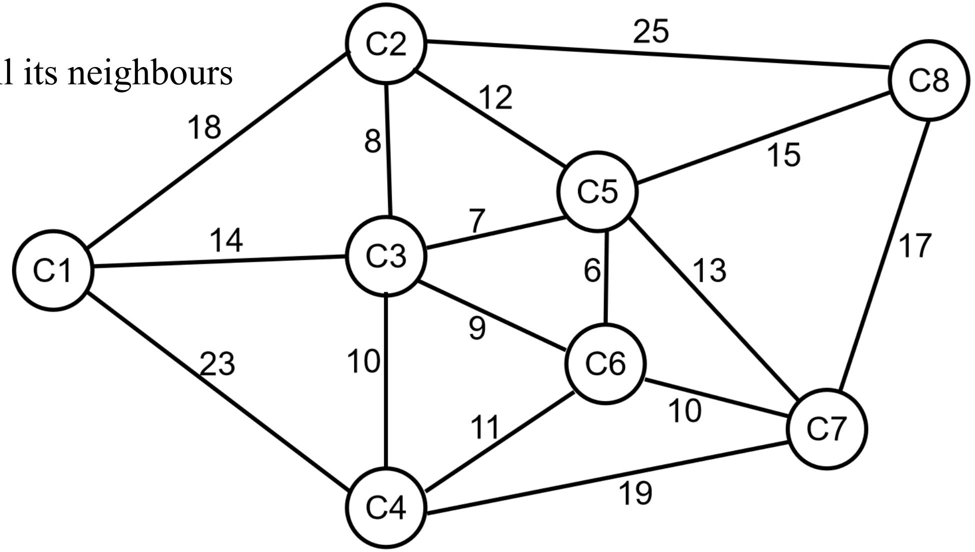
Graph

- **Graph traversal**
  - directed graph G=(V,E)
    - treat an undirected graph as a directed one
  - *depth first search* (DFS)
  - *breadth first search* (BFS)



Graph

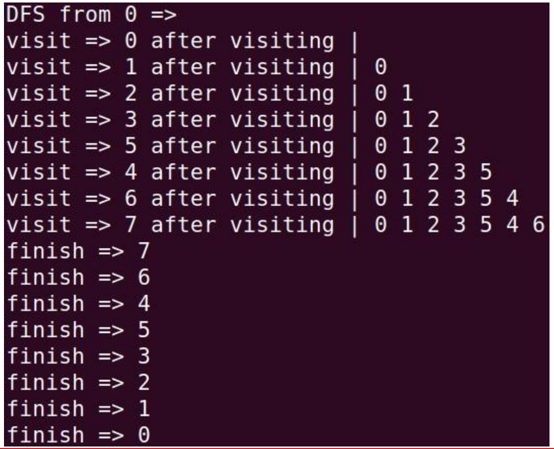
- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours



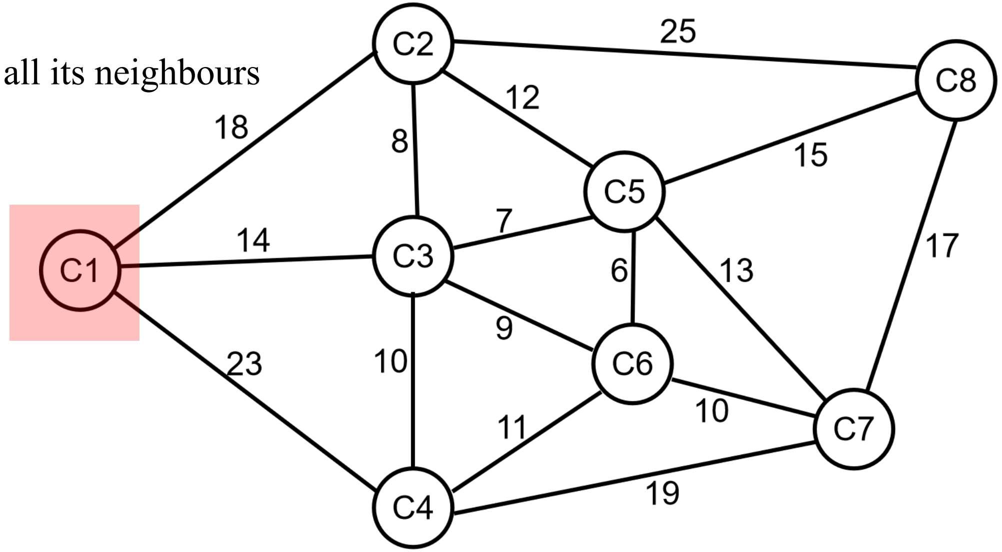
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1



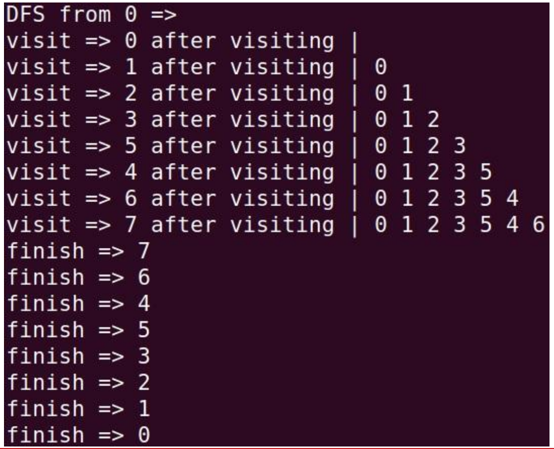
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



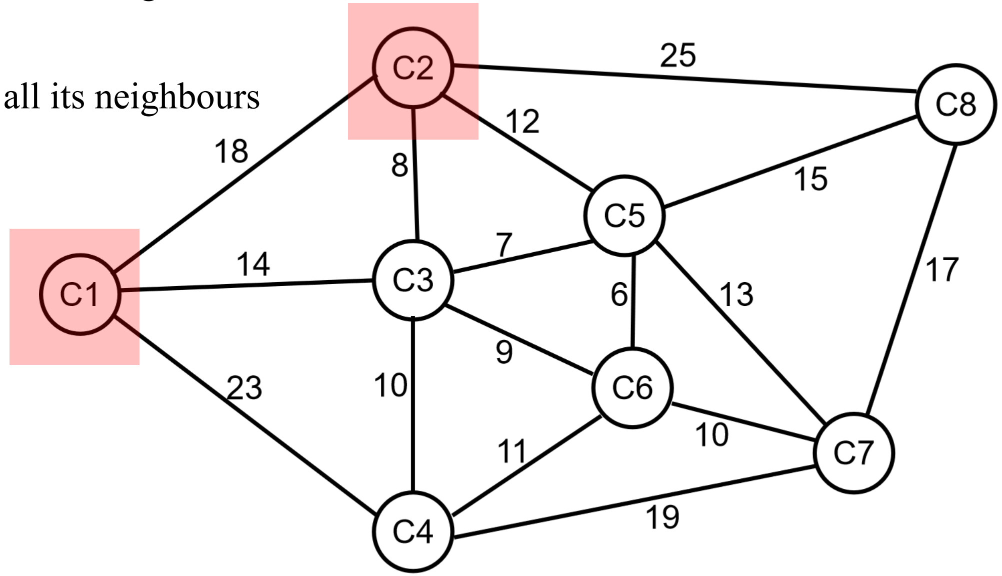
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2



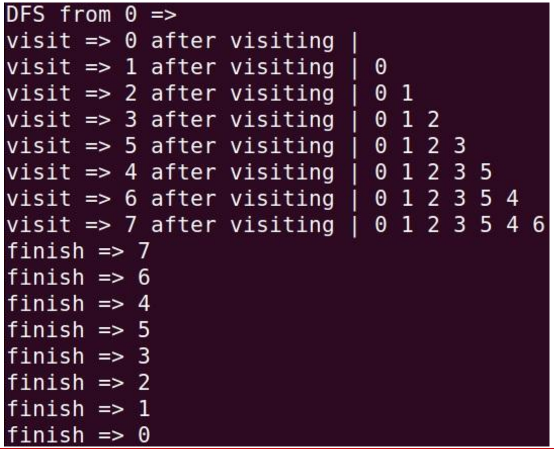
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



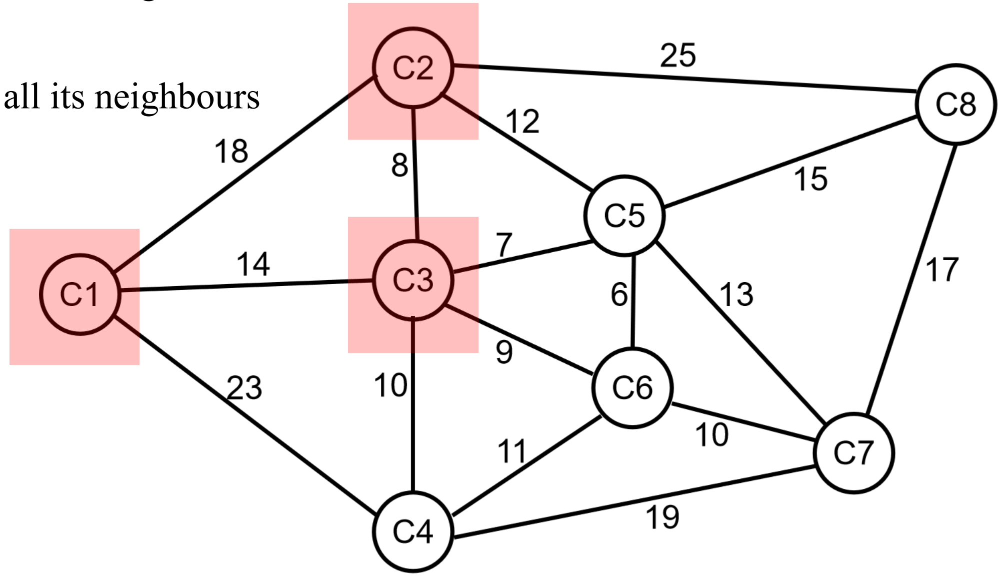
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3



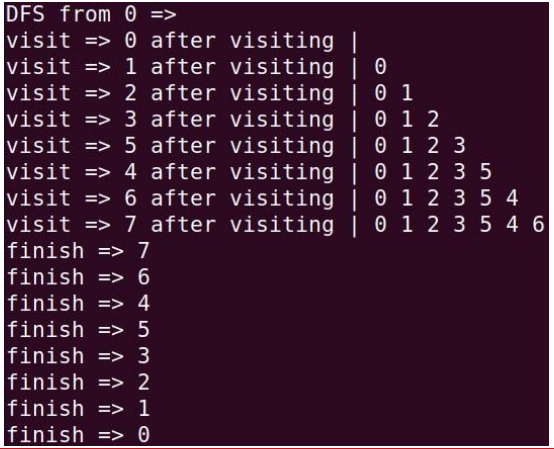
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



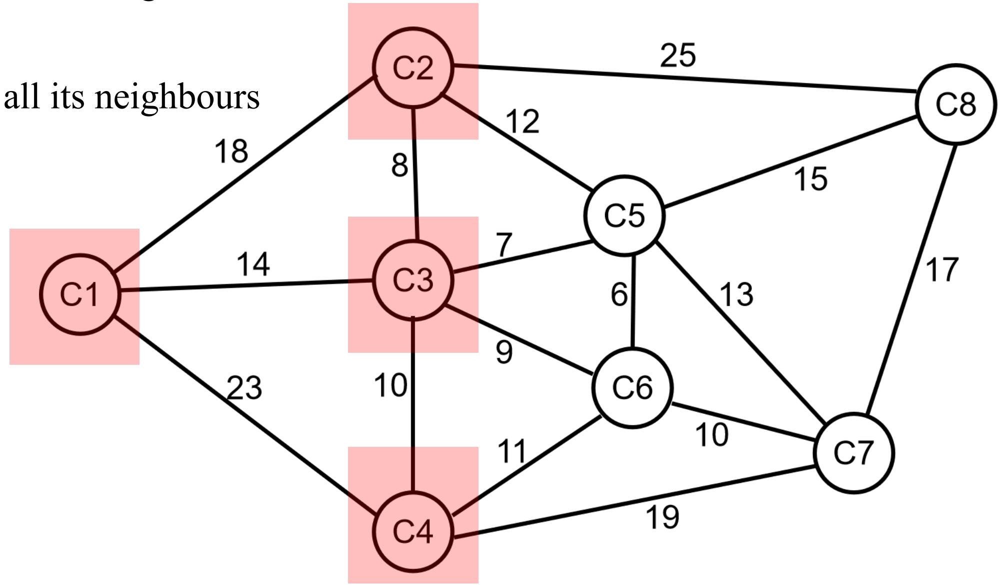
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4



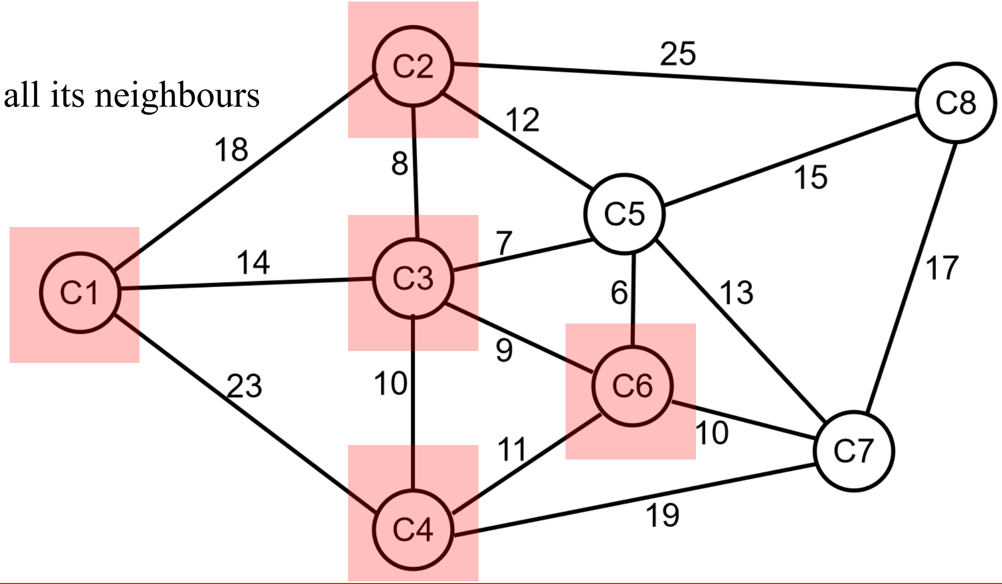
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```

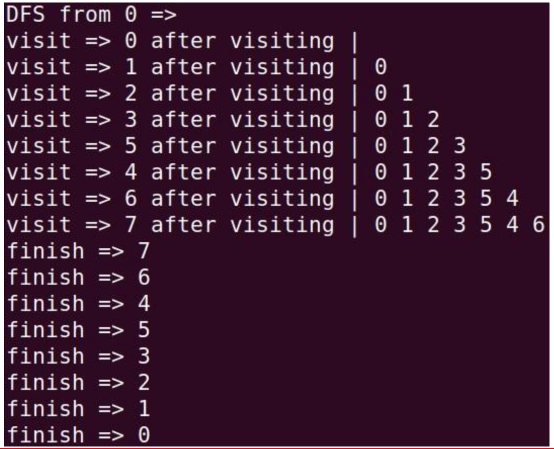


Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6



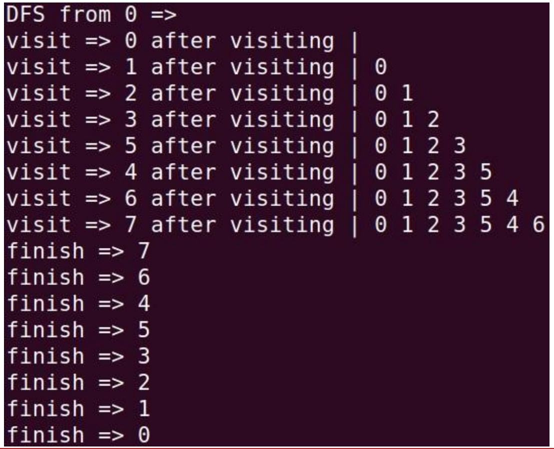


```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```

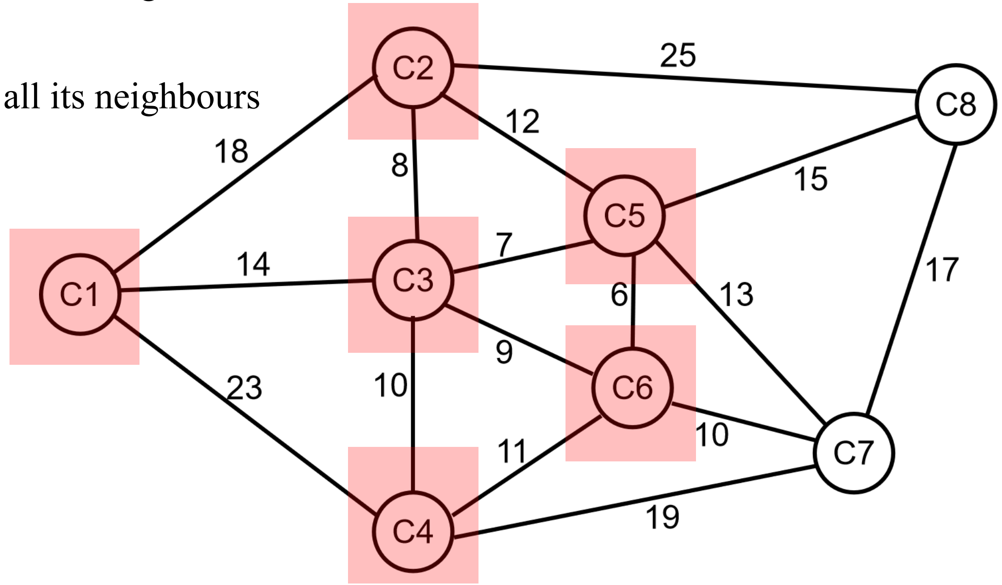
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6 C5



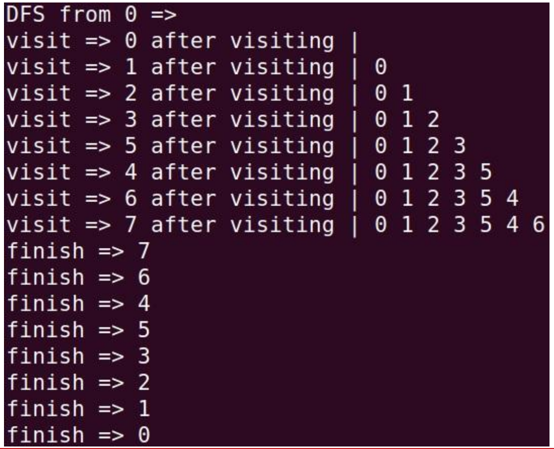
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



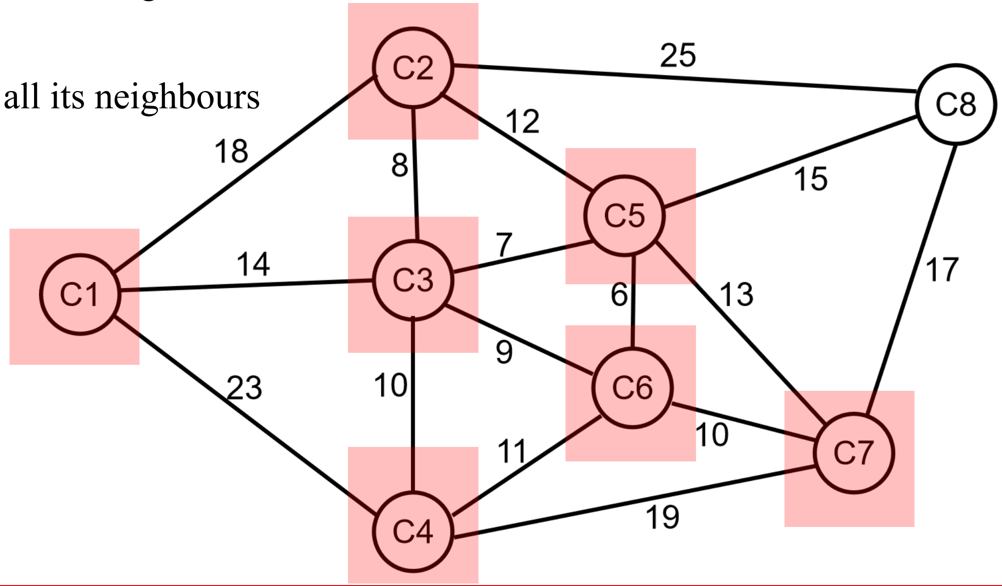
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6 C5 C7



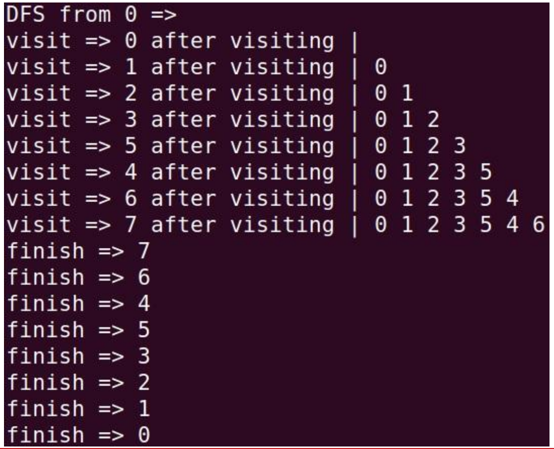
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



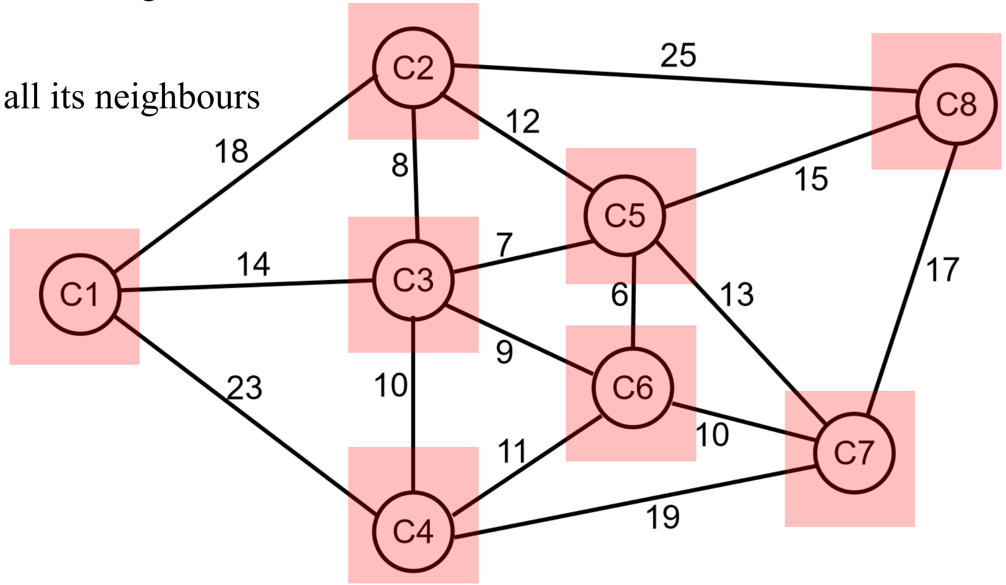
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6 C5 C7 C8



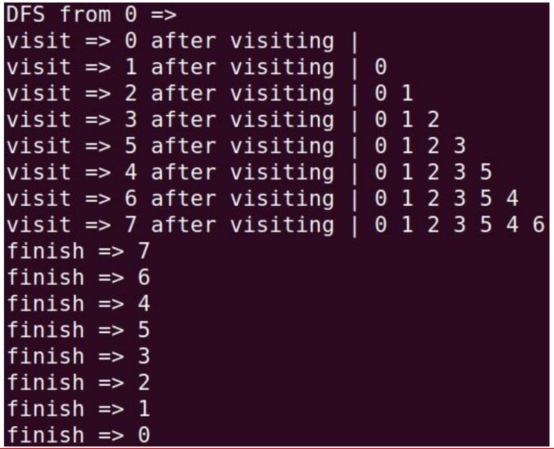
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



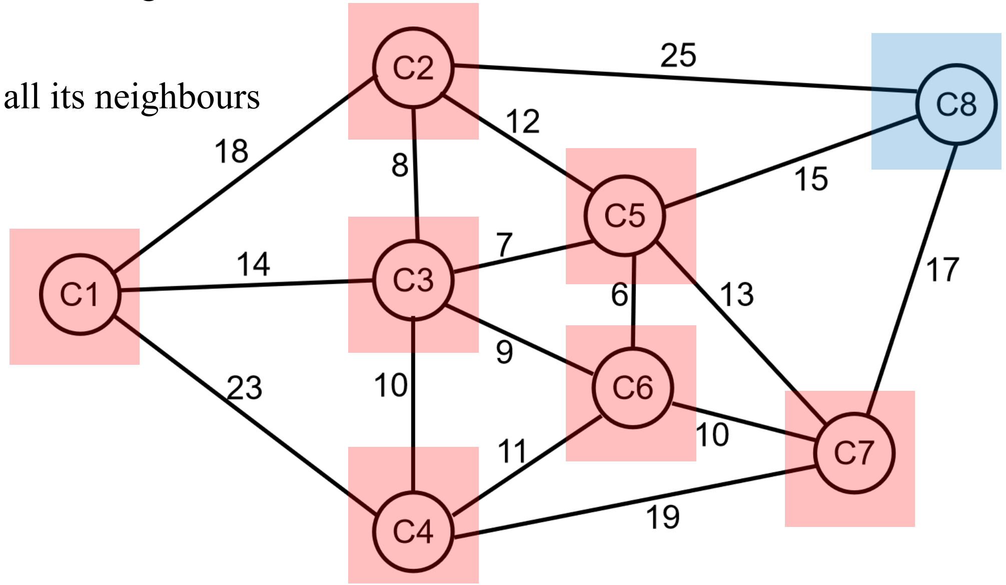
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6 C5 C7



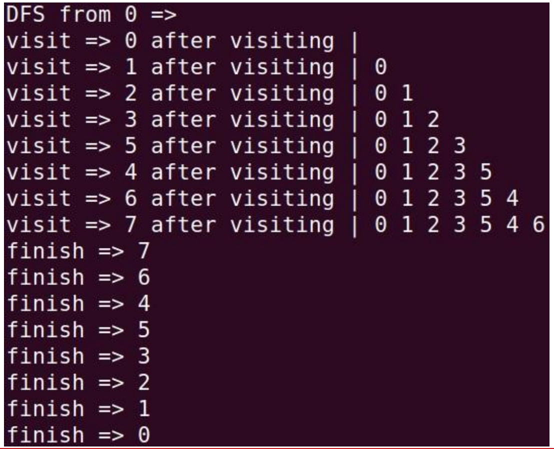
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



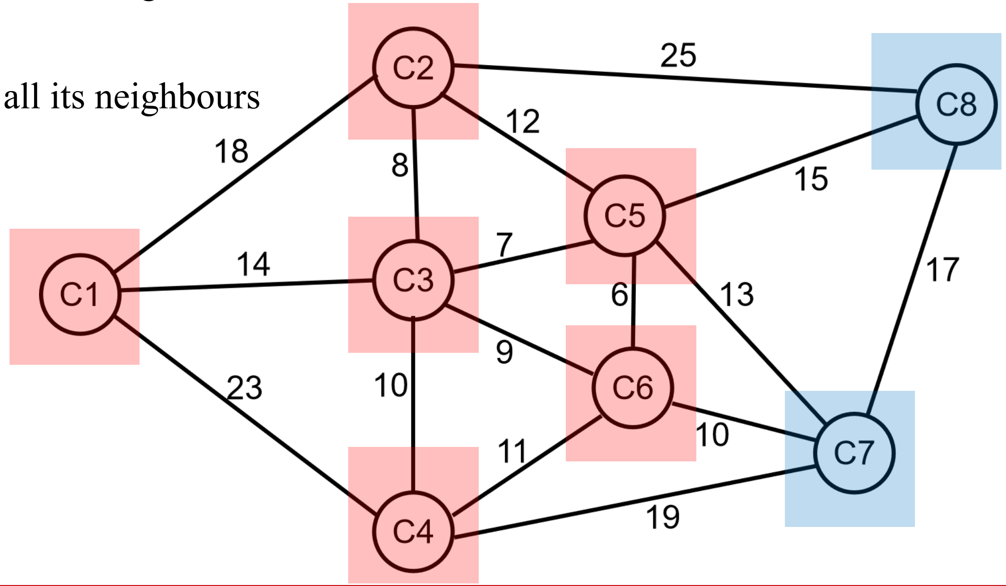
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6 C5



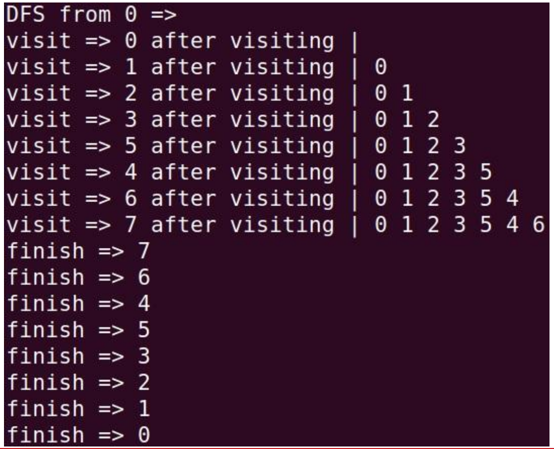
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



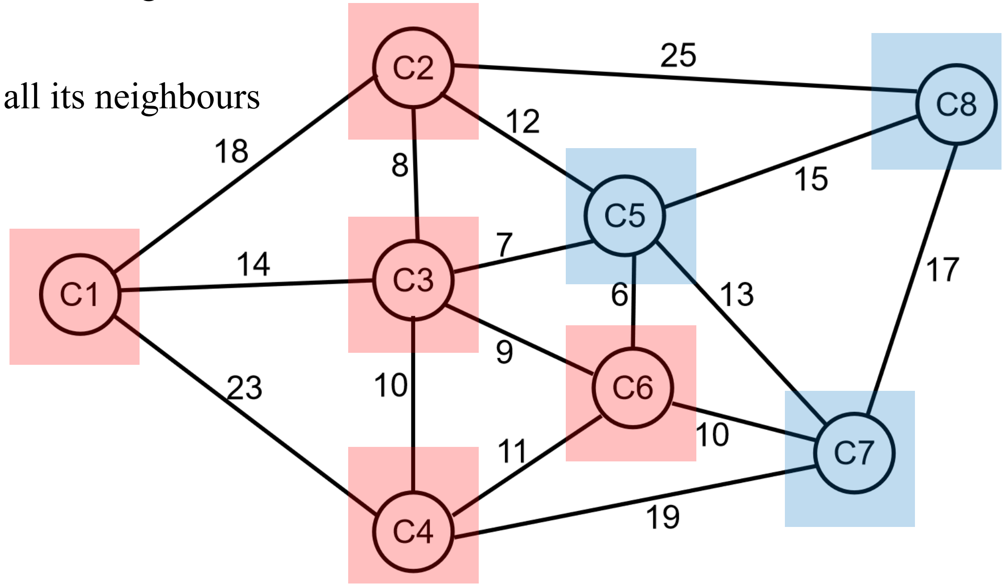
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4 C6



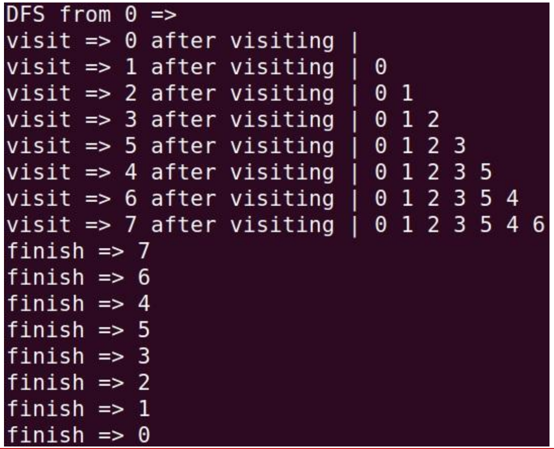
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



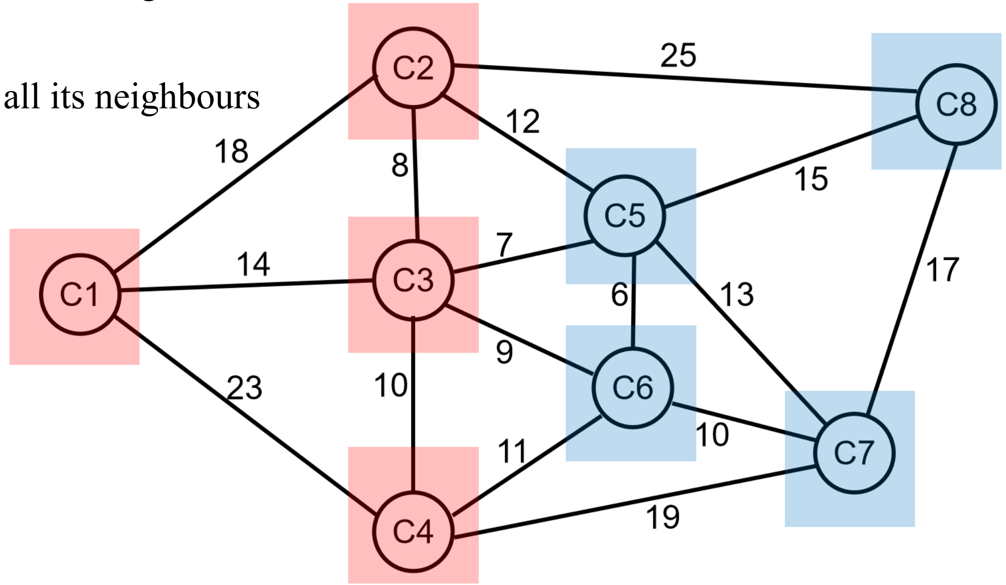
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3 C4



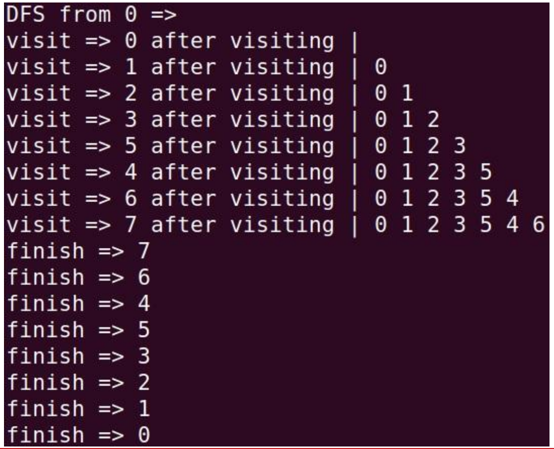
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



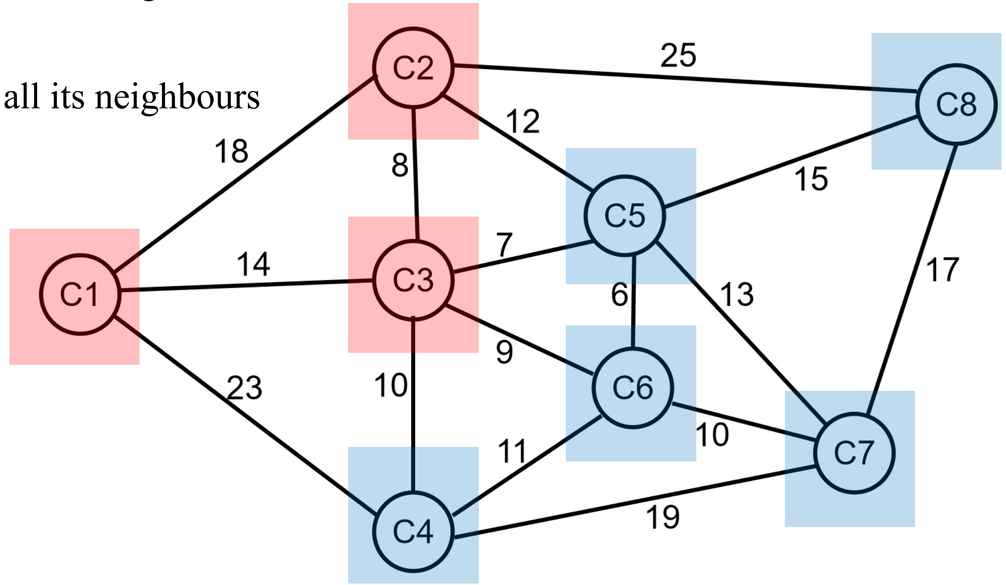
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2 C3



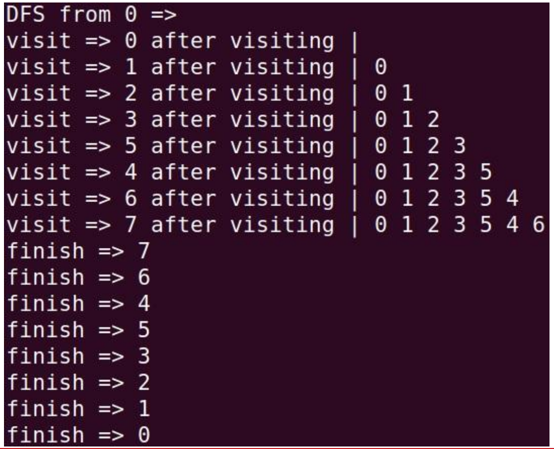
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



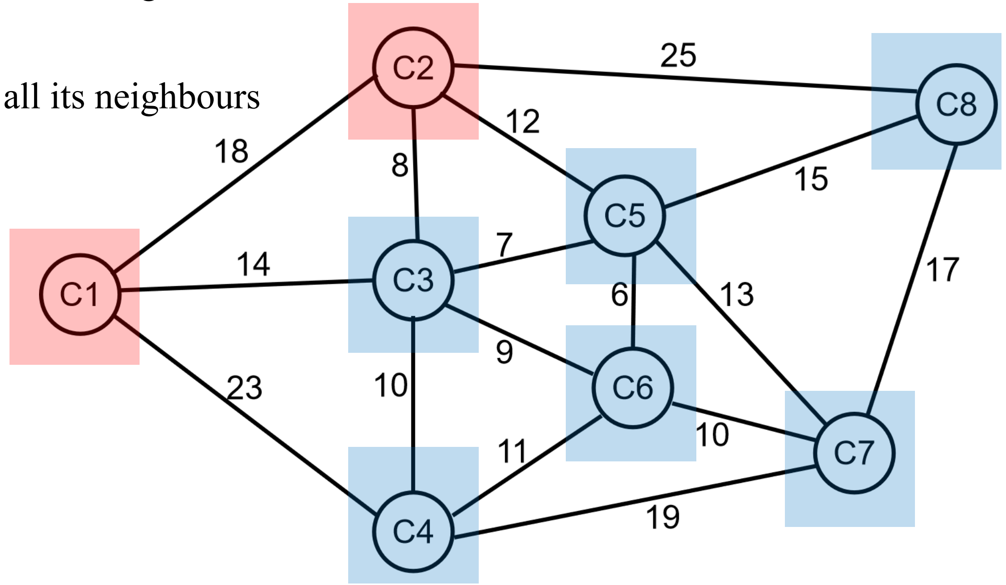
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1 C2



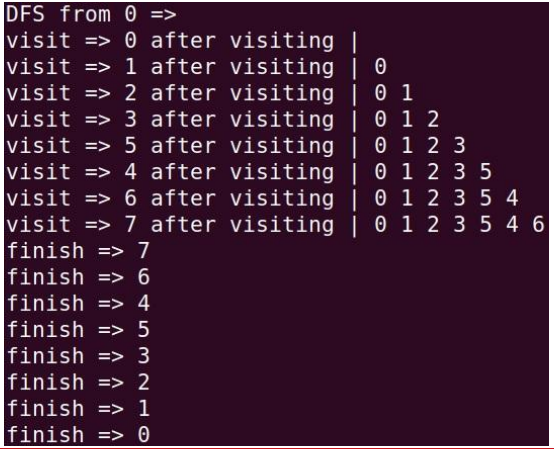
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



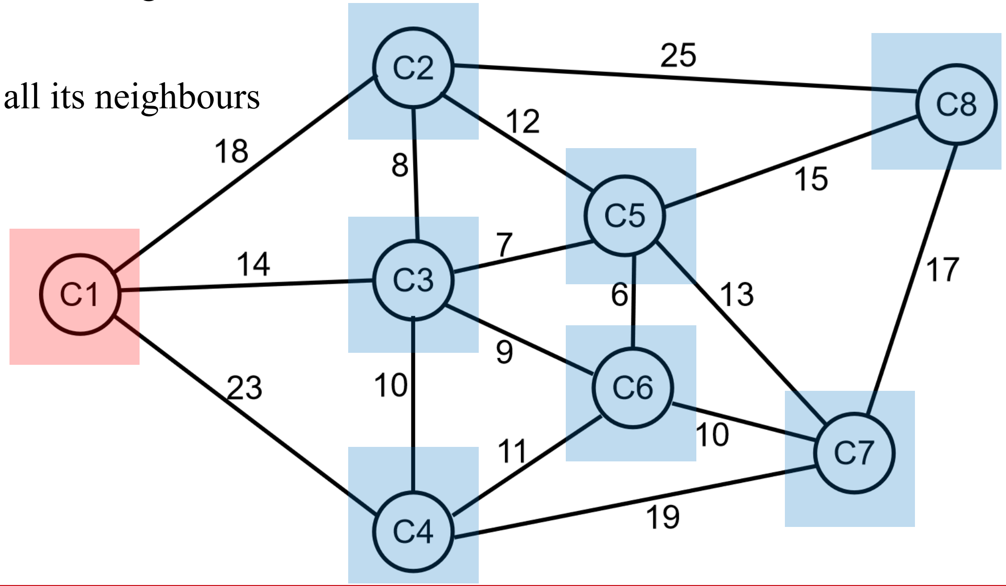
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack : C1



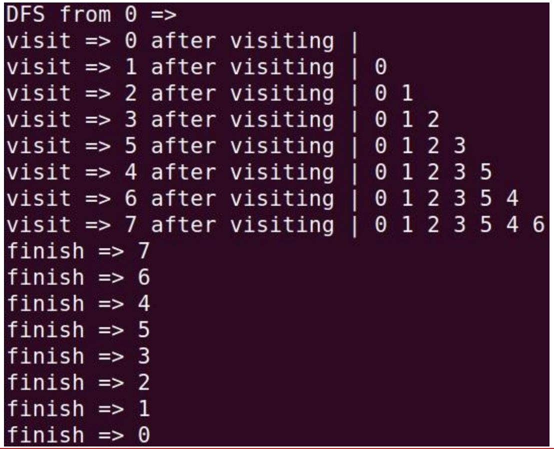
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



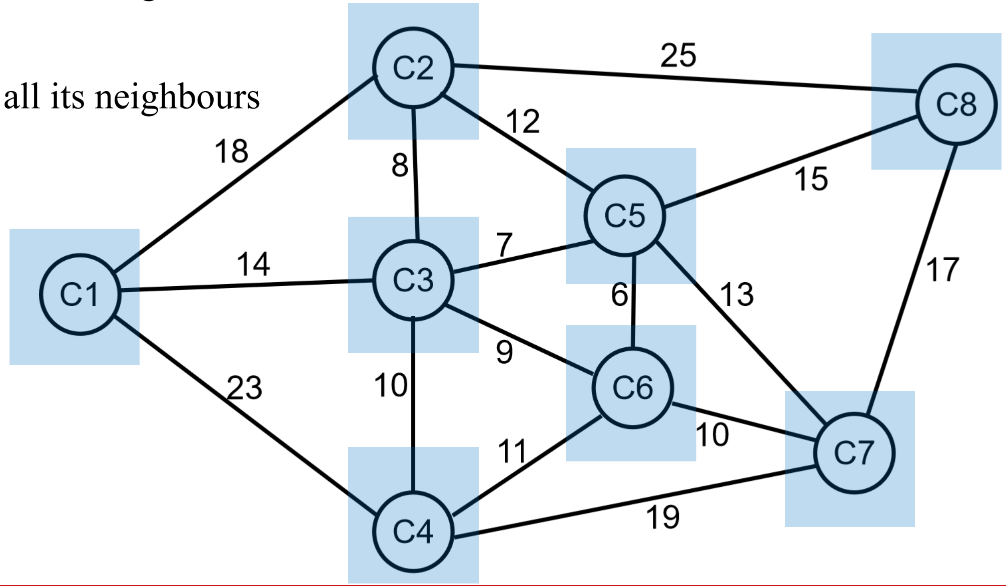
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - ***depth first search*** (DFS)
    - visit a vertex after visiting all its neighbours
  - *breadth first search* (BFS)
    - visit a vertex after visiting all its neighbours

virtual stack :



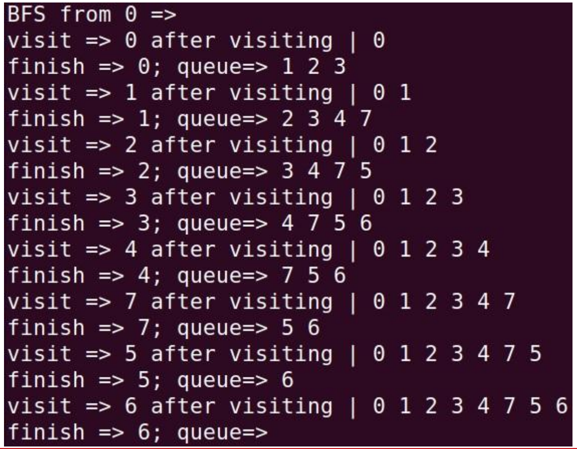
```
DFS from 0 =>
visit => 0 after visiting |
visit => 1 after visiting | 0
visit => 2 after visiting | 0 1
visit => 3 after visiting | 0 1 2
visit => 5 after visiting | 0 1 2 3
visit => 4 after visiting | 0 1 2 3 5
visit => 6 after visiting | 0 1 2 3 5 4
visit => 7 after visiting | 0 1 2 3 5 4 6
finish => 7
finish => 6
finish => 4
finish => 5
finish => 3
finish => 2
finish => 1
finish => 0
```



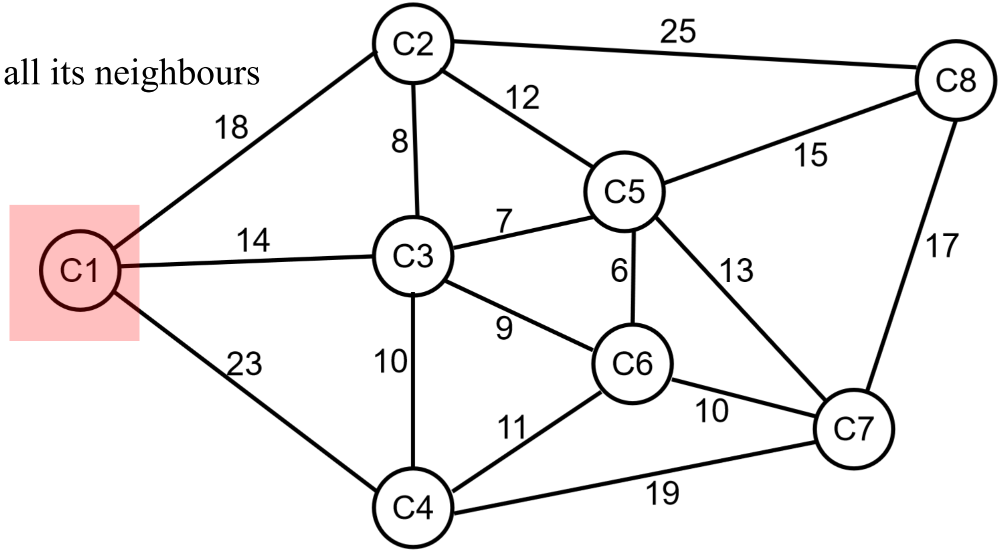
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C1



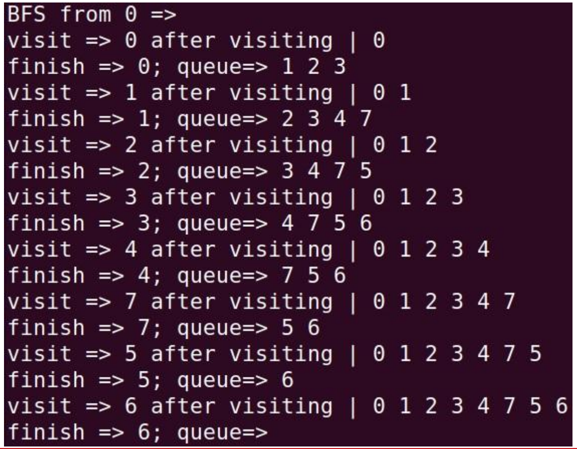
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



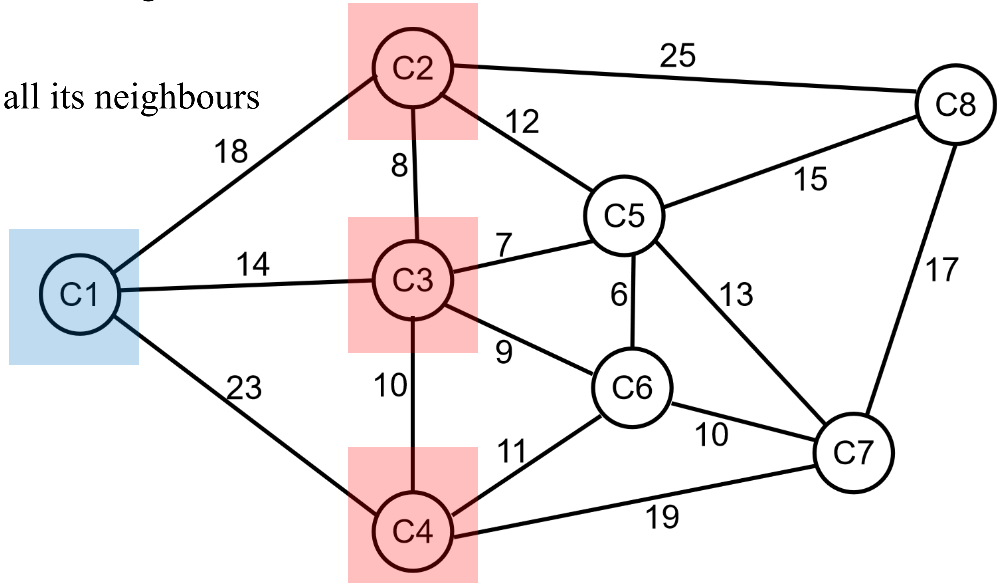
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C2 C3 C4



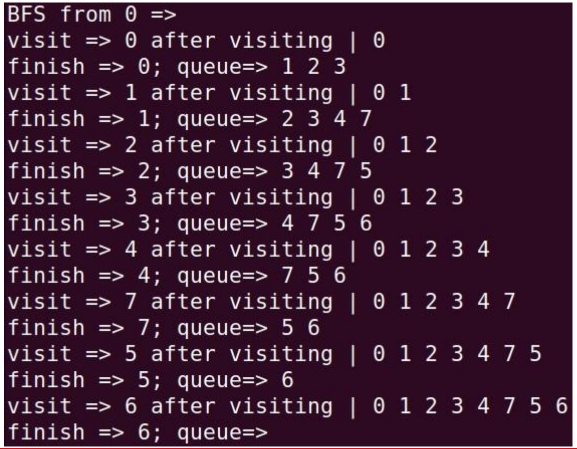
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



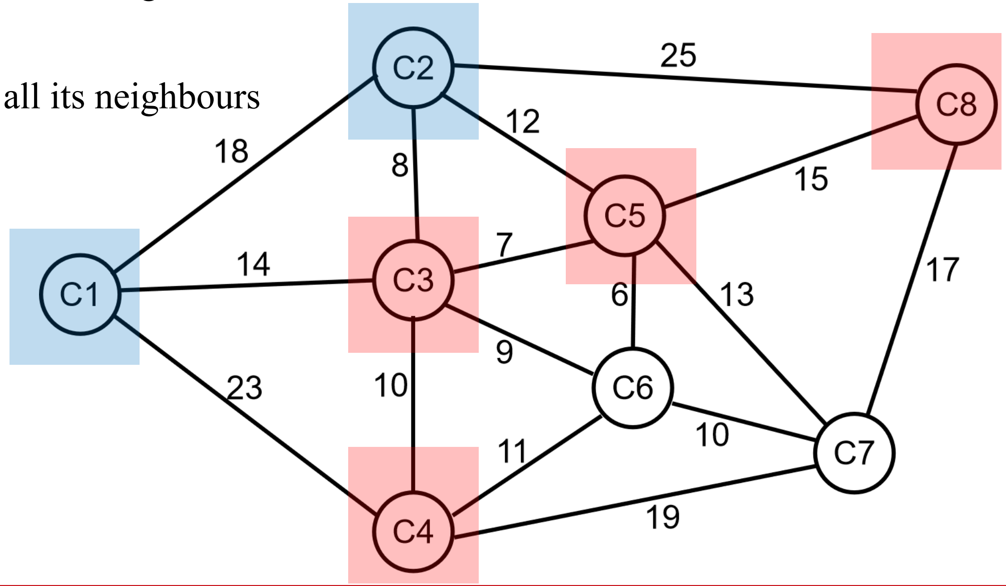
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C3 C4 C5 C8



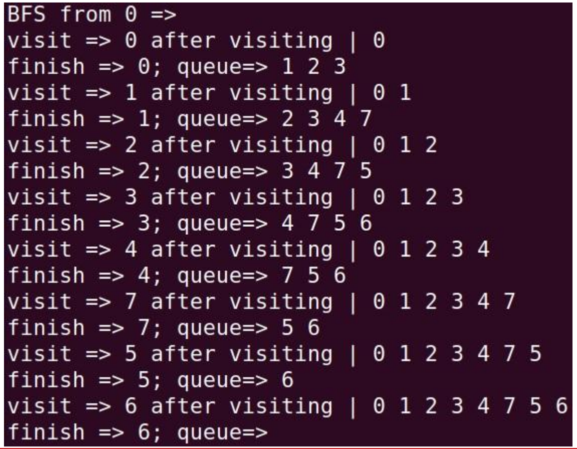
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



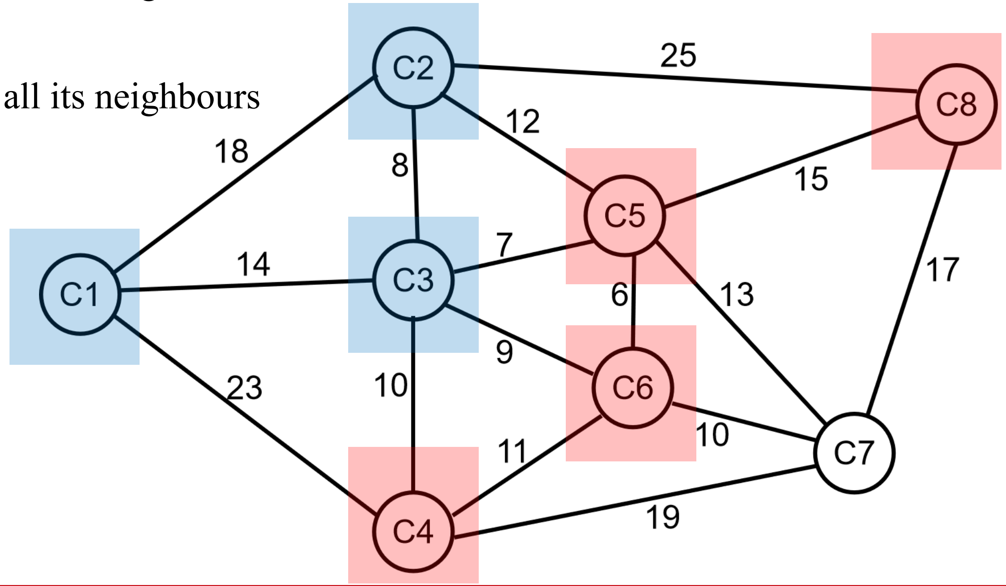
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C4 C5 C8 C6



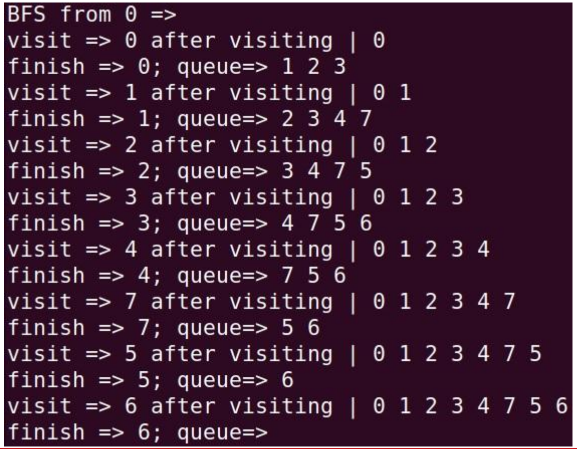
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



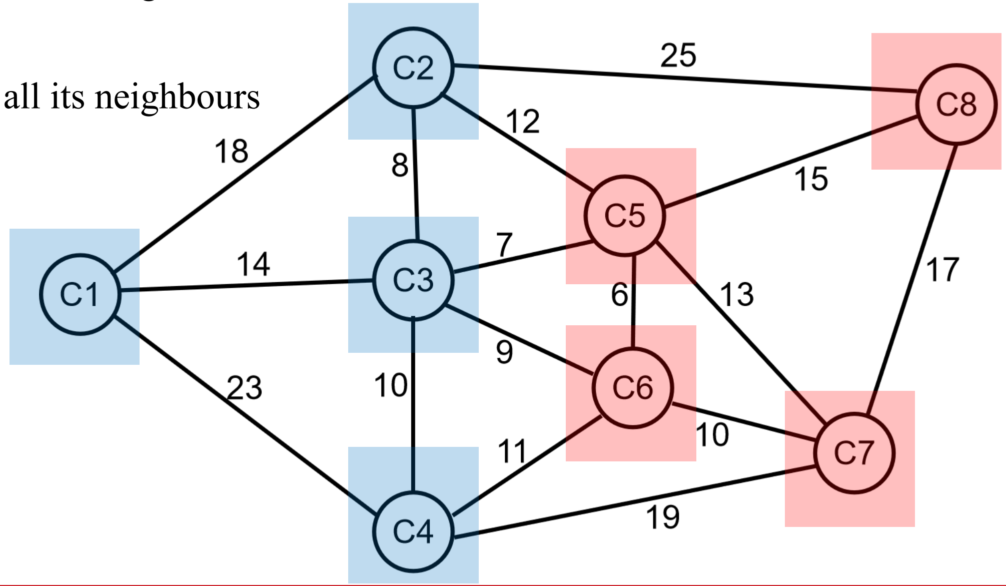
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C5 C8 C6 C7



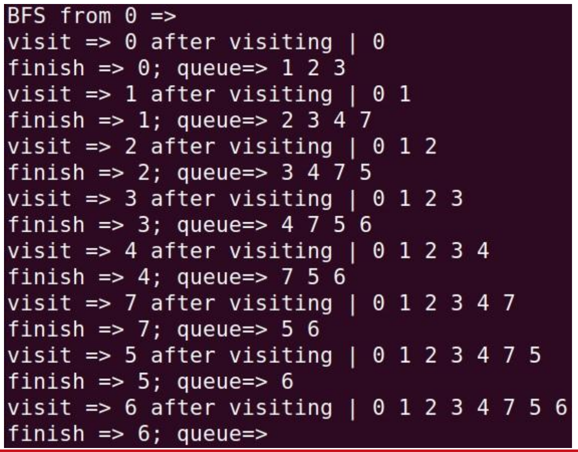
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



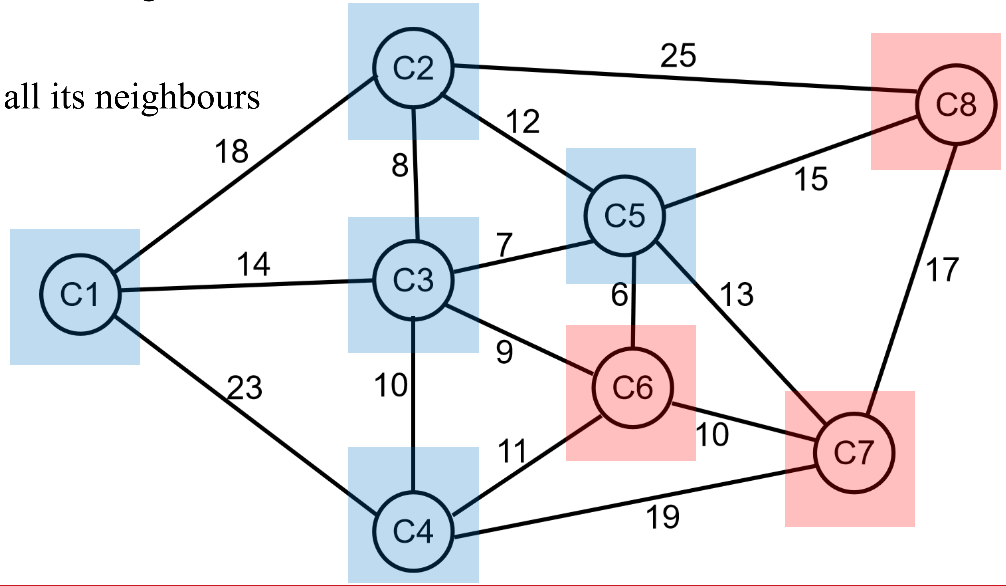
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C8 C6 C7



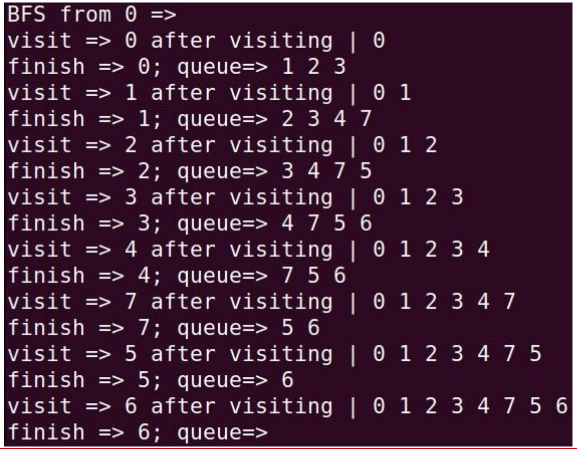
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



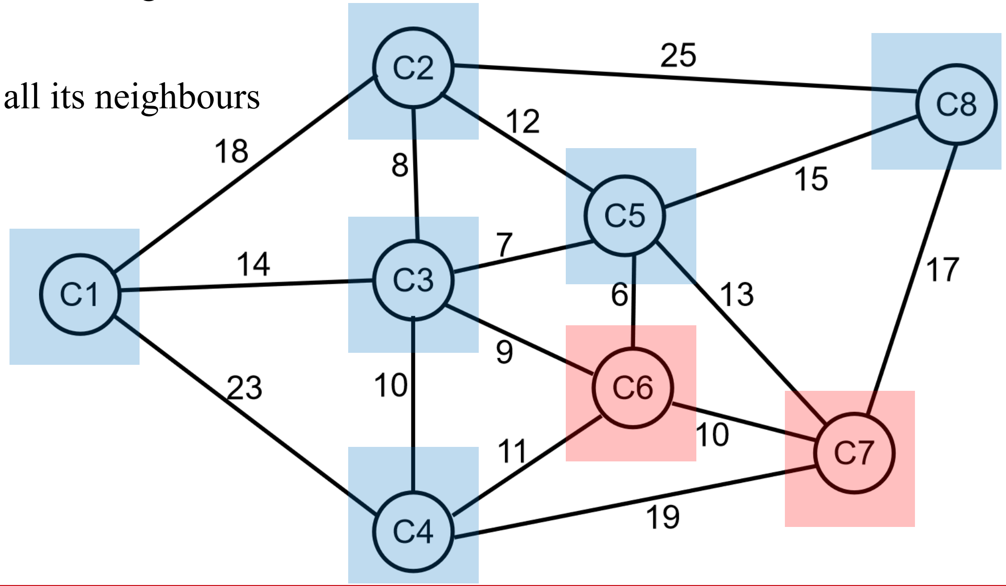
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C6 C7



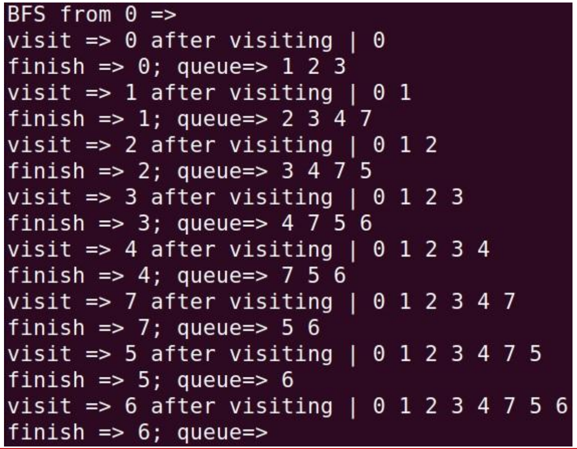
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



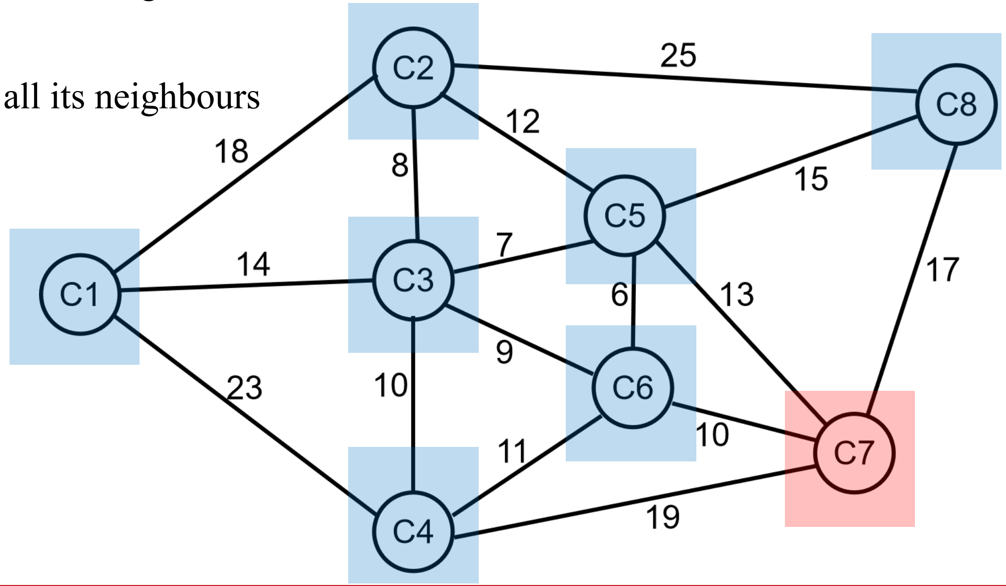
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue : C7



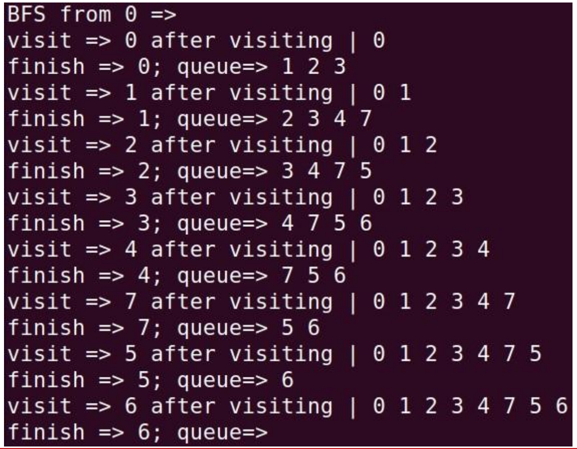
```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



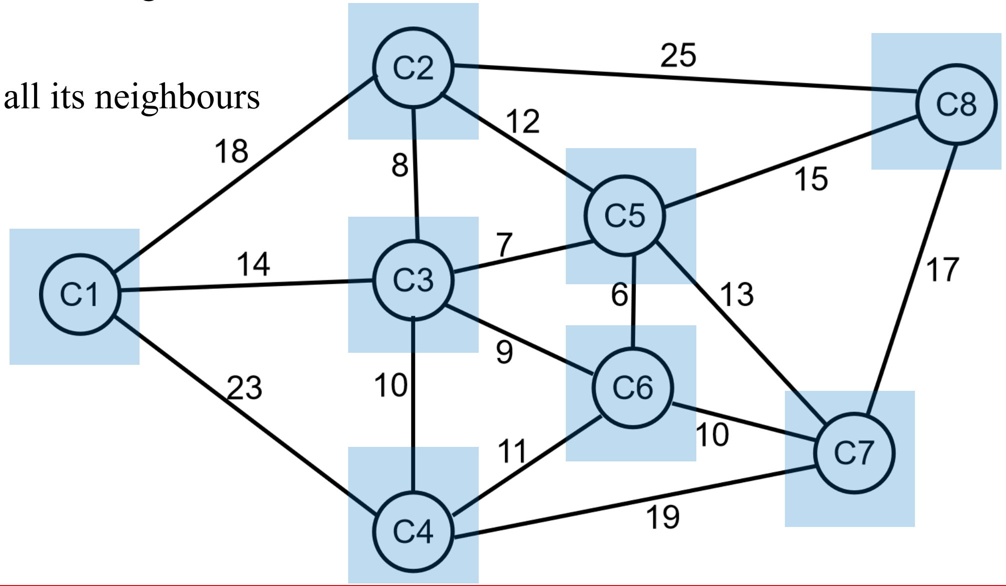
Graph

- **Graph traversal**
  - directed graph G=(V,E)
  - *depth first search* (DFS)
    - visit a vertex after visiting all its neighbours
  - ***breadth first search*** (BFS)
    - visit a vertex after visiting all its neighbours

queue :



```
BFS from 0 =>
visit => 0 after visiting | 0
finish => 0; queue=> 1 2 3
visit => 1 after visiting | 0 1
finish => 1; queue=> 2 3 4 7
visit => 2 after visiting | 0 1 2
finish => 2; queue=> 3 4 7 5
visit => 3 after visiting | 0 1 2 3
finish => 3; queue=> 4 7 5 6
visit => 4 after visiting | 0 1 2 3 4
finish => 4; queue=> 7 5 6
visit => 7 after visiting | 0 1 2 3 4 7
finish => 7; queue=> 5 6
visit => 5 after visiting | 0 1 2 3 4 7 5
finish => 5; queue=> 6
visit => 6 after visiting | 0 1 2 3 4 7 5 6
finish => 6; queue=>
```



THANK YOU

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY