Representation of Graph

Uses of Graph in Data Structures

A graph can be thought of as a data structure that is used to describe relationships between entities.

An entity can be any item that has a distinctive and independent existence. It could either be an actual physical object or an abstract idea. For example, an entity can be a person, place or an organization about which data can be stored.

In the computing world, graphs have become ubiquitous owing to their ability to not only provide abstractions to real life but also demonstrate complicated relationships with ease.

For example, a linked structure of websites can be viewed as a graph.

Basics of Graph in Data Structures

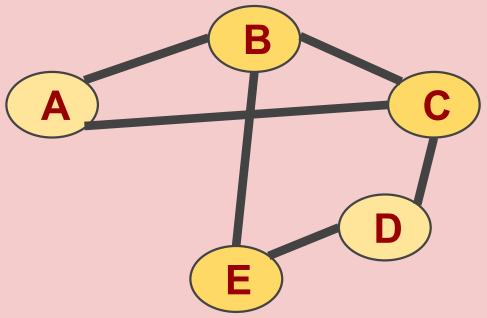
Every graph is a set of points referred to as vertices or nodes which are connected using lines called edges.

The vertices represent entities in a graph. Edges, on the other hand, express relationships between entities.

Hence, while nodes model entities, edges model relationships in a network graph.

*A graph G with a set of V vertices together with a set of E edges is represented as **G= (V,E)**.*

A SIMPLE GRAPH with five nodes and six edges



NODES = A, B, C, D, E

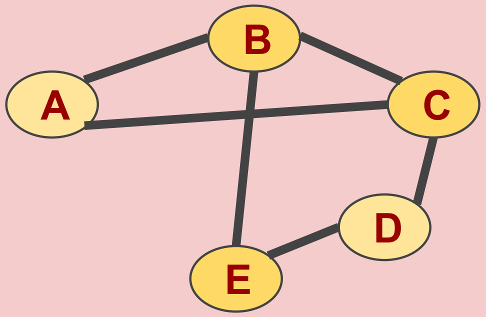
EDGES = A-B, A-C, B-C, B-E, C-D, D-E

Real-World applications of Graph

an edge might represent professional relationships that exist between people in LinkedIn or a personal relationship on a social media platform such as Facebook or Instagram.

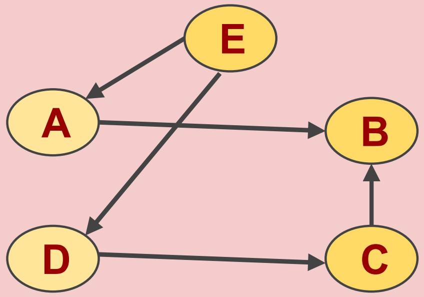
Categories of Graph

UNDIRECTED GRAPH



An undirected graph is directionless. This means that the edges have no directions. In other words, the relationship is mutual. For example, a Facebook or a LinkedIn connection.

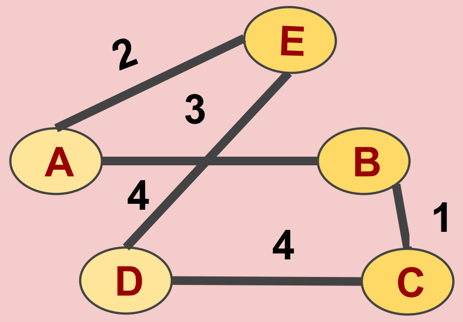
DIRECTED GRAPH



Contrarily, edges of directed graphs have directions associated with them. An asymmetric relationship between a boss and an employee or a teacher and a student can be represented as a directed graph in data structure.

Categories of Graph

WEIGHTED GRAPH



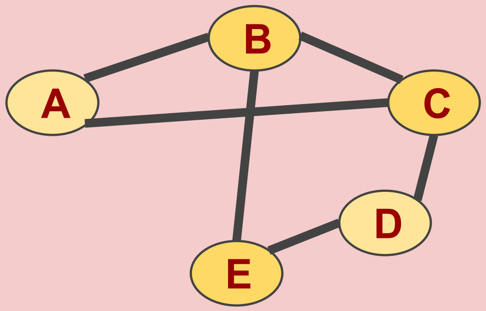
Graphs can also be weighted indicating real values associated with the edges.
Depending upon the specific use of the graph, edge weights may represent quantities such as distance, cost, similarity etc.

Representing Graphs - ADJACENCY MATRIX FOR AN UNDIRECTED GRAPH

An adjacency matrix can be thought of as a table with rows and columns. The row labels and column labels represent the nodes of a graph.
An adjacency matrix is a square matrix where the number of rows, columns and nodes are the same.
Each cell of the matrix represents an edge or the relationship between two given nodes.
For example, adjacency matrix Aij represents the number of links from i to j, given two nodes i and j.

ADJACENCY MATRIX FOR AN UNDIRECTED GRAPH

UNDIRECTED GRAPH



	A	B	C	D	E
A	0	1	0	0	1
B	1	0	1	0	0
C	0	1	0	1	0
D	0	0	1	0	1
E	1	0	0	1	0

To understand how an undirected graph can be represented using an adjacency matrix, consider a small undirected graph with five vertices. Here, A is connected to B, but B is connected to A as well. Hence, both the cells i.e., the one with source A destination B and the other one with source B destination A are marked one. This suffices the requirement of an undirected edge. Observe that the second entry is at a mirrored location across the main diagonal.

Representing Graphs - CODE ADJACENCY MATRIX FOR AN UNDIRECTED GRAPH

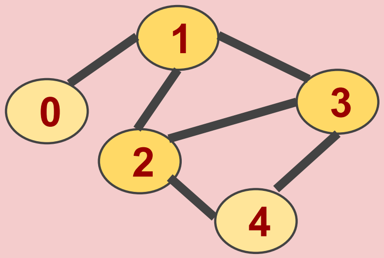
```
#include <stdio.h>
int V;
// Initialize the matrix to zero
void init(int arr[][V])
{
  int i, j;
  for (i = 0; i < V; i++)
    for (j = 0; j < V; j++)
      arr[i][j] = 0;
}

// Add edges
void addEdge(int arr[][V], int i, int j)
{
  arr[i][j] = 1;
  arr[j][i] = 1;
}
```

```
// Print the matrix
void printAdjMatrix(int arr[][V])
 {

  int i, j;
  for (i = 0;i < V;i++)
  {
   for (j = 0;j < V;j++)
   {
     printf("%d ", arr[i][j]);
   }
   printf("\n");
  }
}
```

```
int main()
{
  int e1,e2,me,n,i;
  scanf("%d",&V);
  int adjMatrix[V][V];
  init(adjMatrix);
  n=V;
  me=n*(n-1)/2;
  for(i=1;i<=me;i++)
  {
   scanf("%d%d",&e1,&e2);
   addEdge(adjMatrix, e1,e2);
   if((e1==-1)&&(e2==-1))
      break;
  }
  printAdjMatrix(adjMatrix);
  return 0;
}
```



OUTPUT :

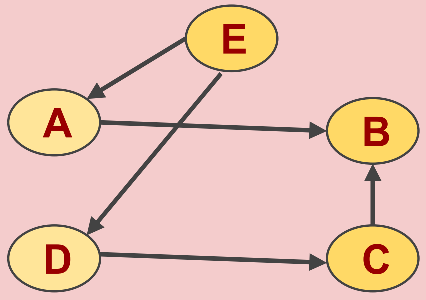
```
5
0 1
1 2
1 3
2 3
2 4
3 4
-1 -1
```

```
0 1 0 0 0
1 0 1 1 0
0 1 0 1 1
0 1 1 0 1
0 0 1 1 0
```

Representing Graphs - ADJACENCY MATRIX FOR A DIRECTED GRAPH

An adjacency matrix can be thought of as a table with rows and columns. The row labels and column labels represent the nodes of a graph.
An adjacency matrix is a square matrix where the number of rows, columns and nodes are the same.
Each row and column correspond to a node or a vertex of a graph. The cells within the matrix represent the connection that exists between nodes.

DIRECTED GRAPH



ADJACENCY MATRIX FOR A DIRECTED GRAPH

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	0	0	0
C	0	1	0	0	0
D	0	0	1	0	0
E	1	0	0	1	0

Since, in the given directed graph, no node is connected to itself, all cells lying on the diagonal of the matrix are marked zero.

For the rest of the cells, if there exists a directed edge from a given node to another, then the corresponding cell will be marked one else zero.

Representing Graphs - CODE ADJACENCY MATRIX FOR A DIRECTED GRAPH

```c
#include<stdio.h>
int V;

//init matrix to 0
void init(int arr[][V])
{
    int i,j;
    for(i = 0; i < V; i++)
        for(j = 0; j < V; j++)
            arr[i][j] = 0;
}

//Add edge. set arr[src][dest] = 1
void addEdge(int arr[][V],int src, int dest)
{
     arr[src][dest] = 1;
}
```

```c
void printAdjMatrix(int arr[][V])
{
     int i, j;
      for(i = 0; i < V; i++)
     {
         for(j = 0; j < V; j++)
         {
             printf("%d ", arr[i][j]);
         }
         printf("\n");
     }
}
```

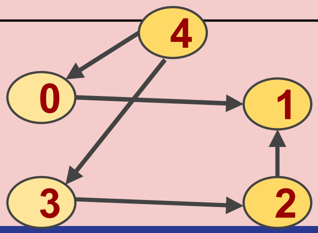
```c
//print the adjMatrix
int main()
{
    int e1,e2,me,n,i;
    scanf("%d",&V);
    int adjMatrix[V][V];

    init(adjMatrix);
    n=V;
    me=n*(n-1);
    for(i=1;i<=me;i++)
    {
    scanf("%d%d",&e1,&e2);
    addEdge(adjMatrix, e1,e2);
    if((e1==-1)&&(e2==-1))
       break;
    }
  printAdjMatrix(adjMatrix);
 return 0;
}
```

OUTPUT :

```
5
0 1
2 1
3 2
4 0
4 3
-1 -1
```

```
0 1 0 0 0
0 0 0 0 0
0 1 0 0 0
0 0 1 0 0
1 0 0 1 0
```


$$\begin{tikzcd}
 & 4 \arrow[dl] \arrow[ddl] & \\
0 \arrow[rr] & & 1 \\
3 \arrow[rr] & & 2 \arrow[u]
\end{tikzcd}$$

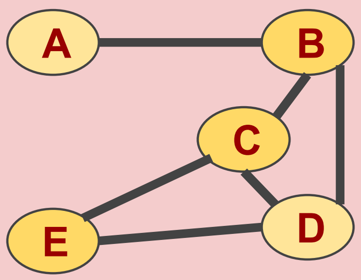
Representing Graphs - ADJACENCY LIST FOR AN UNDIRECTED GRAPH

Fig provides an example of an undirected graph along with its adjacency list.

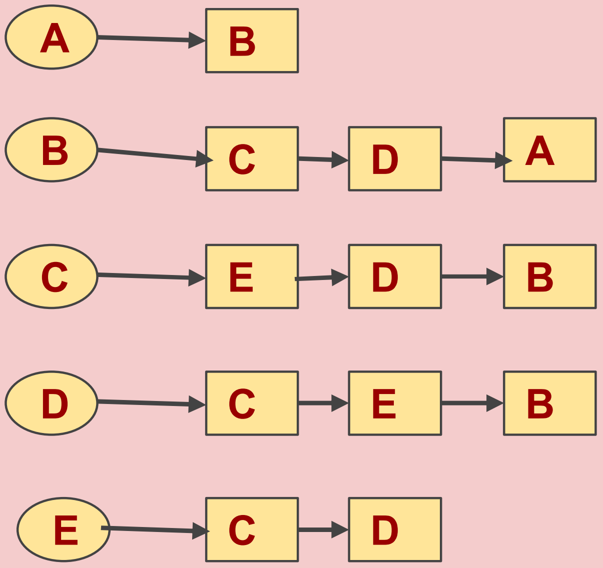
Adjacency list enables faster search process in comparison to adjacency matrix.

However, it is not the best representation of graphs especially when it comes to adding or removing nodes. For example, deleting a node would involve looking through all the adjacency lists to remove a particular node from all lists.

UNDIRECTED GRAPH



ADJACENCY LIST FOR A UNDIRECTED GRAPH



The adjacency set mitigates a few of the challenges posed by adjacency list. Adjacency set is quite similar to adjacency list except for the difference that instead of a linked list; a set of adjacent vertices is provided.

Adjacency list and set are often used for sparse graphs with few connections between nodes.

Contrarily, adjacency matrix works well for well-connected graphs comprising many nodes.

Representing Graphs - CODE ADJACENCY LIST FOR AN UNDIRECTED GRAPH

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
  int vertex;
  struct node* next;
};
struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
};

// Create a node
struct node* createNode(int v)
{
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}
```

```
// Create a graph
struct Graph* createAGraph(int vertices)
{
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  int i;
  for (i = 0; i < vertices; i++)
    graph->adjLists[i] = NULL;

  return graph;
}
```

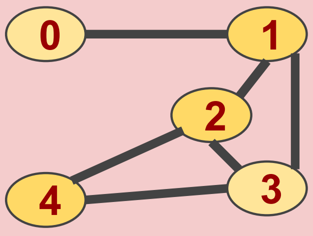
Representing Graphs - CODE ADJACENCY LIST FOR AN UNDIRECTED GRAPH

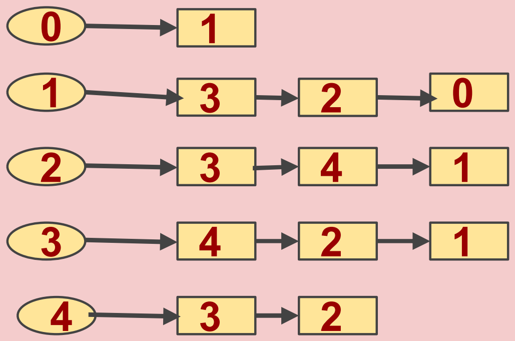
```
// Add edge
void addEdge(struct Graph* graph, int s, int d)
{
  // Add edge from s to d
  struct node* newNode = createNode(d);
  newNode->next = graph->adjLists[s];
  graph->adjLists[s] = newNode;
  // Add edge from d to s
  newNode = createNode(s);
  newNode->next = graph->adjLists[d];
  graph->adjLists[d] = newNode;
}
```

```
// Print the graph
void printGraph(struct Graph* graph)
{
  int v;
  for (v = 0; v < graph->numVertices; v++)
 {
    struct node* temp = graph->adjLists[v];
    printf("Vertex %d\n: ", v);
    while (temp)
   {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}
```

Representing Graphs - CODE ADJACENCY LIST FOR AN UNDIRECTED GRAPH

```
int main()
{
  int main()
 {
    int n,e1,e2,i,me;
    scanf("%d",&n);
    struct Graph* graph = createAGraph(n);
    me=n*(n-1)/2;
    for(i=1;i<=me;i++)
    {
    scanf("%d%d",&e1,&e2);
    addEdge(graph, e1, e2);
    if((e1==-1)&&(e2==-1))
      break;
    }
  printGraph(graph);
  return 0;
  }
}
```





OUTPUT :

```
5
0 1
1 2
1 3
2 3
2 4
3 4
-1 -1
```

```
Vertex 0
: 1 ->

 Vertex 1
: 3 -> 2 -> 0 ->

 Vertex 2
: 3 -> 4 -> 1 ->

 Vertex 3
: 4 -> 2 -> 1 ->

 Vertex 4
: 3 -> 2 ->
```

Representing Graphs - ADJACENCY LIST FOR A DIRECTED GRAPH

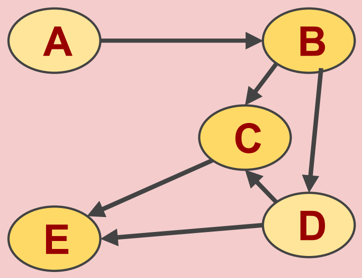
In adjacency list representation of a graph, every vertex is represented as a node object.

The node may either contain data or a reference to a linked list.

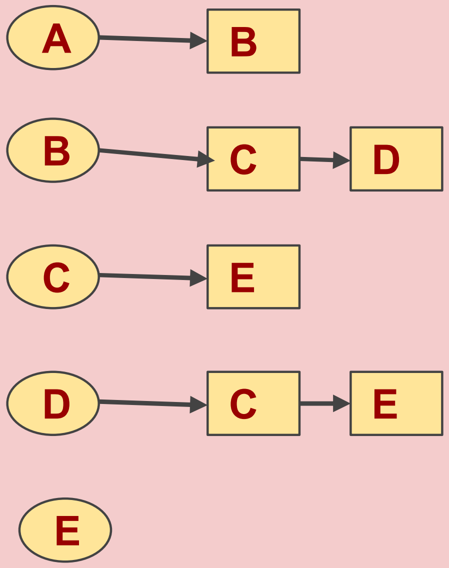
This linked list provides a list of all nodes that are adjacent to the current node.

Consider a graph containing an edge connecting node A and node B. Then, the node A will be available in node B's linked list. Fig shows a sample graph of 5 nodes and its corresponding adjacency list.

DIRECTED GRAPH



ADJACENCY LIST FOR A DIRECTED GRAPH



Note that the list corresponding to node E is empty while lists corresponding to nodes B and D have 2 entries each.

Representing Graphs - CODE ADJACENCY LIST FOR A DIRECTED GRAPH

```
#include <stdio.h>
#include <stdlib.h>

// Define the maximum number of vertices in the graph
int N;

// Data structure to store a graph object
struct Graph
{
   // An array of pointers to Node to represent an adjacency list
   struct Node* head[10];
};

// Data structure to store adjacency list nodes of the graph
struct Node
{
   int dest;
   struct Node* next;
};

// Data structure to store a graph edge
struct Edge
 {
   int src, dest;
  } *edges[50];
```

Representing Graphs - CODE ADJACENCY LIST FOR A DIRECTED GRAPH

```
// Function to create an adjacency list from specified edges
struct Graph* createGraph(struct Edge edges[], int n)
{
    // allocate storage for the graph data structure
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

    // initialize head pointer for all vertices
    for (int i = 0; i < N; i++)
    {
        graph->head[i] = NULL;
    }

    // add edges to the directed graph one by one
    for (int i = 0; i < n; i++)
    {
        // get the source and destination vertex
        int src = edges[i].src;
        int dest = edges[i].dest;
```

```
 // allocate a new node of adjacency list from src to dest
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->dest = dest;

        // point new node to the current head
        newNode->next = graph->head[src];

        // point head pointer to the new node
        graph->head[src] = newNode;
    }
  return graph;
}
```

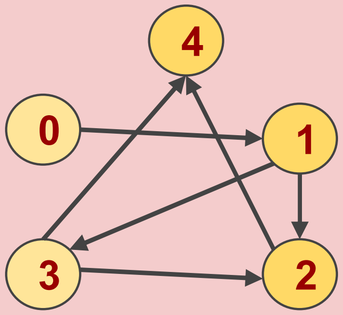
Representing Graphs - CODE ADJACENCY LIST FOR A DIRECTED GRAPH

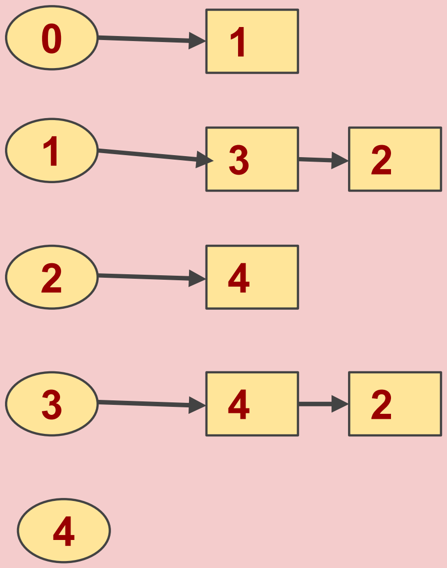
```
// Function to print adjacency list representation of a graph
void printGraph(struct Graph* graph)
{
   for (int i = 0; i < N; i++)
   {
      // print current vertex and all its neighbors
      struct Node* ptr = graph->head[i];
      while (ptr != NULL)
      {
         printf("(%d—>%d)",i,ptr->dest);
         ptr = ptr->next;
         }
         printf("\n");
      }
}
```

```
// Directed graph implementation in C
int main(void)
{
   int n,i;
   scanf("%d",&N);
   scanf("%d",&n);
   // input array containing edges of the graph (as per the above diagram)
   // (x, y) pair in the array represents an edge from x to y
   struct Edge edges[n];
   for (i = 0; i < n; i++)
   {
      // get the source and destination vertex
      scanf("%d",&edges[i].src);
      scanf("%d",&edges[i].dest);
    }
   // construct a graph from the given edges
   struct Graph *graph = createGraph(edges, n);

   // Function to print adjacency list representation of a graph
   printGraph(graph);
   return 0;
}
```

Representing Graphs - CODE ADJACENCY LIST FOR A DIRECTED GRAPH





OUTPUT :

```
5
6
0 1
1 2
1 3
2 4
3 2
3 4
```

```
(0—>1)
(1—>3)(1—>2)
(2—>4)
(3—>4)(3—>2)
```