Graph

- Basic Concepts
  - Representation
  - Adjacency Matrics
  - Adjacency Lists
- Traversal Algorithms
  - Depth-first Search
  - Breadth-first Search
- Applications
  - Spanning Tree
  - Shortest Paths
  - Activity Networks
  - Critical Activities

Introduction

Graphs have been widely used in:

- analysis of electrical circuits
- finding shortest routes
- project planning
- identification of chemical compounds
- statistical mechanics
- genetics
- cybernetics
- linguistics
- social science
- …

Definitions

Graph $G = (V, E)$

- vertices $V(G) \neq \varnothing$
- edges $E(G)$
- undirected graph: $(u, v) = (v, u)$
- directed graph: $\langle u, v \rangle$, u---tail, v---head,

$\langle u,v \rangle \neq \langle v,u \rangle$

Examples

G_1: $V(G_1)=\{0,1,2,3\}$

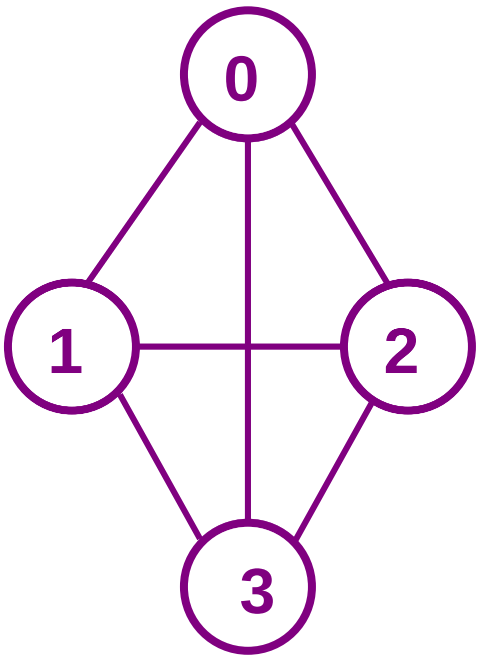
$E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$

G_2: $V(G_2)=\{0,1,2,3,4,5,6\}$

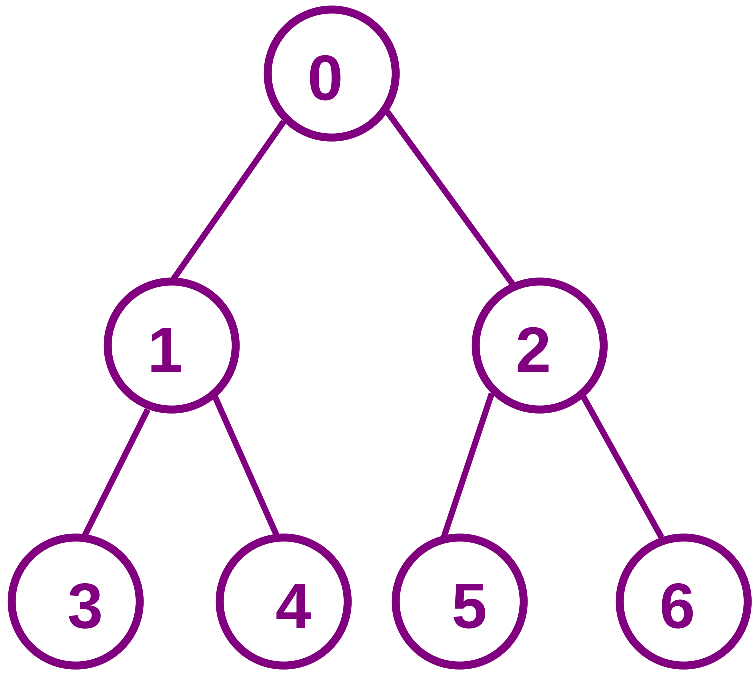
$E(G_2)=\{(0,1),(0,2),,(1,3),(1,4),(2,5),(2,6)\}$

G_3: $V(G_3) = \{0,1,2\}$

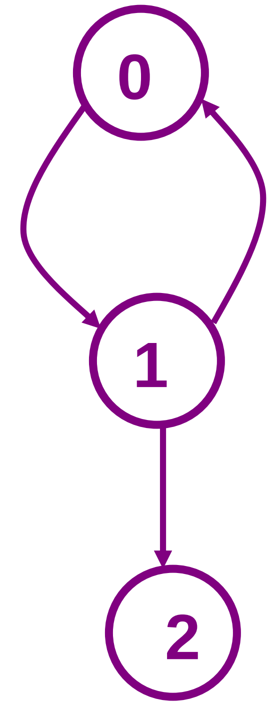
$E(G_3) = \{<0,1>,<1,0>,<1,2>\}$ (directed)



(a) G_1



(b) G_2



(c) G_3

Restrictions:

(1) (v, v) or <v, v> is not legal, such edges are known as self edges.

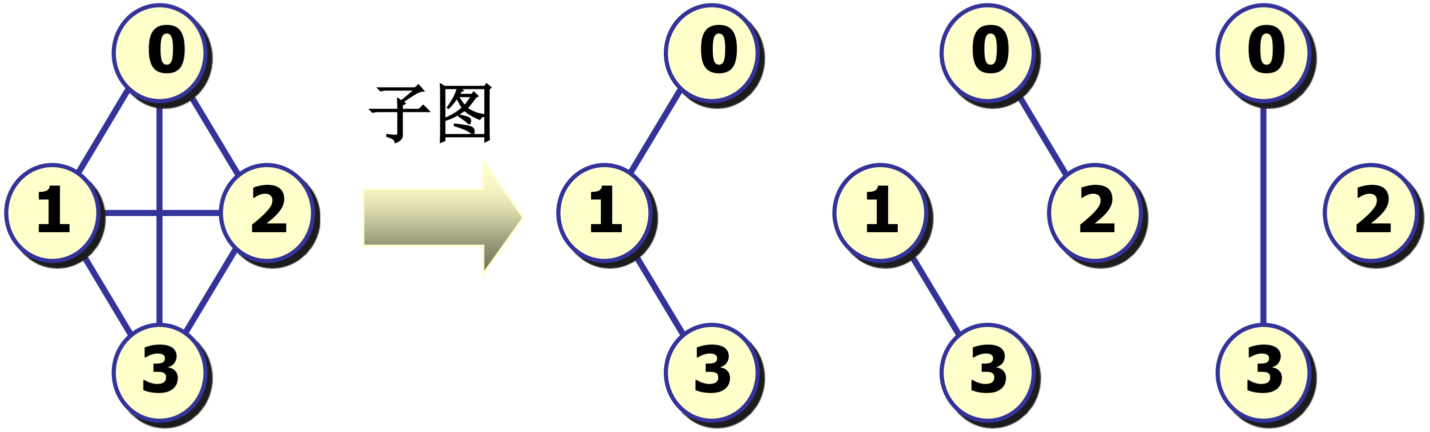
(2) multiple occurrences of the same edges are not allowed. If allowed, we get a multigraph.

- The maximum number of edges in any n-vertex, undirected graph is $n(n-1)/2$, and in directed graph is $n(n-1)$.
- An n-vertex undirected graph with $n(n-1)/2$ edges is said to be complete.

Subgraph

- If (u, v) $\in$E(G), we say u and v are adjacent and edge (u, v) is incident on vertices u and v. If <u, v> is a directed edge, then vertex u is adjacent to v, and v is adjacent from u, <u, v> is incident to u and v.
- A subgraph of G is a graph G` such that V(G`) $\subseteq$ V(G) and E(G`) $\subseteq$ E(G).

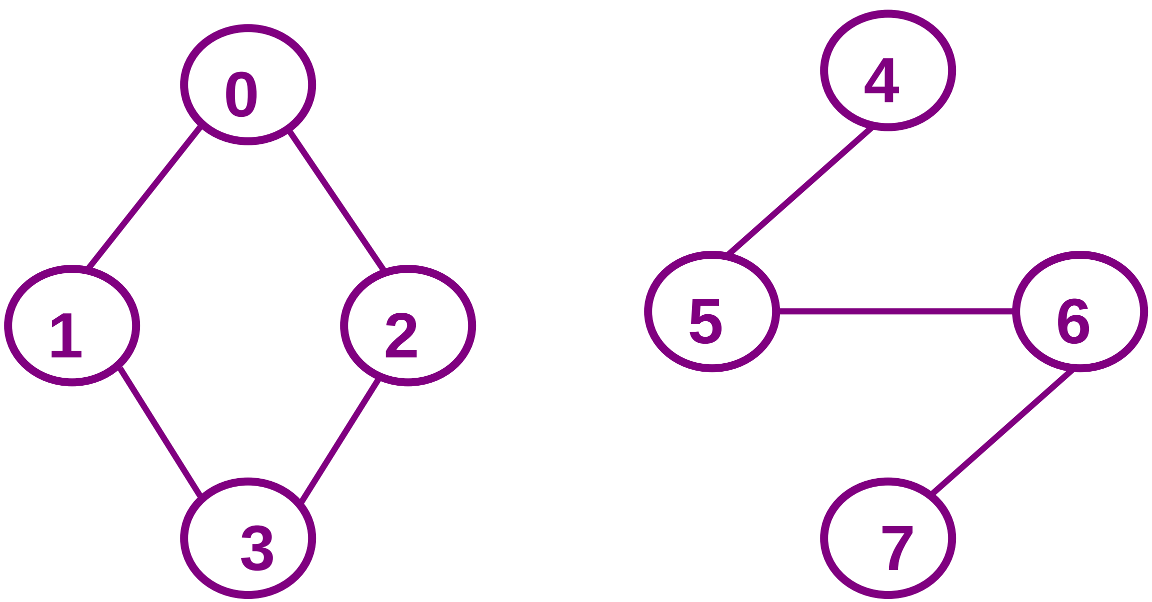
Subgraph



- A path from u to v in G is a sequence of vertices $u, i_1, i_2, \ldots, i_k, v$ such that $(u, i_1), (i_1, i_2), \ldots, (i_k, v)$ are edges in E(G). If G` is directed, then $<u, i_1>, <i_1, i_2>, \ldots, <i_k, v>$ are edges in E(G`).
- The length of a path is the number of edges on it.
- A simple path is a path in which all vertices except possibly the first and last are distinct.

- A cycle is a simple path in which the first and last vertices are the same.
- For directed graph, we have directed paths and cycles.
- In an undirected G, u and v are connected iff there is a path in G from u to v (also from v to u).
- An undirected G is connected iff for every pair of distinct u and v in V(G), there is a path from u to v.

- A connected component is a maximal connected subgraph.



A graph with two connected components

- A tree is a connected acyclic graph.
- A directed G is strongly connected iff for every pair of distinct u and v in V(G), there is a directed path from u to v and also from v to u.
- A strongly connected component is a maximal subgraph that is strongly connected.
- The degree of a vertex is the number of edges incident to it.

- For directed G, the in-degree of $v \in V(G)$ is the number of edges for which v is the head. The out-degree is the number of edges for which v is the tail.

- If d_i is the degree of vertex i in G with n vertices and e edges, then

$$e = \left(\sum_{i=0}^{n-1} d_i \right) / 2$$

- We'll refer to a directed graph as digraph, and a undirected graph as graph.

ADT of Graph

```
class Graph
{ // A non empty set of vertices and a set of undirected
  // edges, where each edge is a pair of vertices.
public:
    virtual ~Graph(){ };
      // virtual destructor
    bool IsEmpty() const {return n==0;};
      // return true iff graph has no vertices
    int NumberOfVertices() const {return n;};
      // return the number of vertices in the graph
    int NumberofEdges() const {return e;};
      // return number of edges in the graph
    virtual int Degree(int u) const =0;
      // return number of edges incident to vertex u
```

```
    virtual bool ExisteEdge(int u, int v) const =0;
      // return true iff graph has edge (u, v)
    virtual void InsertVertex (int v) =0;
      // insert vertex v into graph, v has no incident edges
    virtual void InsertEdge (int u, int v) =0;
      // insert edge (u, v) into graph
    virtual void DeleteVertex (int v);
      // delete v and all edges incident to it
    virtual void DeleteEdge (int u, int v) =0;
      // delete edge (u, v) from the graph
private:
     int n;    // number of vertices
     int e;    // number of edges
};
```

Graph

- Basic Concepts
  - Representation
  - Adjacency Matrics
  - Adjacency Lists
- Traversal Algorithms
  - Depth-first Search
  - Breadth-first Search
- Applications
  - Spanning Tree
  - Shortest Paths
  - Activity Networks
  - Critical Activities

Graph Representations

Three most commonly used representations:

(1) Adjacency matrices

(2) Adjacency lists

(3) Adjacency multilists

The actual choice depends on application.

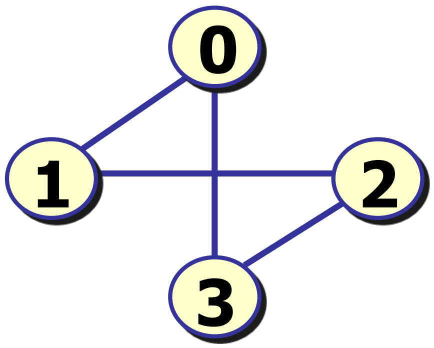
Graph

- Basic Concepts
  - Representation
  - Adjacency Matrics
  - Adjacency Lists
- Traversal Algorithms
  - Depth-first Search
  - Breadth-first Search
- Applications
  - Spanning Tree
  - Shortest Paths
  - Activity Networks
  - Critical Activities

Adjacency Matrix

The adjacency matrix of G is a $n \times n$ array, say a, such that:

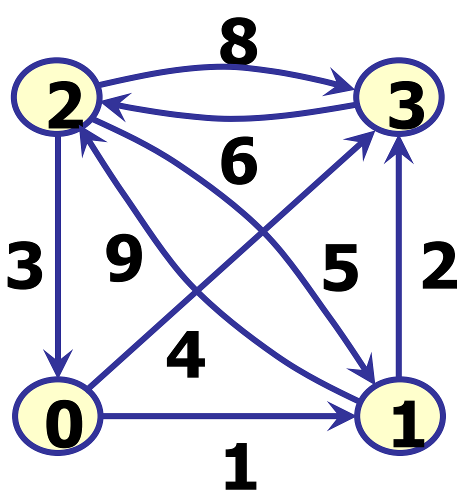
$$a[i,j] = \begin{cases} 1 & \text{iff } (i, j) \in E(G) \text{ (or } \langle i, j \rangle \in E(G) \text{)} \\ 0 & \text{otherwise} \end{cases}$$



$$A.edge = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$A.edge = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$
\mathrm{A.edge}[i][j]=\begin{cases}\mathrm{W}(i,j), & 若 i\neq j 且 <i,j>\in \mathrm{E} 或 (i,j)\in \mathrm{E}\\ \infty, & 若 i\neq j 且 <i,j>\notin \mathrm{E} 或 (i,j)\notin \mathrm{E}\\ 0, & 若 i==j\end{cases}
$$



$$
\mathrm{A.edge}=\begin{bmatrix}0 & 1 & \infty & 4\\ \infty & 0 & 9 & 2\\ 3 & 5 & 0 & 8\\ \infty & \infty & 6 & 0\end{bmatrix}
$$

For an graph, a is symmetric, and

$$d_i = \sum_{j=0}^{n-1} a[i][j]$$

For a digraph, a may not be symmetric, and

$$\text{out-}d_i = \sum_{j=0}^{n-1} a[i][j]$$

$$\text{in-}d_j = \sum_{i=0}^{n-1} a[i][j]$$

Problem: how many edges are there in G?

Using a, we need $O(n^2)$. When $e \ll n^2/2$, and if we explicitly represent only edges in G, then we can solve the above problem in $O(e + n)$.

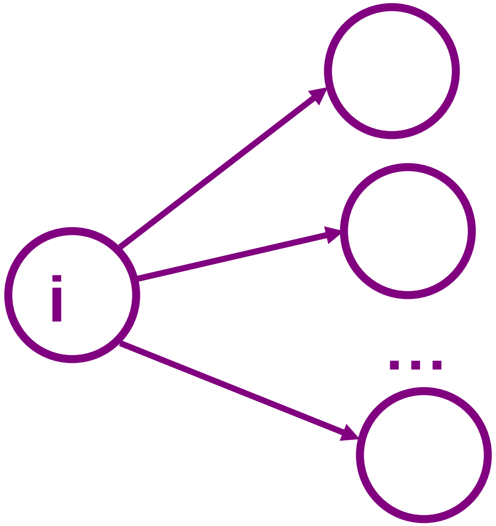
This lead to:

Graph

- Basic Concepts
  - Representation
  - Adjacency Matrics
  - Adjacency Lists
- Traversal Algorithms
  - Depth-first Search
  - Breadth-first Search
- Applications
  - Spanning Tree
  - Shortest Paths
  - Activity Networks
  - Critical Activities

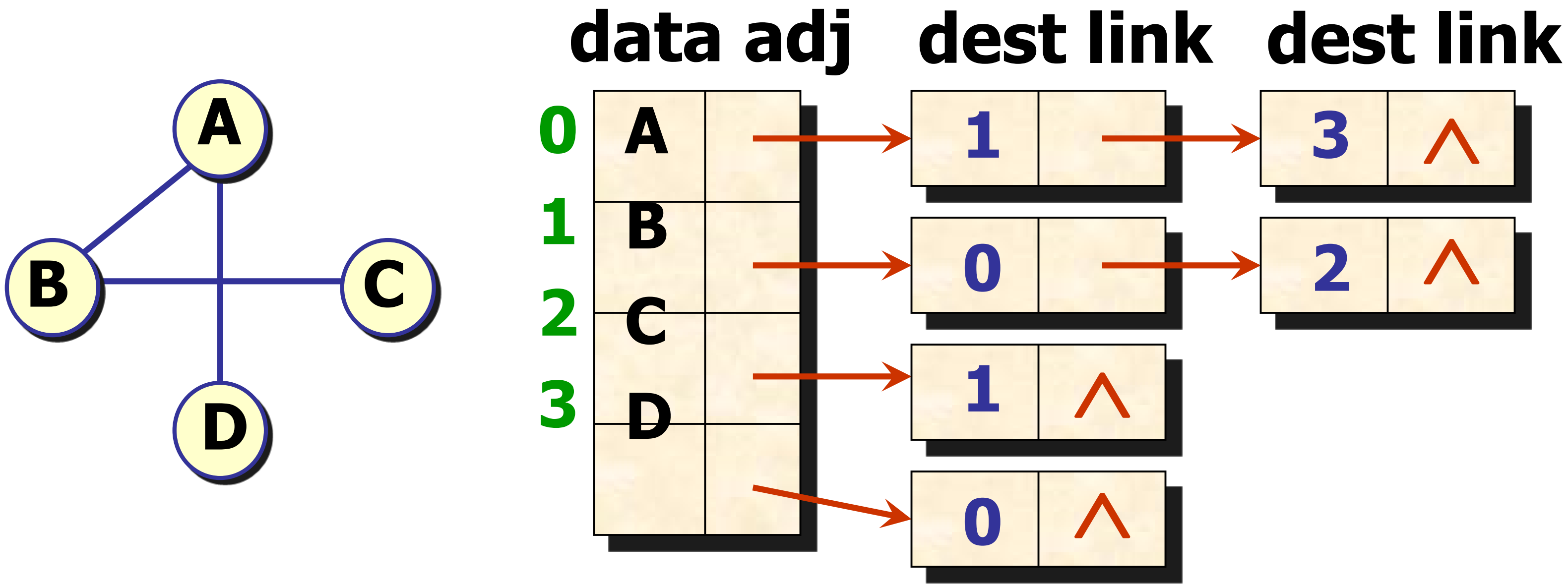
Adjacency Lists

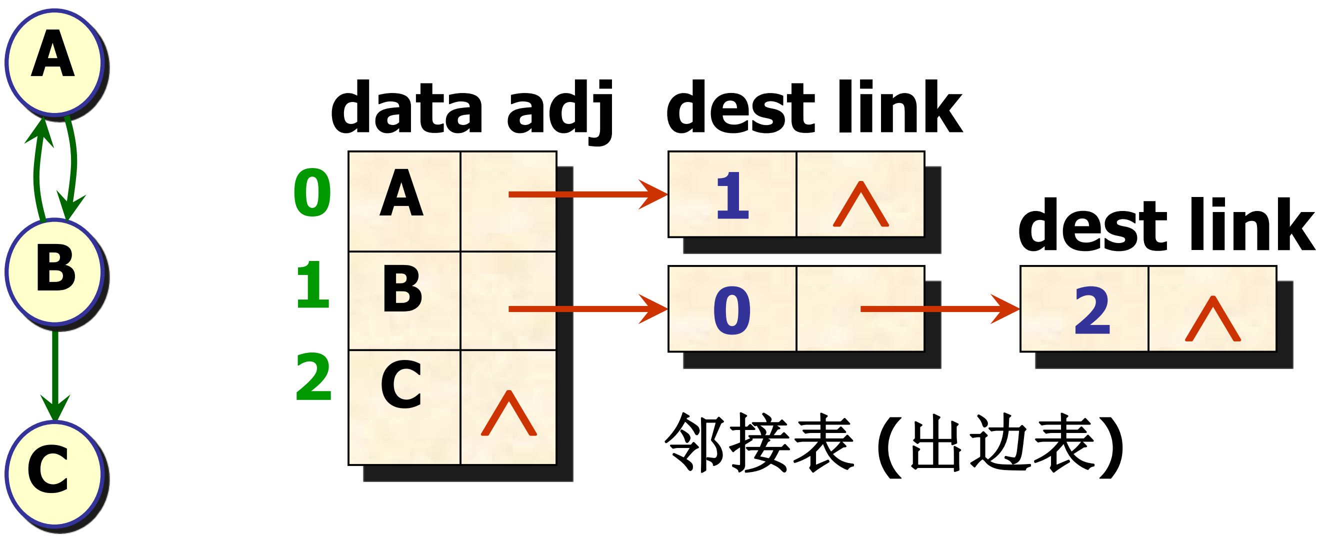
- The n rows of the adjacency matrix are represented as n chains.
- The nodes in chain i represent the vertices adjacent from i.



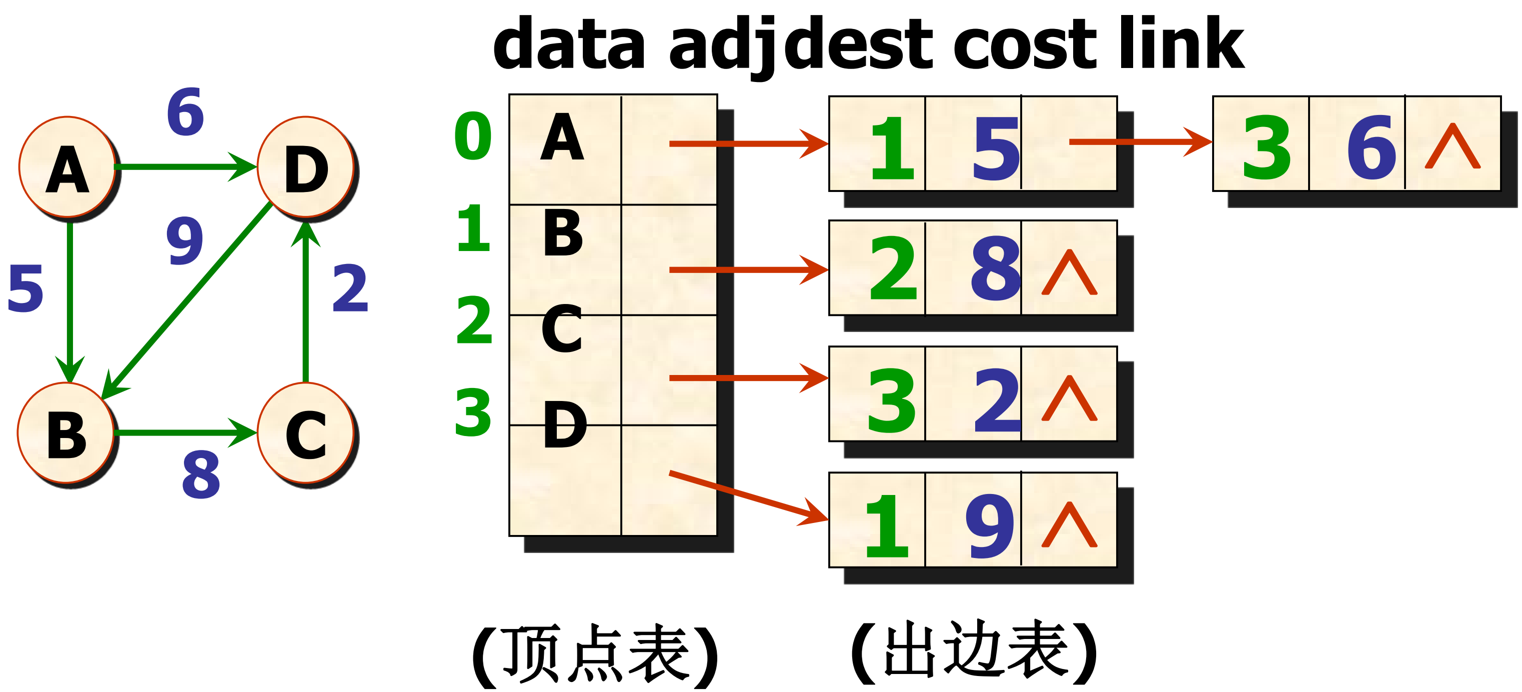
- The data field of a chain node store the index of an adjacent vertex.
- The vertices in each chain are not required to be ordered.
- An array adjLists is used for accessing any chain in O(1).

```
class LinkedGraph
{
public:
    LinkedGraph (const int vertices): e(0)
    {
       if (vertices < 1) throw “Number of vertices must be > 0”;
       n = vertices;
       adjLists = new Chain<int>[n];
    };
private:
    Chain<int>* adjLists;
    int n;
    int e;
};
```


A
B
C
D
data adj
dest link
dest link
0 A
1 B
2 C
3 D
1
3 ^
0
2 ^
1 ^
0 ^



邻接表 (出边表)


data adjdest cost link
A
B
C
D
6
5
9
2
8
0
1
2
3
1 5
3 6 ^
2 8 ^
3 2 ^
1 9 ^
(顶点表)
(出边表)

-- For an undirected graph with n vertices and e edges, this representation requires an array of size n and 2e chain nodes.

-- Now it is possible to determine the total number of edges in G in O(e+n).

-- In case of digraph, the in-degree of a vertex is a little more complex to determine --- use a vector c[n], when traverse from i to j, c[j]++.