SCC120 Fundamentals of Computer Science
Unit 6: Graphs (Representations)

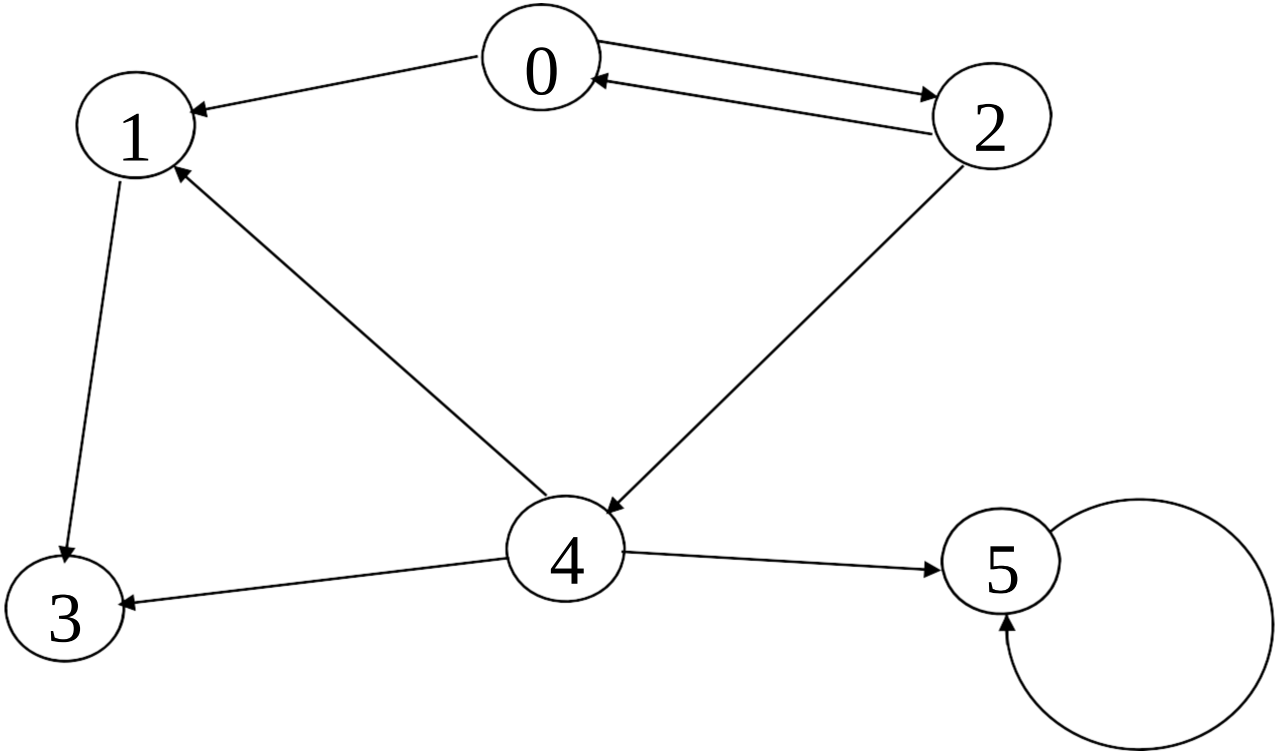
Jidong Yuan
yuanjd@bjtu.edu.cn

Directed Graph Representations

- How to represent a directed graph?
- Five ways:
  - (1) Adjacency Matrix
  - (2) Node Array with Arc Chains
  - (3) Node Chain with Arc Chains
  - (4) Cell-per-Node Model
  - (5) String-based Representations

- Can be adapted to non-directed graphs
- Why look at these representations?
  - For a specific problem, there may be some representation which is more suitable or efficient

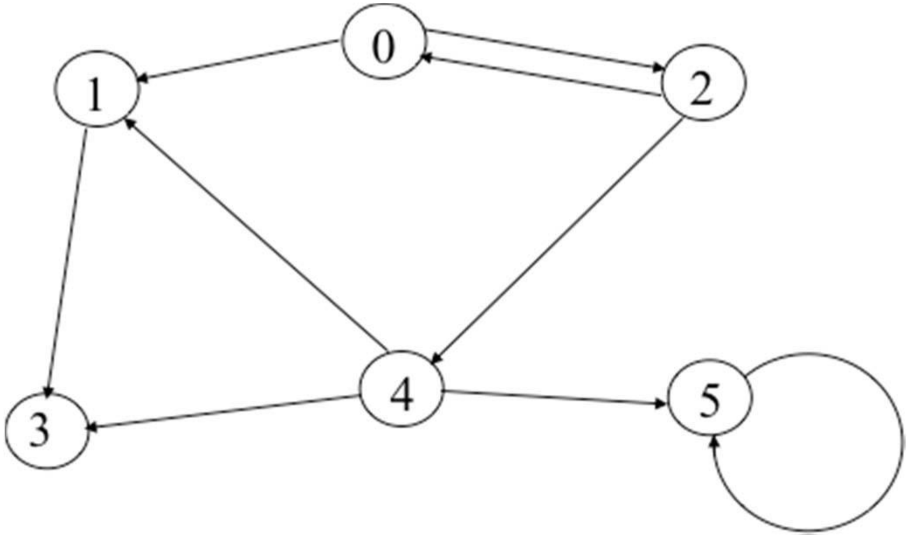
Example of a Graph



(1) An Adjacency Matrix

- This method uses a square array of binary (that is, boolean) values
  - Each element indicates the presence or absence of an arc
  - Alternatively we can hold the associated cost of the arc in the array (as long as there is a distinct value which means “no arc”)
  - Node values would have to be stored in a separate array

An Adjacency Matrix



	0	1	2	3	4	5	
0	O	1	1	O	O	O	
1	O	O	O	1	O	O	
2	1	O	O	O	1	O	
3	O	O	O	O	O	O	
4	O	1	O	1	O	1	
5	O	O	O	O	O	1	

An Adjacency Matrix

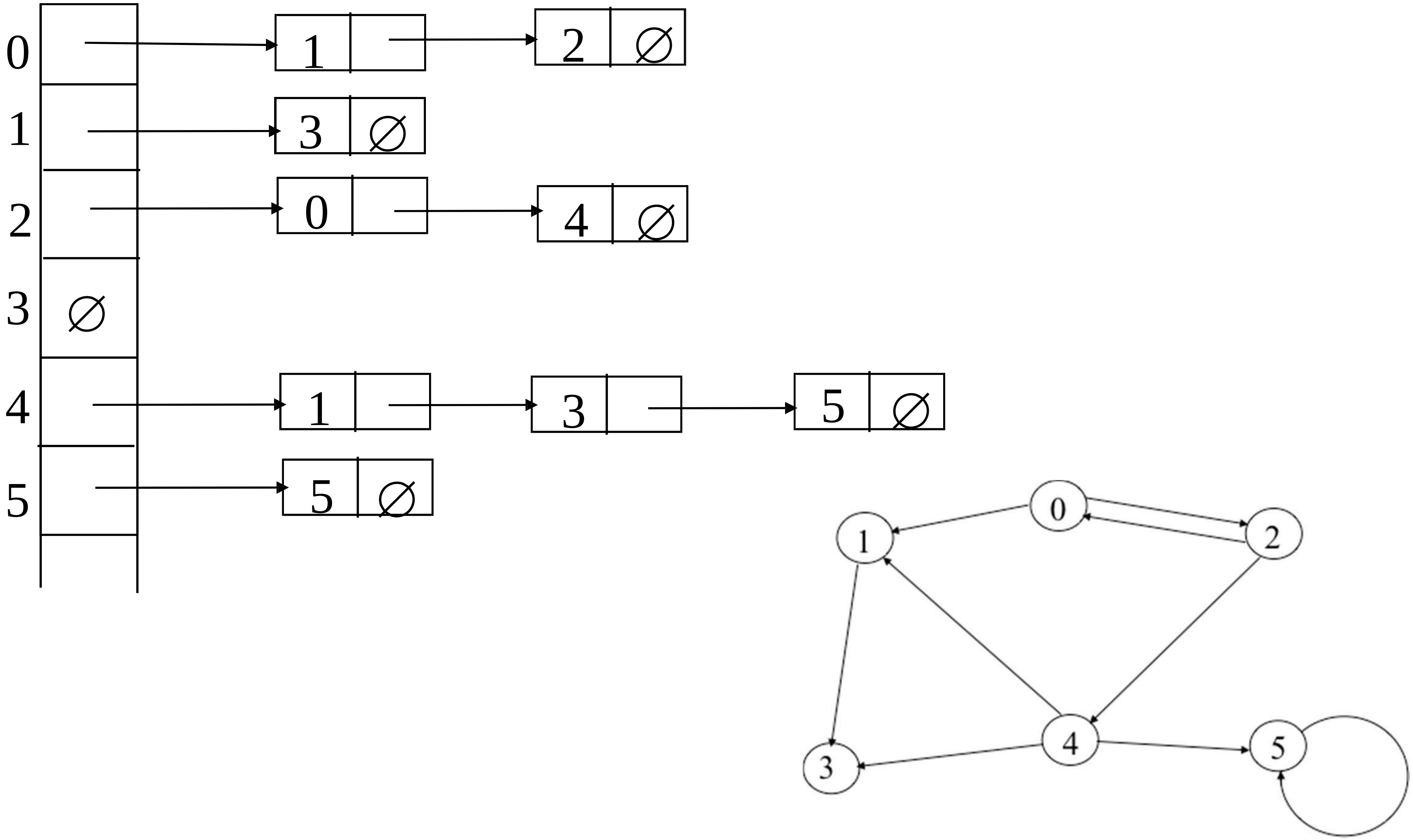
- The maximum weight (the number of nodes) w is limited by the size of the 2D array (-)
- The space required is w^2, so a lot of space is need if the weight is high (-)
- The maximum out-degree is not limited (except by w) (+)
- Node scanning is easy, using a “for” loop (+)

```
for (int i=0; i<w; i++) ...
```

(2) A Node Array with Arc Chains

- We have an array
  - With an element for each node
  - Pointing to a chain of elements, one for each arc starting from this node
  - Each chain element contains
    - the element of the in-node of the arc
    - a pointer to the next arc (“null” for the last one)
  - If arc values are needed, they can go into extra fields in the arc elements
- Node values could be stored in a separate array

A Node Array with Arc Chains



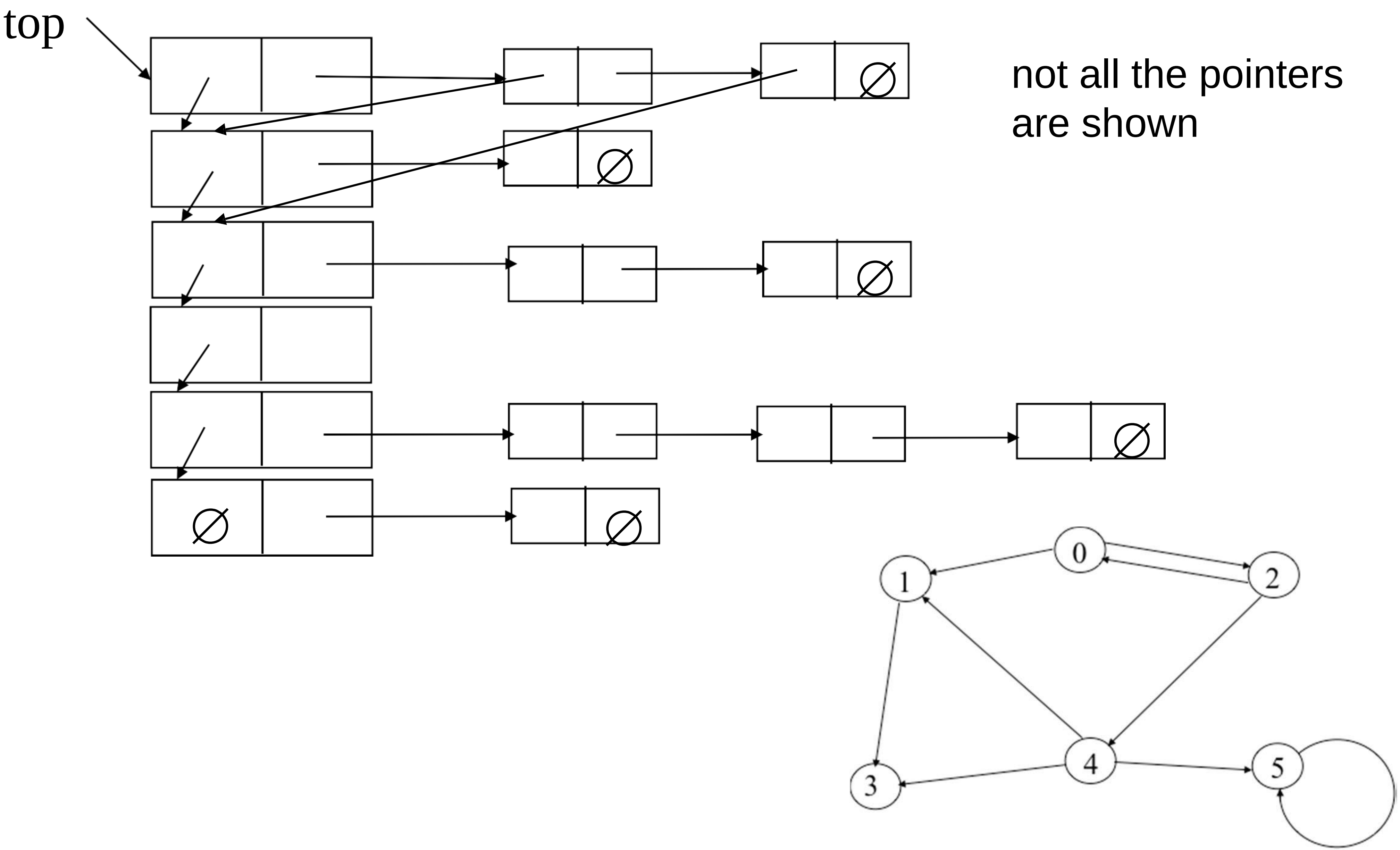
A Node Array with Arc Chains

- The maximum weight w is limited by the size of the array (-)
- The space needed is generally less than with method 1 (adjacency matrix), unless the density is high (+)
- The maximum out-degree is not limited, as each chain can be as long as needed (+)
- Node scanning is easy, using a “for” loop (+)

(3) A Node Chain with Arc Chains

- A Chain of chains: this is similar to method 2 but replacing the node array with a chain (linked list) of nodes
- We do not have node names here, but we have cell pointers
- If arc values are needed, we can have extra fields in the arc cells
- If node values are needed, we can have extra fields in the node cells

A Node Chain with Arc Chains



not all the pointers are shown

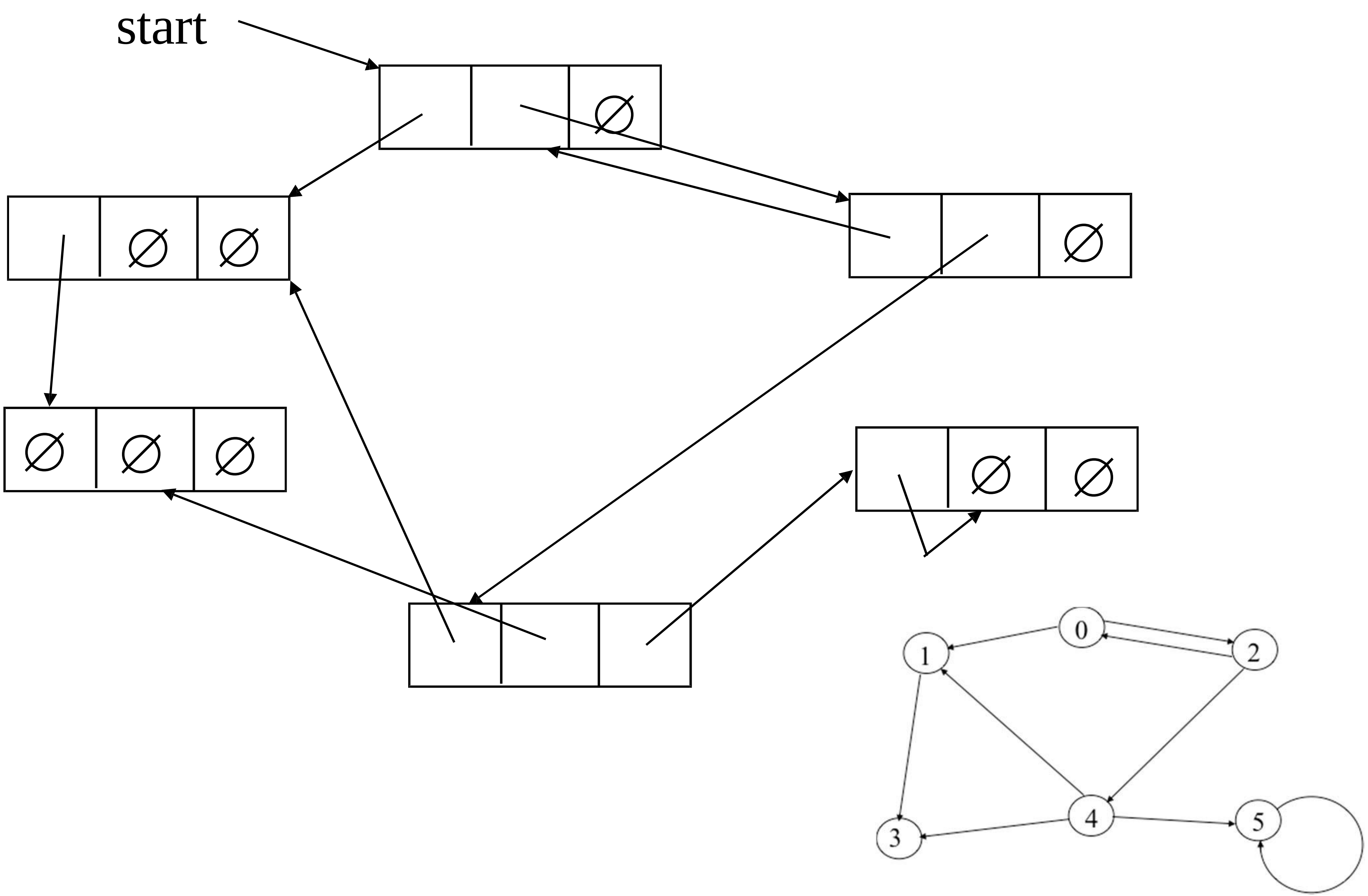
A Node Chain with Arc Chains

- The maximum weight w is not limited (+), though deletion of nodes requires some care
- The representation is not as compact as method 2 (-)
- The maximum out-degree is not limited (+)
- Node scanning is easy, using “while” loop (+)

(4) The Cell-per-Node Model

- This representation is appealing because it maps directly onto the corresponding node-and-arc drawing
- If arc values are needed, an extra field must be included for each pointer field
- If node values are needed, we can have extra fields in the node cells

The Cell-per-Node Model



The Cell-per-Node Model

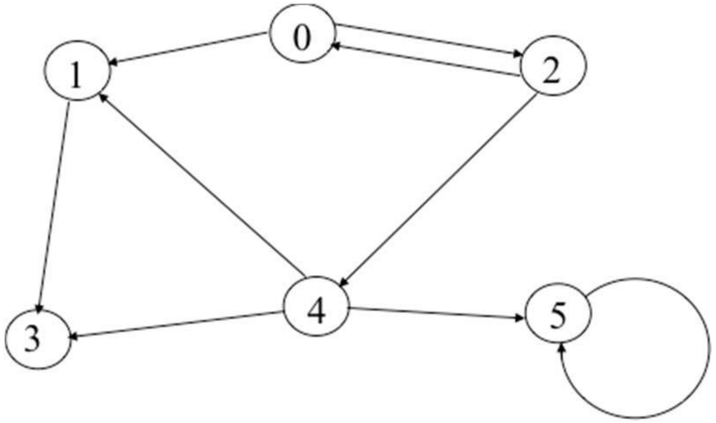
- The maximum weight w is not limited (+), though deletion of nodes requires some care
- It is fairly compact (+)
- The maximum out-degree is limited by the number of pointer fields in each cell (-)
- Nodes generally cannot be scanned, since there may be no cell from which all the others can be reached (-)

The Cell-per-Node Model

- This lack of a method to scan all the nodes is often a serious drawback
- A solution would be to add an extra pointer field to every node cell
  - and to use this to chain all the node cells together in some arbitrary order
  - this chain could then be used for scanning the nodes

(5) String-based Representations 1

- A simple method is simply to list the arcs which are present (with suitable punctuation)
  - 0,1; 0,2; 1,3; 2,0; 2,4; 4,1; 4,3; 4,5; 5,5;
- This doesn’t store the node and arc values (if any)
- An advantage of string-based representations is that they are stored, or transmitted from system to system, without conversion



String-based Representations 1

- Isolated nodes are not represented at all, since they do not appear in any arc specification (-)
- Node scanning is not very efficient (-)

String-based Representations 2

- A better alternative is to provide a separate list of nodes first:
  - 0, 1, 2, 3, 4, 5;   0,1; 0,2; 1,3; 2,0; 2,4; 4,1; 4,3; 4,5; 5,5;
  - Apart from differences in formatting, this is similar   to: G = ({a,b,c}, {(a,a), (a,c), (b,a), (c,a), (c,b)})
- If arc values are needed, they could be inserted into the arc list
- If node values are needed, they could be inserted into the node list

String-based Representations 2

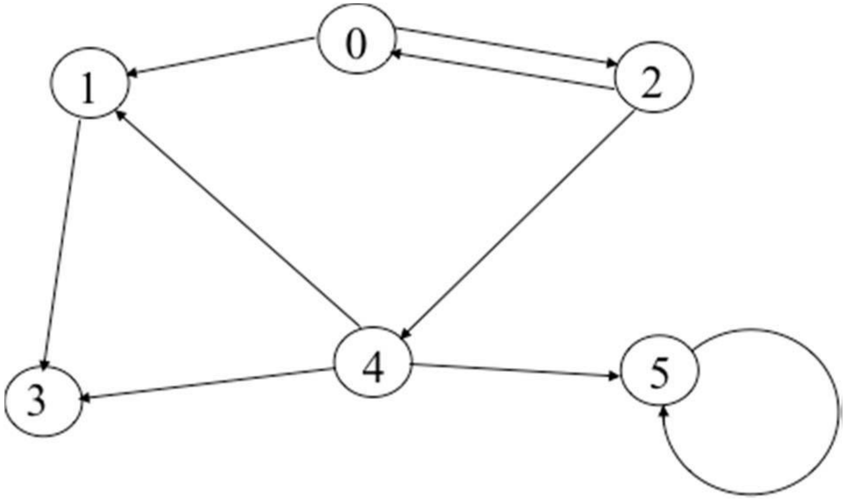
- The representation is quite compact (+)
- Out-degrees are not limited (+)

String-based Representations 3

- A final method lists each node followed by a list of the nodes it "points to":

  0:1,2; 1:3; 2:0,4; 3; 4:1,3,5; 5:5

- This is quite compact (+)
- It may be faster for processing than the other string-based representations (+)



String-based Representations 3

- This can easily be transformed to/from Method 2 (node array with arc chains)
- Method 1 (adjacency matrix) is also straightforward
- Method 3 (node chain with arc chains) is awkward as it does not use node numbers
- Method 4 (cell-per-node model) is even worse

Choosing a Representation

- The choice of a method for representing a graph may depend on various factors, for example:
  - – The nature of the graph (e.g. its weight, density, out-degree)
  - – The required operations (e.g. will the weight keep changing?)
  - – Whether we need to scan the nodes (visit them all quickly)

SCC120 ADT (weeks 7-12)

- Week 7 Abstractions; Set
  Stack
- Week 8 Queues
  Priority Queues
- Week 9-10 Graphs (Terminology)
  Graphs (Traversals)
  Graphs (Representations)
- Week 11
- Week 12