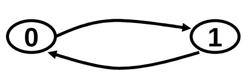
1.00 Lecture 29

Graphs
Shortest Path Algorithms

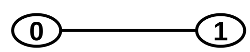
Reading for next time: Big Java 15.1-15.4

Graphs and Networks

- Graph G(N, A) is two sets:
  - N is the set of nodes 0..n-1
  - A is the set of arcs, or pairs of nodes ij, i !=j
- Graphs can be directed or undirected



Directed
<0, 1>
<1, 0>

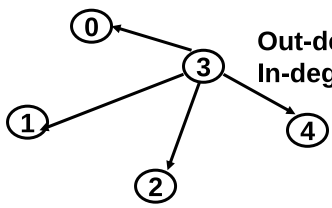


Undirected
(0, 1)

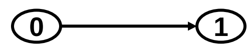
- A network is a graph with a cost associated with each arc in A.
  - We generally don’t permit negative arc costs.
    - Negative cycles are problematic
- There are two kinds of networks in this world…
  - Electrical and its kin…and traffic and its kin…

Networks

- **In an undirected network:**
  - Node i is adjacent to node j if arc ij exists
  - Degree of node is number of adjacent nodes
- **In a directed network:**
  - Node i is adjacent-to node j if arc ij exists
  - Node i is adjacent-from node j if arc ji exists
  - In-degree of node is number of adjacent-from nodes
  - Out-degree of node is number of adjacent-to nodes



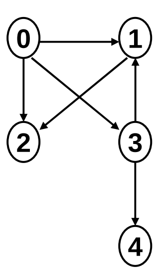
Out-degree of node 3= 4
In-degree of node 3= 0

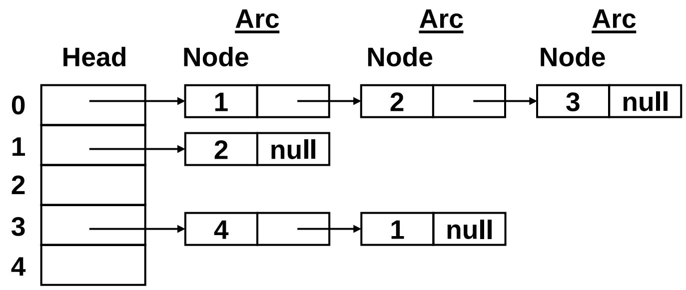


0 adjacent-to 1
1 adjacent-from 0

List representation of graphs

- **Adjacency list of graph is n lists, one for each node i**
  - Adjacency list contains node(s) adjacent from i
  - Variation holds nodes adjacent-to i

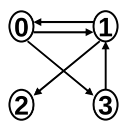


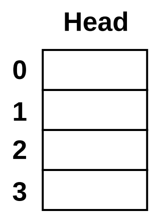


Head holds reference from each node i to the adjacent node list. Arc order arbitrary

Exercise

- Draw the list representation for the following graph with 4 nodes and 5 arcs:

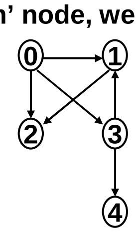




Array representation of graphs

- If no insertion or deletion of nodes and arcs is to be done (or is rare), we dispense with the links and list.
  - If we read the arcs from input and sort by 'from' node, we get:

From	To	Cost	(Arc number)
0	1	43	0
0	2	52	1
0	3	94	2
1	2	22	3
3	4	71	4
3	1	37	5



  - Notice the from node repeats when out-degree > 1

- We recast this structure as arrays H, To, Cost:

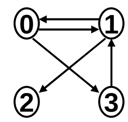
(Node)	H
0	0
1	3
2	4
3	4
4	6
5	6 (sentinel)

(Arc)	To	Cost
0	1	43
1	2	52
2	3	94
3	2	22
4	4	71
5	1	37

Exercise: Array Representation

– Fill in the array representation for the graph (4 nodes, 5 arcs):

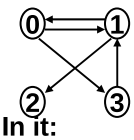
From	To	(Arc number)
		0
		1
		2
		3
		4



(Node)	H
0	
1	
2	
3	
4	
(sentinel)	(last arc+1)

(Arc)	To
0	
1	
2	
3	
4	

Exercise: SmallGraph



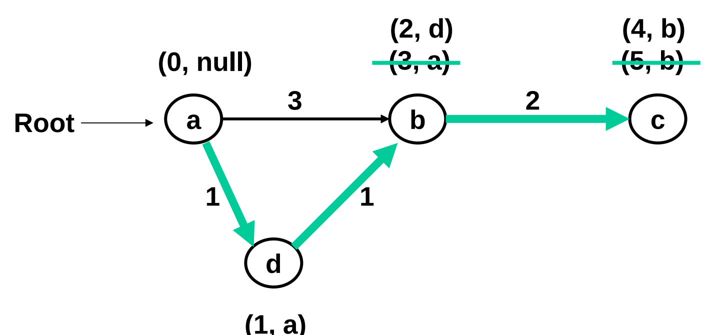
- **Write a program to create and print the small graph you've just modeled**
  – Use the array representation
  – Create class SmallGraph with just a main() method. In it:
    - Create array H (use the {…} syntax for it)
    - Create array To (also use the {…} syntax)
    - Create a variable= number of actual nodes
      – Don't count the sentinel!
    - Print out the arcs in the graph
      – Loop through all the actual nodes (use the H array)
        » Loop through the arcs out of each node (use the To array) and print them
  – The main() method is about 6 lines of code

Shortest paths in networks

- Shortest path algorithm:
  - Builds shortest path tree
  - From a root node
  - To all other nodes in the network.
- All shortest path algorithms are labeling algorithms
  - Labeling is process of finding:
    - Cost from root at each node (its label), and
    - Predecessor node on path from root to node
- Algorithm needs two data structures:
  - Find arcs out of each node
    - Array-based representation of graph itself
  - Keep track of candidate nodes to add to shortest path tree
    - Candidate list (queue) of nodes as they are:
      - Discovered and/or
      - Revisited

Example

(Label, predecessor)



Thick arcs are
shortest path tree with
distances and predecessors

Types of shortest path algorithms

- Label setting. If arc is added to shortest path tree, it is permanent.
  - Dijkstra (1959) is standard label setting algorithm.
  - Fastest for dense networks with average out-degree ~> 30
  - Requires heap (or sorted) arcs, which is slowest step
- Label correcting. If arc is added to tree, it may be altered later if better path if found.
  - Series of algorithms, each faster, depending on how candidate list is managed. Fastest when out-degree ~< 30
    - Bellman-Ford (1958). New node discovered always put on back of candidate list and next node taken from front of list. (Queue)
    - D'Esopo-Pape (1974). New node put on front of candidate list if it has been on list before, otherwise on back ('Sharp labels')
    - Bertsekas (1992). New node put on front of candidate list if its label smaller than current front node, otherwise on back
    - Hao-Kocur (1992). New node is put on front of list if it has been on list before. Otherwise it is put on back of list if label > front node and on front of list if smaller. ('Sharp labels')
- Previous example was label correcting
  - Label setting requires looking at shortest arc at every step

Computational results

CPU times (in milliseconds) on road networks
(HP9000-720 workstation, 1992)

Nodes	*Arcs*	*Visit*	*Dijkstra*	*Bellman*	*D'Esopo*	*Bertsekas*	*Hao-Kocur*
5199	14642	13	98	42	37	21	19
28917	64844	96	1192	590	125	144	104
115812	250808	459	9007	5644	619	789	497
119995	271562	488	13352	7651	708	1183	596
187152	410338	779	27483	15067	1184	1713	926

Times are 300x faster today (hardware- Moore's Law).
Also, slow implementations run 100x slower (lists, sorts, etc.)

Worst case, average performance

Algorithm	Worst case	Average case
Label-correcting	$O(2^n)$	~$O(n)$
Label-setting	$O(n^2)$ with sorting $O(a \lg n)$ with heap	$O(a \lg n)$ with heap

(a= no of arcs, n= no of nodes)

It takes a real sense of humor to use an $O(2^n)$ algorithm in 'hard real-time' applications in telecom, but it works! (Boss went crazy the first time we proposed it)

Label correctors with an appropriate candidate list data structure in fact make very few corrections and run fast

Tree (D,P) and list (CL) arrays

Array	Definition	Description
D	Distance (output)	Current best distance from root to node i
P	Predecessor (output)	Predecessor of node it in shortest path (so far) from root to node i
CL	Candidate list (internal)	List of nodes that are eligible to be added to the growing shortest path tree. CL[i]= NEVER_ON_CL if node has never been on CL ON_CL_BEFORE if node has been on CL before j if node i is now on CL and j next END OF LIST if node is last on CL

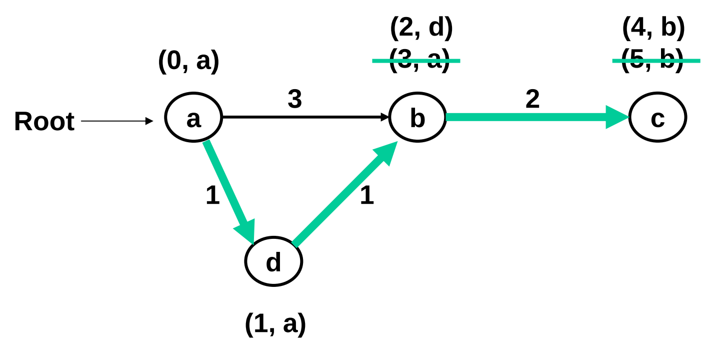
<u>6 1-D arrays for input, output, data structures:</u>

Graph input and data structure: Head, To, Dist
Tree output and data structure: D, P
Candidate list to control algorithm: CL

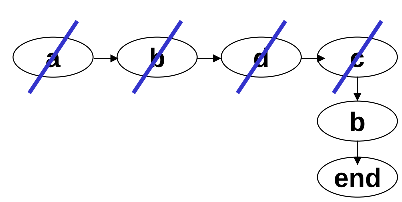
Method

- **Initialize:**
  - **P: Shortest path tree= {root}**
  - **D: Distance from root to all nodes= "infinity"**
  - **CL: Candidate list= {root}, at end of list**
- **At each step:**
  - **A node i is removed from front of CL**
  - **For each arc ij leaving node i where the distance from the root to node j is shortened by going via node i, add node j to CL:**
    - **If CL[j] == ON_CL_BEFORE, add j to front of CL**
    - **If CL[j] == NEVER_ON_CL:**
      - **If D[j] < D[front node on CL], add j to front of CL**
      - **Else add j to end of CL**
    - **If CL[j] > 0, j is now on CL. Do nothing.**
    - **If CL[j] == END_OF_LIST, terminate algorithm**

Example



i	*P*	*D*	*CL*
a	a	0	ON_BEF
b	d	2	END
c	b	4	ON_BEF
d	a	1	ON_BEF



Code, p.1

```
public class ShortPathTest {
    public static void main(String[] args) {
        Graph g= new Graph();
        g.shortHK();   }  }

public class Graph {
    private int nodes;
    private int arcs;
    private int[] head;          // Nodes + 1 (sentinel) slots
    private int[] to;            // Arcs
    private int[] dist;          // Arcs
    private int root;
    private int[] P;             // Predecessor
    private int[] D;             // Label

    Graph() {
    // Set nodes=4, arcs=4, root=1; head, to,dist as in example
    // In general, read network from file or generate on fly
    }
```

Code, p.2

```
public void shortHK() {
       // Constants—could be in Graph as static
       final int MAX_COST= Integer.MAX_VALUE/2;
       final int EMPTY= Short.MIN_VALUE;
       final int NEVER_ON_CL= -1;
       final int ON_CL_BEFORE= -2;
       final int NOT_ON_CL= -3;
       final int END_OF_CL= Integer.MAX_VALUE;
       D= new int[nodes];
       P= new int[nodes];
       int[] CL= new int[nodes];
       // Initialize
       for (int i=0; i < nodes; i++) {
         D[i]= MAX_COST;
         P[i]= EMPTY;
         CL[i]= NEVER_ON_CL; }
       D[root]= 0;
       CL[root]=  END_OF_CL;
       int lastOnList= root;
       int firstNode= root;
```

```
do {
  int Dfirst= D[firstNode];
  for(int link=head[firstNode]; link<head[firstNode+1]; link++){
    int outNode= to[link];          // Loop thru arcs out of node
    int DoutNode= Dfirst + dist[link];
    if (DoutNode < D[outNode]) {  // Do something only if impvt
      P[outNode]= firstNode;
      D[outNode]= DoutNode;
      int CLoutNode= CL[outNode];
      if (CLoutNode==NEVER_ON_CL || CLoutNode==ON_CL_BEFORE) {
        int CLfirstNode= CL[firstNode];
        if (CLfirstNode != END_OF_CL &&        // Front of CL
          (CLoutNode==ON_CL_BEFORE || DoutNode<D[CLfirstNode])){
          CL[outNode]= CLfirstNode;
          CL[firstNode]= outNode; }
        else {                                 // Back of CL
          CL[lastOnList]= outNode;
          lastOnList= outNode;
          CL[outNode]= END_OF_CL;  }  }  }  } // End for loop
    int nextCL= CL[firstNode];                // Go to next node
    CL[firstNode]= ON_CL_BEFORE;
    firstNode= nextCL;
        } while (firstNode < END_OF_CL); }  } // End do loop
```

Manage CL

Summary

- Shortest path algorithm
  - 22 lines of code, after initialization
    - Down from 200+ lines 25 years ago for d'Esopo-Pape
  - One addition operation, otherwise only increment, compare
  - 3 data structures (queue-as-list, network, tree) as arrays
    - They control the very simple algorithm very efficiently
  - Linked list would be too expensive
    - Memory allocation in small chunks is very slow
  - Separate data structures and algorithm would be too expensive
    - Method call overhead noticeable in real time algorithms
  - One preprocessing trick used by Hao-Kocur:
    - Sort arcs out of node by distance. Get a bit of 'Dijkstra effect'
  - This is the opposite extreme to the typical Java style that emphasizes flexibility, reuse, generality
  - This is somewhat typical of embedded systems, real-time algorithms
    - Nothing is truly typical, because all are tuned, use special cases

Applications, other graphs

- Shortest path applications to flows in networks:
  - Traffic, telecom, data, water, task scheduling, …
  - Hao-Kocur used in telecom software, optical routers, transport software, …
- Other network methods:
  - Spanning trees, min cost flows, matching, …
  - Many matrix problems can be cast as networks
    - Graph is the matrix; tree is the basis/solution
    - All integer variables, usually avoids precision hassles
  - Combinatorial or decision problems (and games)
    - Use graphs directly, and use other graph algorithms as subproblems in their solution methods