EXPLORING GRAPHS

By
Prof. S. J. Soni
Assistant Professor
Computer Engg. Department
SPCE, Visnagar

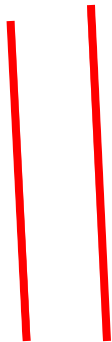
Graphs and Games: An Introduction

- Consider the game which is one of the many variants of **Nim**, also known as the **Marienbad Game.**
- Initially there is a heap of matches on the table between two players. [At least two matches]
- The first player may remove as many matches as he likes, except that he must take at least one and he must leave at least one.
- Thereafter, each player in turn must remove at least one match and at most twice the number of matches his opponent just took.
- The Player who removes the last match wins. There are no draws.

Graphs and Games: An Introduction

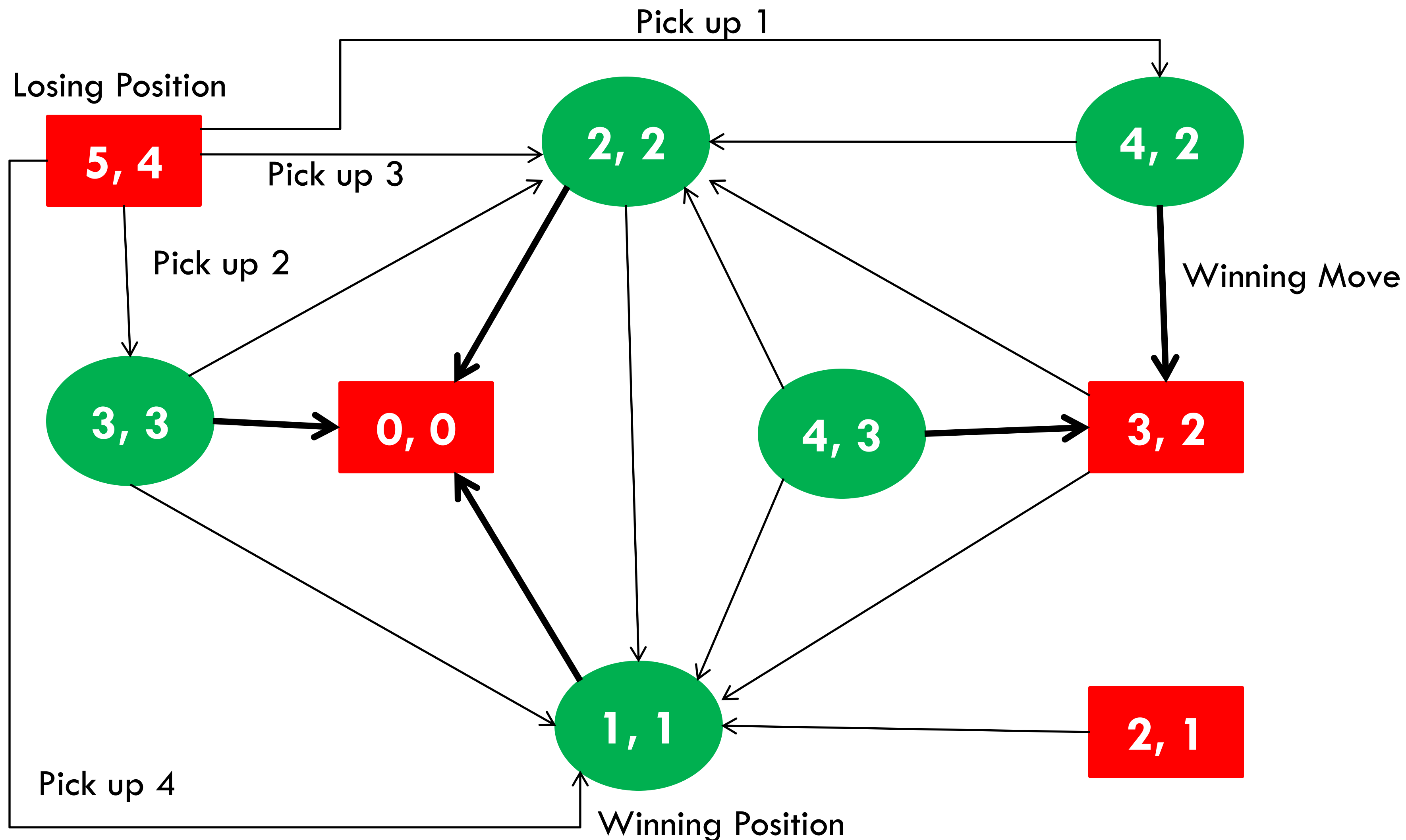
- Suppose that at some stage in this game you find yourself in front of a pile of five matches.
- Your opponent has just picked up two matches, and now it is your turn to play.
- You may take one, two, three and four matches: however you may not take all five.
- What should you do ?

Opponent



You ?????

Graphs and Games: An Introduction



Graphs and Games: An Introduction

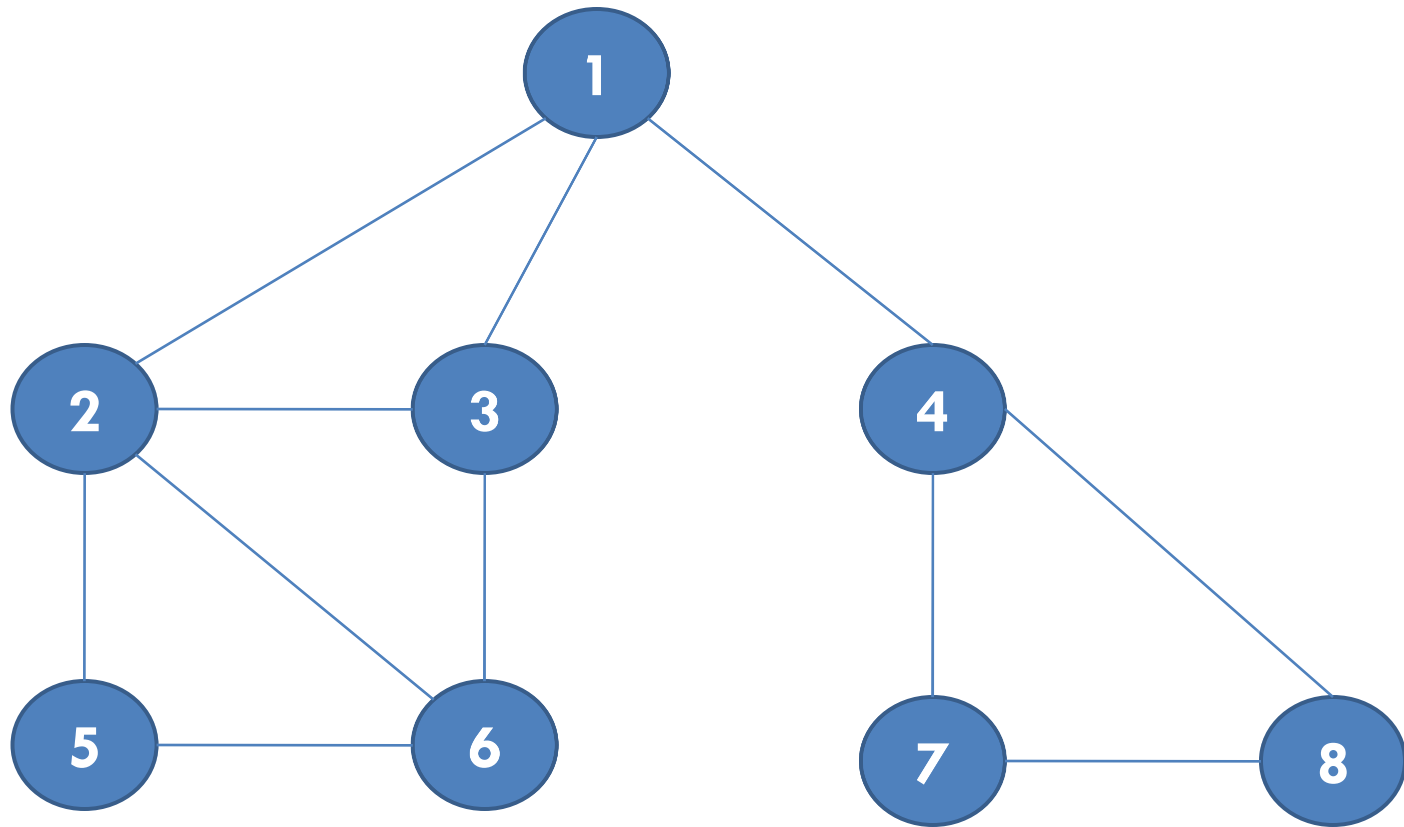
- The **recursive algorithm** can be used to determine whether a position is winning or losing.
- But the main drawback is, it calculates the same value over and over.
- So it can be solved using **Dynamic Programming** algorithm.

Traversing Trees

- There are three techniques often used.
  - Preorder, Postorder and Inorder

- **Preconditioning**
  - If we have to solve several similar instances of the same problem, it may be worthwhile to invest some time in calculating auxiliary results that can thereafter be used to speedup the solution of each instance. This is preconditioning.

Depth First Search: Undirected Graphs

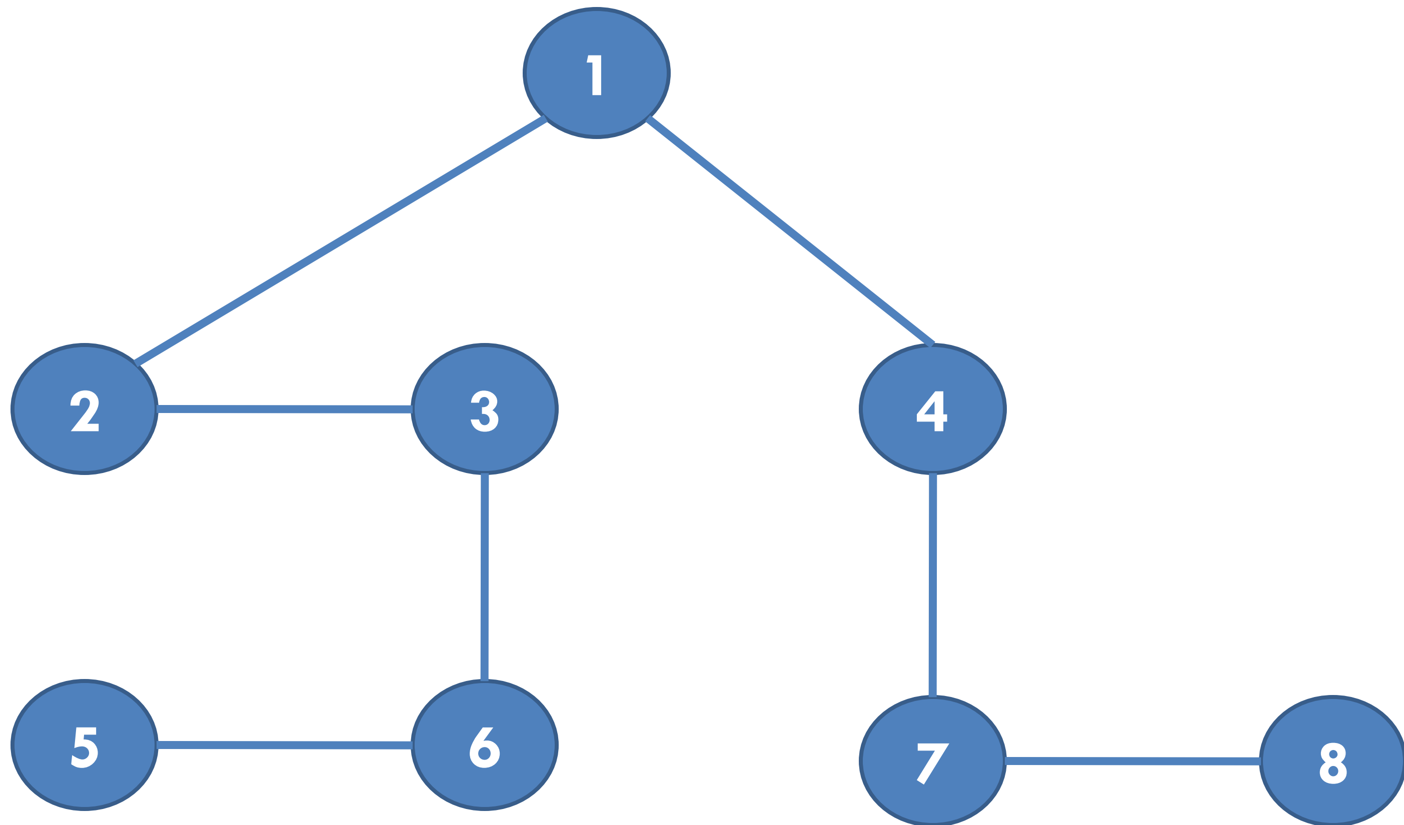


Depth First Search: Undirected Graphs

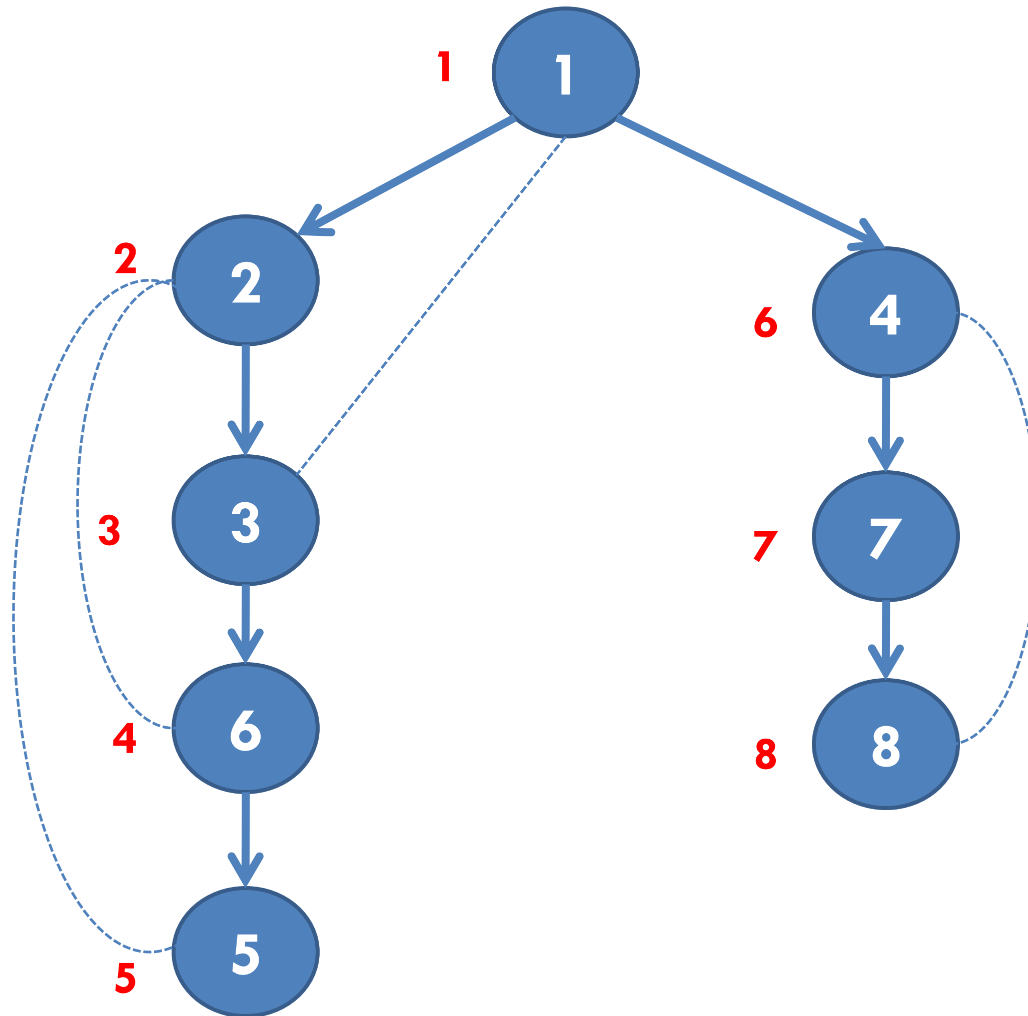
1.	dfs(1)	initial call
2.	dfs(2)	recursive call
3.	dfs(3)	recursive call
4.	dfs(6)	recursive call
5.	dfs(5)	recursive call, progress blocked
6.	dfs(4)	neighbor of node 1 not visited
7.	dfs(7)	recursive call
8.	dfs(8)	recursive call, progress blocked

9. There are no more blocks to visit

Depth First Search: Undirected Graphs



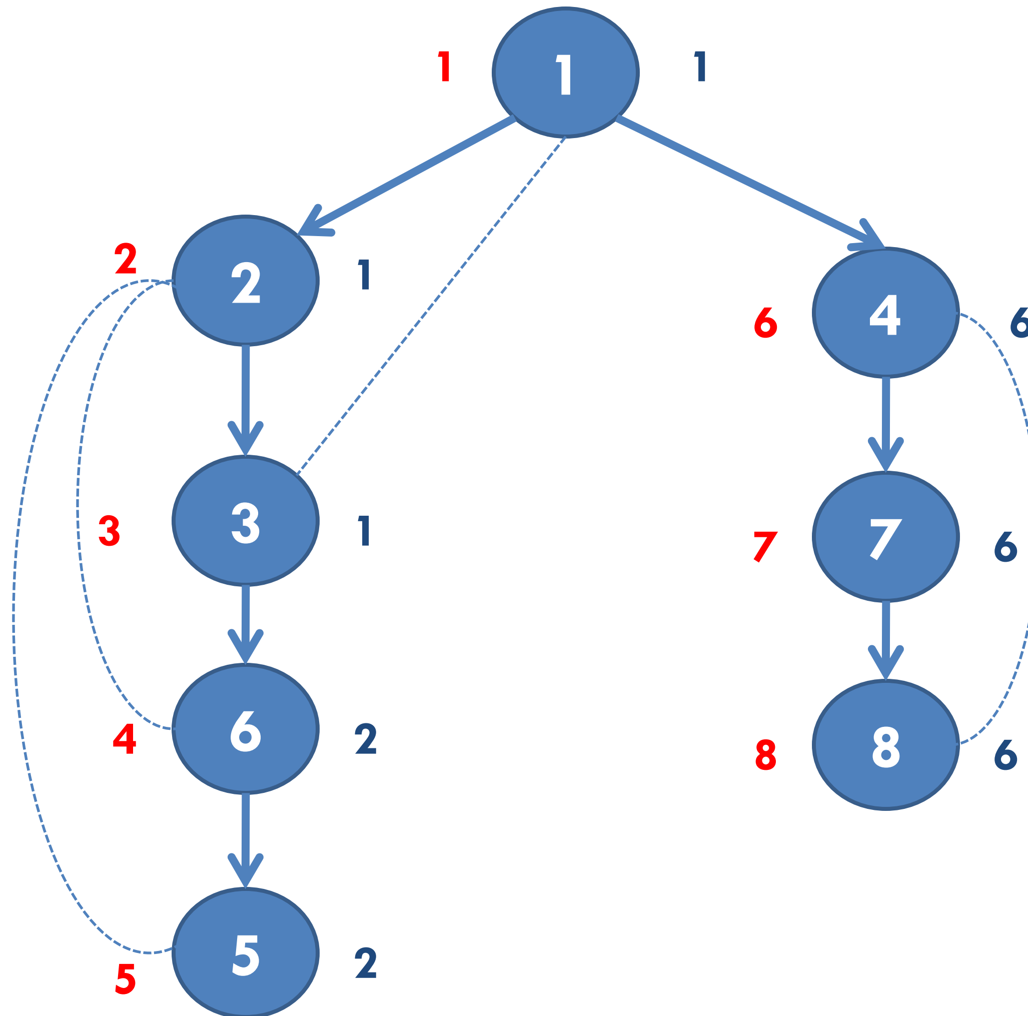
Depth First Search: Undirected Graphs



Articulation Points

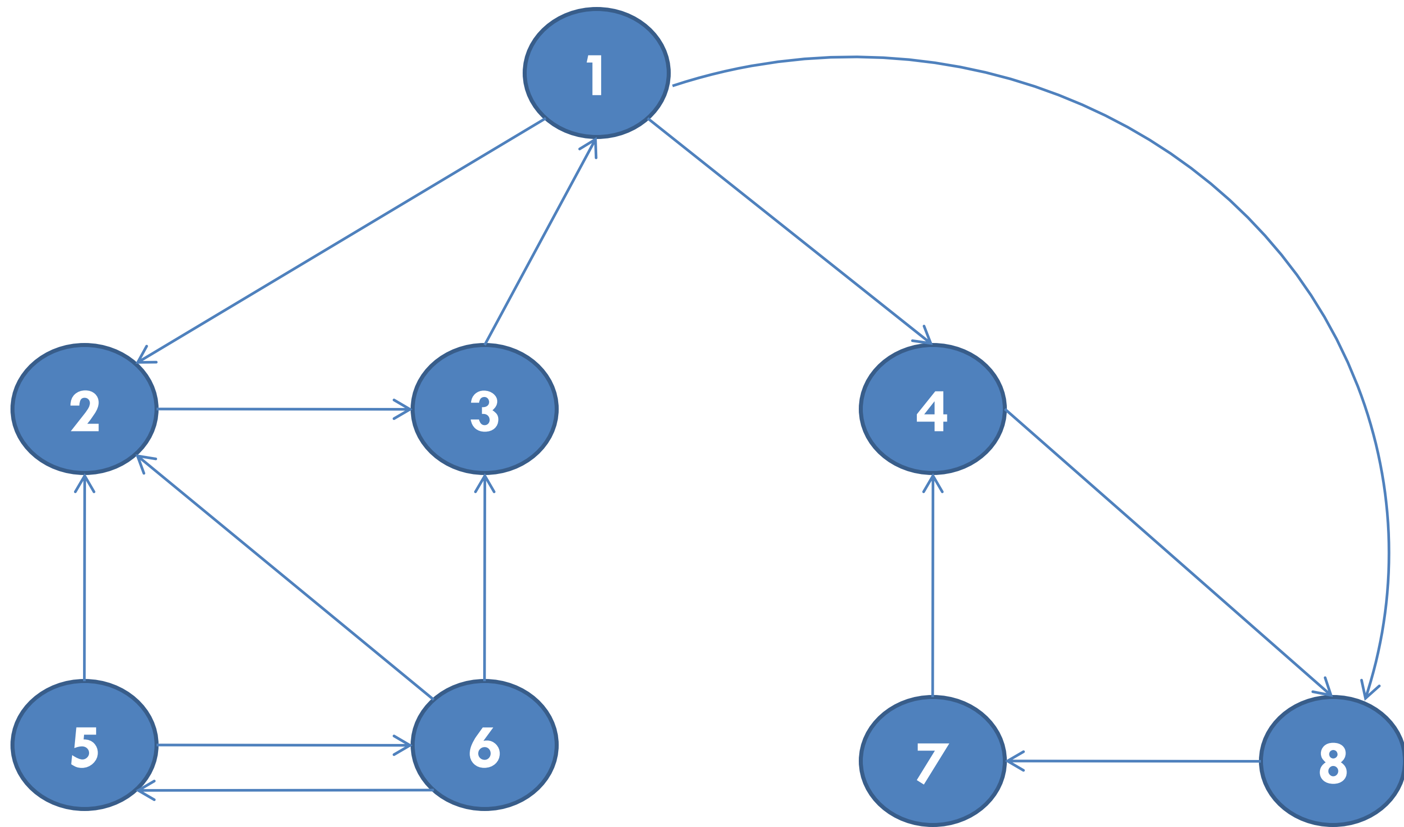
- A node v of a connected graph is an articulation point if the subgraph obtained by deleting v and all the edges incident on v is no longer connected.
- For example, node 1 is an articulation point of the previously discussed graph; if we delete it, there remain two connected components {2,3,5,6} and {4,7,8}.
- A graph G is biconnected (or unarticulated) if it is connected and has no articulation point.

Finding Articulation Points



prenum on the left, ***highest*** on the right.

Depth First Search: Directed Graphs

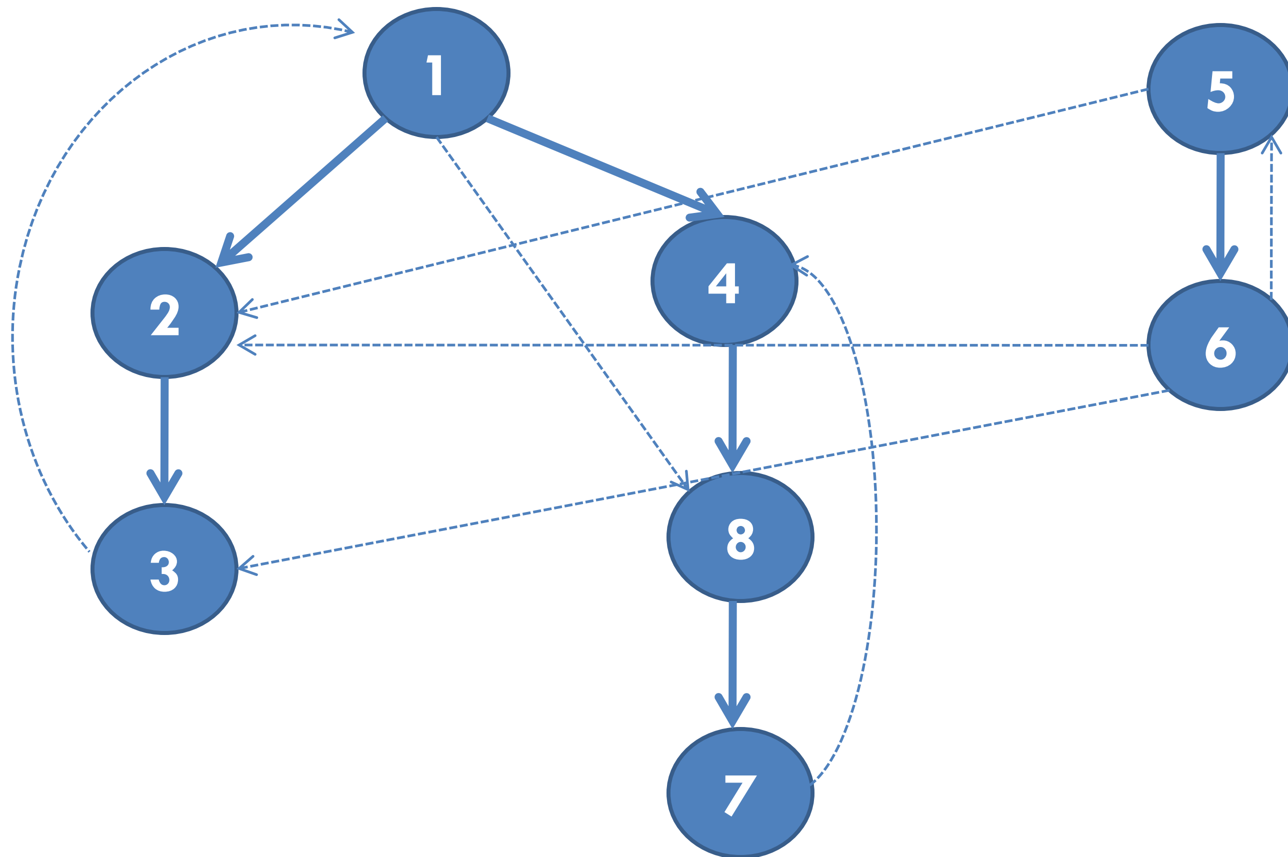


Depth First Search: Directed Graphs

1. dfs(1)	initial call
2. dfs(2)	recursive call
3. dfs(3)	recursive call, progress blocked
4. dfs(4)	neighbor of node 1 not visited
5. dfs(8)	recursive call
6. dfs(7)	recursive call, progress blocked
7. dfs(5)	new starting point
8. dfs(6)	recursive call, progress blocked

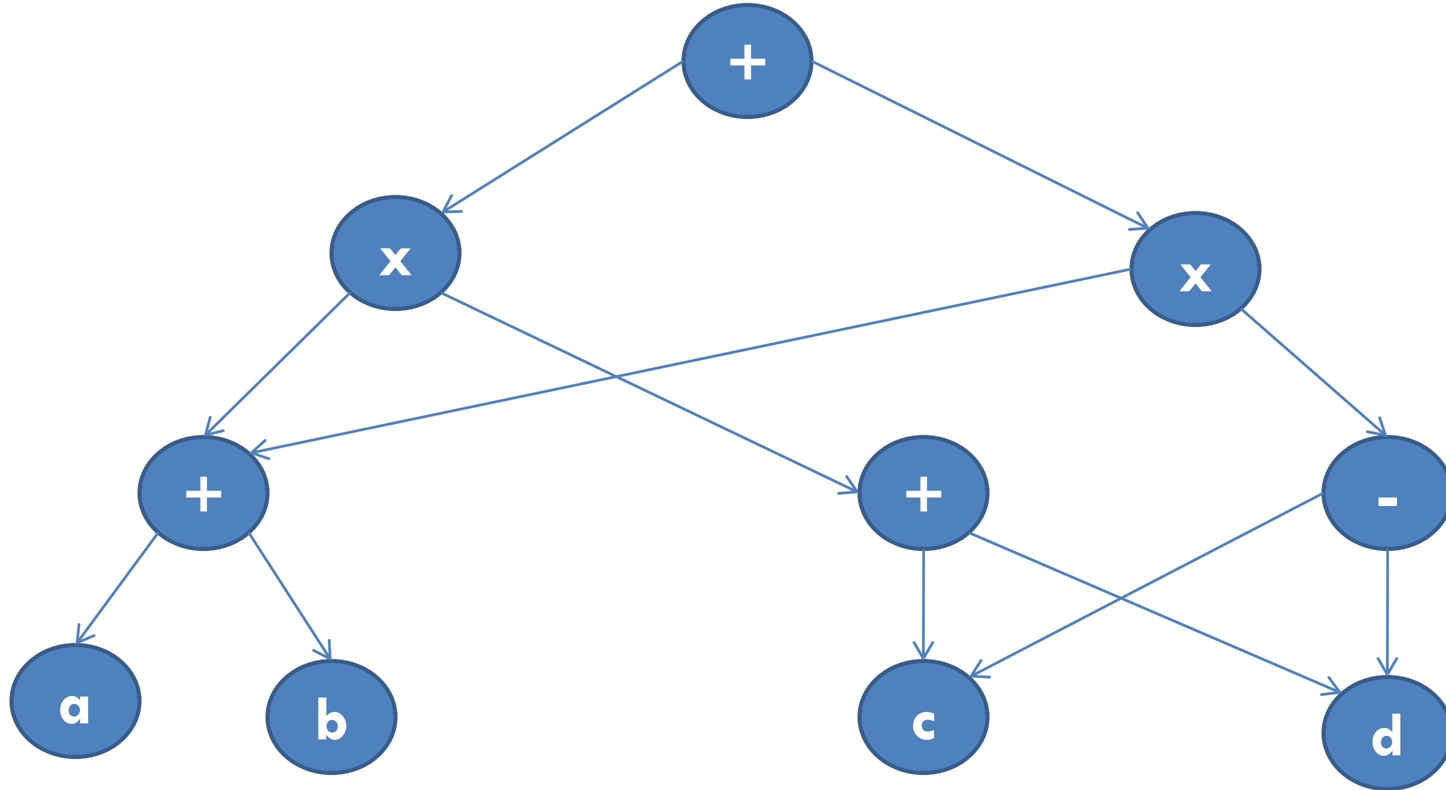
9. There are no more blocks to visit

Depth First Search: Directed Graphs



Acyclic Graphs: Topological Sorting

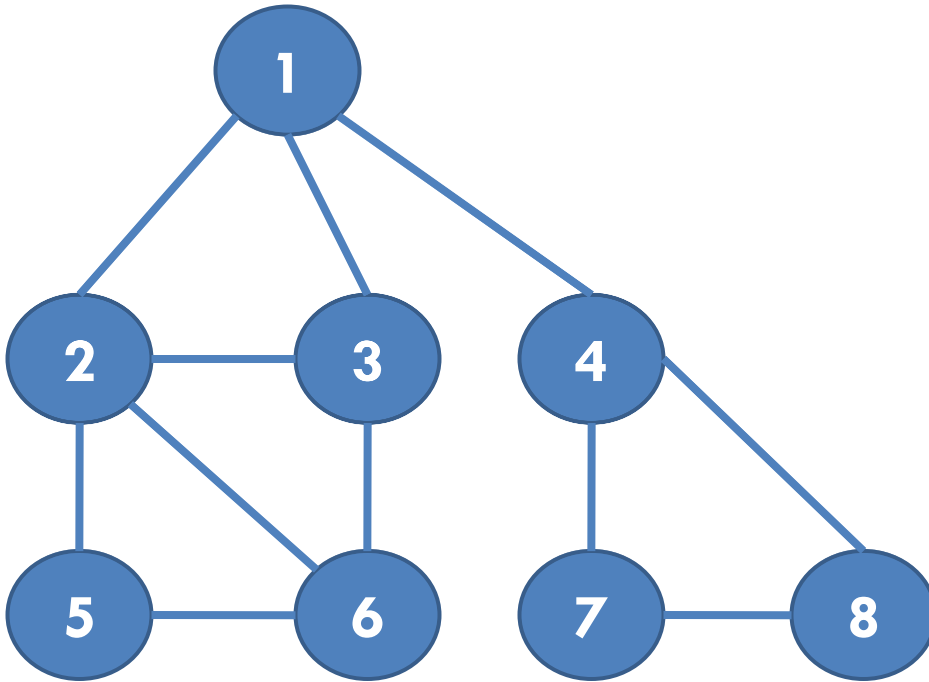
(a + b) (c + d) + (a + b) (c - d)   Arithmetic Expression



Breadth-first Search

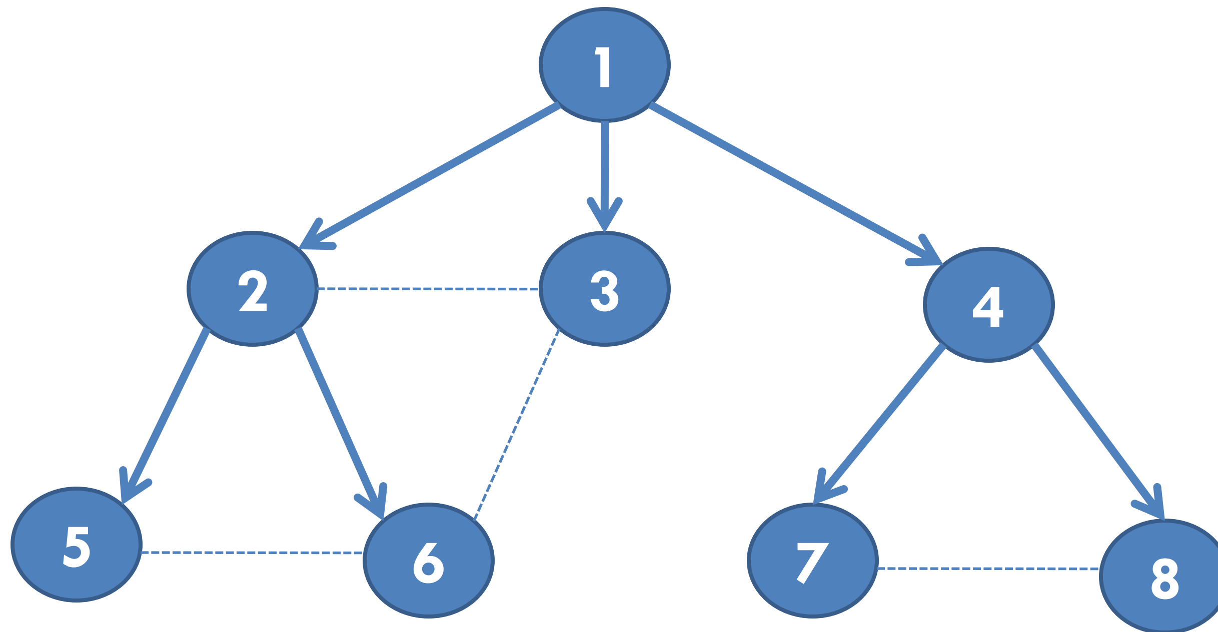
- Unlike depth-first search, breadth-first search is not naturally recursive.
- It is most often used to carry-out a partial exploration of an infinite (or unmanageably large) graph, or to find the shortest path from one point to another point.
- Mostly depth-first search uses push & pop operations in the order “last come first serve”
- While Breadth-first search uses enqueue & dequeue operation in the order “first come, first serve”

Breadth-first Search



	Node Visited	Q
1	**1**	**2, 3, 4**
2	**2**	**3, 4, 5, 6**
3	**3**	**4, 5, 6**
4	**4**	**5, 6, 7, 8**
5	**5**	**6, 7, 8**
6	**6**	**7, 8**
7	**7**	**8**
8	**8**	**--**

Breadth-first Search



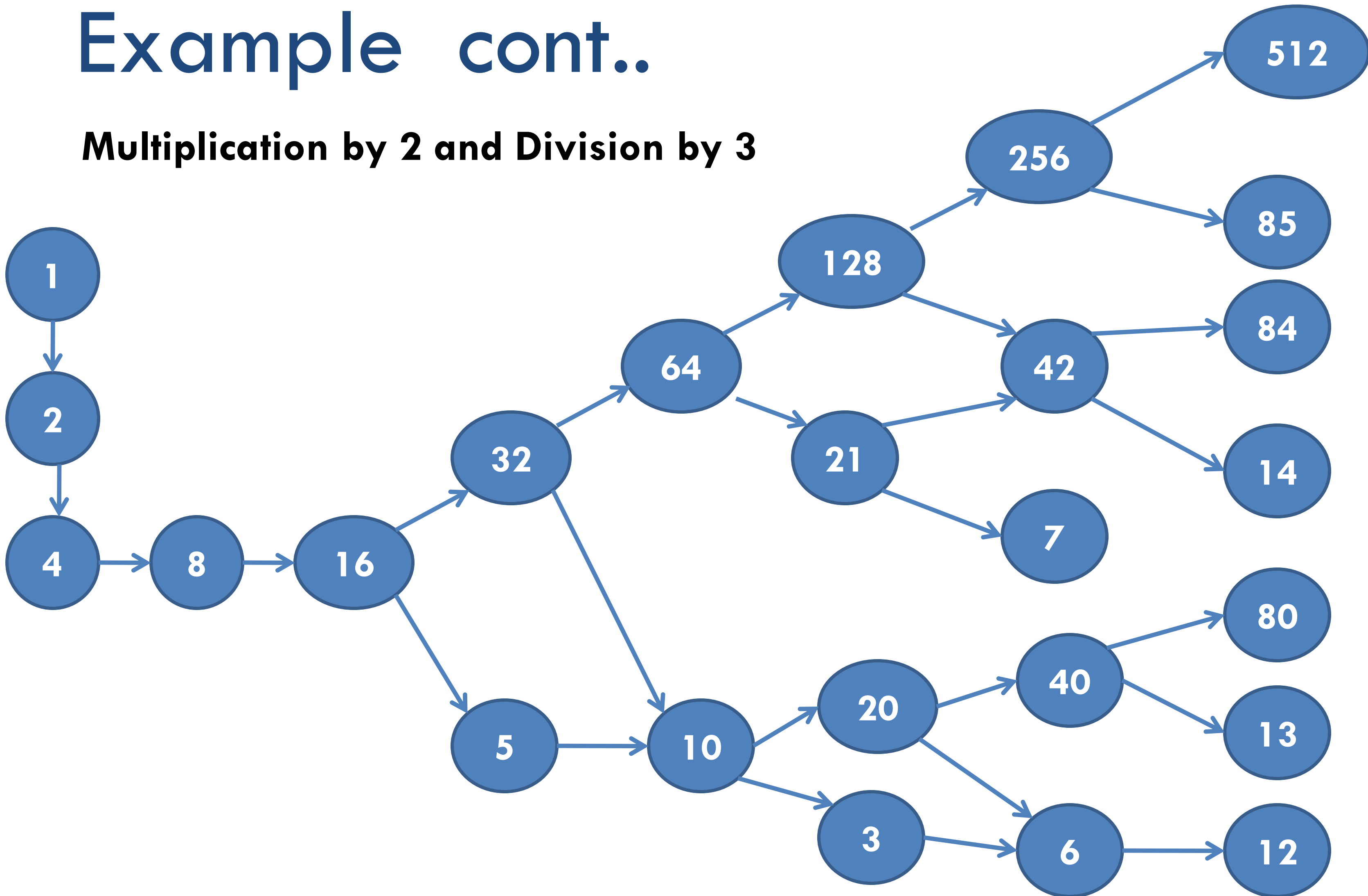
Breadth-first Search - Example

- It is most often used to carry-out a partial exploration of an infinite (or unmanageably large) graph, or to find the shortest path from one point to another point.
- Consider for example the following problem.
  - *The value 1 is given. To construct other values, two operations are available:*

    multiplication by 2 and division by 3.
  - *For second operation, the operand must be greater than 2 (so we cannot reach 0), and any resulting fraction is dropped.*

Example cont..

Multiplication by 2 and Division by 3



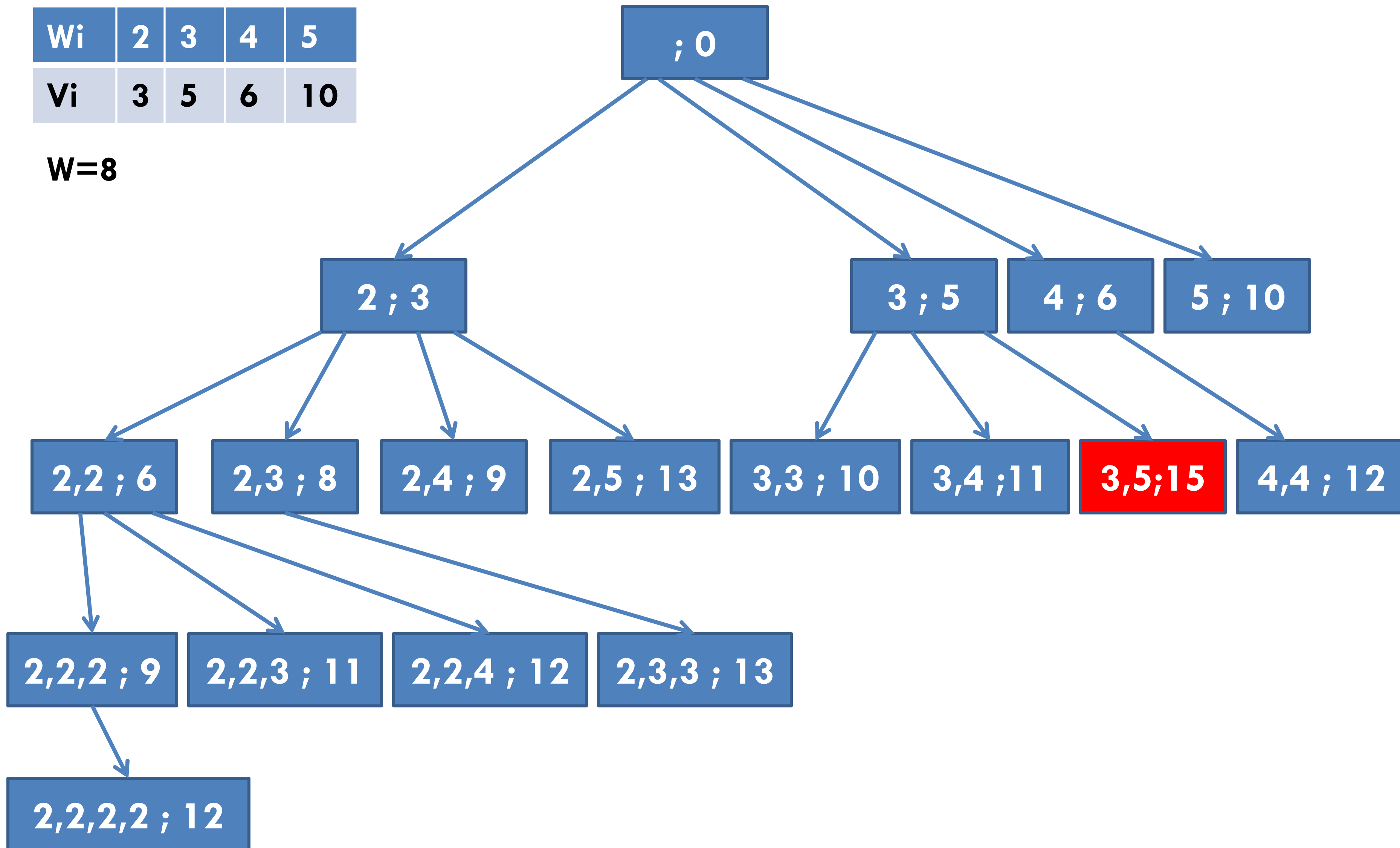
10 = 1 X 2 X 2 X 2 X 2 / 3 X 2

13 = 1 X 2 X 2 X 2 X 2 / 3 X 2 X 2 X 2 / 3

Backtracking

- If the graph contains a large number of nodes, and particularly if it is infinite, it may be wasteful or infeasible to build it *explicitly* in computer storage. E.g. Game of Chess
- In such a situation we use an *implicit* graph. This is one for which we have available a description of its nodes and edges, so relevant parts of the graph can be built as the search progresses. In this way computing time is saved whenever the search succeeds before the entire graph has been constructed.

Backtracking – Knapsack Problem(3)



Suppose that we have n types of object, and that an adequate number of objects of each type are available.

Branch-and-bound

- Like backtracking, branch-and-bound is a technique for exploring an implicit directed graph. Again, this graph is usually acyclic or even a tree.
- This time, we are looking for the optimal solution to some problem.

Assignment Problem

- In the assignment problem, n agents are to be assigned n tasks, each agent having exactly one task to perform.
- If agent i, 1<= i <=n, is assigned task j, 1<= j <=n, then the cost of performing this particular task will be c_{ij}
- Given the complete matrix of cost, the problem is to assign agents to tasks so as to minimize the total cost of executing the n tasks.

Assignment Problem

- For example, suppose three agents a, b and c are to be assigned tasks 1, 2 and 3, and the cost matrix is as follows:

	1	**2**	**3**
a	4	7	3
b	2	6	1
c	3	9	4

- Here, the optimal assignment is a→2, b→3 and c→1, whose cost is 7 + 1 + 3 = 11

Assignment Problem - Example

- For example, suppose four agents a, b, c and d are to be assigned tasks 1, 2, 3 and 4, and the cost matrix is as follows:

	1	**2**	**3**	**4**
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Assignment Problem - Example

- Obtain an **upper bound**, a→1, b→2, c→3, d→4 whose cost is 11 + 15 + 19 + 28 = 73
- The optimal solution to the problem cannot cost more than this.
- Another solution is a→4, b→3, c→2, d→1, whose cost is 40 + 13 + 17 + 17 = 87
- But it's not improved over the first.
- So, Upper Bound is 73

Assignment Problem - Example

- Now, Obtain a **lower bound.**
- First find out minimum value from *each column*. That is 11 + 12 + 13 + 22 = 58
- Second find out minimum value from *each row*. That is 11 + 13 + 11 + 14 = 49
- But it's not improved over the first.
- So, Upper Bound is 58

Pulling these facts together, we know that the answer to our instance lies somewhere in [58…73]

Assignment Problem - Example

- For example, suppose four agents a, b, c and d are to be assigned tasks 1, 2, 3 and 4, and the cost matrix is as follows:

	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Upper Bound

Lower Bound

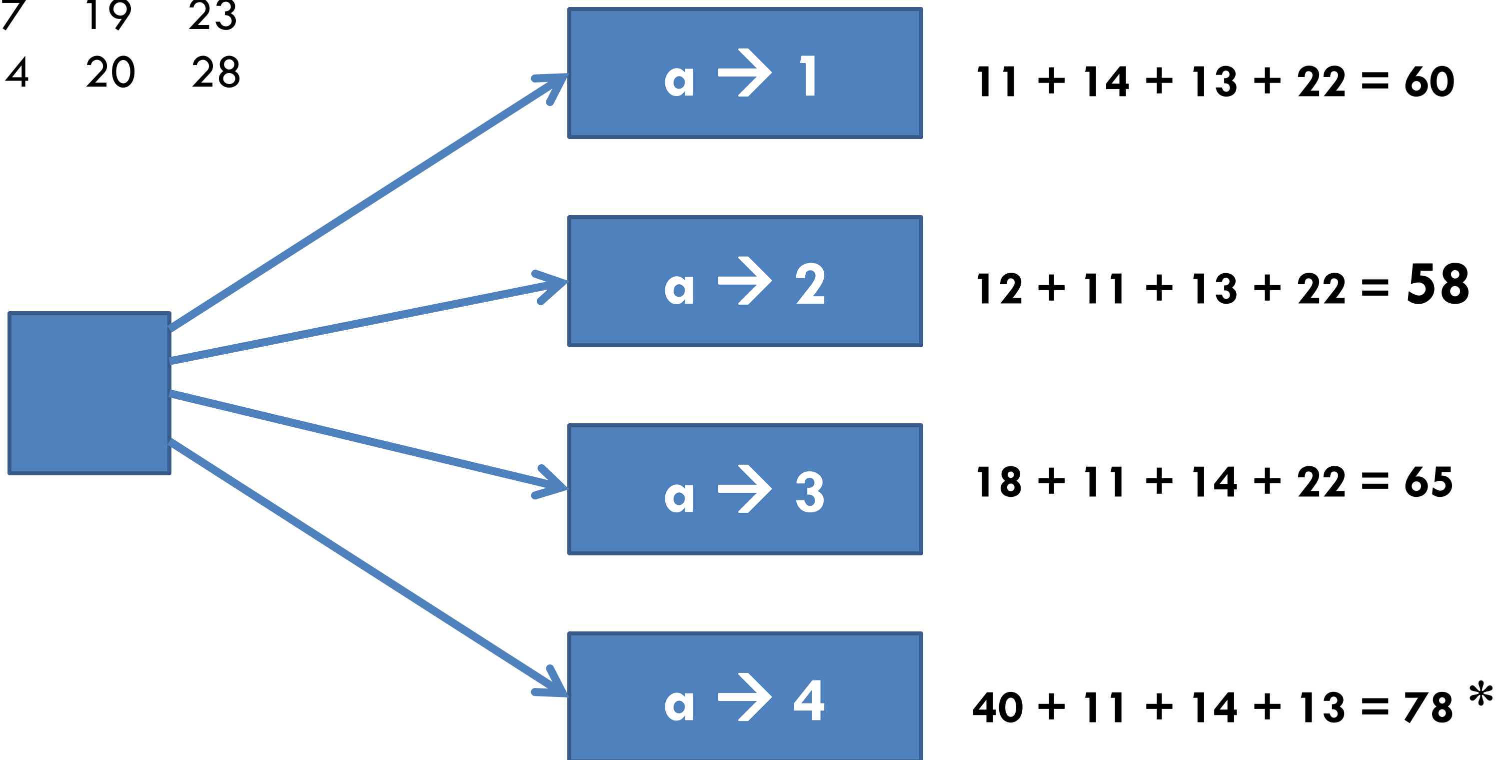
Upper Bound

Lower Bound

Assignment Problem - Example

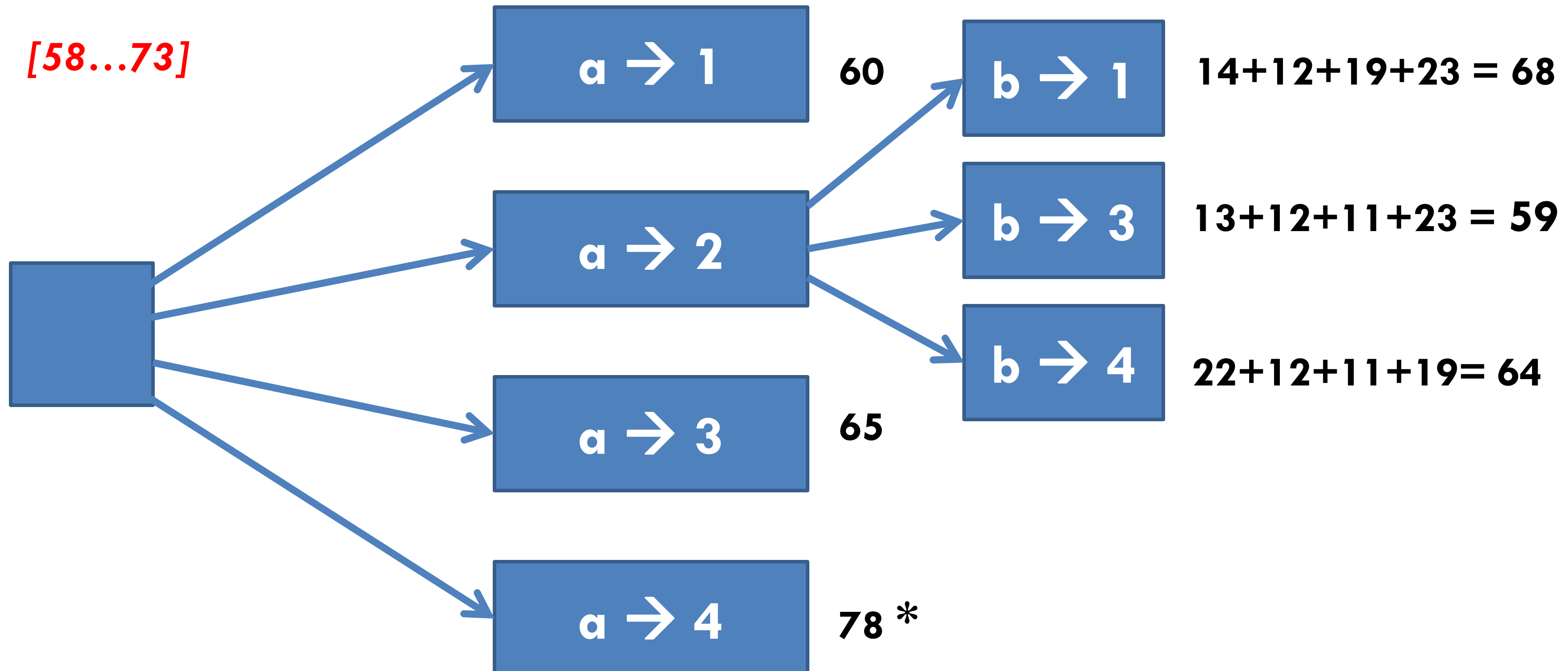
	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

[58…73]



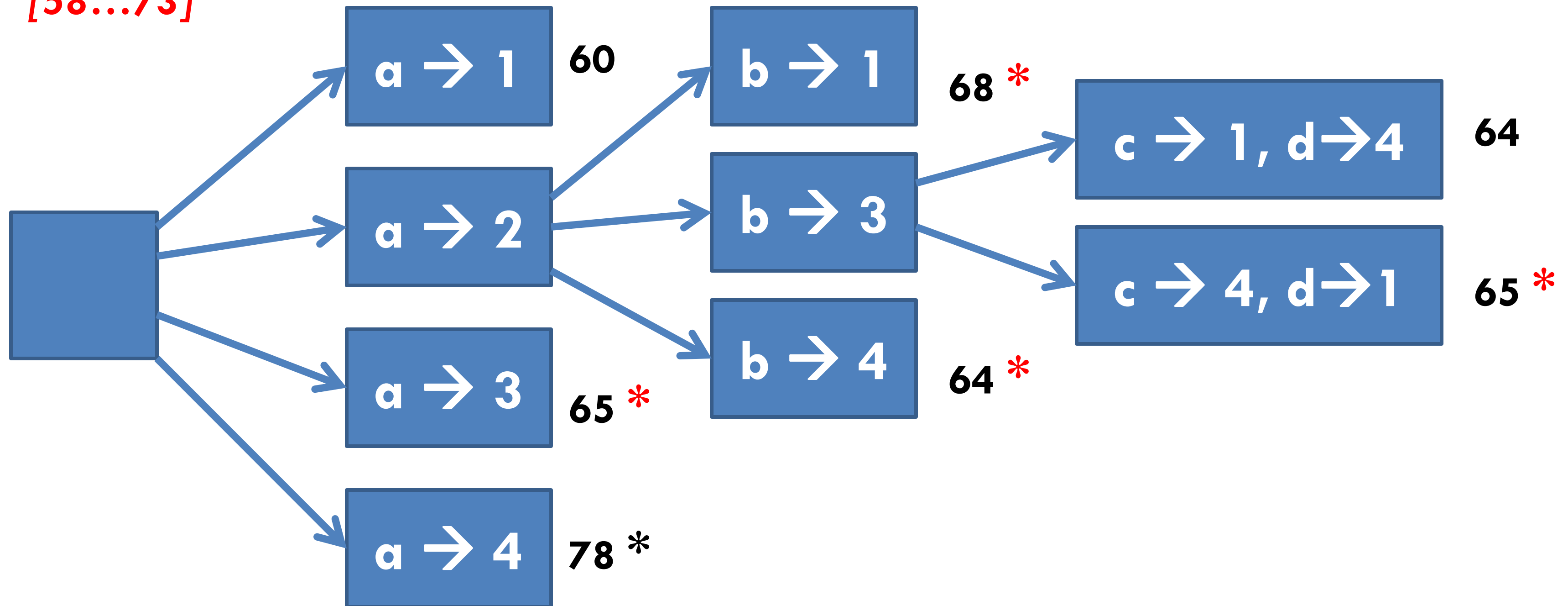
Assignment Problem - Example

	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

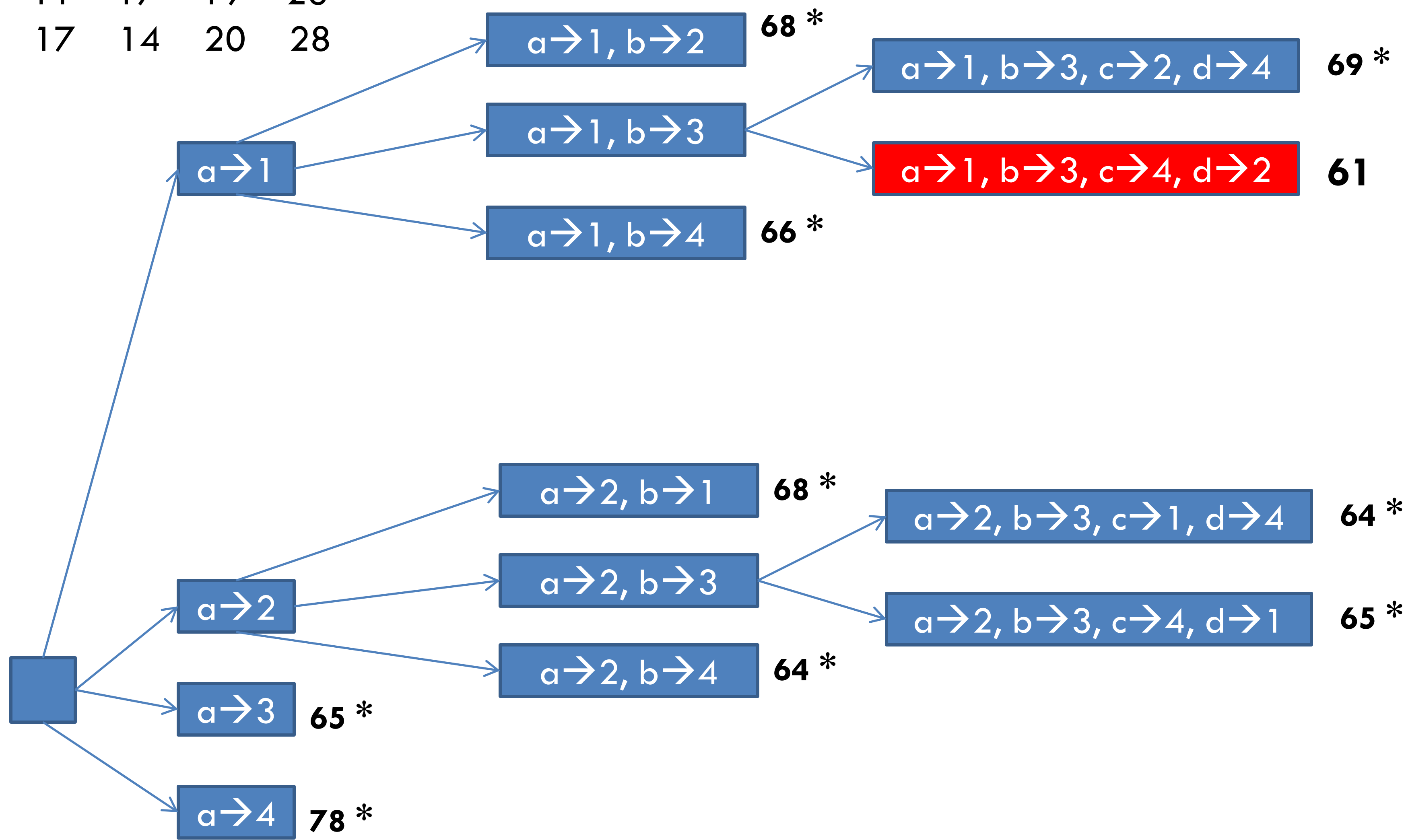


Assignment Problem - Example

	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28



To obtain the correct answer, we constructed just 15 of the 41 nodes.