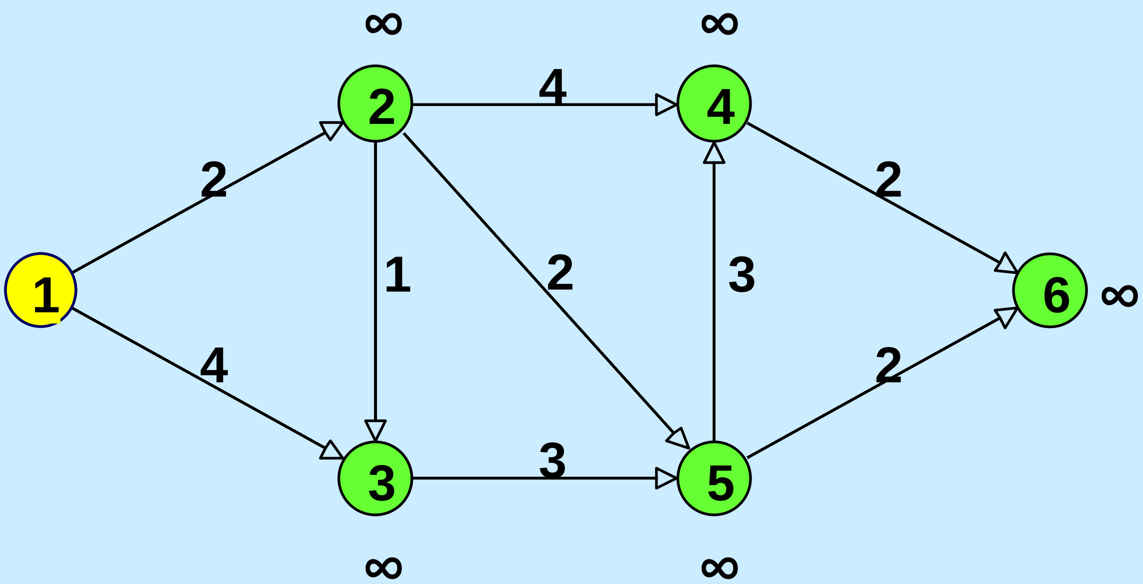
15.082J & 6.855J & ESD.78J
September 23, 2010

Dijkstra's Algorithm for the Shortest Path Problem

Single source shortest path problem



Find the shortest path from a source node to each other node.

Assume: (1) all arc lengths are non-negative

(2) the network is directed

(3) there is a path from the source node to all other nodes

Overview of today's lecture

- Dijkstra's algorithm
  - animation
  - proof of correctness (invariants)
  - time bound
- A surprising application (see the book for more)
- A Priority Queue implementation of Dijkstra's Algorithm (faster for sparse graphs)

A Key Step in Shortest Path Algorithms

- **In this lecture, and in subsequent lectures, we let d( ) denote a vector of temporary distance labels.**

- **d(i) is the length of some path from the origin node 1 to node i.**

- **Procedure Update(i)**

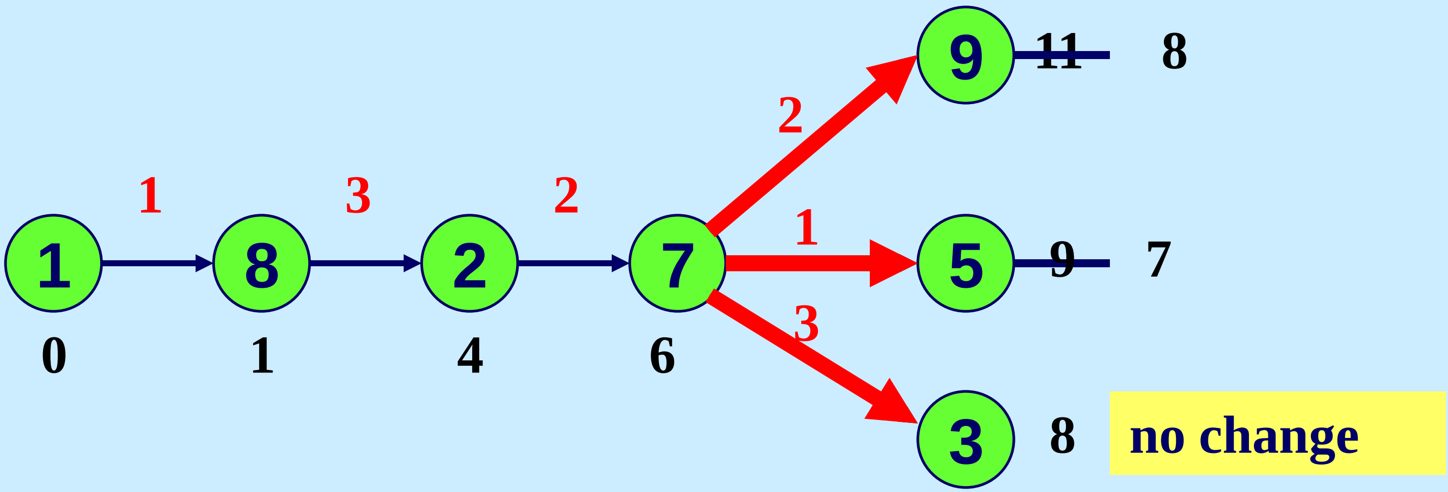
  for **each (i,j) $\in$ A(i)** ***do***

  if **d(j) > d(i) + c_{ij}** ***then*** **d(j) : = d(i) + c_{ij} and pred(j) : = i;**

- **Update(i)**
- **used in Dijkstra's algorithm and in the label correcting algorithm**

Update(7)

d(7) = 6 at some point in the algorithm, because of the path 1-8-2-7

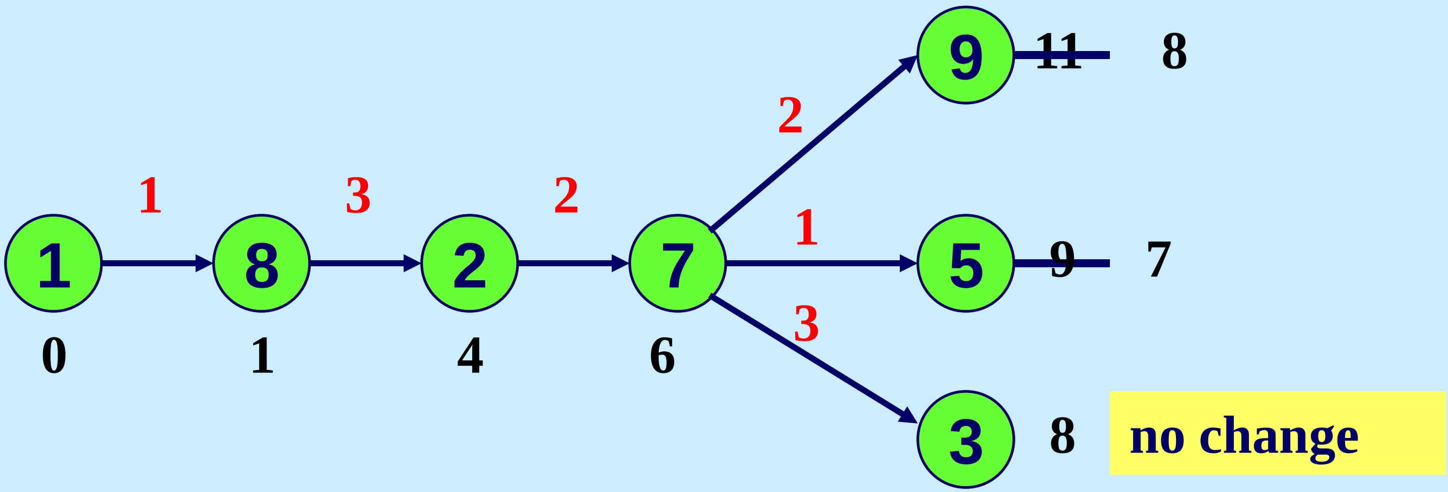


Suppose 7 is incident to nodes 9, 5, 3, with temporary distance labels as shown.

We now perform Update(7).

On Updates

Note: distance labels cannot increase in an update step. They can decrease.



We do not need to perform Update(7) again, unless d(7) decreases. Updating sooner could not lead to further decreases in distance labels.

In general, if we perform Update(j), we do not do so again unless d(j) has decreased.

Dijkstra's Algorithm

Let d*(j) denote the shortest path distance from node 1 to node j.

Dijkstra's algorithm will determine d*(j) for each j, in order of increasing distance from the origin node 1.

S denotes the set of *permanently labeled* nodes.

That is, d(j) = d*(j) for j $\in$ S.

T = N\S denotes the set of *temporarily labeled* nodes.

Dijkstra's Algorithm

$S := \{1\}$; $T = N - \{1\}$;

$d(1) := 0$ and $pred(1) := 0$; $d(j) = \infty$ for $j = 2$ to n;

update(1);

while $S \neq N$ ***do***

(*node selection, also called FINDMIN)*

let $i \in T$ be a node for which

$d(i) = \min \{d(j) : j \in T\}$;

$S := S \cup \{i\}$; $T := T - \{i\}$;

Update(i)

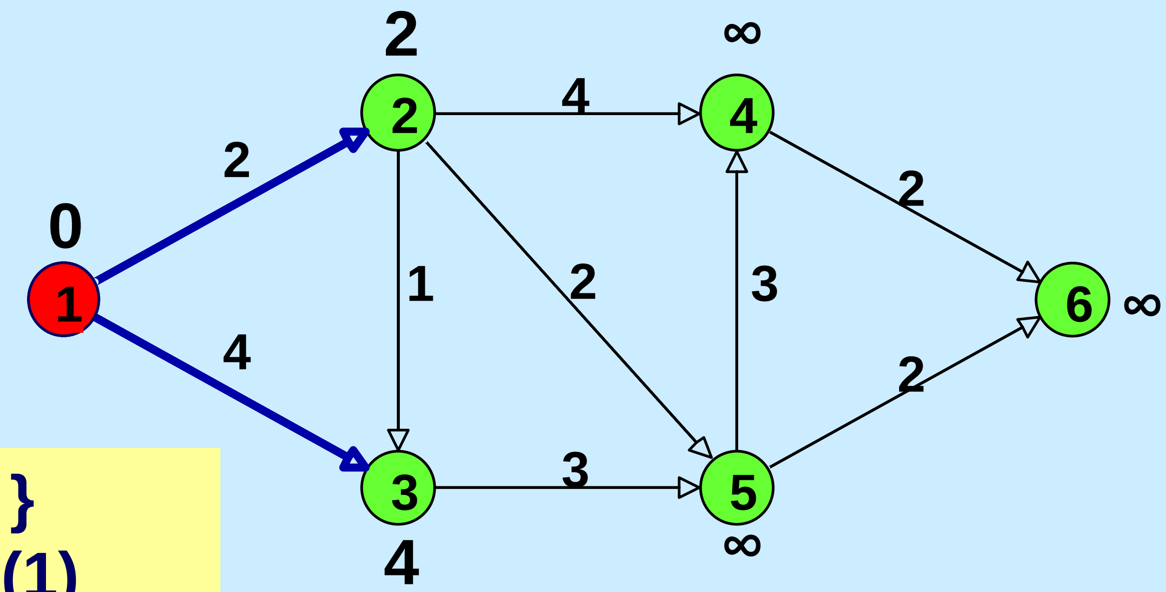
Dijkstra's Algorithm Animated

Invariants for Dijkstra's Algorithm

1. If $j \in S$, then $d(j) = d^*(i)$ is the shortest distance from node 1 to node j.
2. (after the update step) If $j \in T$, then $d(j)$ is the length of the shortest path from node 1 to node j in $S \cup \{j\}$, which is the shortest path length from 1 to j of scanned arcs.

Note: S increases by one node at a time. So, at the end the algorithm is correct by invariance 1.

Verifying invariants when S = { 1 }

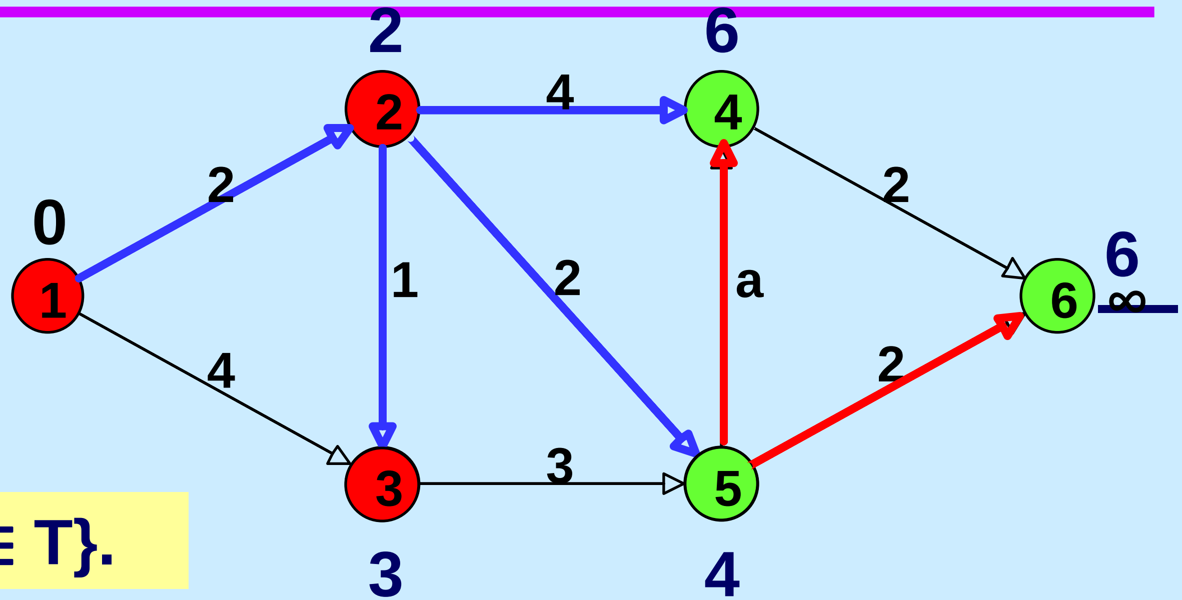


Consider S = { 1 }
and after update(1)

1. **If j ∈ S, then d(j) is the shortest distance from node 1 to node j.**
2. **3. If j ∈ T, then d(j) is the length of the shortest path from node 1 to node j in S ∪ {j}.**

Verifying invariants Inductively

Assume that the invariants are true before a node selection



d(5) = min {d(j) : j ∈ T}.

Any path from 1 to 5 passes through a node k of T. The path to node k has distance at least d(5). So d(5) = d*(5).

Suppose 5 is transferred to S and we carry out Update(5). Let P be the shortest path from 1 to j with j ∈ T.

If 5 ∉ P, then invariant 2 is true for j by induction. If 5 ∈ P, then invariant 2 is true for j because of Update(5).

A comment on invariants

It is the standard way to prove that algorithms work.

- Finding the best invariants for the proof is often challenging.
- A reasonable method. Determine what is true at each iteration (by carefully examining several useful examples) and then use all of the invariants.
- Then shorten the proof later.

Complexity Analysis of Dijkstra's Algorithm

- **Update Time:** update(j) occurs once for each j, upon transferring j from T to S. The time to perform all updates is O(m) since the arc (i,j) is only involved in update(i).
- **FindMin Time:** To find the minimum (in a straightforward approach) involves scanning d(j) for each $j \in T$.
  - Initially T has n elements.
  - So the number of scans is n + n-1 + n-2 + … + 1 = $O(n^2)$.
- **$O(n^2)$ time in total.** This is the best possible only if the network is *dense*, that is m is about n^2.
- We can do better if the network is *sparse*.

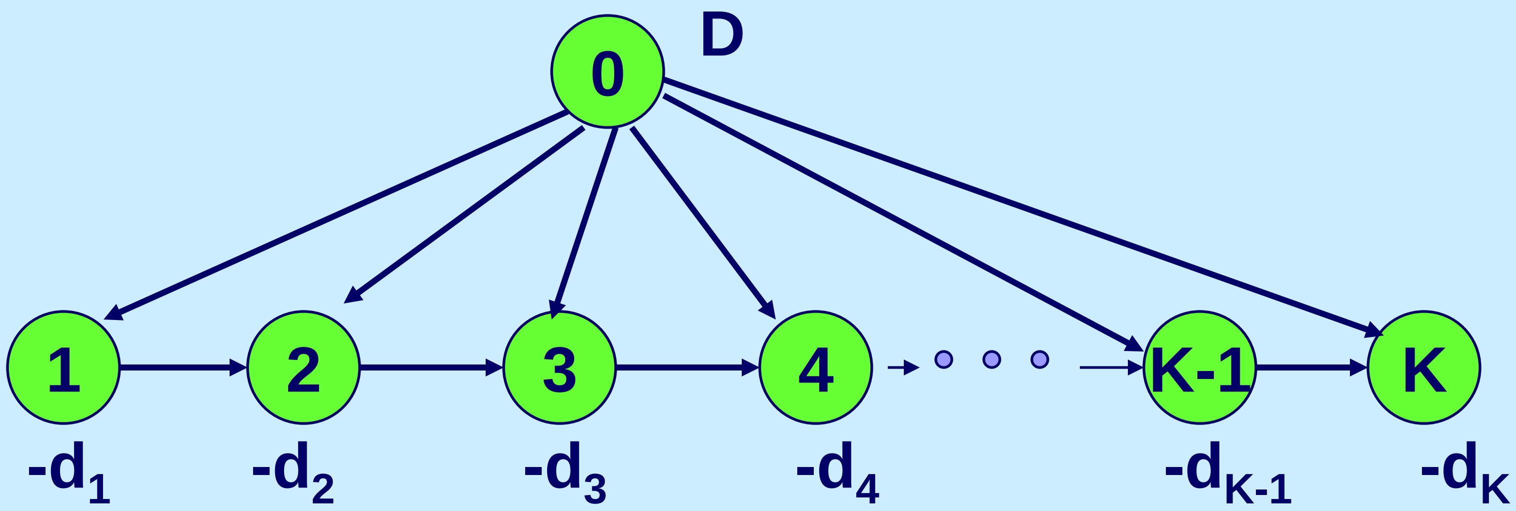
Application 19.19. Dynamic Lot Sizing

- K periods of demand for a product. The demand is d_j in period j. Assume that $d_j > 0$ for j = 1 to K.
- Cost of producing p_j units in period j: $a_j + b_j p_j$
- h_j : unit cost of carrying inventory from period j

- Question: what is the minimum cost way of meeting demand?

- Tradeoff: more production per period leads to reduced production costs but higher inventory costs.

Application 19.19. Dynamic Lot Sizing (1)



Flow on arc (0, j): amount produced in period j

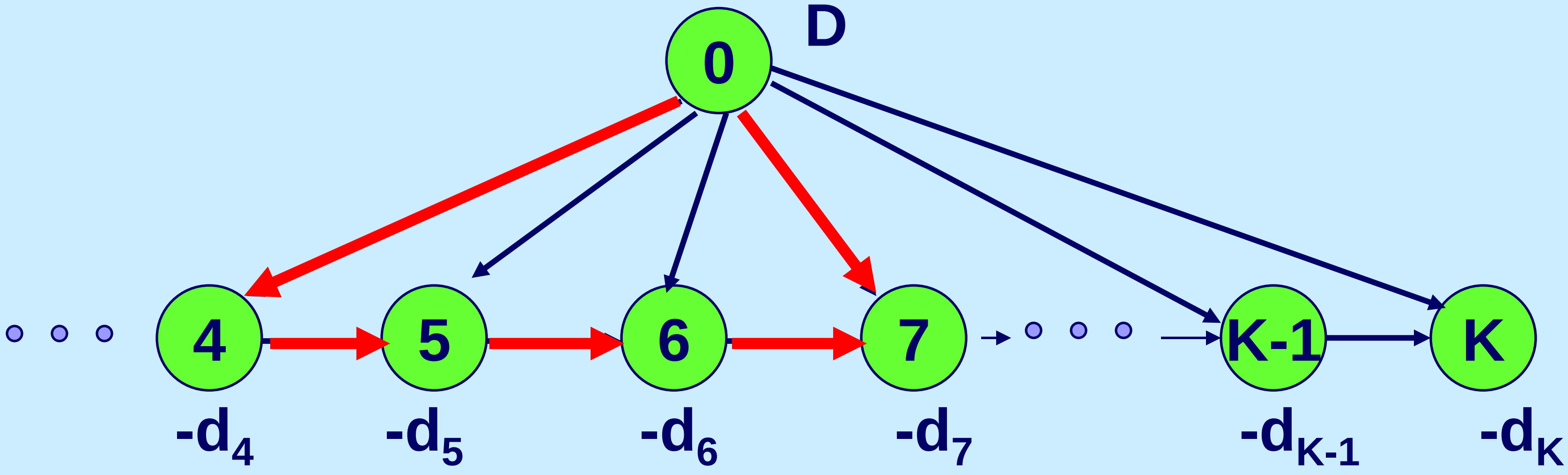
Flow on arc (j, j+1): amount carried in inventory from period j

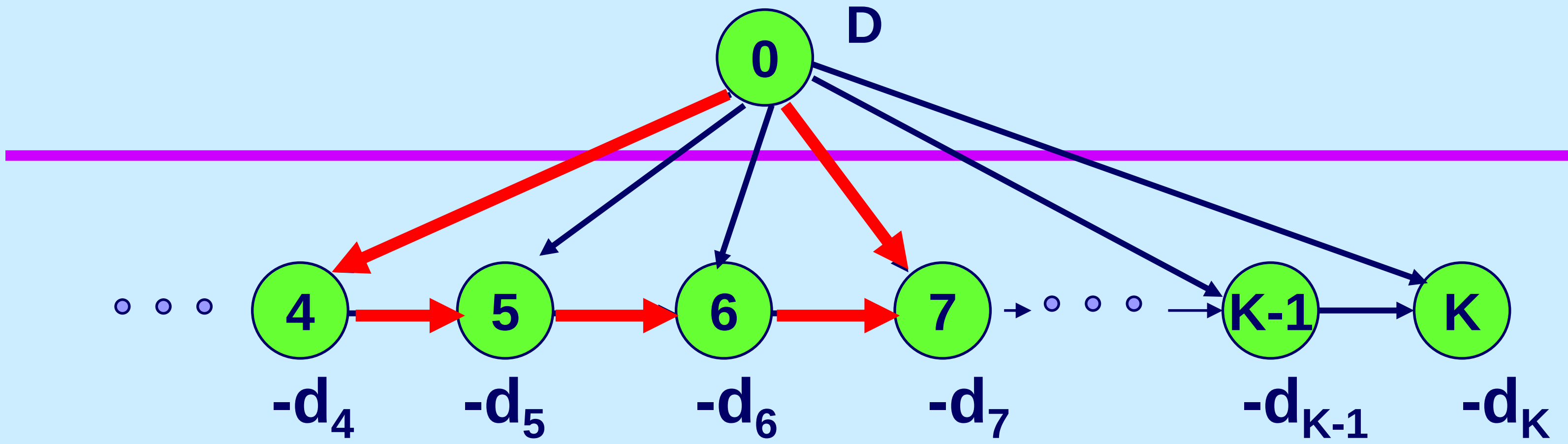
Lemma: There is production in period j or there is inventory carried over from period j-1, but not both.

Lemma: There is production in period j or there is inventory carried over from period j-1, but not both.

Suppose now that there is inventory from period j-1 and production in period j. Let period i be the last period in which there was production prior to period j, e.g., j = 7 and i = 4.

Claim: There is inventory stored in periods i, i+1, …, j-1





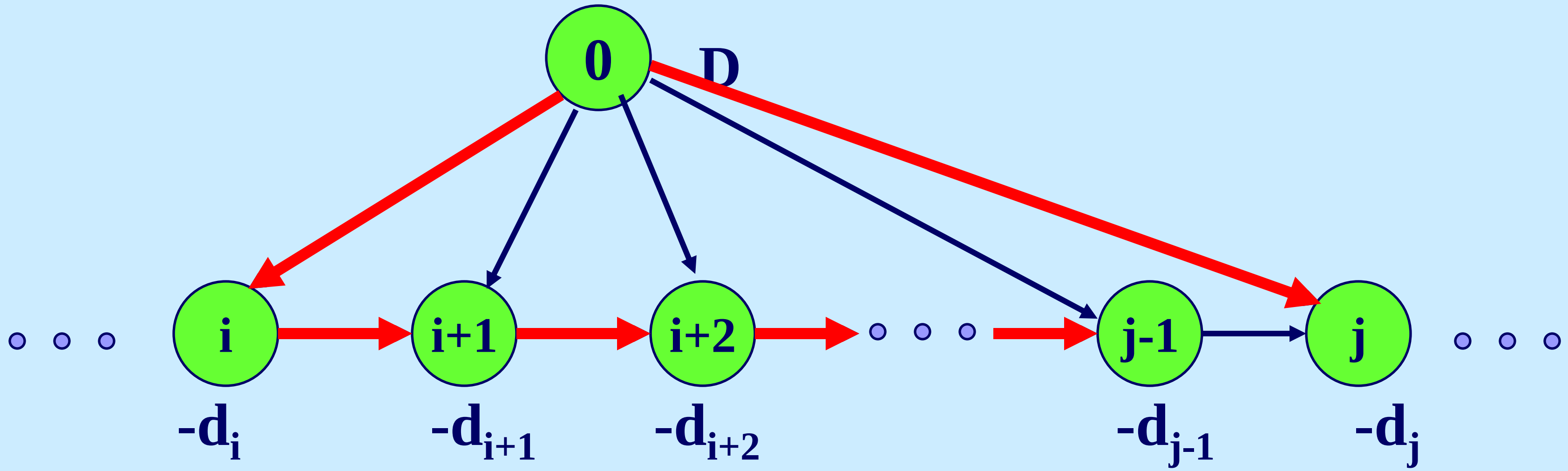
Thus there is a cycle C with positive flow. C = 0-4-5-6-7-0. Let x_{07} be the flow in (0,7).

The cost of sending Δ units of flow around C is linear (ignoring the fixed charge for production). Let $Q = b_4 + h_4 + h_5 + h_6 – b_7$.

- **If Q < 0, then the solution can be improved by sending a unit of flow around C.**
- **If Q > 0, then the solution can be improved by decreasing flow in C by a little.**
- **If Q = 0, then the solution can be improved by increasing flow around C by x_{07} units (and thus eliminating the fixed cost a_7).**
- **This contradiction establishes the lemma.**

Corollary. Production in period i satisfies demands exactly in periods i, i+1, …, j-1 for some j.

Consider 2 consecutive production periods i and j. Then production in period i must meet demands in i+1 to j-1.



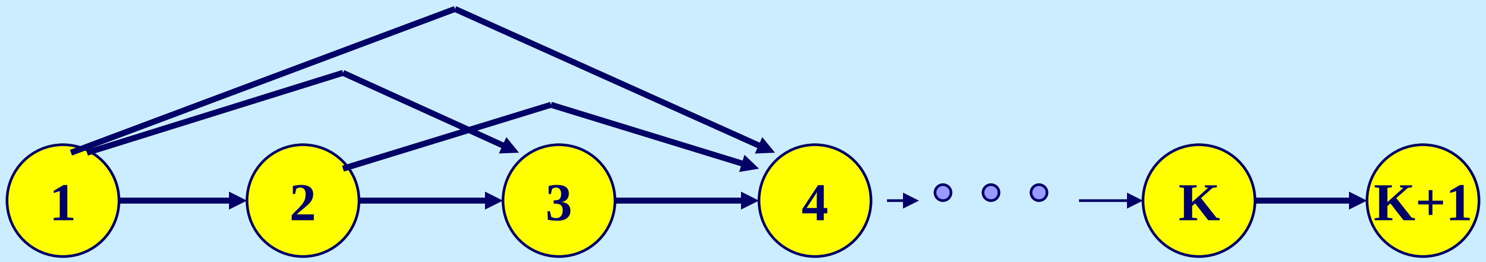
Let c_{ij} be the (total) cost of this flow.

$$c_{ij} = a_i + b_i(d_i + d_{i+1} + \ldots + d_{j-1})$$

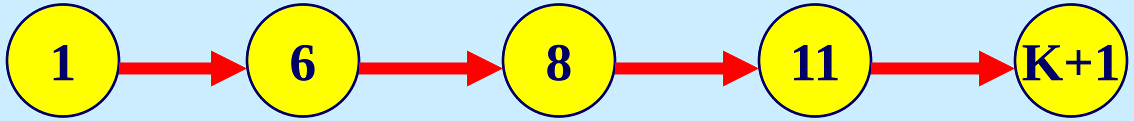
$$+ h_i(d_{i+1} + d_{i+2} + \ldots + d_{j-1})$$

$$+ h_{i+1}(d_{i+2} + d_{i+3} + \ldots + d_{j-1}) + \ldots \; h_{j-2}(d_{j-1})$$

Let c_{ij} be the cost of producing in period i to meet demands in periods i, i+1, …, j-1 (including cost of inventory). Create a graph on nodes 1 to K+1, where the cost of (i,j) is c_{ij}.



Each path from 1 to K+1 gives a production and inventory schedule. The cost of the path is the cost of the schedule.



Interpretation: produce in periods 1, 6, 8 and 11.

Conclusion: The minimum cost path from node 1 to node K+1 gives the minimum cost lot-sizing solution.

Next

- A speedup of Dijkstra's algorithm if the network is sparse

- New Abstract Data Type: Priority Queues

Priority Queues

- In the shortest path problem, we need to find the minimum distance label of a temporary node. We will create a data structure B that supports the following operations:

  1. Initialize(B): Given a set $T \subseteq N$, and given distance labels d, this operation initializes the data structure B.

  2. Findmin(B): This operation gives the node in T with minimum distance label

  3. Delete(B, j): This operation deletes the element j from B.

  4. Update(B, j, δ): This operation updates B when d(j) is changed to δ.

- In our data structure, Initialize will take O(n) steps. Delete Update, and FindMin will each take O(log n) steps.

Storing B in a complete binary tree.

- The number of nodes is n (e.g., 8)

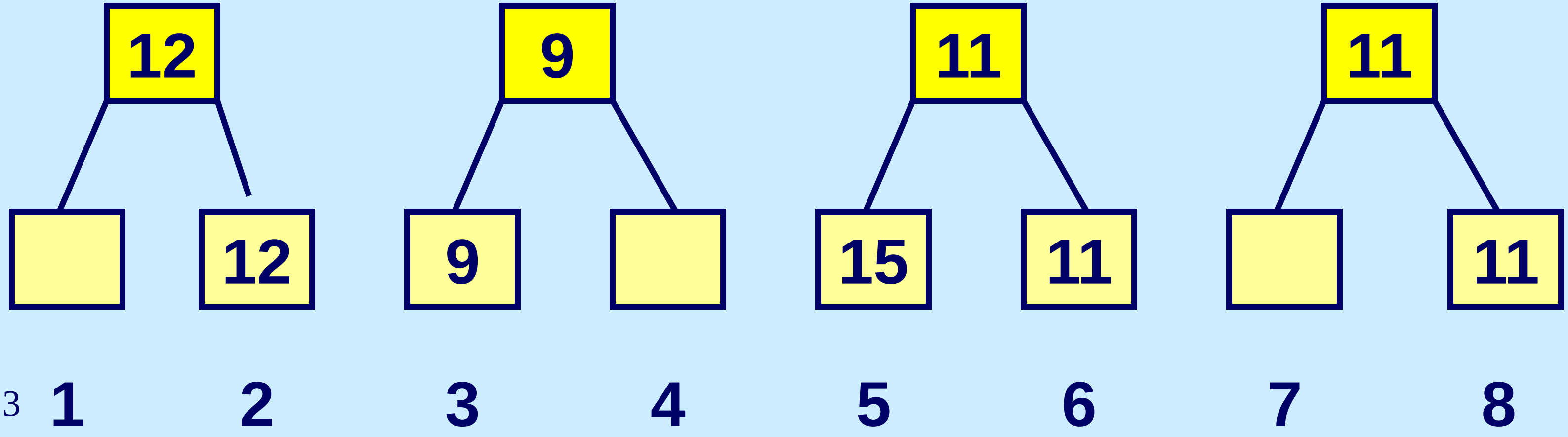
j	1	2	3	4	5	6	7	8
j∈T?	no	yes	yes	no	yes	yes	no	yes
d(j)	--	12	9		15	11		11

The parent will contain the minimum distance label of its children.

	12	9		15	11		11
1	2	3	4	5	6	7	8

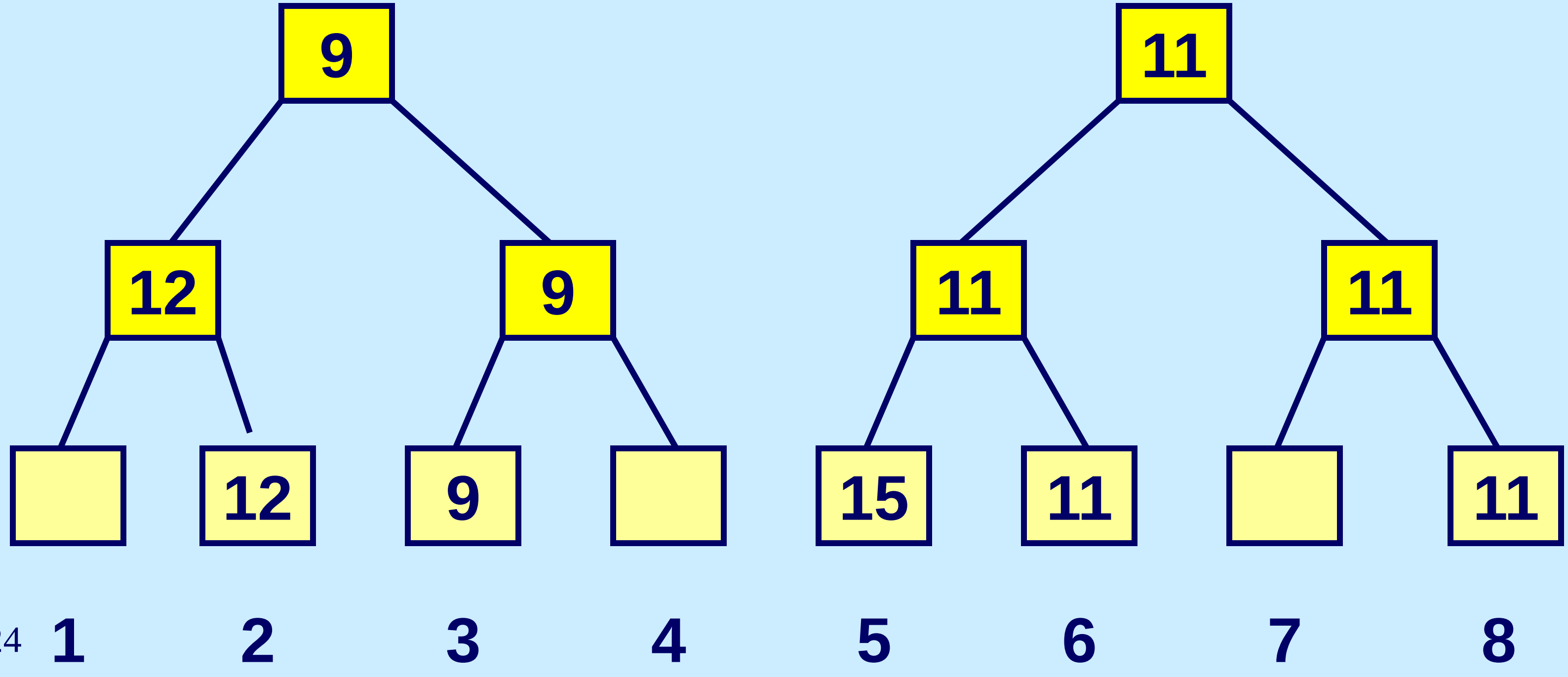
Storing B in a complete binary tree.

j	1	2	3	4	5	6	7	8
j∈T?	no	yes	yes	no	yes	yes	no	yes
d(j)	--	12	9		15	11		11



Storing B in a complete binary tree.

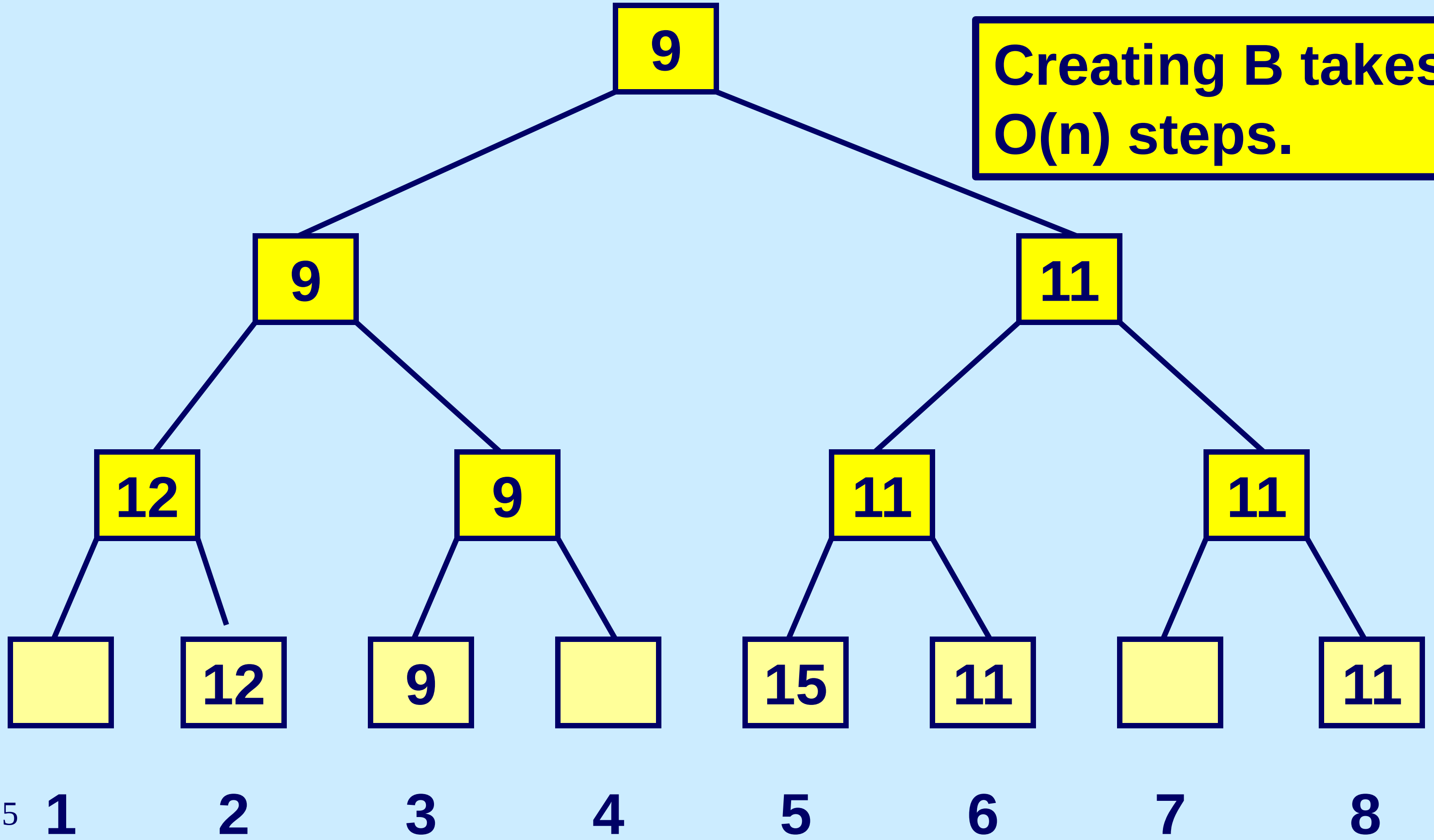
j	1	2	3	4	5	6	7	8
j∈T?	no	yes	yes	no	yes	yes	no	yes
d(j)	--	12	9		15	11		11



Storing B in a complete binary tree.

j	1	2	3	4	5	6	7	8
j∈T?	no	yes	yes	no	yes	yes	no	yes
d(j)	--	12	9		15	11		11

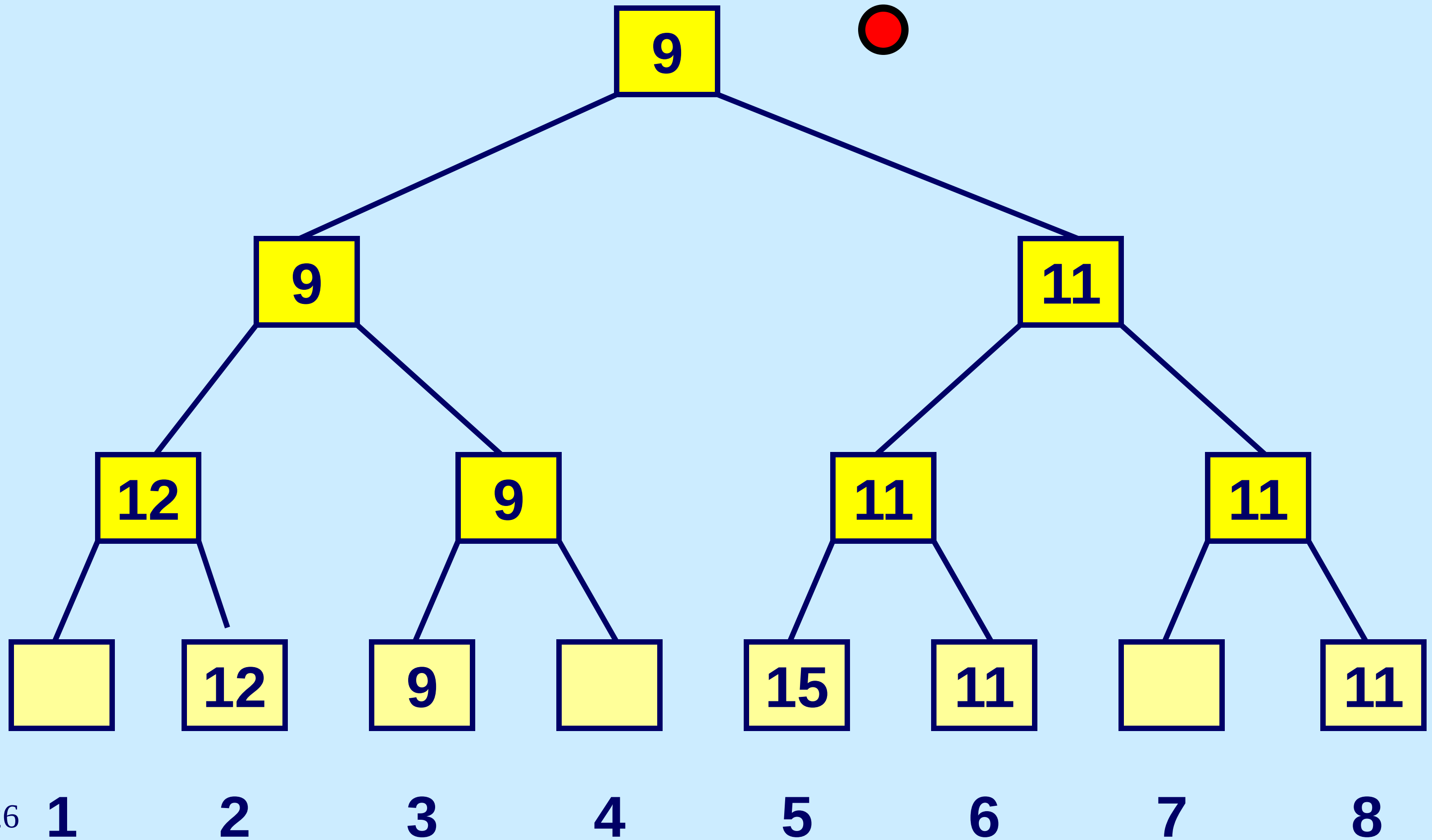
Creating B takes O(n) steps.



Finding the minimum element

Start at the top and follow the minimum value

FindMin takes O(log n) steps.

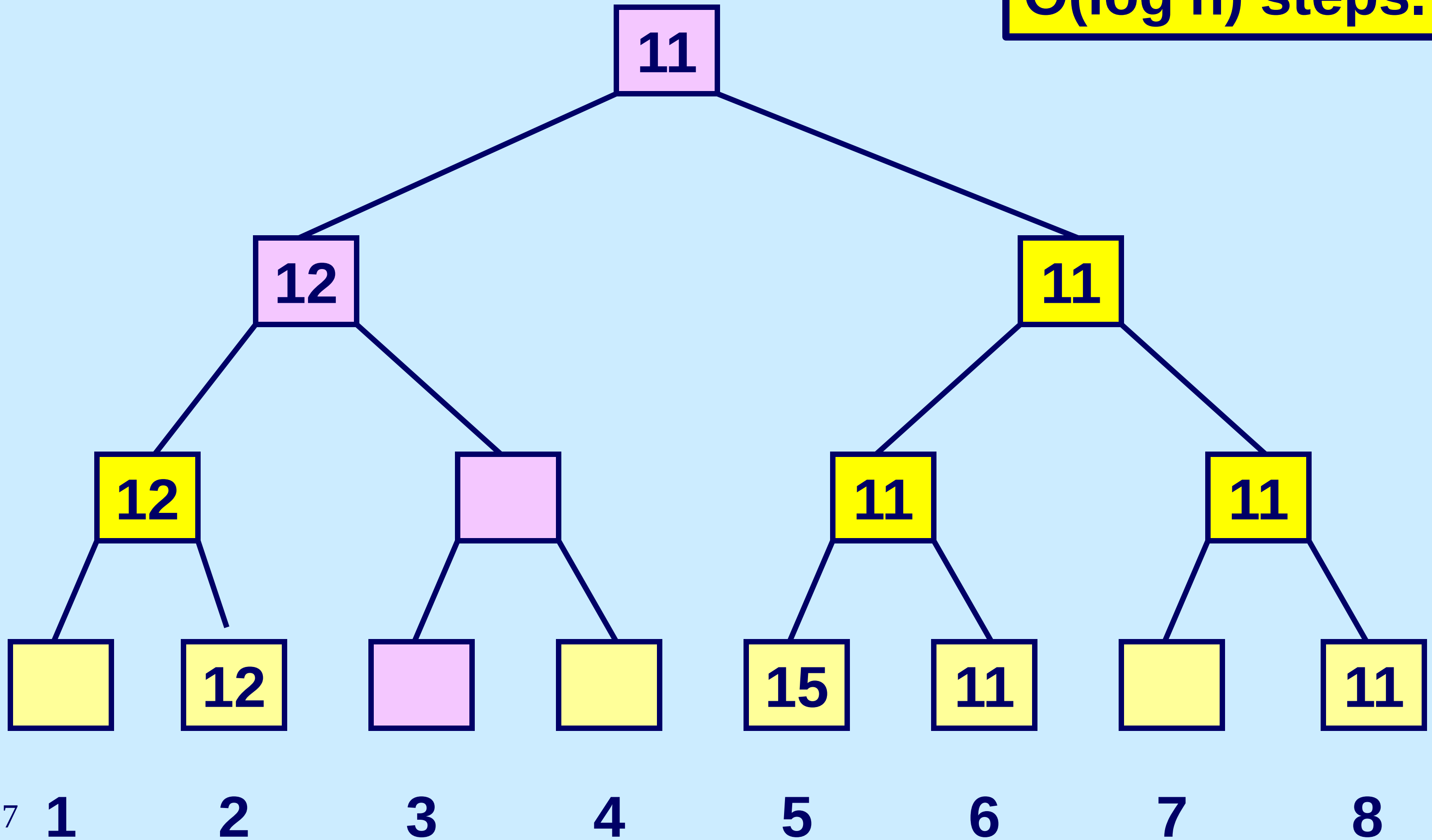


Deleting or inserting or changing an element

Suppose that node 3 is deleted from T.

Start at the bottom and work upwards

O(log n) steps.



Complexity Analysis using Priority Queues

- **Update Time:** update(j) occurs once for each j, upon transferring j from T to S. The time to perform all updates is O(m log n) since the arc (i,j) is only involved in update(i), and updates take O(log n) steps.

- **FindMin Time:** O(log n) per find min.
  O(n log n) for all find min's

- **O(m log n) running time**

Comments on priority queues

- Usually, “binary heaps” are used instead of a complete binary tree.
  - similar data structure
  - same running times up to a constant
  - better in practice
- There are other implementations of priority queues, some of which lead to better algorithms for the shortest path problem.

Summary

- Shortest path problem, with
  - Single origin
  - non-negative arc lengths
- Dijkstra's algorithm (label setting)
  - Simple implementation
  - Dial's simple bucket procedure
- Application to production and inventory control.
- Priority queues implemented using complete binary trees.

15.082J / 6.855J / ESD.78J Network Optimization
Fall 2010