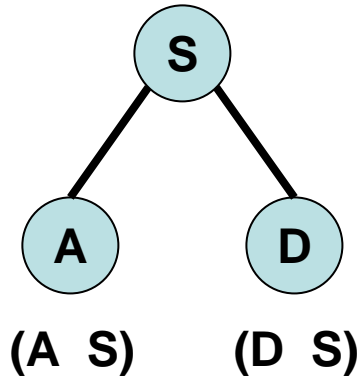


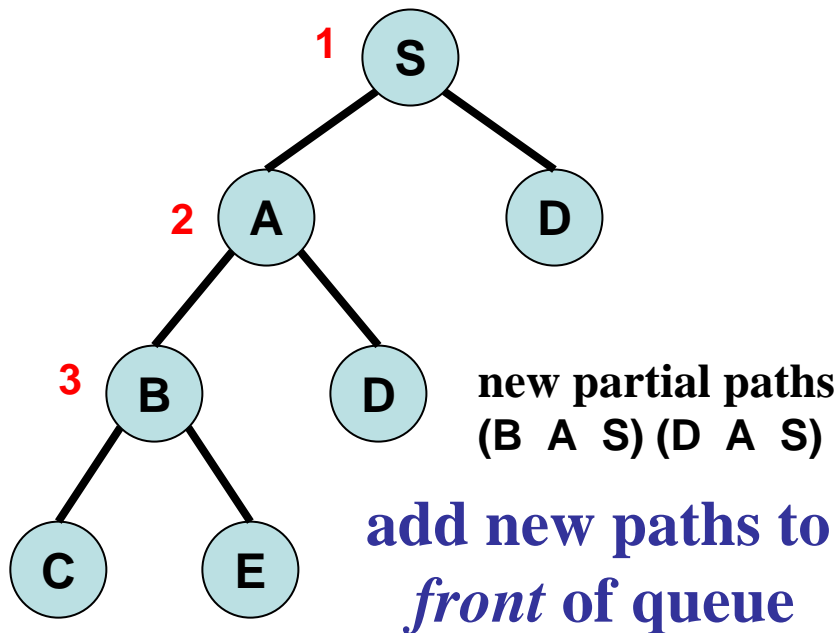
Representing partial solution paths



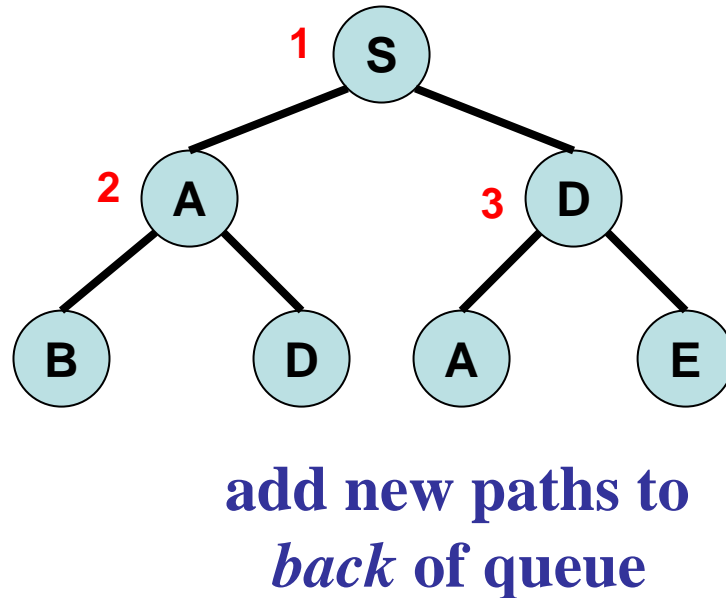
Queue of partial paths extended so far:

((S))
front ((A/S) (D S)) back
path to be extended next


Depth-first search



Breadth-first search



To conduct a blind search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - Add the new paths, if any,

- If the goal node is found, announce success; otherwise announce failure

To conduct a *depth-first* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - Add the new paths, if any,
to the *front* of the queue
- If the goal node is found, announce success; otherwise announce failure

To conduct a *breadth-first* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - Add the new paths, if any,
to the *back* of the queue
- If the goal node is found, announce success; otherwise announce failure

To conduct a *non-deterministic* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - Add the new paths, if any,
at random places in the queue
- If the goal node is found, announce success; otherwise announce failure

To conduct a *hill-climbing* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - ***Sort the new paths, if any, by the estimated distance between their terminal nodes and the goal***
 - ***Add the new paths, if any, to the *front* of the queue***
- If the goal node is found, announce success; otherwise announce failure

To conduct a *best-first* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - **Add the new paths, if any, to the queue**
 - ***Sort the entire queue by the estimated distance from the terminal nodes to the goal, with *least-cost paths in front****
- If the goal node is found, announce success; otherwise announce failure

To conduct a *branch & bound* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - **Add the new paths, if any, to the queue**
 - ***Sort the entire queue by the path lengths, with *least-cost paths in front****
- If the goal node is found, announce success; otherwise announce failure

To conduct an A^* search:

- Form an initial queue consisting of a single path that contains only the root node
- Until the first path in the queue terminates at the goal node or the queue is empty
 - Remove the first path from the queue
 - Create new paths by extending the first path to all the neighbors of the terminal node
 - Reject all new paths with loops
 - **Add the new paths, if any, to the queue**
 - **If two or more paths reach a common node, delete all those paths except the one that reaches the common node with the minimal cost**
 - ***Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with *least-cost paths in front****
- If the goal node is found, announce success; otherwise announce failure