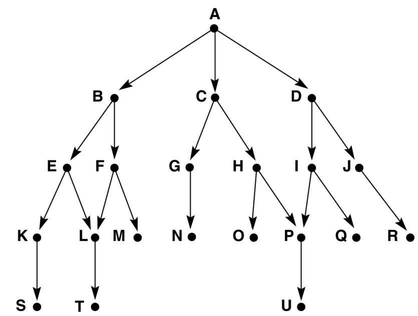
COMP 6721 Applied Artificial Intelligence (Fall 2023)

Worksheet #1: Solving Problems by Searching

This is an <u>active learning</u> exercise; we'll work on it during the lecture in <u>teams of two!</u>

Breadth-First Search. Let's apply the BFS algorithm discussed in the lecture on an example:

```
begin
   open := [Start];                                                  % initialize
   closed := [ ];
   while open ≠ [ ] do                                               % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS                       % goal found
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;   % loop check
                  put remaining children on right end of open           % queue
               end
      end
   return FAIL                                                       % no states left
end.
```



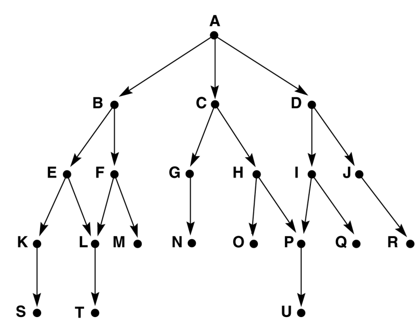
Assume **U** is the **goal state**. Note that **open** is a **queue**:

1. open = $[A_{\text{null}}]$, closed = [ ]
2. open = $[B_A\ C_A\ D_A]$, closed = $[A]$
3. open = $[C_A\ D_A\ E_B\ F_B]$, closed = $[B\ A]$
4. open = [....], closed = [........]
5. open = [....], closed = [................]
6. open = [....], closed = [........................]
7. open = [....], closed = [................................]

Depth-First Search. Now we do the same for the DFS algorithm:

Function depth_first_search algorithm

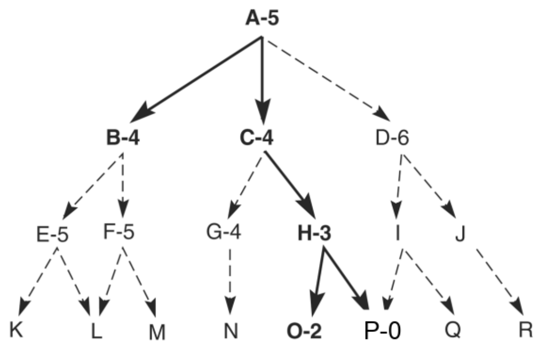
```
begin
   open := [Start];                                                  % initialize
   closed := [ ];
   while open ≠ [ ] do                                               % states remain
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS                          % goal found
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;   % loop check
               put remaining children on left end of open            % stack
            end
      end;
   return FAIL                                                       % no states left
end.
```



Again, assume **U** is the **goal state**. Note that **open** is a **stack**:

1. open = $[A_{\text{null}}]$, closed = [ ]
2. open = $[B_A\ C_A\ D_A]$, closed = $[A]$
3. open = $[E_B\ F_B\ C_A\ D_A]$, closed = $[B\ A]$
4. open = $[K_E\ L_E\ F_B\ C_A\ D_A]$, closed = $[E\ B\ A]$
5. open = $[S_K\ L_E\ F_B\ C_A\ D_A]$, closed = $[K\ E\ B\ A]$
6. open = [....], closed = [....]
7. open = [....], closed = [....]
8. open = [....], closed = [....]
9. open = [....], closed = [....]
10. open = [....], closed = [....]

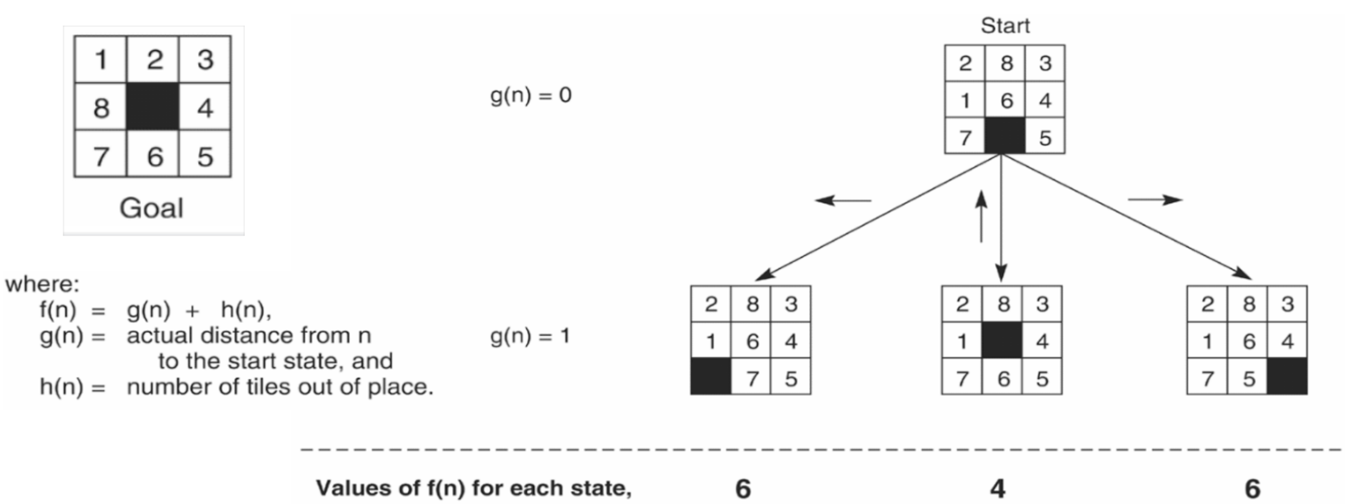
Best-First Search. Next, we try a best-first (greedy) search. We have a heuristic $h(n)$ that estimates the cost for each path. The goal is **P**. At each step, expand the node with the *lowest* cost (as predicted by the heuristic), i.e., sort the open list by $h(n)$ (smallest first):



1. open = $[A^5_{\text{null}}]$, closed = [ ]
2. open = $[B^4_A\ C^4_A\ D^6_A]$ *(random choice)*, closed = $[A]$
3. open = $[C^4_A\ E^5_B\ F^5_B\ D^6_A]$, closed = $[B\ A]$
4. open = [....], closed = [........................]
5. open = [....], closed = [........................]
6. .... ???

Finally, *extract the path to the solution* from the search result: ....

Algorithm A. Compute the next step of the Algorithm A on the 8-puzzle:



1. Pick the state with the *lowest* total cost $f(n)$
2. and compute the next possible search states, including the new values of $f(n)$, $g(n)$ and $h(n)$.