16.410-13 Recitation 2 Notes

Problem 1: Complexity of Iterative Deepening Search

Recall from the lecture notes that the (worst-case) running time of BFS was

$$T_{\text{bfs}} = 1 + b + b^2 + \cdots + b^d + (b^{d+1} - b) = O(b^{d+1}),$$

where b is the branching factor and d is the depth of the goal in the search tree.

In iterative deepening search, we perform a depth-first search for each level. Thus the lower levels of the tree, i.e., those that are closer to the root, are visited more often. More precisely, the kth level of the tree is $d + 1 - k$ times. (We denote the root note as the 0th level, the children to the root as the 1st level, and so on.) Noting that the kth level has b^k nodes, the worst-case running time of the iterative deepening search can be written as

$$\begin{aligned} T_{\text{ids}} &= (d+1)\, b^0 + (d)\, b^1 + (d-1)\, b^2 + \cdots + (d+1-k)\, b^k + \cdots + (d+1-(d-1))\, b^{d-1} + (d+1-d)\, b^d \\ &= (d+1) + d\, b + (d-1) b^2 + \cdots + 2\, b^{d-1} + b^d. \qquad (1) \end{aligned}$$

Also, multiplying this equation by b, we obtain

$$b\, T_{\text{ids}} = (d+1)\, b + d\, b^2 + (d-1)\, b^3 + \cdots + 2b^d + b^{d+1} \qquad (2)$$

Then, subtracting (1) and (2) yields

$$(b-1)\, T_{\text{ids}} = (d+1) + b + b^2 + b^3 + \cdots + b^d + b^{d+1}$$

The right hand side can be computed exactly using (finite) power series, i.e.,

$$(b-1)\, T_{\text{ids}} = d + \frac{b^{d+2} + 1}{b - 1}$$

Dividing both sides by $b - 1$,

$$T_{\text{ids}} = \frac{b^{d+2} d\, (b-1) + 1}{(b-1)^2} = O(b^d).$$

Hence, for large b, we see that the iterative-deepening search scales better than the breadth-first search.

Problem 2: Analysis of Depth-first and Breadth-first Search

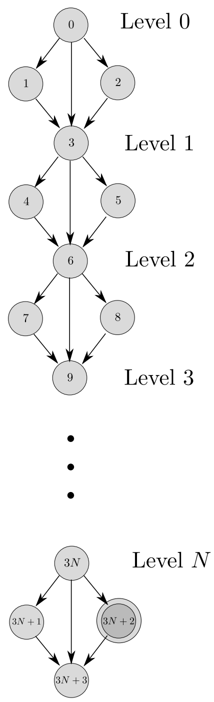

Figure 1: Graph for Problem 1. Goal vertex is marked with a double circle.

Consider the graph given in Figure 1 and derive a precise analytical expression for the following both for depth-first and for breadth-first search. In both cases, carry your analysis both when the algorithm is maintaining a *visited list* and when it is not. You should only provide upper and lower bounds for breath-first search without a visited vertices list.

i. the number of paths that are examined,

ii. the largest number of paths that will be under consideration at any given time,

iii. the length of the path returned.

Solution to Problem 1

Solution for depth-first search with no visited nodes list

Note that the depth-first search algorithm will place all the descendents of the node to the queue and expand the left-most node. In Figure 2(a), all the nodes that are expanded by the algorithm are shown. The nodes that are in the queue are shown in blue.

- The number of vertices visited is $3N + 3$.
- The largest number paths that will be in the queue is $2N + 3$.
- The path returned looks as follows:

  $$(0, 1, 3, 4, 6, 7, \cdots, 3(N-1), 3(N-1)+1, 3N, 3N+2).$$

  The length of this path is $2N + 1$.

Solution for depth-first search with visited nodes list

The paths that are in the queue are labeled in Figure 2(b). For the depth-first search the following results are obtained:

- The number of vertices visited is $3N + 3$.
- The largest number of paths in the queue is $N + 3$.
- The path returned looks as follows:

  $$(0, 3, 6, 9, \ldots, 3N, 3N + 2)$$

  Hence the length of the path is $N + 2$ (i.e., the shortest path).

Solution for the breadth-first search with visited nodes list

- The number of vertices visited is $3N + 3$.
- The largest number of paths in the queue is 3.
- The path returned looks as follows:

$$(0, 3, 6, 9, \ldots, 3N, 3N + 2)$$

Hence the length of the path is $N + 2$ (i.e., the shortest path).

Solution for the breadth-first search without visited nodes list

- The number of vertices visited is $\Theta(2^N)$.
- The largest number of paths in the queue is $\Theta(2^N)$.
- The path returned looks as follows:

$$(0, 3, 6, 9, \ldots, 3N, 3N + 2)$$

Hence the length of the path is $N + 2$ (i.e., the shortest path).

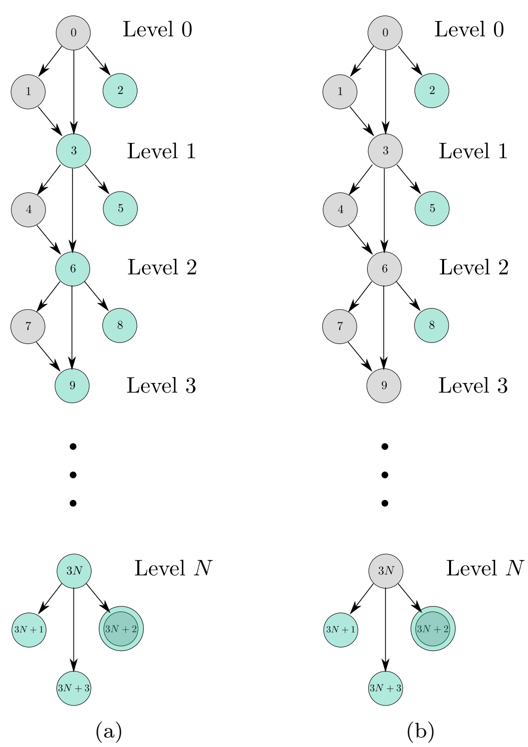

Figure 2:

16.410 / 16.413 Principles of Autonomy and Decision Making
Fall 2010