edX

Q6: A* Search: Batch Node Expansion
Problem 6: A* Search: Batch Node Expansion

Part 1

8/8 points (ungraded)

Recall that A* graph search can be implemented in pseudo-code as follows:

```
1:  function A*-GRAPH-SEARCH(problem, fringe)
2:      closed ← an empty set
3:      fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
4:      loop do
5:          if fringe is empty then return failure
6:          node ← REMOVE-FRONT(fringe)
7:          if GOAL-TEST(problem, STATE[node]) then return node
8:          if STATE[node] is not in closed then
9:              add STATE[node] to closed
10:             child-nodes ← EXPAND(node, problem)
11:             fringe ← INSERT-ALL(child-nodes, fringe)
```

Now consider the following batch version of A* graph search.
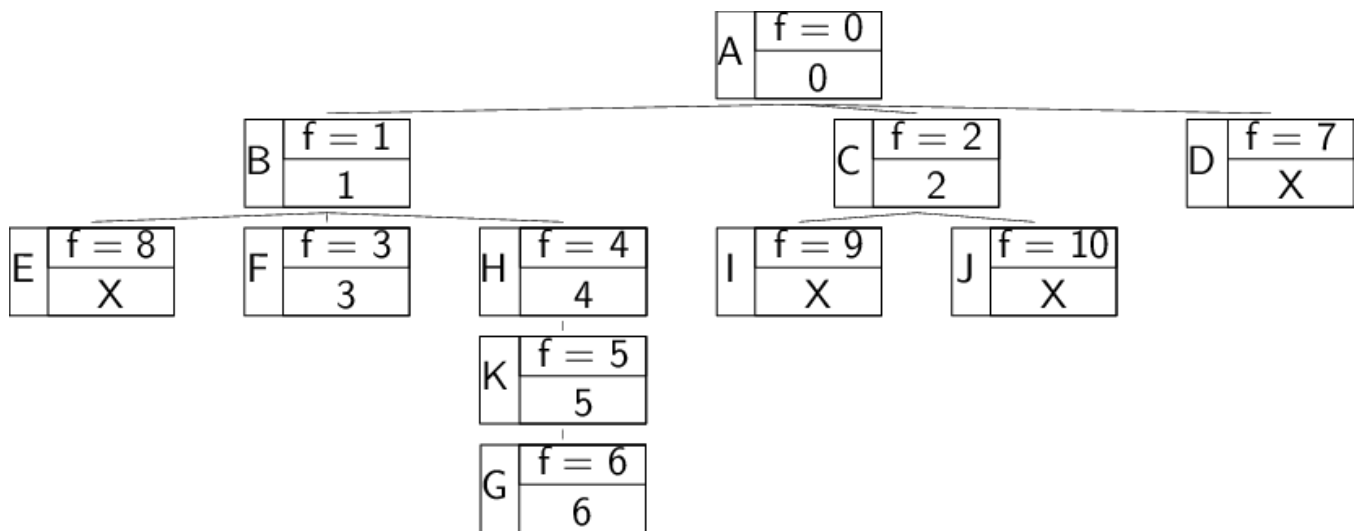
```
 1: function A*-BATCH(problem, fringe, k)
 2:     closed ← an empty set
 3:     fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
 4:     loop do
 5:         i ← 0
 6:         next-set ← an empty set
 7:         while i < k and fringe is not empty do
 8:             next-set ← INSERT(REMOVE-FRONT(fringe))
 9:             i ← i + 1
10:         for node in next-set do
11:             if GOAL-TEST(problem, STATE[node]) then
12:                 return node
13:             if STATE[node] not in closed then
14:                 closed ← INSERT(STATE[node])
15:                 child-nodes ← EXPAND(node, problem)
16:                 fringe ← INSERT-ALL(child-nodes, fringe)
```

Rather than process the nodes from the fringe one at a time, this code instead pulls k nodes off the fringe, and processes them in a random order.
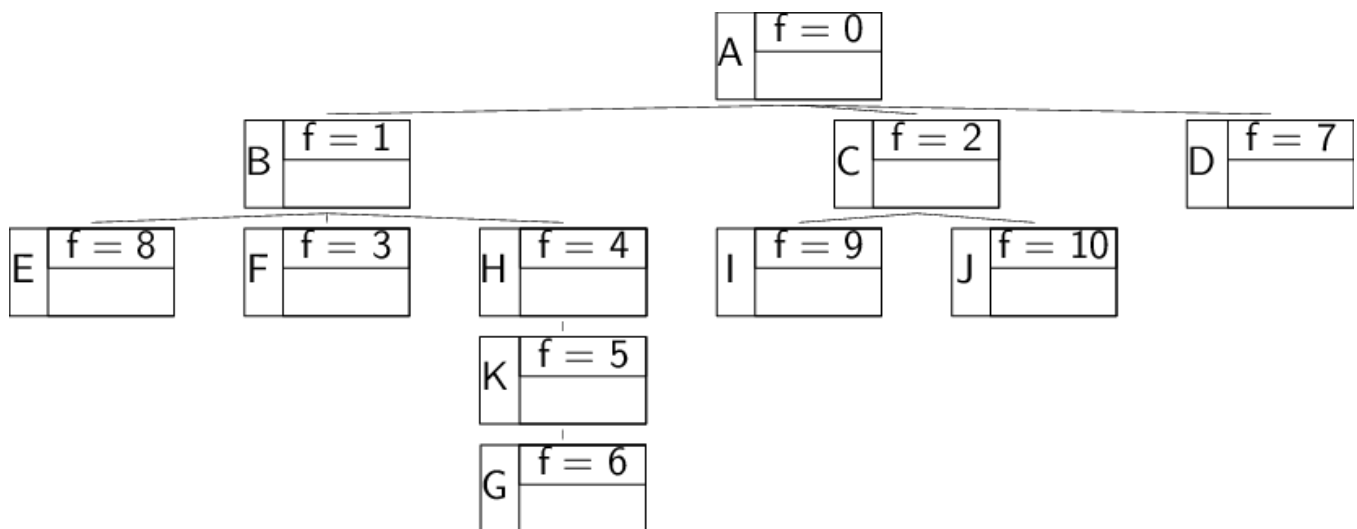
For this problem, we are interested in determining the iteration of the outermost loop at which each node is expanded.

Consider running this new A*-BATCH algorithm on a search problem with the search tree shown in the diagram below with $k = 1$. Each node is drawn with the state at the left, the f-value at the top-right $(f(n) = g(n) + h(n))$, and the iteration at which a node was expanded is at the bottom-right, with an 'X' if that node was not expanded. G is the unique goal node. In the diagram below, we can see that the start node A was expanded during iteration 0, then node B was expanded during iteration 1, node C was expanded during iteration 2, node F was expanded during iteration 3, node H was expanded during iteration 4, node K was expanded during iteration 5, and node G was expanded during iteration 6. Nodes D,E,I,J were never expanded.

Tree (top diagram):

A: f = 0, 0
B: f = 1, 1
C: f = 2, 2
D: f = 7, X
E: f = 8, X
F: f = 3, 3
H: f = 4, 4
I: f = 9, X
J: f = 10, X
K: f = 5, 5
G: f = 6, 6

In this question you'll complete similar diagrams by filling in the node expansion iterations for the cases of $k = 2$ and $k = 3$. Note that now multiple nodes can (and typically will!) be expanded during any given iteration.

For each node below, fill in the node expansion iterations for the case of $k = 2$ and fill in an 'x' for any node that is not expanded.

Tree (second diagram, to fill):

A: f = 0
B: f = 1
C: f = 2
D: f = 7
E: f = 8
F: f = 3
H: f = 4
I: f = 9
J: f = 10
K: f = 5
G: f = 6

A: 0 ✔
B: 1 ✔
C: 1 ✔
D: 3 ✔

E: [x] ✔

F: [2] ✔

H: [2] ✔

I: [x] ✔

J: [x] ✔

K: [3] ✔

G: [4] ✔

[Submit]

✔ Correct (8/8 points)

Part 2

8/8 points (ungraded)

For the same tree, fill in the node expansion iterations for the case of $k = 3$ and fill in an 'x' for any node that is not expanded.

A: [0] ✔

B: [1] ✔

C: [1] ✔

D: 1 ✔

E: 2 ✔

F: 2 ✔

H: 2 ✔

I: 3 ✔

J: 3 ✔

K: 3 ✔

G: 4 ✔

Submit

✔ Correct (8/8 points)