PS3.6 Notes: Planning in a Grid World
MS Branicky
2023-01-31

In lecture, we covered the following search algorithm:

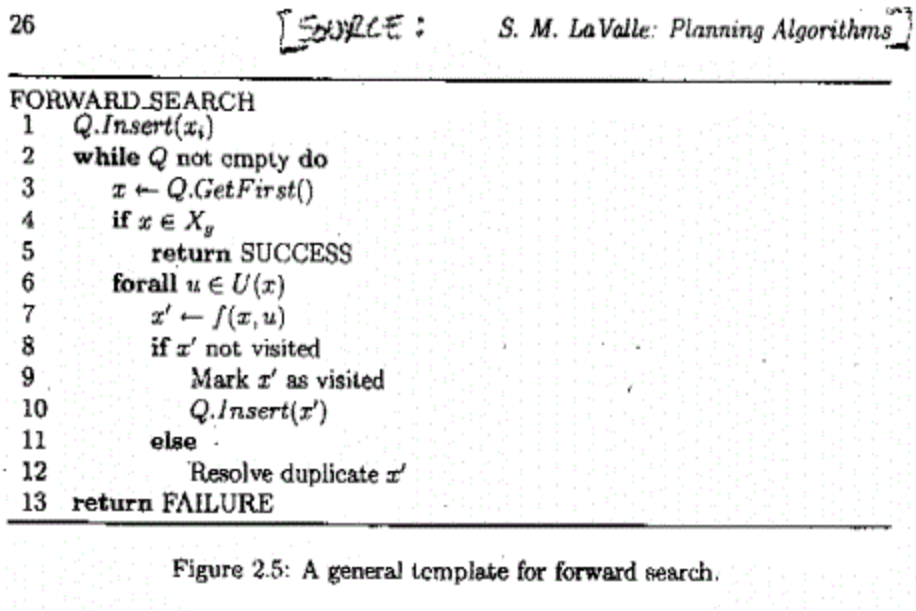
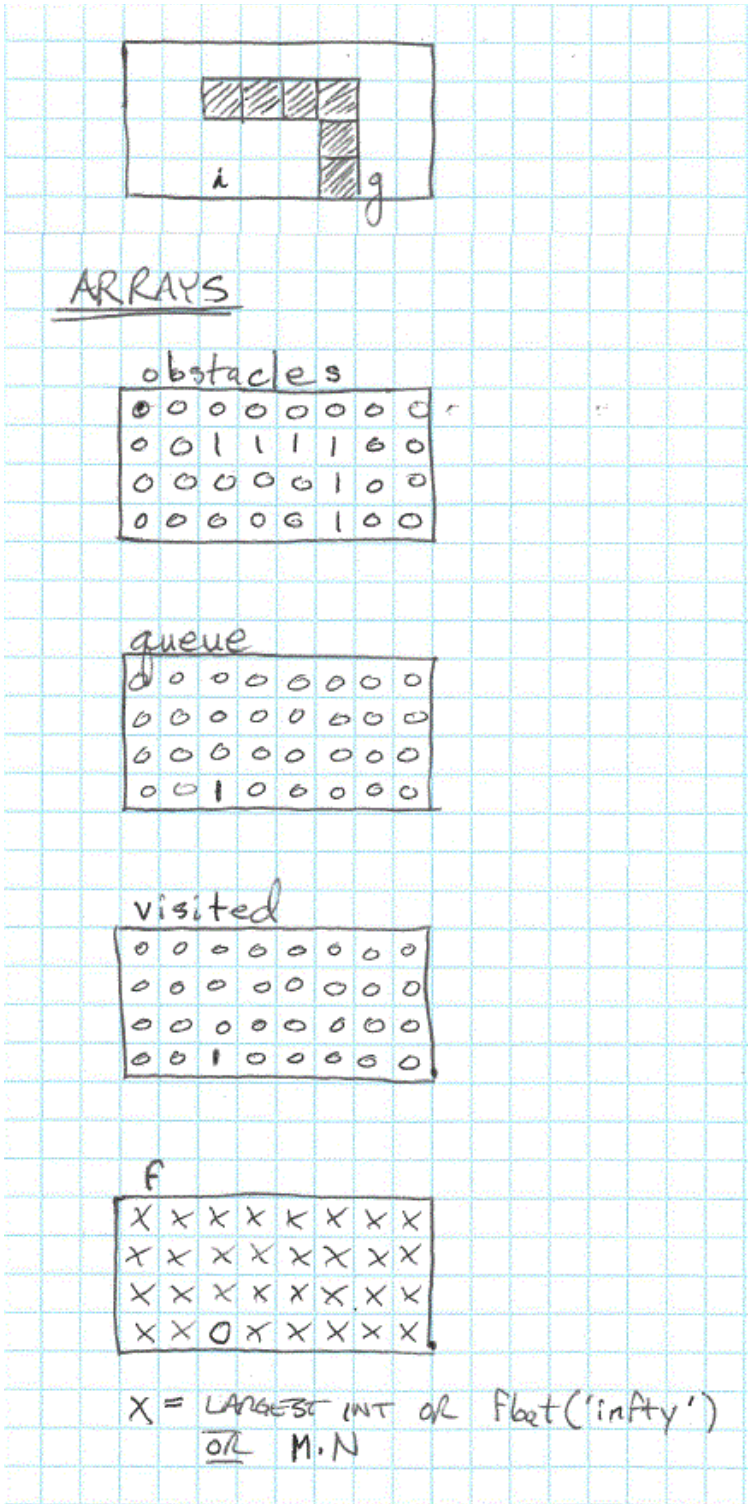


Figure 2.5: A general template for forward search.

We also covered the Swapping Counters notebook, which included this Python implementation of the above in the case of BFS, which follows the above almost line-by-line:

```
# Forward-Search Loop [LIST IMPLEMENTATION SHOWN]
while len(newreach)>0:
  b = newreach.pop(0)
  if b == goal:
    print("goal found: moves=", reachable[b], ", elapsed=",
time()-start_time)
    break
  for move in possible_moves(b):
    if move not in reachable:
      newreach.append(move)
      reachable[move] = reachable[b]+1
    else:  # resolve (f(n)=depth(n) would never be revised for BFS)
      if reachable[b]+1 < reachable[move]:
        print("revised", b, move, reachable[b]+1, reachable[move] )
        reachable[move] = reachable[b]+1
```

You can also implement the algorithm without lists, priority queues, etc. using arrays like this:




```
FORWARD_SEARCH
1   Q.Insert(x_i)
2   while Q not empty do
3       x ← Q.GetFirst()
4       if x ∈ X_g
5           return SUCCESS
6       forall u ∈ U(x)
7           x' ← f(x, u)
8           if x' not visited
9               Mark x' as visited
10              Q.Insert(x')
11          else
12              Resolve duplicate x'
13  return FAILURE
```

To the left, f is an array holding the total cost f(n) for node n

Adding one for g(n) can help for A* and UCS, too!

In any case, think about …

… How do you perform Q.insert in this case?

… How do you perform GetFirst?