18: Problem solving via search and matrices
Lecture 18: Problem solving via search and matrices

- Problem solving via search
- Matrices
- Assignment 9
Search algorithm

Keep track of a list of states that we could visit; we’ll call it `to_visit`.

General idea:

- take a state off the `to_visit` list
- if it’s the goal state
  - we’re done!
- if it’s not the goal state
  - Add all of the next possible states to the `to_visit` list
  - repeat
Search algorithms

- add the start state to to_visit

- Repeat
  - take a state off the to_visit list
  - if it’s the goal state
    - we’re done!
  - if it’s not the goal state
    - Add all of the next possible states to the to_visit list

- Depth first search (DFS): to_visit is a stack

- Breadth first search (BFS): to_visit is a queue
Implementing the state space

‣ What the “world” looks like.

‣ We’ll define the world as a collection of discrete states.

‣ States are connected if we can get from one state to another by taking a particular action.

‣ The set of all possible states is called the state space.
Implementing the state space

‣ What the “world” looks like.
  ‣ We’ll define the world as a collection of discrete states.
  ‣ States are connected if we can get from one state to another by taking a particular action.
  ‣ The set of all possible states is called the state space.

‣ State:
  ‣ Is this the goal state? (is_goal function)
  ‣ What states are connected to this state? (next_states function)
Search variants implemented

- add the start state to to_visit
- Repeat
  - take a state off the to_visit list
  - if it’s the goal state
    - we’re done!
  - if it’s not the goal state
    - Add all of the next possible states to the to_visit list

```python
def dfs(start_state):
    s = Stack()
    return search(start_state, s)

def bfs(start_state):
    q = Queue()
    return search(start_state, q)

def search(start_state, to_visit):
    to_visit.add(start_state)
    while not to_visit.is_empty():
        current = to_visit.remove()
        if current.is_goal():
            return current
        else:
            for s in current.next_states():
                to_visit.add(s)
    return None
```
In what order would this variant visit the states?

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
        return None
```

Order: 1, 2, 5
In what order would this variant visit the states?

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
    return None
```

Order: 1, 2, 5, 3, 6, 9, 7, 8
In what order would this variant visit the states?

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
        return None
```

- Order: 1, 2, 5, 3, 6, 9, 7, 8
- What search algorithm is this?
In what order would this variant visit the states?

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
    return None
```

- Order: 1, 2, 5, 3, 6, 9, 7, 8
- DFS!
DFS with a stack

```python
def dfs(start_state):
    s = Stack()
    return search(start_state, s)

def search(start_state, to_visit):
    to_visit.add(start_state)

    while not to_visit.is_empty():
        current = to_visit.remove()

        if current.is_goal():
            return current
        else:
            for s in current.next_states():
                to_visit.add(s)

    return None

Order: 1, 4, 3, 8, 7, 6, 9, 2, 5
One last DFS variant

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
        return None

def dfs(state):
    if state.is_goal():
        return [state]
    else:
        result = []
        for s in state.next_states():
            result += dfs(s)
        return result
```

How is this different?
One last DFS variant

```python
def search(state):
    if state.is_goal():
        return state
    else:
        for s in state.next_states():
            result = search(s)
            if result != None:
                return result
        return None

def dfs(state):
    if state.is_goal():
        return [state]
    else:
        result = []
        for s in state.next_states():
            result += dfs(s)
        return result
```

> Return ALL solutions found, not just one.
Lecture 18: Problem solving via search and matrices

- Problem solving via search
- Matrices
- Assignment 9
What is a matrix?

A matrix is a two-dimensional structure, e.g.,

```
0 1 0
1 8 2
5 0 3
```

- It has rows and columns.
- The second row is: 1 8 2
- The second column is:
  1
  8
  0

Since we are computer scientists, we'll start indexing at 0. That means that the first row is row 0 and the first column is column 0.
Indexing into matrices

- Individual entries in a matrix can be referenced by specifying a row and a column.

```
0 1 0
1 8 2
5 0 3
```

- Let's say that the matrix above is called $m$, what entry does $m[1][2]$ represent?
  - In math, we might write this as $m(1, 2)$.
  - $1 =$ second row, $2 =$ third column, that is $m[1][2]$ is 2.

- How would we get at the 3 in the above matrix?
  - $m[2][2]$
Implementing matrices in Python

- We can use lists of lists!

```python
>>> m = [[0, 1, 0], [1, 8, 2], [5, 0, 3]]
>>> m
[[0, 1, 0], [1, 8, 2], [5, 0, 3]]
```
```python
>>> m[1][2]
2
>>> m[2][2]
3
```

- Could also have constructed this as:

```python
>>> m = []
>>> m.append([0,1,0])
>>> m.append([1,8,2])
>>> m.append([5,0,3])
>>> m
[[0, 1, 0], [1, 8, 2], [5, 0, 3]]
```
```python
>>> m[1][2]
2
>>> m[2][2]
3
```
Implementing matrices in Python

- what does m[1] represent?
  - the second row!

```python
>>> m[1]
[1, 8, 2]
```

- matrices are just lists of lists.
what do zero_matrix and zero_matrix2 do?

- They both create a size x size matrix with all entries zero.
- zero_matrix does this an entry at a time.
- zero_matrix2 does this a row at a time.

```python
>>> zero_matrix(3)
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> zero_matrix2(2)
[[0, 0], [0, 0]]
>>> zero_matrix(1)
[[0]]
>>> m = zero_matrix(2)
>>> m
[[0, 0], [0, 0]]
>>> m[1][1] = 100
>>> m
[[0, 0], [0, 100]]
```
MATRICES

matrix.py

- what does random_matrix do?
  - It creates a size x size matrix with random ints between 0 and size x size

```python
>>> random_matrix(3)
[[6, 2, 1], [2, 6, 1], [0, 3, 9]]
>>> random_matrix(3)
[[5, 3, 9], [7, 4, 1], [8, 2, 3]]
>>> random_matrix(3)
[[6, 9, 7], [8, 4, 7], [1, 6, 5]]
```
How would we print out a matrix in a more normal form (one row at a time)?
- Iterate through the rows and print each out.
- Look at the `print_matrix` and `print_matrix2` function.

What does the identity function do?
- It creates an identity size by size matrix with all zeros except for ones along the diagonal.

How would we sum up all the numbers in a matrix?
- Iterate over each entry and add them up.
- Look at the `matrix_sum` function.

What does `len(m)` give us?
- The number of rows (remember, list of lists).

What does `len(m[row])` give us?
- The number of columns (in that row, technically).

Look at the `matrix_sum2` and `matrix_sum3` functions.
- They use the `sum` function to sum up each row and then add that to the total.
copying matrices

- Be careful when you want to create a deep copy of a matrix. See the code below. What’s the problem?

```python
>>> m = [[1, 2], [3, 4]]
>>> n = m[:]
>>> n[0][0] = 0
>>> n
[[0, 2], [3, 4]]
>>> m
[[0, 2], [3, 4]]
```
copying matrices

- If you want to copy a matrix and avoid aliasing issues, you should either:
  - use the copy module
    ```python
    import copy
    copy.deepcopy(m)
    ```
  - or by creating a deep copy of each row and appending it to a new list.
    ```python
    >>> m = [[1, 2], [3, 4]]
    >>> n = []
    >>> for row in m:
    ...     n.append(row[:])
    ...
    >>> n
    [[1, 2], [3, 4]]
    >>> n[0][0] = 0
    >>> n
    [[0, 2], [3, 4]]
    >>> m
    [[1, 2], [3, 4]]
    ```
How would you represent a tic tac toe board?

- As a 3 by 3 matrix.
- Each entry has one of three values:
  - empty
  - X
  - O
Lecture 18: Problem solving via search and matrices

- Problem solving via search
- Matrices
- Assignment 9
N-queens problem

- Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

Solution(s)?
N-queens problem

- Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.
N-queens problem

- Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

Solution(s)?
N-queens problem

- Place N queens on an N by N chess board such that none of the N queens are attacking any other queen.

- How do we solve this with search:
  - What is a state?
  - What is the start state?
  - What is the goal?
  - How do we transition from one state to the next?
Search algorithm

- add the start state to to_visit
- Repeat
  - take a state off the to_visit list
  - If it’s the goal state
    - we’re done!
  - If it’s not the goal state
    - Add all of the next possible states to the to_visit list
- Any problem that we can define these three things can be plugged into the search algorithm!
Resources

- `search_variants.py`
- `matrix.py`
- `tic_tac_toe.py`

Homework

- `Assignment 9`