

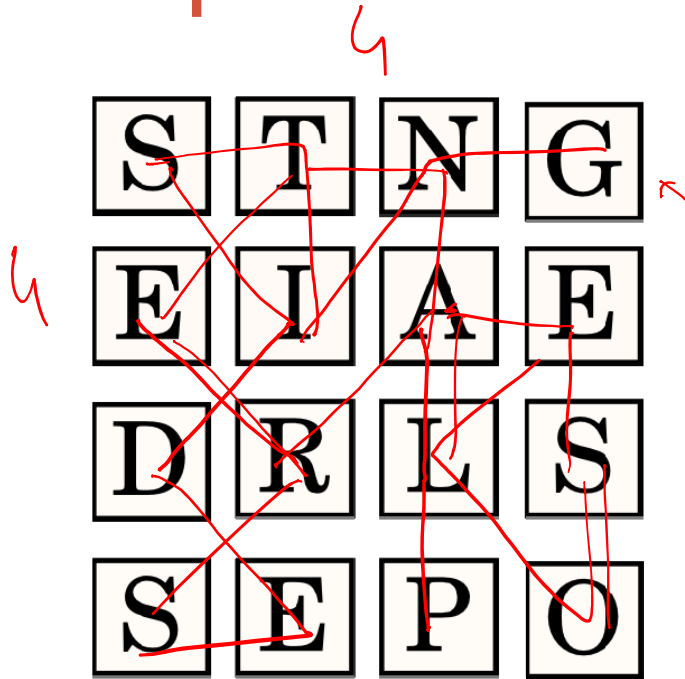
# CSE 100: PA4, BOGGLE

---

# Announcements

- PA4 writeup and skeleton are now available
- Please read the writeup thoroughly —
- Challenge: SETUP and DESIGN is harder than previous Pas
- We have given you a basic design to get you started (and finished with the checkpoint) quickly
- Checkpoint due next week, Wed May 20<sup>th</sup> at 10:00PM
  - Start today

# Graphs in PA4 (HINT!!)



Checkpoint :  
Implement

- setBoard(): Given a 2D array of strings populate the board
- isOnBoard(): Given a word, check if it is on the board

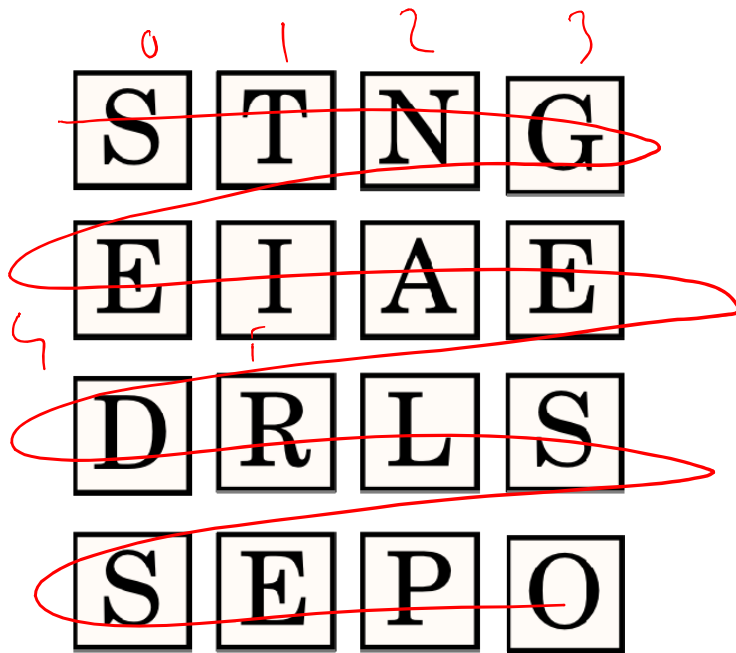
The game of Boggle:

- Two players, one board (MxN) dies
- Each die face contains a string
- Player that constructs the maximum number of unique words is the winner

Rules of the game:

- Words are constructed by traversing a sequence of adjacent dies (an acyclic path starting at any dice)
- Two dice are adjacent if they are next to each other horizontally, vertically, or diagonally.
- Die can only be used once in a word (acyclic path)

# Graphs in PA4 (HINT!!)



This is the boggle board... how would you represent it in your code?

A. As a linked list

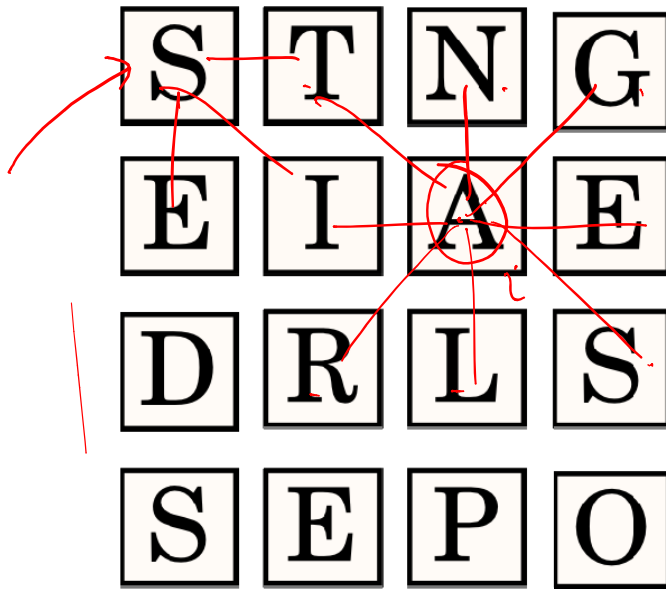
B. As a 2D array ←

C. As a tree ←

D. Graph ←

E. A vector ←

# Graphs in PA4 (HINT!!)



This is also a graph... where are the edges? (fill them in)

# Graphs in PA4 (HINT!!)

|   |   |   |   |
|---|---|---|---|
| S | T | N | G |
| E | I | A | E |
| D | R | L | S |
| S | E | P | O |

This is also a graph... where are the edges? (fill them in)

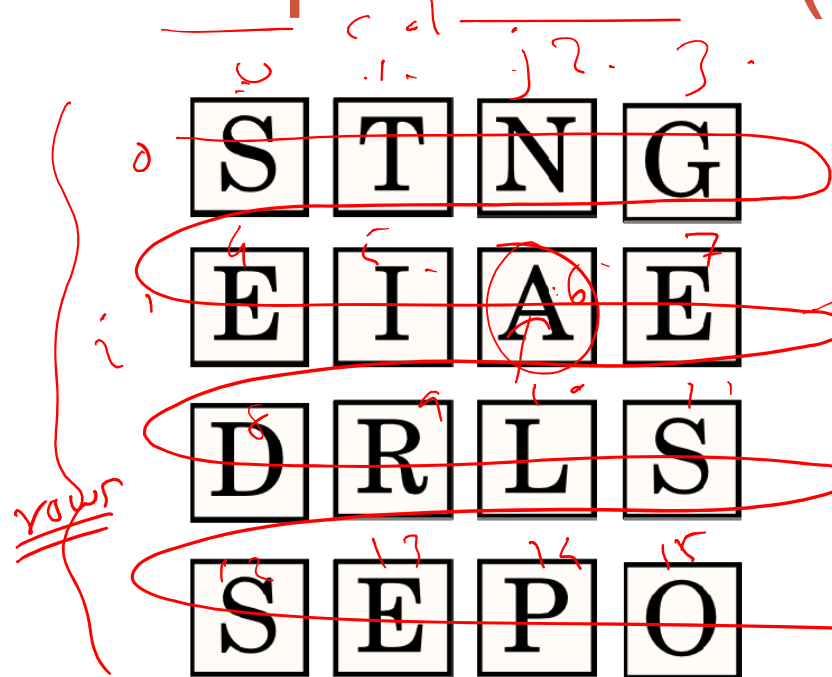
$i$

$i-5, i-4, i-3$

$i-1, i+1$

$i+3, i+4, i+5$

# Graphs in PA4 (HINT!!)



```
/**
 * Vector representing the boggle board.
 */
std::vector<BoardPos> board;
//Elements are in row major order
```

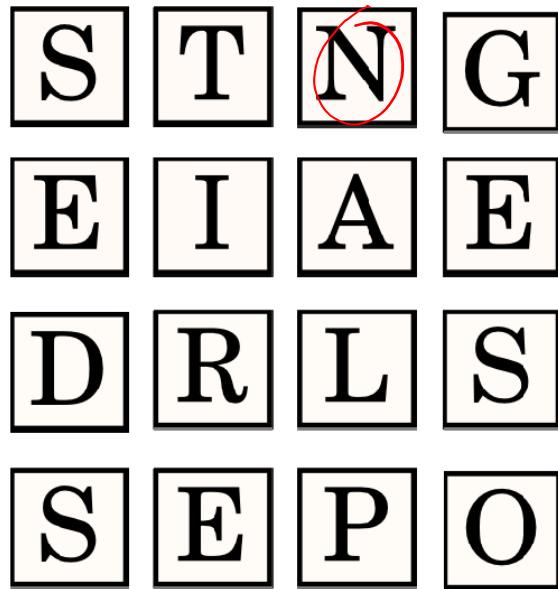
If row is the number of rows and col is the number of columns, how would you access the element in the ith row and jth column of the 2D array using the vector representation

$$i \times \text{col} + j$$

$$4 + 2 = 6$$

- A. board[i+j]
- B. board[i\*col+j]
- C. board[j\*col+i]
- D. board[i\*row+j]
- E. board[j\*row+i]

## PA4: BogglePlayer

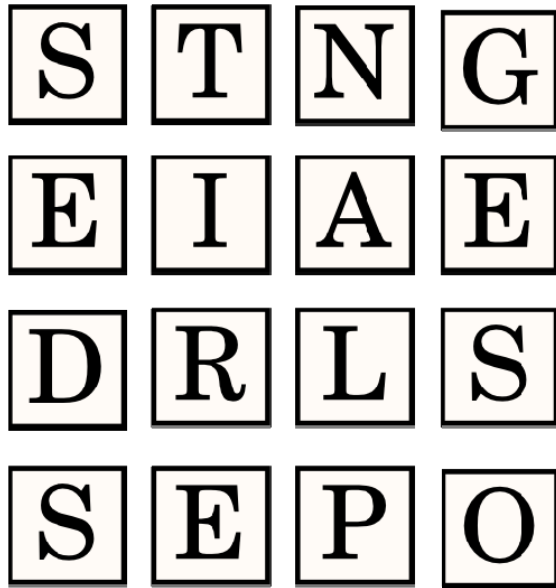


```
class BoardPos {  
    public:  
        BoardPos(const std::string &text);  
        const std::string &getText() const;  
        bool getVisited() const;  
        void setVisited(bool visited);  
    private:  
        const std::string text;  
        bool visited;  
};
```

Why do we have the visited field?

---

## PA4: BogglePlayer



Which of the following algorithms best applies to our problem i.e searching for a word on the board?

- A. BFS
- ☒ B. DFS
- C. Dijkstra's
- D. Prims

```
std::vector<int> BogglePlayer::isOnBoard(const std::string &word_to_check)
```

# PA4: isOnBoard

Graph Traversal here we come!

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

```

DFS(G,v) ( v is the vertex where the search starts )
  Stack S := {}; ( start with an empty stack )
  for each vertex u, set visited[u] := false;
  push S, v;
  while (S is not empty) do
    u := pop S;
    if (u is not visited) then
      visited[u] := true;
      for each unvisited neighbour w of u
        push S, w;
      end if
    end while
  end DFS()
  
```

What would we be finding by running (vanilla) DFS on this graph, starting with source node C<sub>5</sub> (assume the index of each visited node is stored in the order in which nodes are visited)?

- A. All sequences of words on the board starting with C
- ☒ B. A sequence that consists of all letters on the board starting with C
- C. A sequence that consists of some of the letters on the board starting with C
- D. None of the above

# PA4: isOnBoard

(h, v, word)

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

DFS(G,v) ( v is the vertex where the search starts )  
 Stack S := {}; ( start with an empty stack )  
 for each vertex u, set visited[u] := false;  
 push S, v;  
 while (S is not empty) do  
   u := pop S;  
   if (NOT u.visited) then  
 set u.visited:=true;  
 for each unvisited neighbour w of u  
   push S, w;  
   end if  
 end while  
 END DFS()

What is the simplest change you can make to DFS to get us one step closer to our solution?

# PA4: isOnBoard

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

```

DFS(G,v) ( v is the vertex where the search starts )
  Stack S := {}; ( start with an empty stack )
  for each vertex u, set visited[u] := false;
  push S, v;
  while (S is not empty) do
    u := pop S;
    if (NOT u.visited) then
      set u.visited:=true;
      for each unvisited neighbour w of u
        push S, w;
      end if
    end while
  END DFS()

```

What is the simplest change you can make to DFS to get us one step closer to our solution?

- Pass in the word that we are searching for 
- In our traversal only go down paths that match with the next letter in the word that we are searching for 

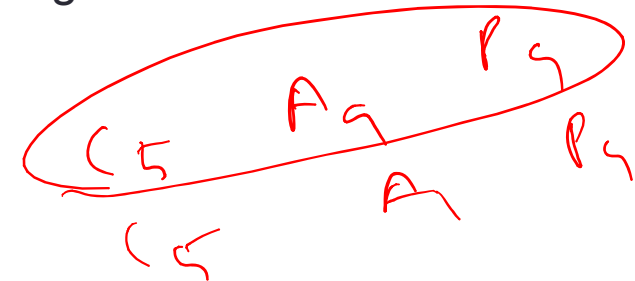
**isOnBoard**(G,v, word) ( v is the vertex where the search starts,  
word is the word we are looking for )

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

```

Stack S := {}; ( start with an empty stack ), pos:=0
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if (NOT u.visited AND (u.letter == word[pos]))
    pos:=pos+1
    u.visited := true;
    for each unvisited neighbour w of u
      push S, w;
    end if
  end while
END DFS()

```



What is the sequence of visited vertices if the word that I am trying to find is CAP and the source vertex is C<sub>5</sub>

- ☒ A. C<sub>5</sub> A<sub>1</sub> P<sub>4</sub>  
☐ B. C<sub>5</sub> P<sub>4</sub> A<sub>1</sub>  
☐ C. Cannot say



**isOnBoard**(G,v, **word** ) ( v is the vertex where the search starts,  
**word is the word we are looking for** )

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

Stack S := {}; ( start with an empty stack ), **pos:=0**

for each vertex u, set visited[u] := false;

push S, v;

while (S is not empty) do

u := pop S;

if (NOT u.visited **AND** u.letter == **word[pos]**)

**pos:=pos+1**

u.visited := true;

for each unvisited neighbour w of u

push S, w;

end if

end while

END DFS()

What is the sequence of visited vertices if the word that I am trying to find is CAP and the source vertex is C<sub>5</sub>

- A. C<sub>5</sub> A<sub>1</sub> P<sub>4</sub>
- B. C<sub>5</sub> P<sub>4</sub> A<sub>1</sub>
- C. Cannot say

Am I doing unnecessary extra work at any point?

**isOnBoard**(G,v, **word** ) ( v is the vertex where the search starts,  
**word is the word we are looking for** )

|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

Stack S := {}; ( start with an empty stack ), **pos:=0**

for each vertex u, set visited[u] := false;

push S, v;

while (S is not empty) do

u := pop S;

if (NOT u.visited **AND** u.letter == **word[pos]**)

u.visited := true;

**pos:=pos+1**

**if pos==length(word)**

**return**

for each unvisited neighbour w of u

push S, w;

end if

end while

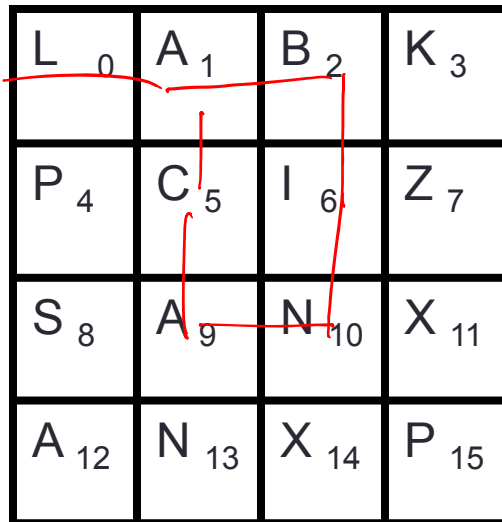
END DFS()

Am I done? What happens if I search for “CANIBAL”?

A. We can always find it without any problem

**B.** We may or may not find it

## Need Backtracking!



|                 |                 |                 |                 |
|-----------------|-----------------|-----------------|-----------------|
| L <sub>0</sub>  | A <sub>1</sub>  | B <sub>2</sub>  | K <sub>3</sub>  |
| P <sub>4</sub>  | C <sub>5</sub>  | I <sub>6</sub>  | Z <sub>7</sub>  |
| S <sub>8</sub>  | A <sub>9</sub>  | N <sub>10</sub> | X <sub>11</sub> |
| A <sub>12</sub> | N <sub>13</sub> | X <sub>14</sub> | P <sub>15</sub> |

```

isOnBoard(G,v, word ) ( v is the vertex where the search starts,
                        word is the word we are looking for )
Stack S := {}; ( start with an empty stack ), pos:=0
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if (NOT u.visited AND u.letter ==word[pos])
    u.visited := true;
    pos:=pos+1
    if pos==length(word)
      return
    for each unvisited neighbour w of u
      push S, w;
  end if
end while
END DFS()
  
```

## Need Backtracking!

|                     |                     |                     |                 |
|---------------------|---------------------|---------------------|-----------------|
| L <sub>0</sub><br>✓ | A <sub>1</sub><br>✓ | B <sub>2</sub><br>✓ | K <sub>3</sub>  |
| P <sub>4</sub>      | C <sub>5</sub><br>✓ | I <sub>6</sub>      | Z <sub>7</sub>  |
| S <sub>8</sub>      | A <sub>9</sub>      | N <sub>10</sub>     | X <sub>11</sub> |
| A <sub>12</sub>     | N <sub>13</sub>     | X <sub>14</sub>     | P <sub>15</sub> |

**isOnBoard**(G,v, **word**) ( v is the vertex where the search starts,  
**word is the word we are looking for** )

```

Stack S := {}; ( start with an empty stack ), pos:=0
for each vertex u, set visited[u] := false;
push S, v;
while (S is not empty) do
  u := pop S;
  if u.revisit == true
    u.revisit := false;
    u.visited := false;
    pos:=pos-1
  else if (NOT u.visited AND u.letter ==word[pos])
    u.visited := true;
    pos:=pos+1
    if pos==length(word)
      return
    u.revisit := true;
    push S, u
    for each unvisited neighbour w of u
      push S, w;
    end if
  end while
END DFS()

```

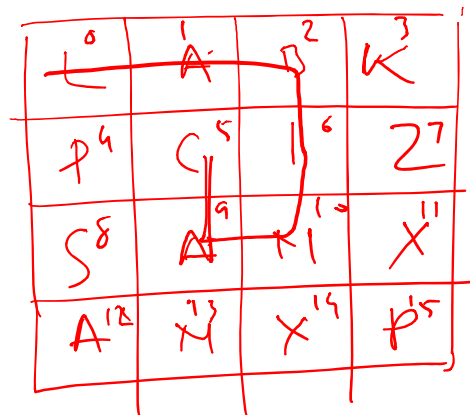
CAP

$(\begin{pmatrix} \checkmark & p \\ A & A \end{pmatrix} \textcircled{3})$   
 $p \circ \gamma =$

```

5
5 C 0 0 0
5109864210
0 L 1 0 0
510986421
1 A 1 0 0
5109864216420
0 L 2 0 0
510986421642
2 B 2 0 0
51098642164
4 P 2 0 0
5109864216
6 I 2 0 0
510986421
1 A 2 1 1
51098642
2 B 1 0 0
5109864
4 P 1 0 0
510986
6 I 1 0 0
51098
8 S 1 0 0
5109
9 A 1 0 0
510914131210864
4 P 2 0 0
51091413121086
6 I 2 0 0
5109141312108
8 S 2 0 0
510914131210
10 N 2 0 0
5109141312101514131176
6 I 3 0 0
5109141312101514131176117321
1 A 4 0 0
510914131210151413117611732
2 B 4 0 0
510914131210151413117611732731
1 A 5 0 0
51091413121015141311761173273140
0 L 6 0 0

```



CANIBAL

5 10 9