Chapter 3 Stacks（2）
Stack Applications

College of Computer Science, CQU

Application 4
Evaluating RPN Expressions

- RPN?

  A notation for arithmetic expressions:

  operators are written after the operands.

- Infix notation: operators written between the operands
- Postfix notation (RPN): operators written after the operands
- Prefix notation : operators written before the operands
- Examples:

INFIX	RPN (POSTFIX)	PREFIX
A + B	A B +	+ A B
A * B + C	A B * C +	+ * A B C
A * (B + C)	A B C + *	* A + B C
A–(B –(C – D))	A B C D - - -	- A - B - C D
A – B – C – D	A B - C - D -	- - - A B C D

Application 4
Evaluating RPN Expressions

- "By hand": Underlining technique:

1. Scan the expression from left to right to find an operator.
2. Locate ("underline") the last two preceding operands and combine them using this operator.
3. Repeat until the end of the expression is reached.

- Example:

2 3 4 + 5 6 - - *

→ 2 3 4 + 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 5 6 - - *

→ 2 7 -1 - *

→ 2 7 -1 - *

→ 2 8 * → 2 8 * → 16

Application 4
Evaluating RPN Expressions

- STACK ALGORITHM

Receive:An RPN expression.

Return: A stack whose top element is the value of RPN expression(unless an error occurred).

1. Initialize an empty stack.
2. Repeat the following until the end of the expression is encountered:

a. Get next token (constant, variable, arithmetic operator) in the RPN expression.

b. If token is an operand, push it onto the stack.

If it is an operator, then

(i) Pop top two values from the stack.

If stack does not contain two items, error due to a malformed RPN Evaluation terminated

(ii) Apply the operator to these two values.

(iii) Push the resulting value back onto the stack.

3. When the end of expression encountered, its value is on top of the stack (and, in fact, must be the only value in the stack).

Application 4
Evaluating RPN Expressions

- Unary minus causes problems

Example:

5 3 - -

→ 5 3 - -

→ 5 -3 - → 8

5 3 - -

→ 5 3 - -

→ 2 - → -2

Use a different symbol:

5 3 ~ -

5 3 - ~

Application 4
Evaluating RPN Expressions

- Example: 2 3 4 + 5 6 - - *

Push 2

Push 3

Push 4

Read +

    Pop 4, Pop 3, 3 + 4 = 7

    Push 7

Push 5

Push 6

Read –

    Pop 6, Pop 5, 5 - 6 = -1

    Push -1

Read –

    Pop -1, Pop 7, 7 - -1 = 8

    Push 8

Read *

    Pop 8, Pop 2, 2 * 8 = 16

Application 5
Converting Infix to RPN

1. Initialize an empty stack of operators.

2. While no error has occurred and end of infix expression not reached

a. Get next Token in the infix expression.

b. If Token is

(i) a left parenthesis: Push it onto the stack.

(ii) a right parenthesis:Pop and display stack elements until a left parenthesis is encountered,

but do not display it. (Error if stack empty with no left parenthesis found.)

(iii) an operator:If stack empty or Token has higher priority than top stack element, push Token on stack.(Left parenthesis in stack has lower priority than operators)

Otherwise, pop and display the top stack element; then repeat the comparison of Token with new top stack item.

(iv) an operand: Display it.

3. When end of infix expression reached,pop and display stack items until stack is empty.

Application 5
Converting Infix to RPN

- **Example: ((A+B)*C)/(D-E)**

Push (	
Push (	
Display A	**A**
Push +	
Display B	**AB**
Read)	
Pop +, Display +, Pop (	**AB+**
Push *	
Display C	**AB+C**
Read)	
Pop *, Display *, Pop (	**AB+C* // stack now empty**
Push /	
Push (	
Display D	**AB+C*D**
Push –	
Display E	**AB+C*DE**
Read)	
Pop -, Display -, Pop (	**AB+C*DE-**
Pop /, Display /	**AB+C*DE-/**

Application 5 Converting Infix to RPN

```
void convertingInfixToRPN(const string& exp)
{
    Stack<char>* op = new Stack<char>();
    char ch;
    for(int i=0; i<exp.length; i++) {
        switch(exp[i]){
            case '(':   op->push(exp[i]);
                        break;
            case ')':   while( (ch=op->pop()) != '(' )  print(ch);
                        break;
            case '*':
            case '/':   while(op->length()>0 && (op->topValue()=='*' || op->topValue()=='/') )
                            print(op->pop());
                        op->push(exp[i]);
                        break;
            case '+':
            case '-':   while( op->length()>0 && op->topValue()!='(' ) print(op->pop());
                        op->push(exp[i]);
                        break;
            default:    print(exp[i]);
        }
    }
    while (op->length()>0) print(op->pop());
}
```

Application 5’
Converting RPN to Infix

- **Example: AB+C*DE-/**

Push A

Push B

Pop B, Pop A, push (A+B)

Push C

Pop C, Pop (A+B), push ((A+B)*C)

Push D

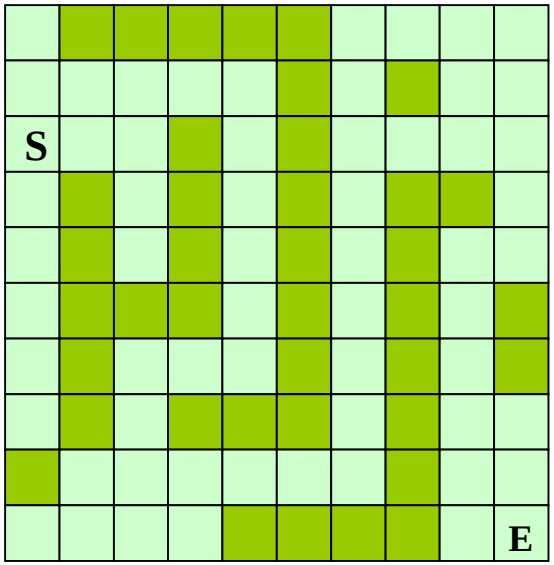
Push E

Pop E, Pop D, push (D-E)

Pop (D-E), Pop ((A+B)*C), push (((A+B)*C)/(D-E))

Application 6
Maze

- A maze is a rectangular area with an entrance and an exit.
- The interior of a maze contains walls or obstacles that one cannot walk through.
- We view the maze as broken up into equal size squares, some of which are part of the walls and the others are part of the hallways. Thus they are "open".
- One of the open squares is designated as the Start position and another as the Exit position.



Maze M

Application 6
Maze

- So we will use an enumerated type which will include constants OPEN and WALL.

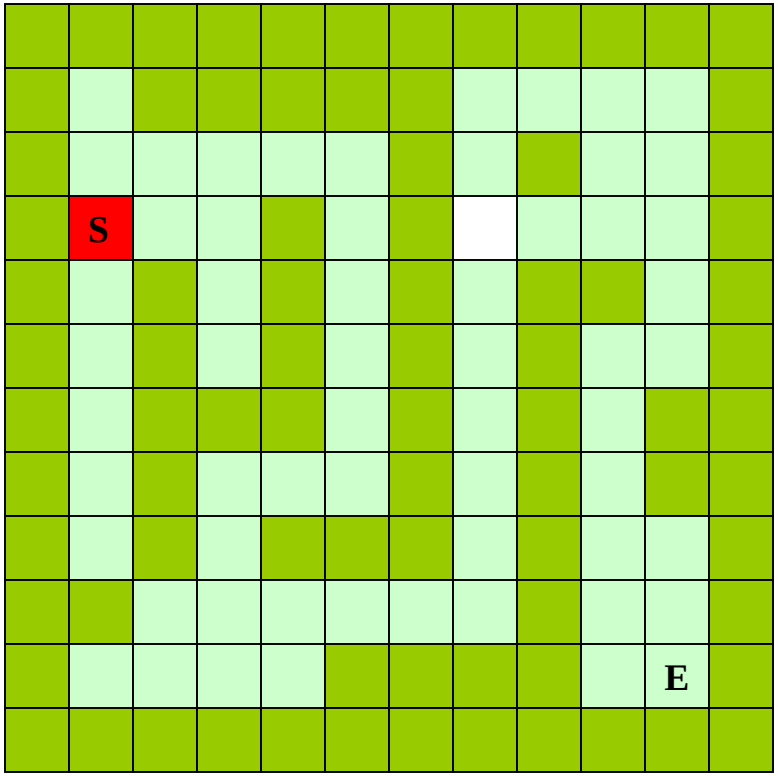
- To reflect our search method for finding an exit path, we will also have a

  third state: VISITED.

- In traversing the maze, we can only move forward into an OPEN position .

- Thus a given position has 4 possible neighboring positions in the directions east, south, west and north.

Searching the Example Maze

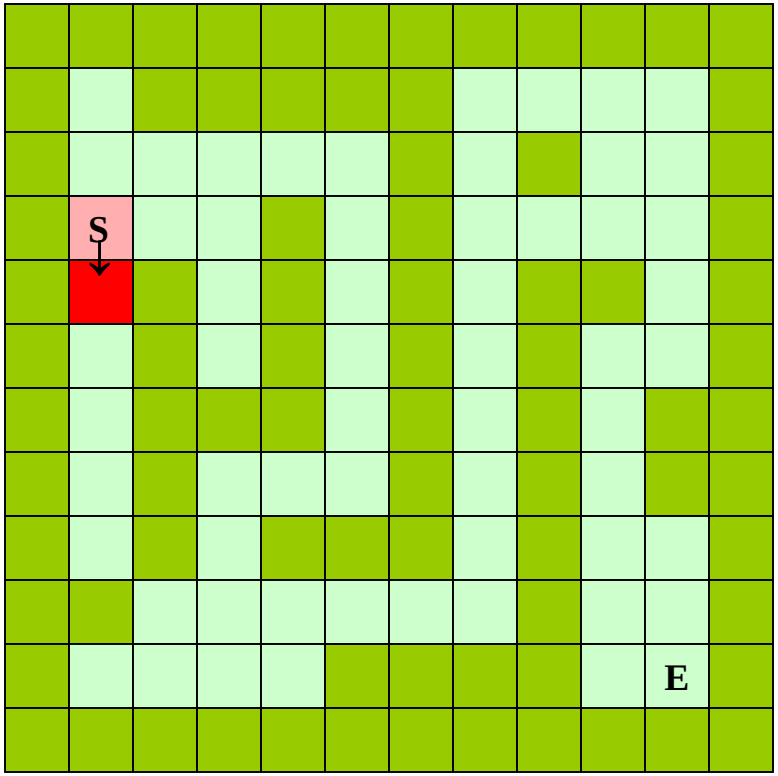


Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



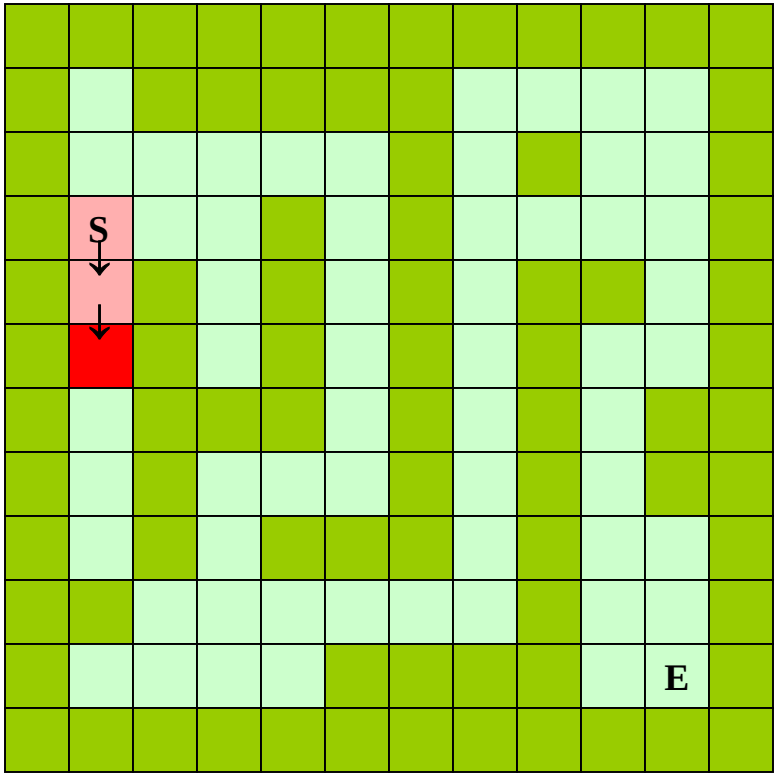
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



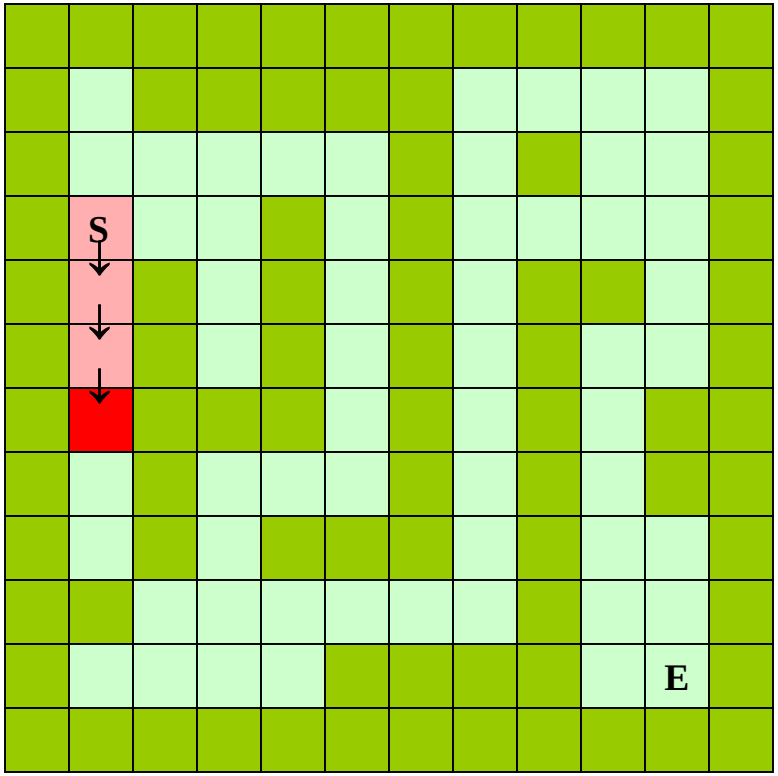
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



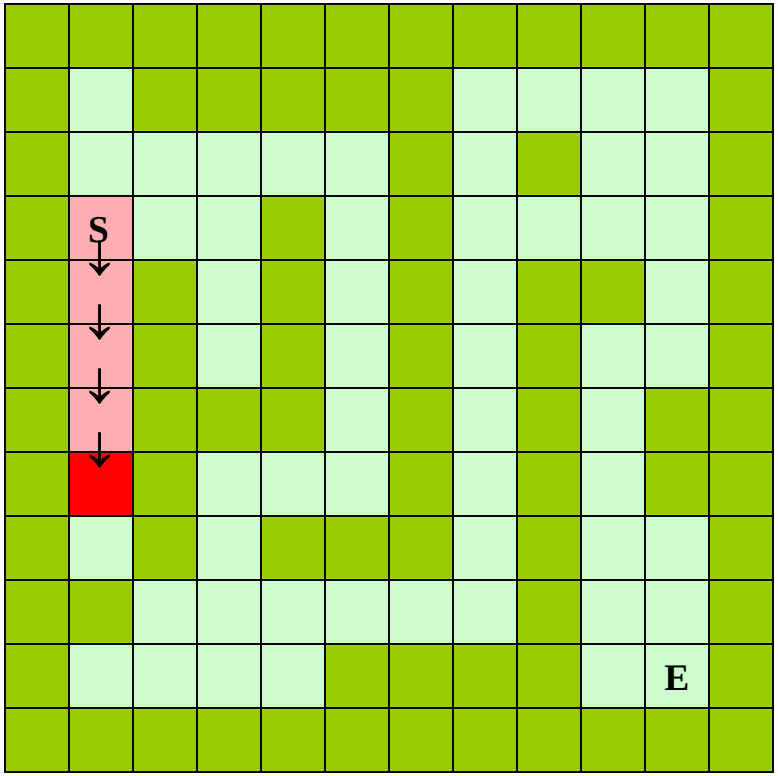
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



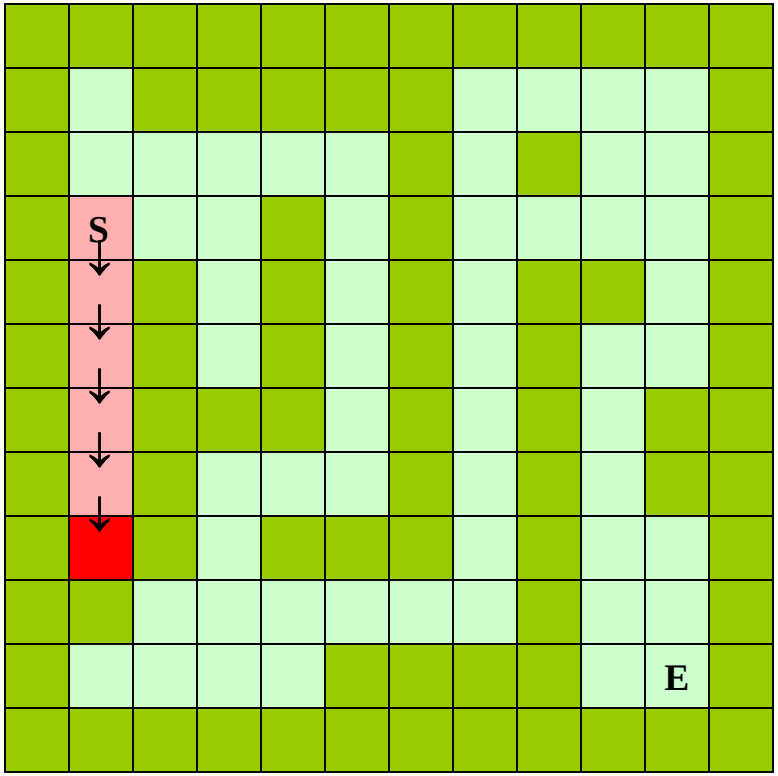
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



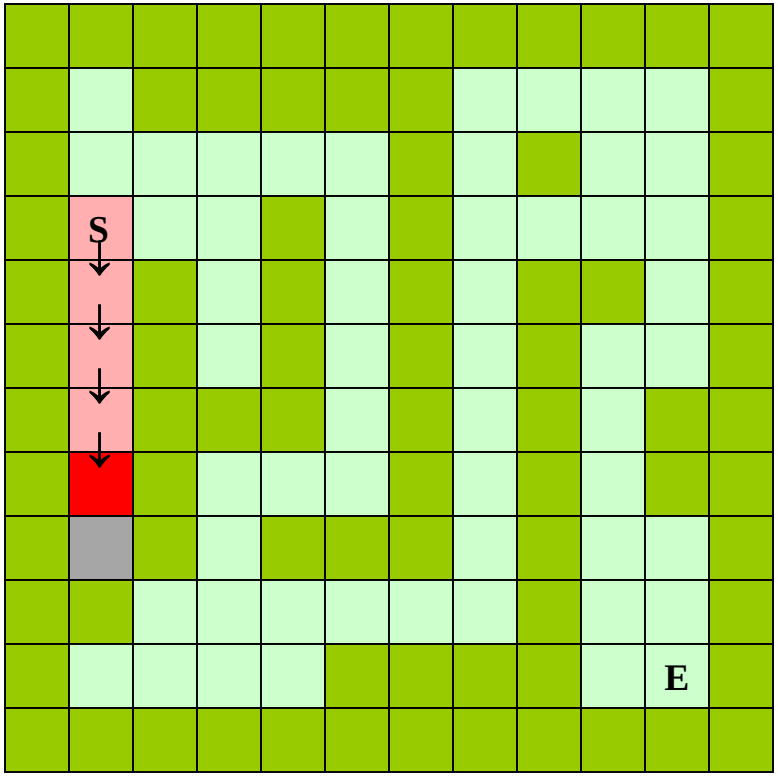
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



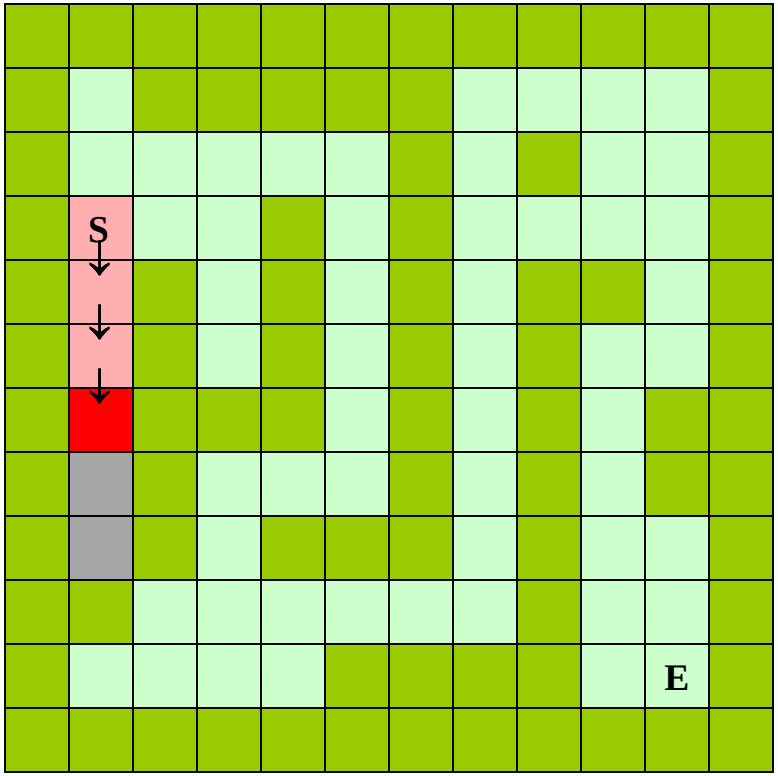
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



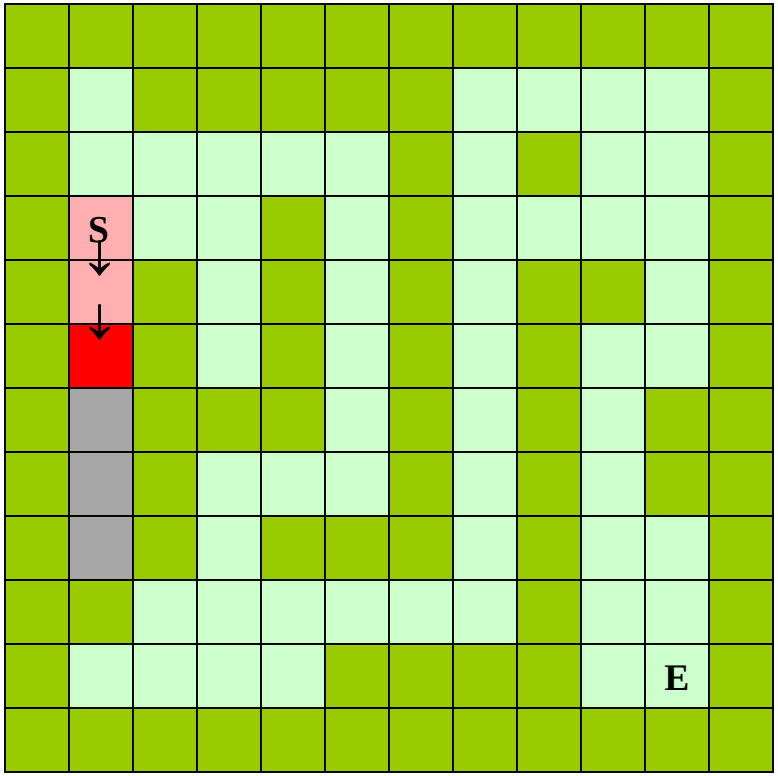
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



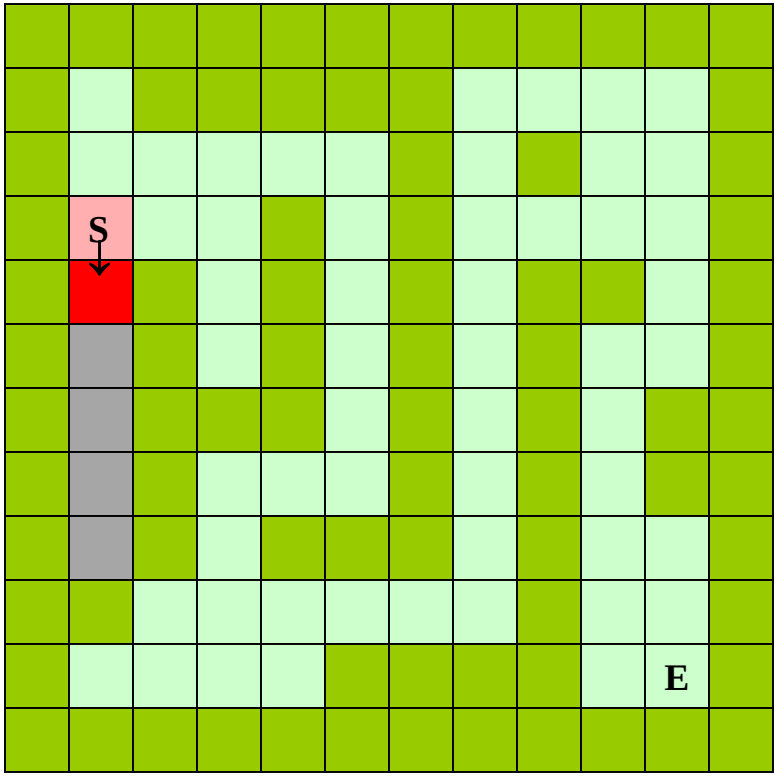
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



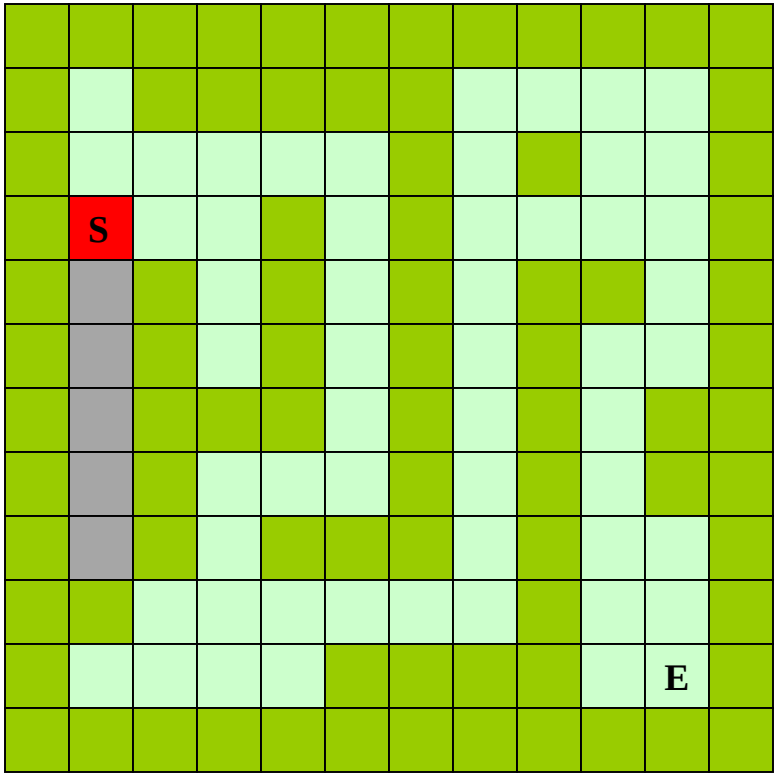
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



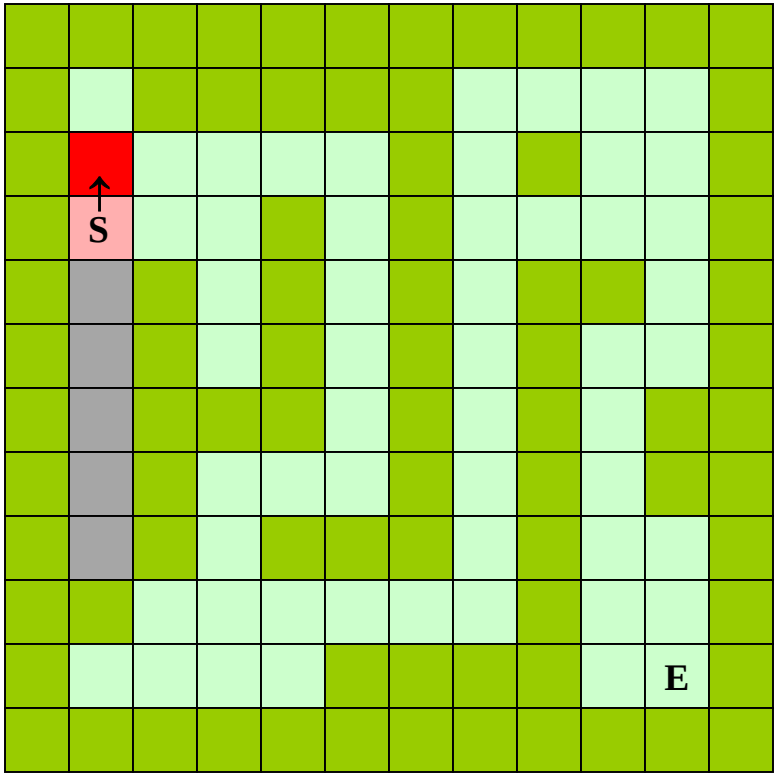
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



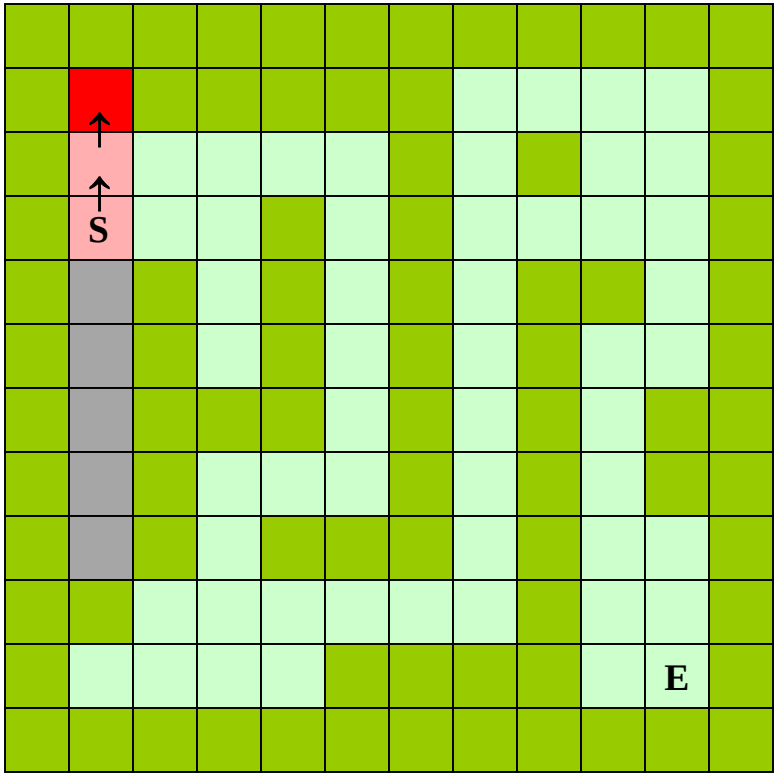
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



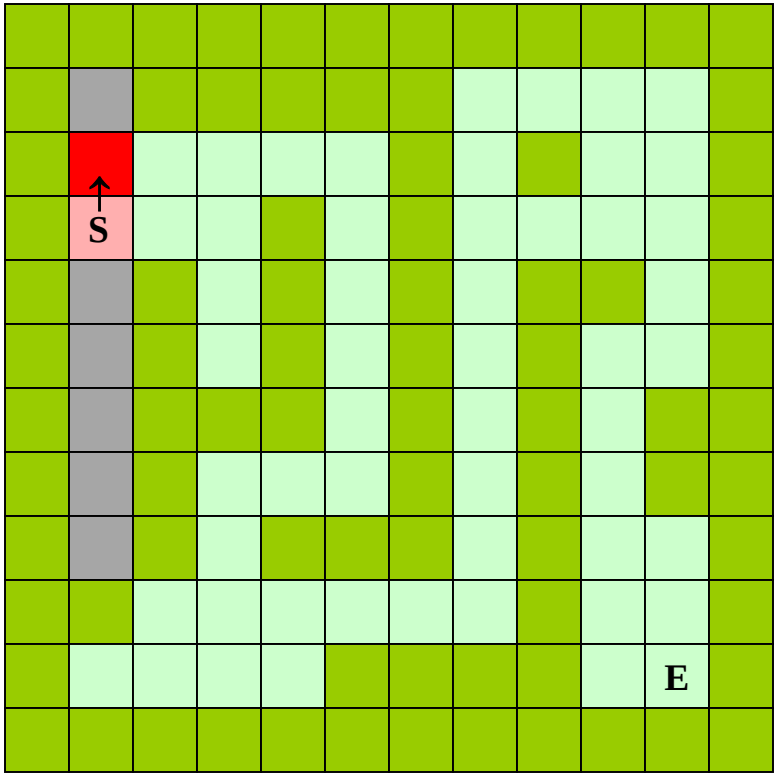
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



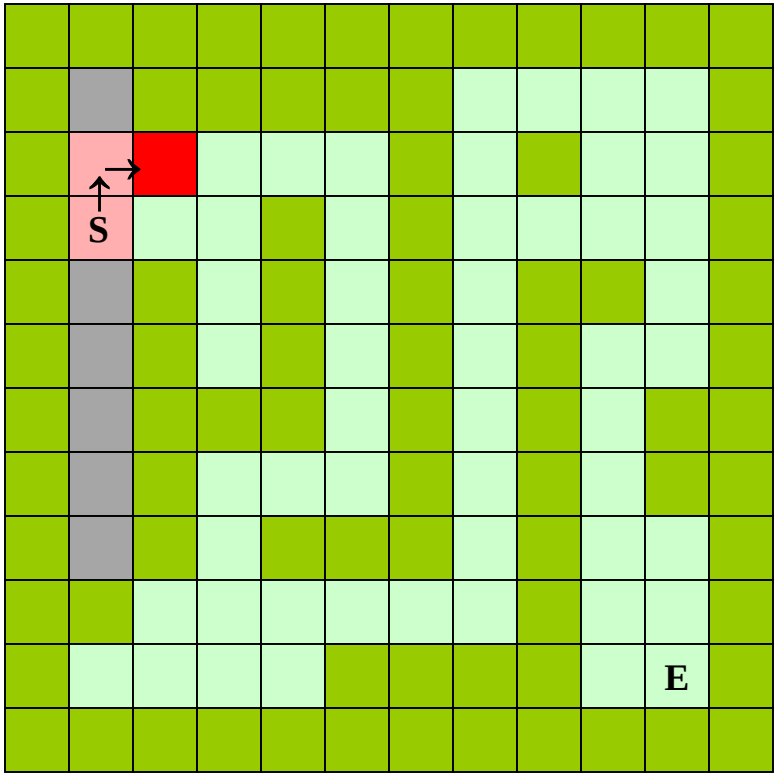
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



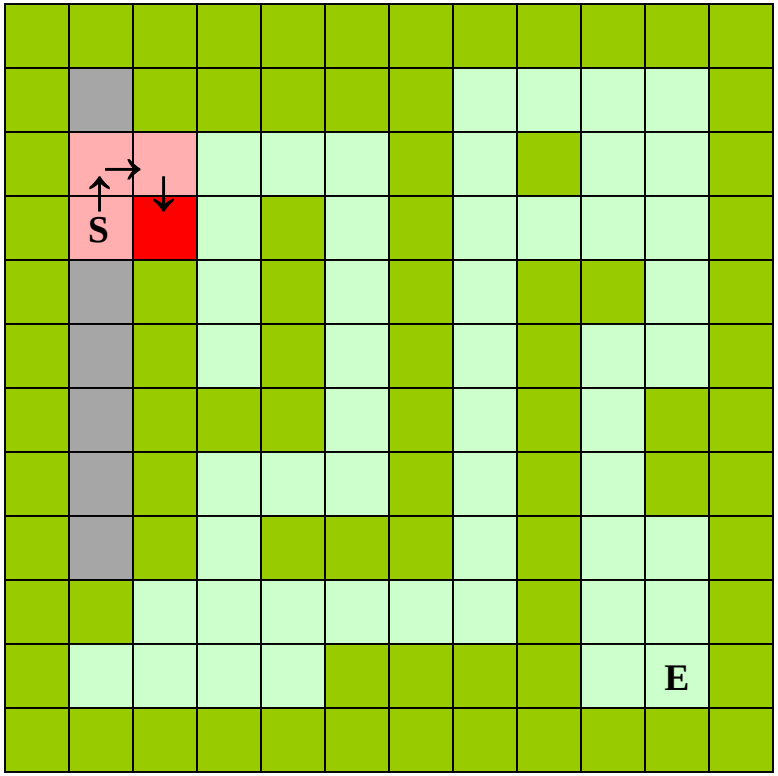
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



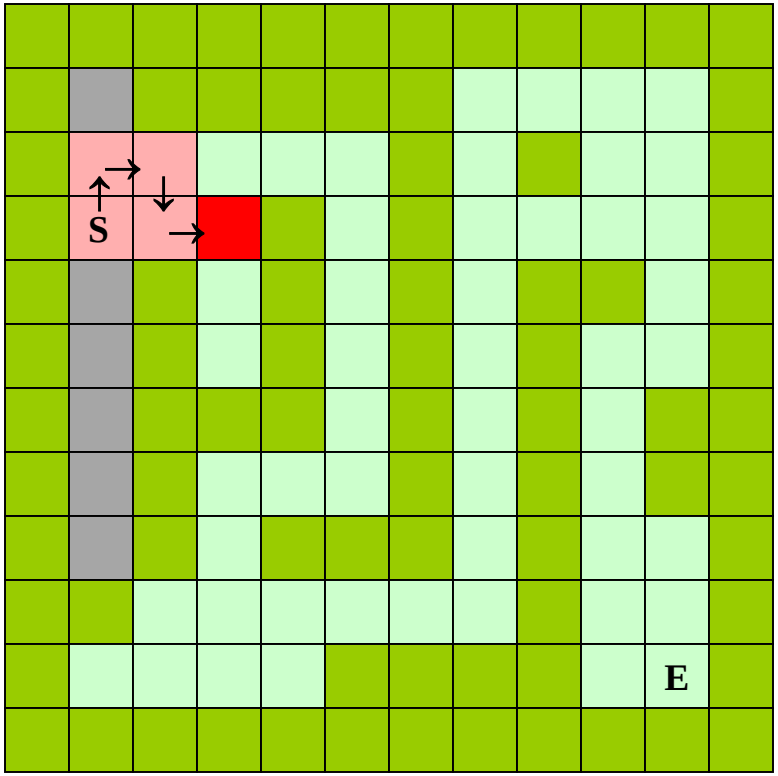
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



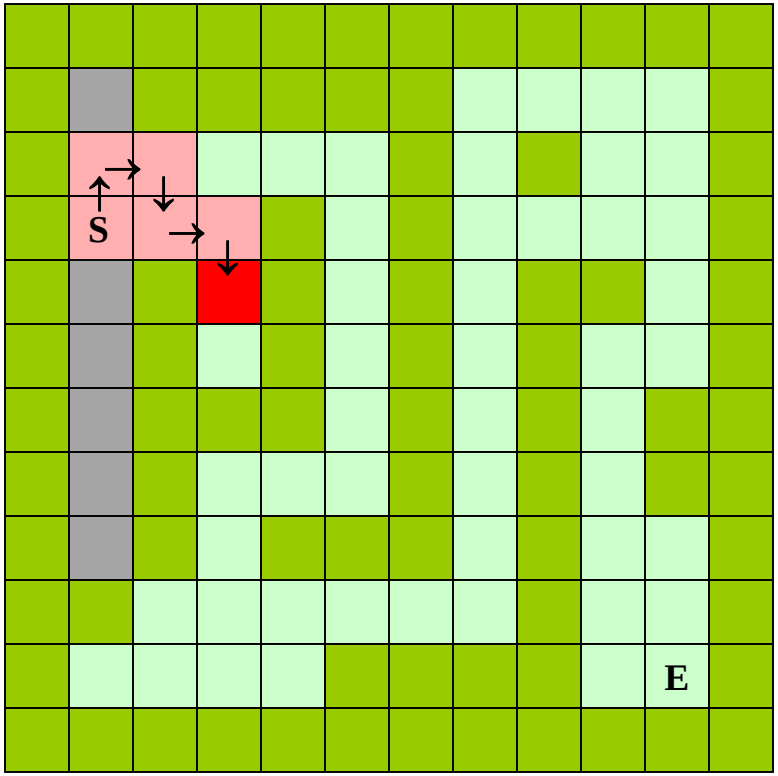
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



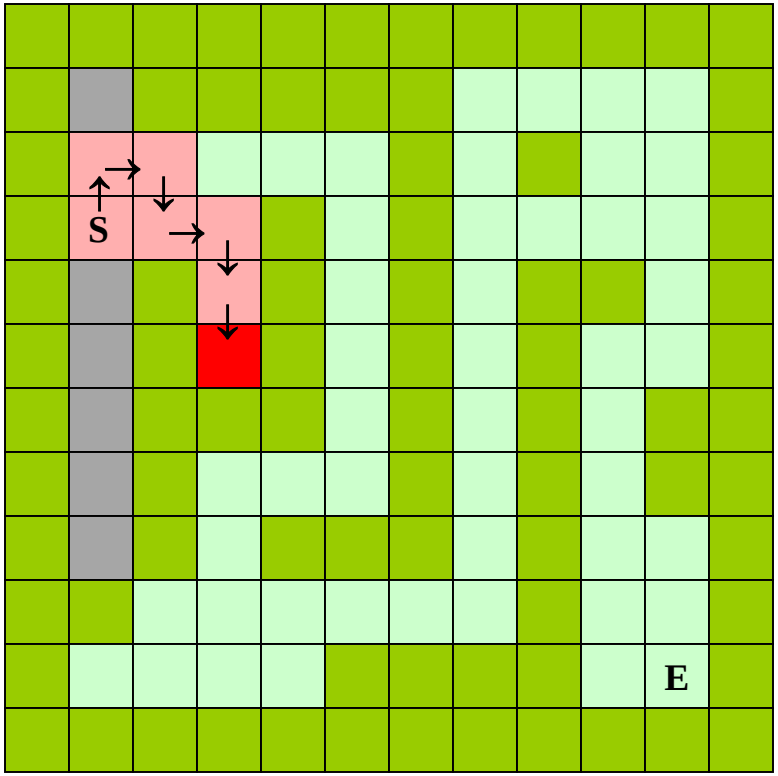
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



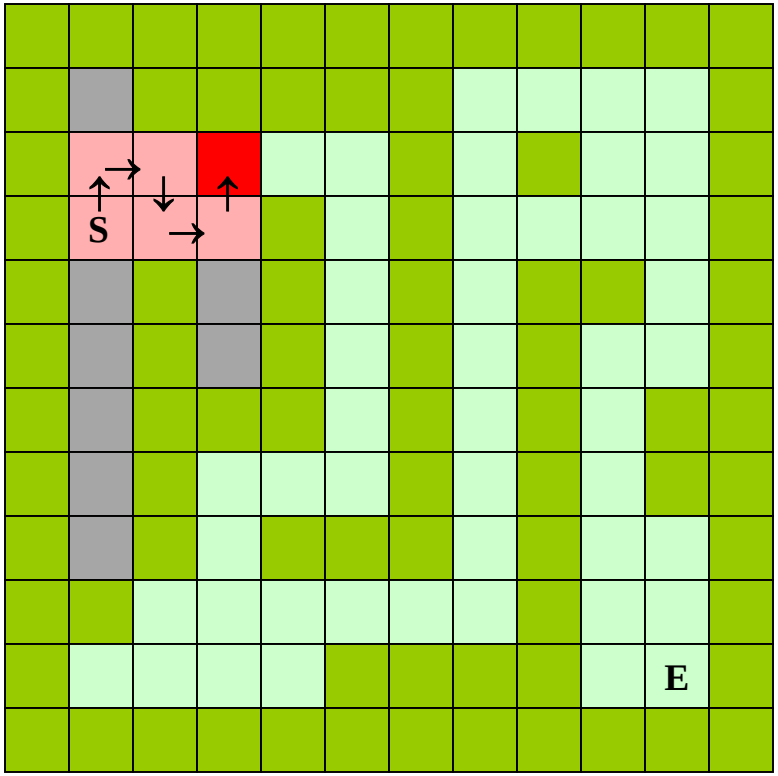
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



Three steps combined

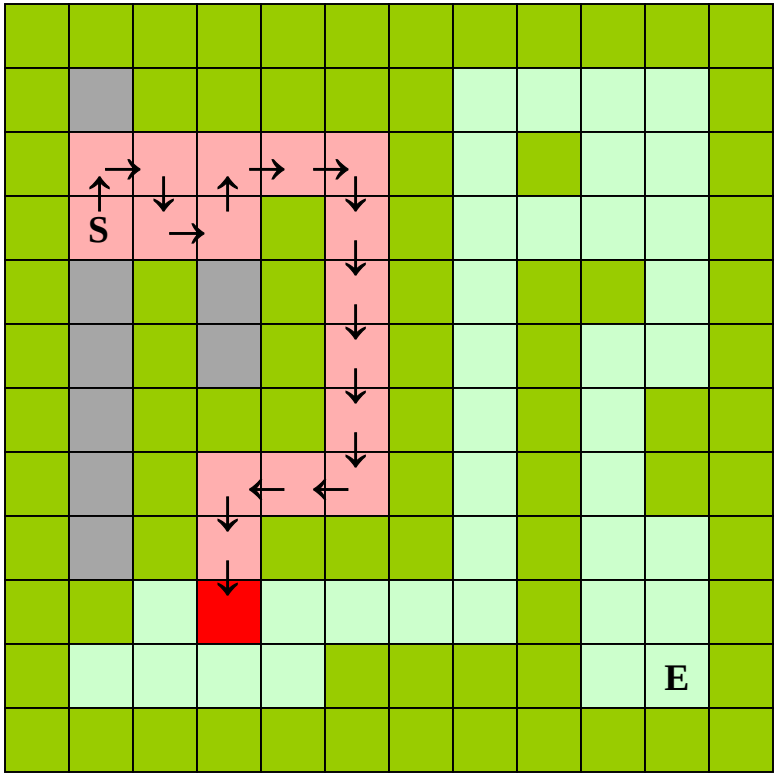
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



Several steps combined

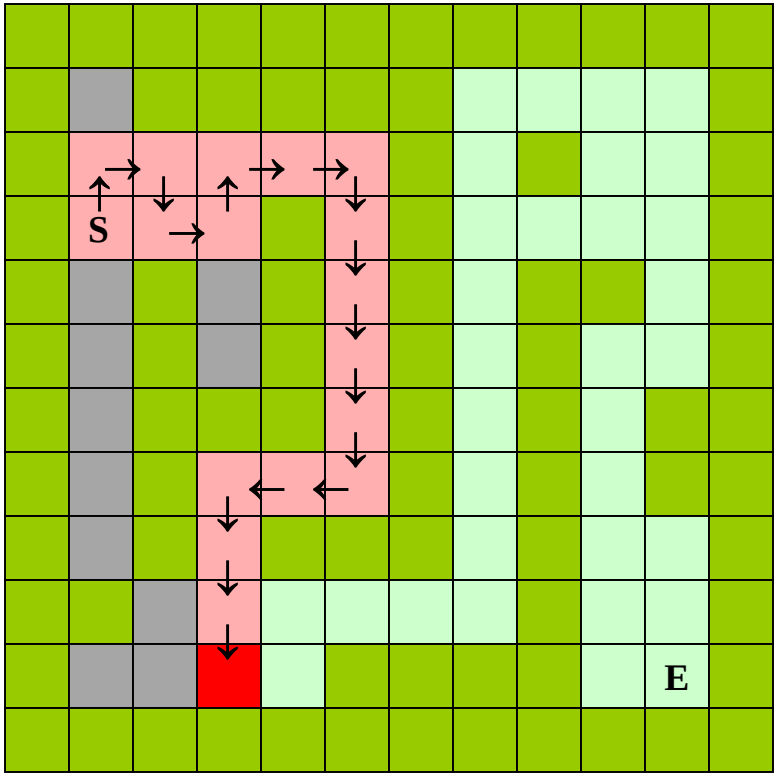
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



Several steps combined

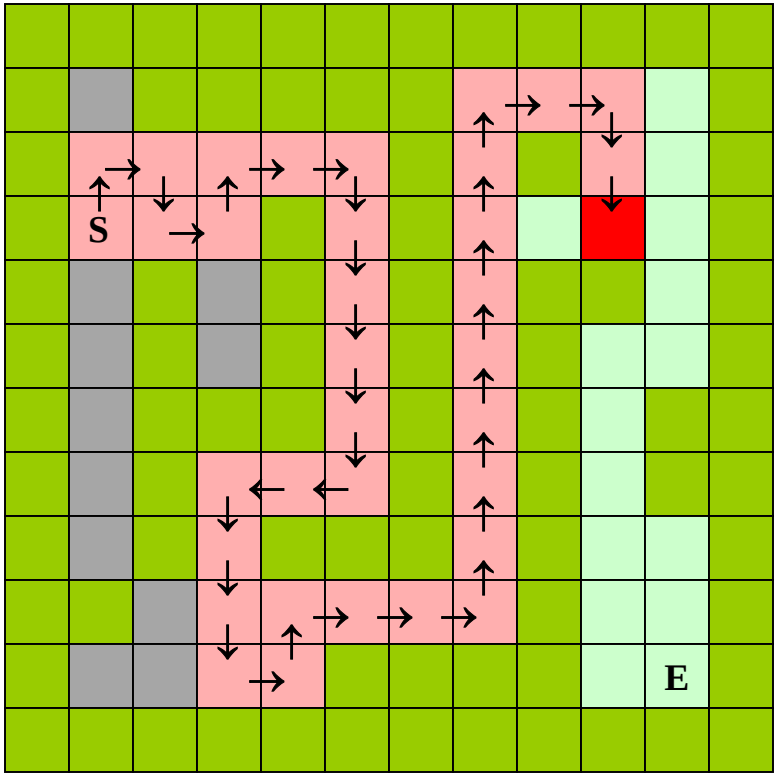
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



Several steps combined

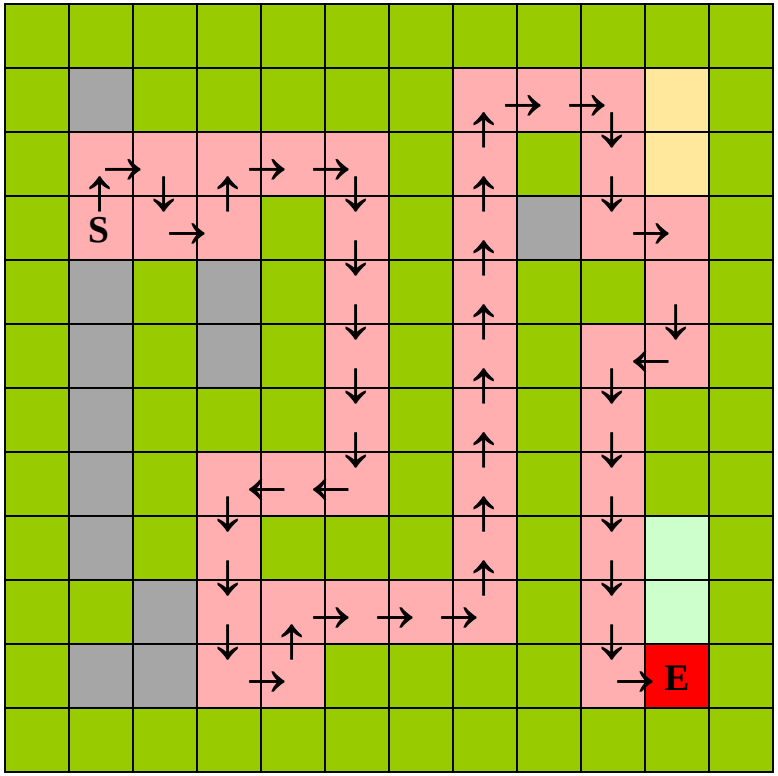
Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Searching the Example Maze



Several steps combined

Arrows indicate the current path

Current position

Visited and pending (on current path)

Visited and Done (not on current path)

Application 6
Maze： Search Method

- First we mark the Start position as VISITED and make it the current position.
- When exploring from a current position P, we look for an open neighbor in a fixed pattern: east, south, west and north.
- When an OPEN neighbor Q is found, we mark it as visited.
- In this case we say that Q was entered from P and that P is "pending": we have not finished exploring from P.
- We first check to see if Q is the Exit position.
- If so, the search concludes successfully. In this case, we print out the path from the Start to the Exit.
- If not, we continue exploring from position Q.

Application 6
Maze： Search Method

- If all four neighbors of Q are either WALL or VISITED, we return to the position P from which Q was first visited and continue exploring from P.
- That is, we “back out” from Q to P. This is the only way we can move to a previously visited position.
- Once we back out from Q, we will never return. Also, Q will not be on the path.
- If we return to the Start position and find no OPEN neighbors, the search concludes unsuccessfully: there is no path from the Start to the Exit.

Application 6
Maze

- We now consider how to carry out the algorithm described previously.
- In particular, how do we keep track of the position from which the current position was entered?
- Consider this: Suppose we go from S to P, then P to Q, then Q to R and R to T, then at position T find no OPEN neighbors.
- Then we back out to R. Suppose R has no OPEN neighbors. Then we should back out to Q.
- If Q has no OPEN neighbors, we back out to P and then to S.
- So the Path progresses like this:
- S
- S→P
- S →P →Q
- S →P→Q →R
- S →P→Q →R →T
- S →P→Q →R
- S →P→Q
- S →P
- S

So we will maintain a stack of Positions.
When we first visit a Position, we push it on the stack.
To back out from a Position, we pop the top of the stack and continue exploring from there.

Summary

- ADT stack operations have a last-in, first-out (LIFO) behavior
- Stack applications
- A strong relationship exists between recursion and stacks

面试题

- 判断元素出栈、入栈顺序的合法性。如入栈的序列(1,2,3,4,5)，出栈序列为 (3,2,4,1,5)) 或 (3,4,1,5,2)

- 一个数组实现两个栈(共享栈)

- 在Stack<int>中增加成员函数min(), 返回当前栈中的最小值。在该栈中，调动min、push、及pop的时间复杂度都是O(1)。

  （Hint: 为了保证当我们弹出当前最小元素后，下一个最小元素也能够立即得出。因此我们可以把每次的最小元素放到另一个辅助栈中）