Stack ADT

Tiziana Ligorio

Today's Plan

Questons?

Stack ADT

Abstract Data Types

Bag

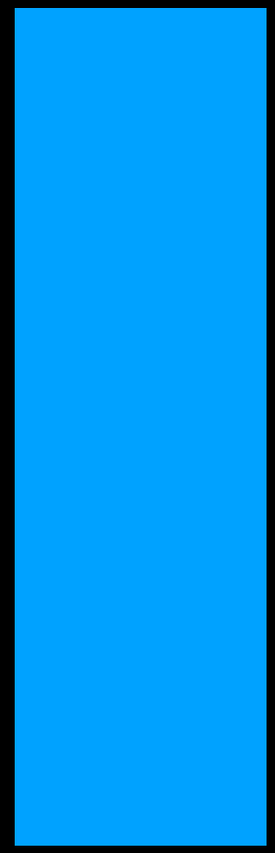
List

Stack

Stack

A data structure representing a stack of things

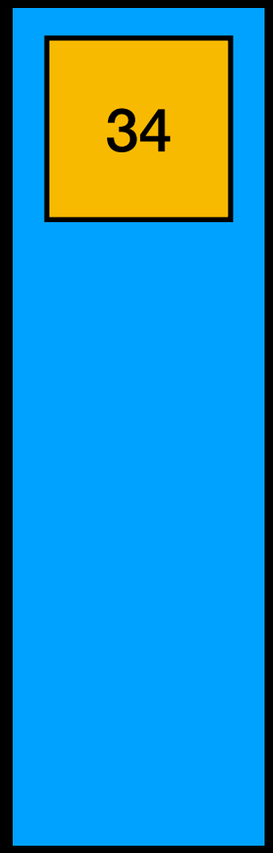
Objects can be pushed onto the stack or popped from the stack



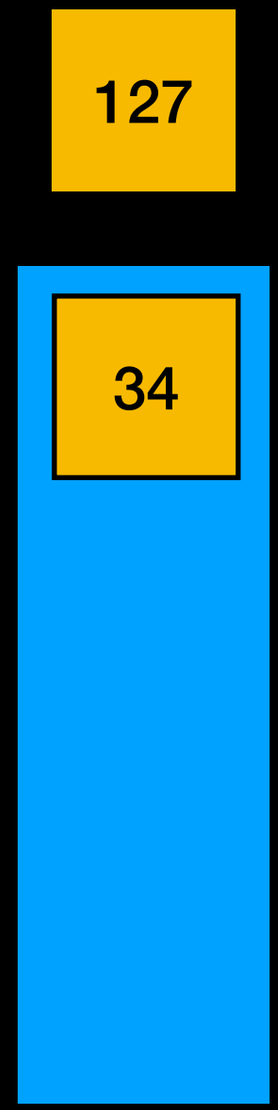
Stack

A data structure representing a stack of things

Objects can be pushed onto the stack or popped from the stack



Stack



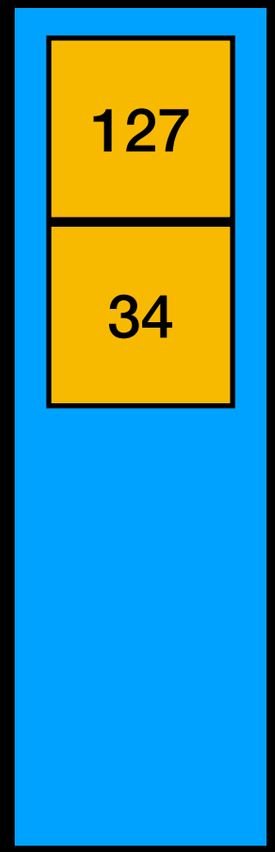
A data structure representing a stack of things

Objects can be pushed onto the stack or popped from the stack

Stack

A data structure representing a stack of things

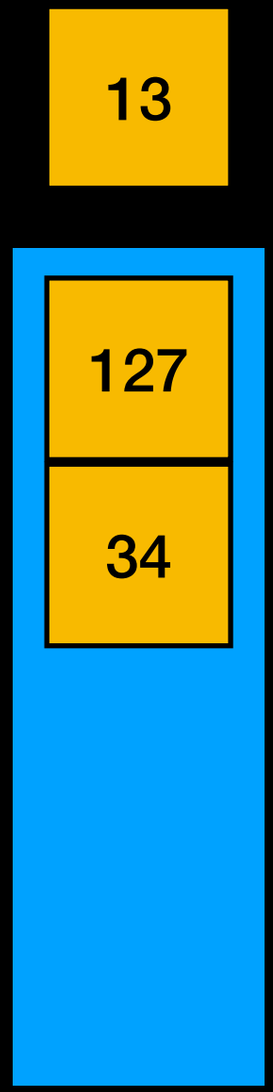
Objects can be pushed onto the stack or popped from the stack



Stack

A data structure representing a stack of things

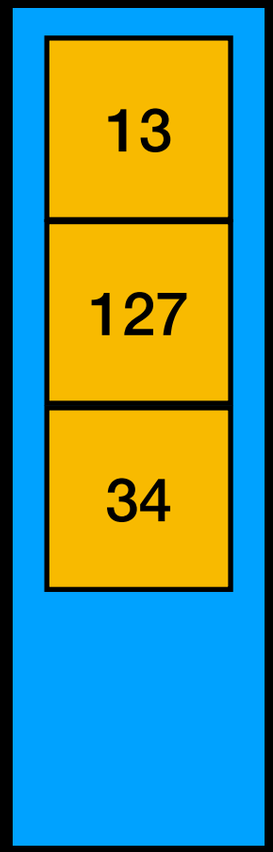
Objects can be pushed onto the stack or popped from the stack



Stack

A data structure representing a stack of things

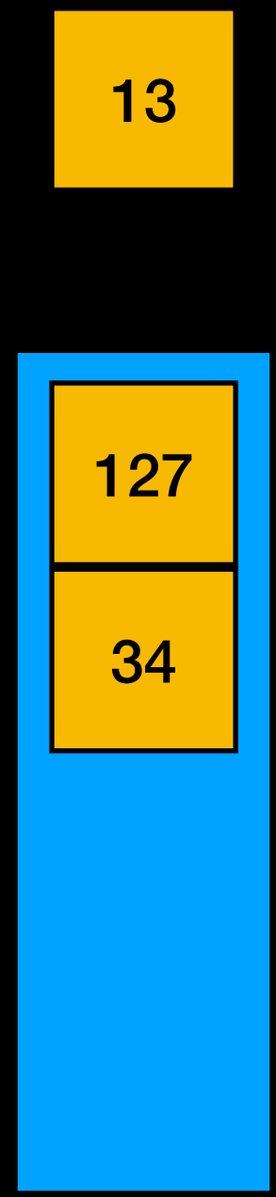
Objects can be pushed onto the stack or popped from the stack



Stack

A data structure representing a stack of things

Objects can be pushed onto the stack or popped from the stack



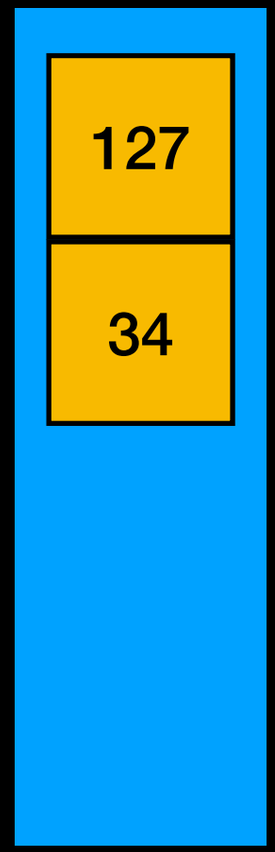
Stack

A data structure representing a stack of things

Objects can be pushed onto the stack or popped from the stack

LIFO: Last In First Out

Only top of stack is accessible (top), no other objects on the stack are visible



Applications

Very simple structure

Many applications:

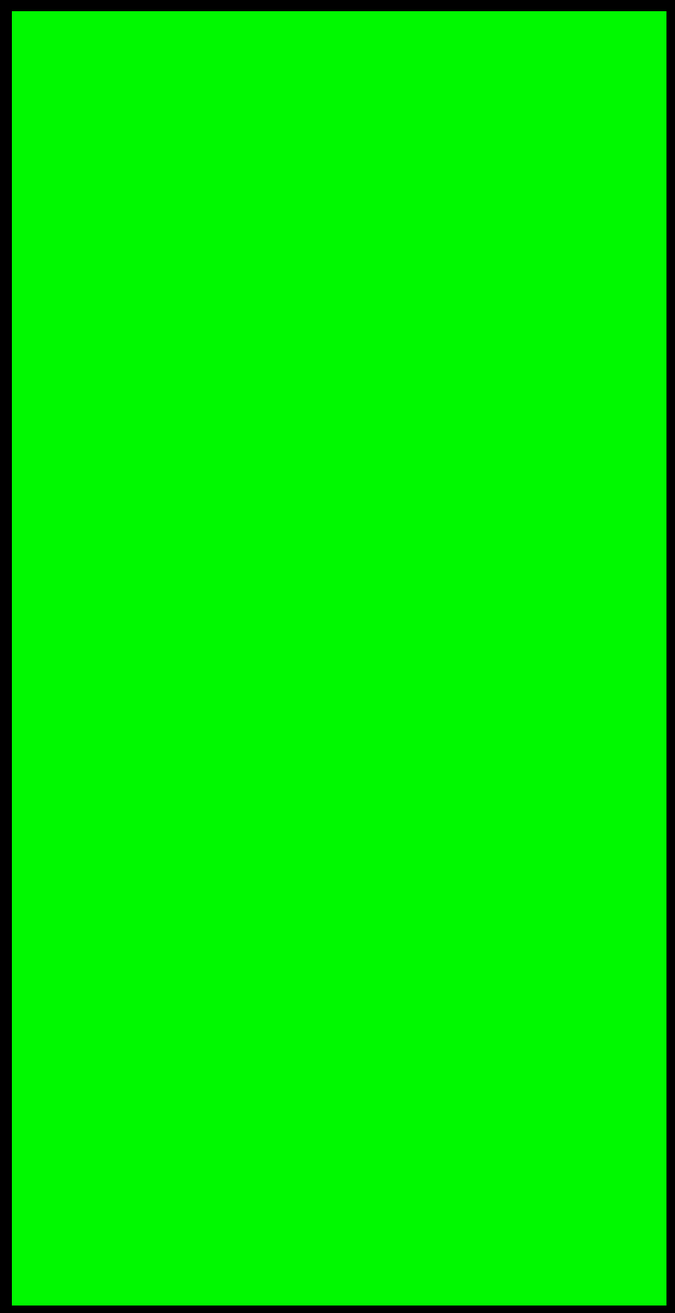
program stack
balancing parenthesis
evaluating postfix expressions
backtracking
. . . and more

Program Stack

Program Stack

```
1  void f(int x, int y)
2  {
3     int a;
4     // stuff here
5     if(a<13)
6        a = g(a);
7     // stuff here
8  }

9  int g(int z)
10 {
11    int p ,q;
12    // stuff here
13    return q;
14 }
```

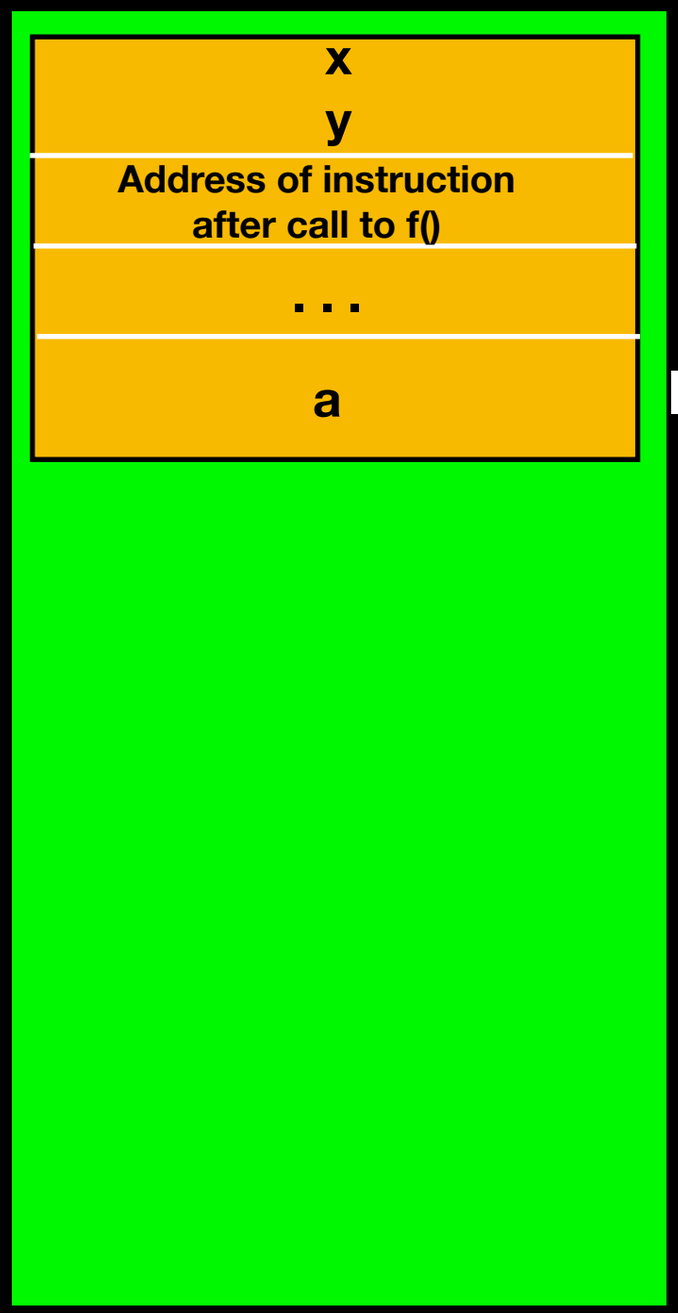


Program Stack

```
1   void f(int x, int y)
2   {
3      int a;
4      // stuff here
5      if(a<13)
6          a = g(a);
7      // stuff here
8   }


9   int g(int z)
10  {
11     int p ,q;
12     // stuff here
13     return q;
14  }
```

Stack Frame for f()



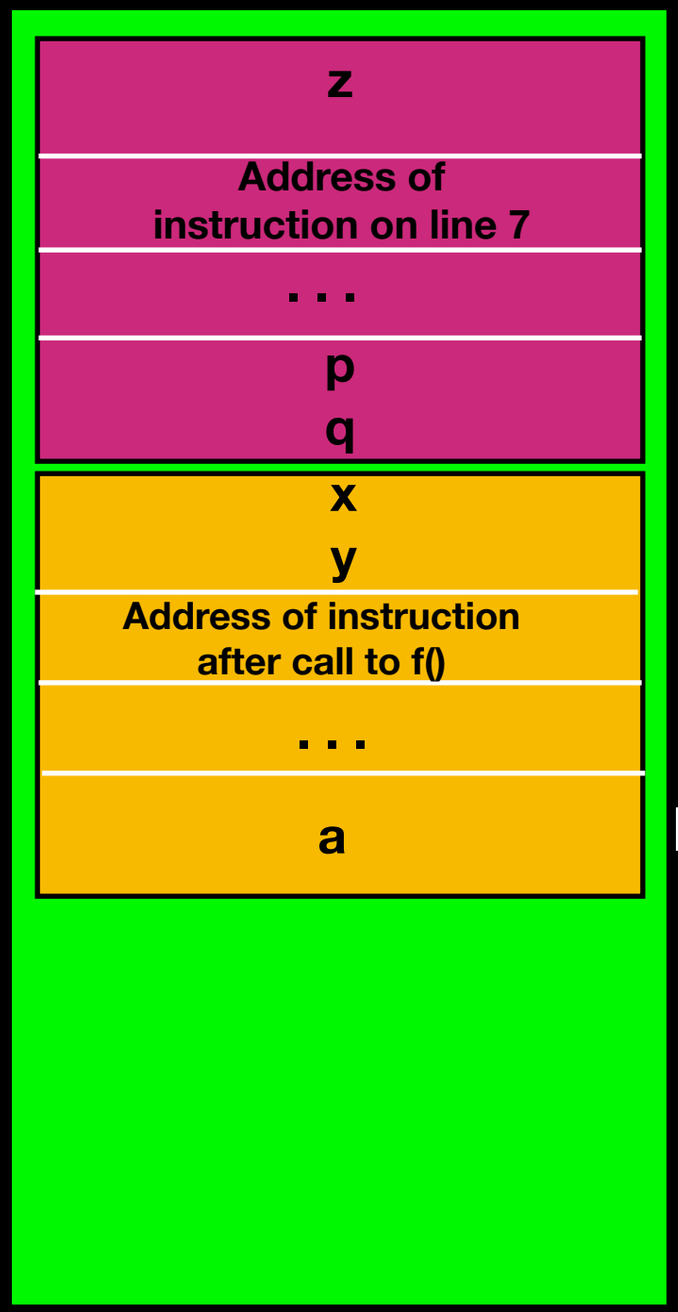
Program Stack

```
1   void f(int x, int y)
2   {
3      int a;
4      // stuff here
5      if(a<13)
6          a = g(a);
7      // stuff here
8   }

9   int g(int z)
10  {
11     int p ,q;
12     // stuff here
13     return q;
14  }
```

Stack Frame for g()

Stack Frame for f()



parameters

return address

local variables

parameters

return address

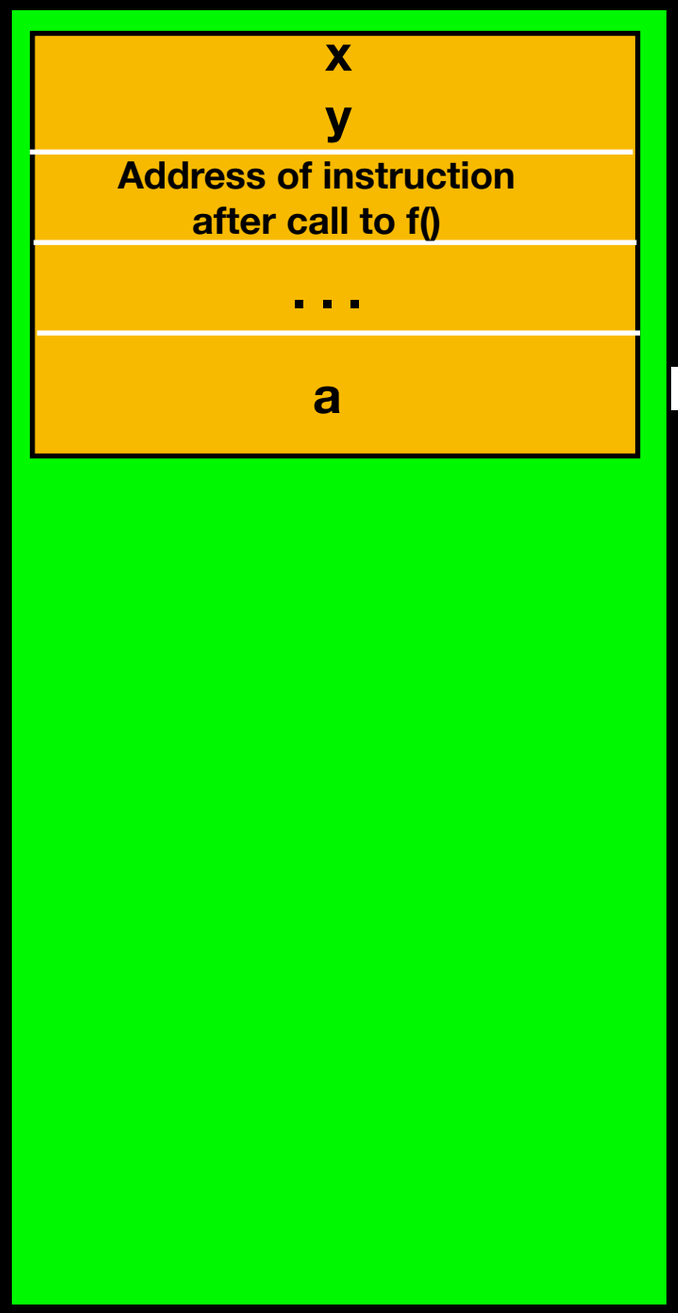
local variables

Program Stack

```
1   void f(int x, int y)
2   {
3      int a;
4      // stuff here
5      if(a<13)
6          a = g(a);
7      // stuff here
8   }


9   int g(int z)
10  {
11     int p ,q;
12     // stuff here
13     return q;
14  }
```

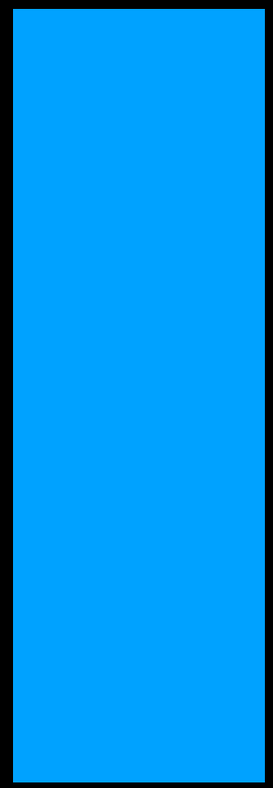
Stack Frame for f()



Balancing Parentheses

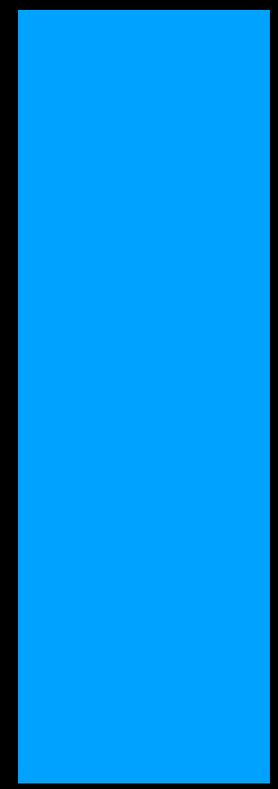
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```

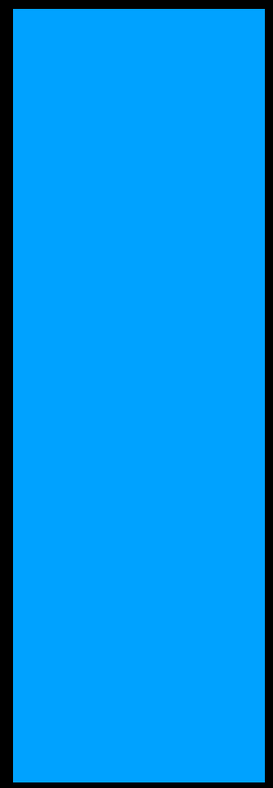
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



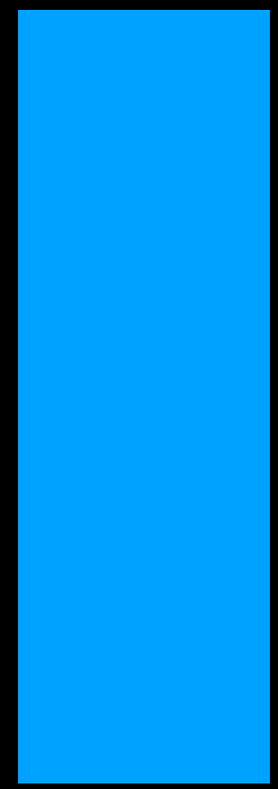
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



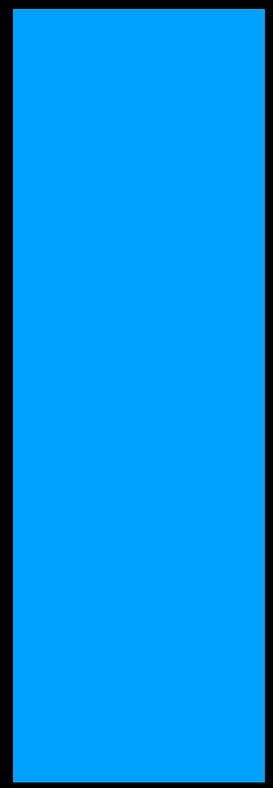
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

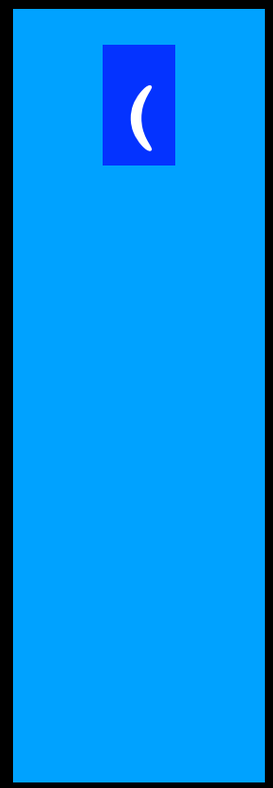
```
int f(){if(x*(y+z[i])<47){x += y}}
```

Balancing Parentheses

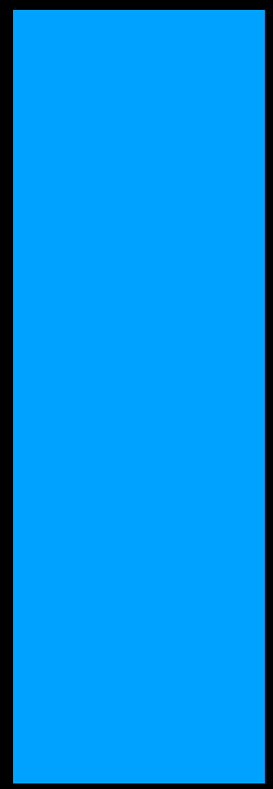
```
int f(){if(x*(y+z[i])<47){x += y}}
```


push



Balancing Parentheses

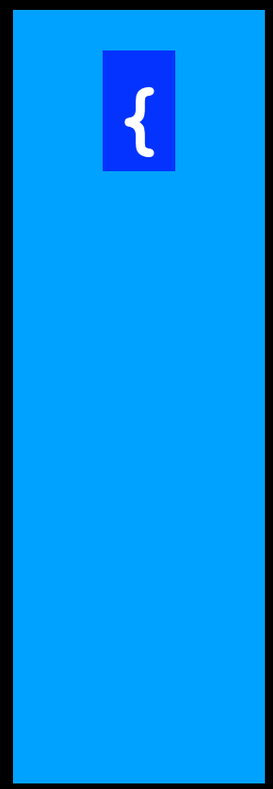
```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

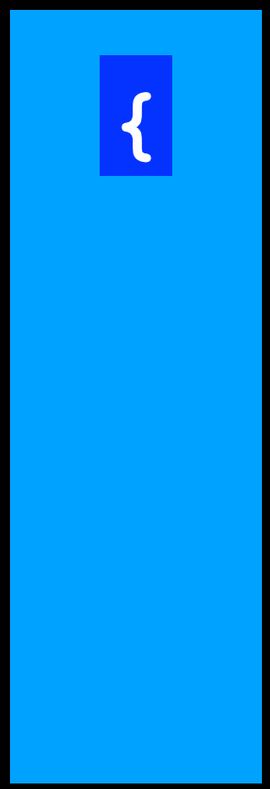
```
int f(){if(x*(y+z[i])<47){x += y}}
        ↑
```

push



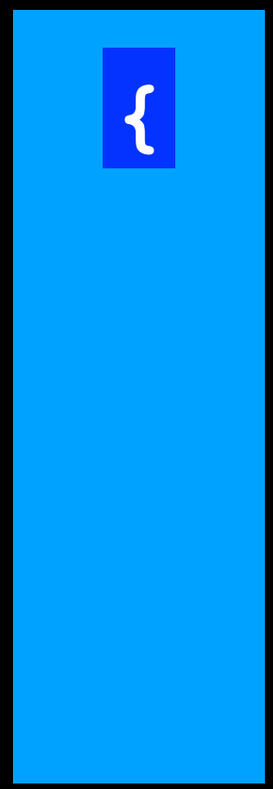
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

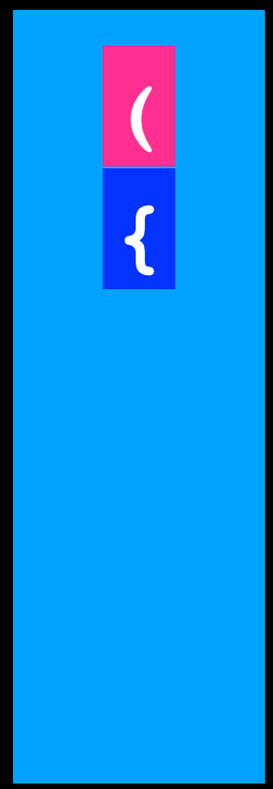
```
int f(){if(x*(y+z[i])<47){x += y}}
```

Balancing Parentheses

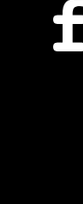
```
int f(){if(x*(y+z[i])<47){x += y}}
```

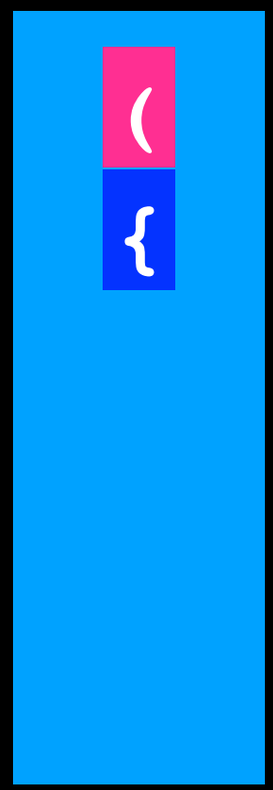
push



Balancing Parentheses

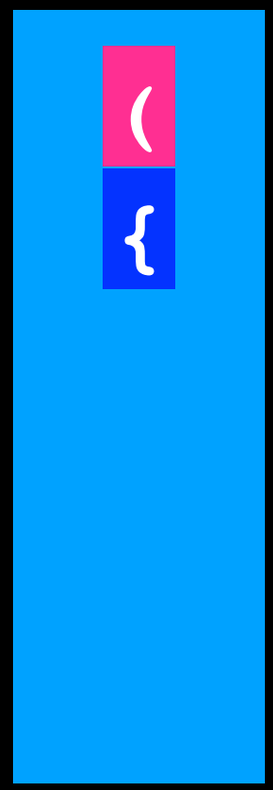
```
int f(){if(x*(y+z[i])<47){x += y}}
```





Balancing Parentheses

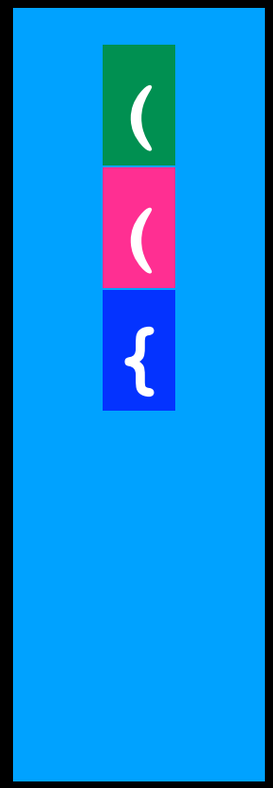
```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

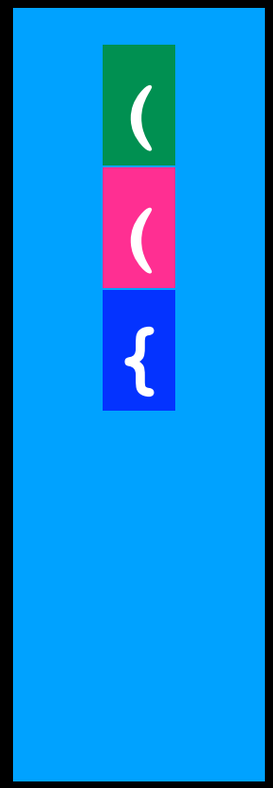
push

```
int f(){if(x*(y+z[i])<47){x += y}}
```



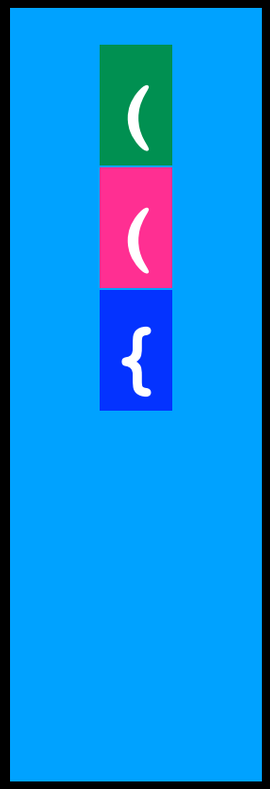
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



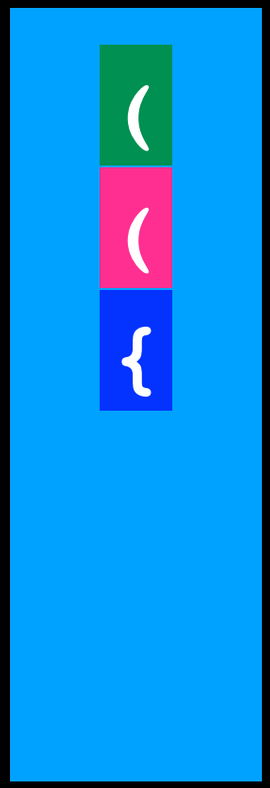
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

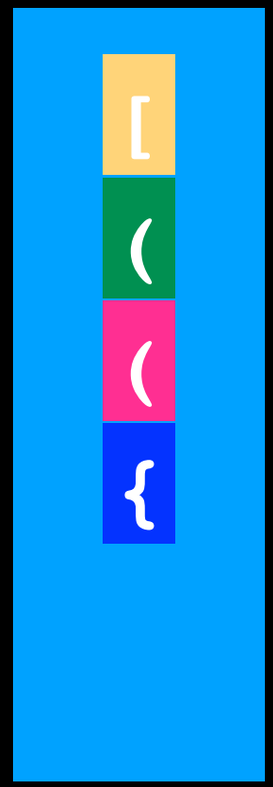
```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

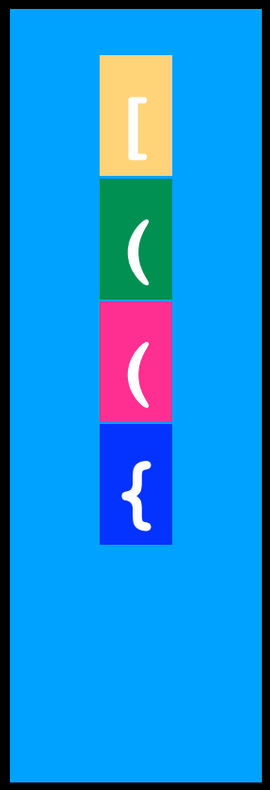
```
int f(){if(x*(y+z[i])<47){x += y}}
```

push



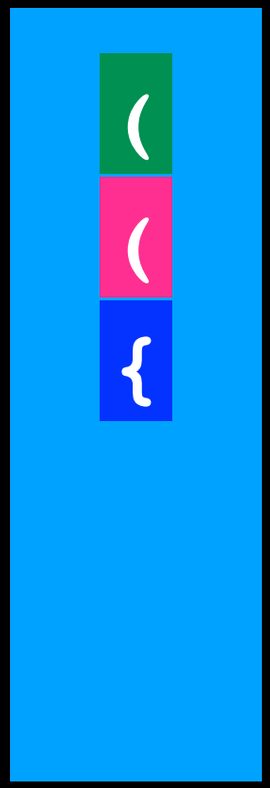
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

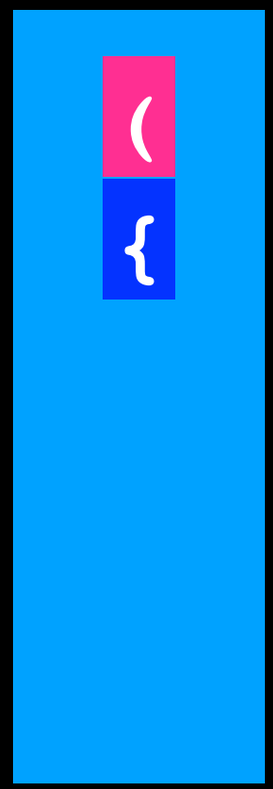
```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

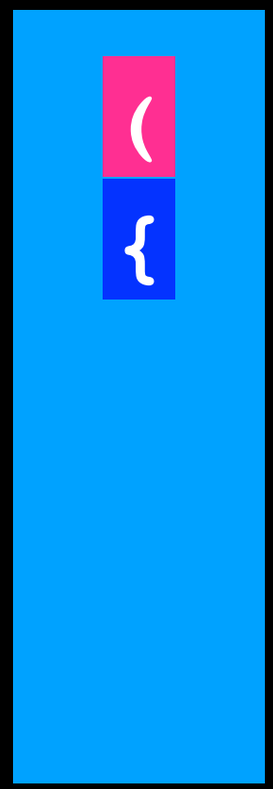
```
int f(){if(x*(y+z[i])<47){x += y}}
```

pop



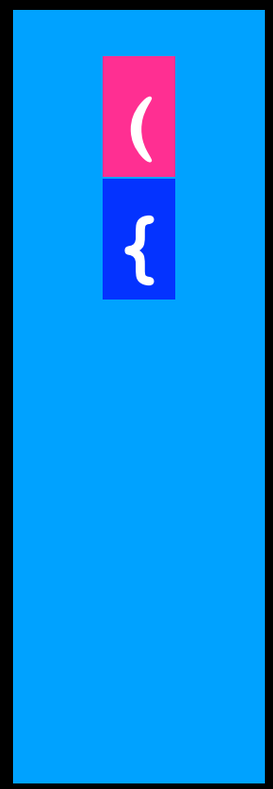
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



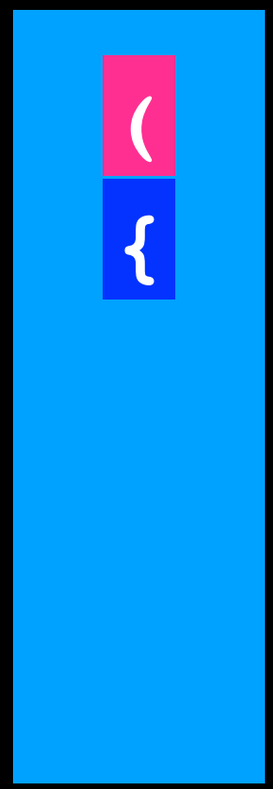
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



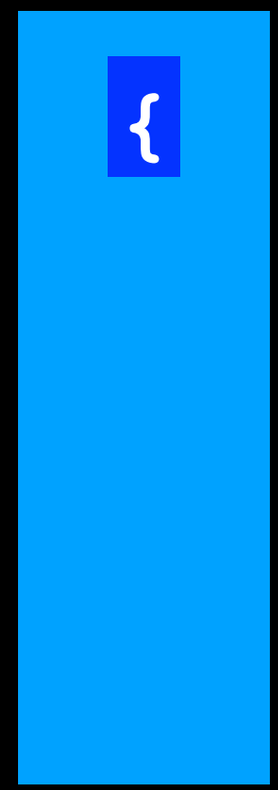
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



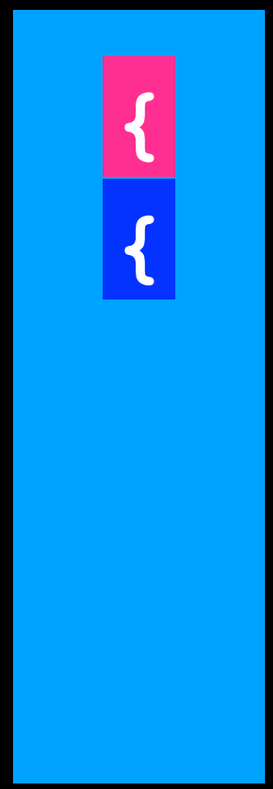
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



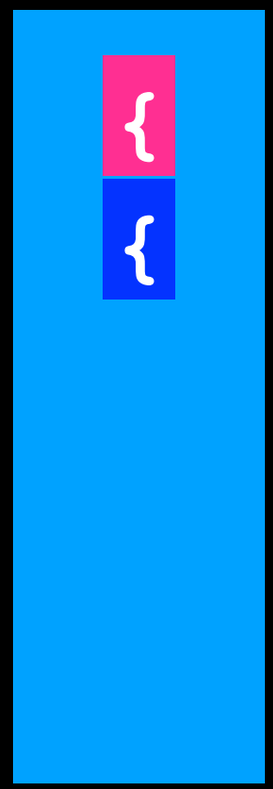
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



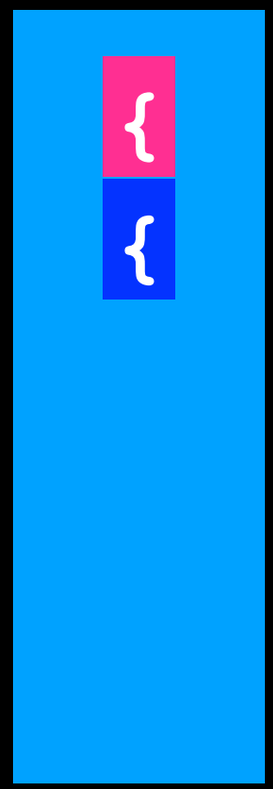
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



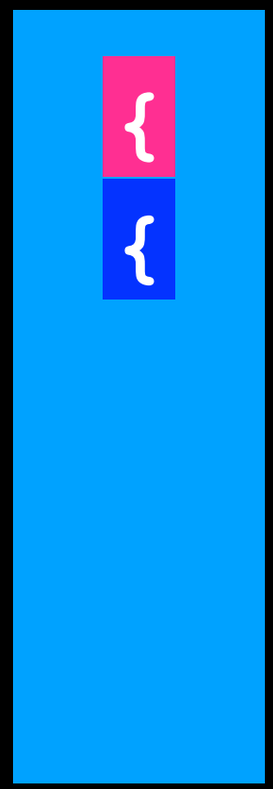
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



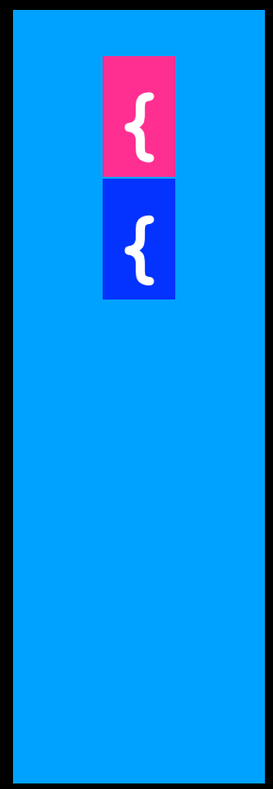
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



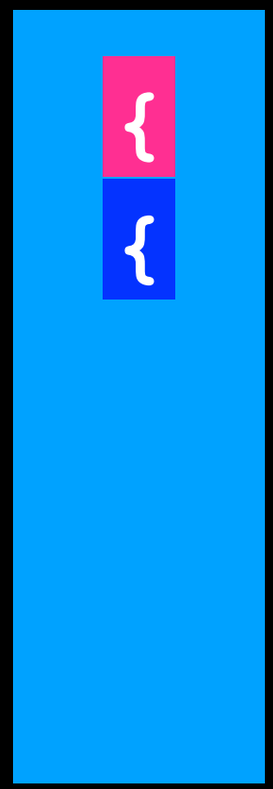
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



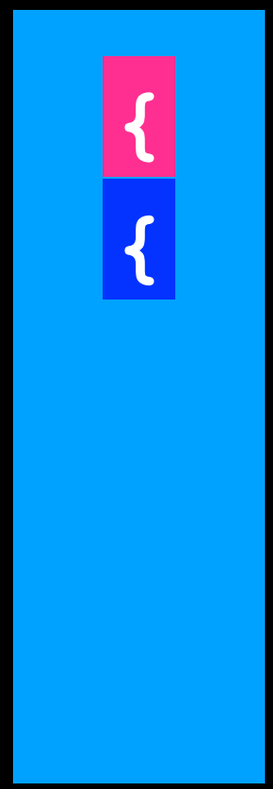
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



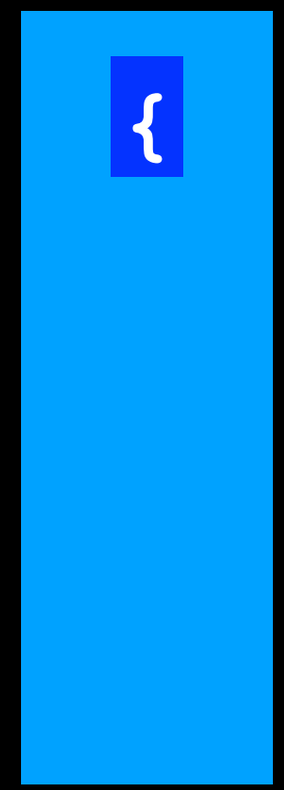
Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```



Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```

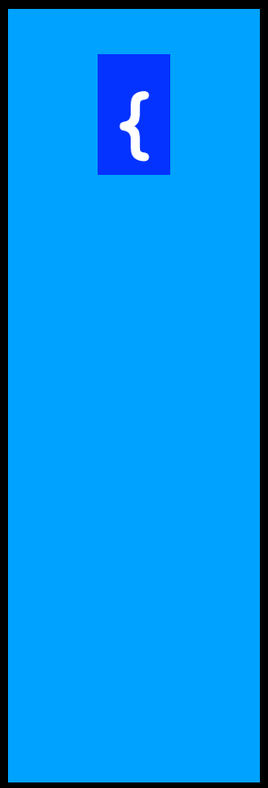


Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}}
```

Balancing Parentheses

```
int f(){if(x*(y+z[i])<47){x += y}
```



Finished reading
Stack not empty
Parentheses NOT balanced

Balancing Parentheses

```
for(char ch : st)
{
   if ch is an open parenthesis character
       push it on the stack
   else if ch is a close parenthesis character
       if it matches the top of the stack
       pop the stack
       else
          return unbalanced
   // else it is not a parenthesis
}

if stack is empty
   return balanced
else
   return unbalanced
```

Evaluating Postfix Expressions

Postfix Expressions

Operator applies to the two operands immediately preceding it

Infix:	**Postfix:**
2 * (3 + 4)	**2 3 4 + ***
2 * 3 + 4	**2 3 * 4 +**

Evaluating Postfix Expressions

Operator applies to the two operands immediately preceding it

Postfix:

2 3 4 + *

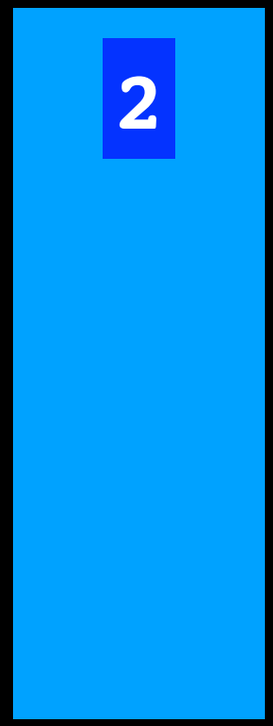
Assumptions / simplifications:

- String is syntactically correct postfix expression
- No unary operators
- No exponentiation operation
- Operands in string are single integer values

Evaluating Postfix Expressions

Postfix:

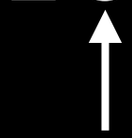
2 3 4 + *

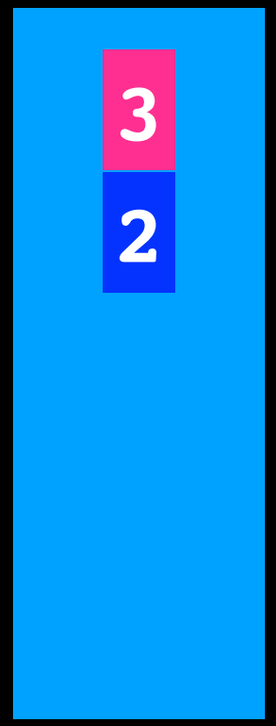


Evaluating Postfix Expressions

Postfix:

2 3 4 + *

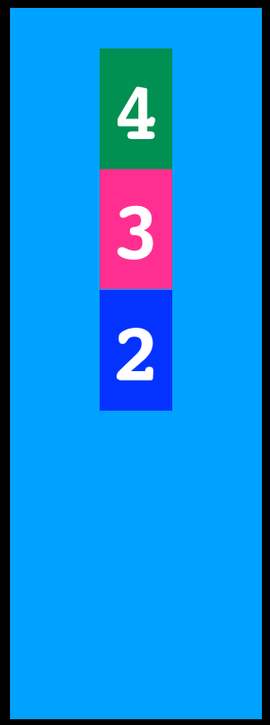




Evaluating Postfix Expressions

Postfix:

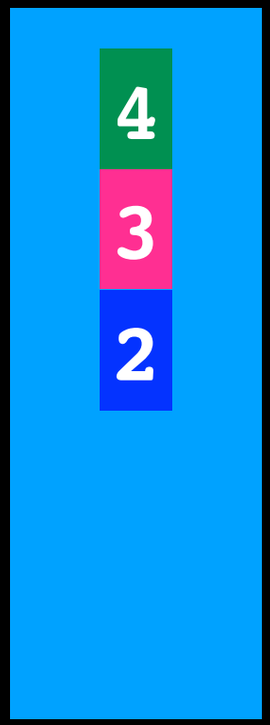
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

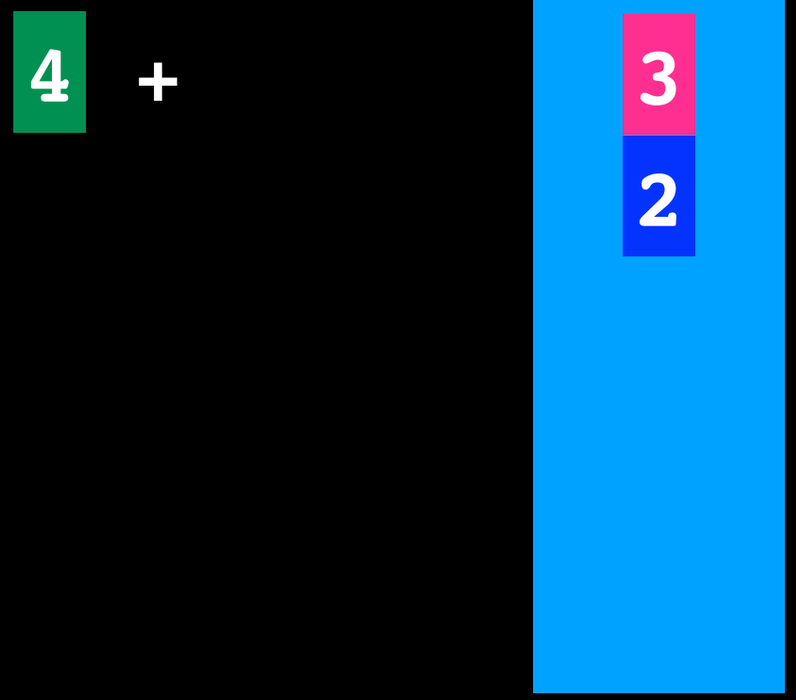
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

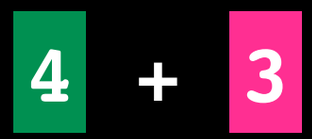
2 3 4 + *

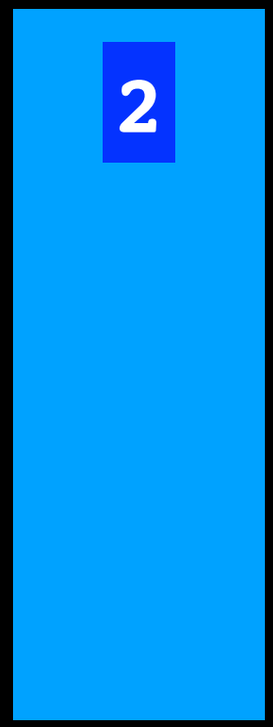


Evaluating Postfix Expressions

Postfix:

2 3 4 + *

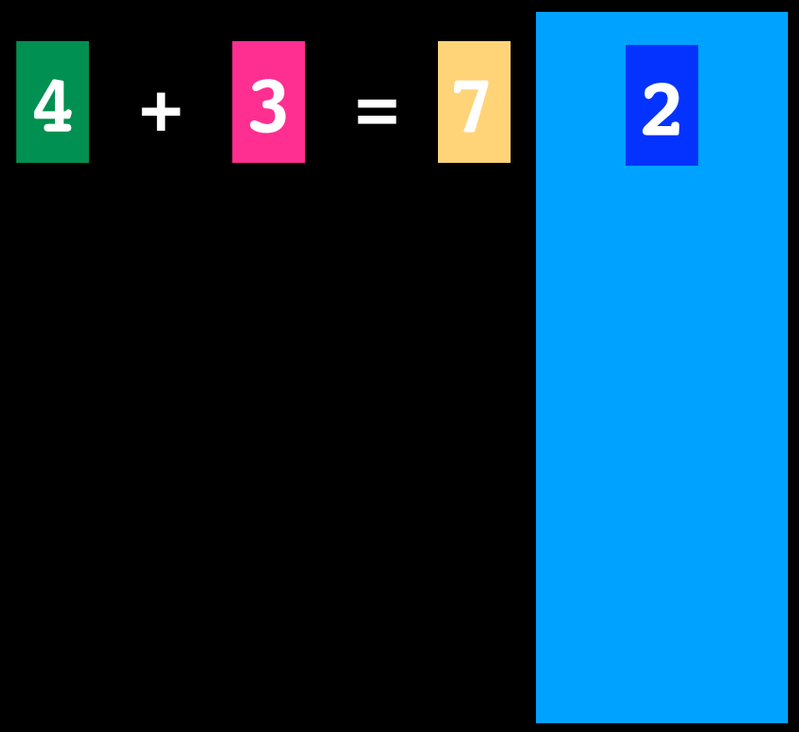




Evaluating Postfix Expressions

Postfix:

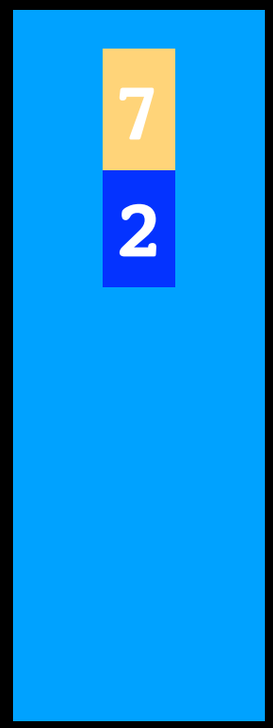
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

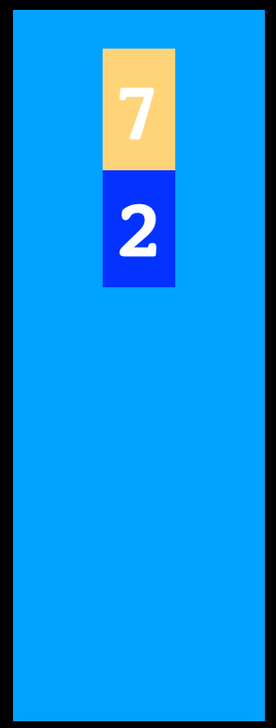
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

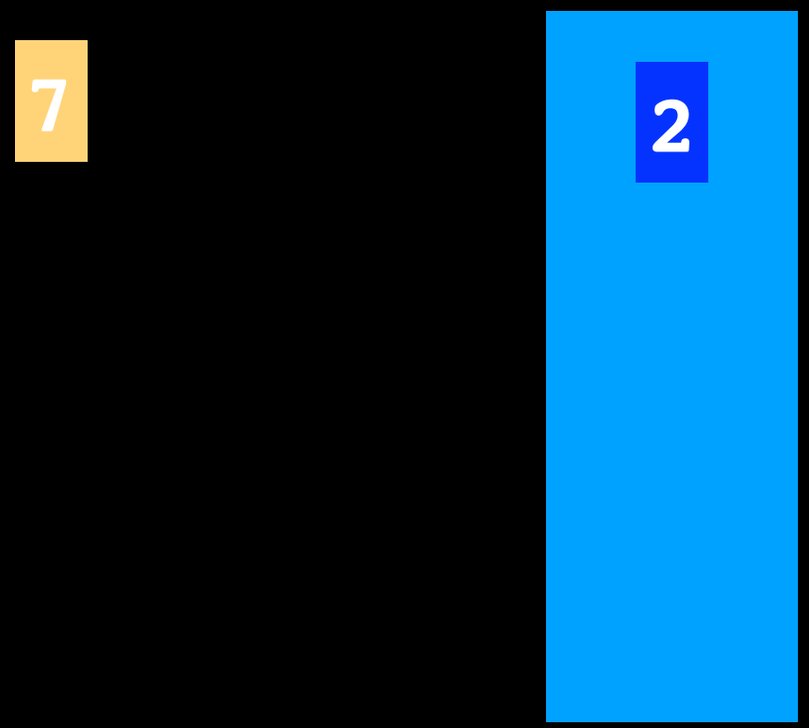
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

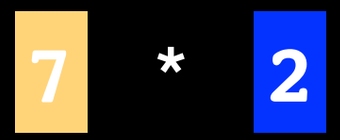
2 3 4 + *

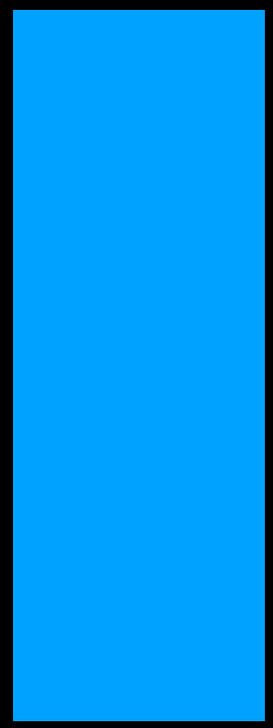


Evaluating Postfix Expressions

Postfix:

2 3 4 + *

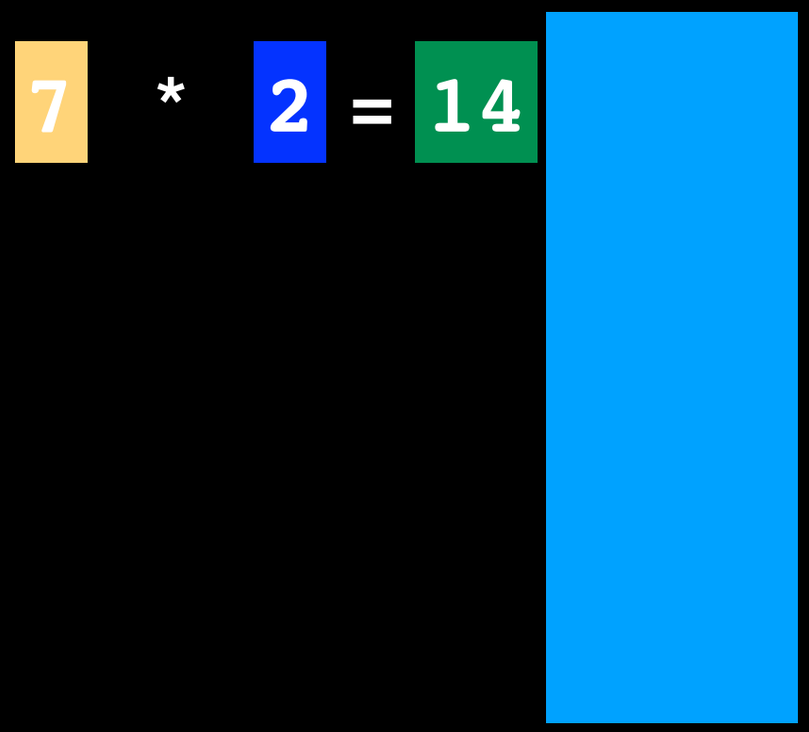




Evaluating Postfix Expressions

Postfix:

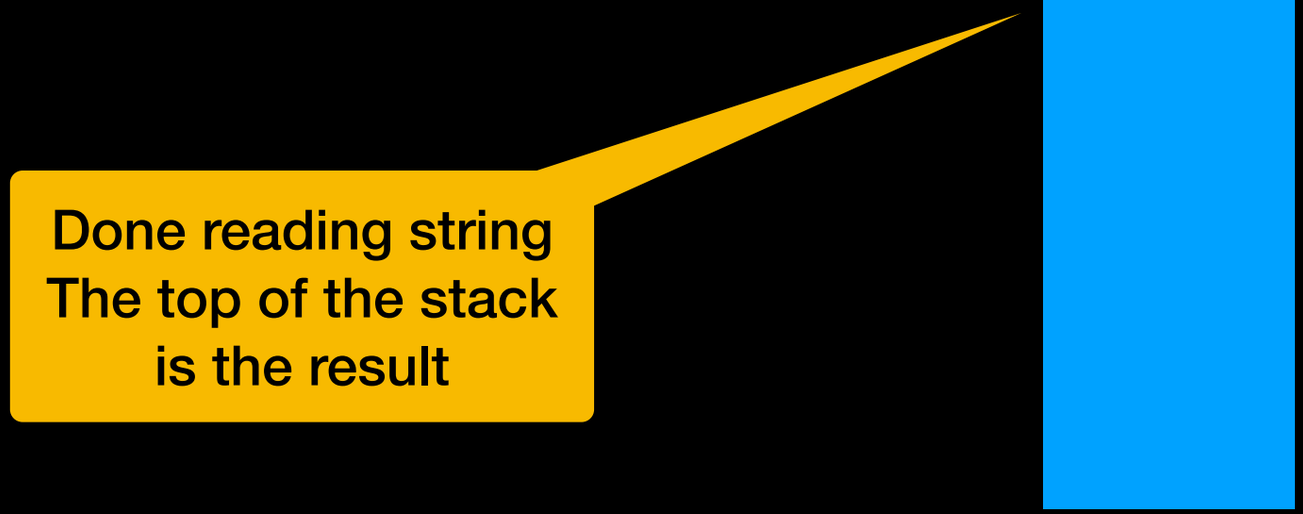
2 3 4 + *



Evaluating Postfix Expressions

Postfix:

2 3 4 + *



Evaluating Postfix Expressions

Operator applies to the two operands immediately preceding it

Postfix:

2 3 * 4 +

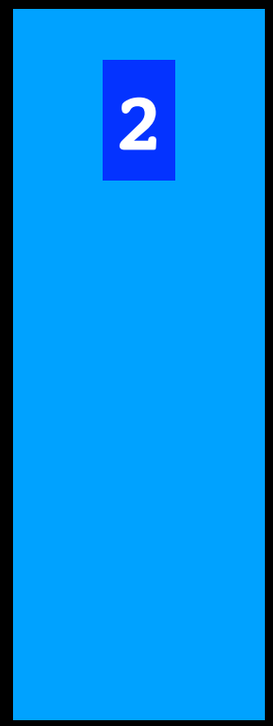
Assumptions / simplifications:

- string is syntactically correct postfix expression
- No unary operators
- No exponentiation operation
- Operands in string are single integer values

Evaluating Postfix Expressions

Postfix:

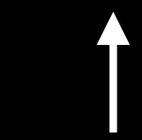
2 3 * 4 +

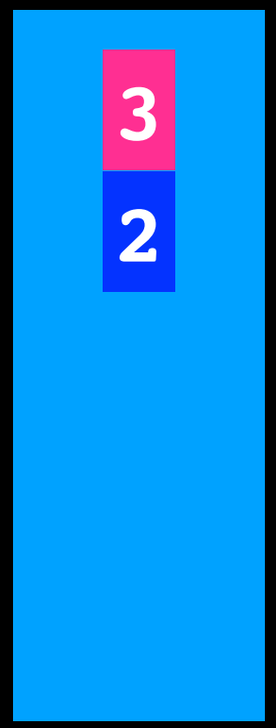


Evaluating Postfix Expressions

Postfix:

2 3 * 4 +

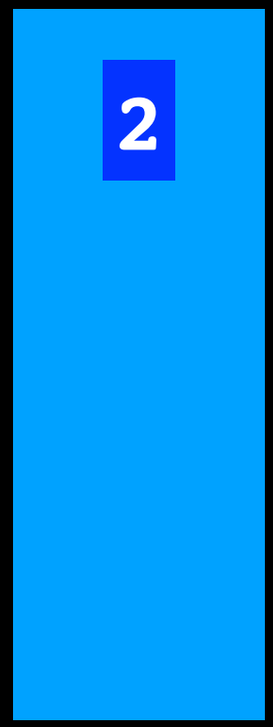




Evaluating Postfix Expressions

Postfix:

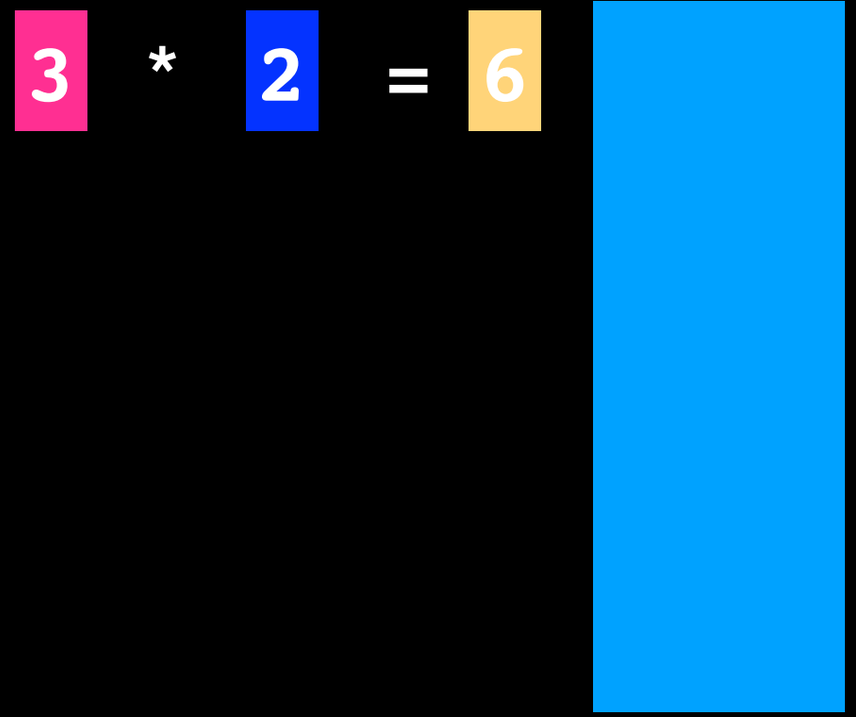
2 3 * 4 +



Evaluating Postfix Expressions

Postfix:

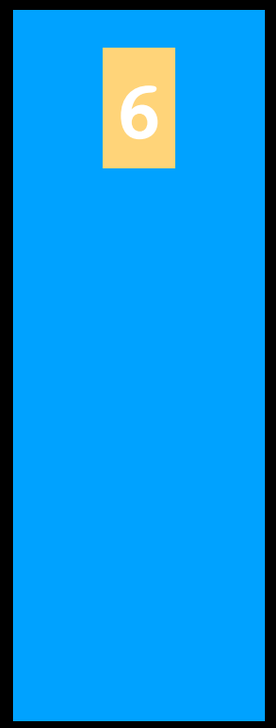
2 3 * 4 +



Evaluating Postfix Expressions

Postfix:

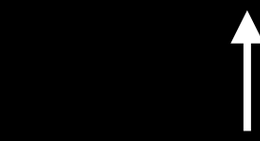
2 3 * 4 +

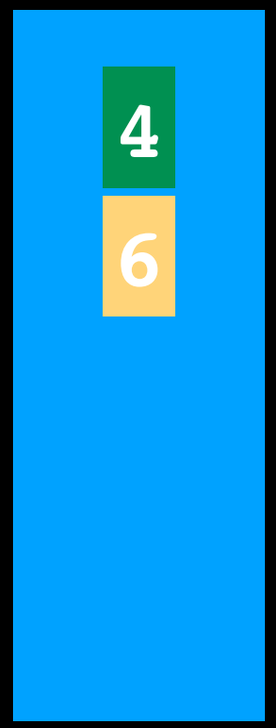


Evaluating Postfix Expressions

Postfix:

2 3 * 4 +

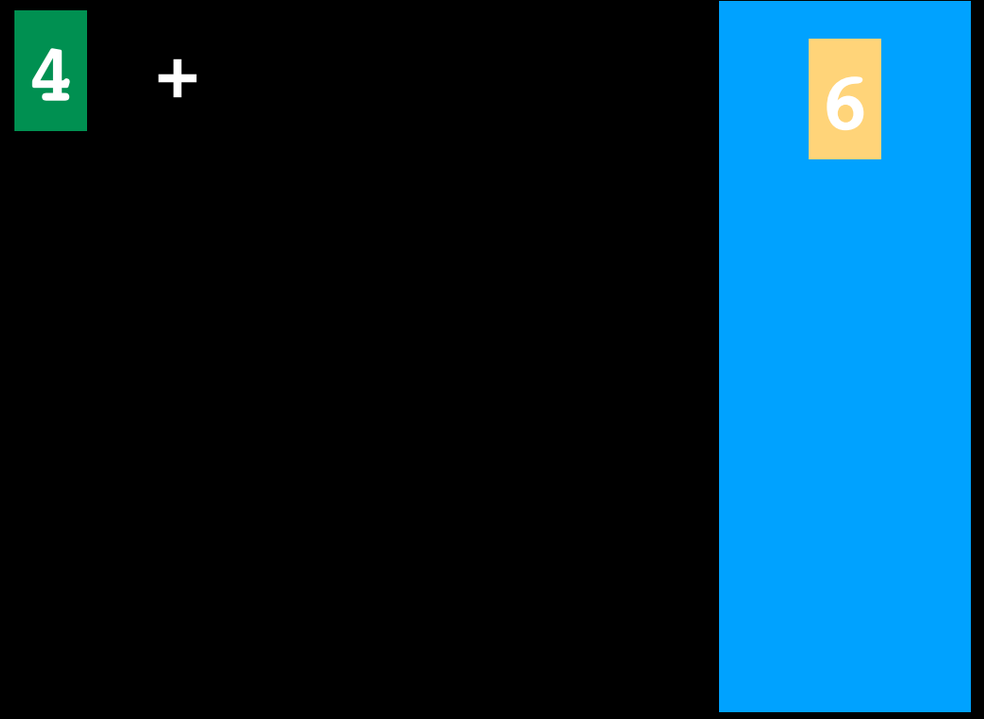




Evaluating Postfix Expressions

Postfix:

2 3 * 4 +

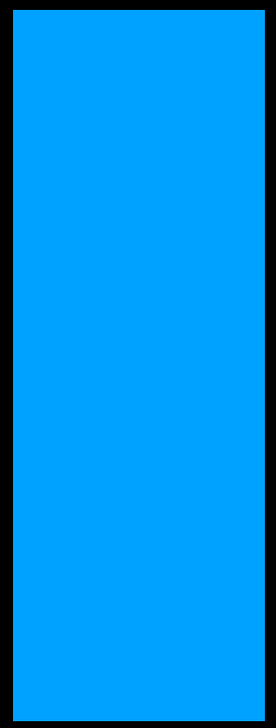


Evaluating Postfix Expressions

Postfix:

2 3 * 4 +

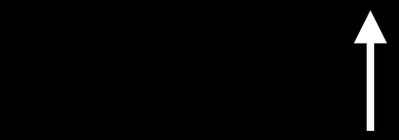
4 + 6 = 10



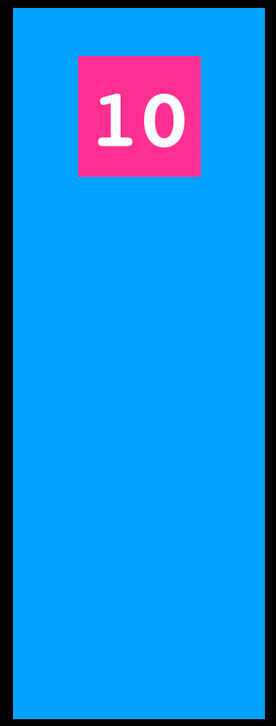
Evaluating Postfix Expressions

Postfix:

2 3 * 4 +



Done reading string
The top of the stack
is the result

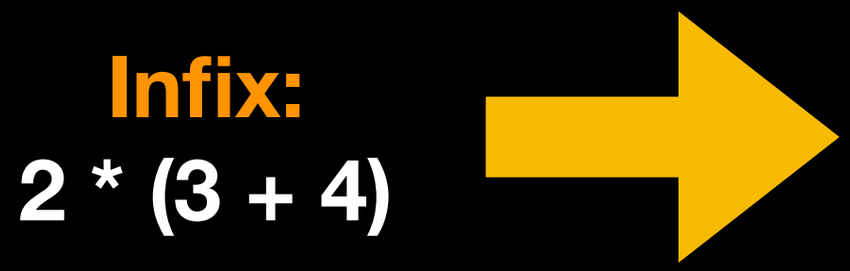


Evaluating Postfix Expressions

```
for(char ch : st)
{
   if ch is an operand
       push it on the stack
   else // ch is an operator op
   {
      //evaluate and push the result
     operand2 = pop stack
     operand1 = pop stack
     result = operand1 op operand2
     push result on stack
    }
}
```

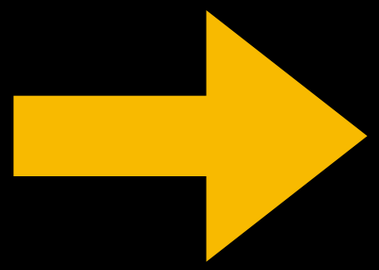
Lecture Activity

Describe an algorithm that translates the infix expression below into postfix (you can use drawings to explain):



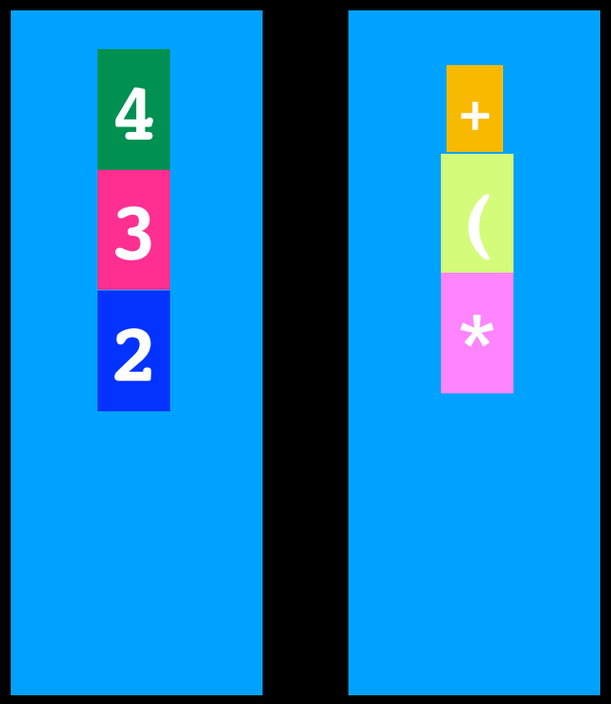
Hint: use 2 stacks, one for operators and parentheses another one for the operands and postfix expression. Once converted use the empty stack to invert the order

Infix:
2 * (3 + 4)



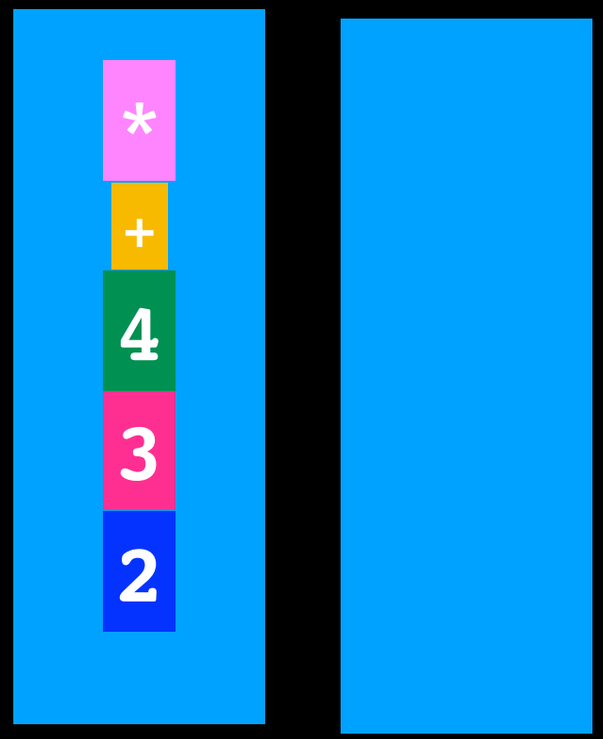
Postfix:
2 3 4 + *

1. Read characters onto corresponding stack until ‘)’



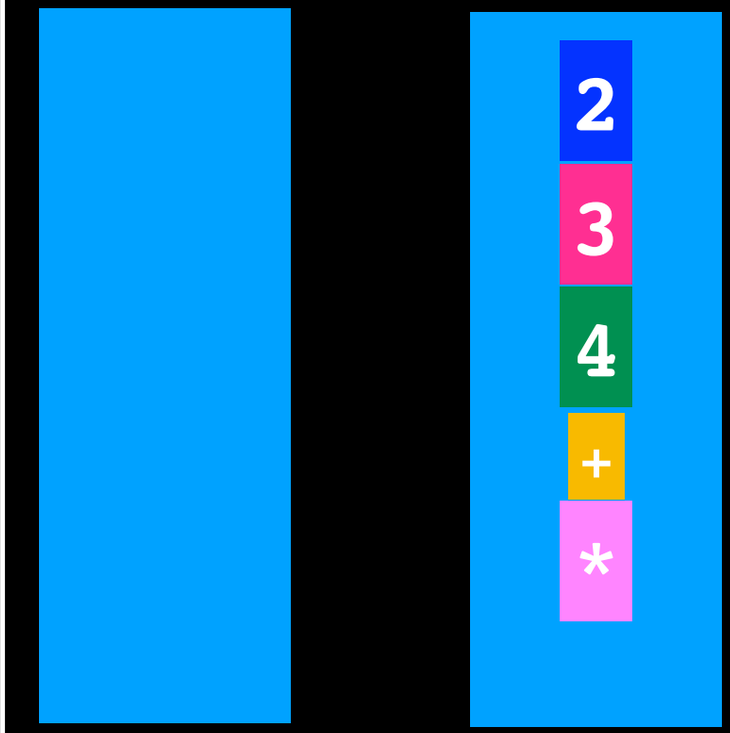
Postfix Stack | Operator Stack

2. Pop operator stack and push it onto postfix stack ignoring ‘(‘



Postfix Stack | Operator Stack

3. Push everything onto empty stack to invert
Then read pop and print.

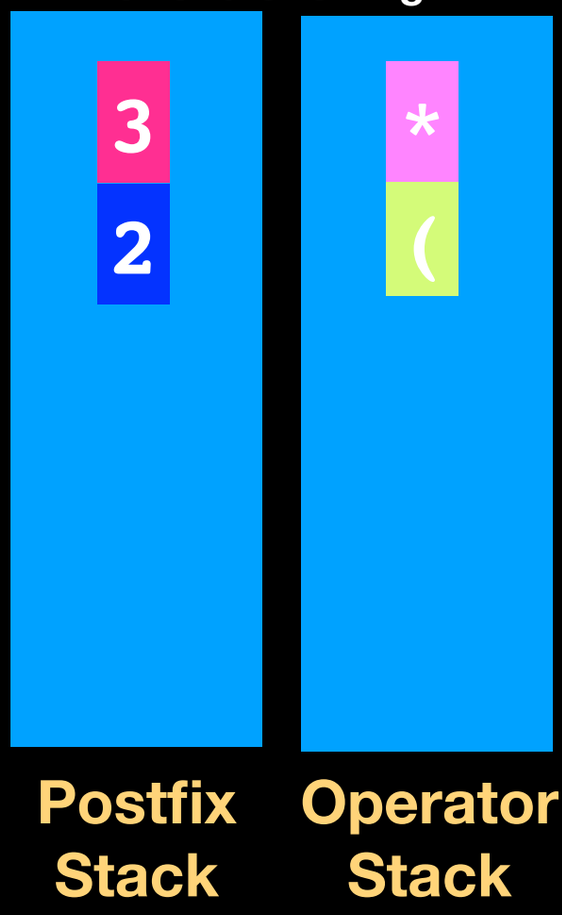


Postfix Stack | Operator Stack

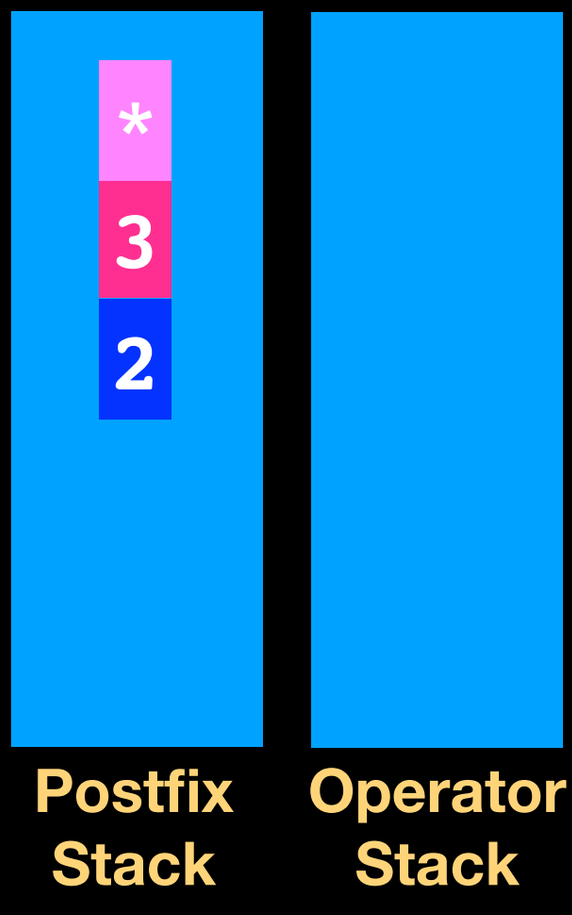
Infix:
(2 * 3)+ 4

Postfix:
2 3 *4 +

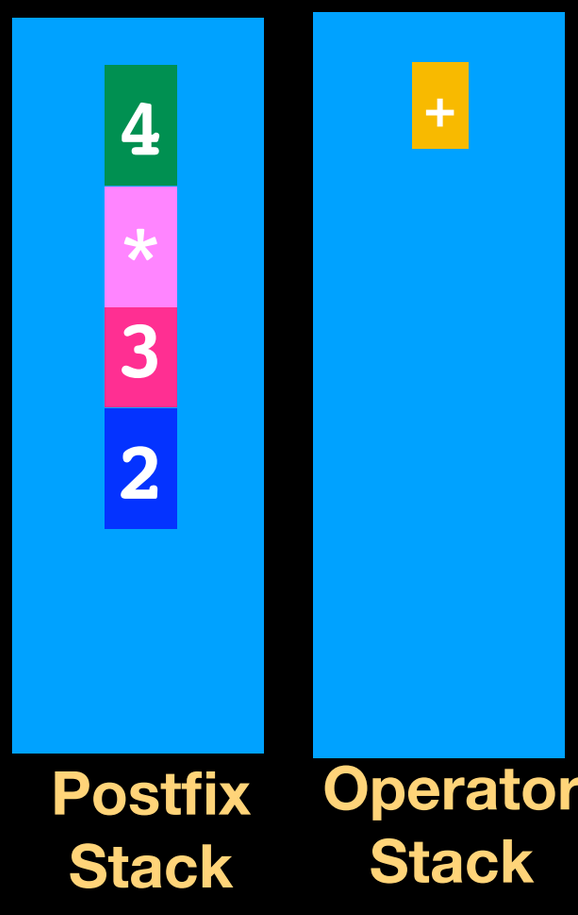
1. Read characters onto corr. stack until ‘)’ or end of string



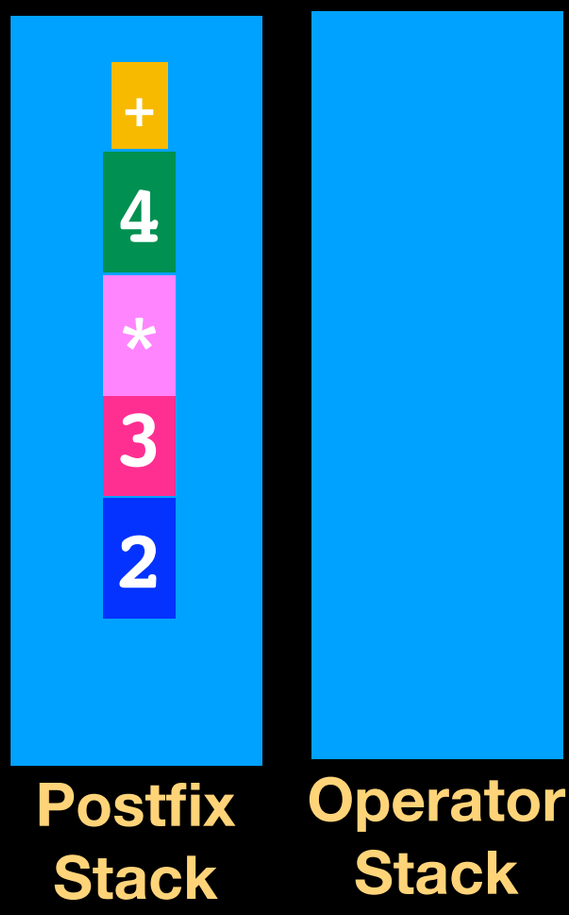
2. If reading a ‘)’, move operators to Postfix Stack until a ‘(‘ discard it and continue reading string



3. Keep reading until ‘)’ -> 2.
or end of string -> 4.



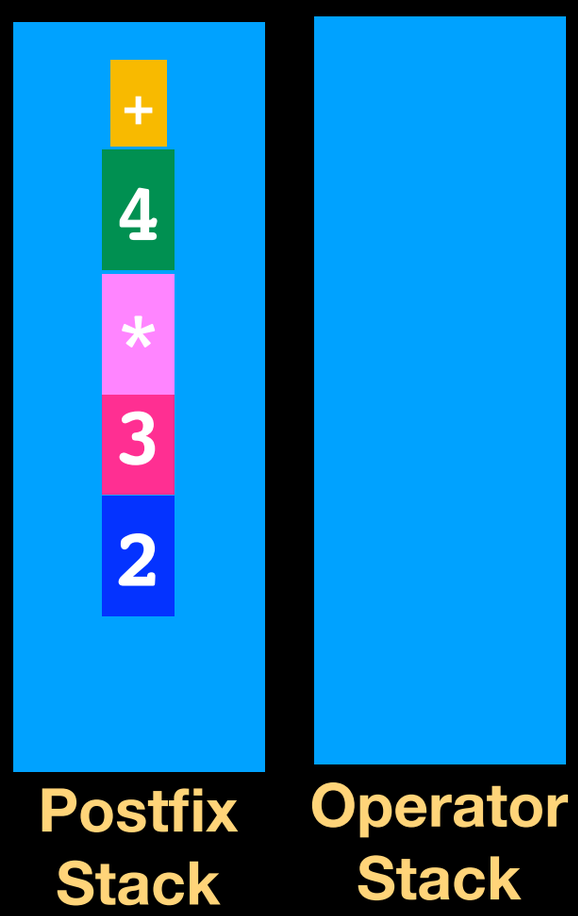
4. Move operators to Postfix Stack



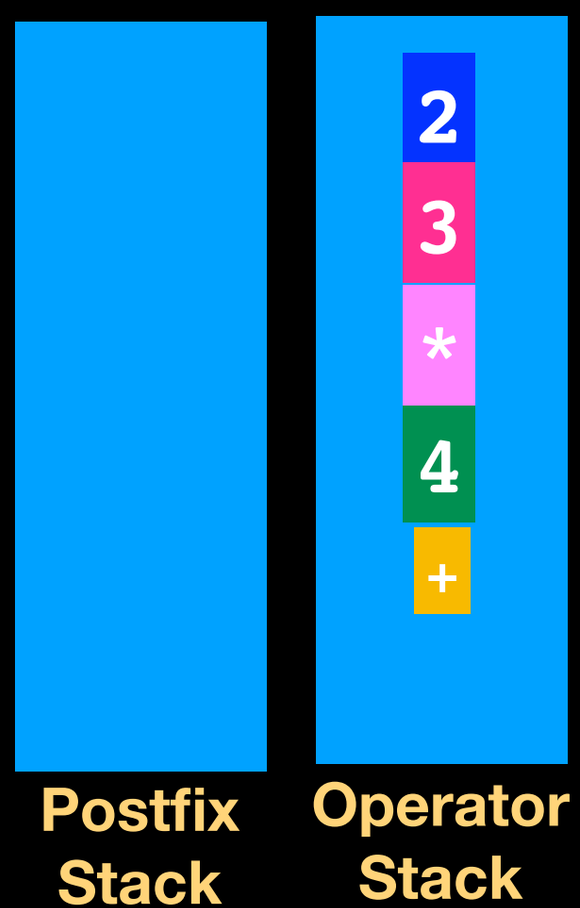
Infix:
(2 * 3)+ 4

Postfix:
2 3 *4 +

4. Move operators to Postfix Stack



5. Pop and push onto empty stack to invert, then print



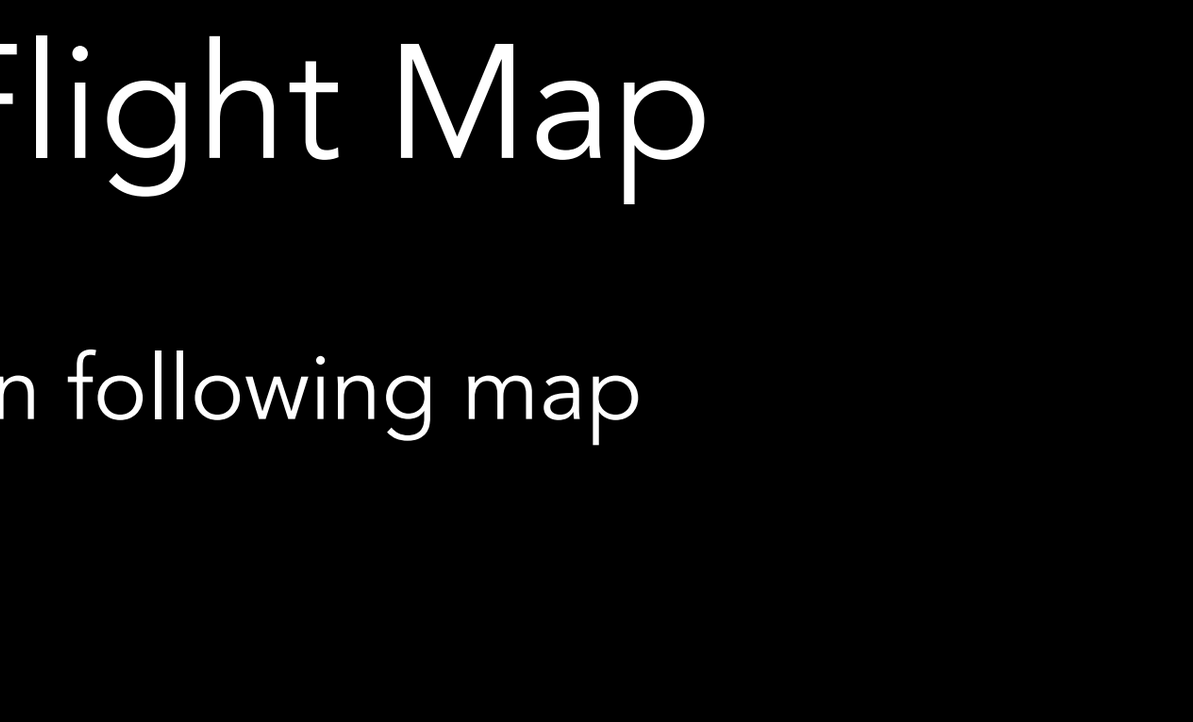
Search a Flight Map

Fly from Origin to Destination following map

1. Reach destination

2. Reach city with no departing flights (dead end)

3. Go in circles forever



Backtracking

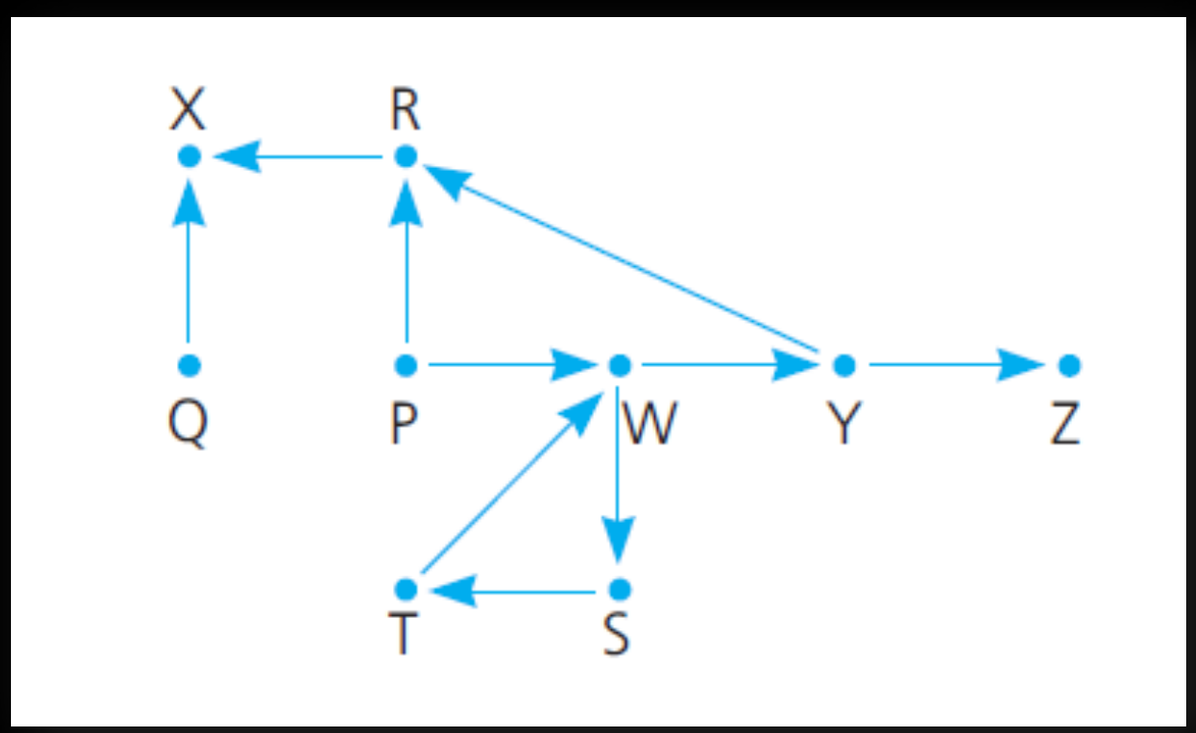
C = visited

C = backtracked

Avoid dead end by backtracking

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z



Backtracking

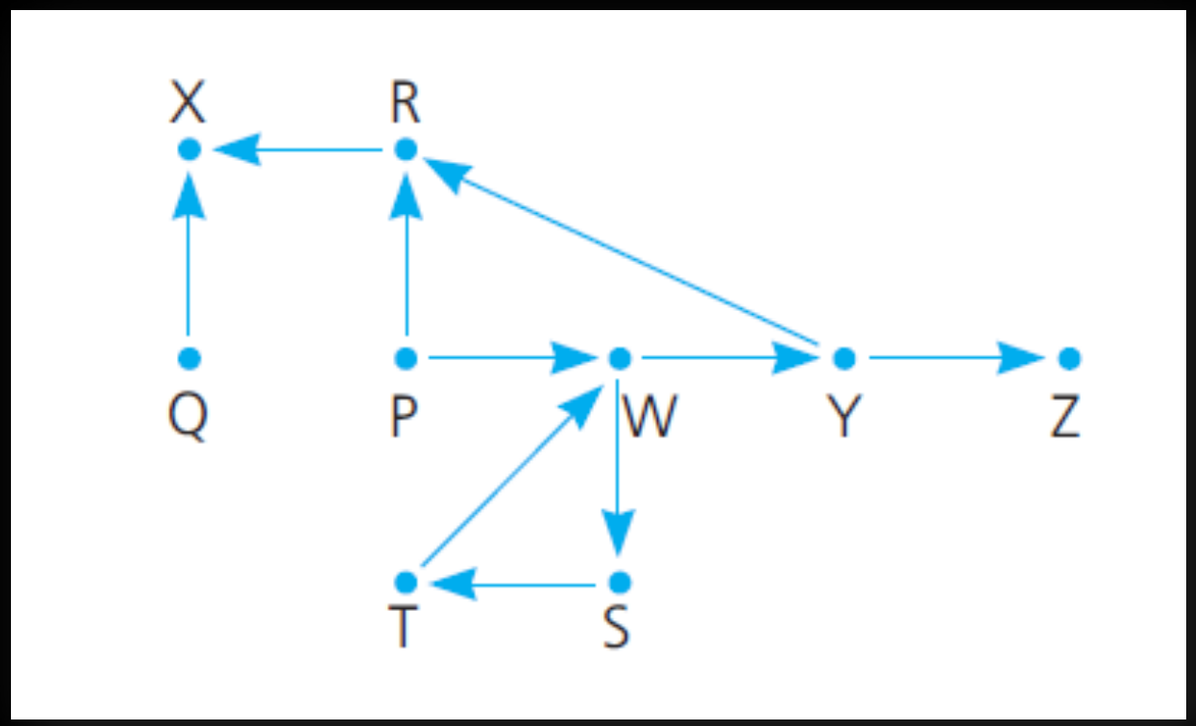
Avoid dead end by backtracking

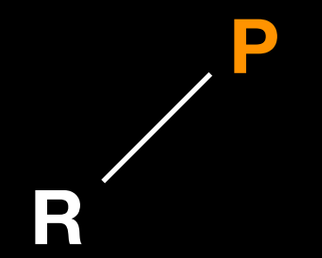
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

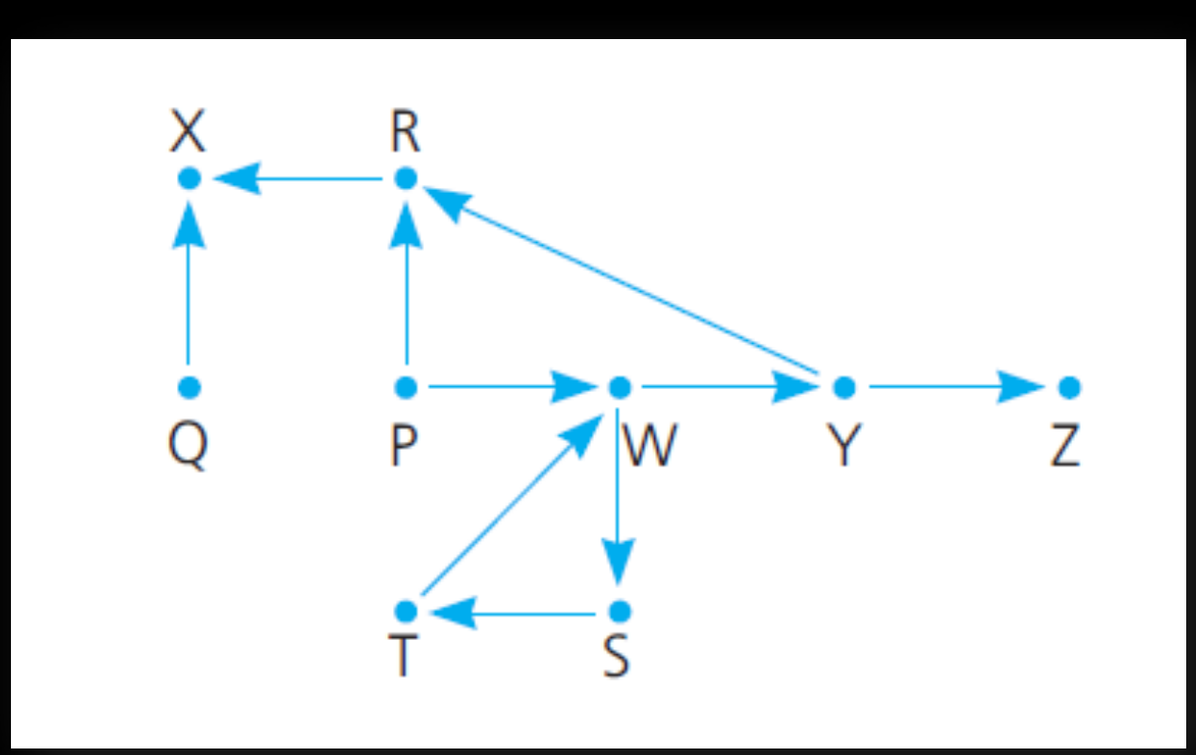
Avoid dead end by backtracking

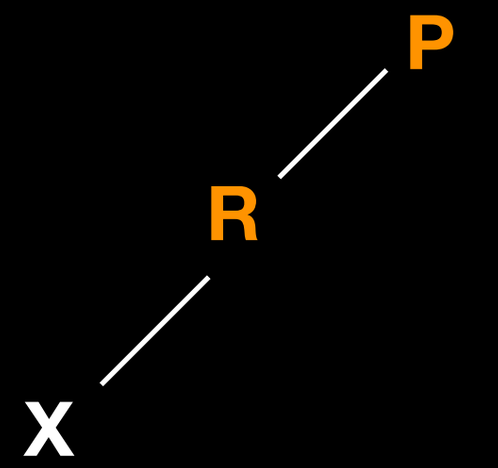
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

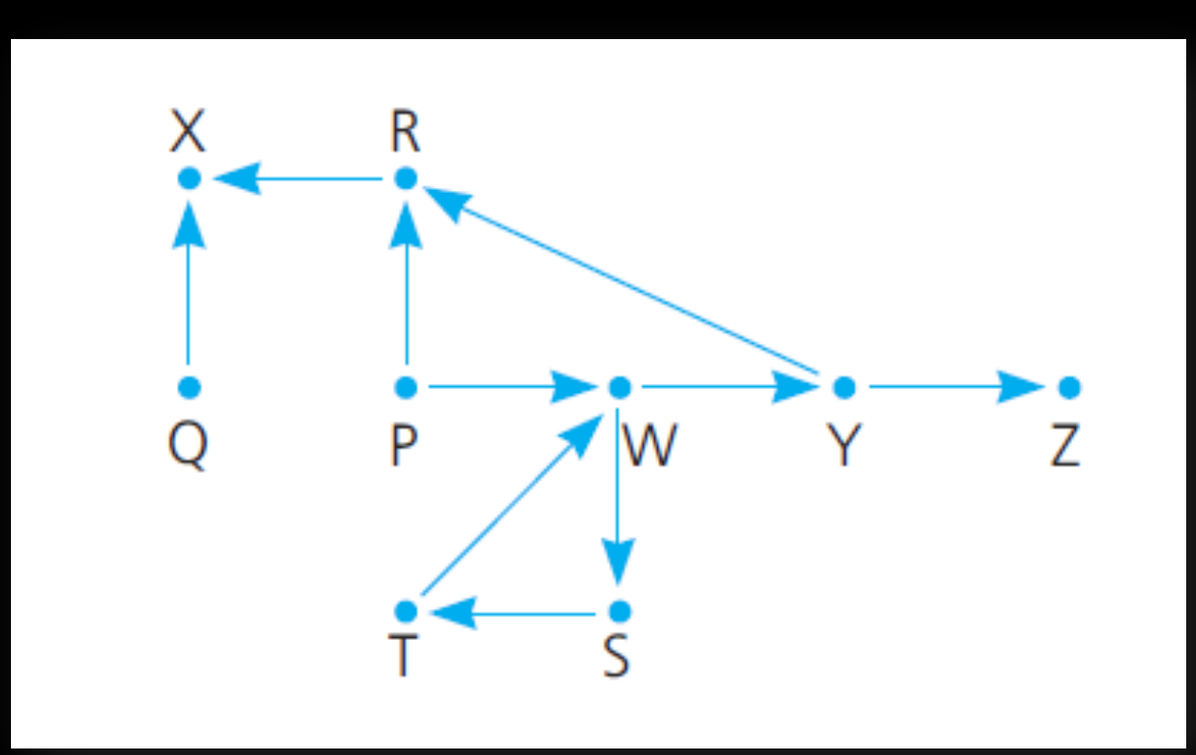
Avoid dead end by backtracking

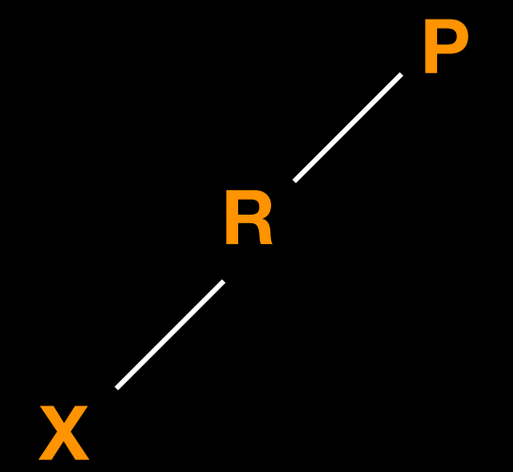
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

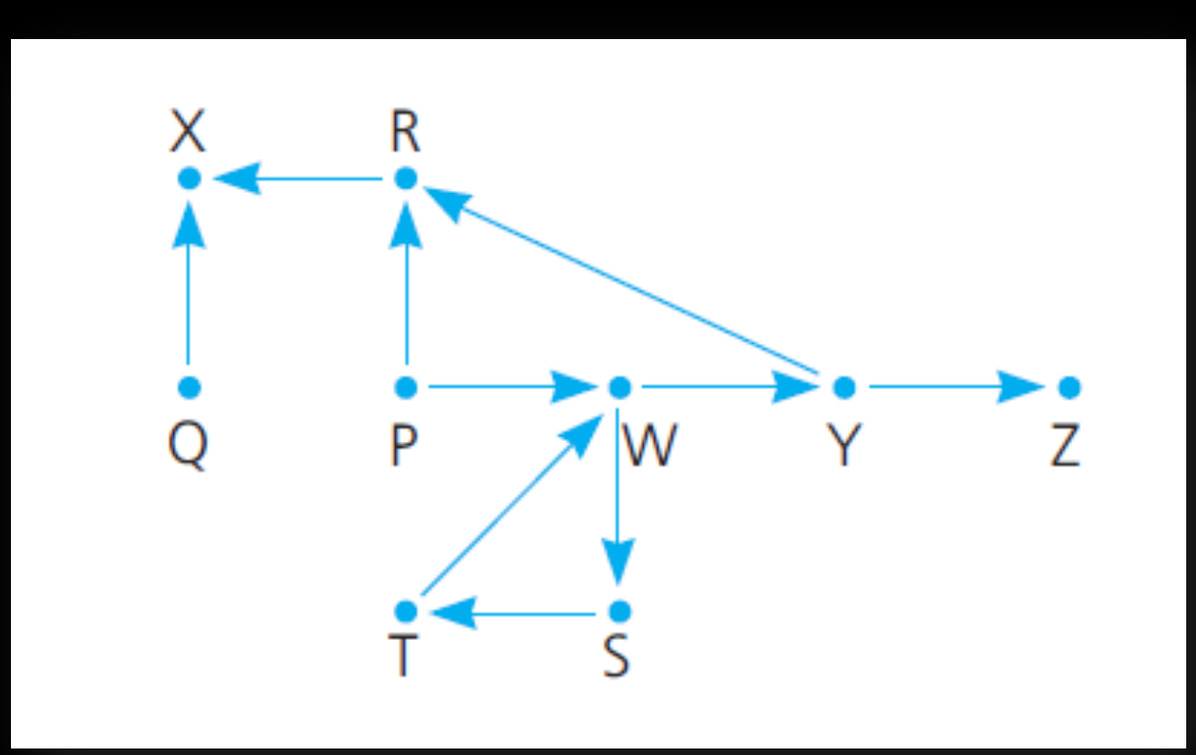
Avoid dead end by backtracking

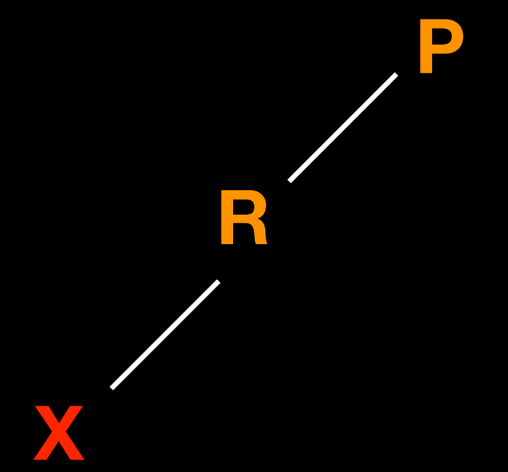
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

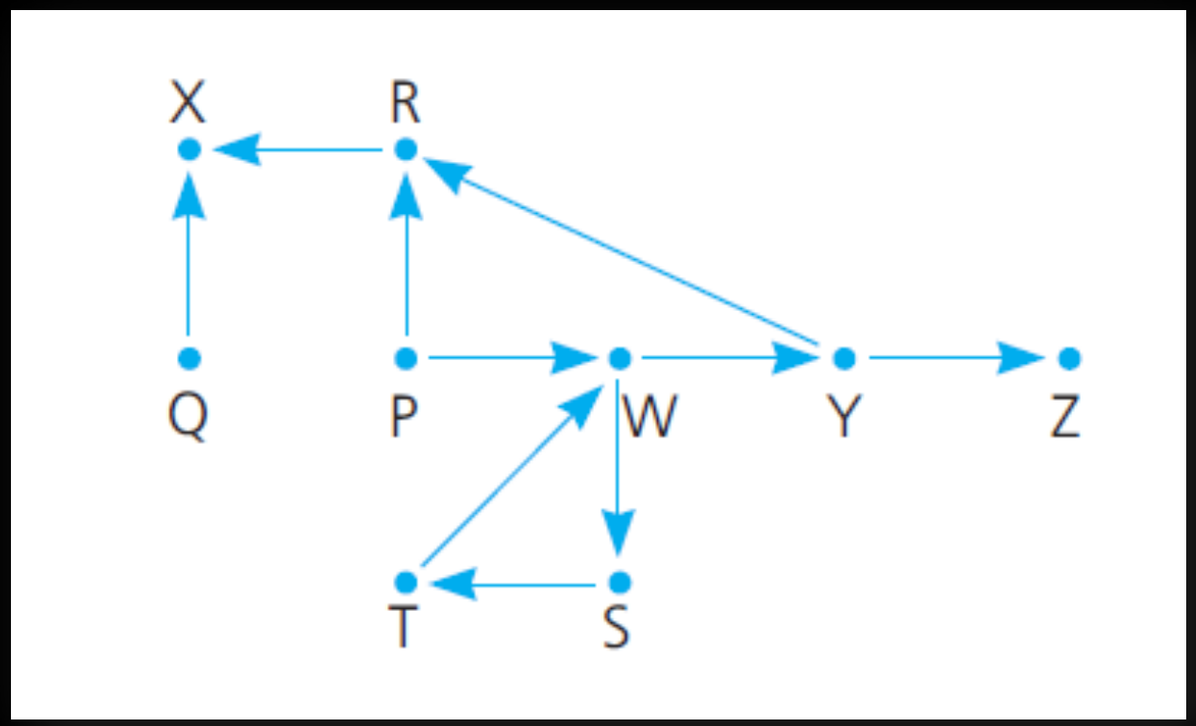
Avoid dead end by backtracking

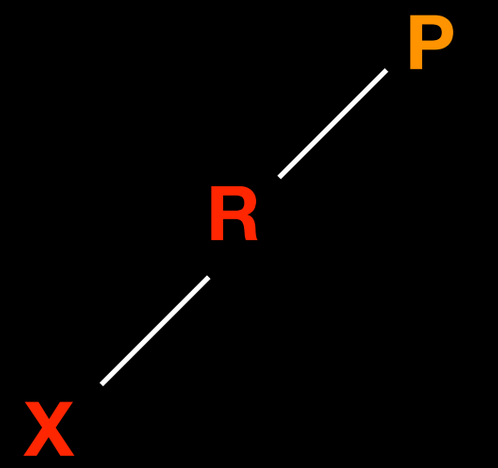
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

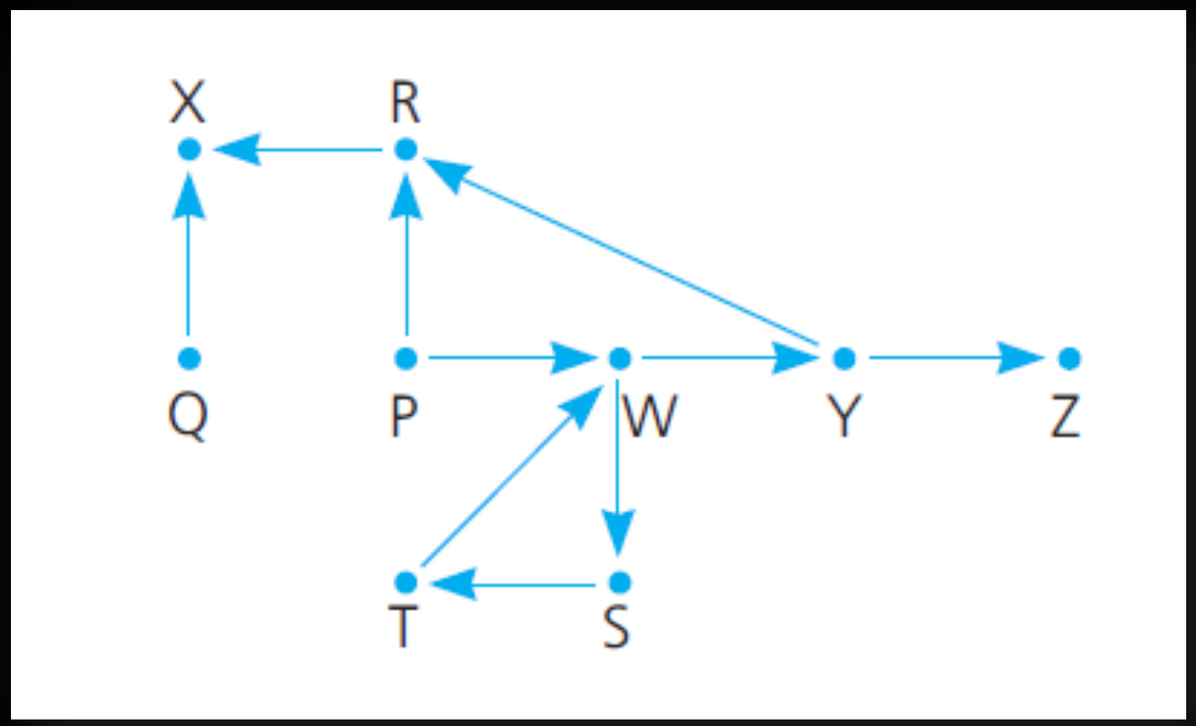
Avoid dead end by backtracking

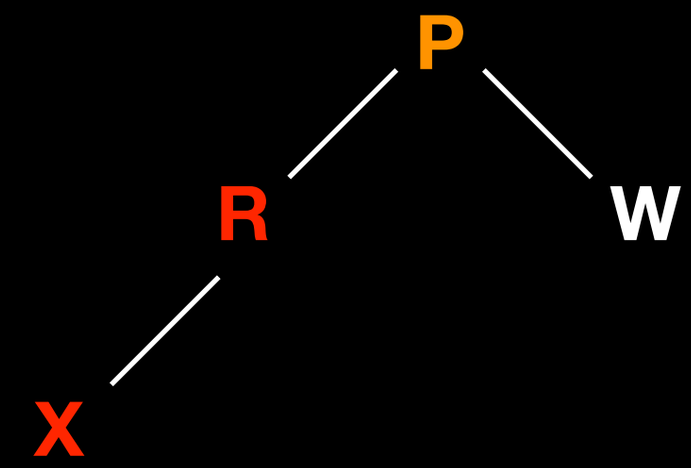
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

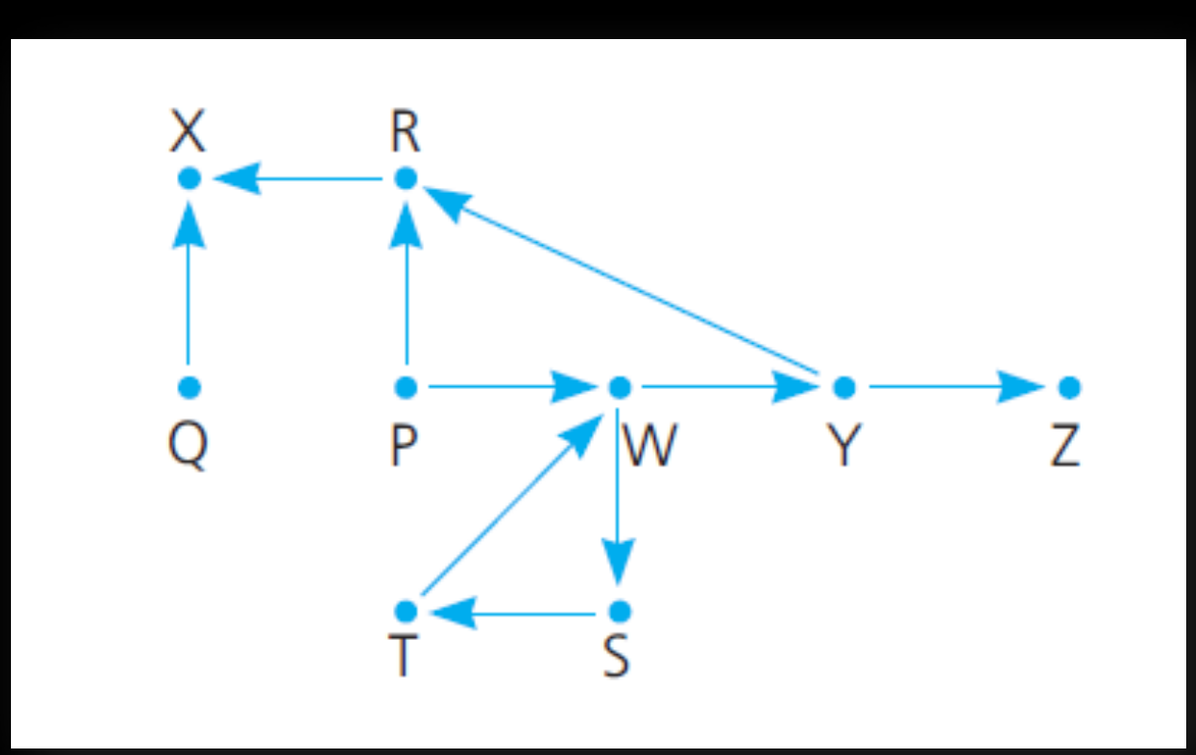
Avoid dead end by backtracking

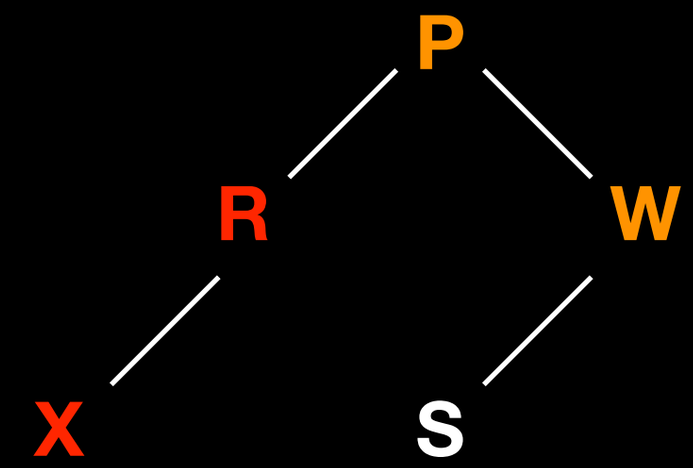
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

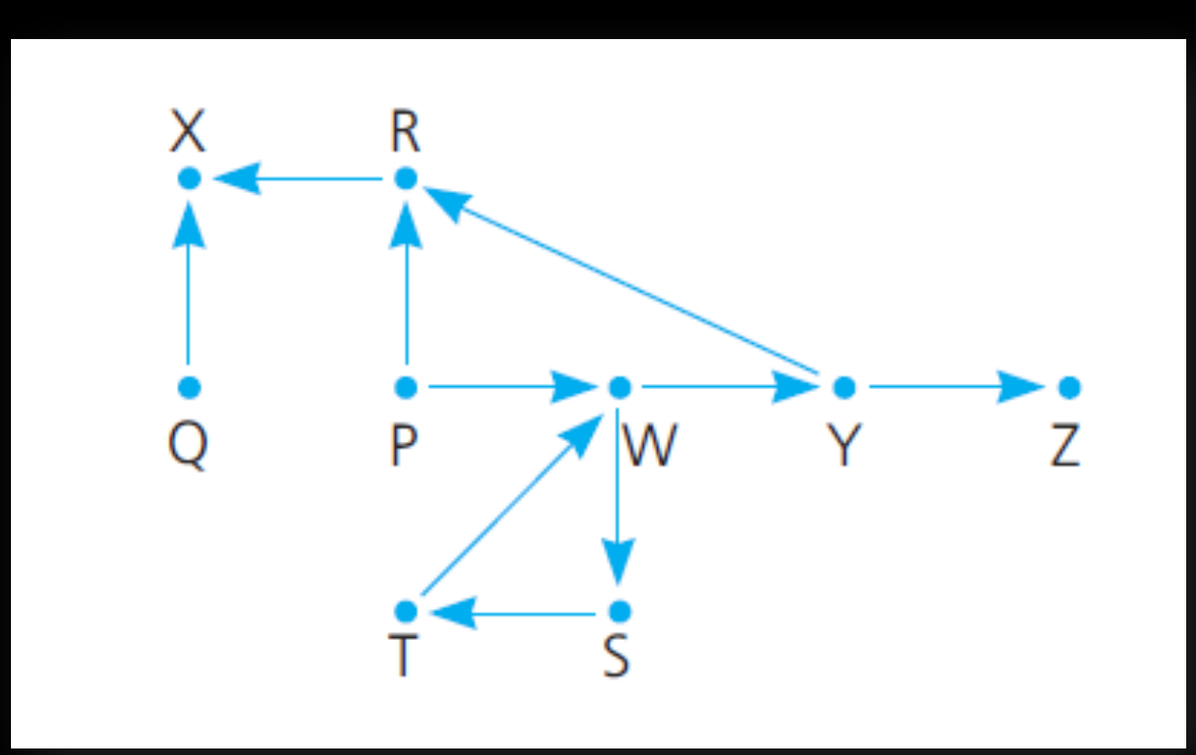
Avoid dead end by backtracking

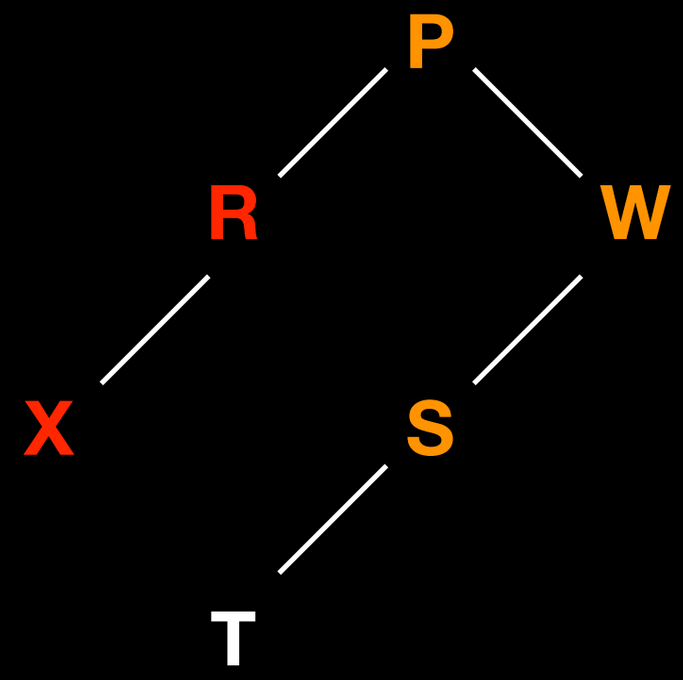
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

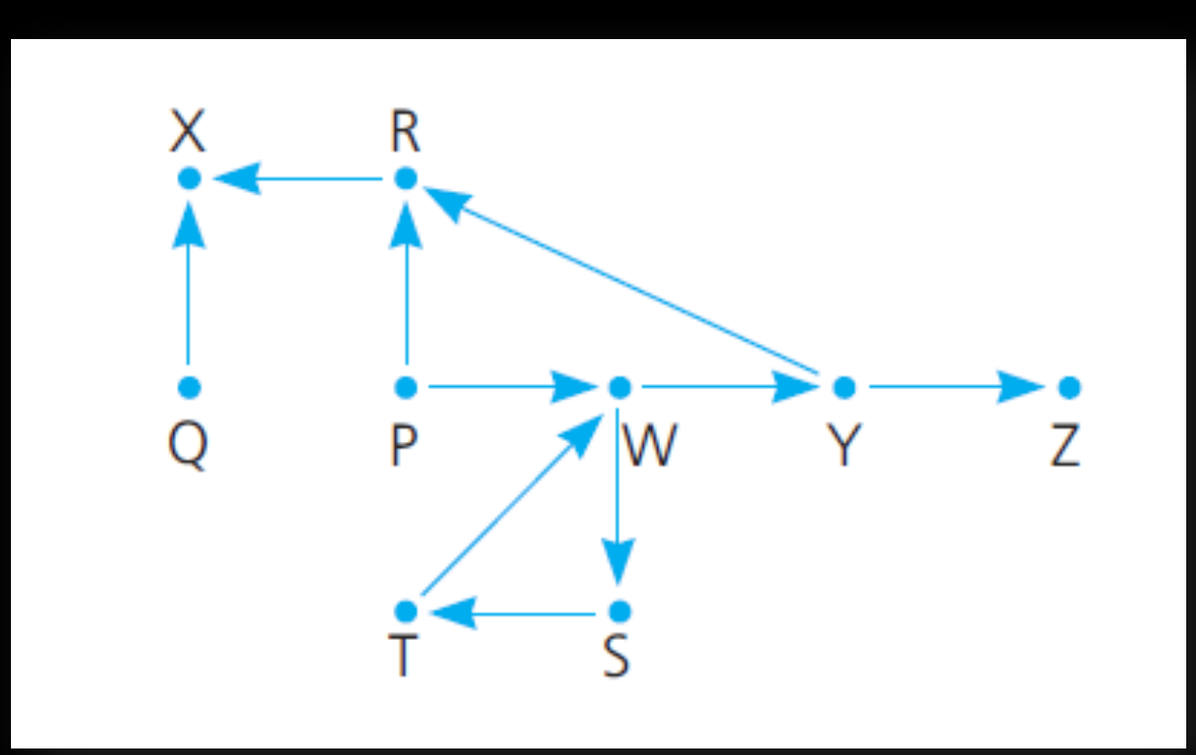
Avoid dead end by backtracking

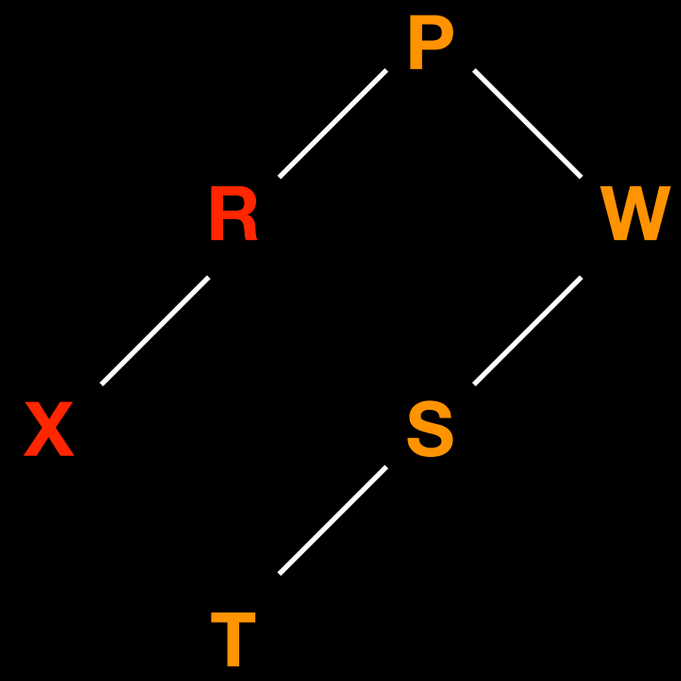
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

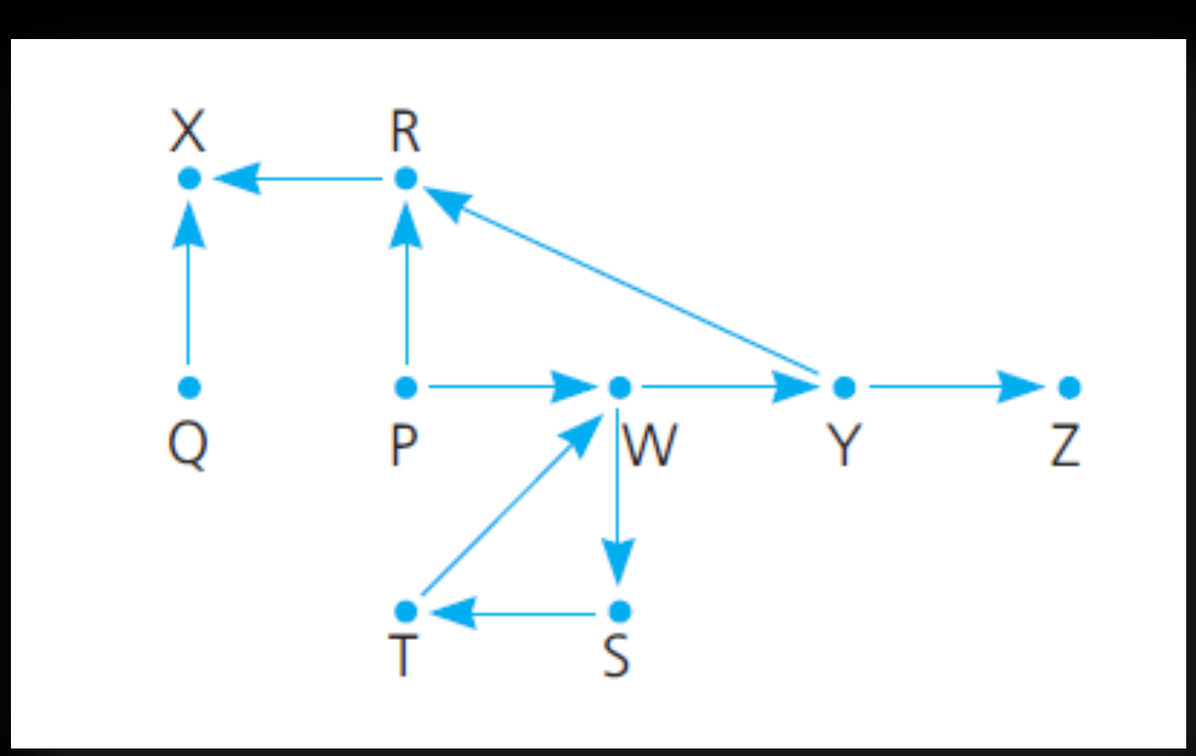
Avoid dead end by backtracking

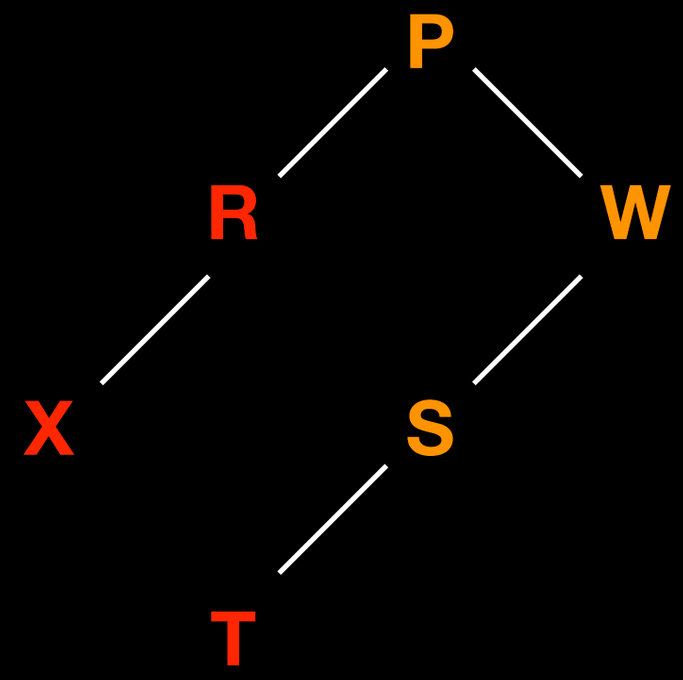
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

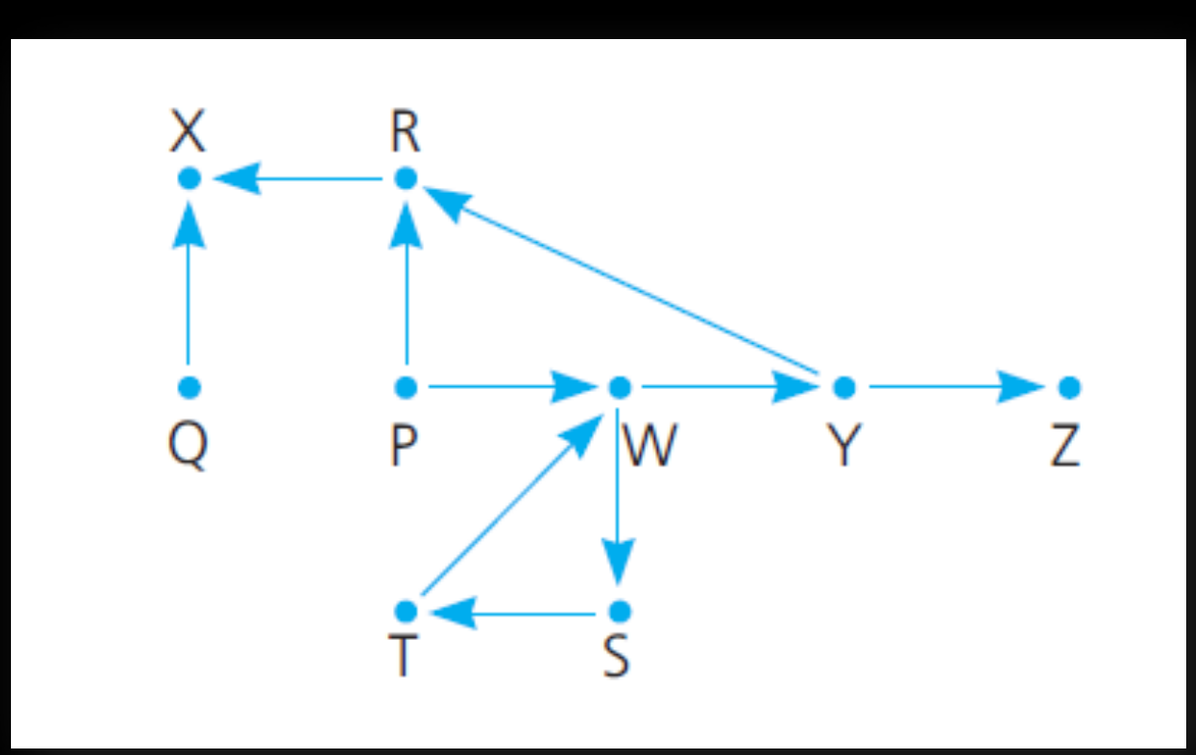
Avoid dead end by backtracking

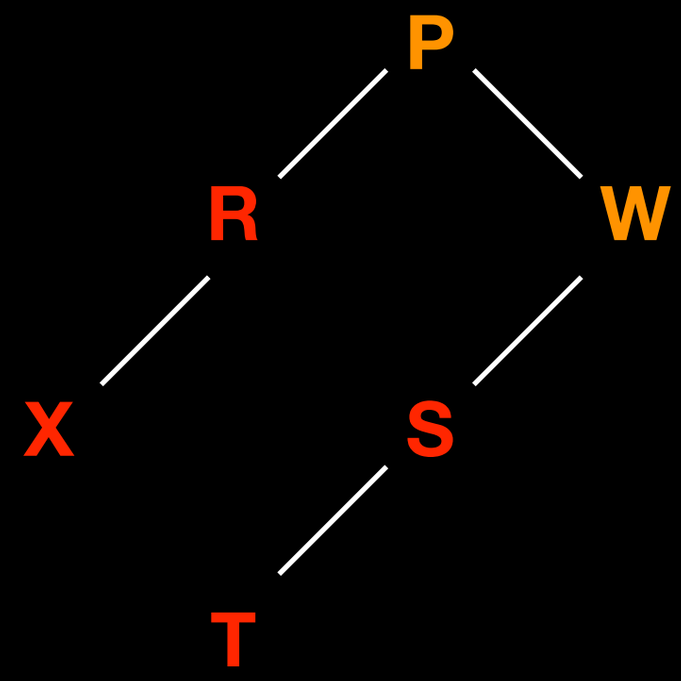
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

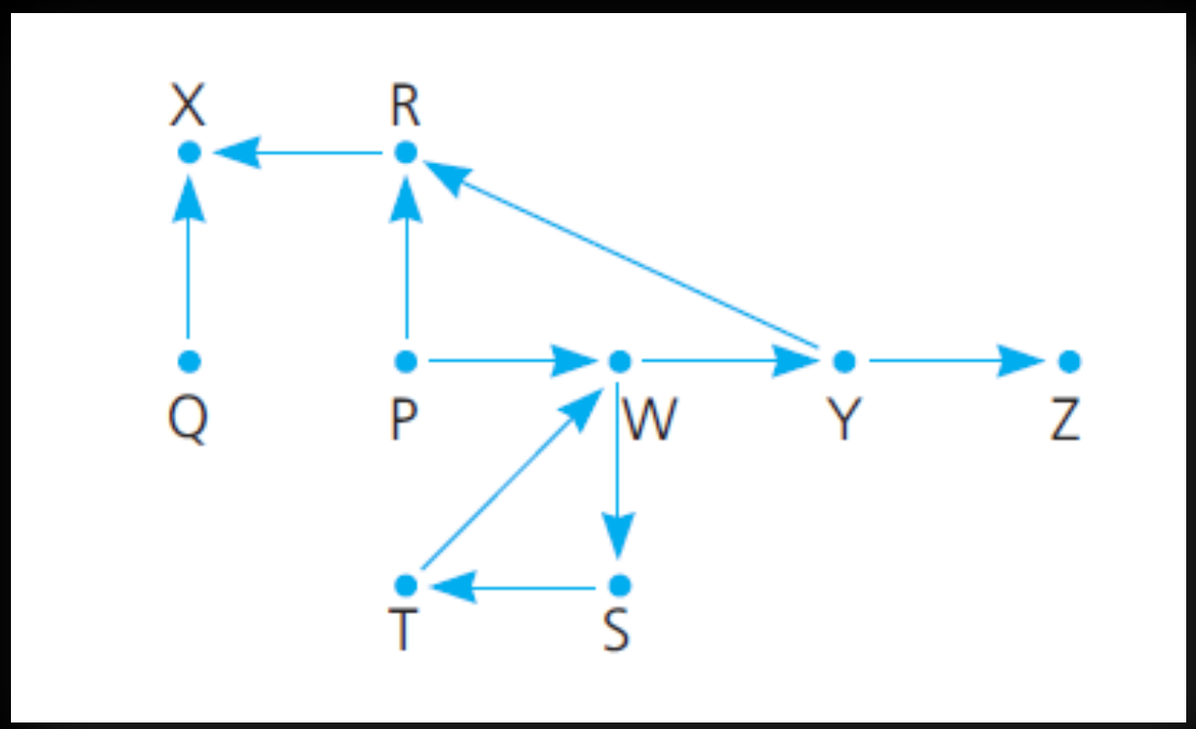
Avoid dead end by backtracking

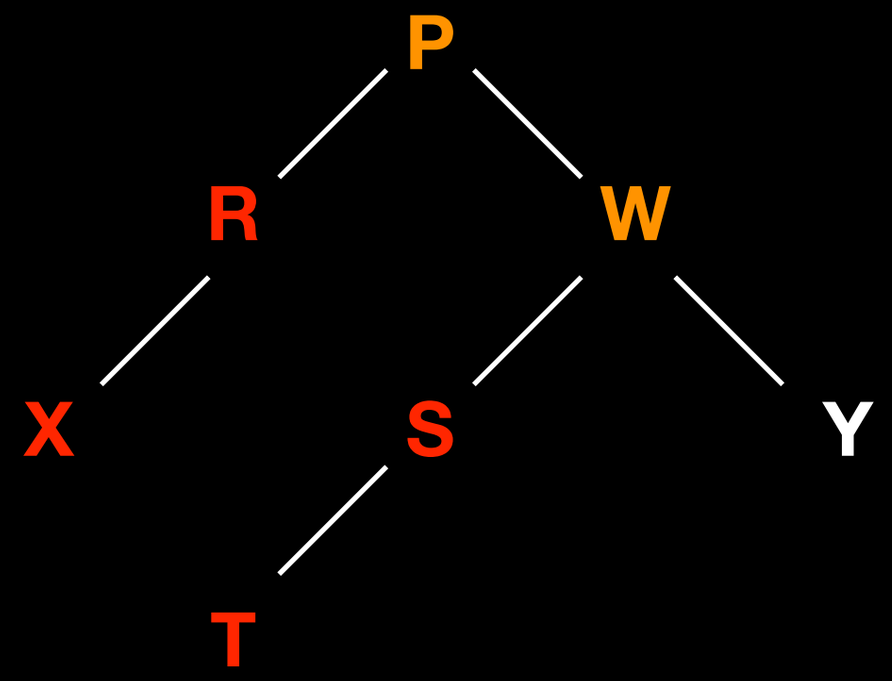
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

Origin = P , Destination = Z





Backtracking

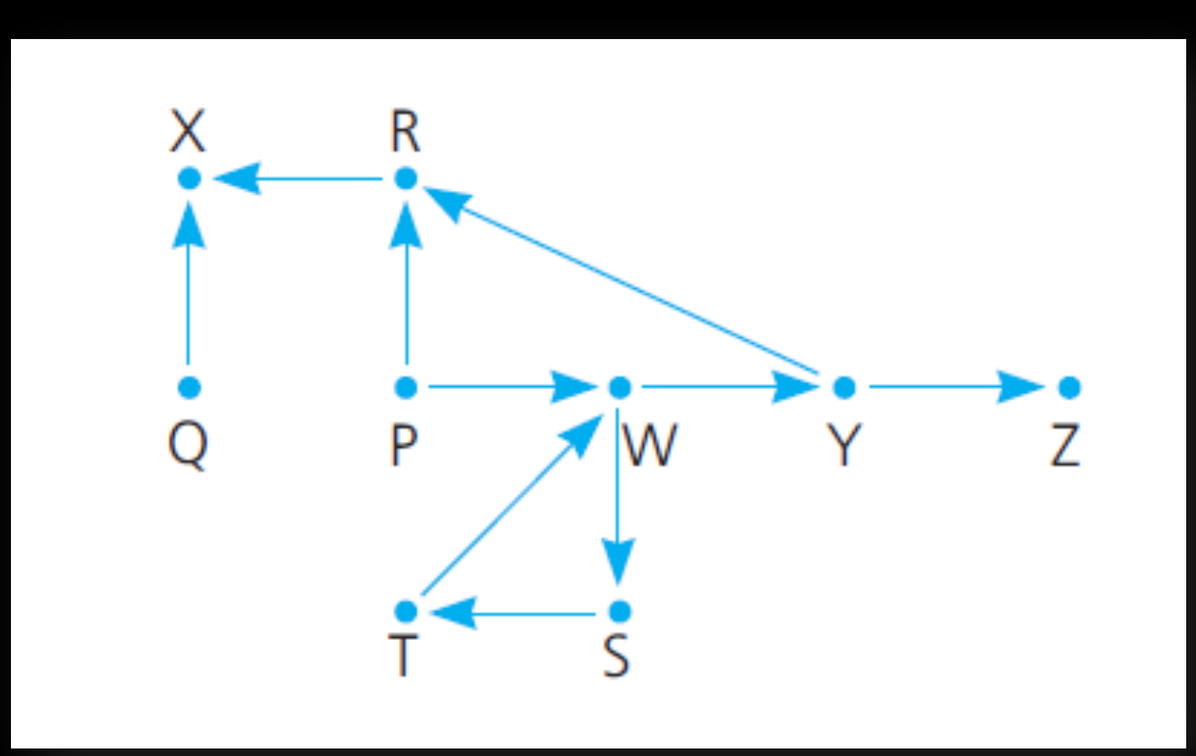
Avoid dead end by backtracking

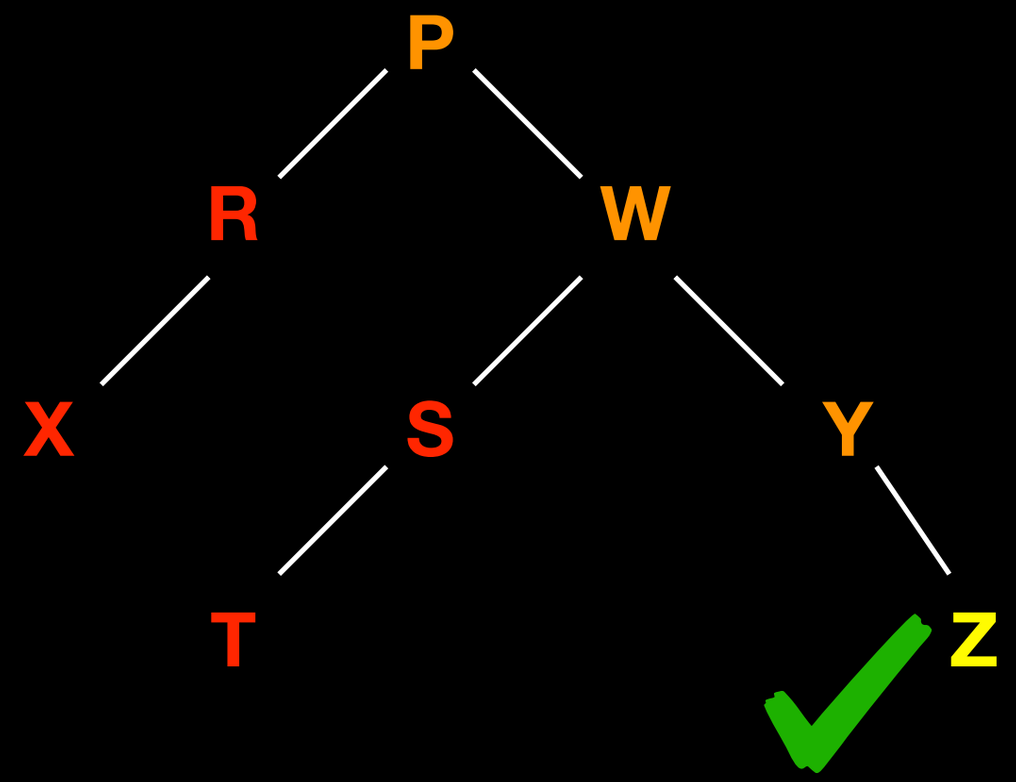
C = visited

C = backtracked

Avoid traveling in circles by marking visited cities

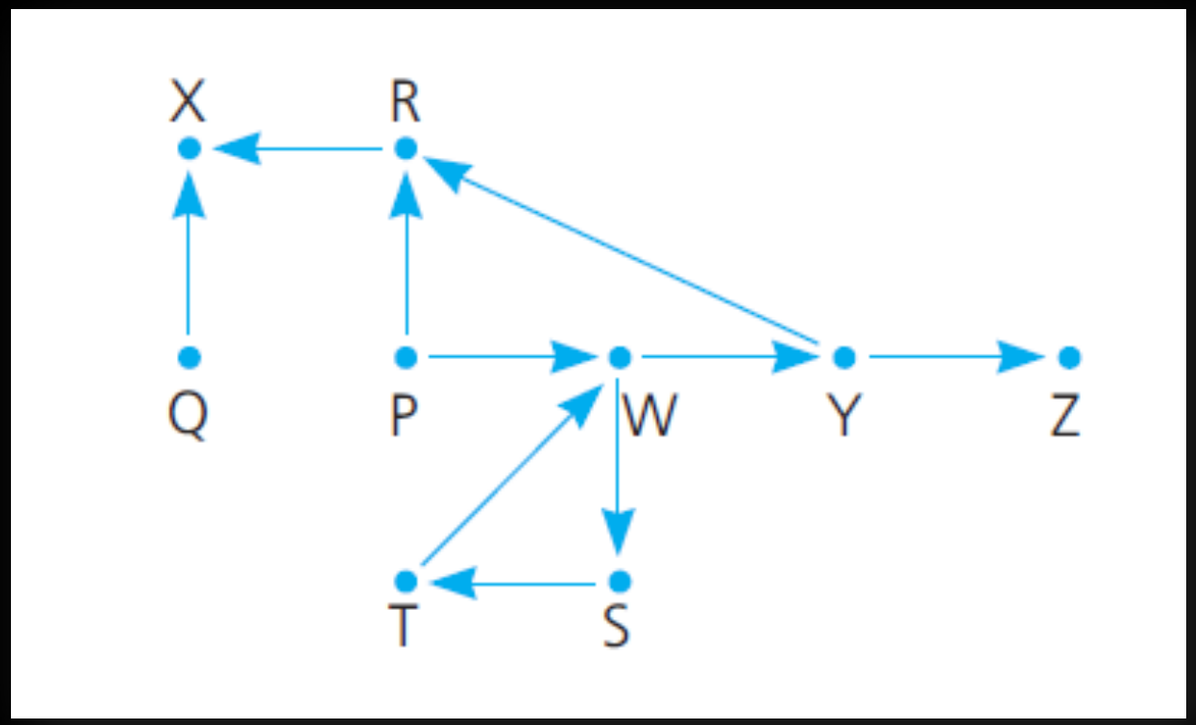
Origin = P , Destination = Z

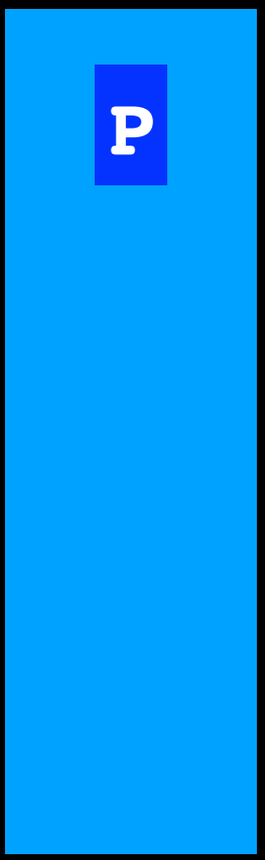




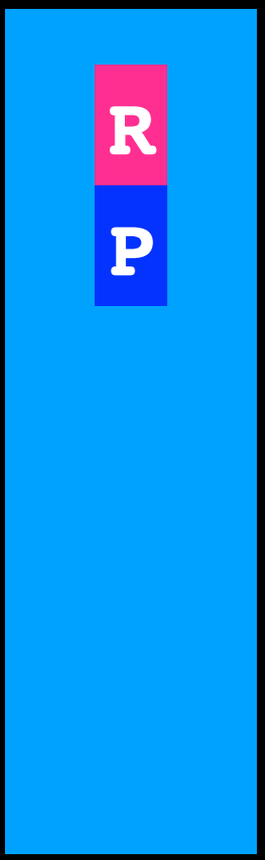
Backtracking

Origin = P , Destination = Z

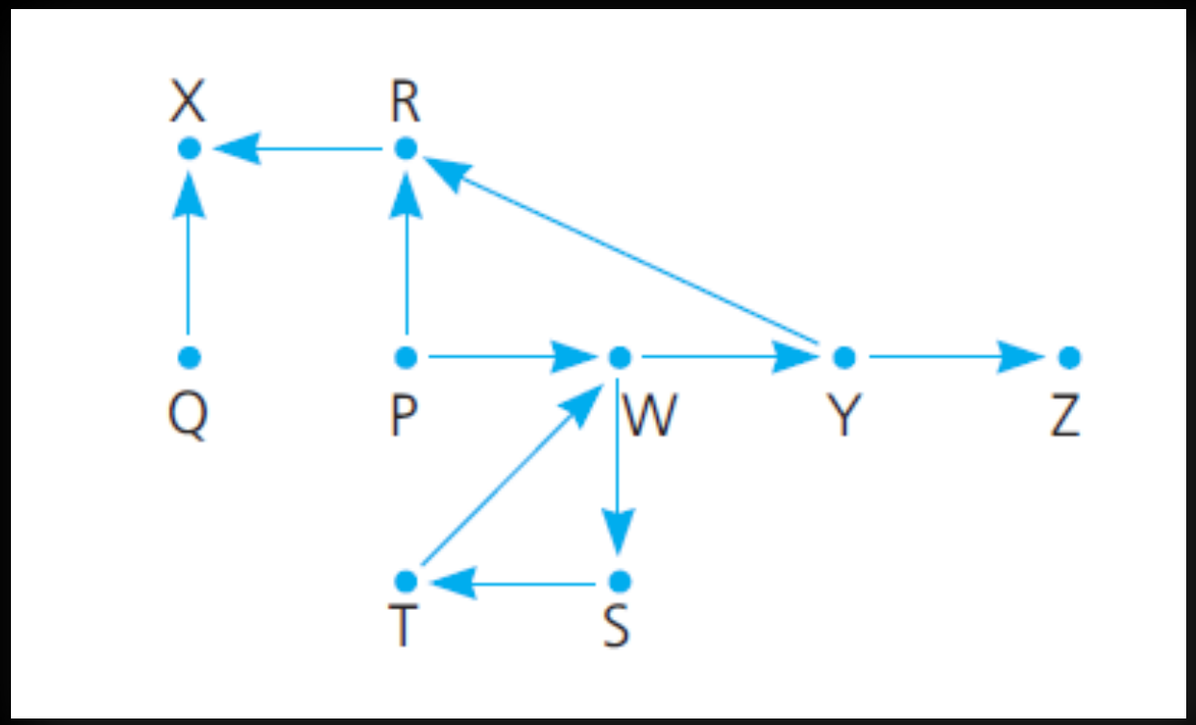




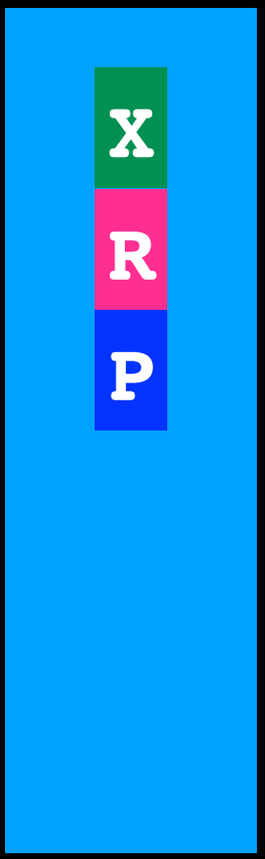
Backtracking



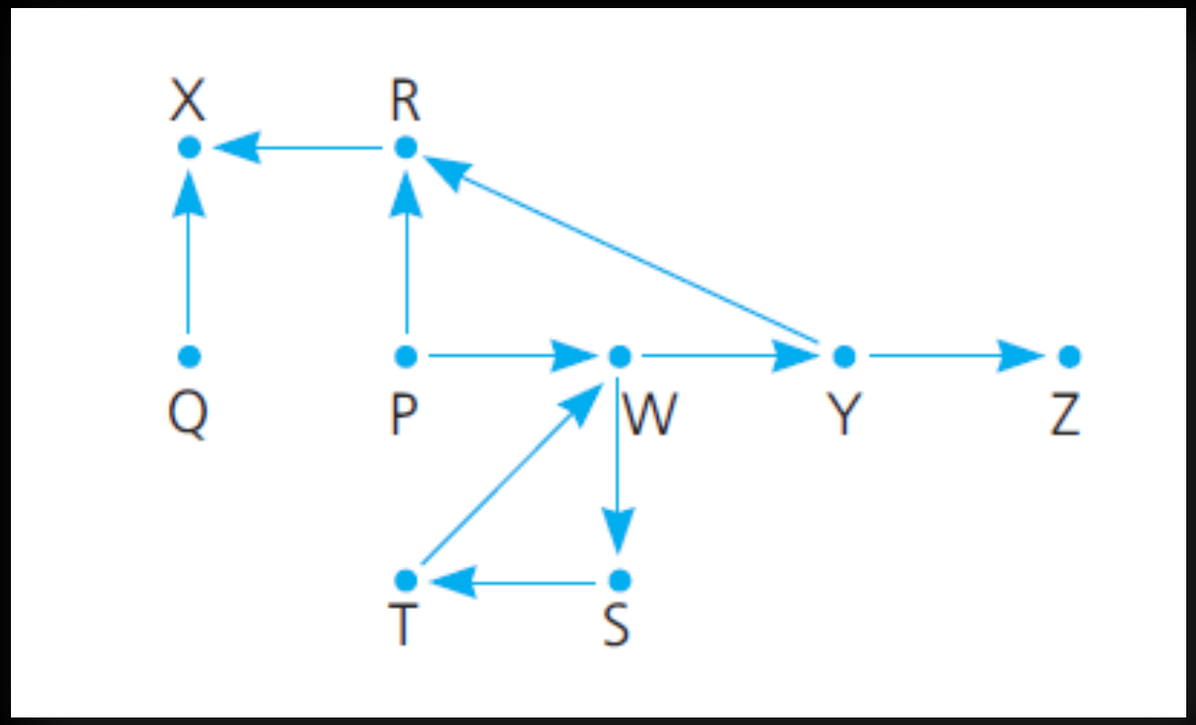
Origin = P , Destination = Z



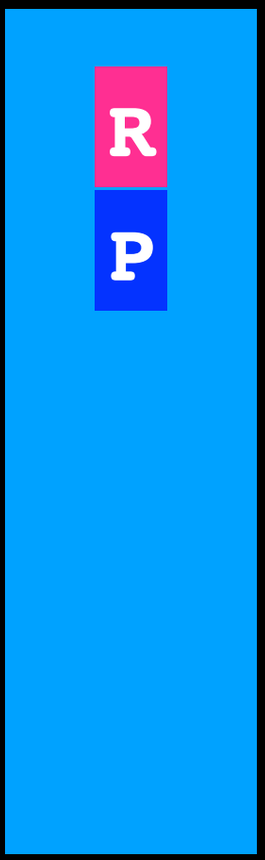
Backtracking



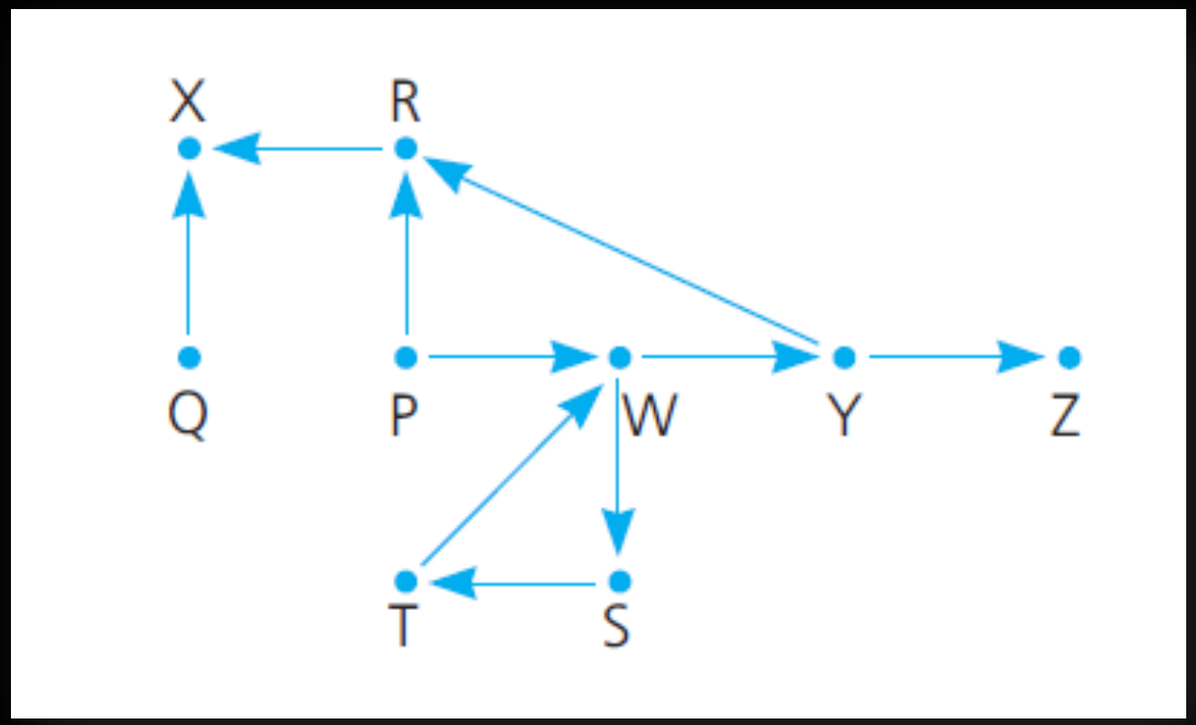
Origin = P , Destination = Z



Backtracking

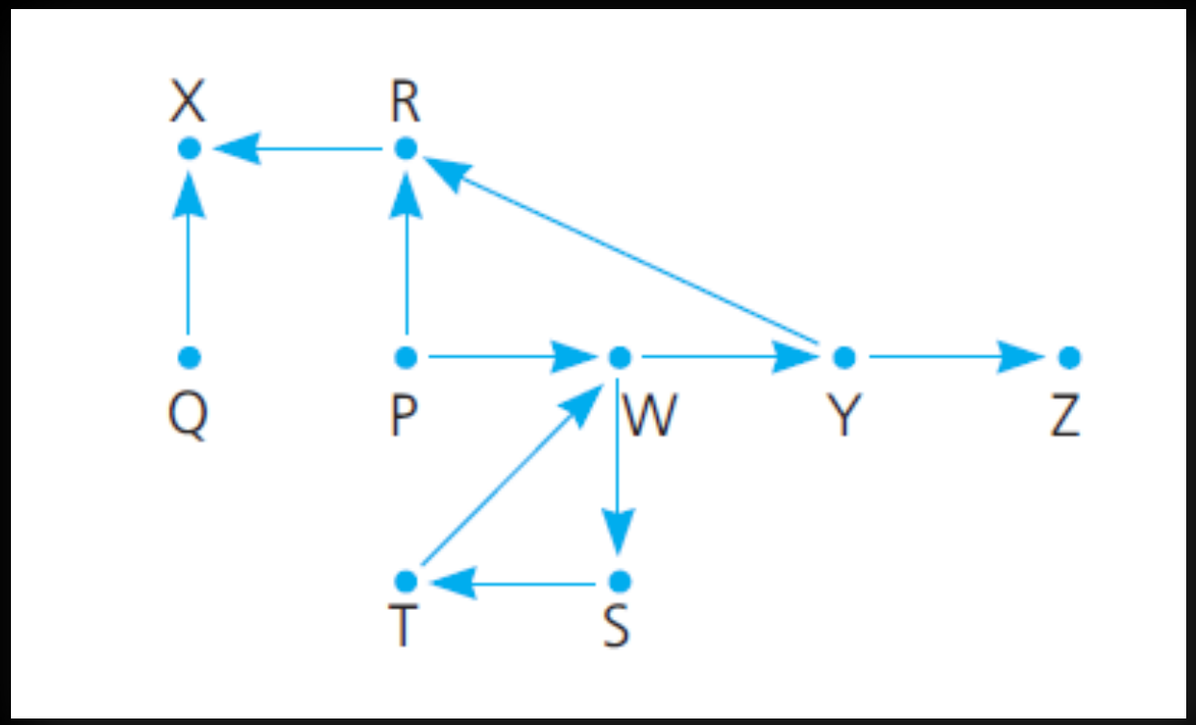


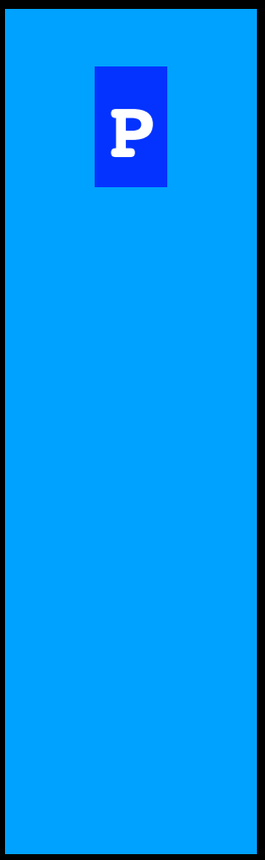
Origin = P , Destination = Z



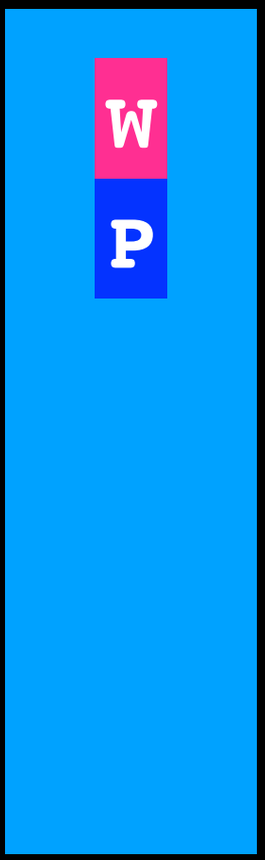
Backtracking

Origin = P , Destination = Z

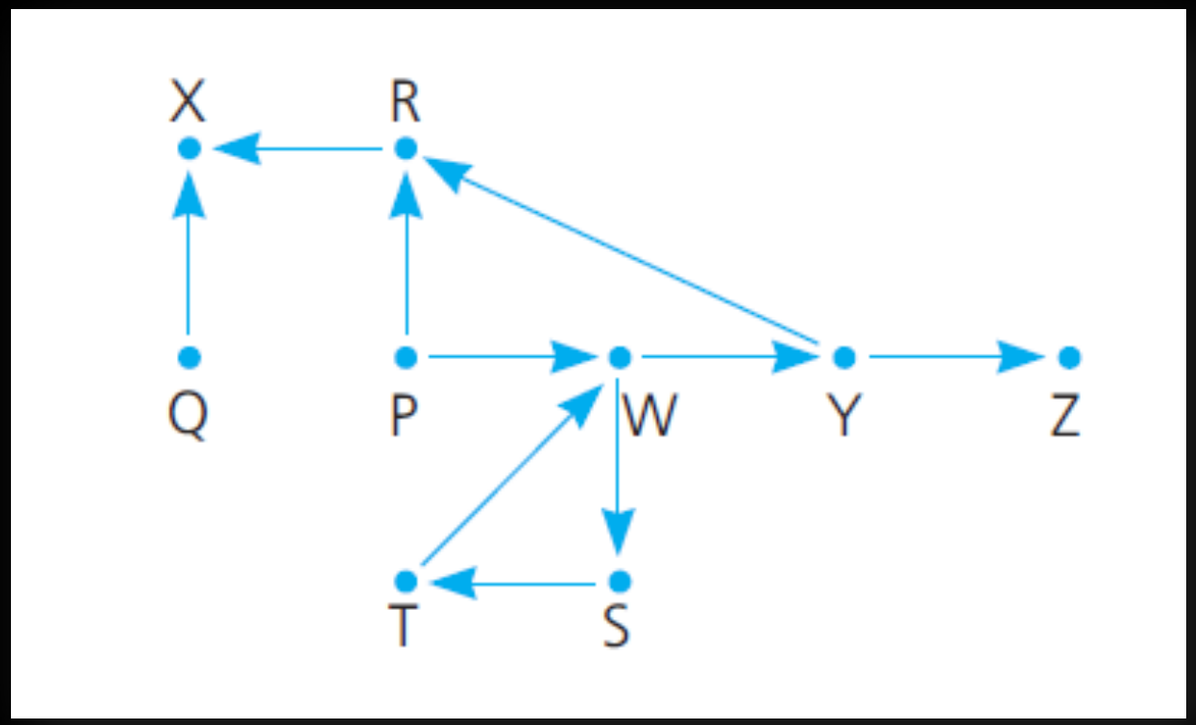




Backtracking

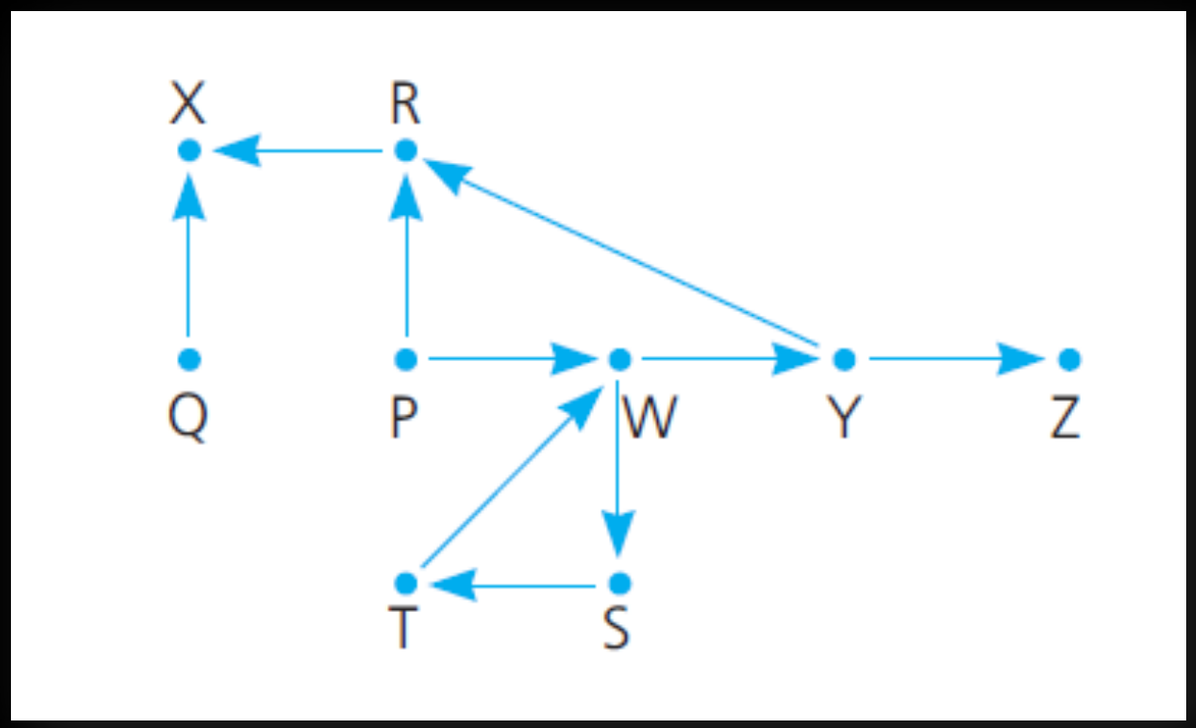


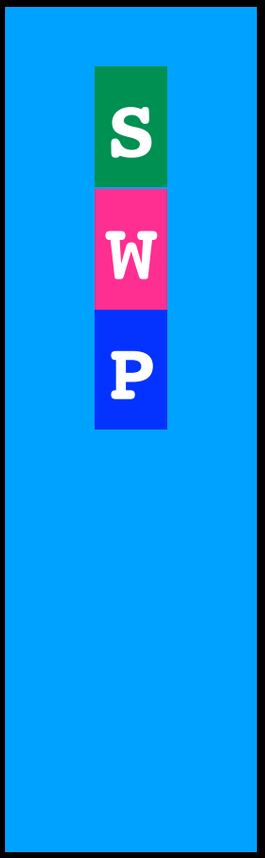
Origin = P , Destination = Z



Backtracking

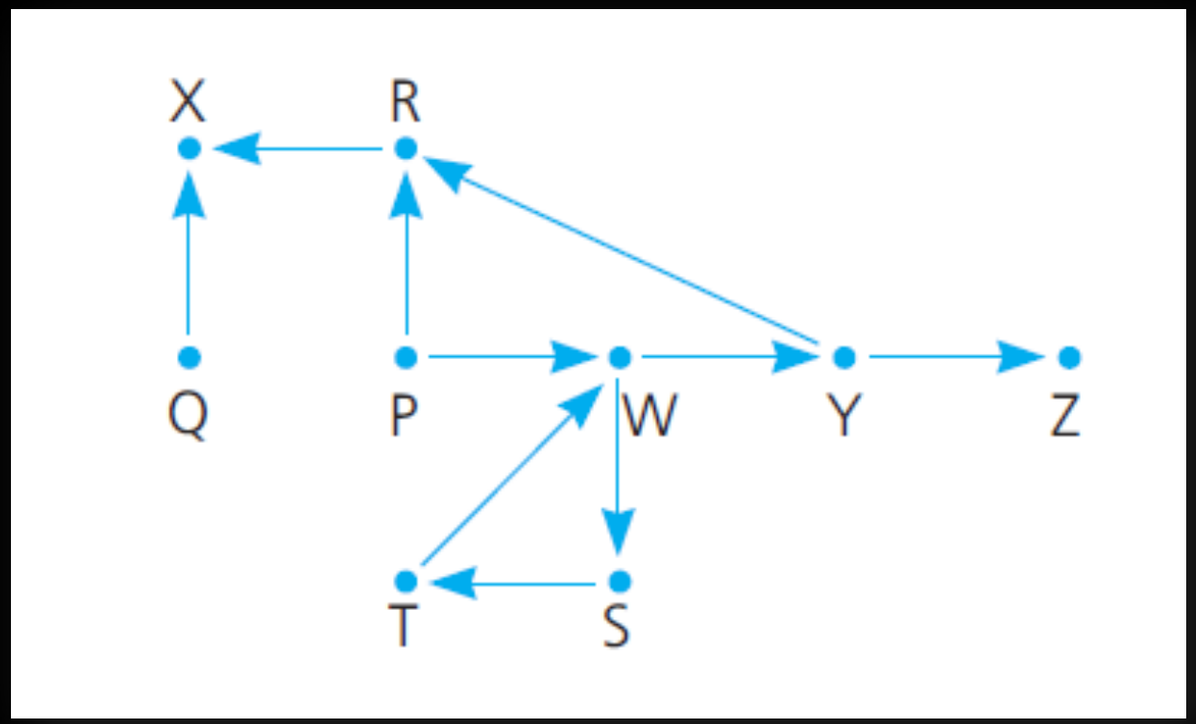
Origin = P , Destination = Z

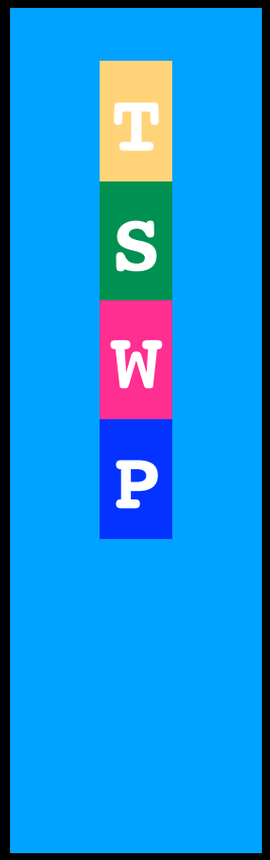




Backtracking

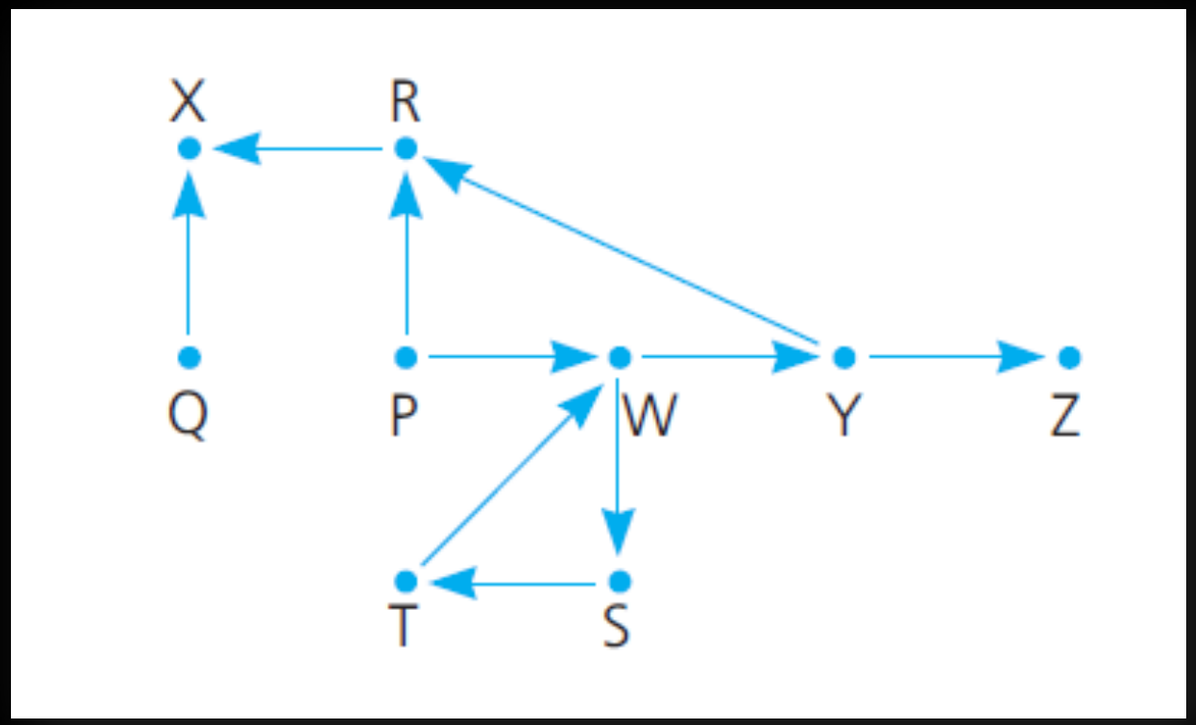
Origin = P , Destination = Z

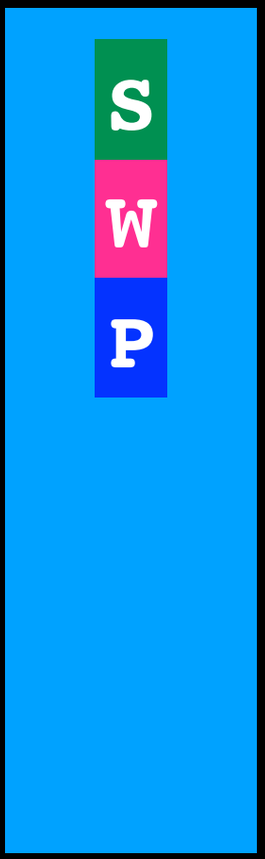




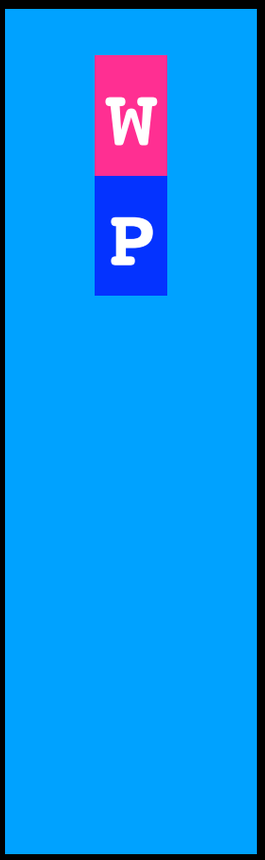
Backtracking

Origin = P , Destination = Z

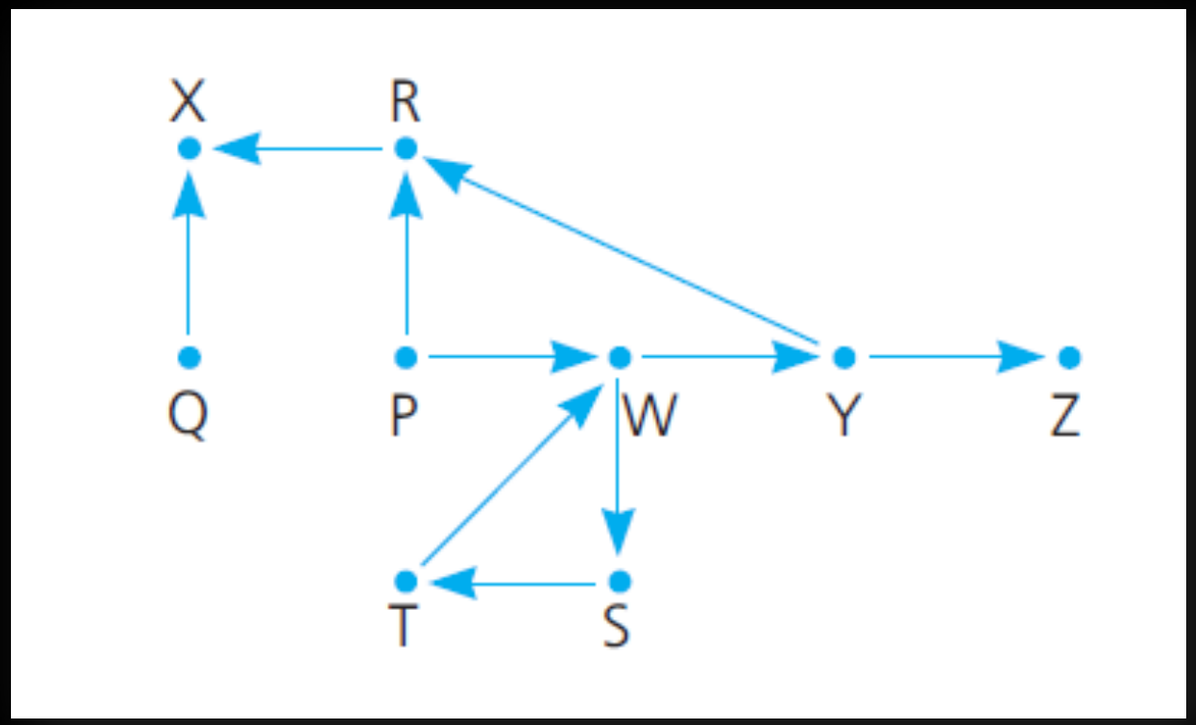




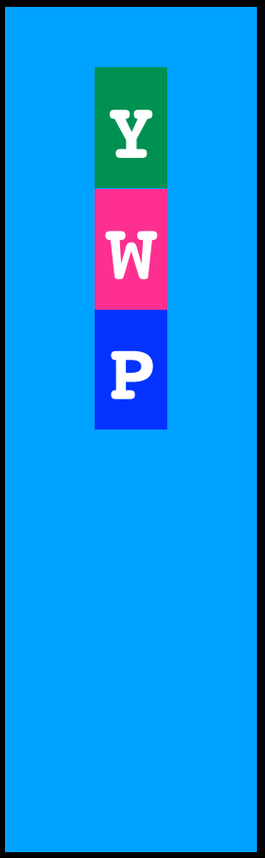
Backtracking



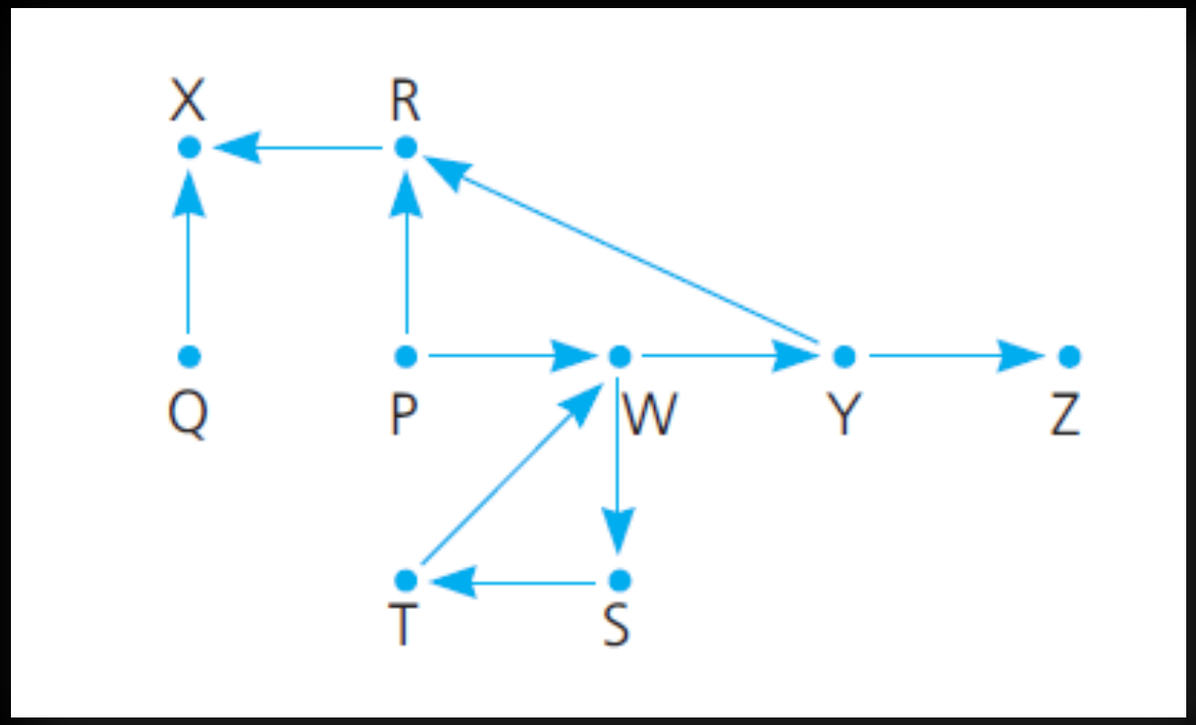
Origin = P , Destination = Z



Backtracking

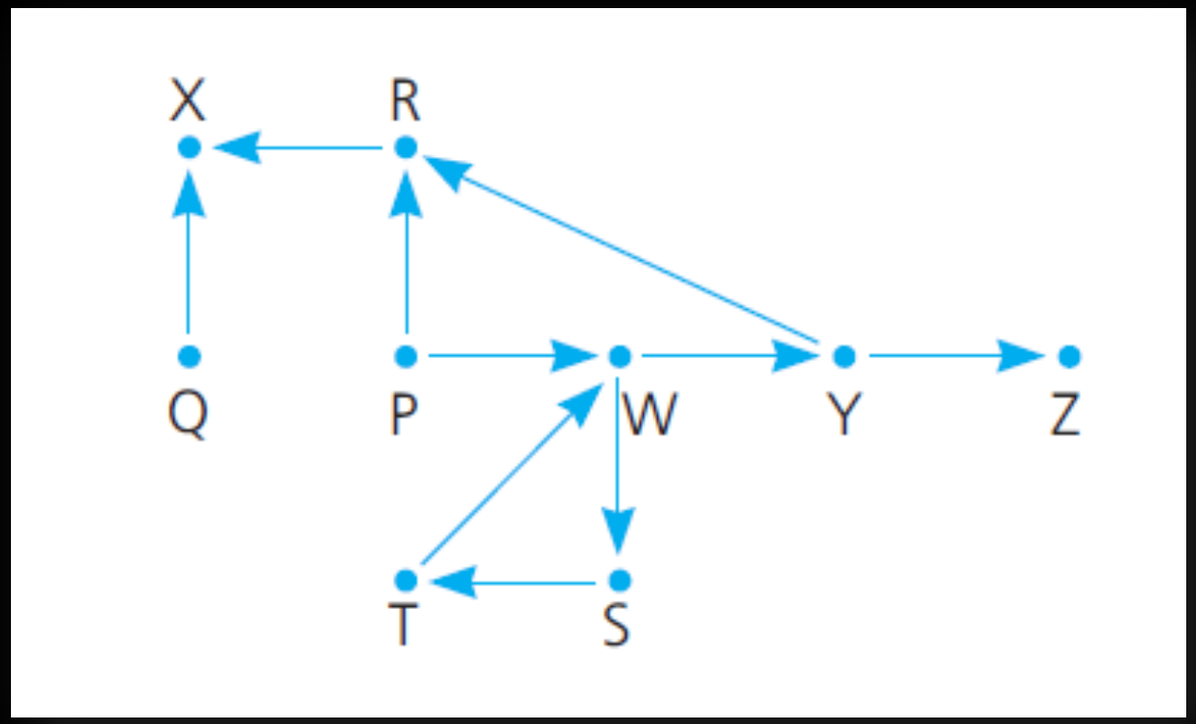


Origin = P , Destination = Z



Backtracking

Origin = P , Destination = Z



Backtracking

```
while(not found flights from origin to destination)
{
   if no flight exists from city on top of stack to
   unvisited destination
       pop the stack //BACKTRACK
   else
   {
       select an unvisited city C accessible from city
       currently at top of stack
       push C on stack
       mark C as visited
    }
}
```

Program Stack and Recursion

Recursion works because function waiting for result/ return from recursive call are on program stack

Order of execution determined by **stack**

More Applications

Balancing anything!

-html tags (e.g matches

Reverse characters in a word or words in a sentence

Undo mechanism for editors or backups

Traversals (graphs / trees)

. . .

Stack ADT

```
#ifndef STACK_H_
#define STACK_H_

template<<typename ItemType>
class Stack
{

public:
    Stack();
    void push(const ItemType& new_entry); //adds an element to top of stack
    void pop(); // removes element from top of stack
    ItemType top() const; // returns a copy of element at top of stack
    int size() const; // returns the number of elements in the stack
    bool isEmpty() const;//returns true if no elements on stack, else false

private:
          //implementation details here


};    //end Stack

#include "Stack.cpp"
#endif // STACK_H_`
```