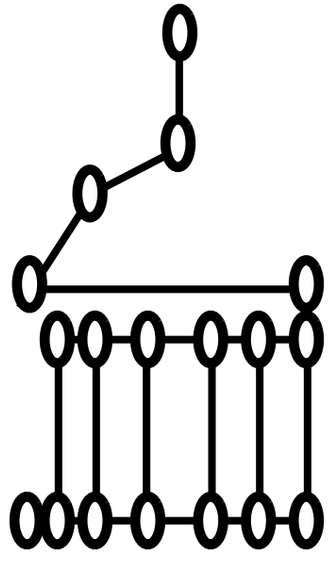
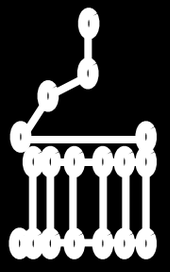
Lecture 31: More on using standard C++ libraries

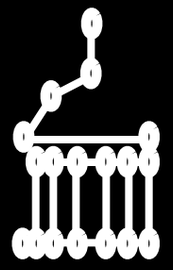
57:017 Computers in Engineering
Fall 2014





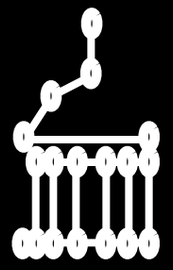
Reminders/announcements

- Use clicker channel 14 (Go/Ch →14 → Go/Ch)
- **Mini-assignment** due **4/15** (Wednesday) by **12:30 p.m.**
  - ICON quiz over use of standard C++ libraries
- Homework 5 (individual assignment) due **4/22** (Friday) by **11:59 p.m.**
- Checkout a working copy of your **individual** repository today:
  - svn co $CIE/hawkID --username=hawkID



Major topics of CIE

Part I	Part II	Part III
Fundamental C programming concepts with engineering applications	Advanced C programming (including dynamic data structures)	Object-oriented programming with C++
Chapters 1-8; parts of chapter 12	Chapters 10 and 12	Chapters 15-18, 20 and 22



Major topics of CIE

Part III

Procedural programming using C++

Monday, 11/3

Introduction to object-oriented programming

Since Wednesday, 11/5

classes/objects

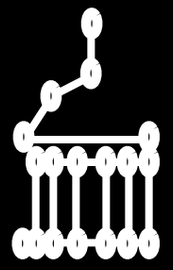
inheritance

templates

standard template library

Object-oriented programming with C++

Chapters 15-18, 20 and 22



Recall (from Wednesday, 11/5): basic C++ class/object concepts

class "interface"

```
#ifndef COMPLEXNUMBER_H
#define COMPLEXNUMBER_H
class ComplexNumber
{
public:
    void setRealPart(double real);
    void setImagPart(double imag);
    double getRealPart( ) const;
    double getImagPart( ) const;
    double getMagnitude( ) const;
    double getPhaseAngleInRadians( ) const;
private:
    double m_real;
    double m_imag;
};
#endif
```

ComplexNumber.h

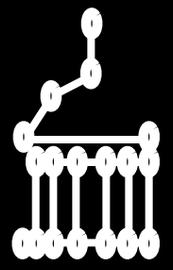
```
#include "ComplexNumber.h"
#include <cmath>

void ComplexNumber::setRealPart(double real)
{
    m_real = real;
}
void ComplexNumber::setImagPart(double imag)
{
    m_imag = imag;
}
double ComplexNumber::getRealPart( ) const
{
    return m_real;
}
double ComplexNumber::getImagPart( ) const
{
    return m_imag;
}
...
```

ComplexNumber.cpp

class "implementation"

main.cpp

```
#include <iostream>
#include "ComplexNumber.h"
int main()
{
    ComplexNumber c1;
    c1.setRealPart(3);
    c1.setImagPart(4);
    std::cout << c1.getMagnitude() << std::endl;
}
```



Recall (from Friday, 11/7): use of example classes from the standard C++ library

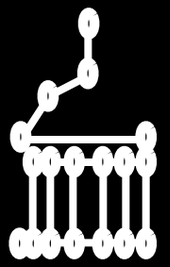
```
void stringClickerQuestion()
{
    std::string myDog1 = "Teddy";
    std::string myDog2 = "JT";
    std::string generalSentence = "dog and dog are very cute.";
    generalSentence.replace(generalSentence.find("dog"), 3, myDog1);
    generalSentence.replace(generalSentence.find("dog"), 3, myDog2);
    std::cout << generalSentence << std::endl;
}
```

std::string

```
void vectorExample()
{
    std::vector<int> myNumbers;
    std::cout << "initial size of int vector = " << myNumbers.size() << std::endl;
    for (int n=0; n < 10; n++)
    {
        myNumbers.push_back(n); // add to end
    }
    for (size_t i=0; i < myNumbers.size(); i++)
    {
        std::cout << myNumbers[i] << " ";
    }
    std::cout << std::endl;
}
```

templated

std::vector



Recall (from Friday, 11/7): container class concepts

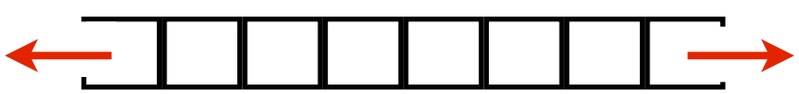
Sequence containers:

vector

"dynamically resizable array"

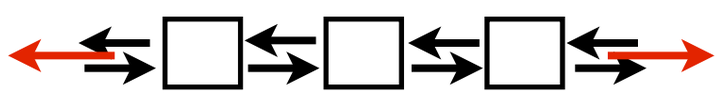
deque

"double-ended queue"



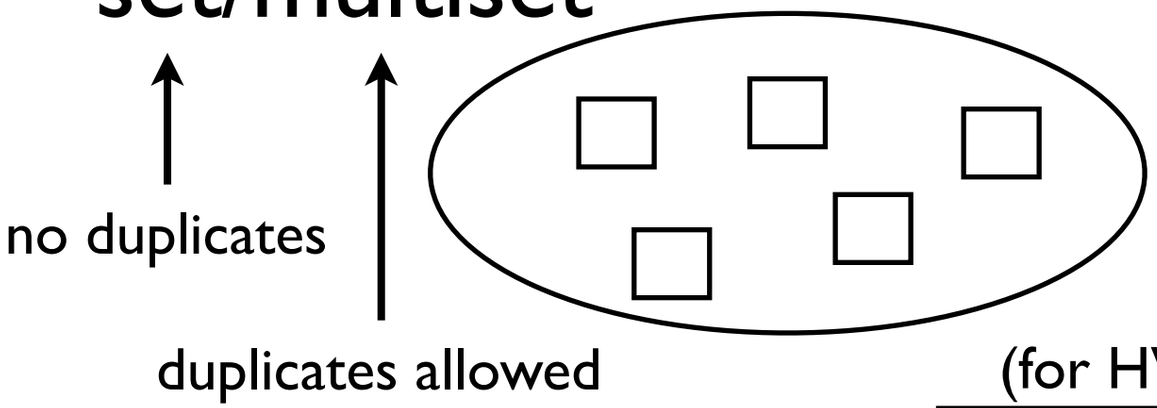
list

"doubly linked list"



Associative containers:

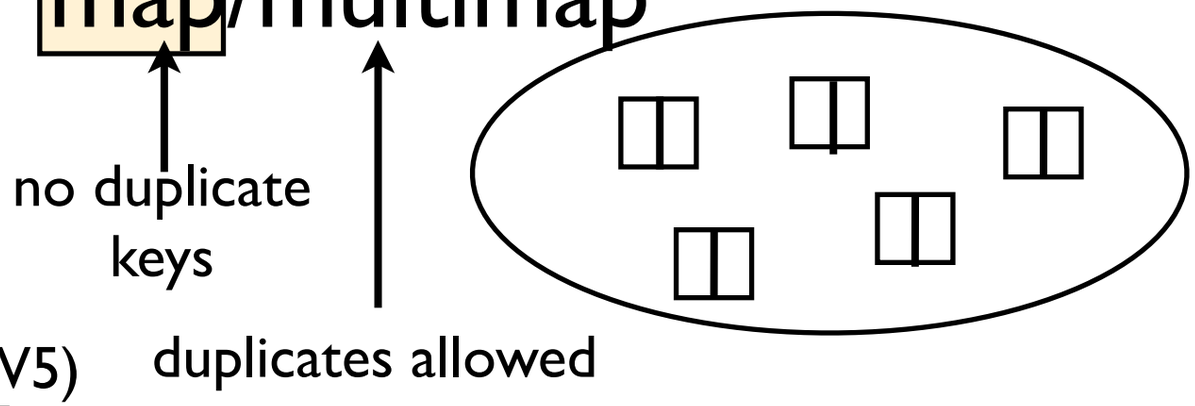
set/multiset



(for HW6)

map/multimap

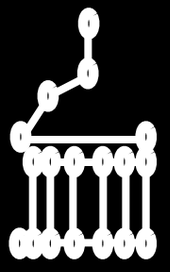
key/value pairs



Container adapters:

(for HW5)

stack, queue, priority_queue, ...



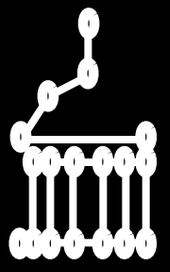
Recall (from Friday, 11/7): container classes are "templated"

```
std::vector<int> myNumbers;
```

type of the elements in your vector container

```
std::vector< std::string > myStrings;
```

We will see how to create our own templated functions and classes later...

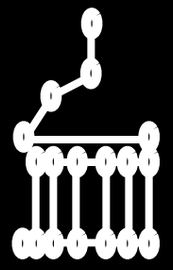


Today's topics (continuing to use existing C++ classes/objects)

- Using std::stack
- Introduction to homework 5

Reading: copied selections from The C++ Standard Library: A Tutorial and Reference (see ICON)
Mini-assignment (due W, 4/15 by 12:30 pm): ICON quiz over use of standard C++ libraries

Using std::stack



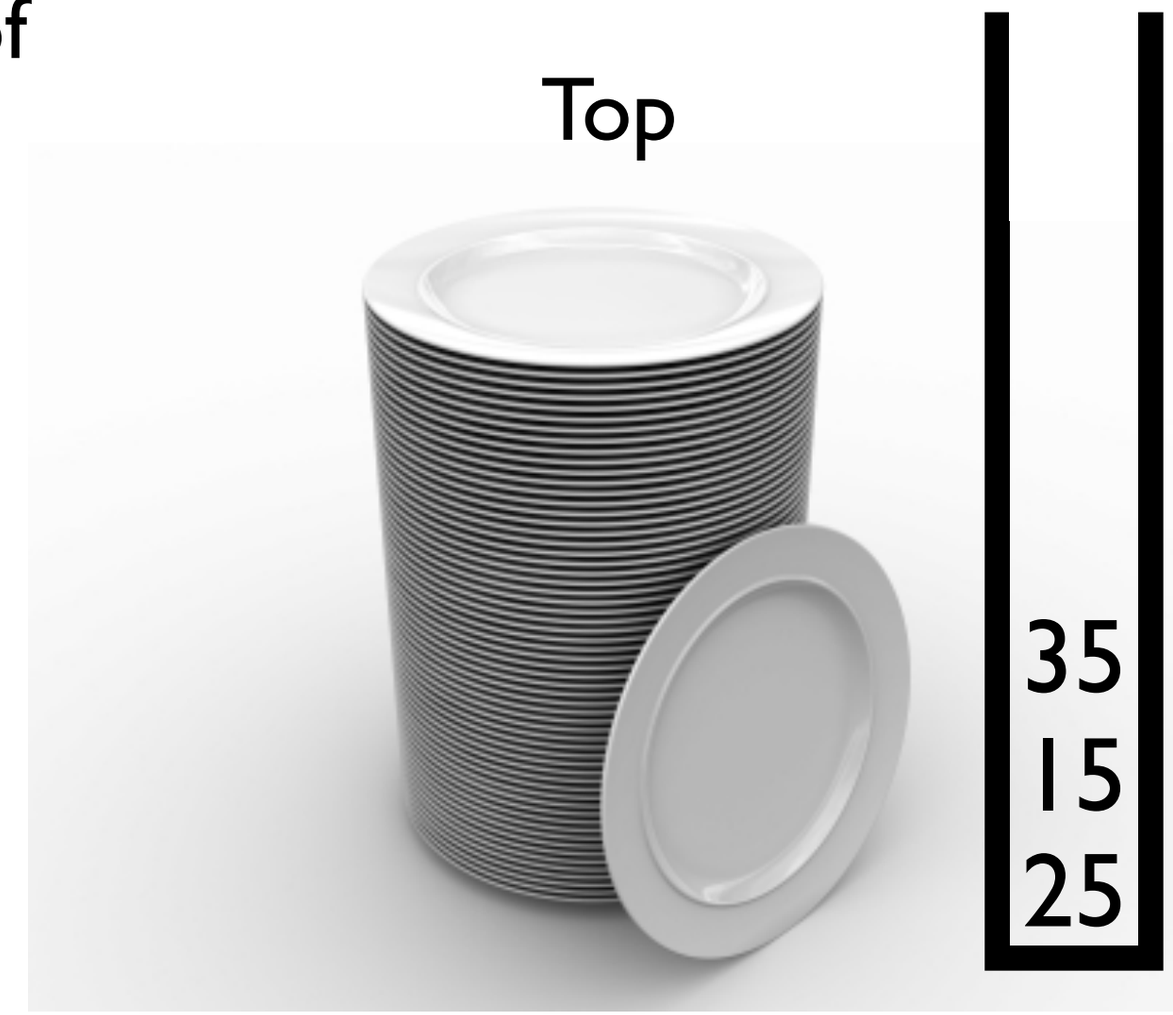
Recall basic stack concepts

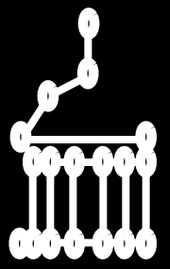
Basic operations:

push: add element to top of stack

pop: delete element from top of stack and return value

Note: using std::stack, we will need two operations to pop an element





Using std::stack

Basic operations:

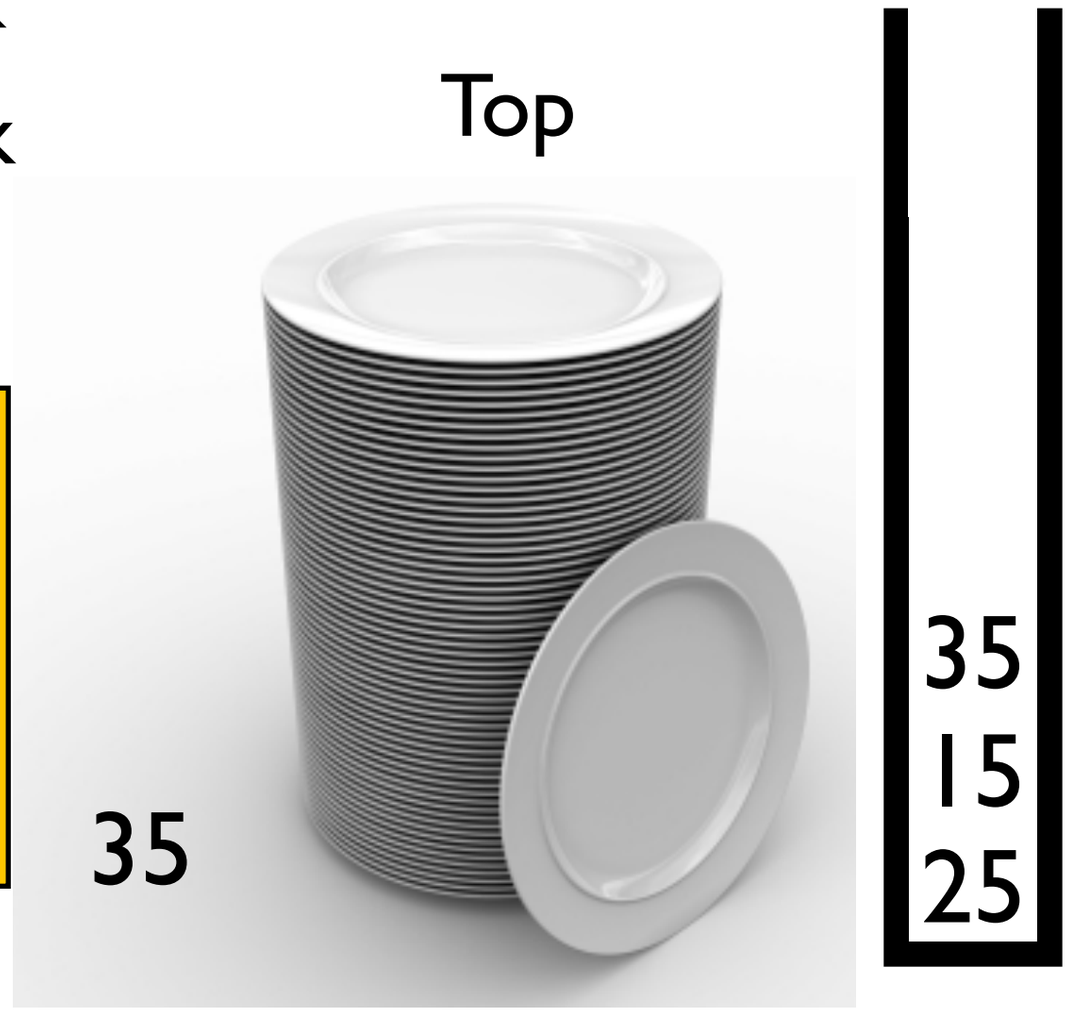
push: add element to top of stack

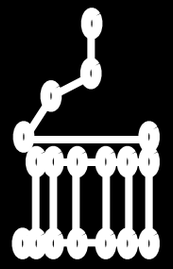
top: return top element from stack

pop: delete top element from stack

```
#include <stack>
```

```
std::stack< int > myStack;
myStack.push(25);
myStack.push(15);
myStack.push(35);
std::cout << myStack.top() << std::endl;
myStack.pop();
```





Using std::stack

Basic operations:

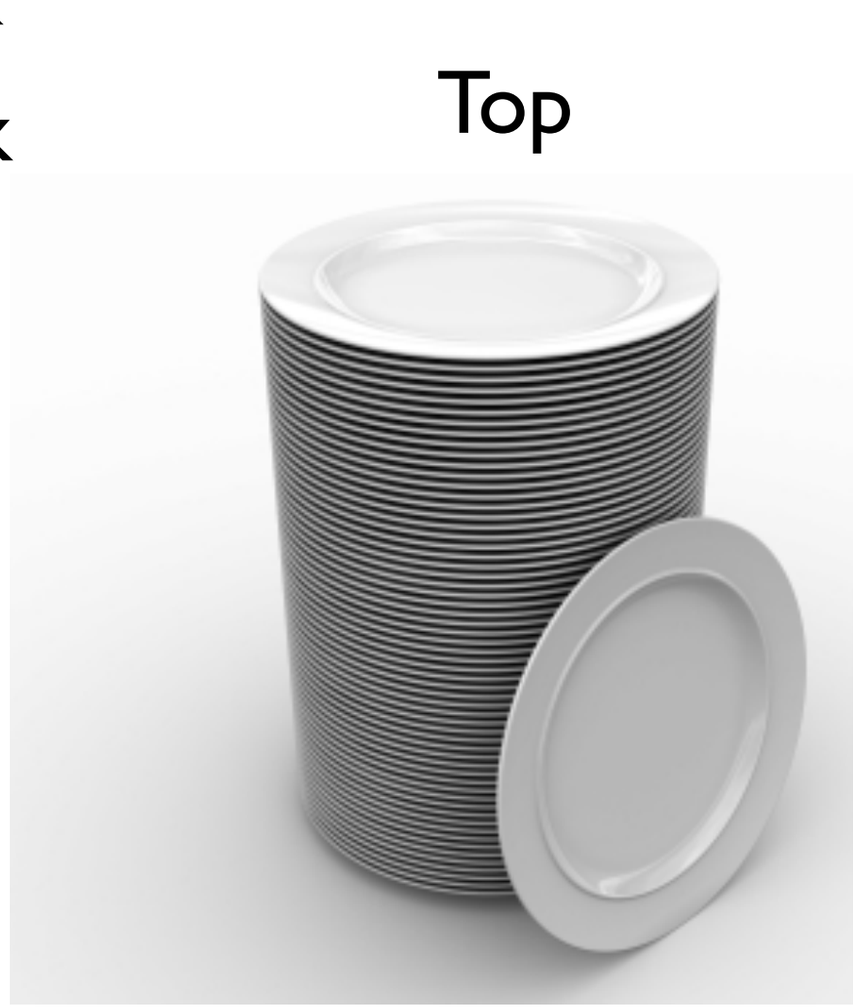
push: add element to top of stack

top: return top element from stack

pop: delete top element from stack

```
#include <stack>
```

```
std::stack< int > myStack;
myStack.push(25);
myStack.push(15);
myStack.push(35);
std::cout << myStack.top() << std::endl;
myStack.pop();
```



Using std::stack

Another operation:

empty: returns true if stack empty;
false otherwise

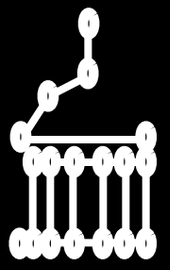
Note: return type is `bool`

New C++ Keywords

```
bool boolVar1 = true;
bool boolVar2 = false;

if (boolVar1)
{
    std::cout << "boolVar1 is true" << std::endl;
}
if (! boolVar2)
{
    std::cout << "boolVar2 is false" << std::endl;
}
```

basic bool example

Top



Using std::stack

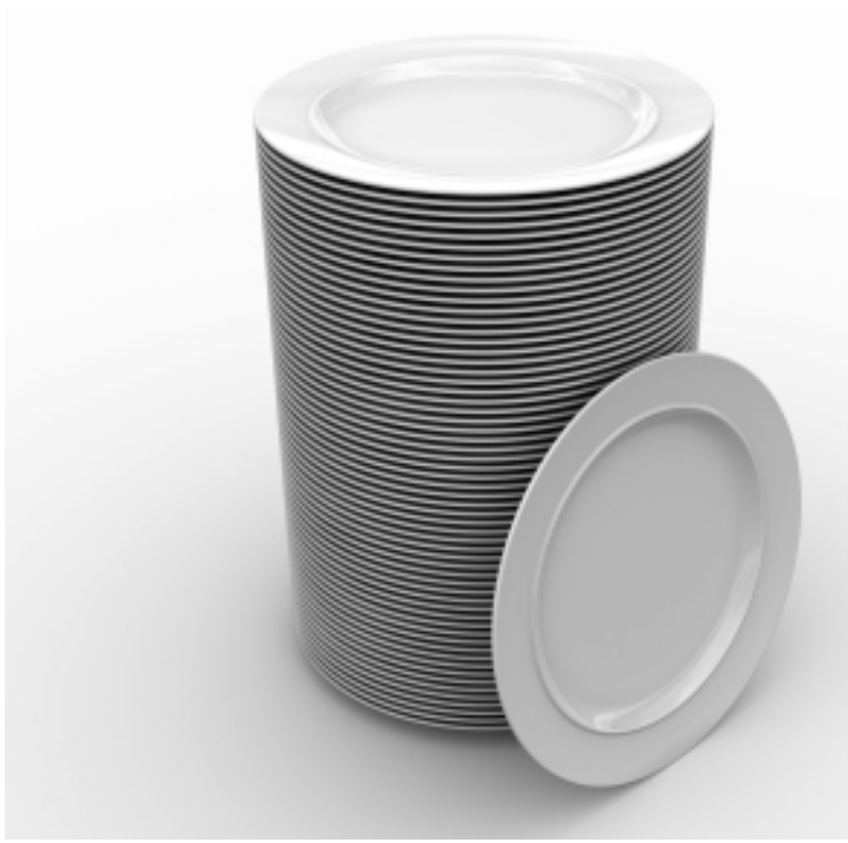
Another operation:

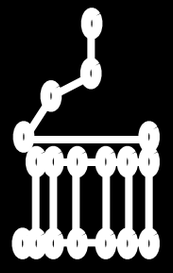
empty: returns true if stack empty;
false otherwise

Note: return type is `bool`

stack bool example

```
while(! myStack.empty() )
{
    int val = myStack.top(); // obtain top value
    myStack.pop(); // remove top value from stack
}
```





CQ: What is the output of the following function?

```
void stackExample1()
{
    std::stack< int > myStack;

    myStack.push(4);
    int count = 1; // count the total number of pushes onto stack
    while(! myStack.empty() )
    {
        int val = myStack.top(); // obtain top value
        myStack.pop(); // remove top value from stack

        if (val > 0)
        {
            myStack.push(val-1);
            count++;
        }
    }

    std::cout << count << std::endl;
}
```

A: 1

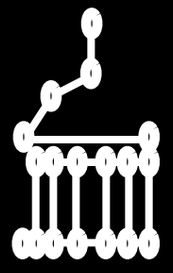
B: 2

C: 3

D: 4

E: 5

F: 6



CQ: What is the output of the following function?

```
void stackExample1()
{
    std::stack< int > myStack;

    myStack.push(4);
    int count = 1; // count the total number of pushes onto stack
    while(! myStack.empty() )
    {
        int val = myStack.top(); // obtain top value
        myStack.pop(); // remove top value from stack

        if (val > 0)
        {
            myStack.push(val-1);
            count++;
        }
    }

    std::cout << count << std::endl;
}
```

A: 1

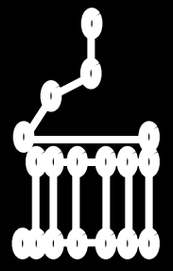
B: 2

C: 3

D: 4

E: 5

F: 6



CQ: What is the output of the following function?

```
void stackExample2()
{
    std::stack< PixelLocation > locStack;
    PixelLocation loc;

    loc.r = 1;
    loc.c = 1;
    locStack.push(loc);
    loc.r = 2;
    loc.c = 2;
    locStack.push(loc);
    loc.r = 3;
    loc.c = 3;
    locStack.push(loc);

    bool done = false;
    while (! done)
    {
        PixelLocation topLoc = locStack.top();
        locStack.pop();
        std::cout << topLoc.r << " " << topLoc.c << std::endl;
        if (topLoc.r <= 2)
        {
            done = true;
        }
    }
}
```

```
struct pixelLocation {
    int r;
    int c;
};

typedef struct pixelLocation PixelLocation;
```

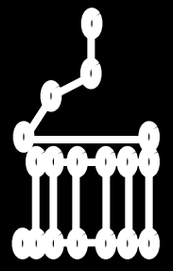
at top

A:
3 3
3 3
(repeated forever)

B:
3 3

C:
3 3
2 2

D:
3 3
2 2
1 1



CQ: What is the output of the following function?

```
void stackExample2()
{
    std::stack< PixelLocation > locStack;
    PixelLocation loc;

    loc.r = 1;
    loc.c = 1;
    locStack.push(loc);
    loc.r = 2;
    loc.c = 2;
    locStack.push(loc);
    loc.r = 3;
    loc.c = 3;
    locStack.push(loc);

    bool done = false;
    while (! done)
    {
        PixelLocation topLoc = locStack.top();
        locStack.pop();
        std::cout << topLoc.r << " " << topLoc.c << std::endl;
        if (topLoc.r <= 2)
        {
            done = true;
        }
    }
}
```

```
struct pixelLocation {
    int r;
    int c;
};

typedef struct pixelLocation PixelLocation;
```

at top

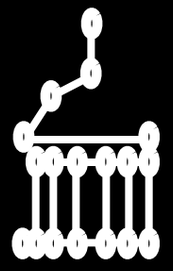
A:
3 3
3 3
(repeated forever)

B: 3 3

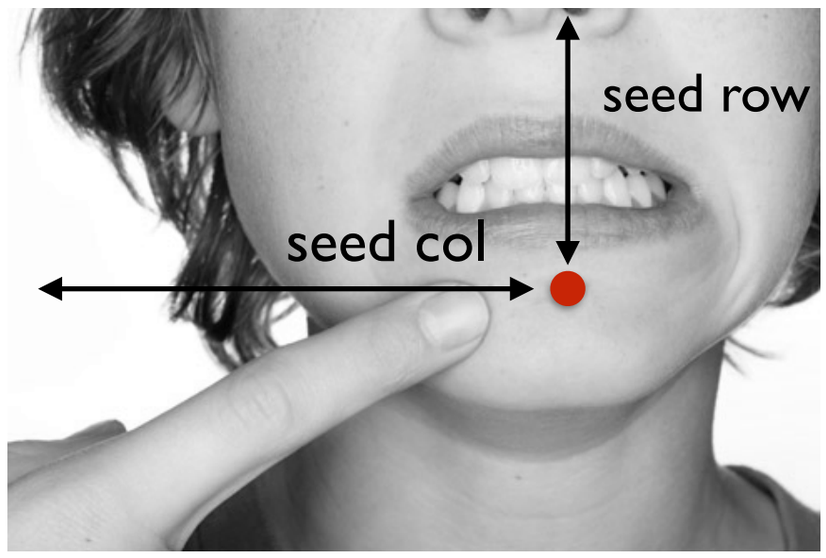
C:
3 3
2 2

D:
3 3
2 2
1 1

HW5: use of stack



Blemish removal



Original image

Blemish removed image

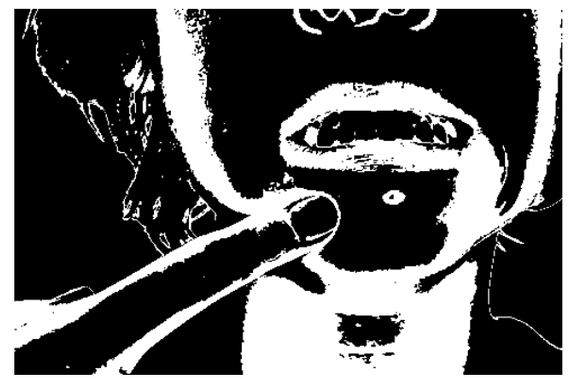
Get a mask for the blemish

User selects a seed pixel

Threshold pixel values around seed pixel value

Find component connected to the seed pixel

Refer to lecture 24



Compiling/running from command line

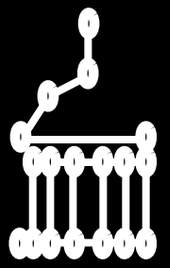
```
> g++ hw5.cpp Image.cpp EasyBMP/EasyBMP.cpp -o hw5.exe

> ./hw5.exe blemish_teeth.bmp output.bmp blemish
```

name for input image

name for output image

type of operation

```
Reading input image: blemish_teeth.bmp
Image Size 327 488
Enter threshold
40
Thresholding image ..
Connected component analysis ..
Saving mask to threshold.bmp
Smoothing .. Please wait !!
Writing final image: output.bmp
```

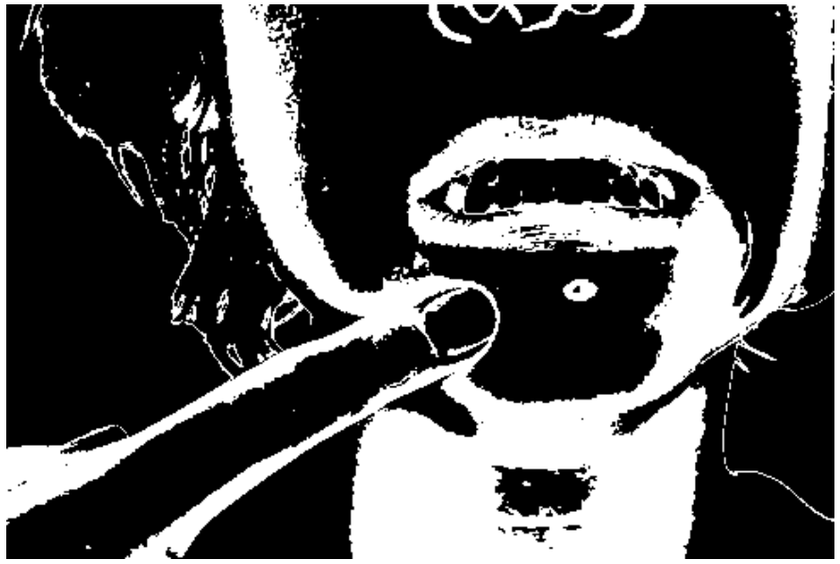


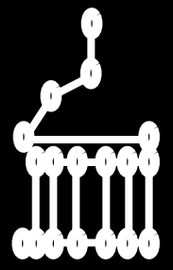
High-level pseudocode (see homework5.pdf for more details)

1. Obtain input/output file names and threshold value from command line.

2. Assign pixels whose absolute difference from seed pixel is less than threshold

Original image





Example of iterating through all pixels in an image

```
Image myImage;
bool success = myImage.readFromBMPFile(inputFileName);
...

// invert image (example of how to iterate through pixels of
// image and modify the image values)
for (int r = 0; r < myImage.getNumRows(); r++)
{
    for (int c = 0; c < myImage.getNumCols(); c++)
    {
        int oldPixelValue = myImage.getPixel(r,c);
        int newPixelValue = 255 - oldPixelValue;
        myImage.setPixel(r,c,newPixelValue);
    }
}
success = myImage.writeToBMPFile(invertedOutputFileName);
...
```

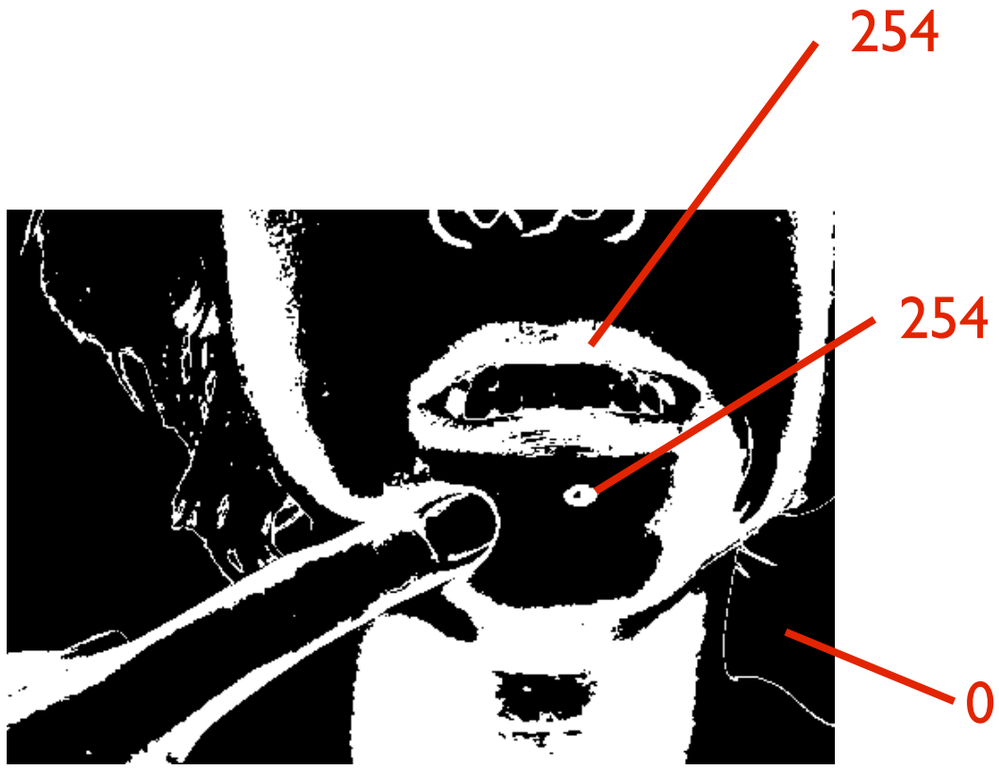
You will write similar nested loops

see lec31ex/main.cpp (includes more code)

High-level pseudocode (see homework5.pdf for more details)

3. Find/modify the "connected component" of background of mask image using the upper-left-hand corner as a seed pixel (row=0; column=0) and specifying the new intensity value of this connected component to be 255.

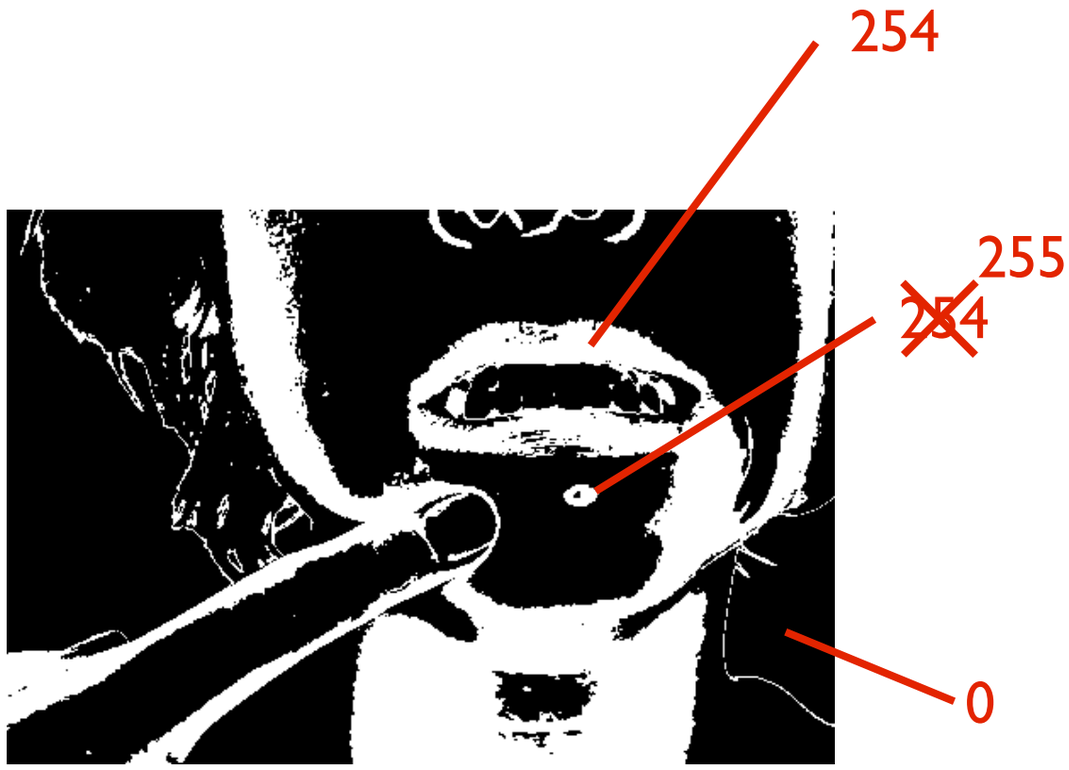
mask

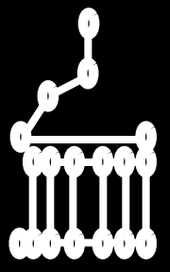


High-level pseudocode (see homework5.pdf for more details)

3. Find/modify the "connected component" of background of mask image using the upper-left-hand corner as a seed pixel (row=0; column=0) and specifying the new intensity value of this connected component to be 255.

mask





High-level pseudocode (see homework5.pdf for more details)

(see homework5.pdf and hw5.cpp for additional details)

```
void markConnectedComponent(Image &image, int seedRow, int seedCol, int ccLabel);
```

pass in mask image

pass in 0

pass in 0

pass in 255

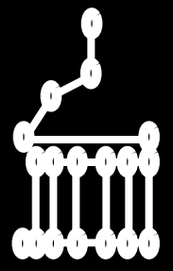
Hints:

use type to store pixel locations (already in hw5.cpp)

```
struct pixelLocation {
    int r;
    int c;
};
typedef struct pixelLocation PixelLocation;
```

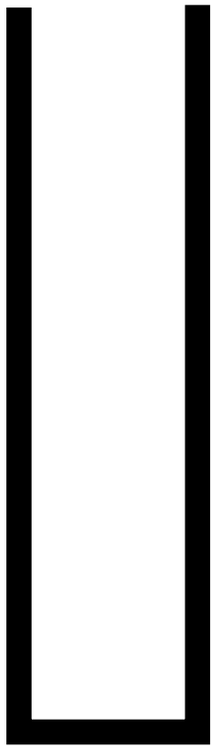
use stack of PixelLocations

```
std::stack< PixelLocation > pixelLocStack;
```

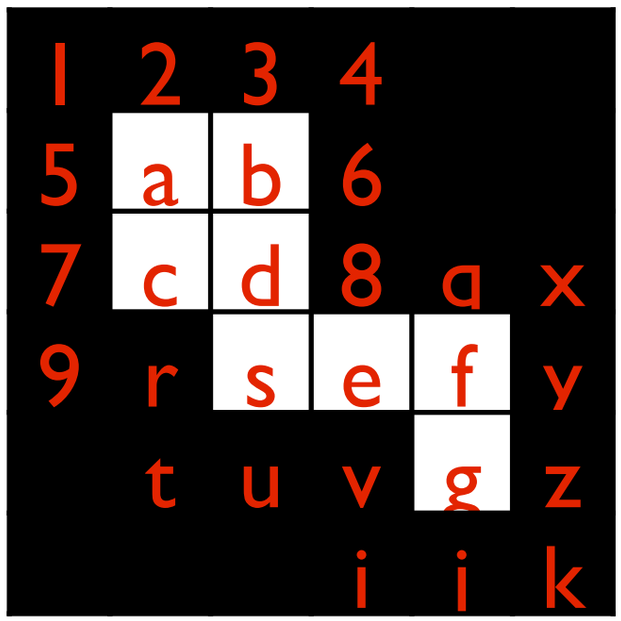


Recall: pseudocode for finding one connected component using depth-first search

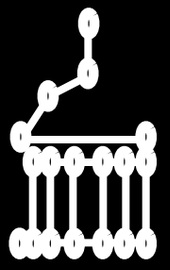
top
of stack



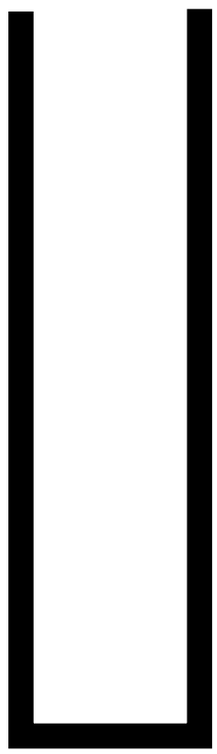
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Recall: pseudocode for finding one connected component using depth-first search



top
of stack



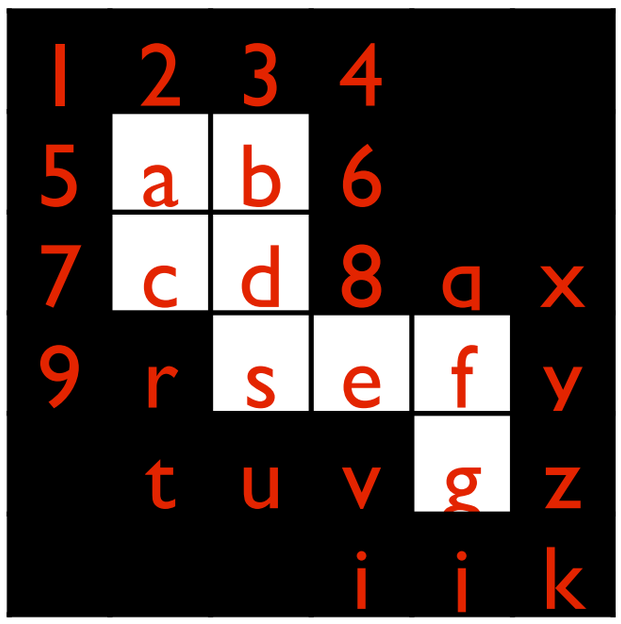
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel
value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1



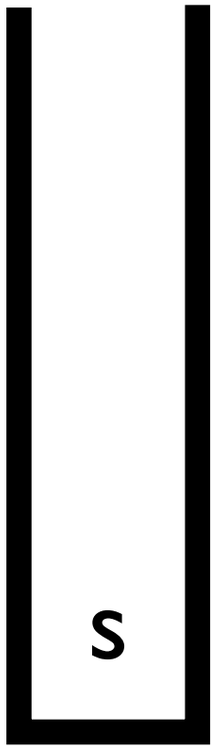

```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
PixelLocation pixelLoc;
pixelLoc.r = seedRow;
pixelLoc.c = seedCol;
pixelLocStack.push(pixelLoc);
```

of stack



push seed node to stack → PixelLocation

while (stack is not empty)

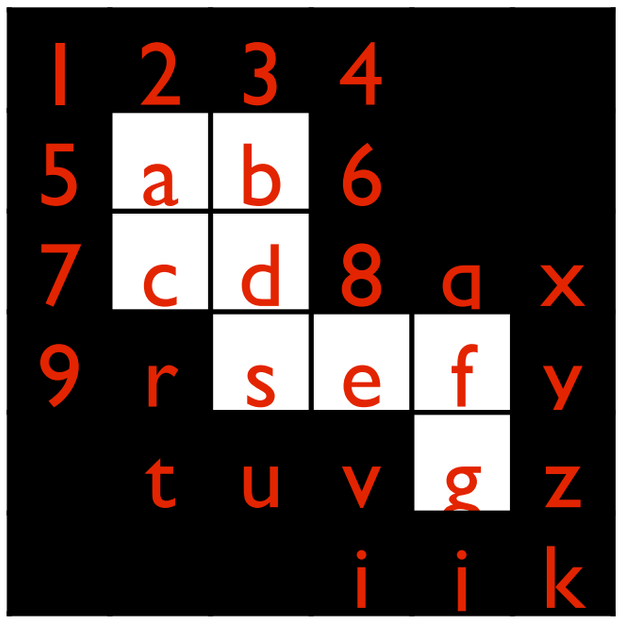
pop node from stack

if color is white: → seed value (254)

change color to yellow → ccLabel value (255)

push all neighbors to stack

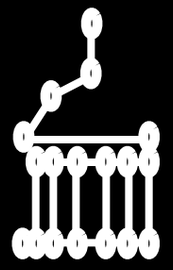
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

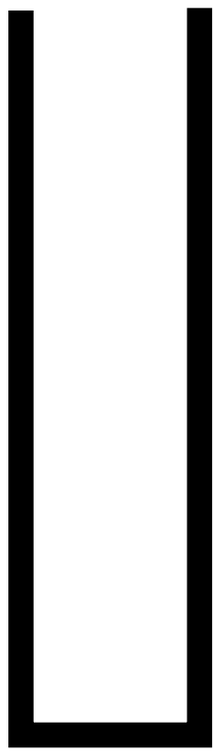
PixelLocation

seed value (254)

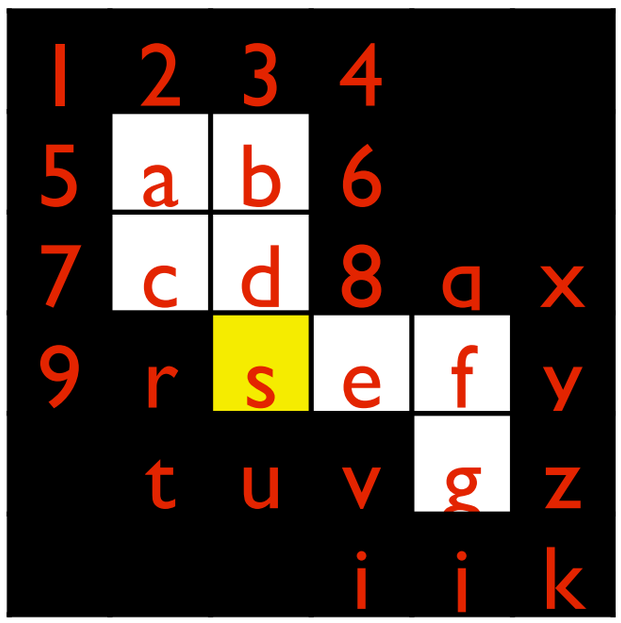
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

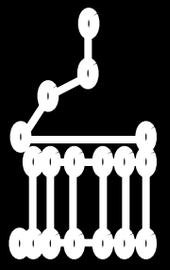


s




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

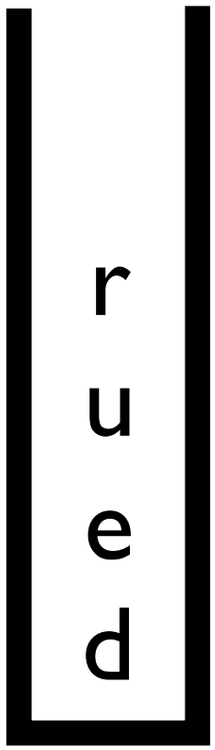
PixelLocation

seed value (254)

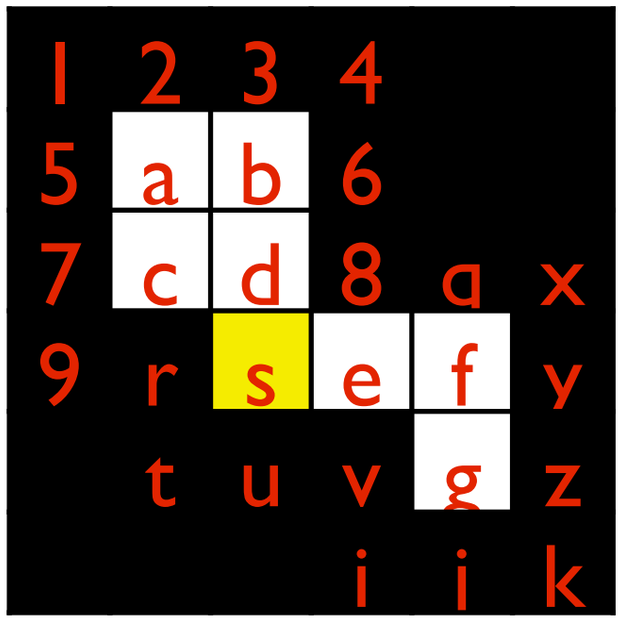
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



s




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

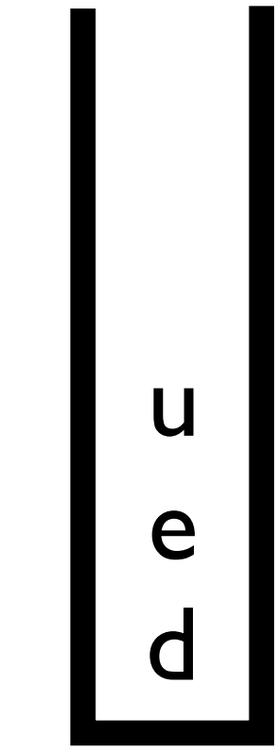
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

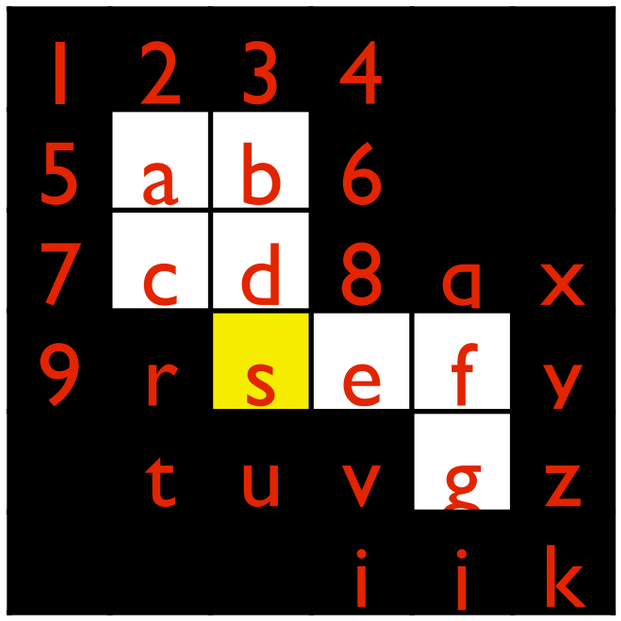
seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1



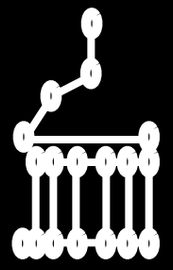
r




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



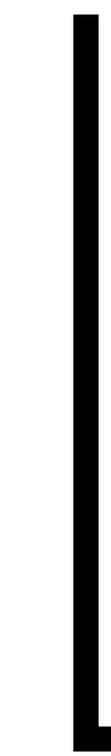
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

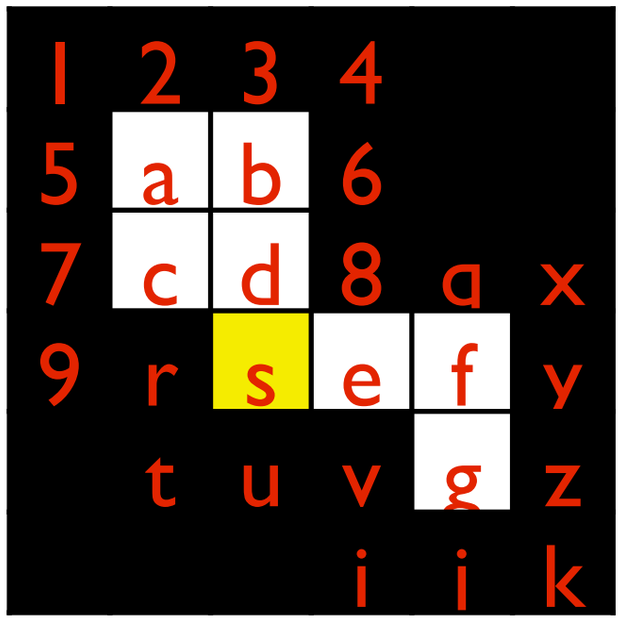
seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1



u




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

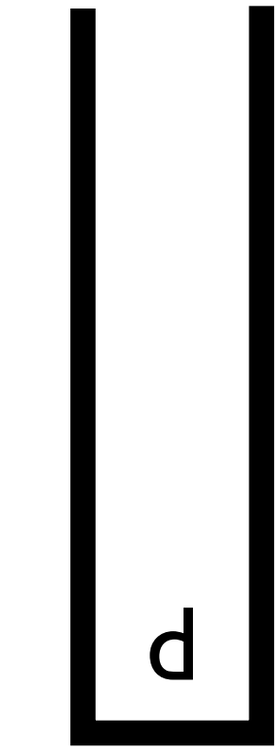
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

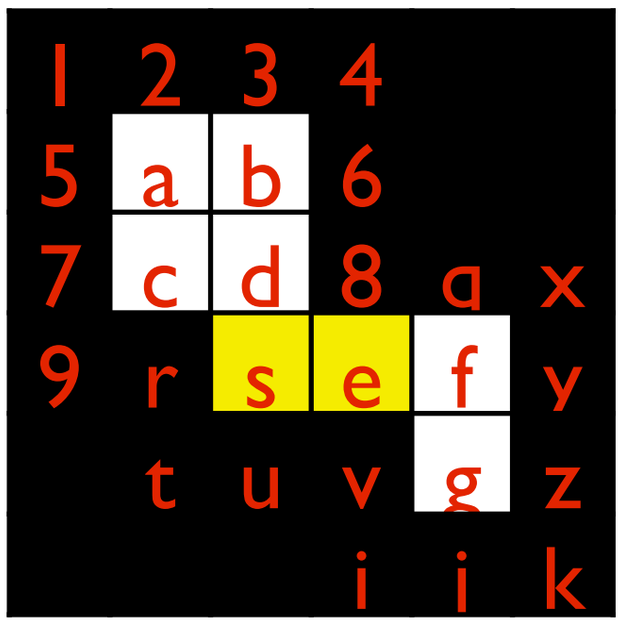
PixelLocation

seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1






```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

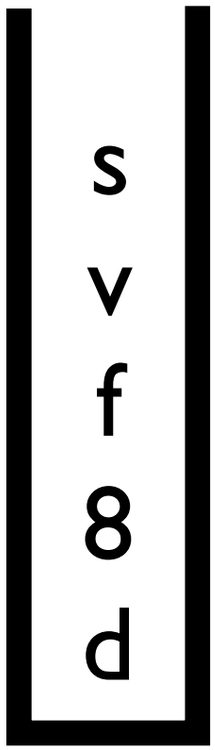
PixelLocation

seed value (254)

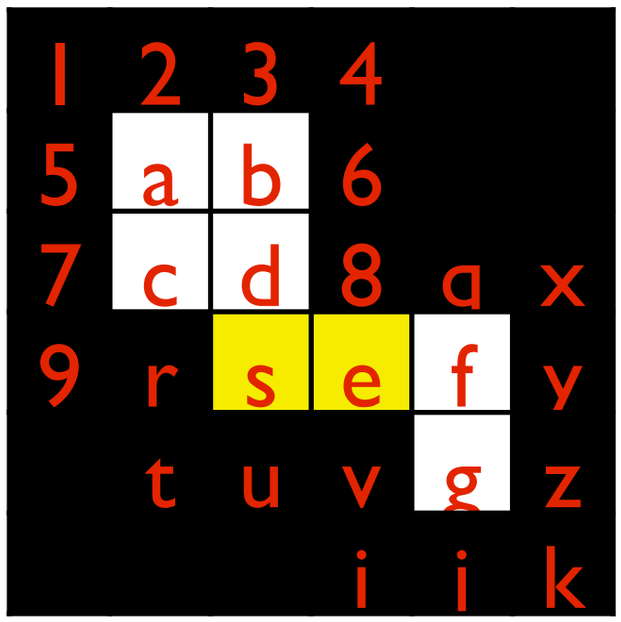
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



e




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

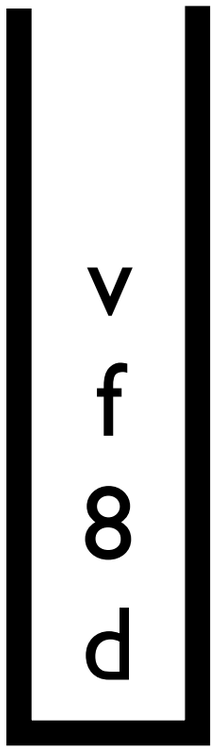
PixelLocation

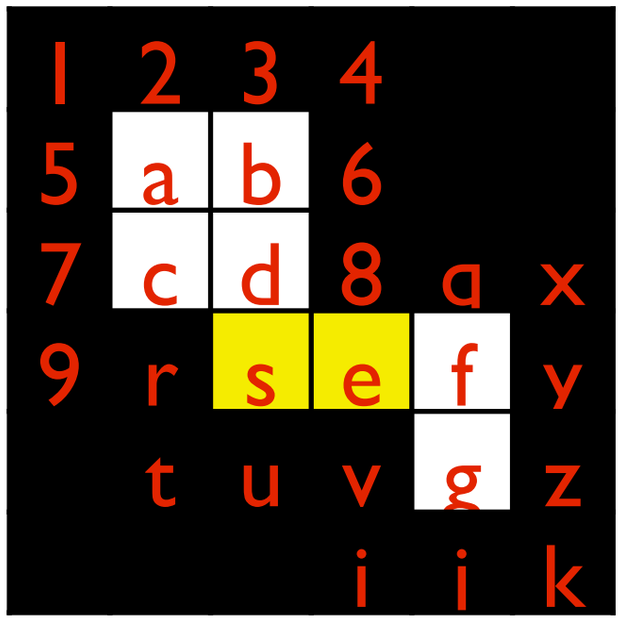
seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack






```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

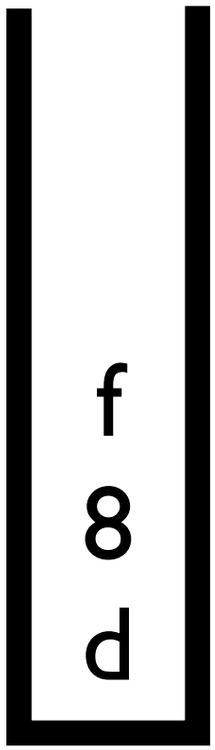
PixelLocation

seed value (254)

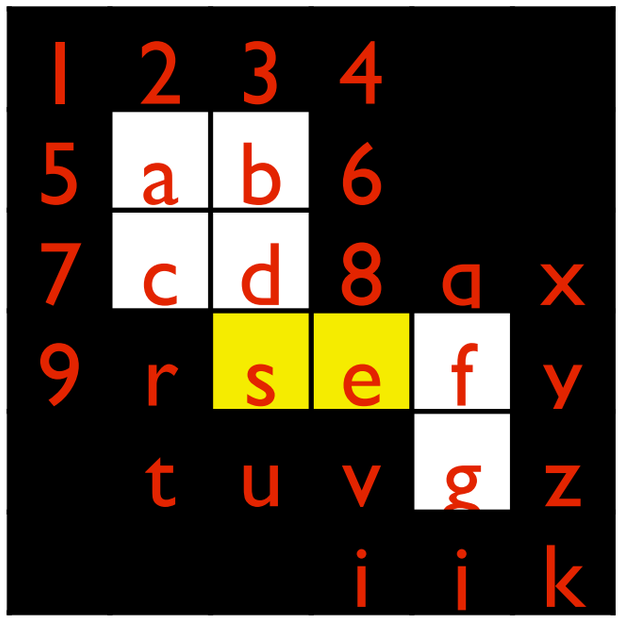
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

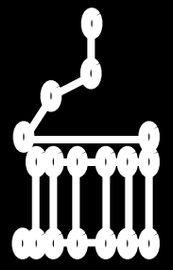


v




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

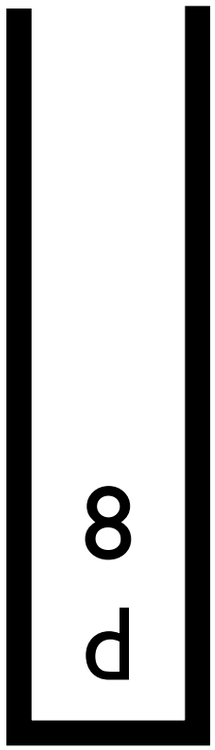
PixelLocation

seed value (254)

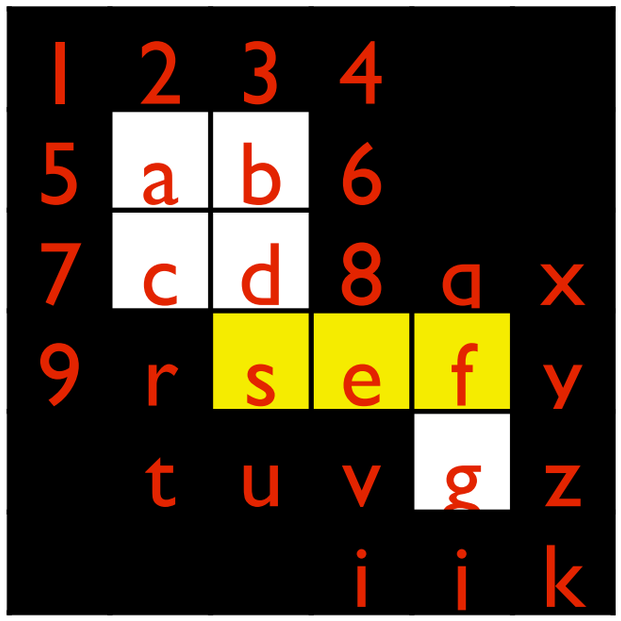
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



f




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

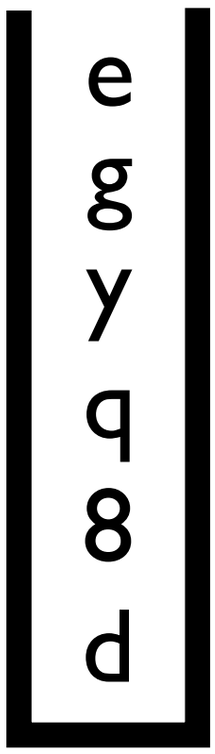
PixelLocation

seed value (254)

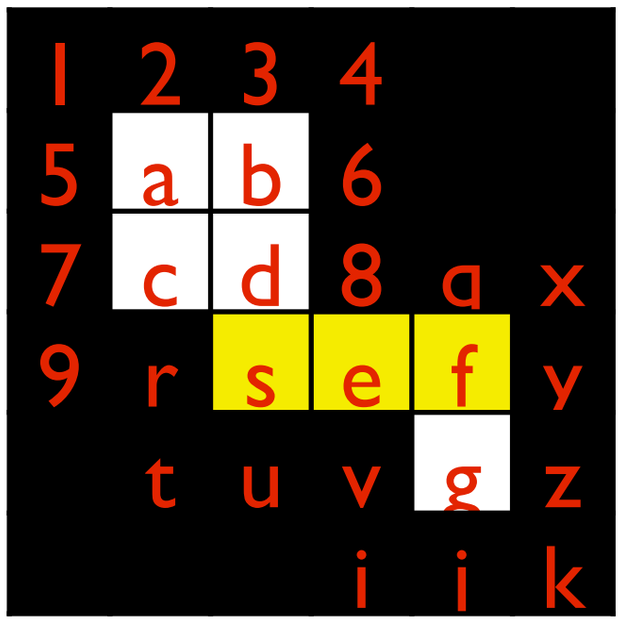
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



f




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

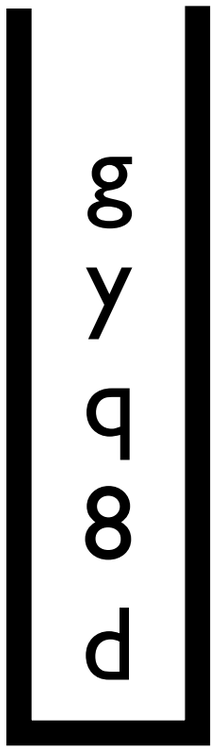
PixelLocation

seed value (254)

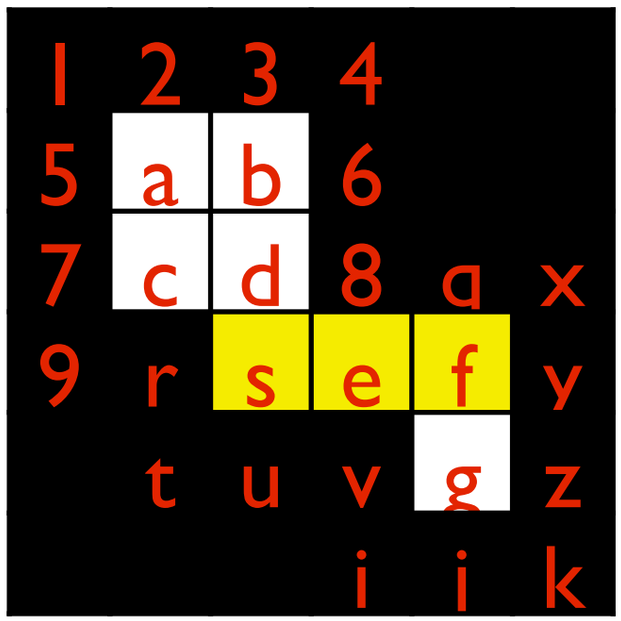
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack



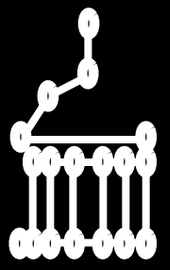
e




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

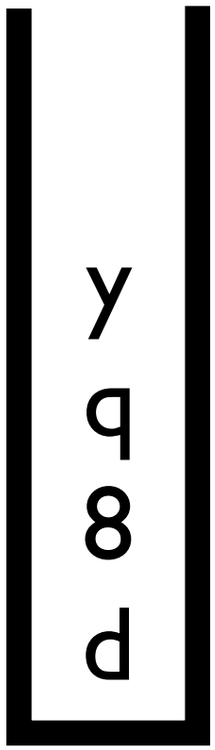
PixelLocation

seed value (254)

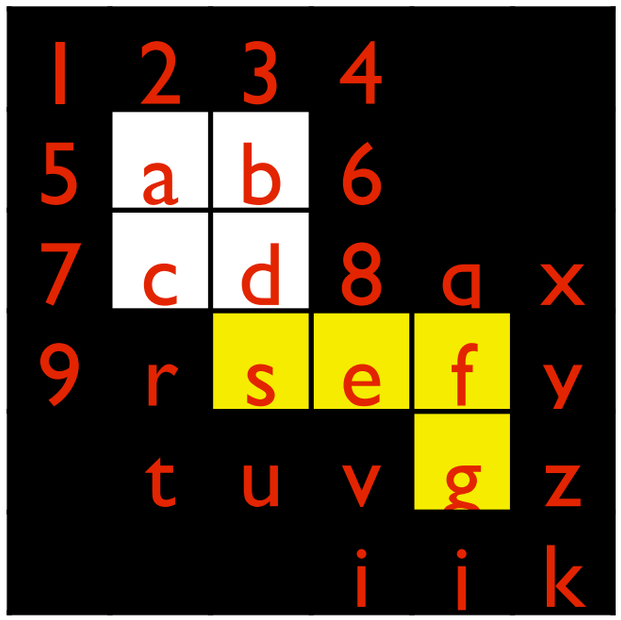
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



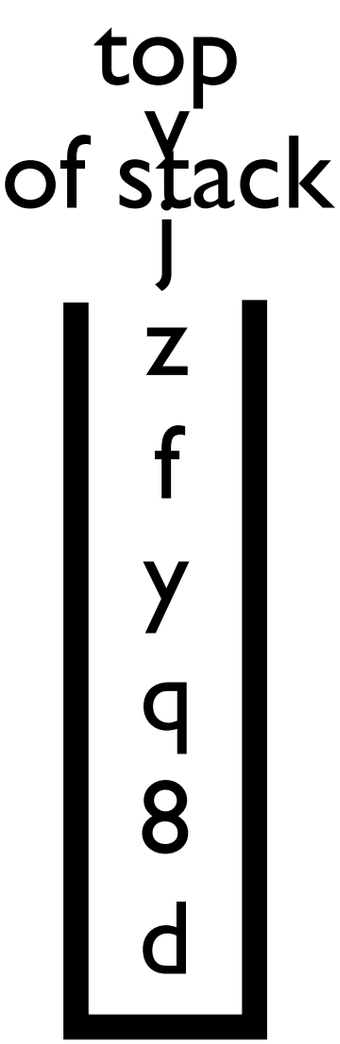
g




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



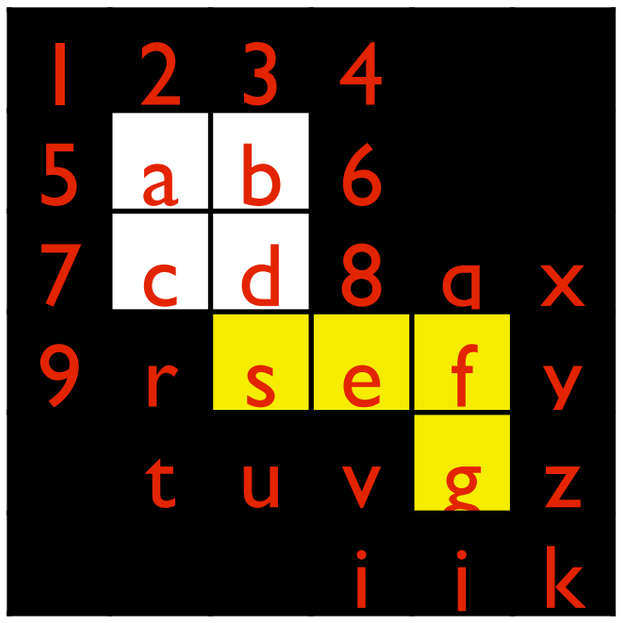

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel value (255)

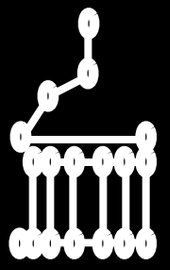
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



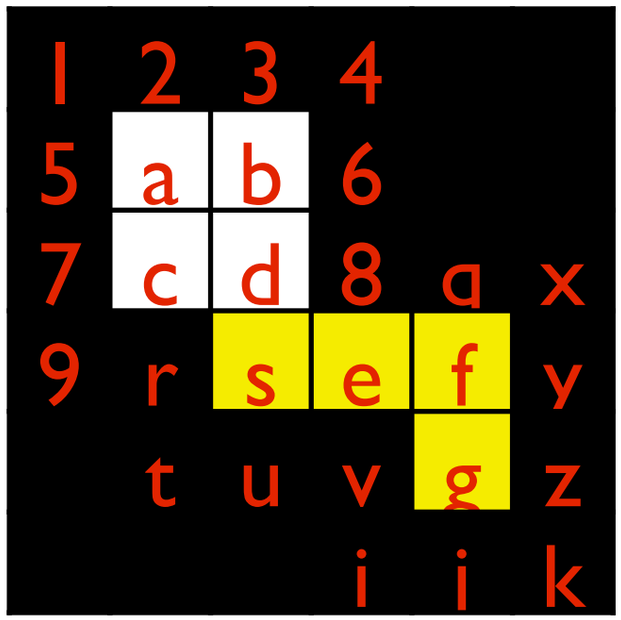
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

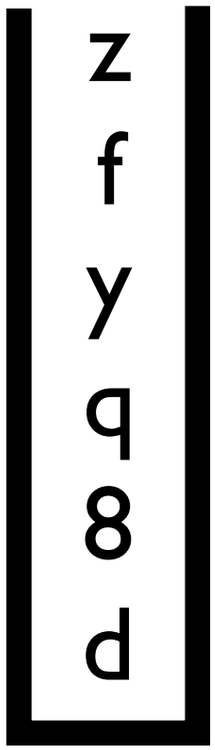
PixelLocation

seed value (254)

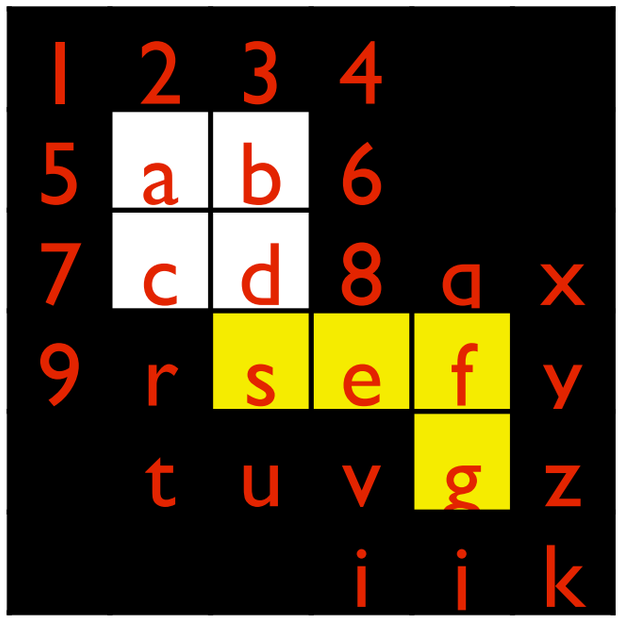
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



j

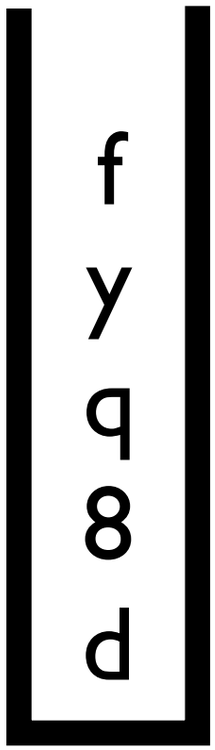



```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

top of stack



z

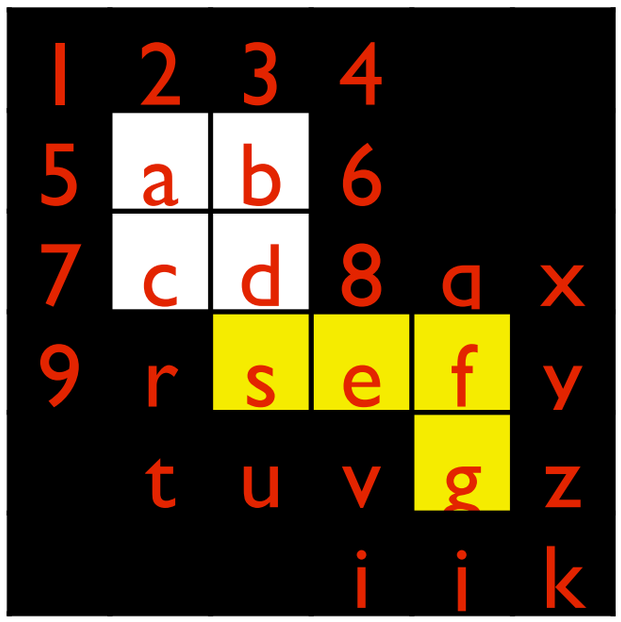
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

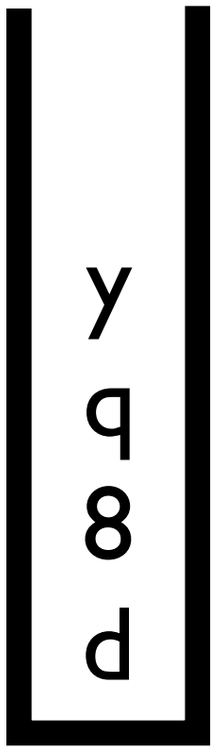
PixelLocation

seed value (254)

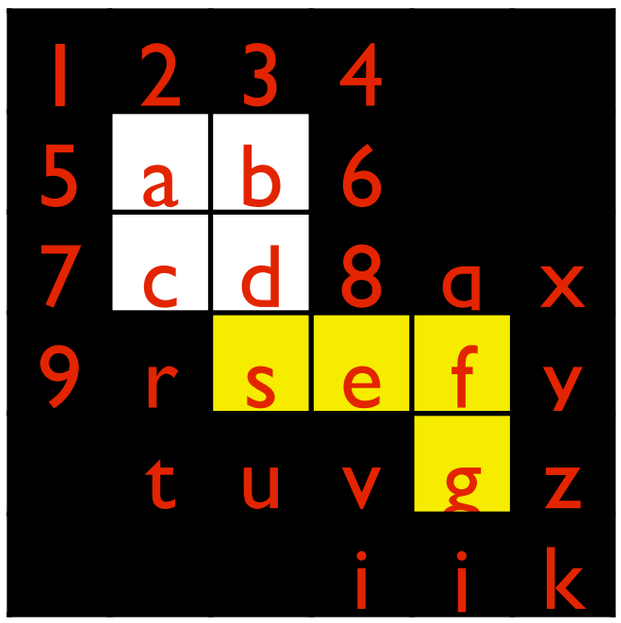
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

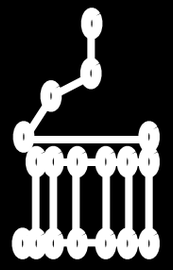


f




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

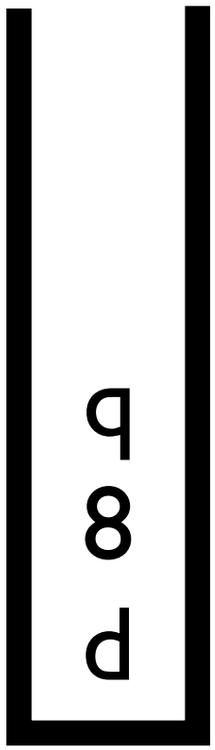
PixelLocation

seed value (254)

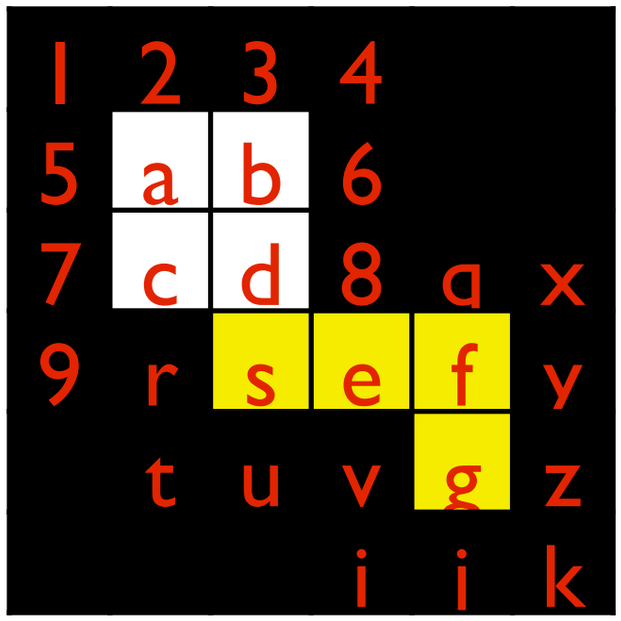
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



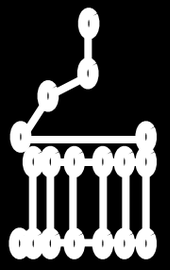
y




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

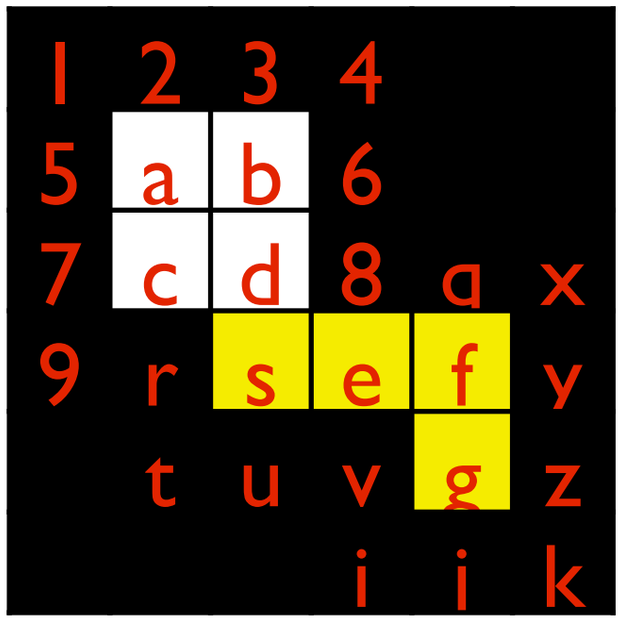
PixelLocation

seed value (254)

ccLabel value (255)

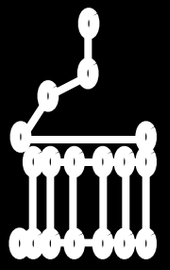
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

q



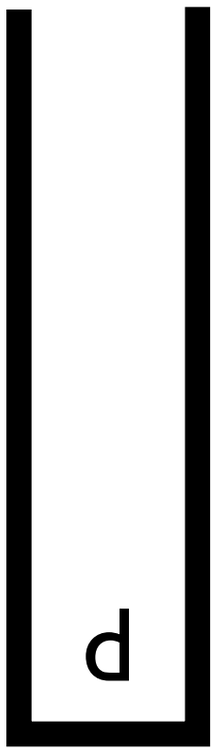

```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

top
of stack



8

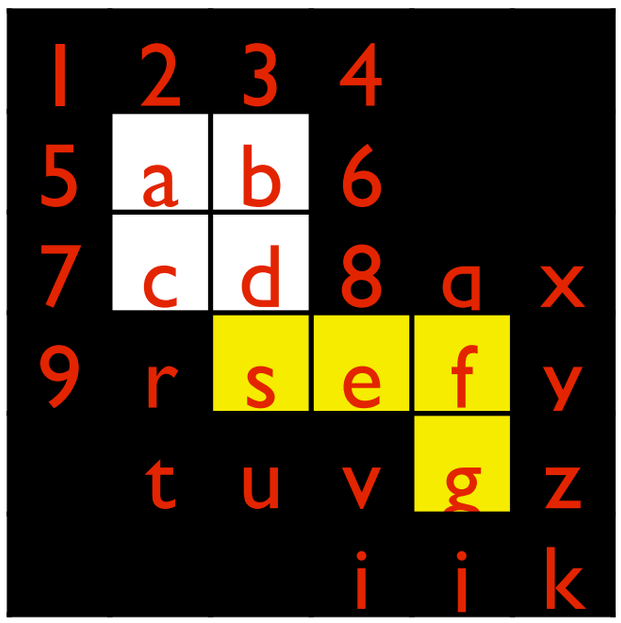
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel
value (255)

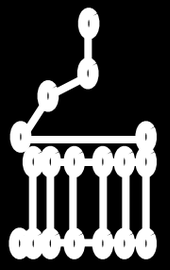
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

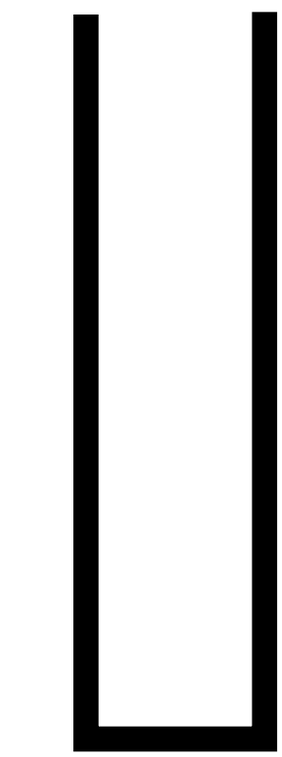



```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search





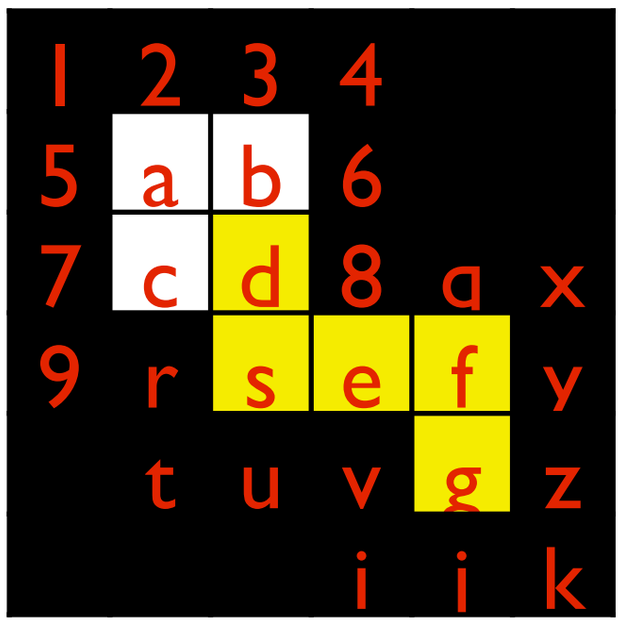

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel value (255)

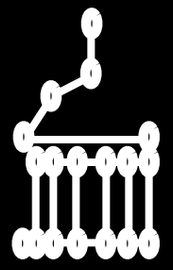
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

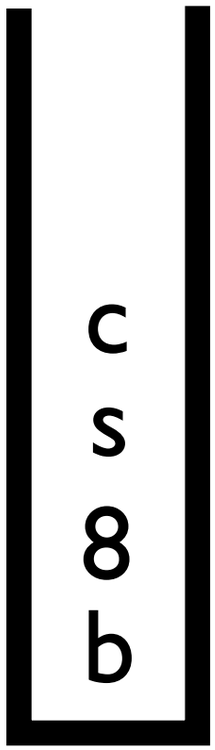
PixelLocation

seed value (254)

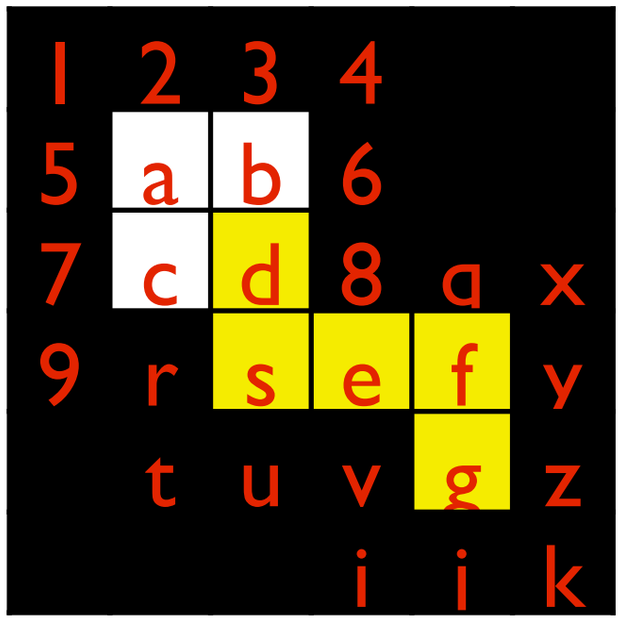
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

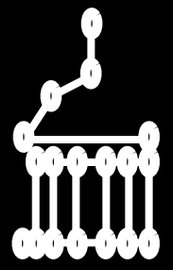


d




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

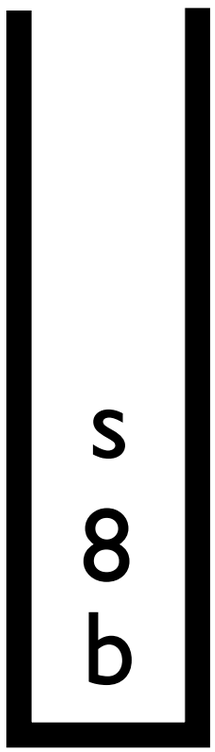
PixelLocation

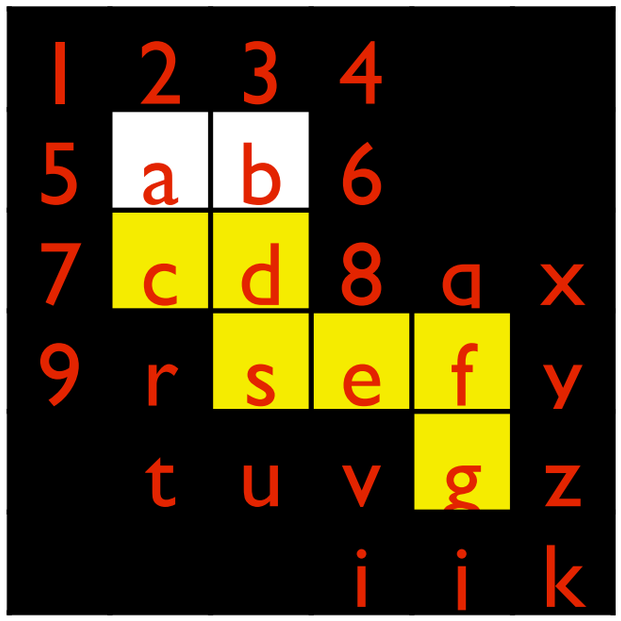
seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



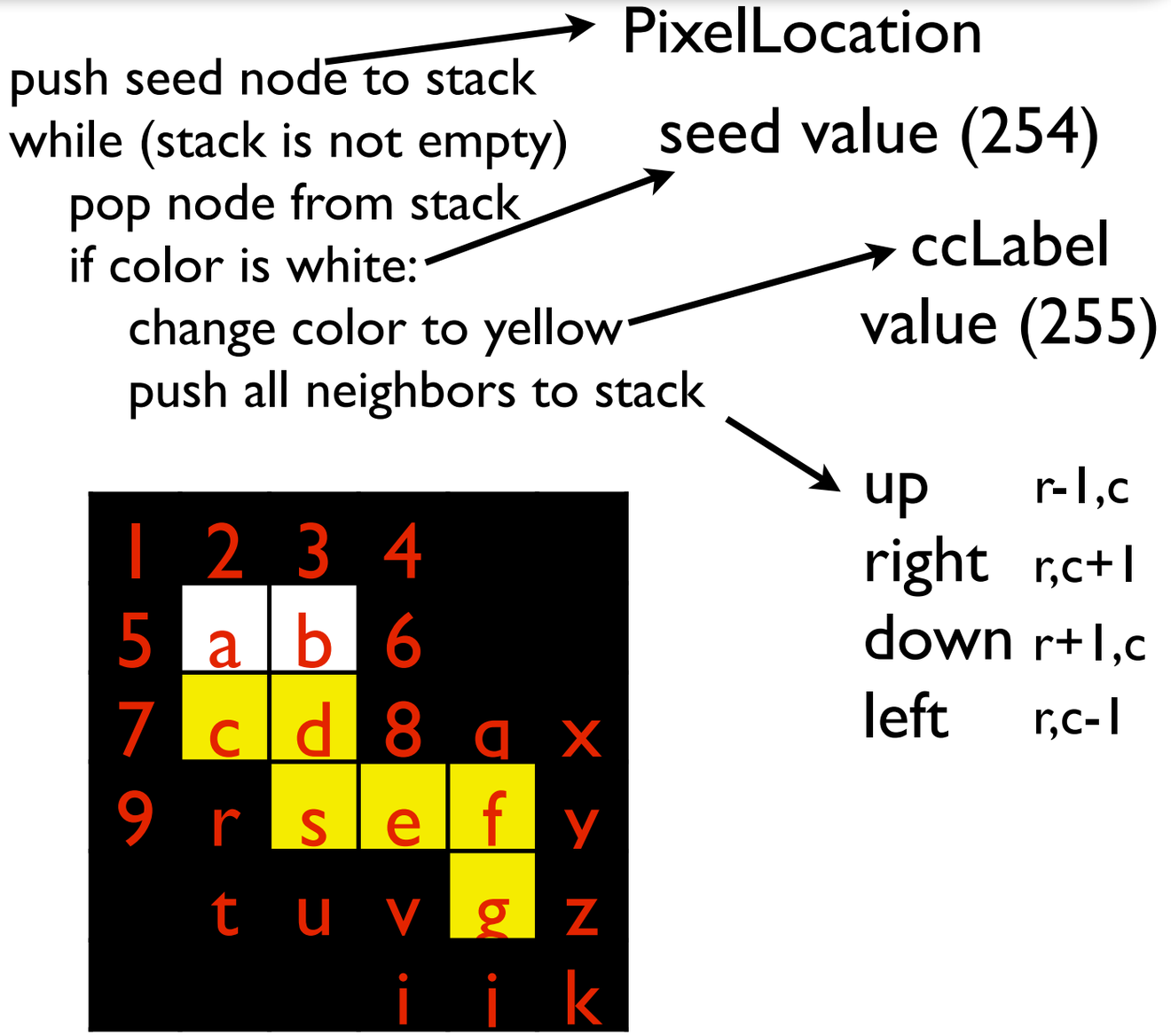


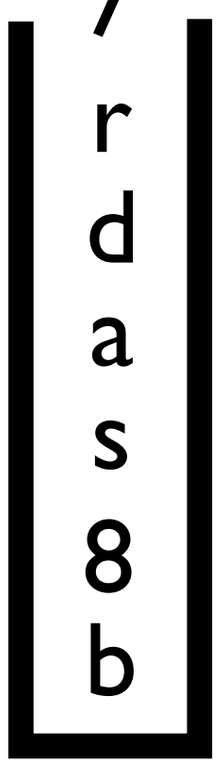
c

```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



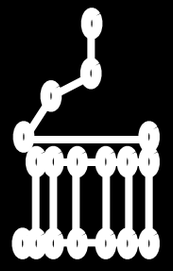


c


```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

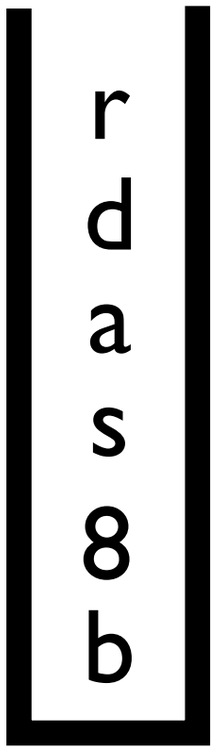
PixelLocation

seed value (254)

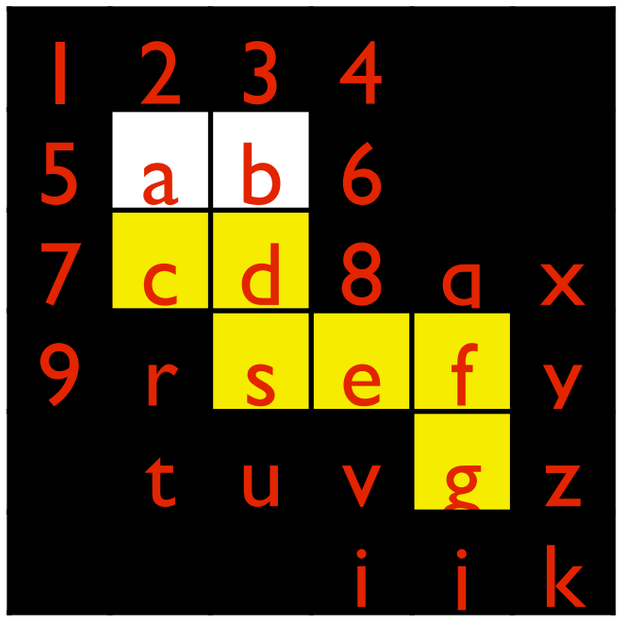
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

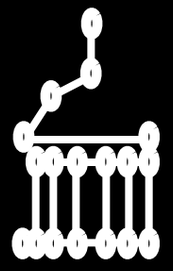


7




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

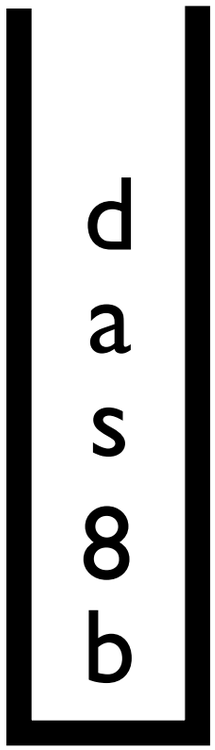
PixelLocation

seed value (254)

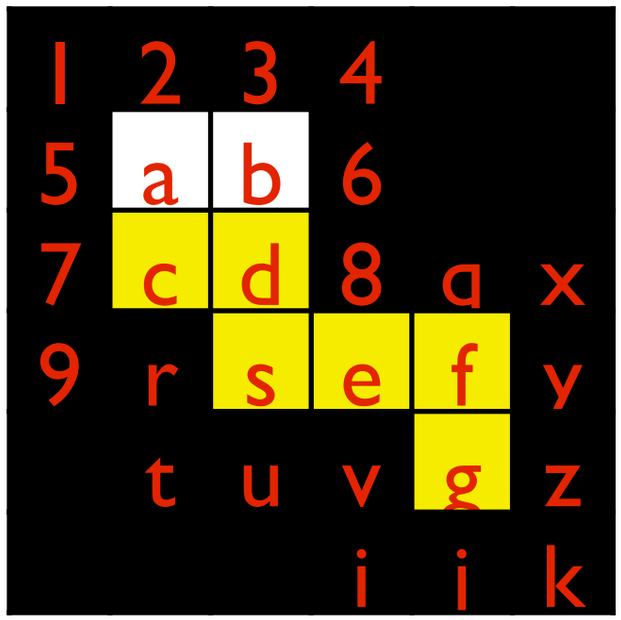
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



r




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

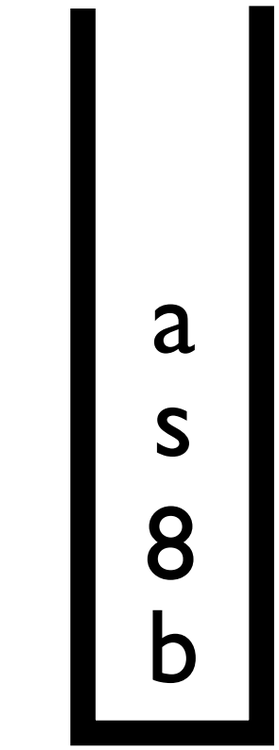
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

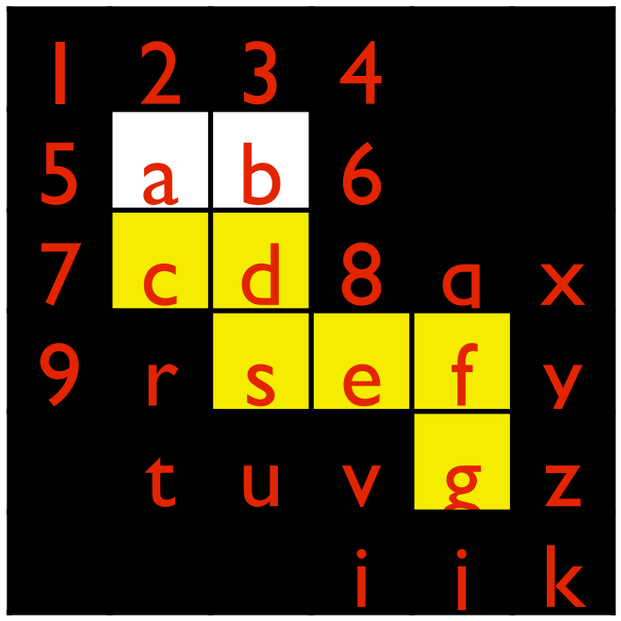
PixelLocation

seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1






```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

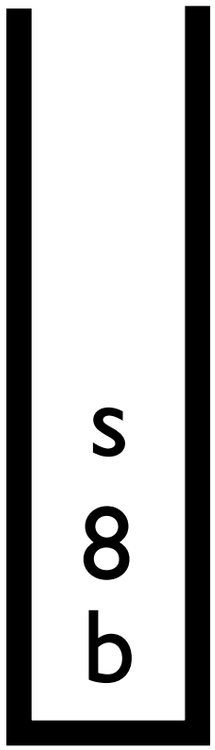
PixelLocation

seed value (254)

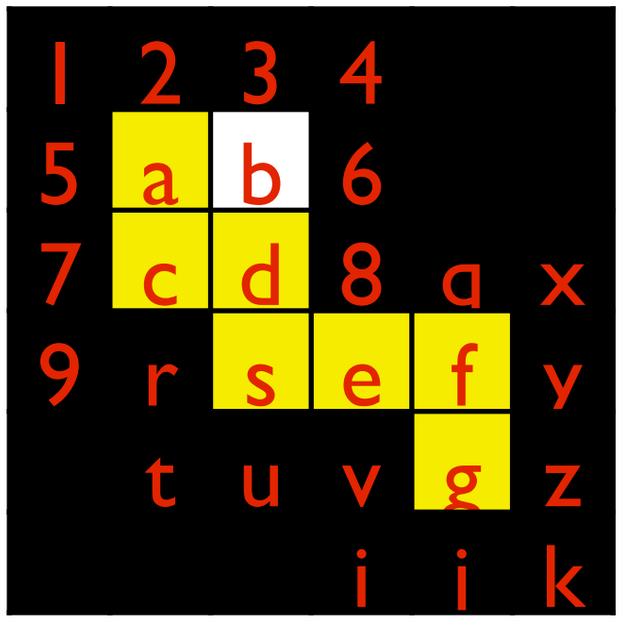
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack



a




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

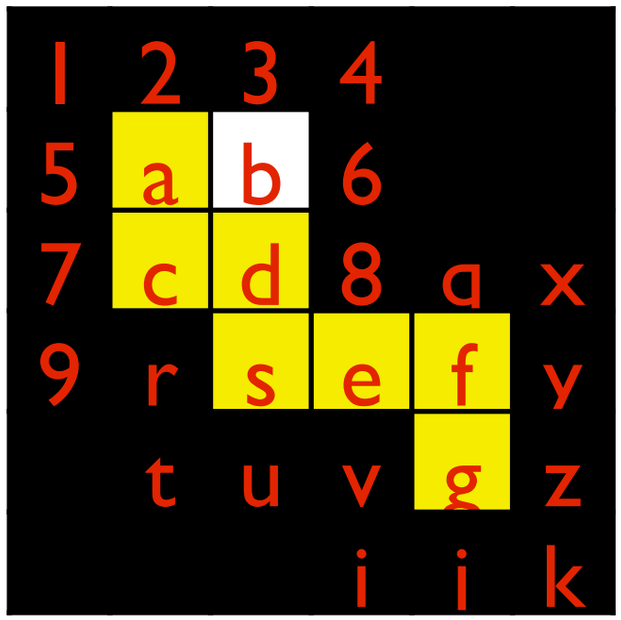
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

5
c
b
2
s
8
b

a




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

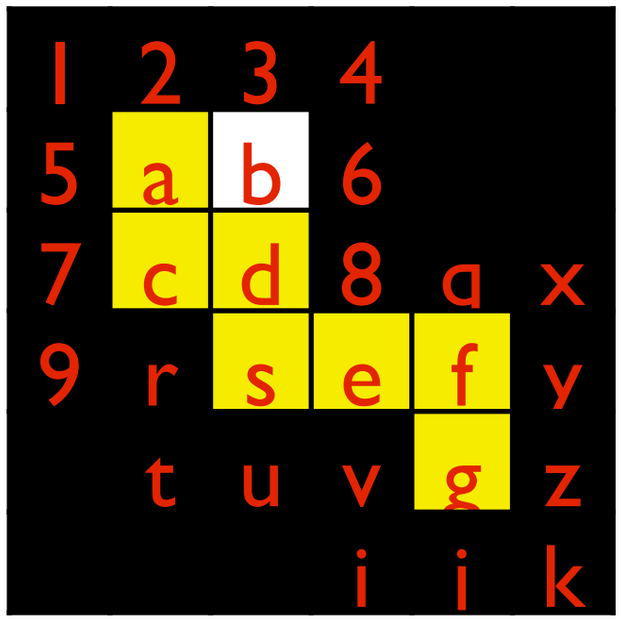
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

c
b
2
s
8
b

5

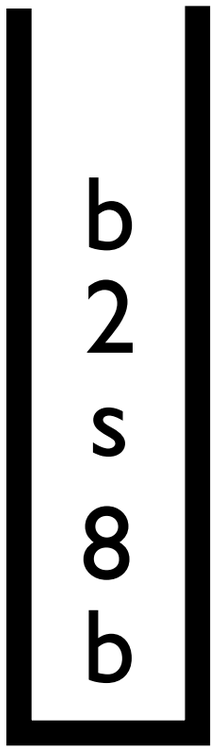



```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

top of stack




```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

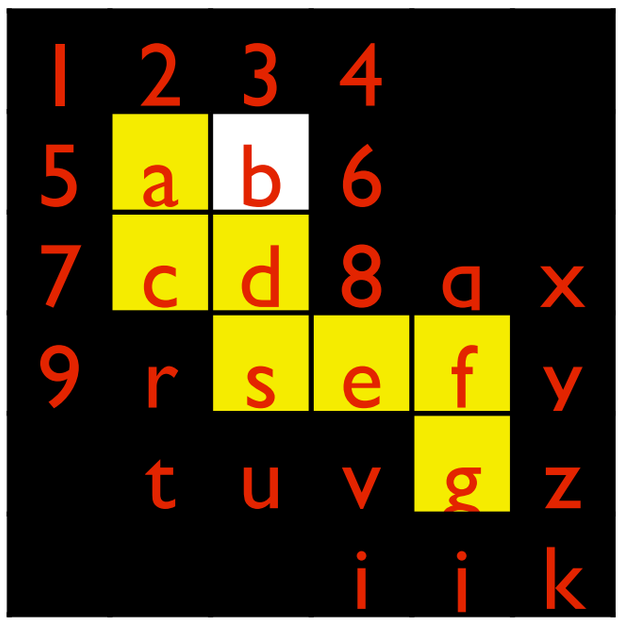
PixelLocation

seed value (254)

ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

c




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

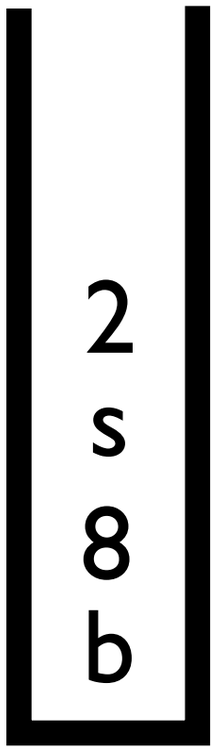
PixelLocation

seed value (254)

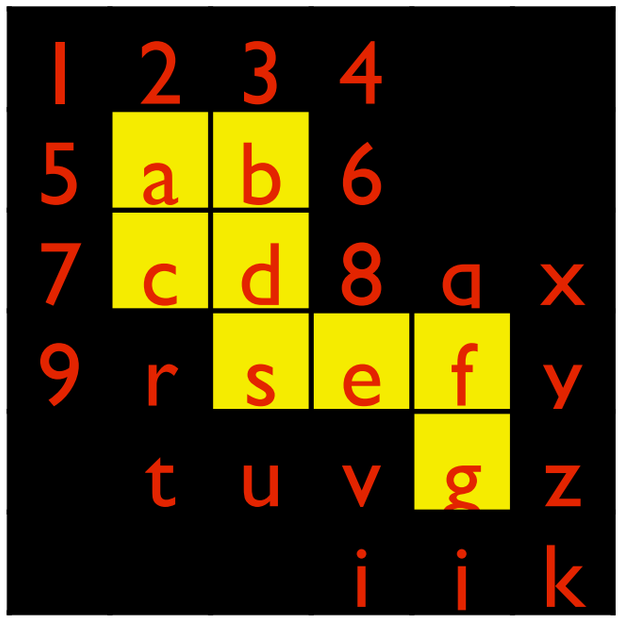
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



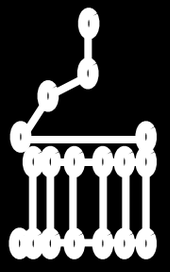
b




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

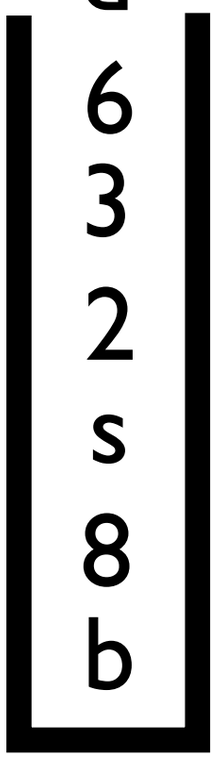
PixelLocation

seed value (254)

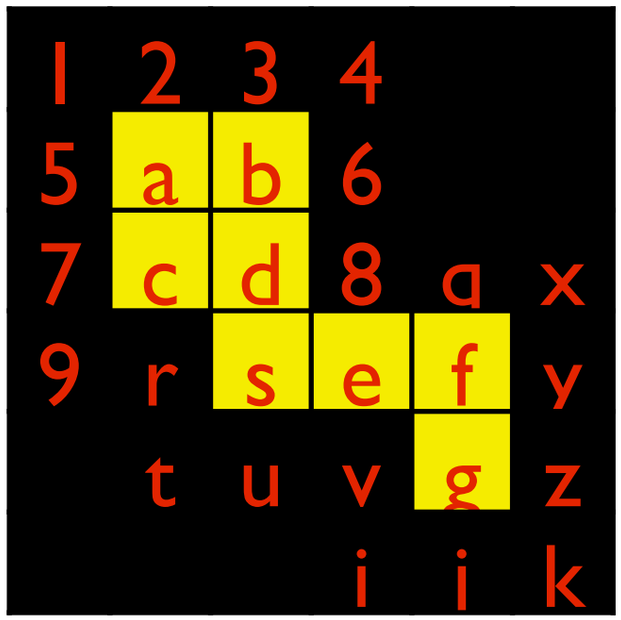
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



b




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

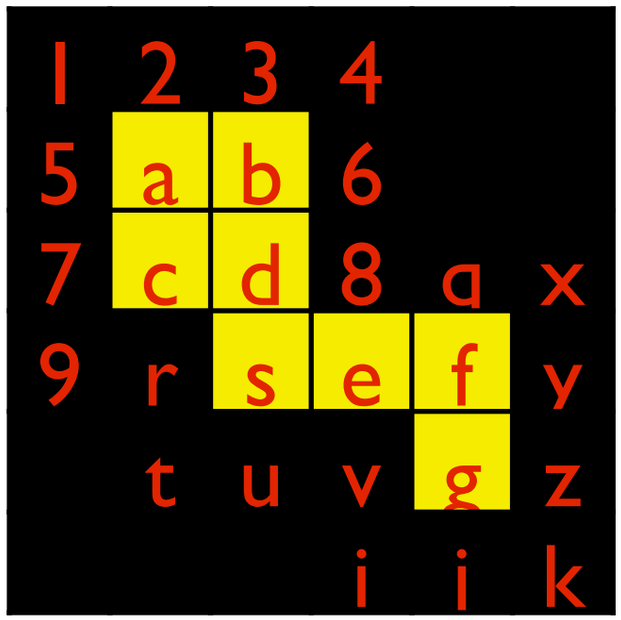
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack

d
6
3
2
s
8
b

a




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

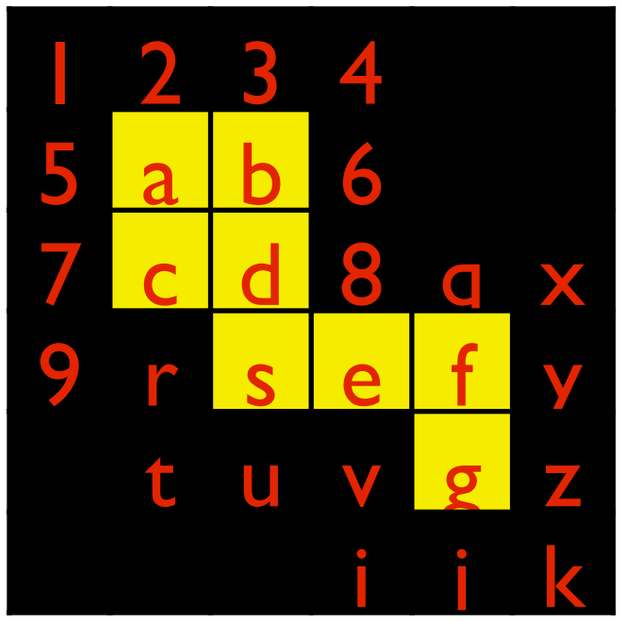
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack

6
3
2
s
8
b

d




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

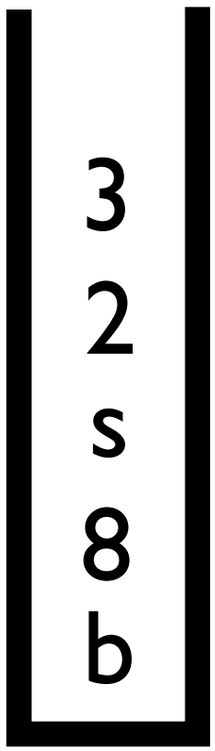
PixelLocation

seed value (254)

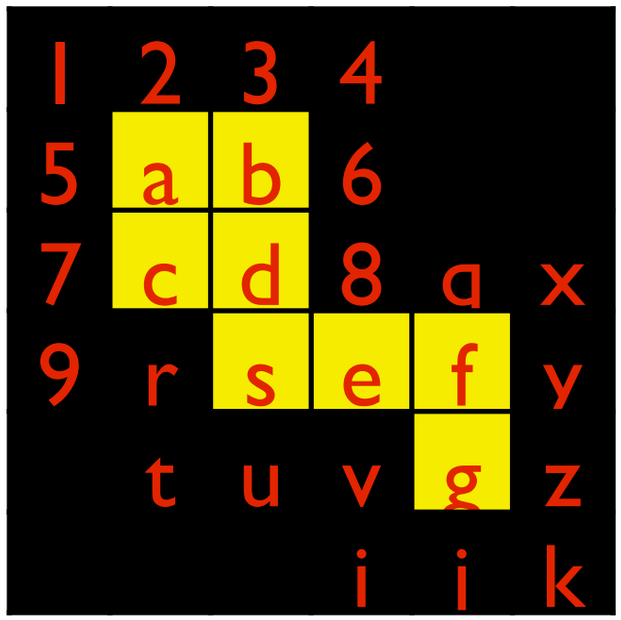
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack

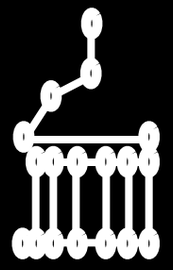


6




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

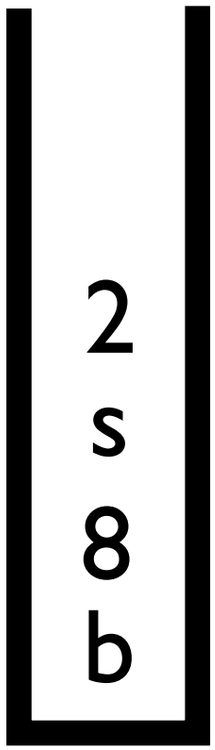
PixelLocation

seed value (254)

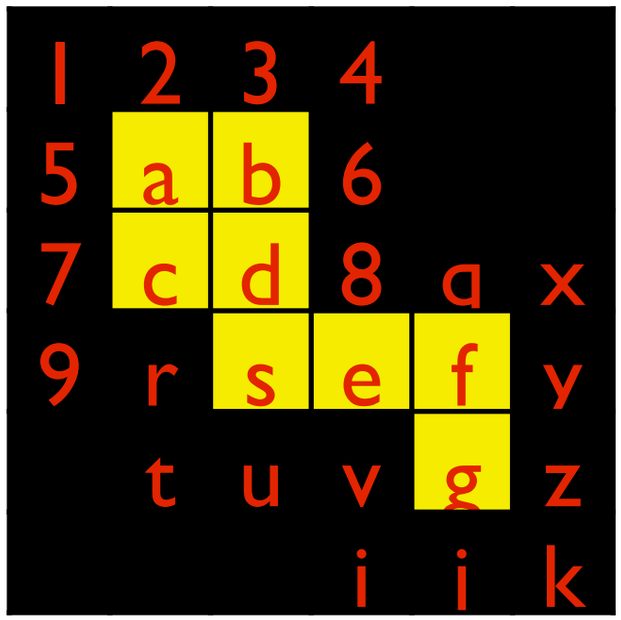
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack



3




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

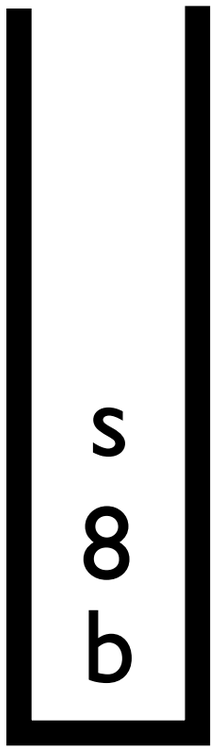
PixelLocation

seed value (254)

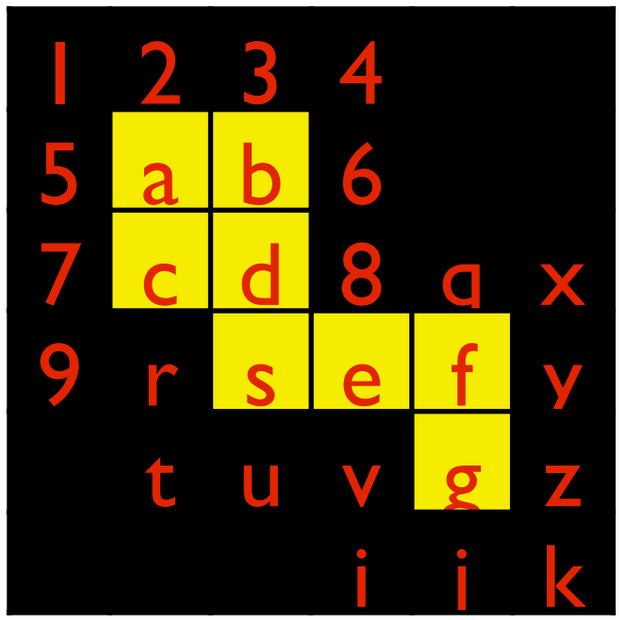
ccLabel value (255)

up r-1,c
right r,c+1
down r+1,c
left r,c-1

top of stack



2




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

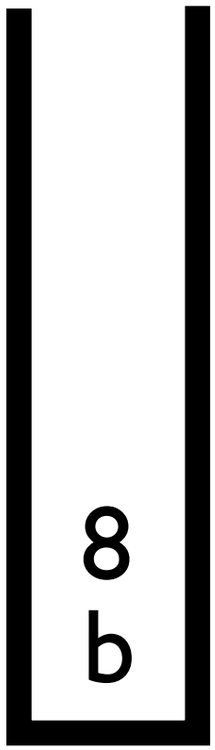
PixelLocation

seed value (254)

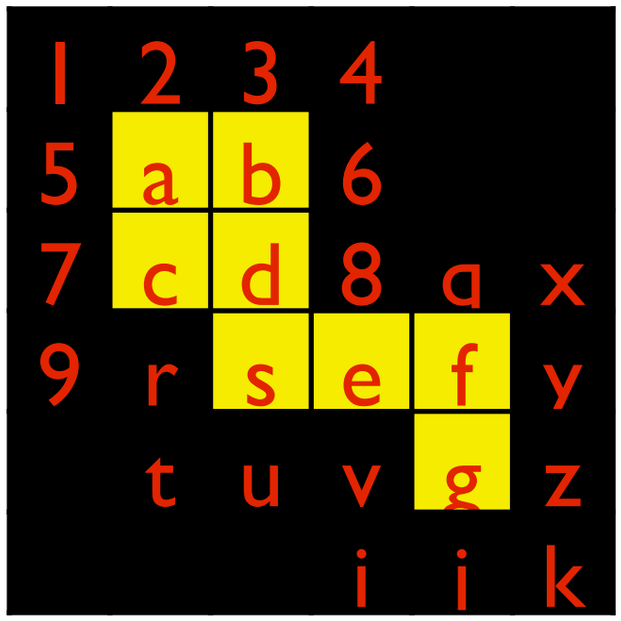
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

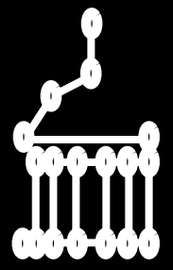


s



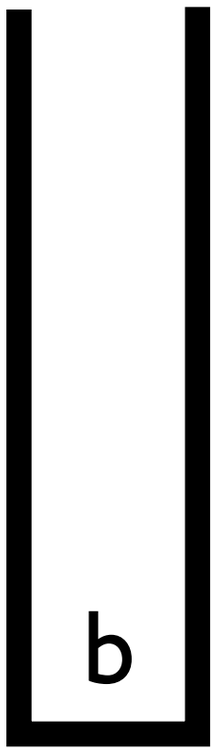

```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



Recall: pseudocode for finding one connected component using depth-first search

top of stack



8

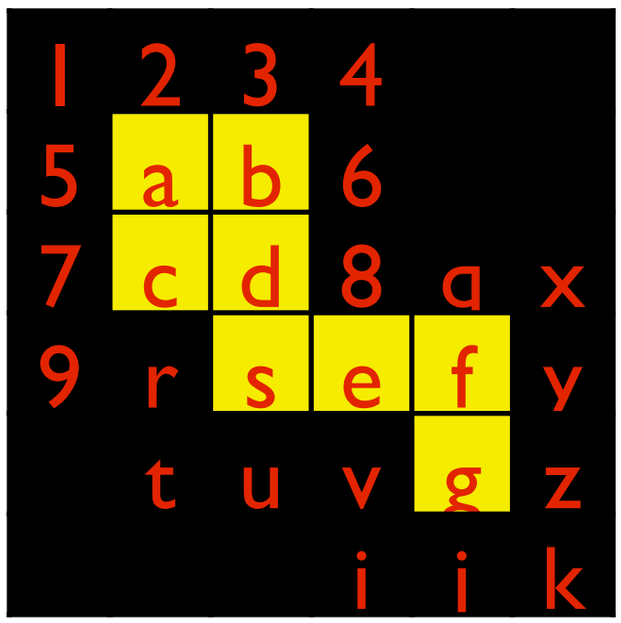
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

PixelLocation

seed value (254)

ccLabel value (255)

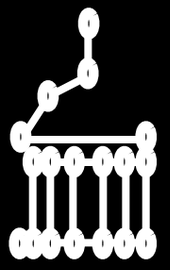
up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1




```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex

Recall: pseudocode for finding one connected component using depth-first search



```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

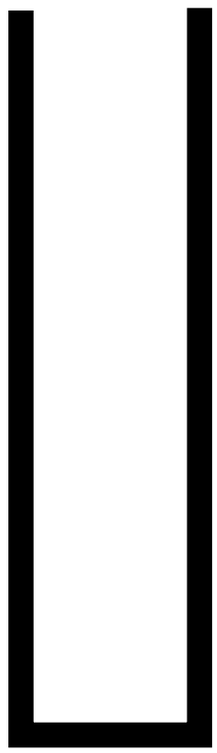
PixelLocation

seed value (254)

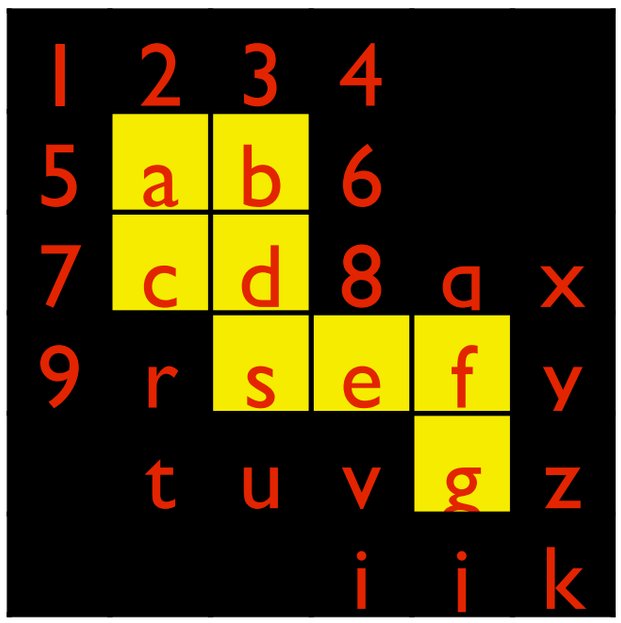
ccLabel value (255)

up	r-1,c
right	r,c+1
down	r+1,c
left	r,c-1

top of stack

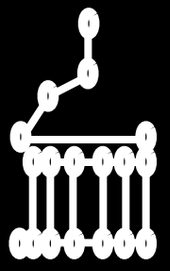


b



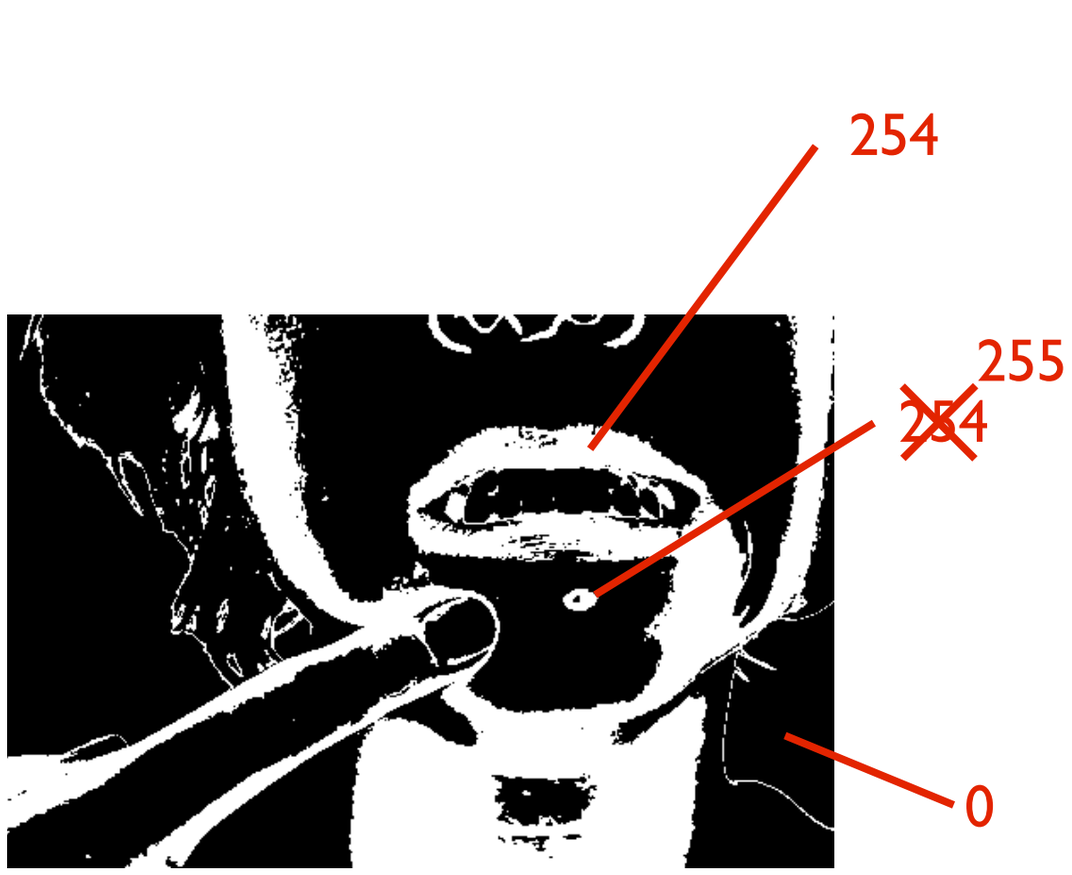

```
void markConnectedComponent(Image &image,
                            int seedRow,
                            int seedCol,
                            int ccLabel);
```

Refer to stack examples in lec31ex



High-level pseudocode (see homework5.pdf for more details)

5. Iterate through all pixels in the mask image, changing any pixel with an intensity of 254 to 0.
6. Write this version of the mask image to a file.



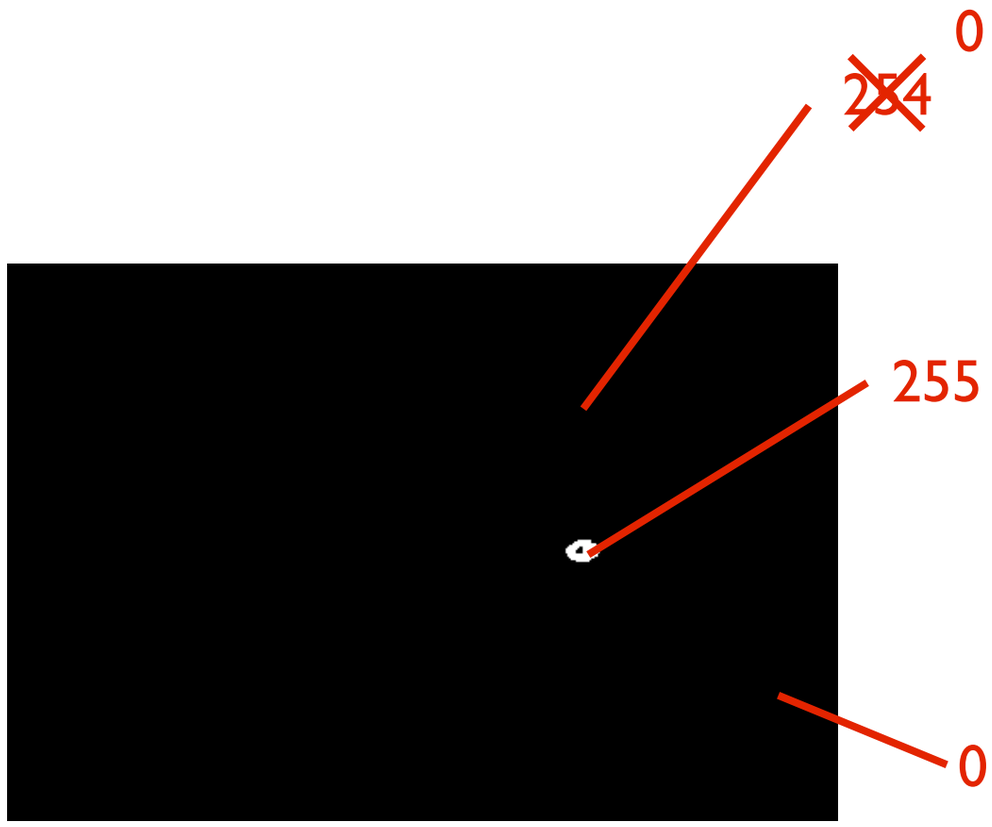
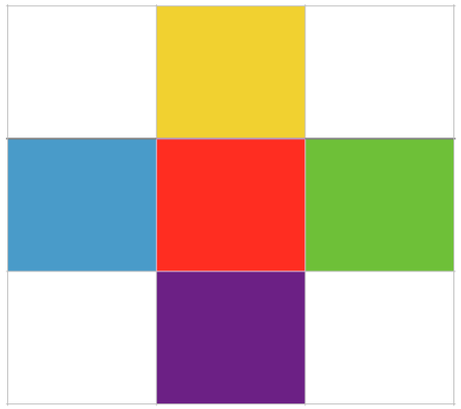


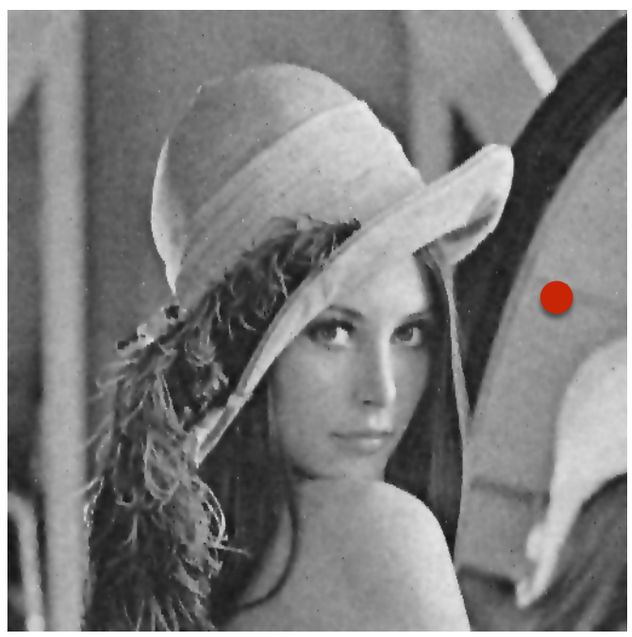
Image smoothing

Original



smoothed_pixel_center = (1-4a) Pixel_center

+ a (Pixel_right + Pixel_left +Pixel_top +Pixel_bottom)



Implement within main; Modify as a function later

Requires the completion of Image class

More details on classes this week