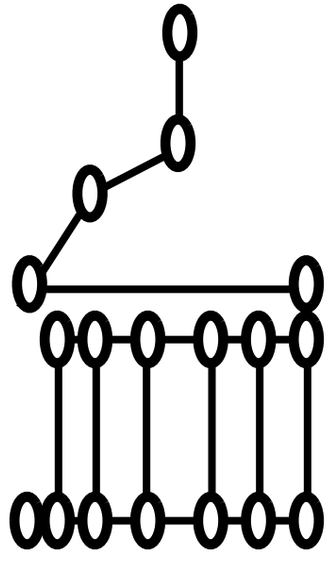
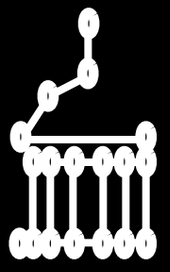
Lecture 24: Stacks and queues

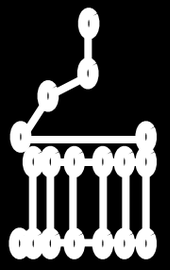
57:017 Computers in Engineering
Spring 2015





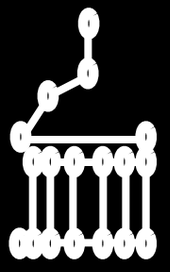
Reminders/announcements

- Use clicker channel 14 (Go/Ch → 14 → Go/Ch)
- **Mini-assignment** due **3/27** (Friday) by **12:30 p.m.**
  - class/lec24ma
- **Homework 4** due **4/5** (Sunday) by **11:59 p.m.**
- **Exam 2** is **4/7** (Thursday) **6:30-8:30 p.m.**
  - Please e-mail cie@engineering.uiowa.edu **today** if you need a make-up exam.
- 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



Schedule for part II (ten content lectures + one review lecture + one exam)

HW3 (individual): due 3/23 (M)

C structures (10.1-10.7):
Lec 17, Lec 18, Lec 19

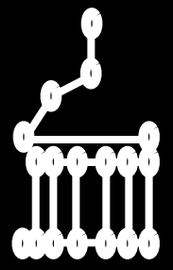
Linked lists, stacks, and queues (12.1-12.6):
Lec 20, Lec 21, Lec 22, Lec 23, Lec 24

Binary trees and recursive functions (5.14-5.16, 12.7)
Lec 25, Lec 26

HW4 (group): due 4/5 (S)

Review: Lec 27

Exam 2: **Tuesday**, 4/7, 6:30-8:30 p.m. in W10 PBB



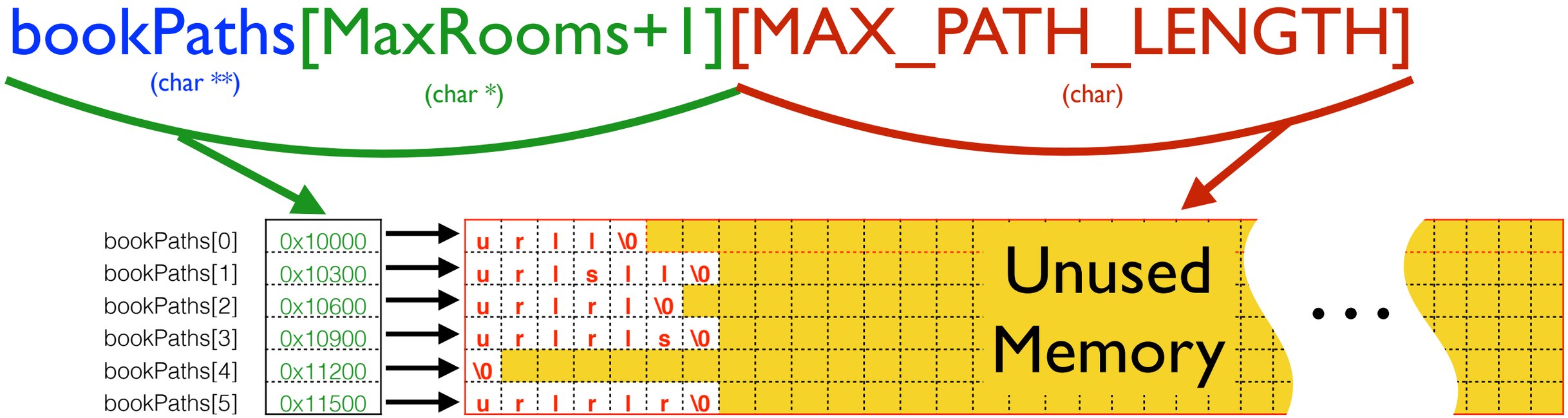
Today's topics

- Preview of HW4
- Introduction to stacks and queues
- Example application of stacks and queues
- Mini-assignment (finishing the implementations of stack/queue operations)

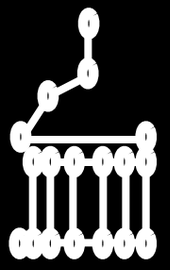
Reading: 12.5-12.6
Mini-assignment (due F, 3/27 by 12:30 a.m.): class/lec24ma

Preview of HW4

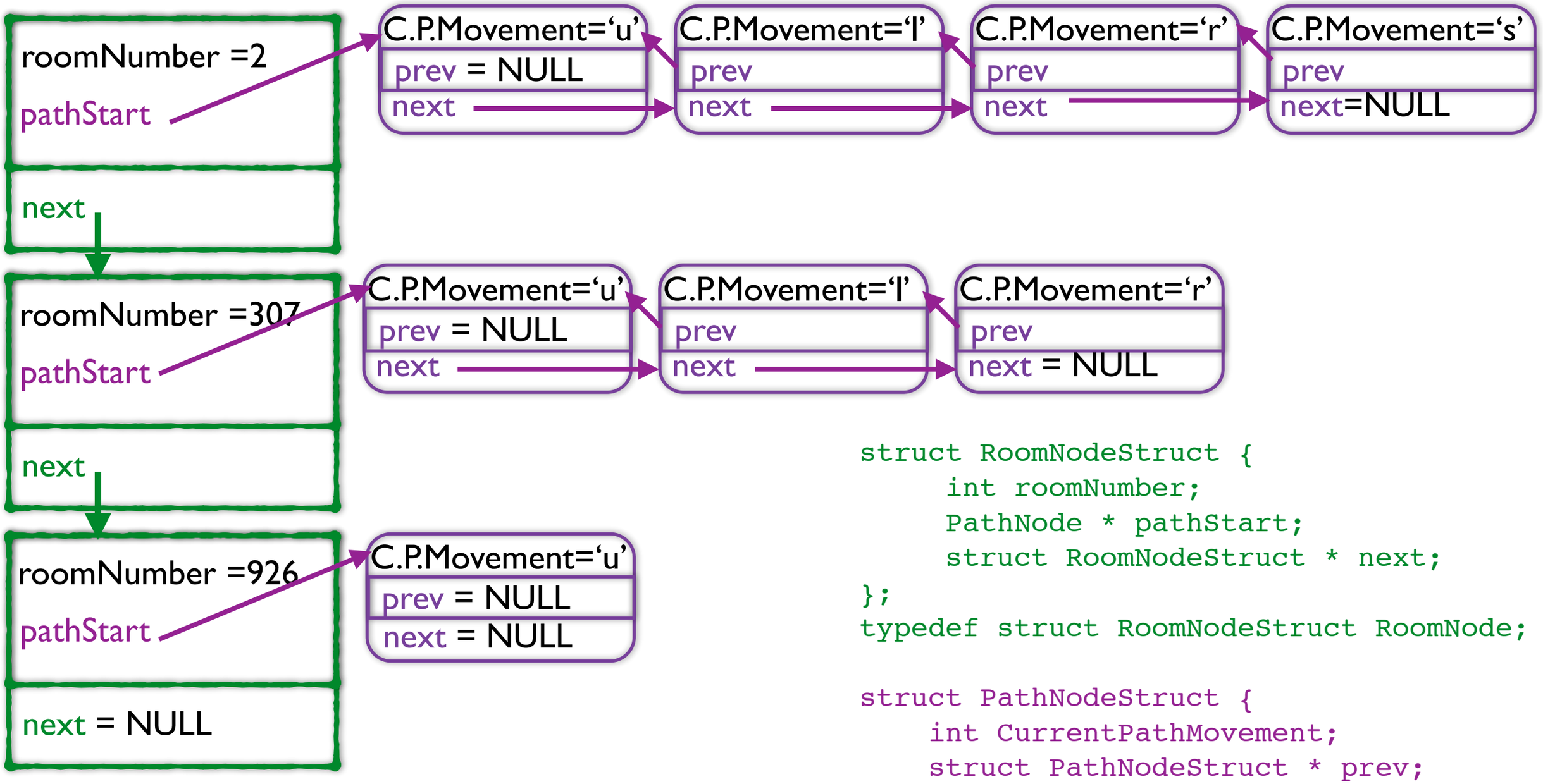
HW 3 Path Storage Arragement



- Robot has Limited Memory
- What if rooms are Not Numbered sequentially?
  - i.e. rooms #'s are [2, 307, 926, 1043]
- What if library expands and adds more rooms?

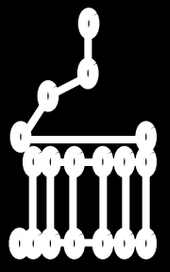


New Efficient Path Storage Needed




```
struct RoomNodeStruct {
      int roomNumber;
      PathNode * pathStart;
      struct RoomNodeStruct * next;
};
typedef struct RoomNodeStruct RoomNode;

struct PathNodeStruct {
    int CurrentPathMovement;
    struct PathNodeStruct * prev;
    struct PathNodeStruct * next;
};
typedef struct PathNodeStruct PathNode;
```

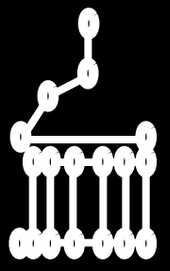


Why Double Linked List? (prev & next)

- We need to traverse the path in both directions!
  - Forward Traverse when going from book depository to room
  - Reverse Traverse when going from room to depository

One way to get reverse path of 5 element single linked list

```
startPtr->next->next->next->next->CurrentPathMovement
startPtr->next->next->next->CurrentPathMovement
startPtr->next->next->CurrentPathMovement
startPtr->next->CurrentPathMovement
startPtr->CurrentPathMovement
```



Final Comments on HW4

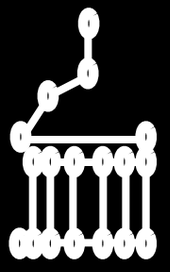
- NOTE: The following structures have *BUGS* in them. THE FOLLOWING CODE IS FOR INSPIRATION, BUT WILL NOT WORK AS CODED!

```
struct RoomNodeStruct {
     int roomNumber;
     PathNode * pathStart;
     struct RoomNodeStruct * next;
};
```

```
struct PathNodeStruct {
    int CurrentPathMovement;
    struct PathNodeStruct * prev;
    struct PathNodeStruct * next;
};
```

```
typedef struct RoomNodeStruct RoomNode; typedef struct PathNodeStruct PathNode;
```

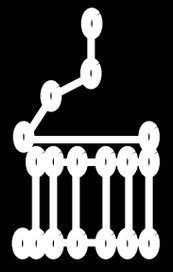
- PHASE 1 May not be rigorously graded if PHASE 2 works perfectly
  - If there are problems with PHASE2, then we will consider PHASE1 (as of 3/28) for partial credit
  - Doing PHASE 1 before PHASE 2 will save you considerable debugging time



CQ: Which of the following best describes how well you think that you understand linked lists?

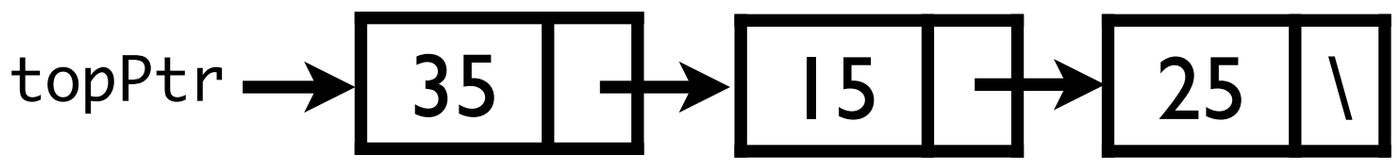
- A: Very well (e.g., you understand the concepts and would feel comfortable implementing all of the functions that we have presented from "scratch")
- B: Fairly well (e.g., you understand the concepts, were mostly comfortable following the code of all of the functions we have presented, and would feel comfortable implementing aspects of the functions that we have presented)
- C: Somewhat (e.g., you mostly understand the concepts and can follow some of the presented code, but overall are still struggling to understand much of the code)
- D: Just barely (e.g., you have some notion of the concept of a linked list, but are completely lost when it comes to looking at any code related to a linked list)
- E: Not at all (e.g., you didn't even know that we had been covering linked lists recently)

Introduction to stacks and queues

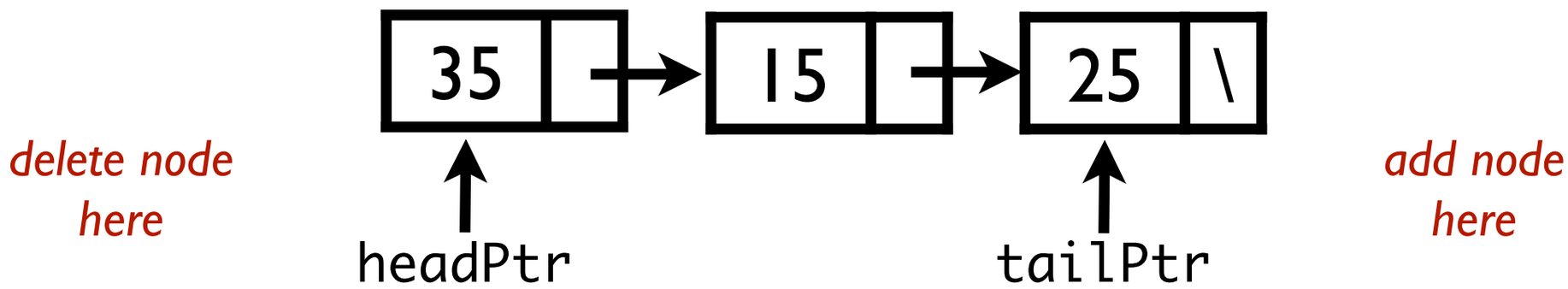


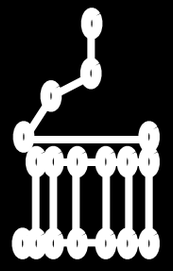
Stacks and queues can be considered constrained variations of linked lists

Stack: only add/delete nodes from the beginning (top) of the linked list



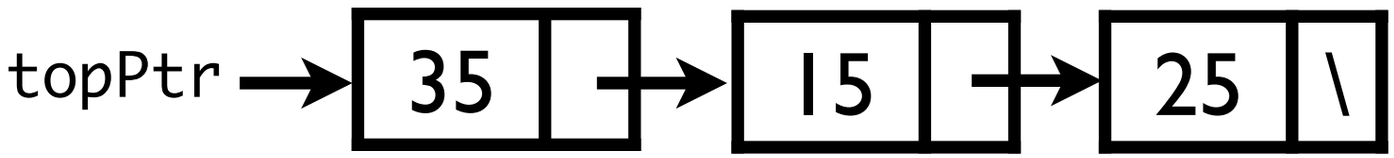
Queue: add nodes to the end (tail) of the linked list; delete nodes from the beginning (head) of the linked list; maintain a pointer to **both** the beginning and end of the linked list





Stacks and queues can be considered constrained variations of linked lists

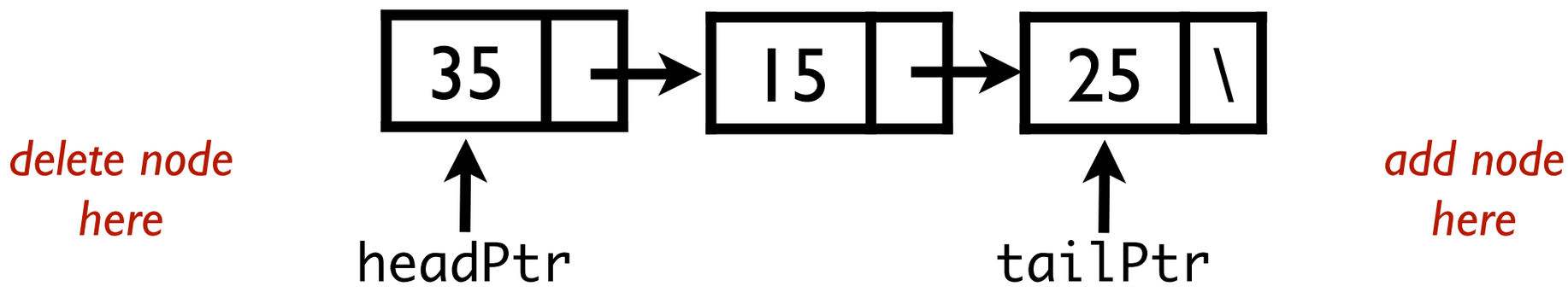
Stack: only add/delete nodes from the beginning (top) of the linked list



add/delete node here

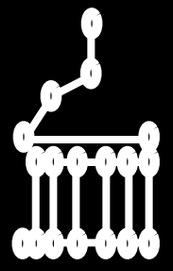
These two data structures are **very** common!

Queue: add nodes to the end (tail) of the linked list; delete nodes from the beginning (head) of the linked list; maintain a pointer to **both** the beginning and end of the linked list



delete node here

add node here



With a stack, you only add/delete nodes from the top of the stack

"First-in, last-out" (FILO) data structure

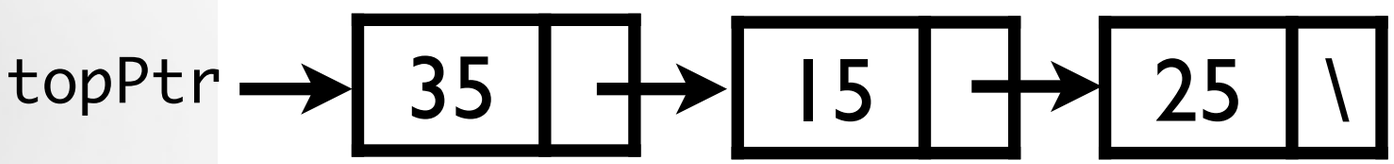
aka "Last-in, first-out" (LIFO) data structure

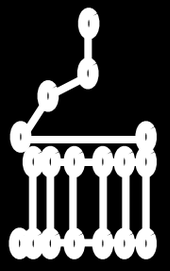
Top

Basic operations:

push: add node to top of stack

pop: delete node from top of stack and return value





With a queue, you add nodes at the tail of the queue and delete nodes from the head of the queue

"First-in, first-out" (FIFO) data structure

Basic operations:

enqueue: add node to tail of queue

dequeue: delete node from head of queue and return value

head

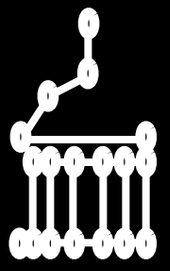
DELETE NODE HERE

dequeue

tail

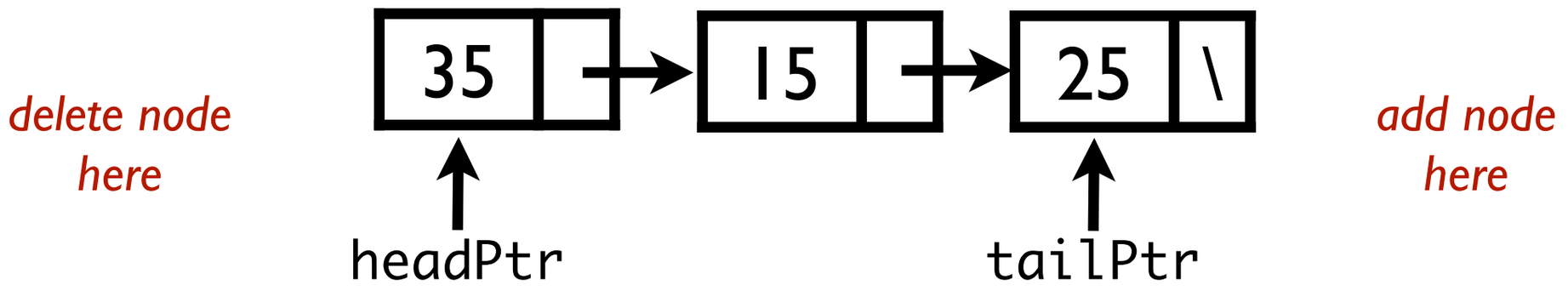
ADD NODE HERE

enqueue



With a queue, you add nodes at the tail of the queue and delete nodes from the head of the queue

"First-in, first-out" (FIFO) data structure



head

DELETE NODE HERE

dequeue

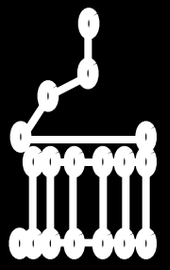
tail

ADD NODE HERE

enqueue

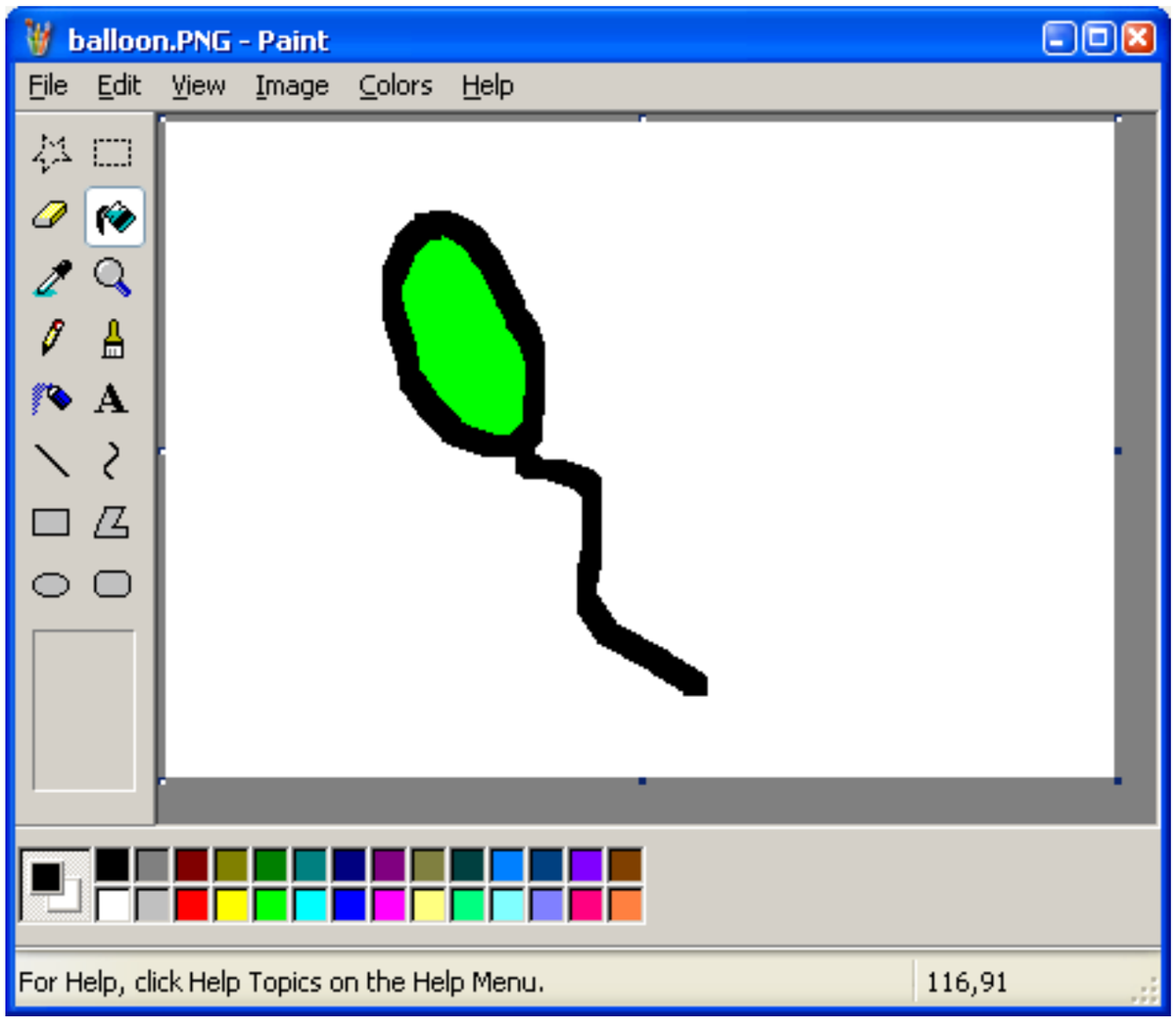
Example application of stacks and queues

You are not explicitly responsible for this material!

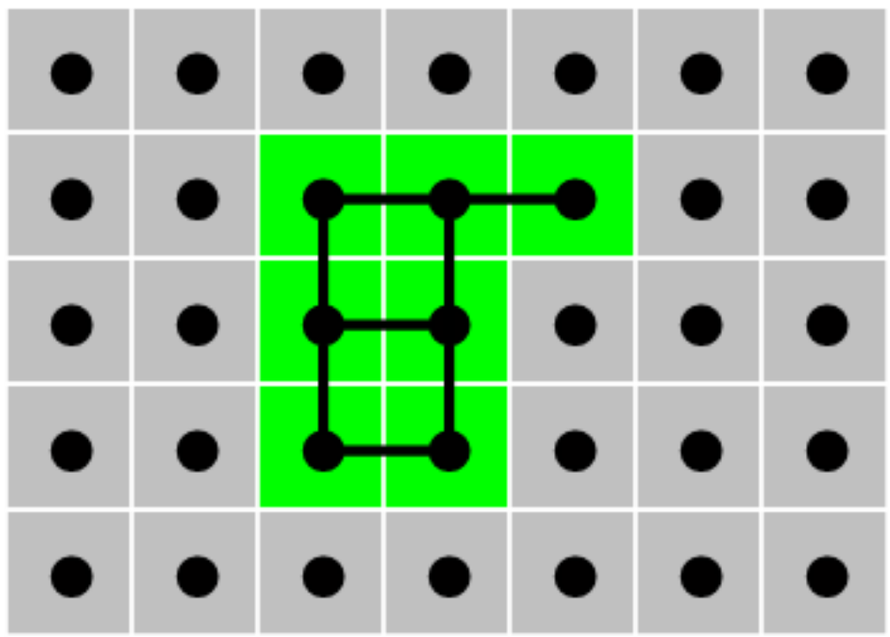


Example application of stacks and queues

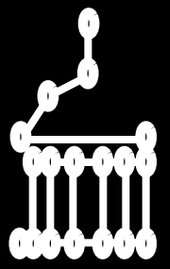
Find the "connected component" of a pixel and change this component's color.



Paint bucket tool

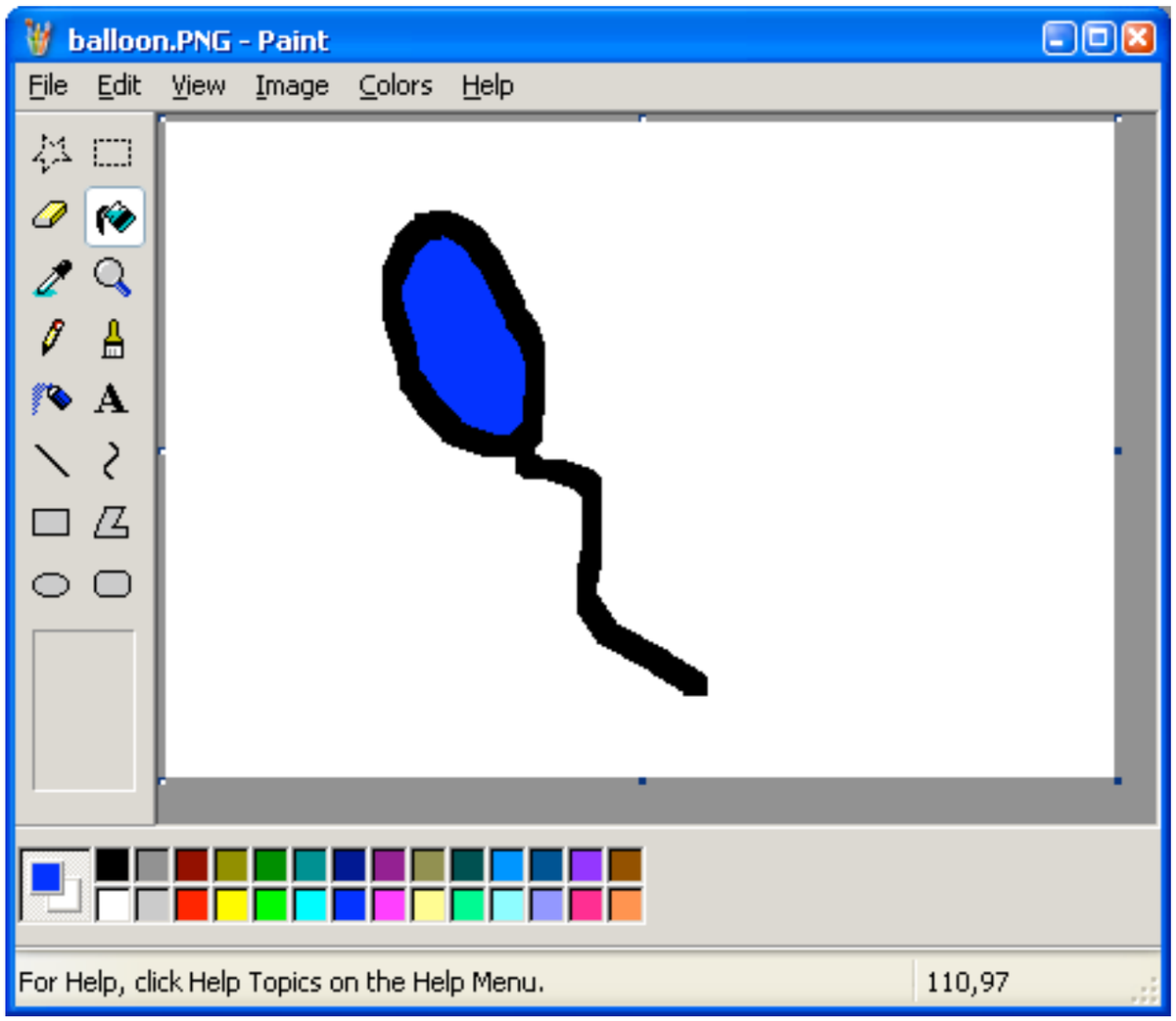


Interior of balloon: one connected component.

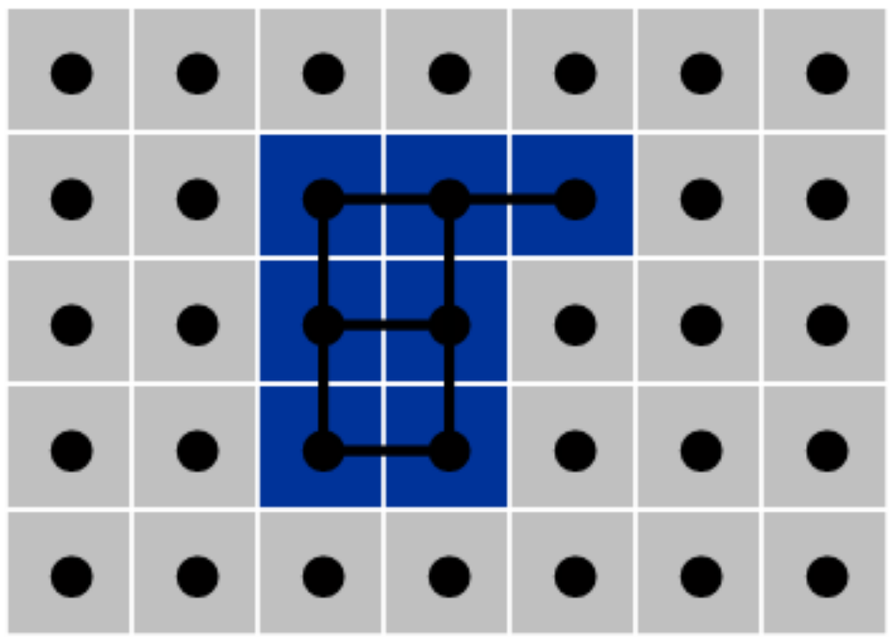


Example application of stacks and queues

Find the "connected component" of a pixel and change this component's color.



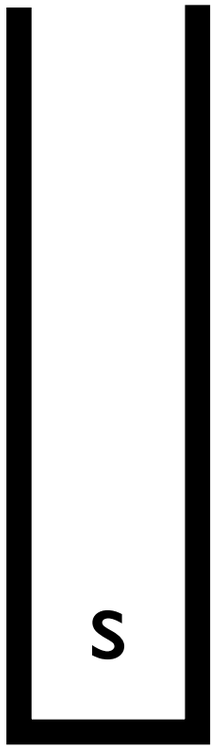
Paint bucket tool



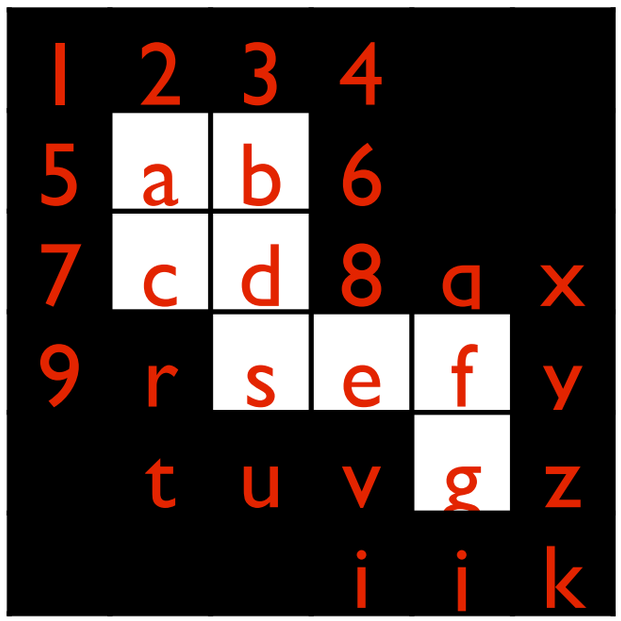
Interior of balloon: one connected component.

Finding a connected component using depth-first search (with a stack)

top
of stack



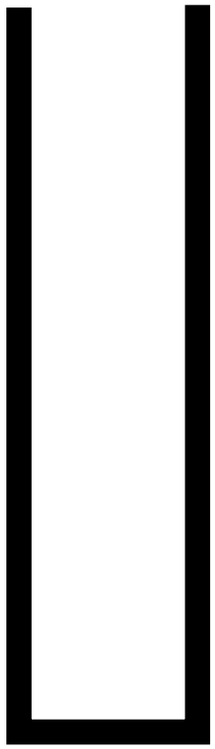

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

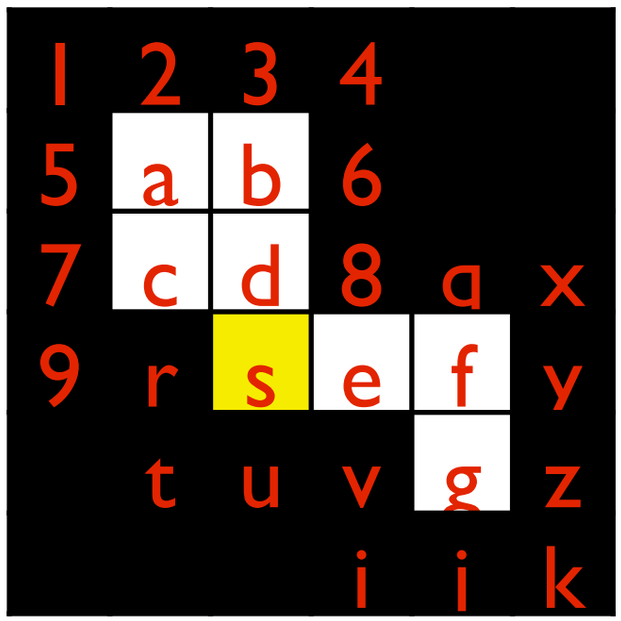
Finding a connected component using depth-first search (with a stack)

top
of stack

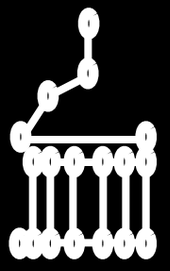


s

```
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
```

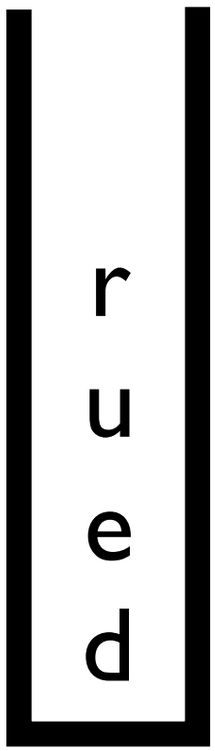


Depth-first search



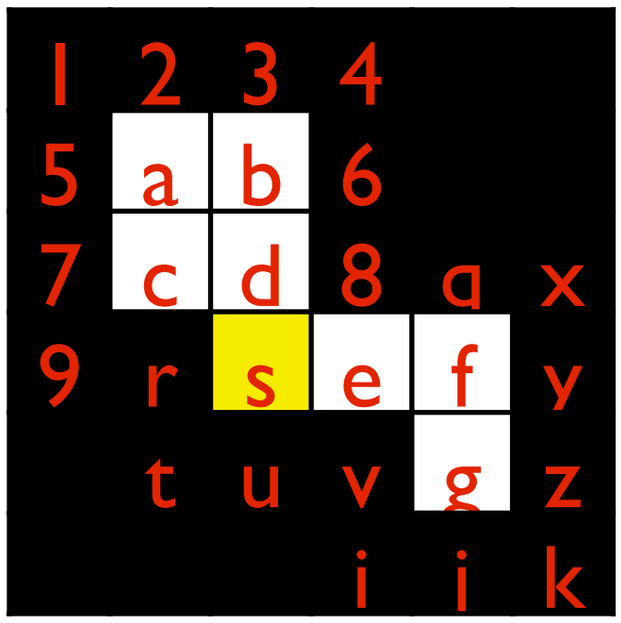
Finding a connected component using depth-first search (with a stack)

top of stack



s

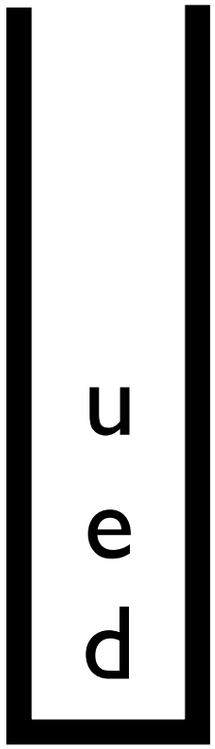
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

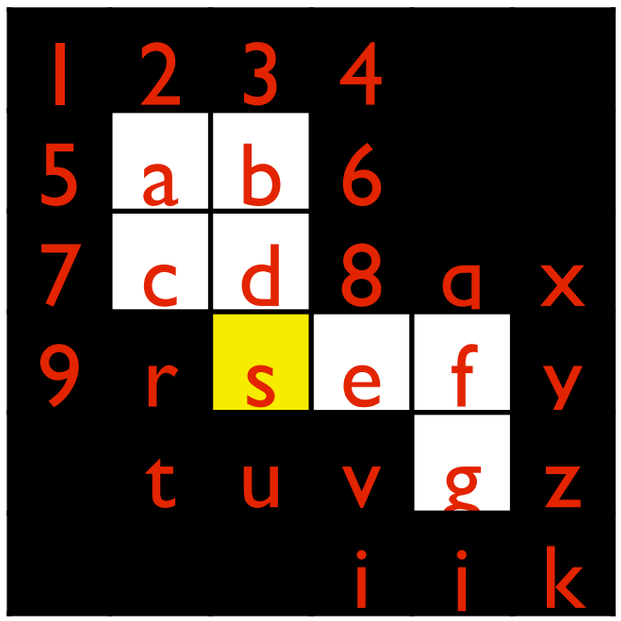
Finding a connected component using depth-first search (with a stack)

top of stack



r

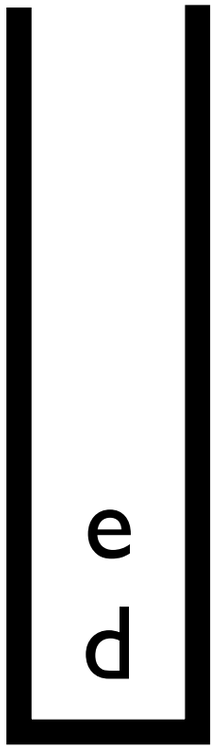
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

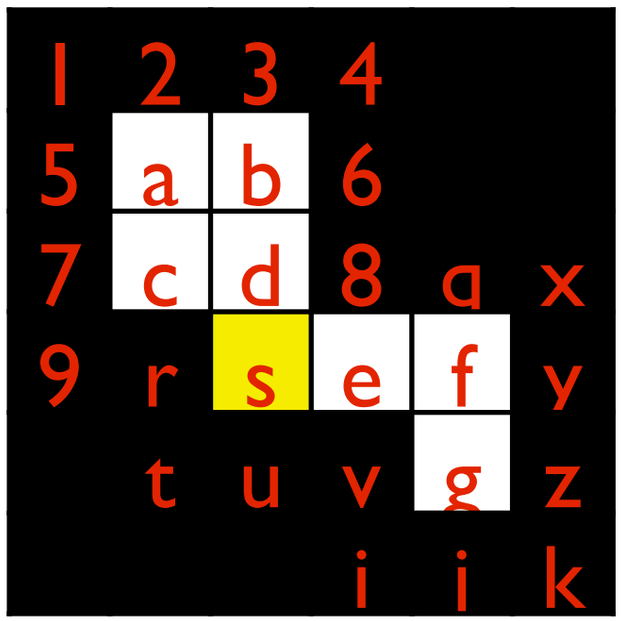
Finding a connected component using depth-first search (with a stack)

top
of stack

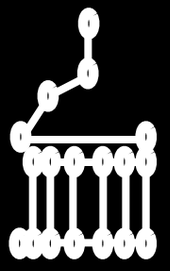


u

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

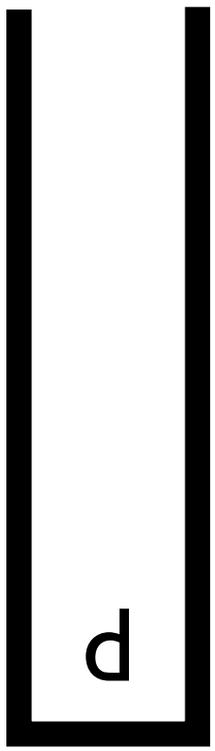


Depth-first search



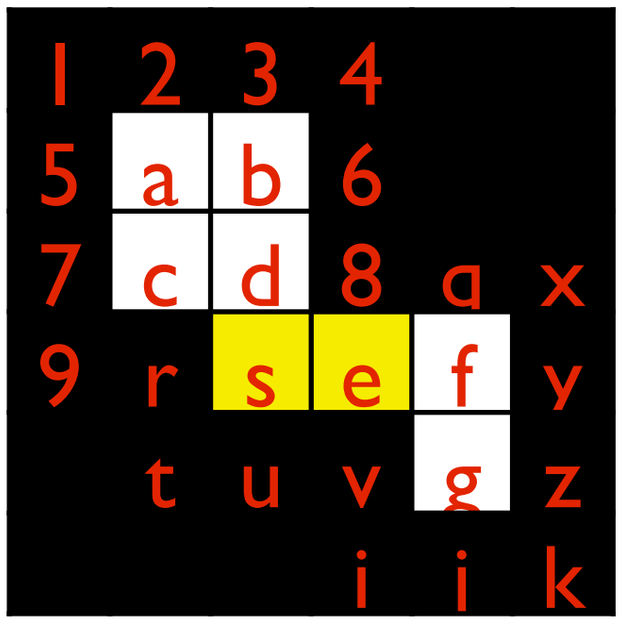
Finding a connected component using depth-first search (with a stack)

top
of stack



e

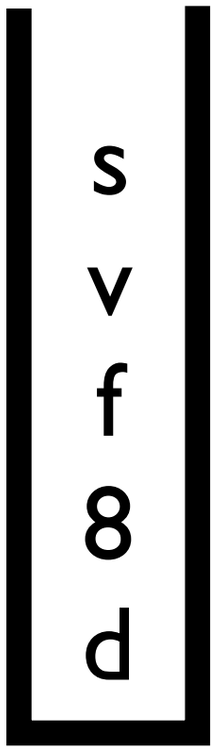
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

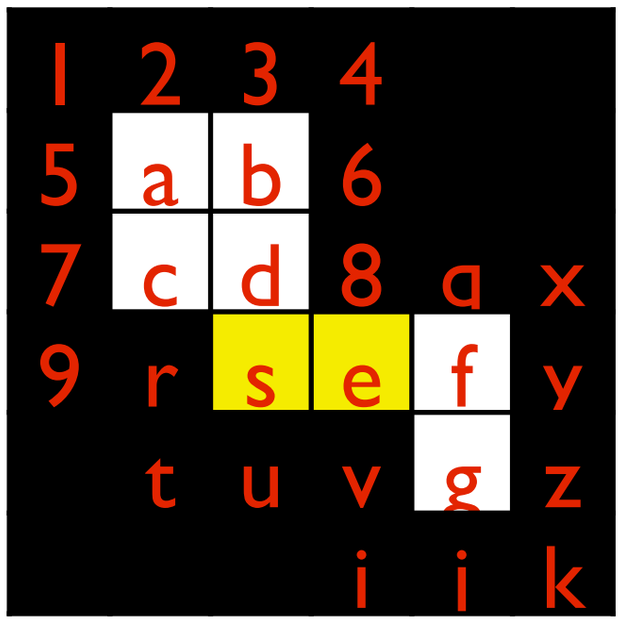
Finding a connected component using depth-first search (with a stack)

top of stack



e

```
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
```



Depth-first search

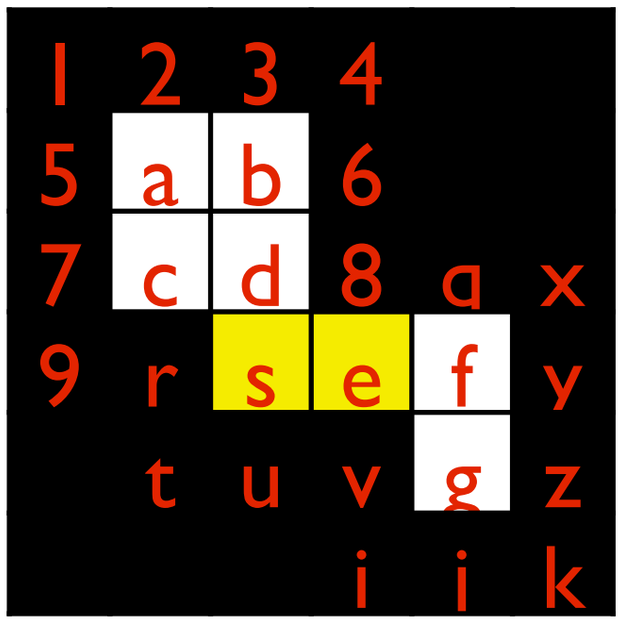
Finding a connected component using depth-first search (with a stack)

top of stack

v
f
8
d

s

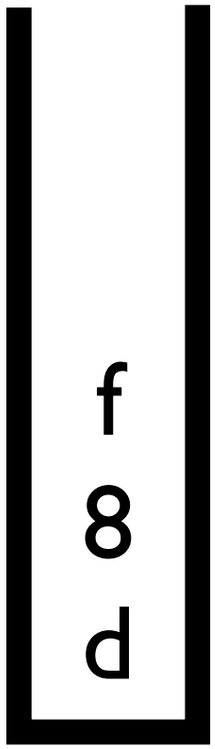
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack



Depth-first search

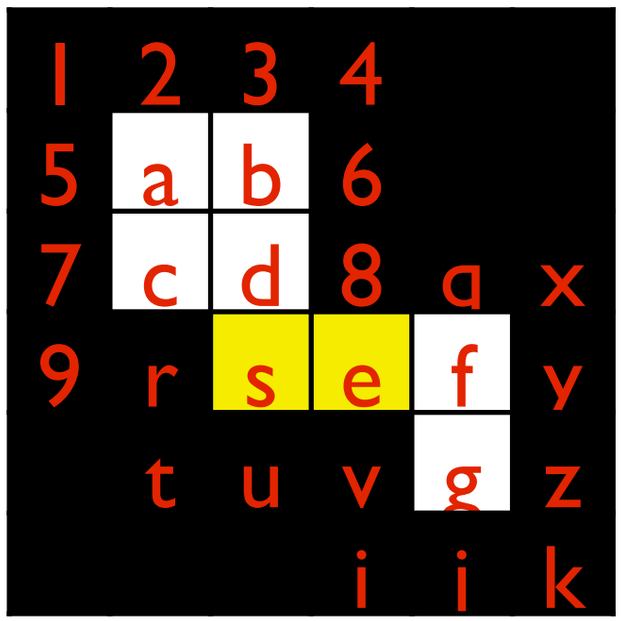
Finding a connected component using depth-first search (with a stack)

top
of stack

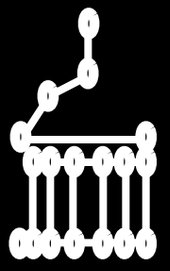


v

```
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
```

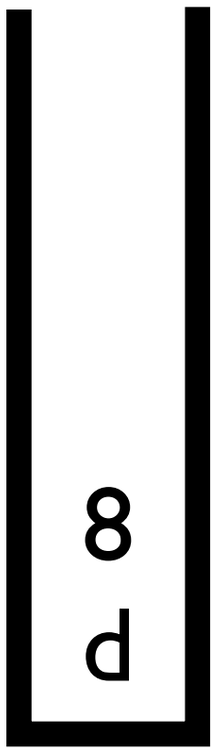


Depth-first search



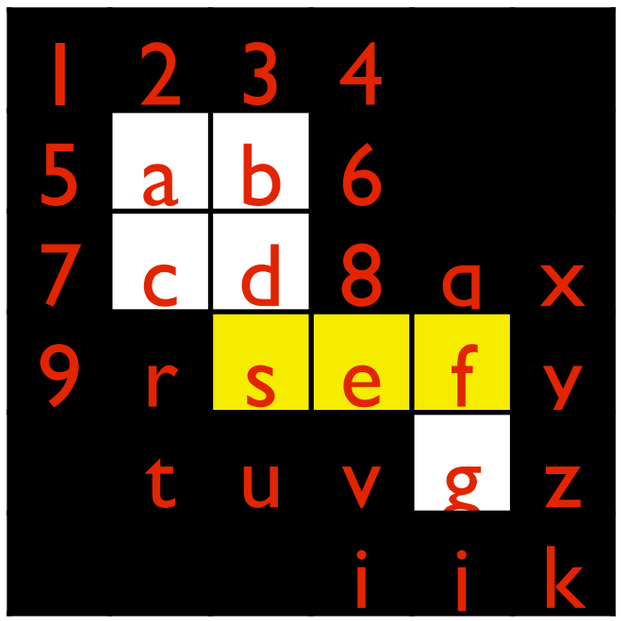
Finding a connected component using depth-first search (with a stack)

top
of stack



f

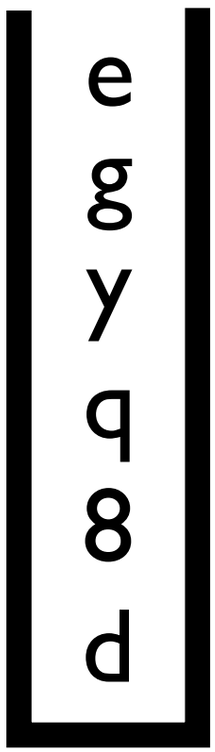
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

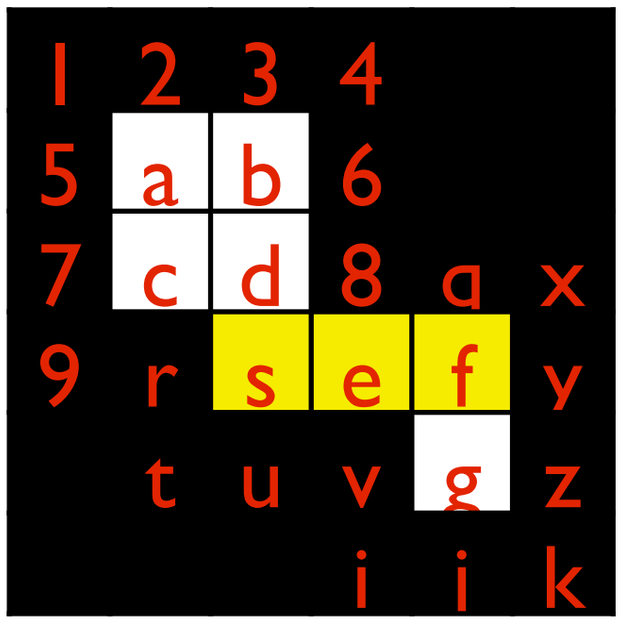
Finding a connected component using depth-first search (with a stack)

top
of stack



f

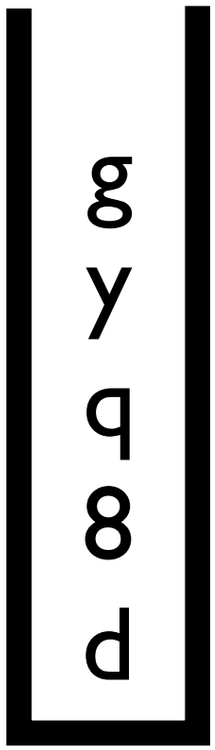
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

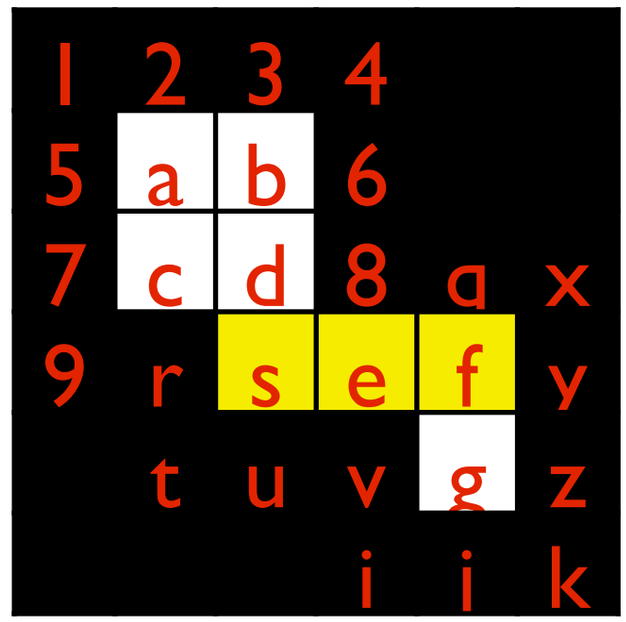
Finding a connected component using depth-first search (with a stack)

top
of stack



e

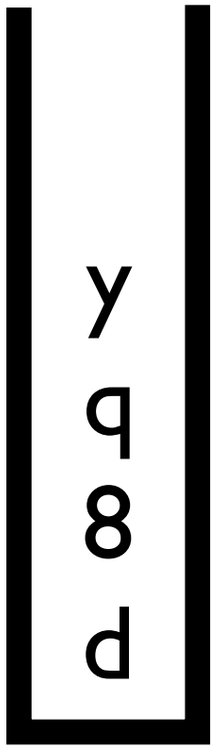
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

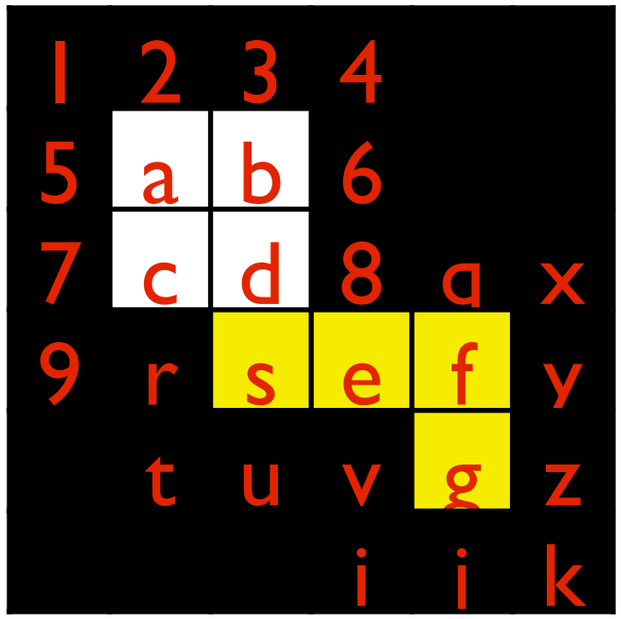
Finding a connected component using depth-first search (with a stack)

top
of stack



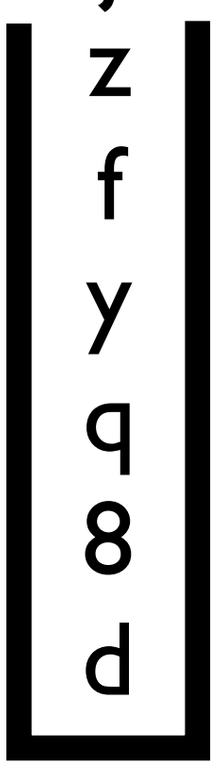
g

```
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
```

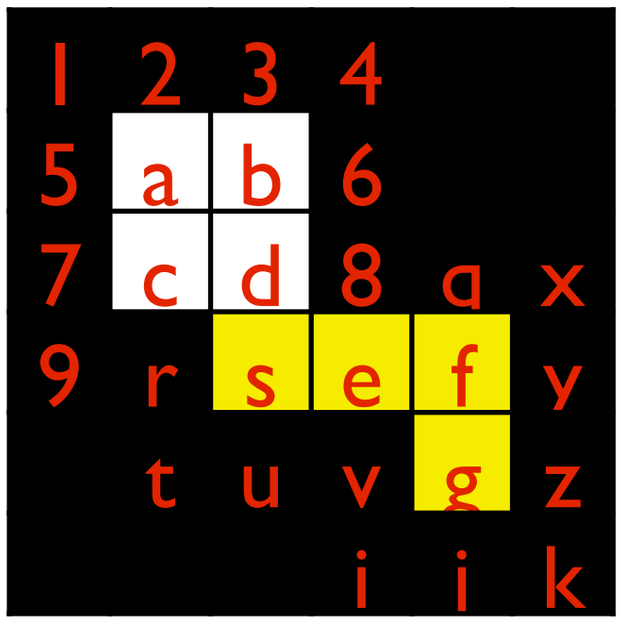


Depth-first search

Finding a connected component using depth-first search (with a stack)



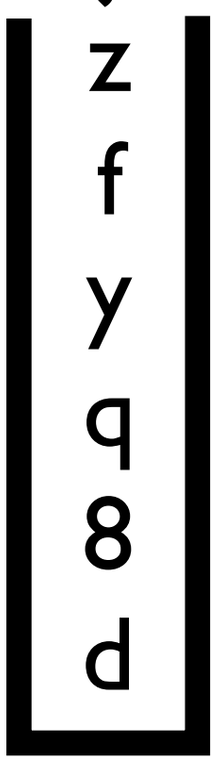

```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

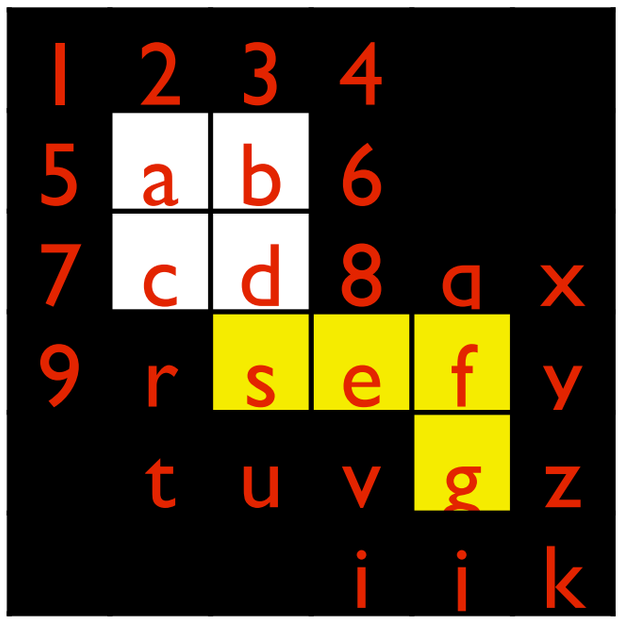
Finding a connected component using depth-first search (with a stack)

top of stack



v

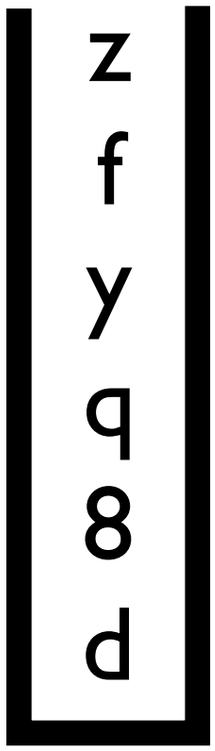
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

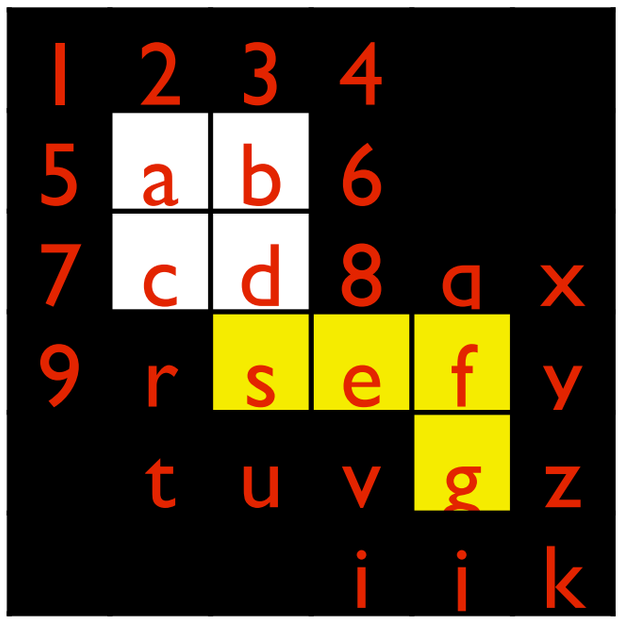
Finding a connected component using depth-first search (with a stack)

top
of stack



j

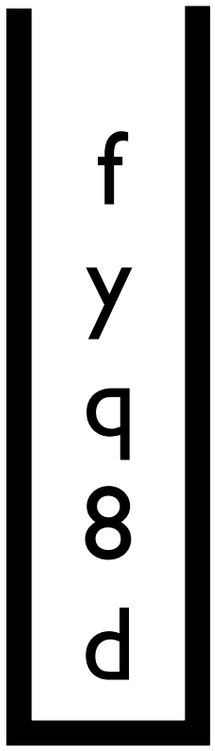
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

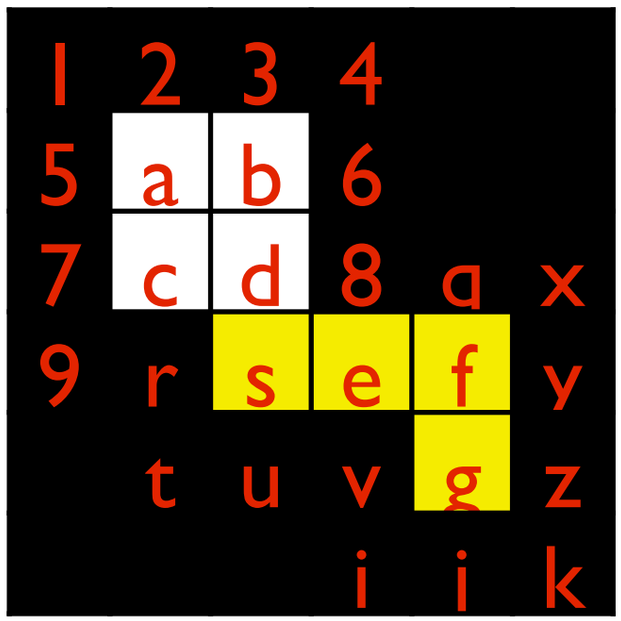
Finding a connected component using depth-first search (with a stack)

top
of stack



z

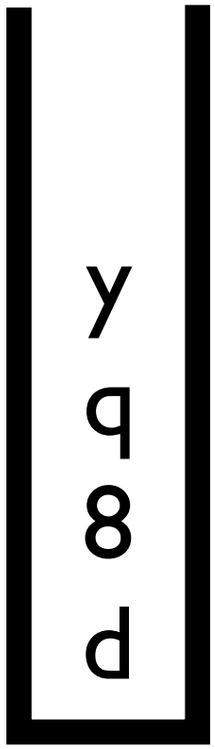
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

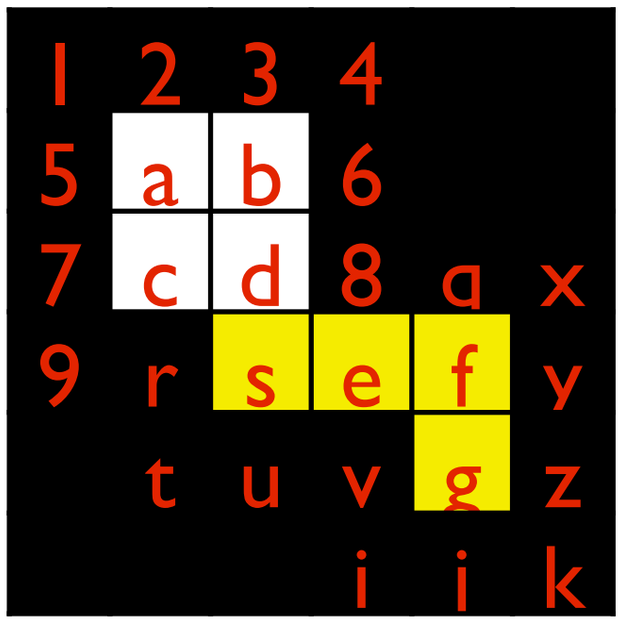
Finding a connected component using depth-first search (with a stack)

top of stack



f

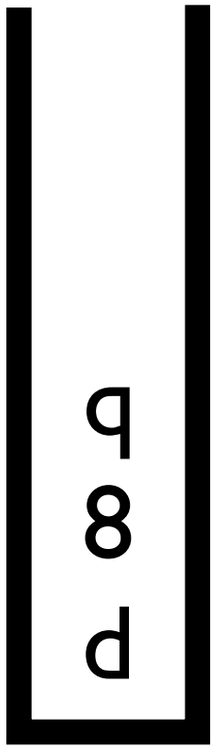
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

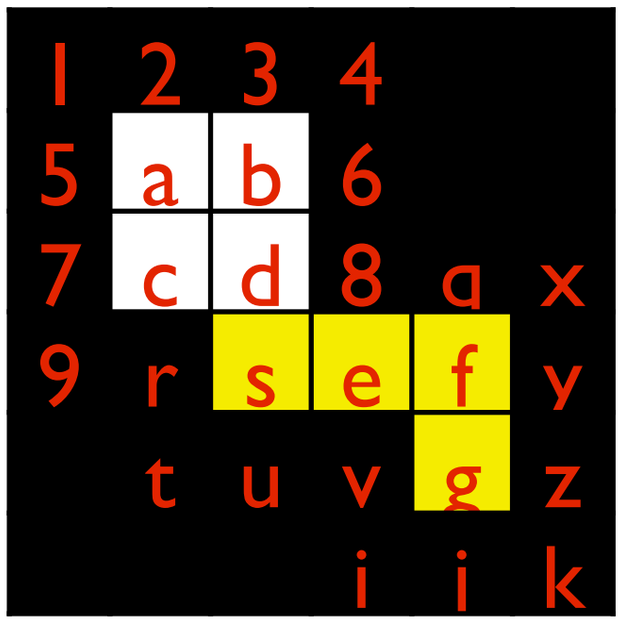
Finding a connected component using depth-first search (with a stack)

top
of stack

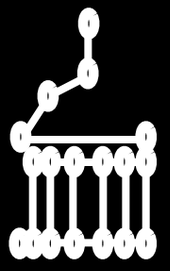


y

```
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
```

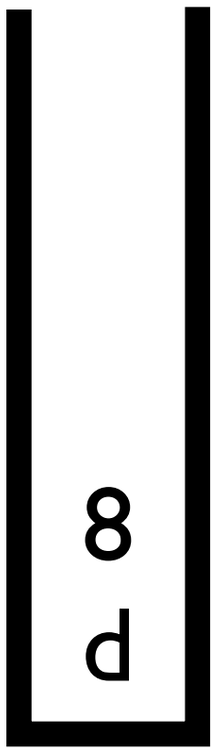


Depth-first search



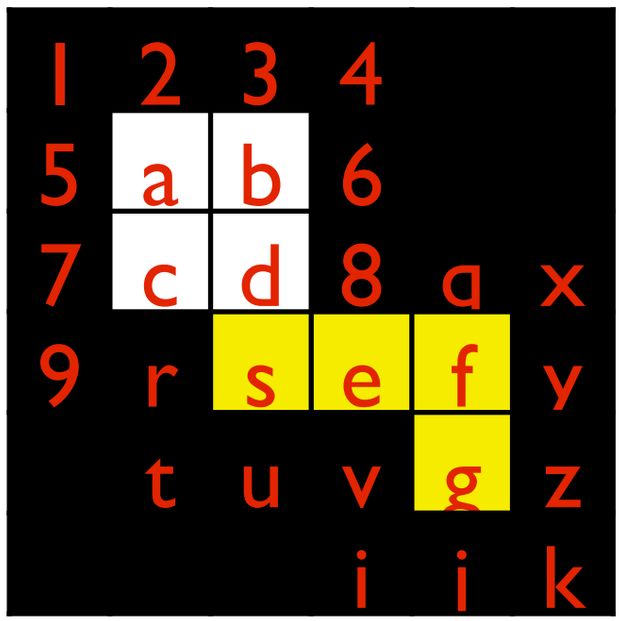
Finding a connected component using depth-first search (with a stack)

top
of stack



q

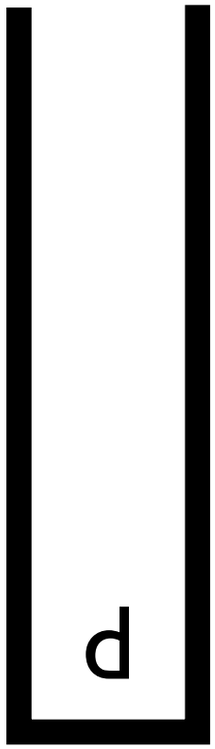
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

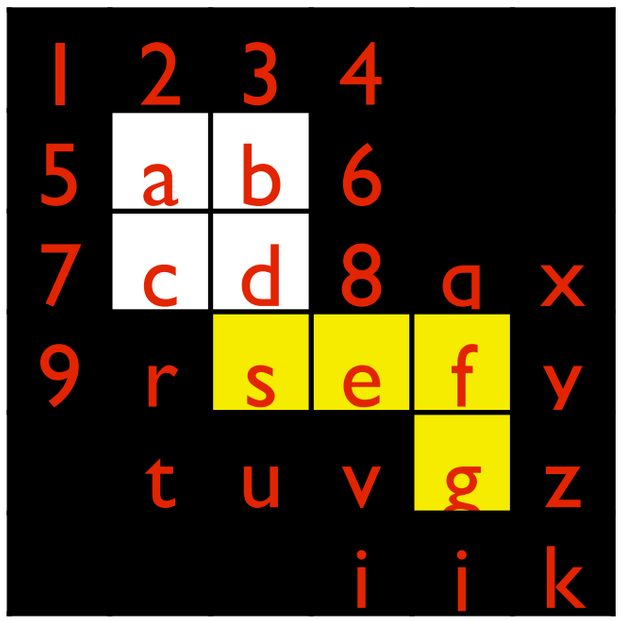
Finding a connected component using depth-first search (with a stack)

top
of stack



8

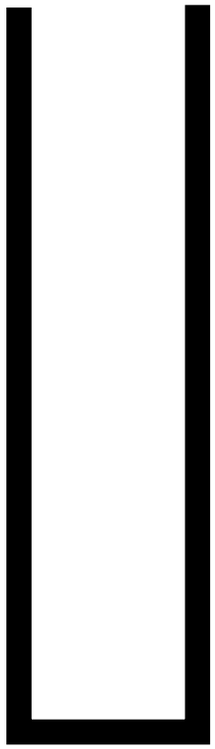
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

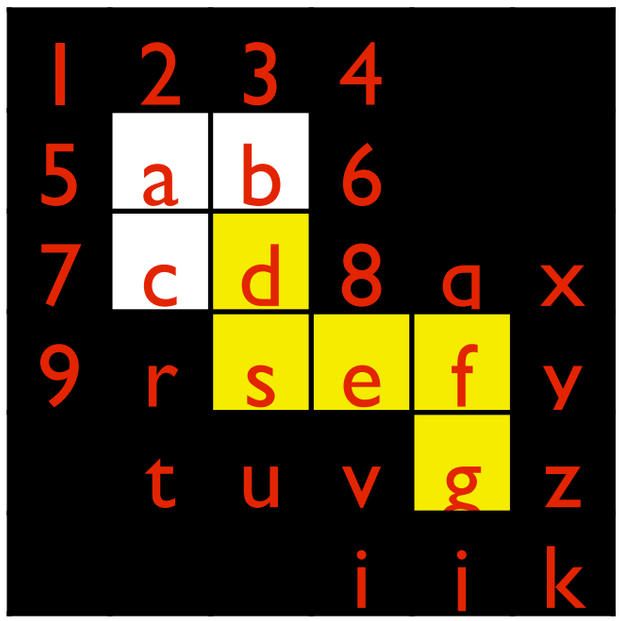
Finding a connected component using depth-first search (with a stack)

top
of stack



d

```
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
```



Depth-first search

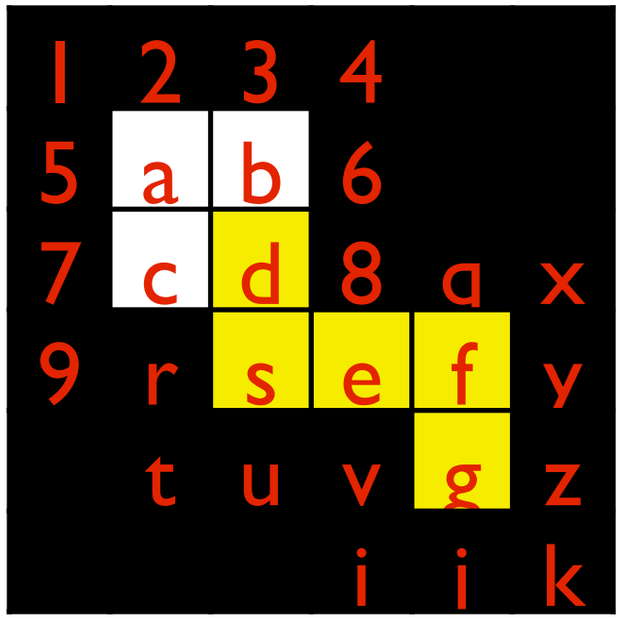
Finding a connected component using depth-first search (with a stack)

top of stack

c
s
8
b

d

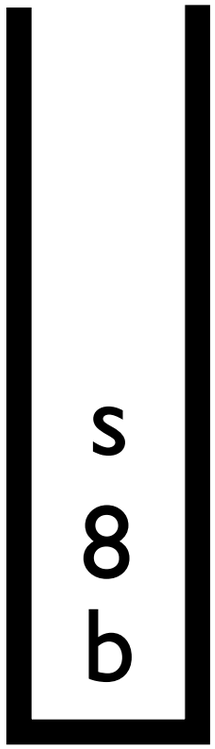
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

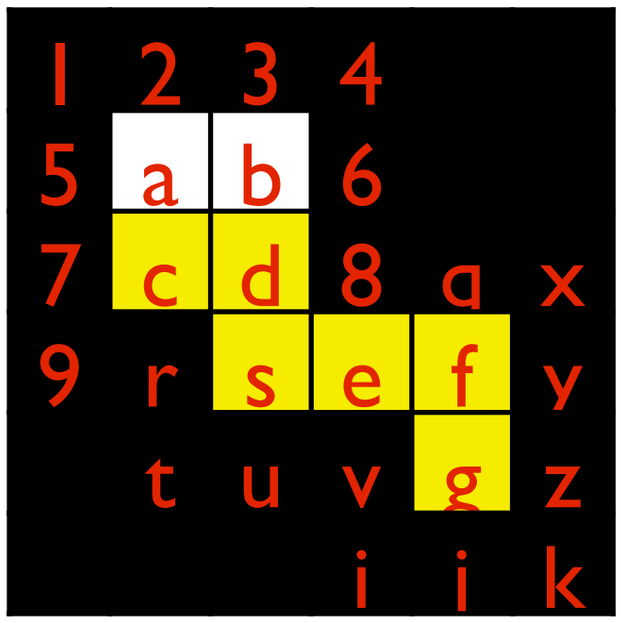
Finding a connected component using depth-first search (with a stack)

top
of stack



c

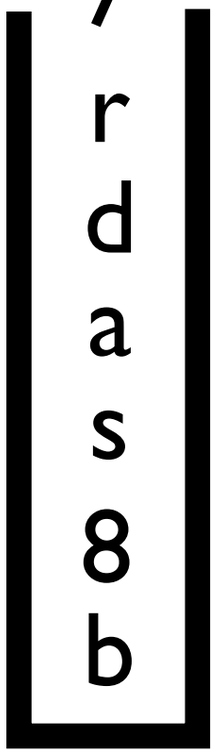
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

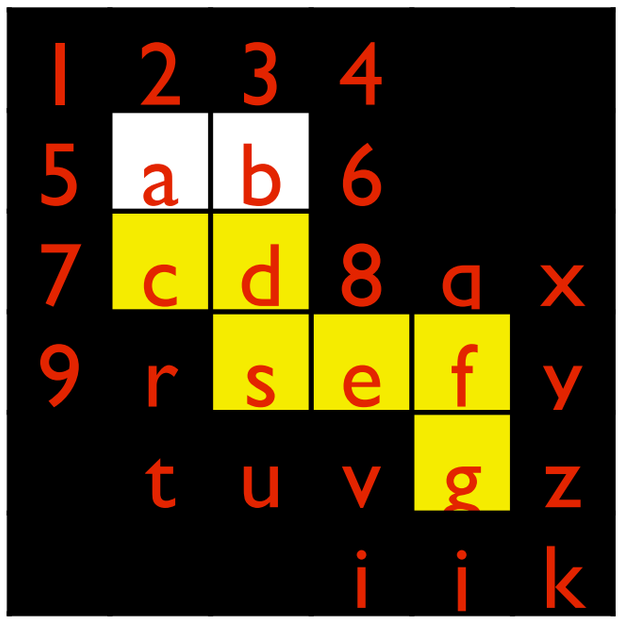
Finding a connected component using depth-first search (with a stack)

top of stack



c

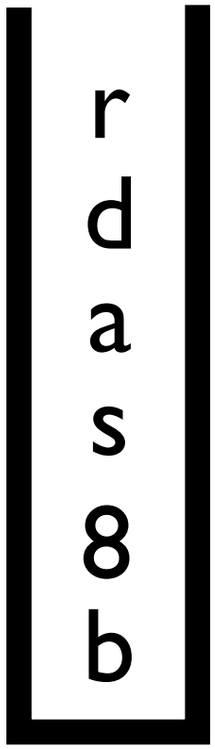
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

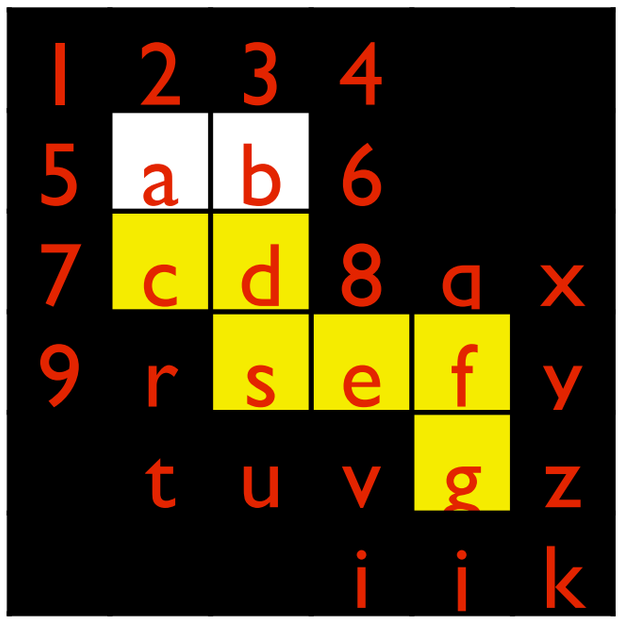
Finding a connected component using depth-first search (with a stack)

top
of stack



7

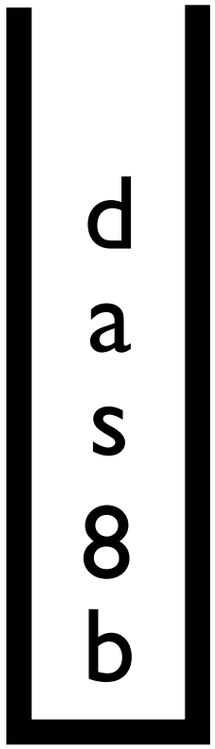
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

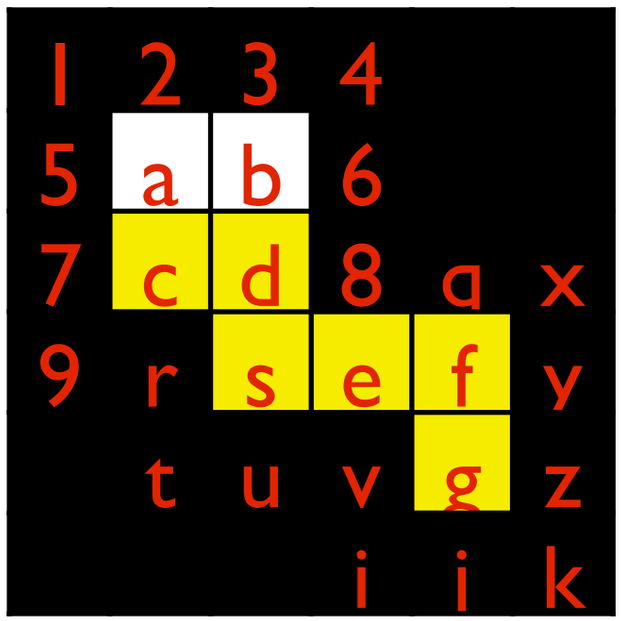
Finding a connected component using depth-first search (with a stack)

top
of stack



r

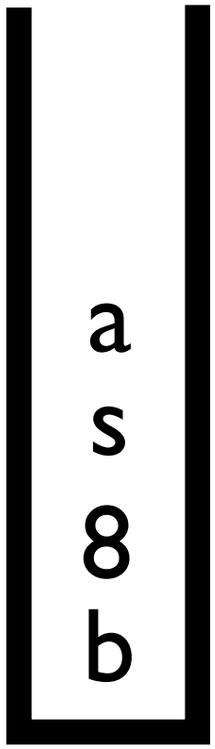
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

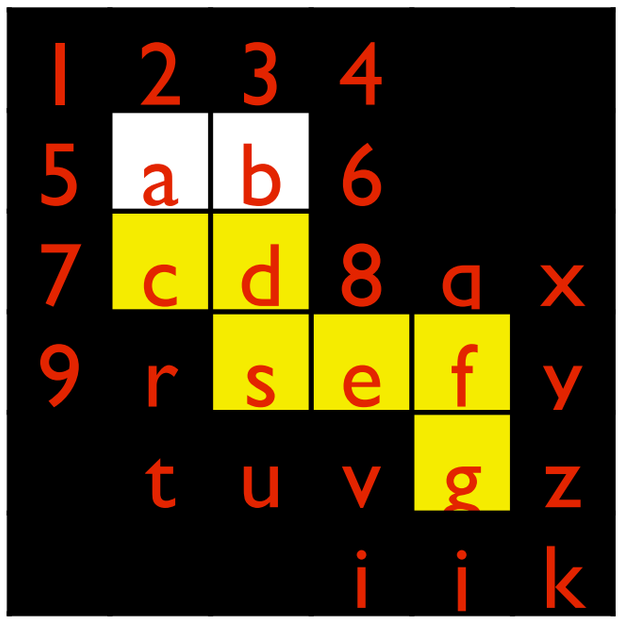
Finding a connected component using depth-first search (with a stack)

top
of stack

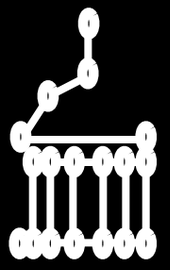


d

```
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
```

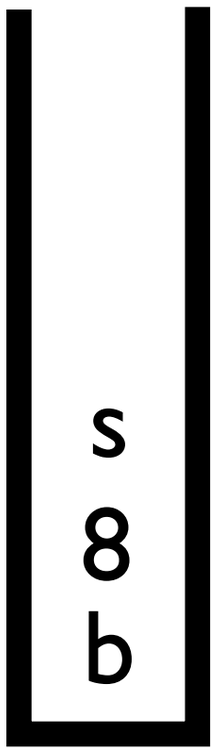


Depth-first search



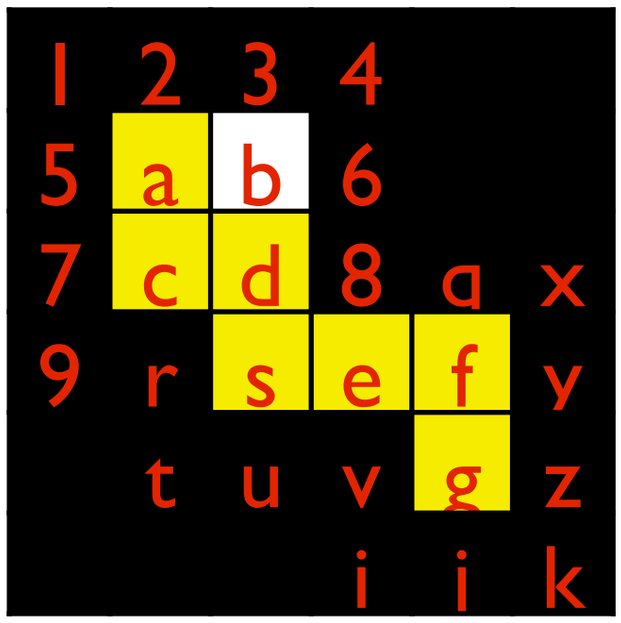
Finding a connected component using depth-first search (with a stack)

top
of stack

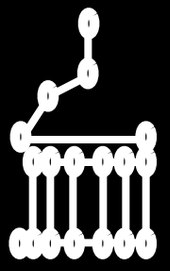


a

```
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
```

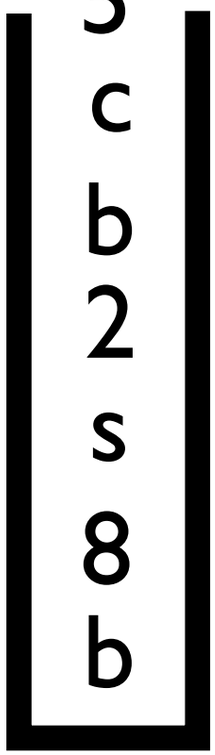


Depth-first search



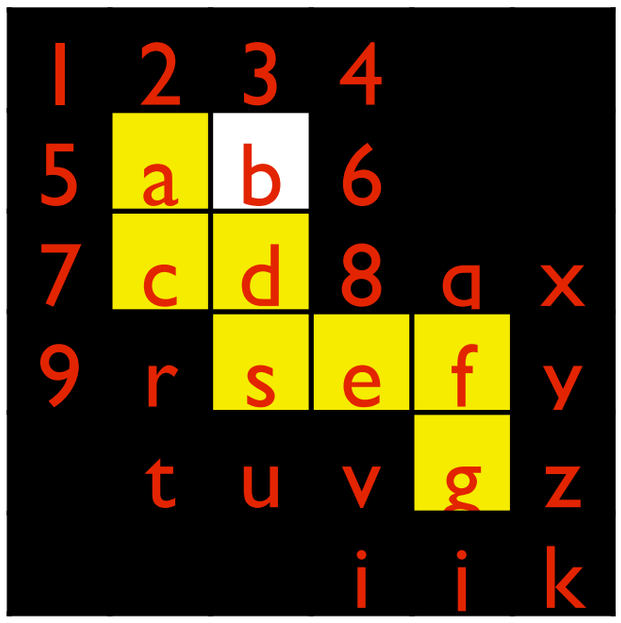
Finding a connected component using depth-first search (with a stack)

top of stack

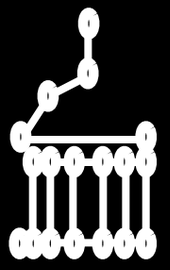


a

```
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
```

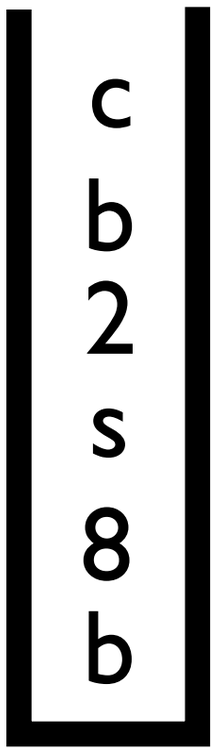


Depth-first search



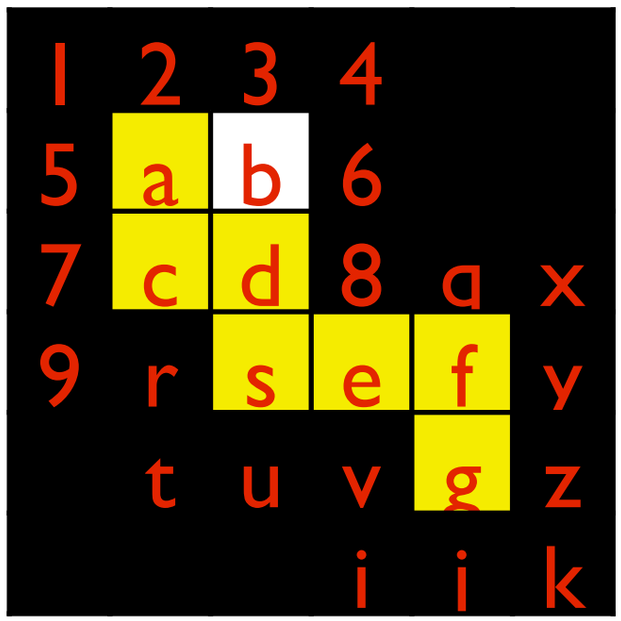
Finding a connected component using depth-first search (with a stack)

top
of stack



5

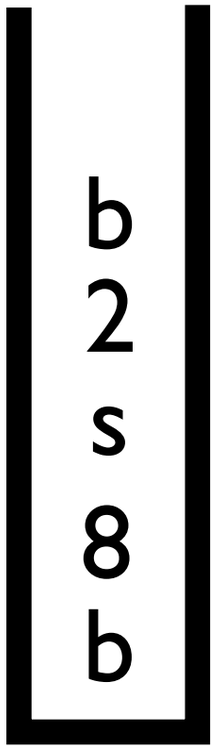
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack



Depth-first search

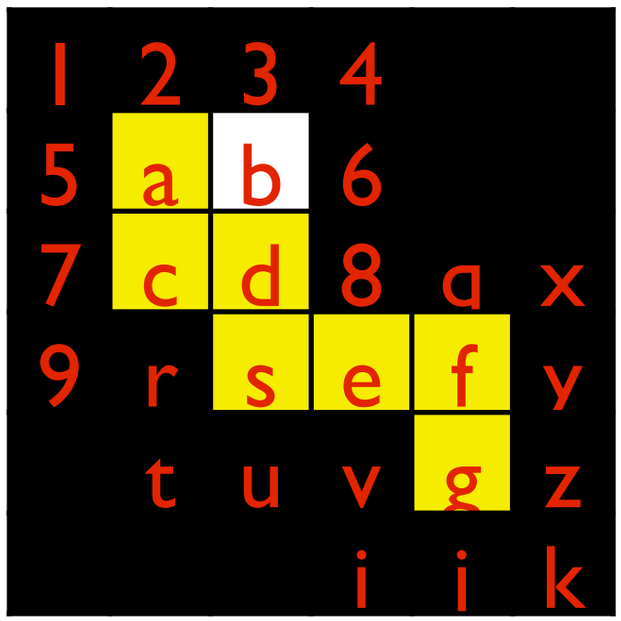
Finding a connected component using depth-first search (with a stack)

top
of stack

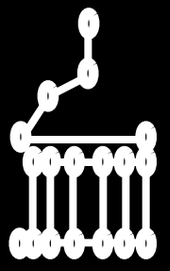


c

```
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
```

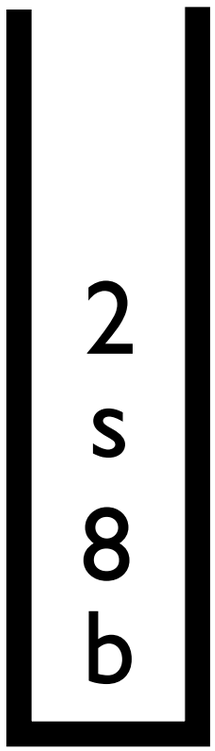


Depth-first search



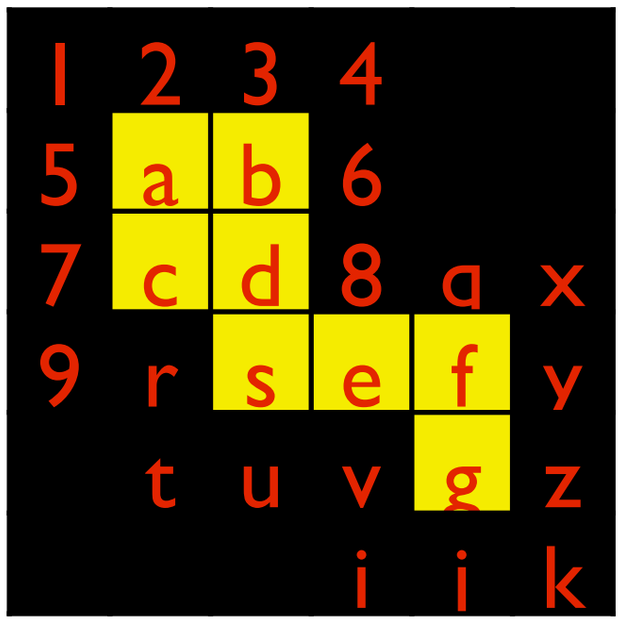
Finding a connected component using depth-first search (with a stack)

top
of stack



b

```
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
```



Depth-first search

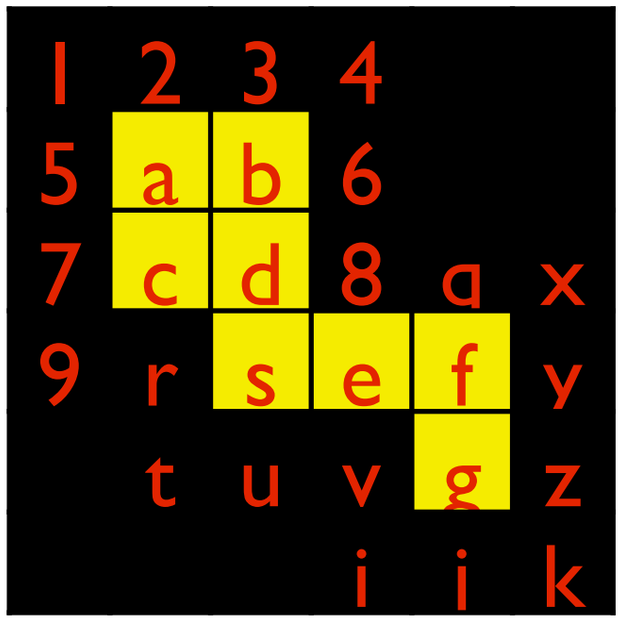
Finding a connected component using depth-first search (with a stack)

top of stack

6
3
2
s
8
b

b

```
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
```



Depth-first search

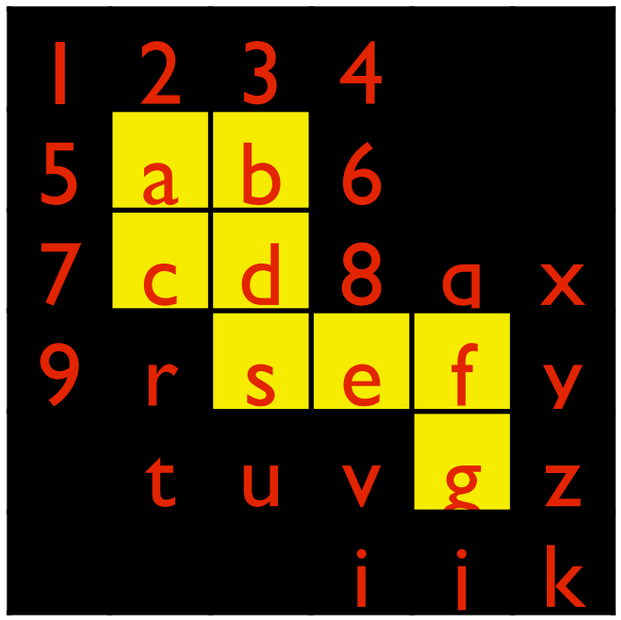
Finding a connected component using depth-first search (with a stack)

top of stack

d
6
3
2
s
8
b

a

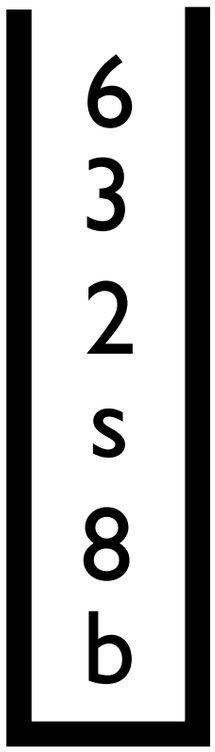
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

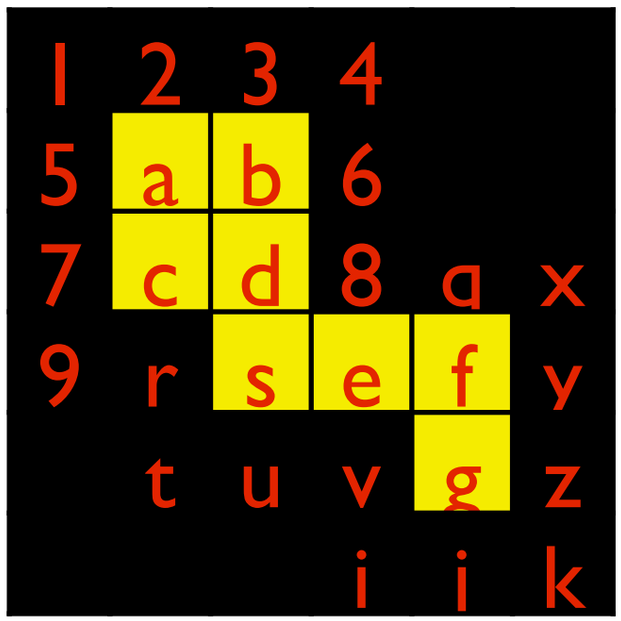
Finding a connected component using depth-first search (with a stack)

top
of stack



d

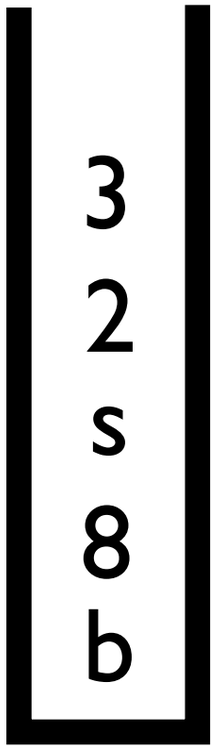
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

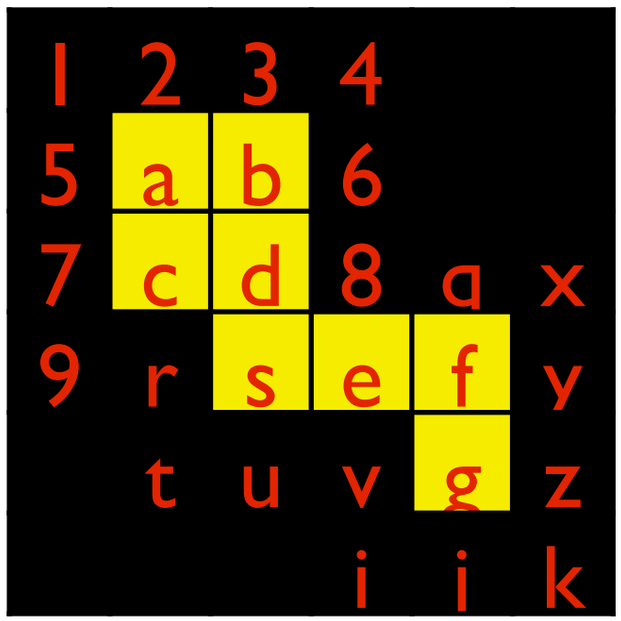
Finding a connected component using depth-first search (with a stack)

top of stack

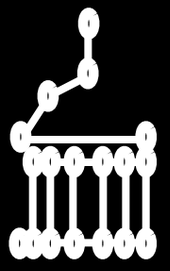


6

```
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
```

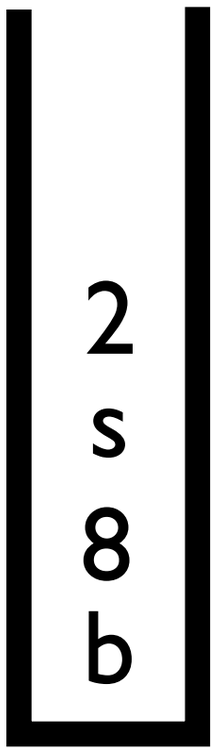


Depth-first search



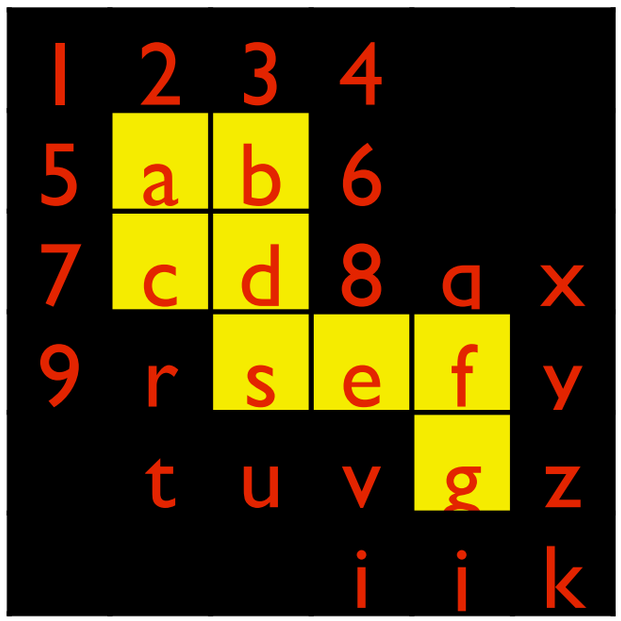
Finding a connected component using depth-first search (with a stack)

top
of stack

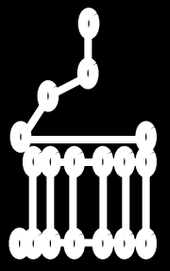


3

```
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
```

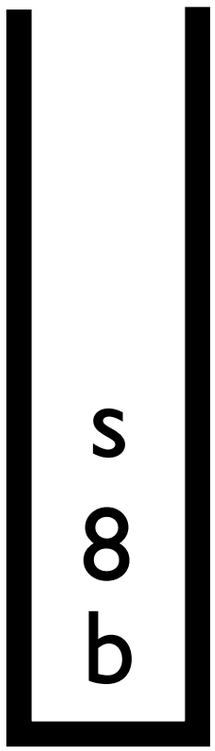


Depth-first search



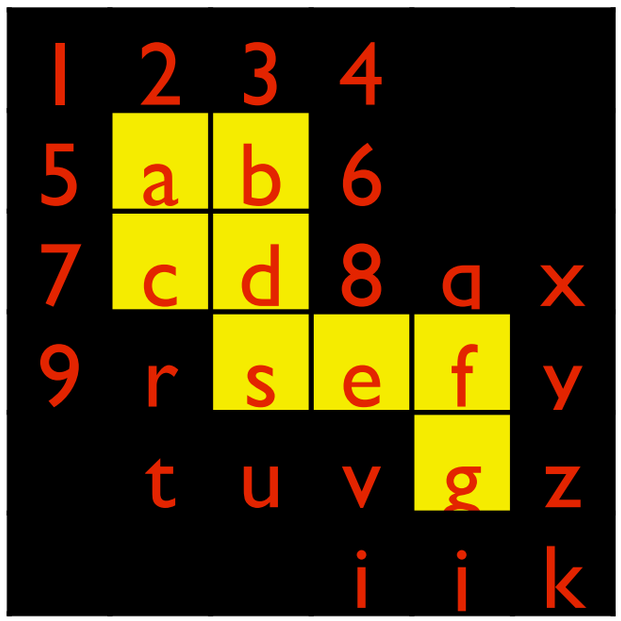
Finding a connected component using depth-first search (with a stack)

top
of stack



2

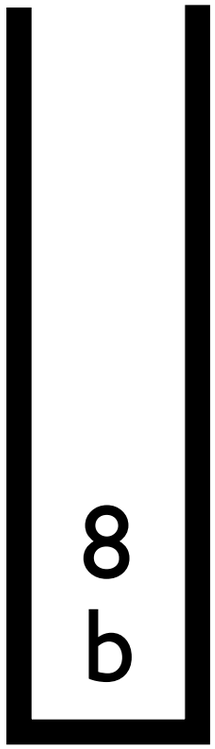
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

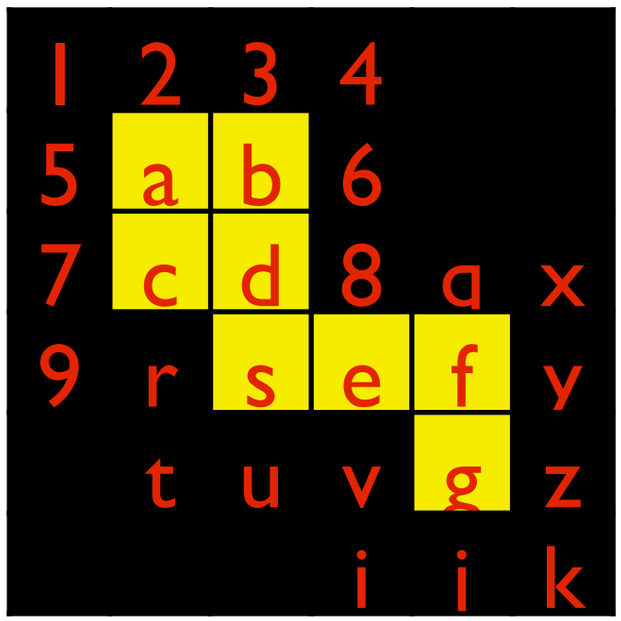
Finding a connected component using depth-first search (with a stack)

top
of stack



s

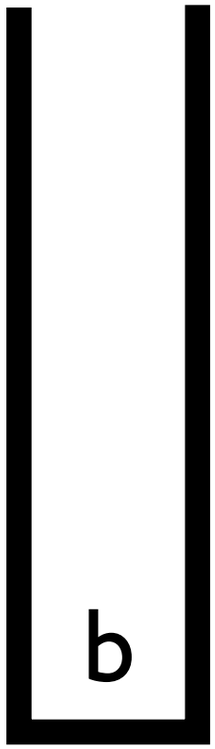
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

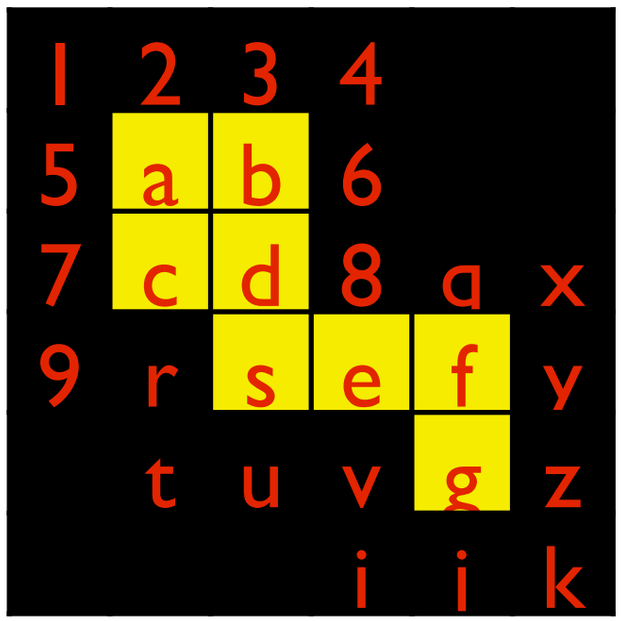
Finding a connected component using depth-first search (with a stack)

top
of stack



8

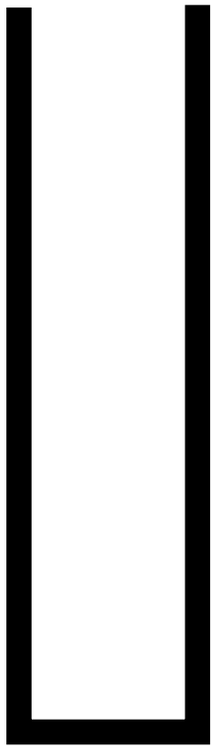
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```



Depth-first search

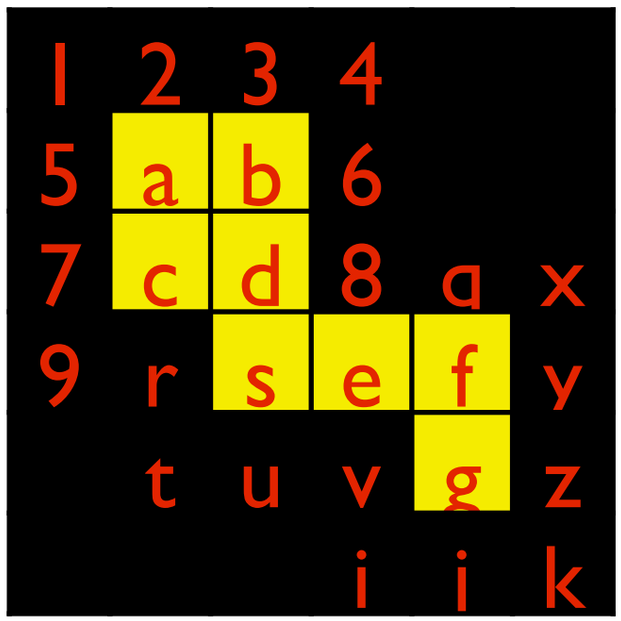
Finding a connected component using depth-first search (with a stack)

top
of stack



b

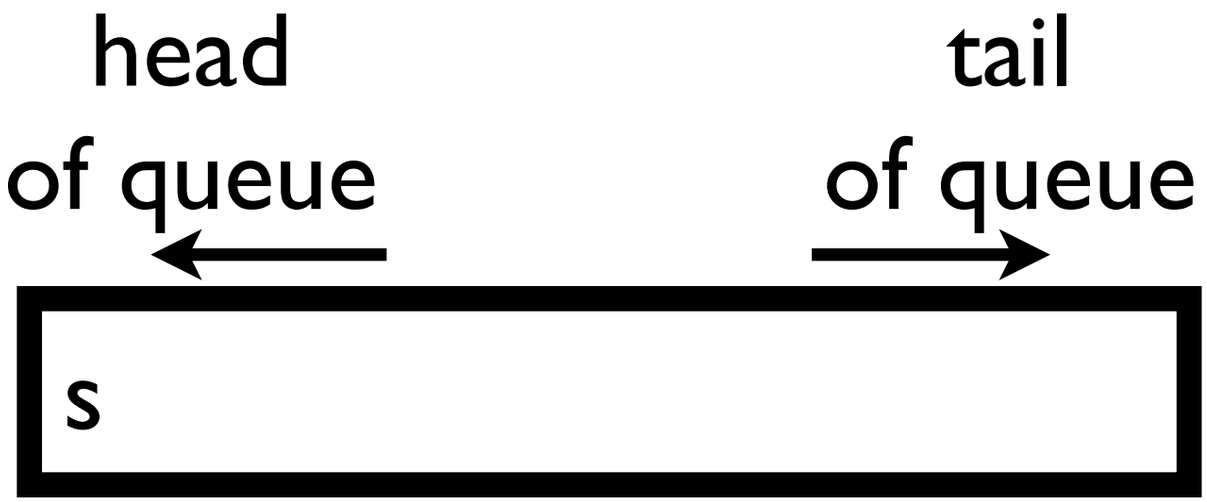
```
push seed node to stack
while (stack is not empty)
    pop node from stack
    if color is white:
        change color to yellow
        push all neighbors to stack
```

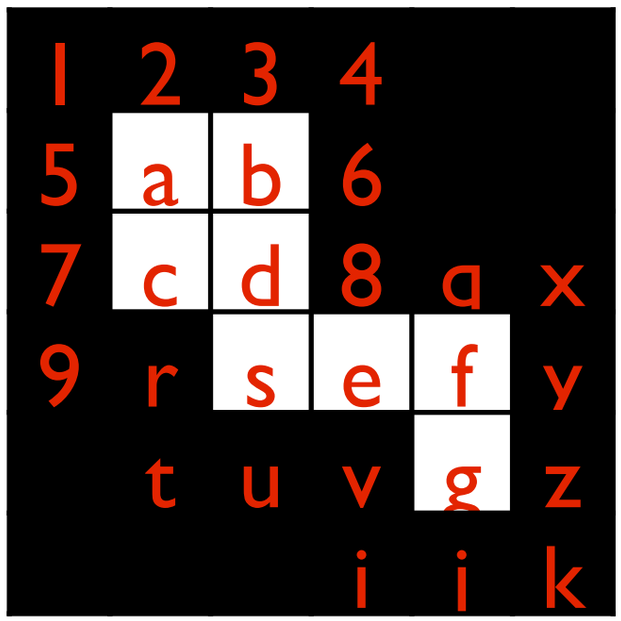


Depth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

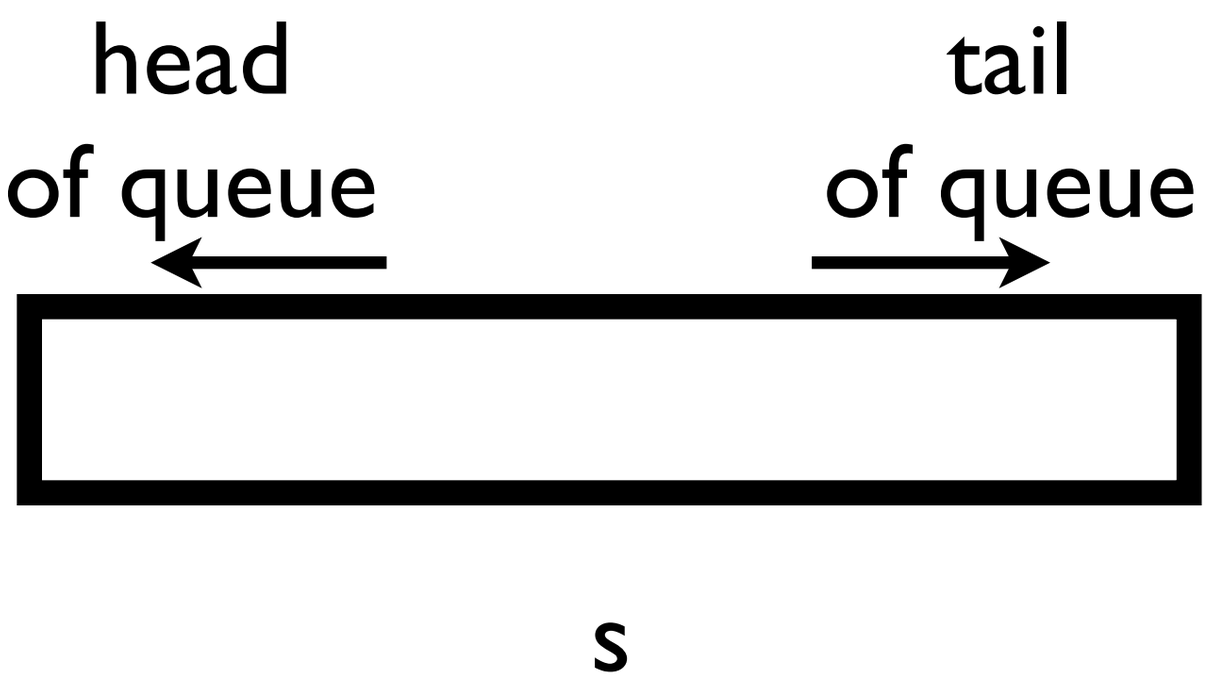


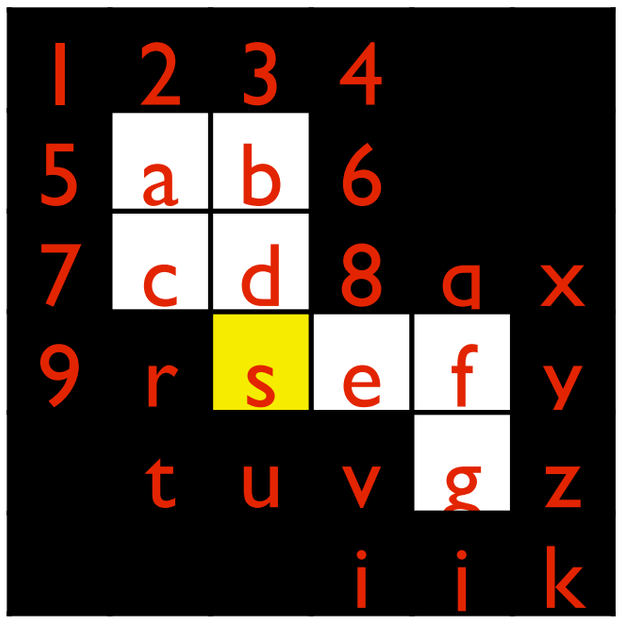


Breadth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```





Breadth-first search

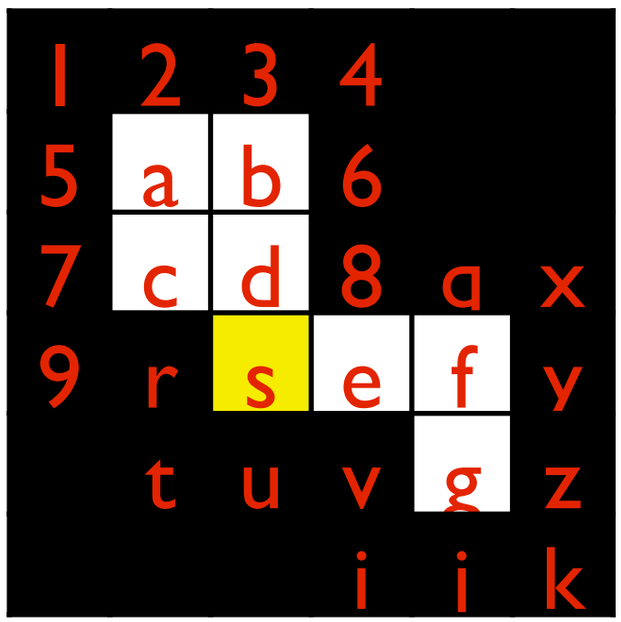
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

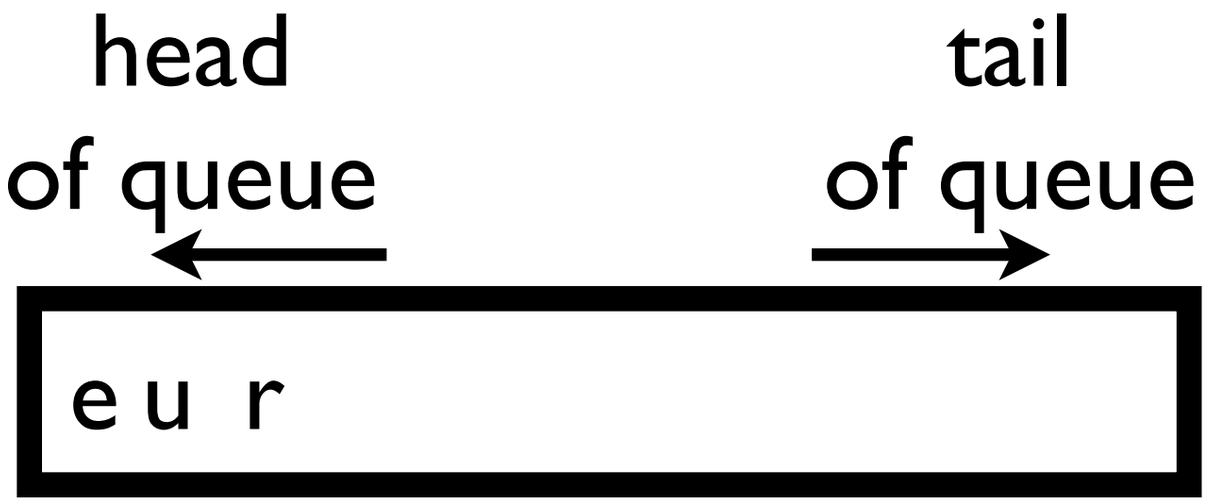
s



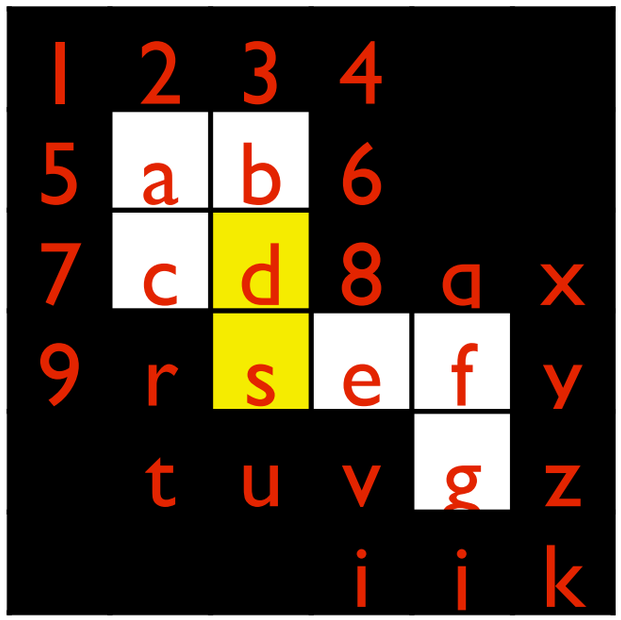
Breadth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```



d



Breadth-first search

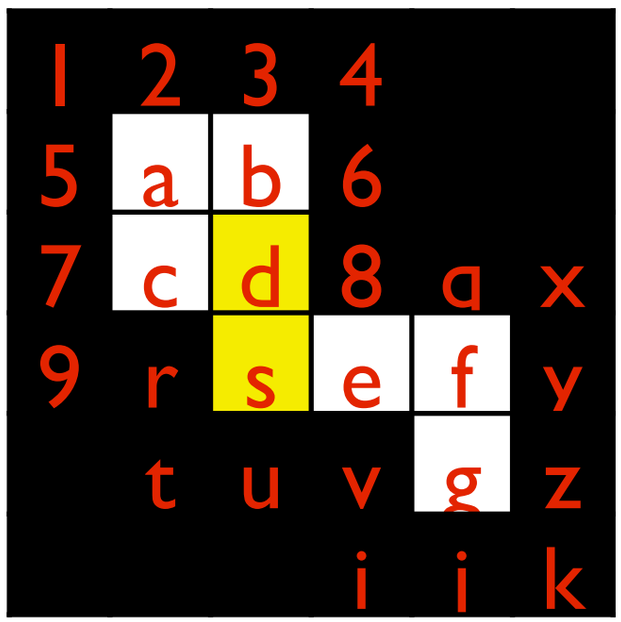
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

d



Breadth-first search

Finding a connected component using breadth-first search

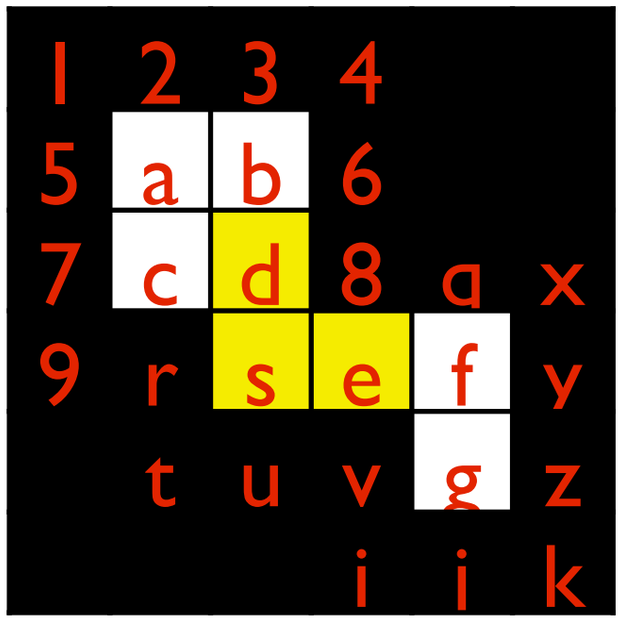
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

u	r	b	8	s	c

e



Breadth-first search

Finding a connected component using breadth-first search

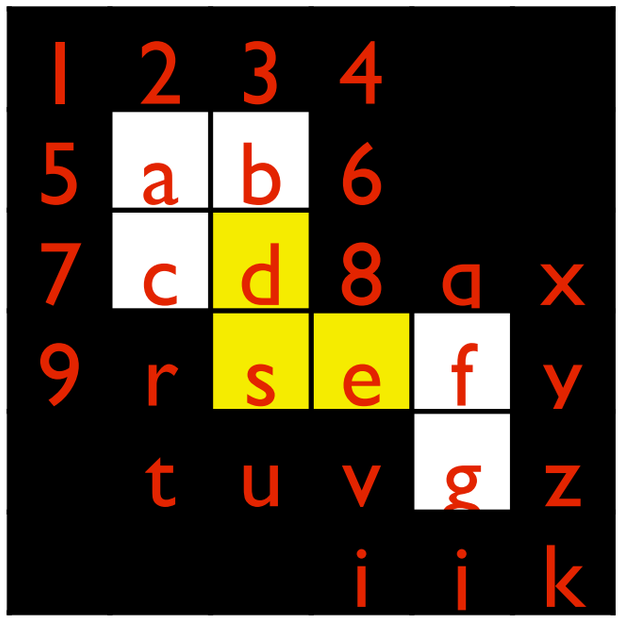
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

u	r	b	8	s	c	8	f	v	s

e



Breadth-first search

Finding a connected component using breadth-first search

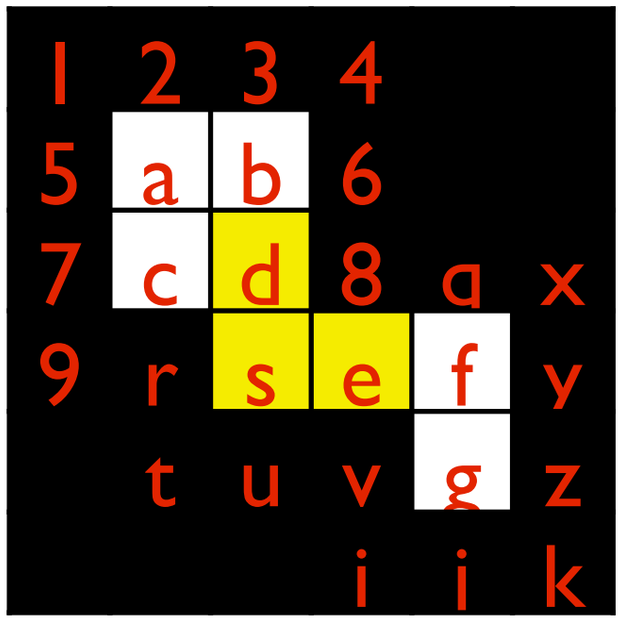
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

r b 8 s c 8 f v s

u



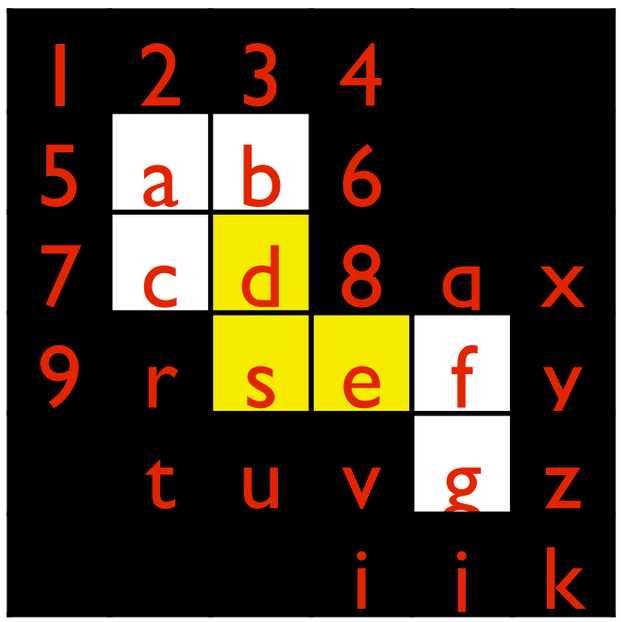
Breadth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →



Breadth-first search

Finding a connected component using breadth-first search

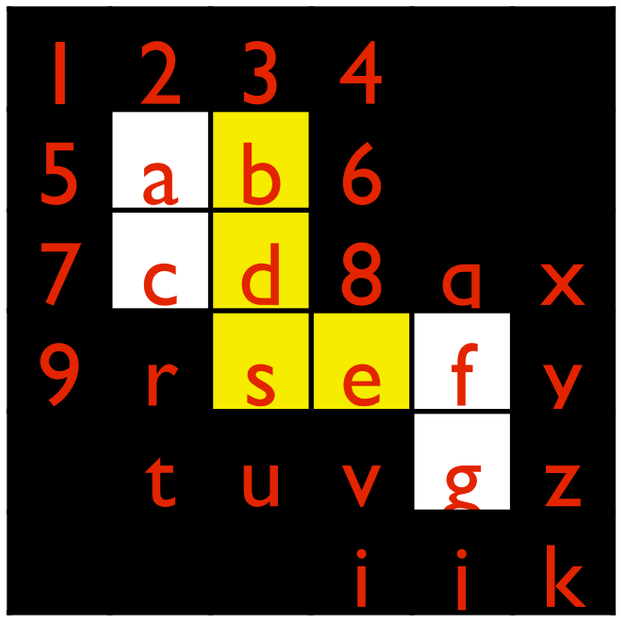
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

8	s	c	8	f	v	s

b



Breadth-first search

Finding a connected component using breadth-first search

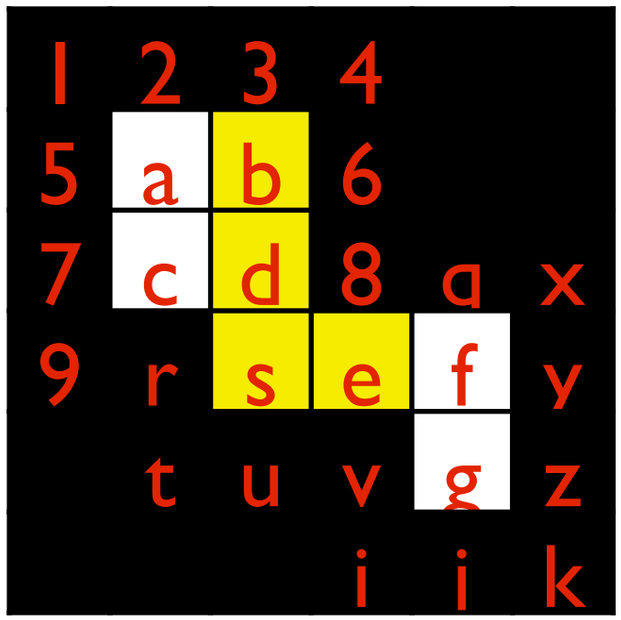
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

8	s	c	8	f	v	s	3	6	d	a

b



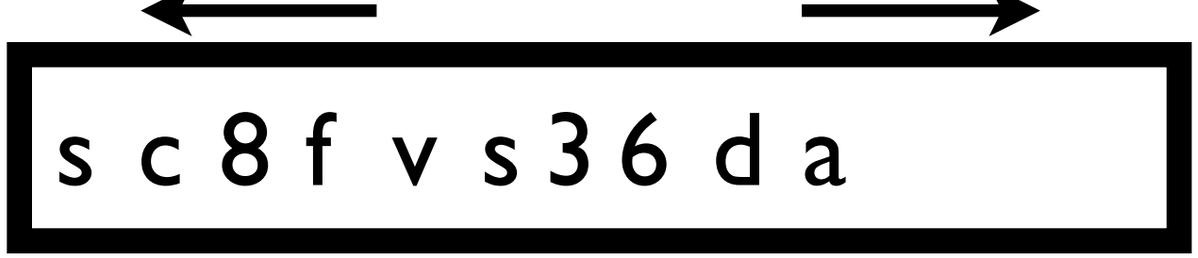
Breadth-first search

Finding a connected component using breadth-first search

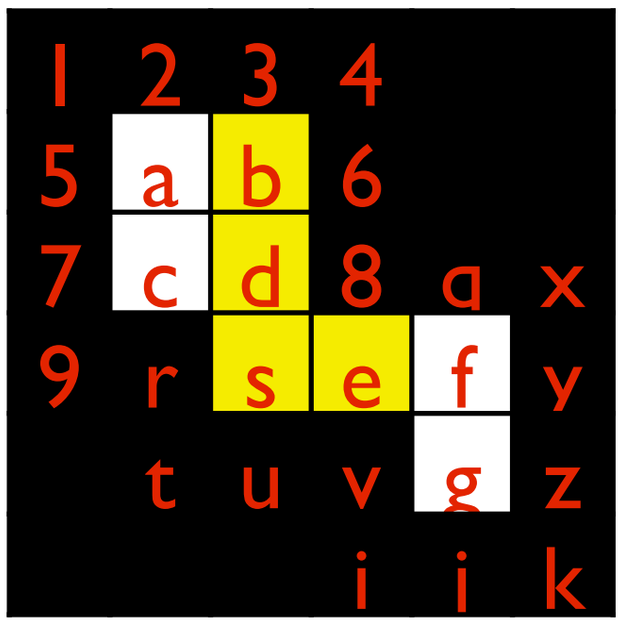
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →



8



Breadth-first search

Finding a connected component using breadth-first search

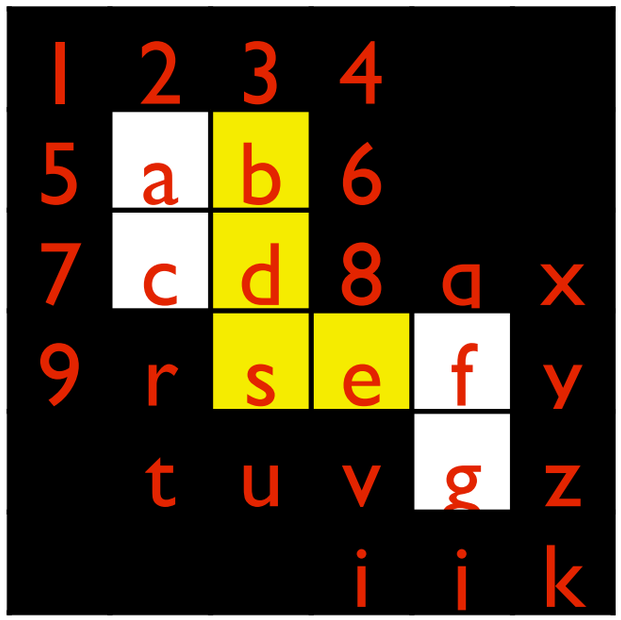
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

c	8	f	v	s	3	6	d	a

s



Breadth-first search

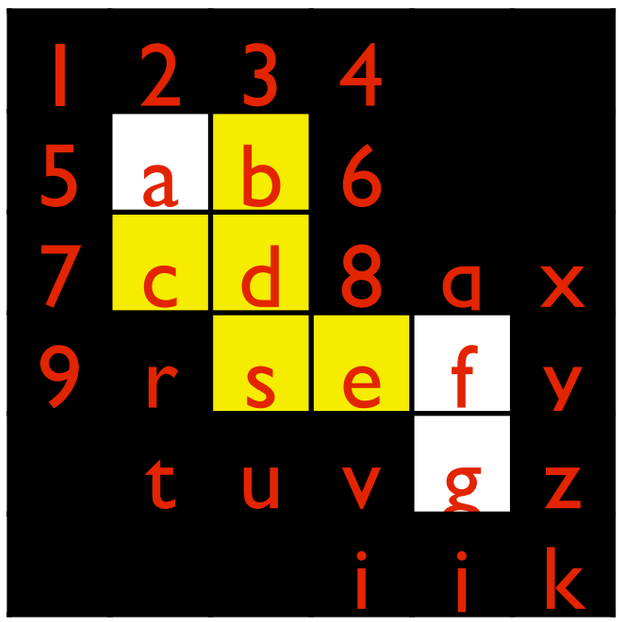
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

c



Breadth-first search

Finding a connected component using breadth-first search

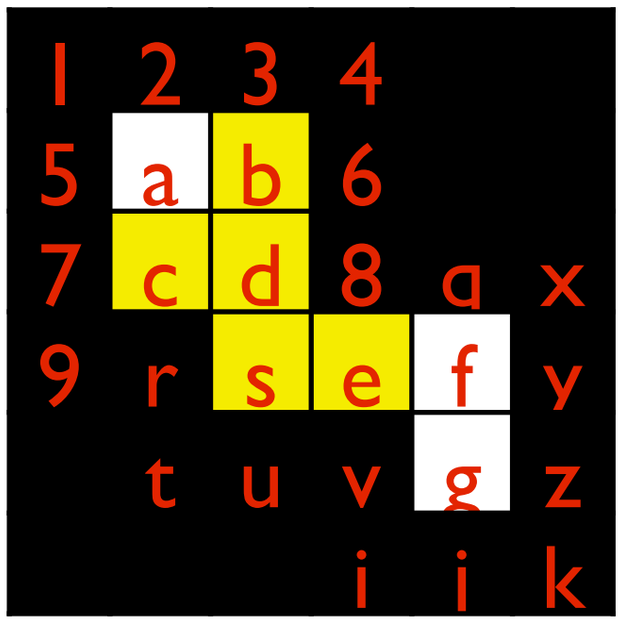
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

8	f	v	s	3	6	d	a	a	d	r	7

c



Breadth-first search

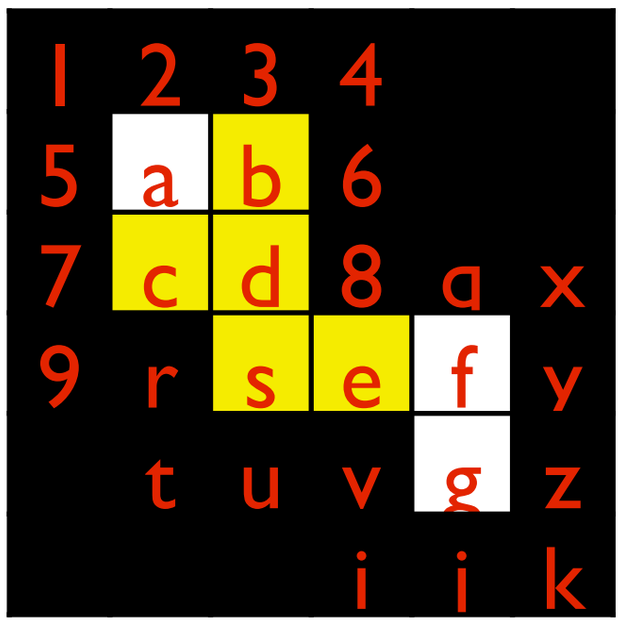
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

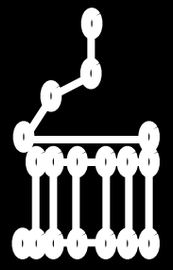
head of queue ←

tail of queue →

8



Breadth-first search



Finding a connected component using breadth-first search

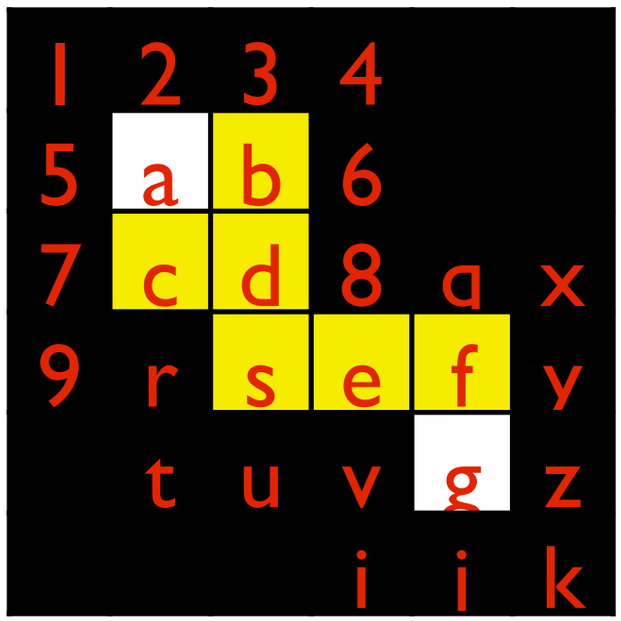
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

v	s	3	6	d	a	a	d	r	7

f



Breadth-first search

Finding a connected component using breadth-first search

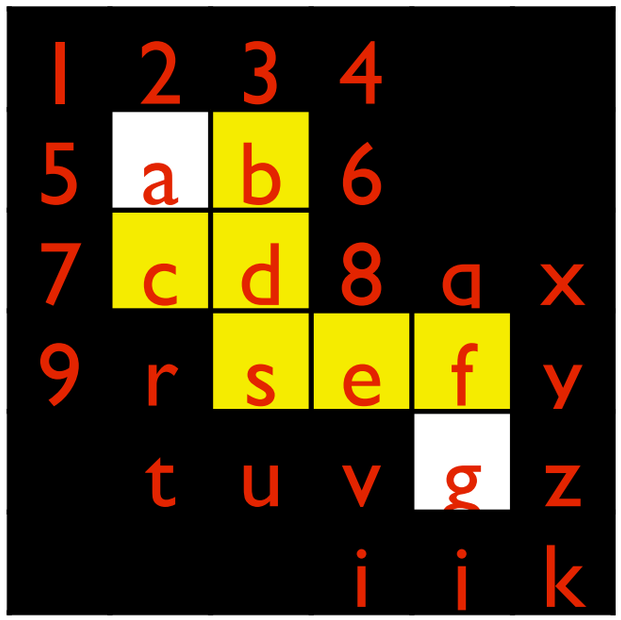
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

v	s	3	6	d	a	a	d	r	7	q	y	g	e

f



Breadth-first search

Finding a connected component using breadth-first search

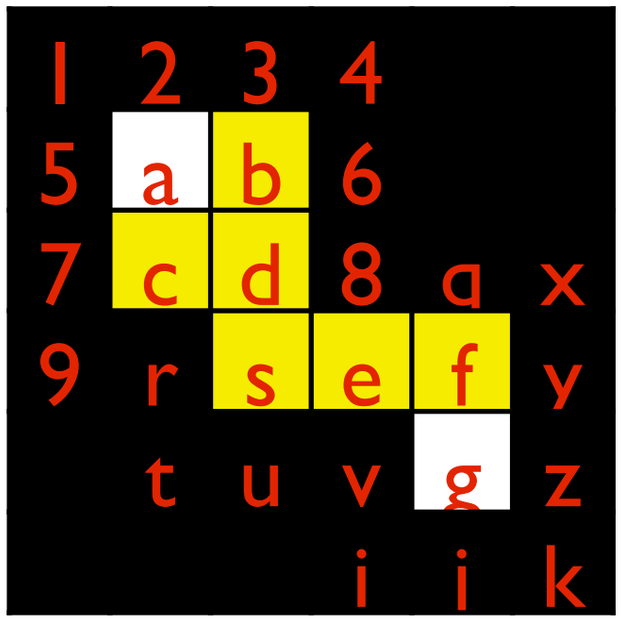
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

s	3	6	d	a	a	d	r	7	q	y	g	e

v



Breadth-first search

Finding a connected component using breadth-first search

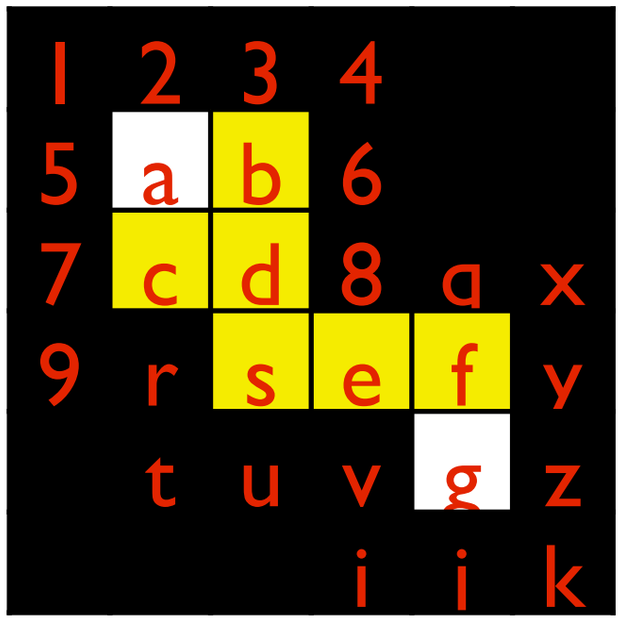
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

3	6	d	a	a	d	r	7	q	y	g	e

s



Breadth-first search

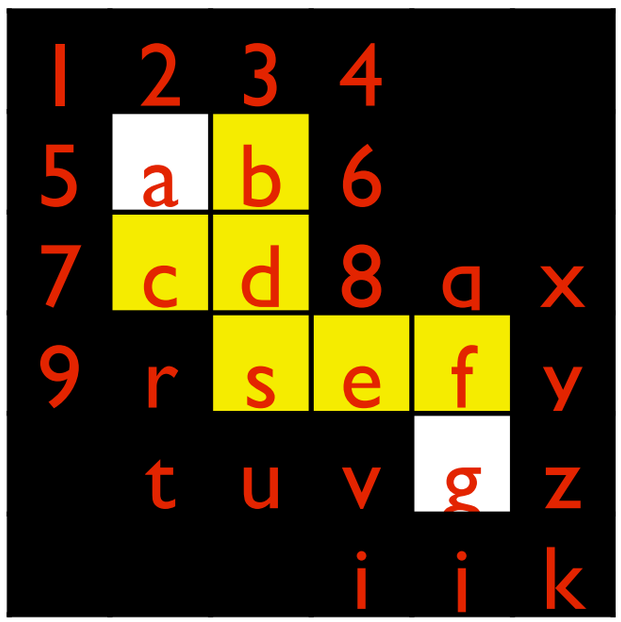
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

3



Breadth-first search

Finding a connected component using breadth-first search

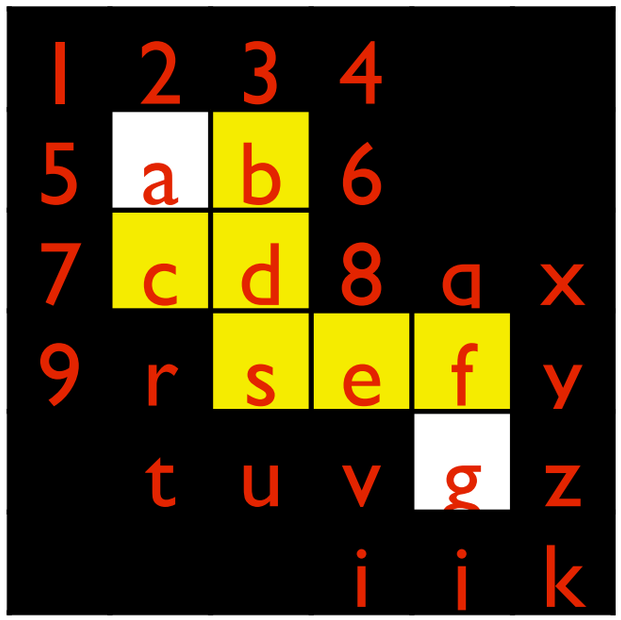
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

d	a	a	d	r	7	q	y	g	e

6



Breadth-first search

Finding a connected component using breadth-first search

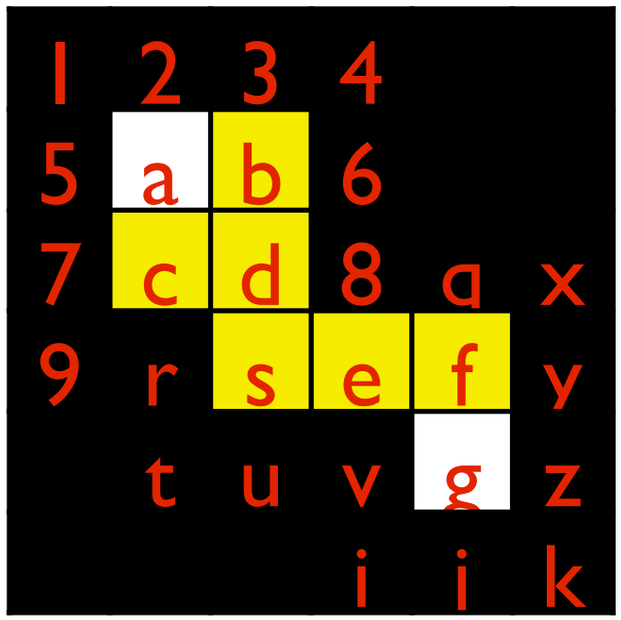
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

a	a	d	r	7	q	y	g	e

d



Breadth-first search

Finding a connected component using breadth-first search

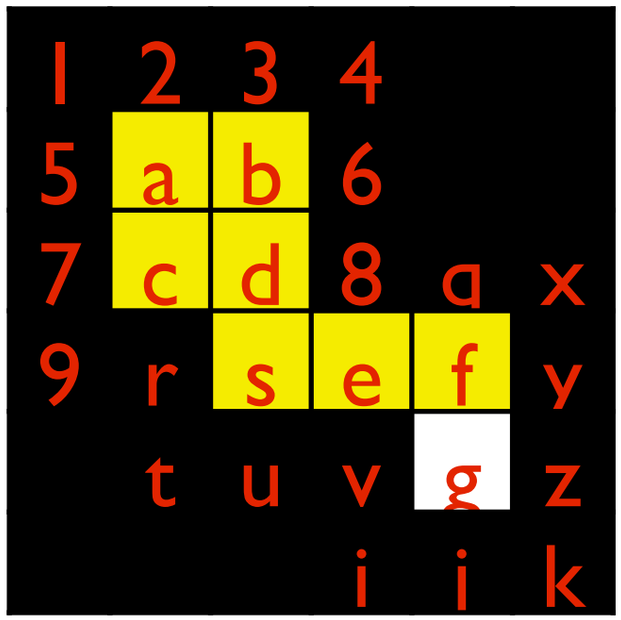
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

a	d	r	7	q	y	g	e

a



Breadth-first search

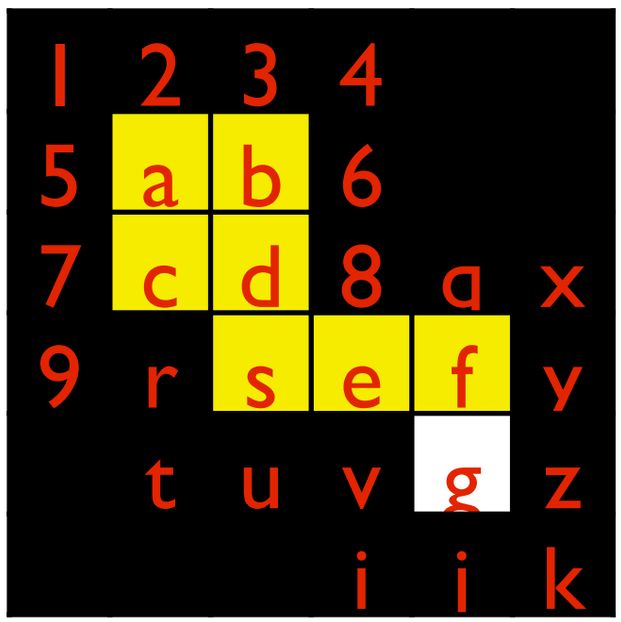
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

a



Breadth-first search

Finding a connected component using breadth-first search

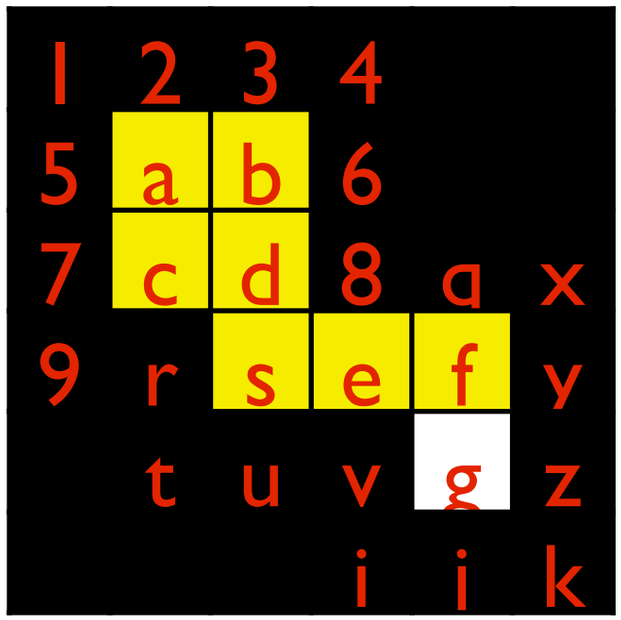
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

d	r	7	q	y	g	e	2	b	c	5

a



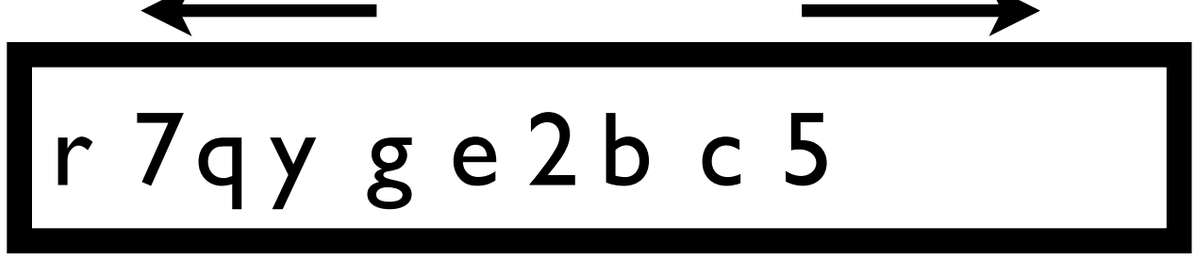
Breadth-first search

Finding a connected component using breadth-first search

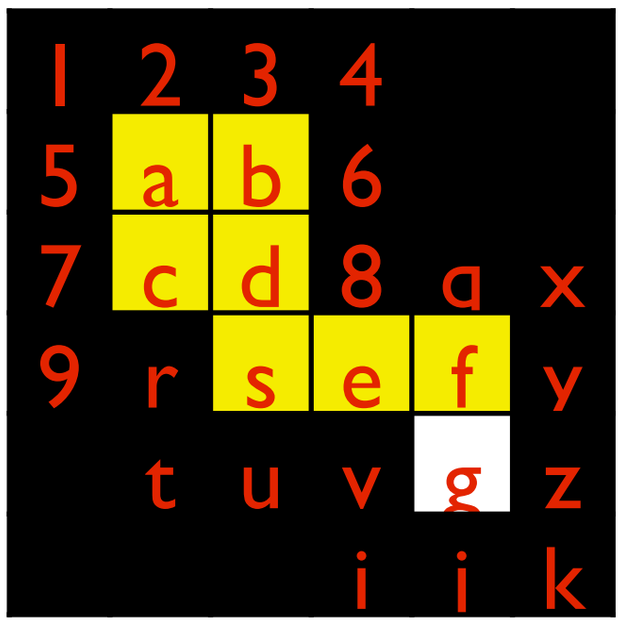
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →



d



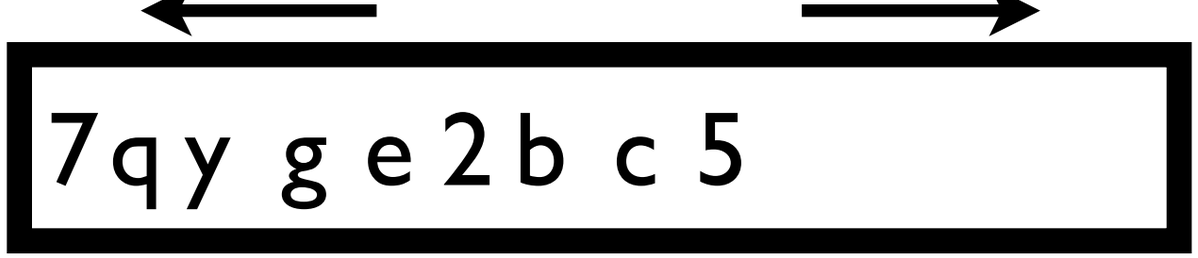
Breadth-first search

Finding a connected component using breadth-first search

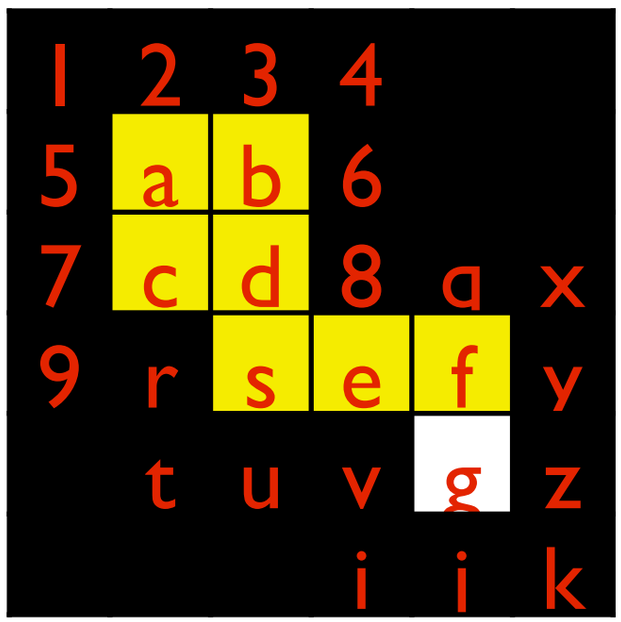
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →



r



Breadth-first search

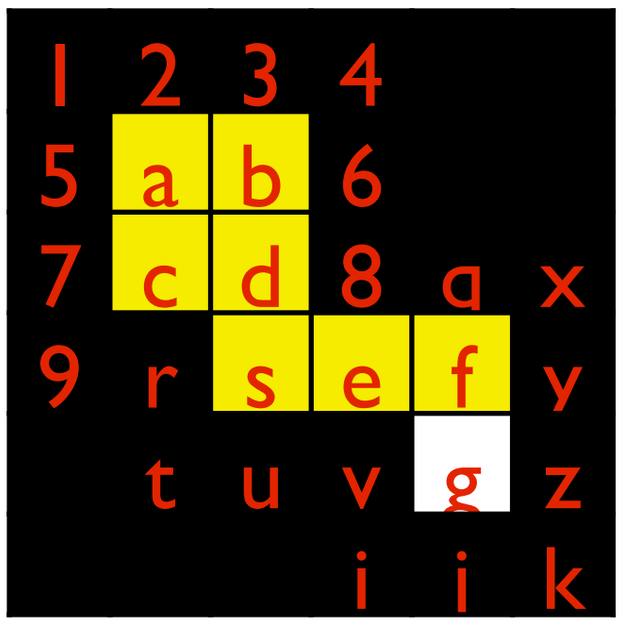
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

7



Breadth-first search

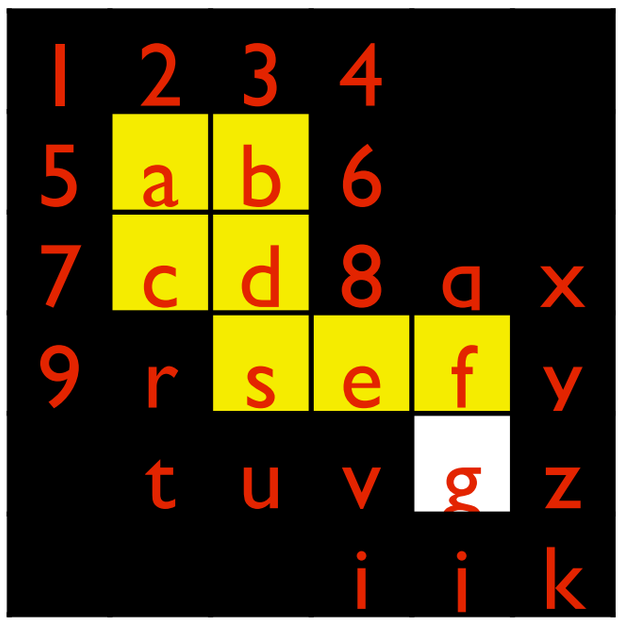
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

q



Breadth-first search

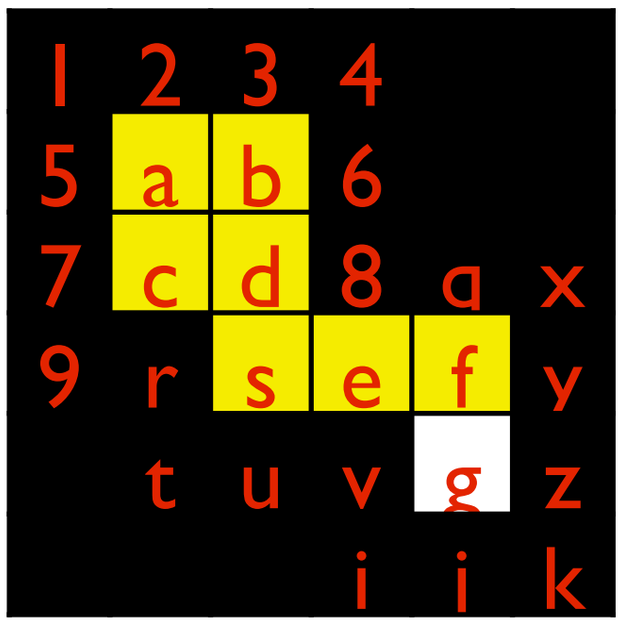
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

y



Breadth-first search

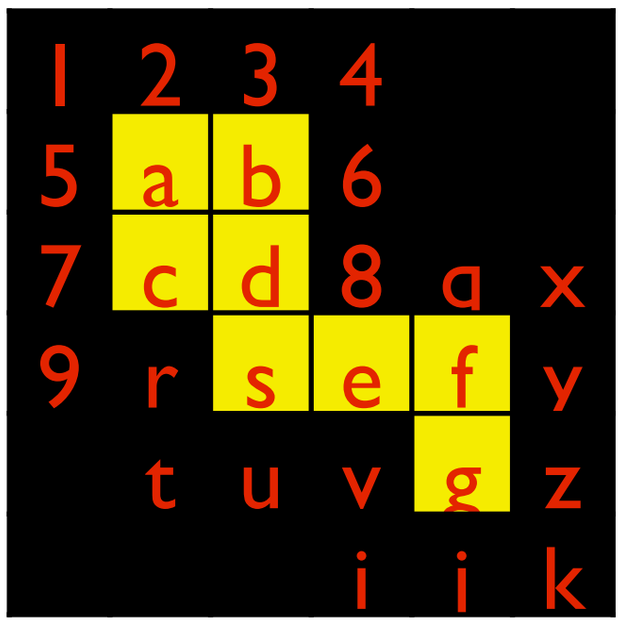
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

g



Breadth-first search

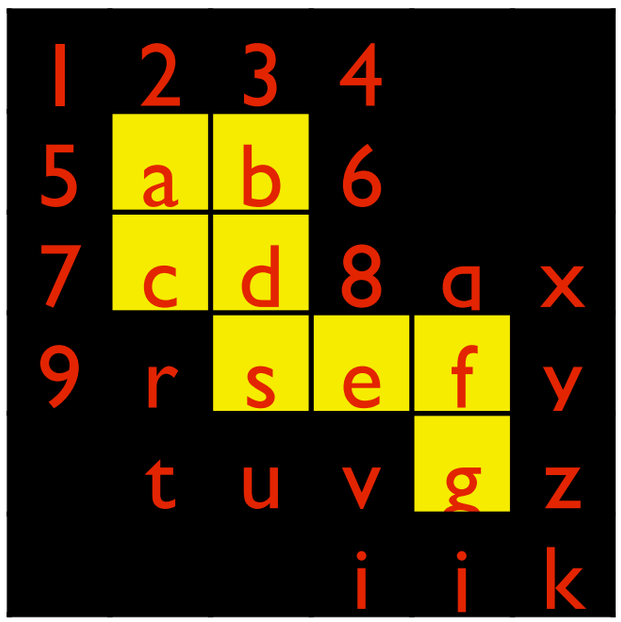
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

g



Breadth-first search

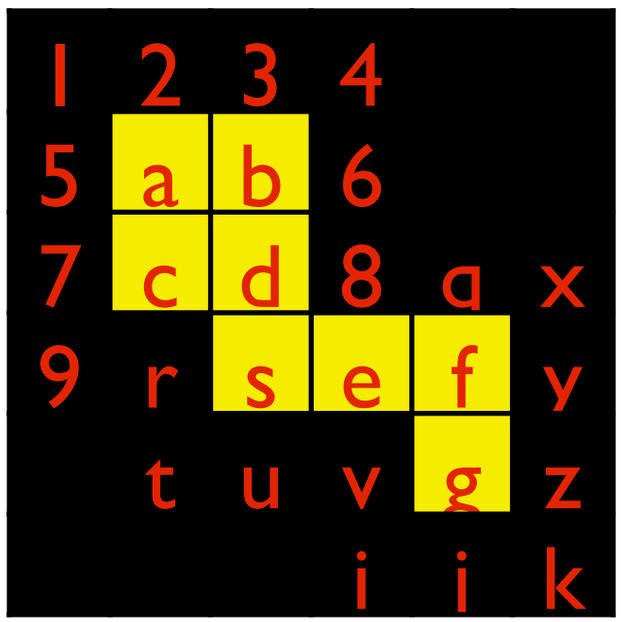
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

e



Breadth-first search

Finding a connected component using breadth-first search

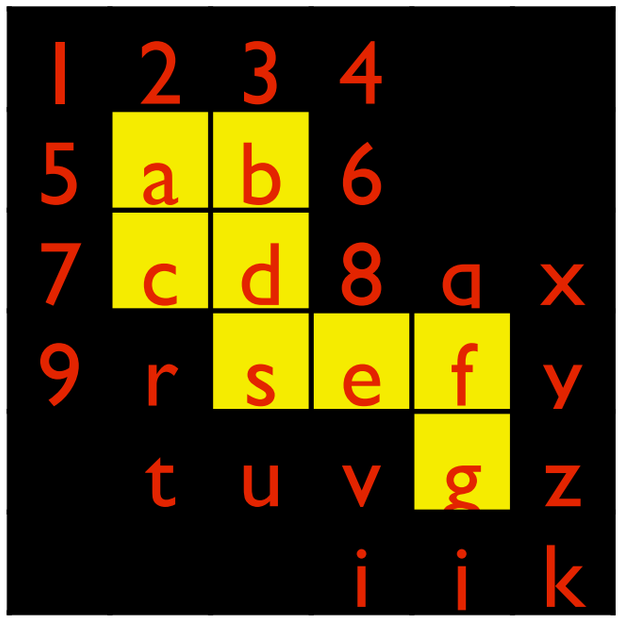
```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

b	c	5	f	z	j	v

2



Breadth-first search

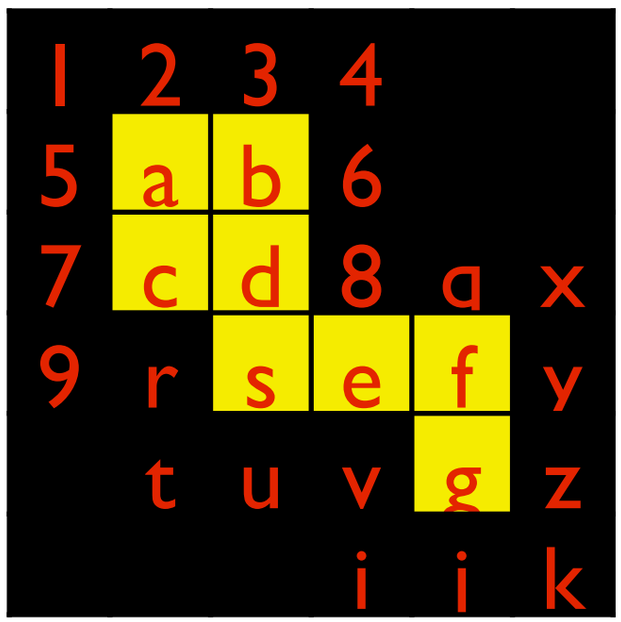
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

b



Breadth-first search

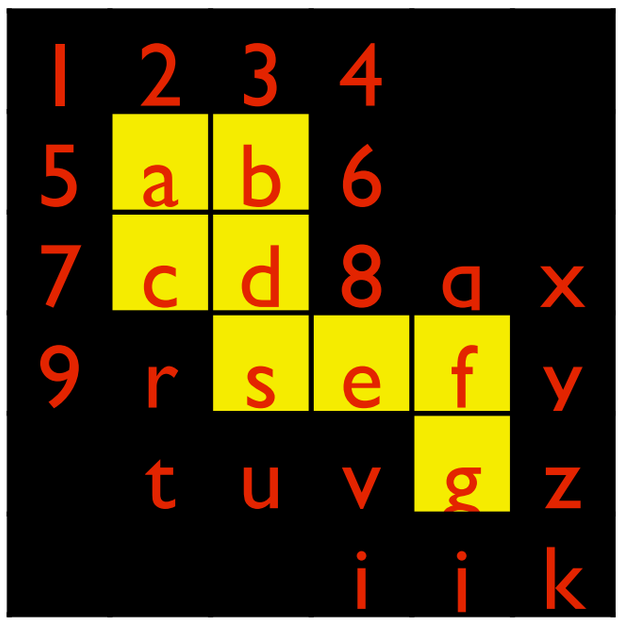
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

c



Breadth-first search

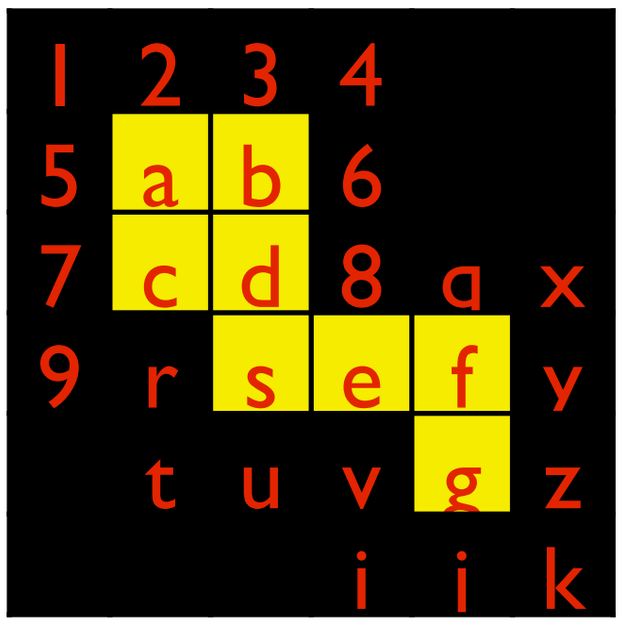
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

5



Breadth-first search

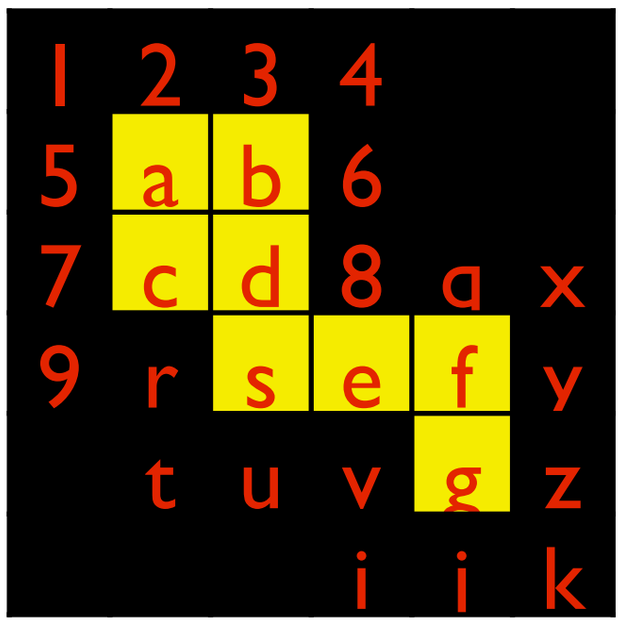
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

tail of queue →

f



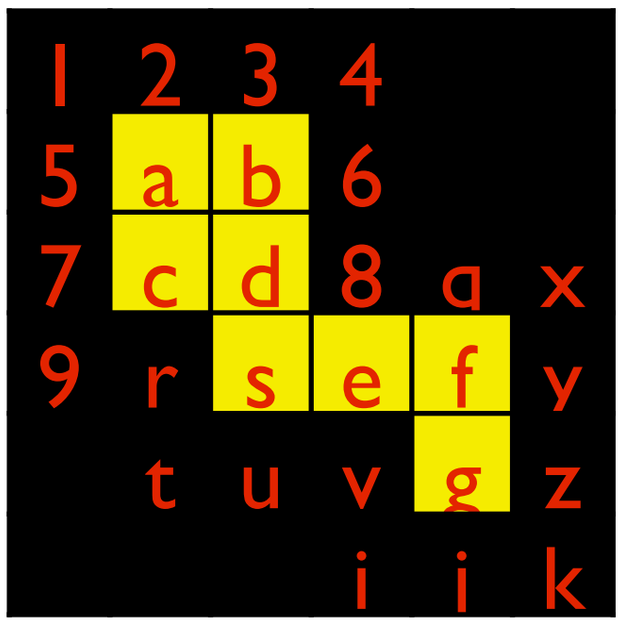
Breadth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

head of queue ←

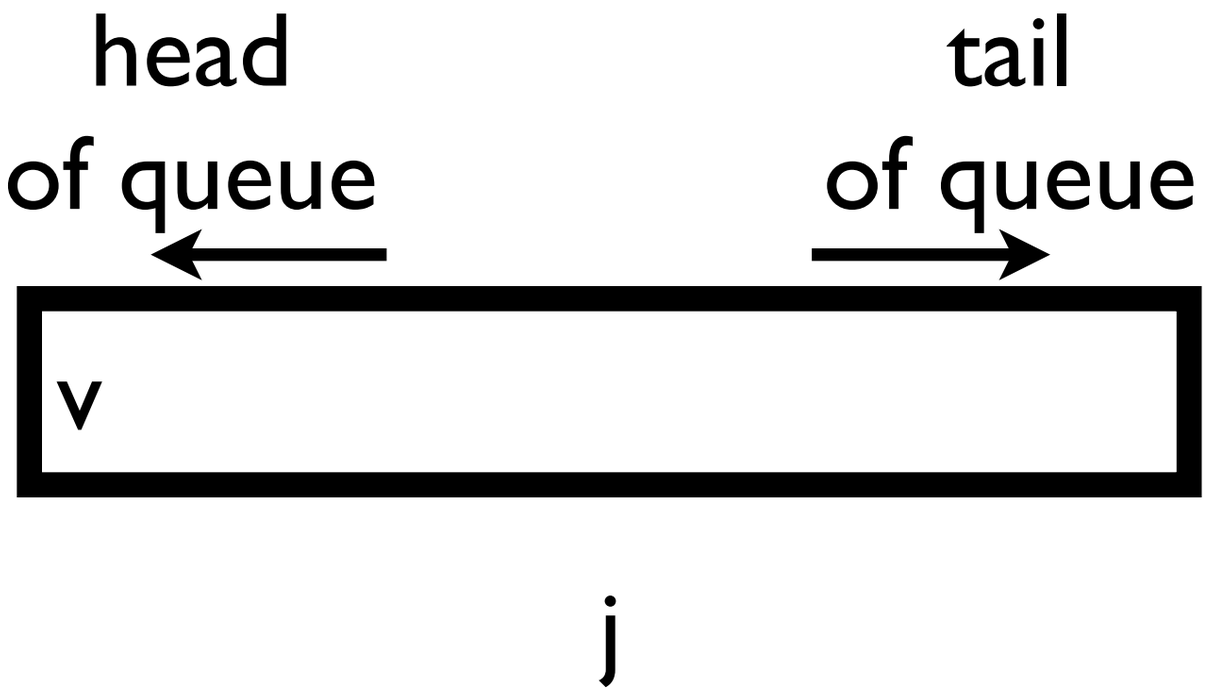
tail of queue →

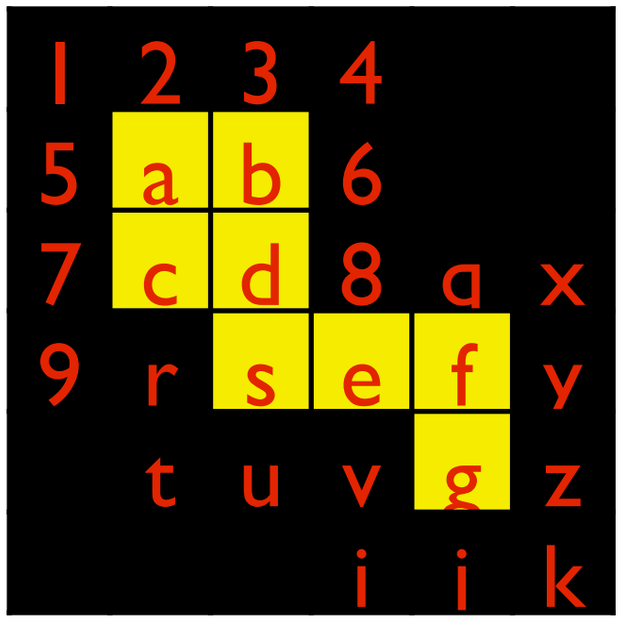


Breadth-first search

Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```





Breadth-first search

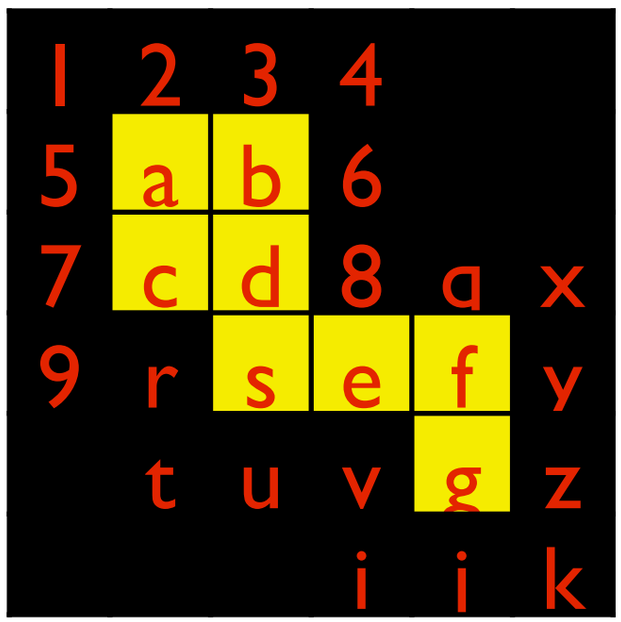
Finding a connected component using breadth-first search

```
enqueue seed node to queue
while (queue is not empty)
    dequeue node from queue
    if color is white:
        change color to yellow
        enqueue all neighbors to queue
```

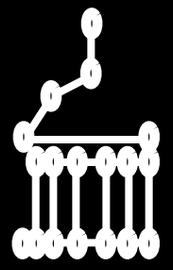
head of queue ←

tail of queue →

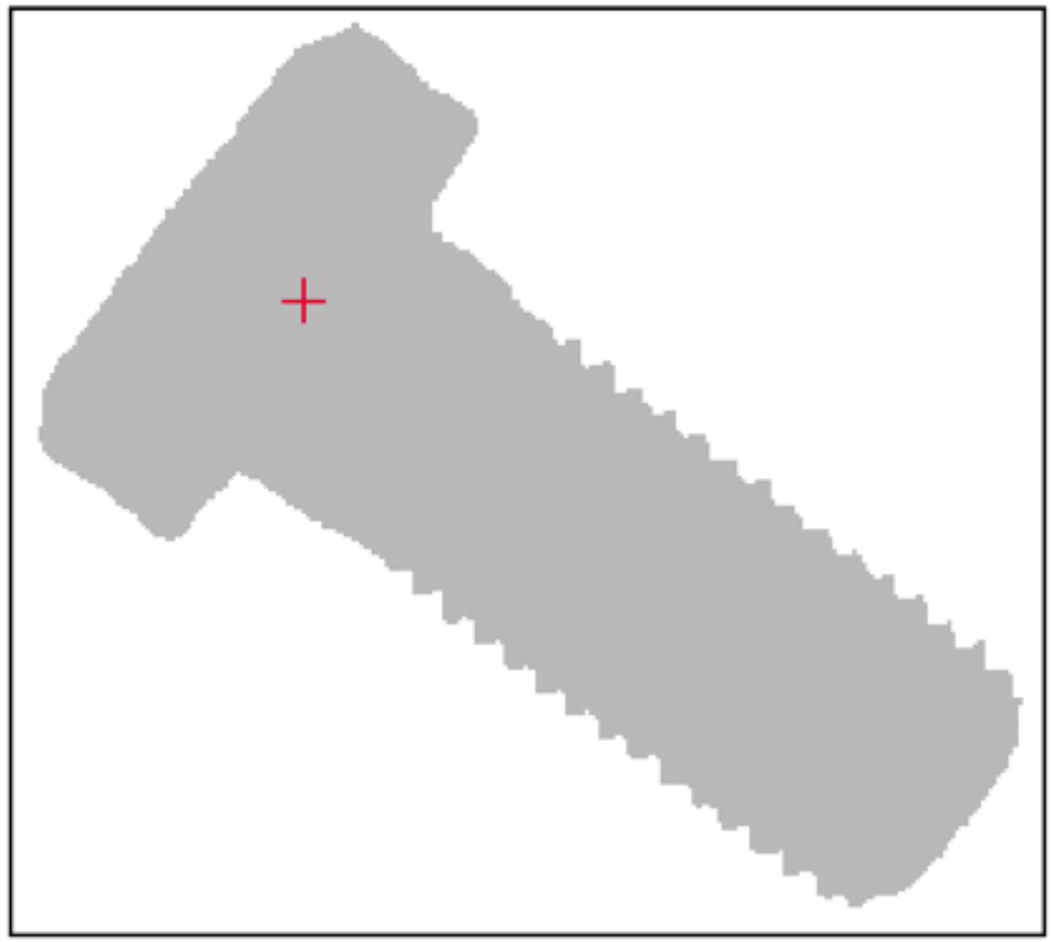
v



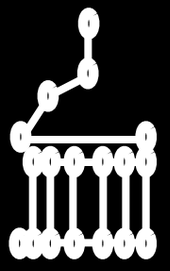
Breadth-first search



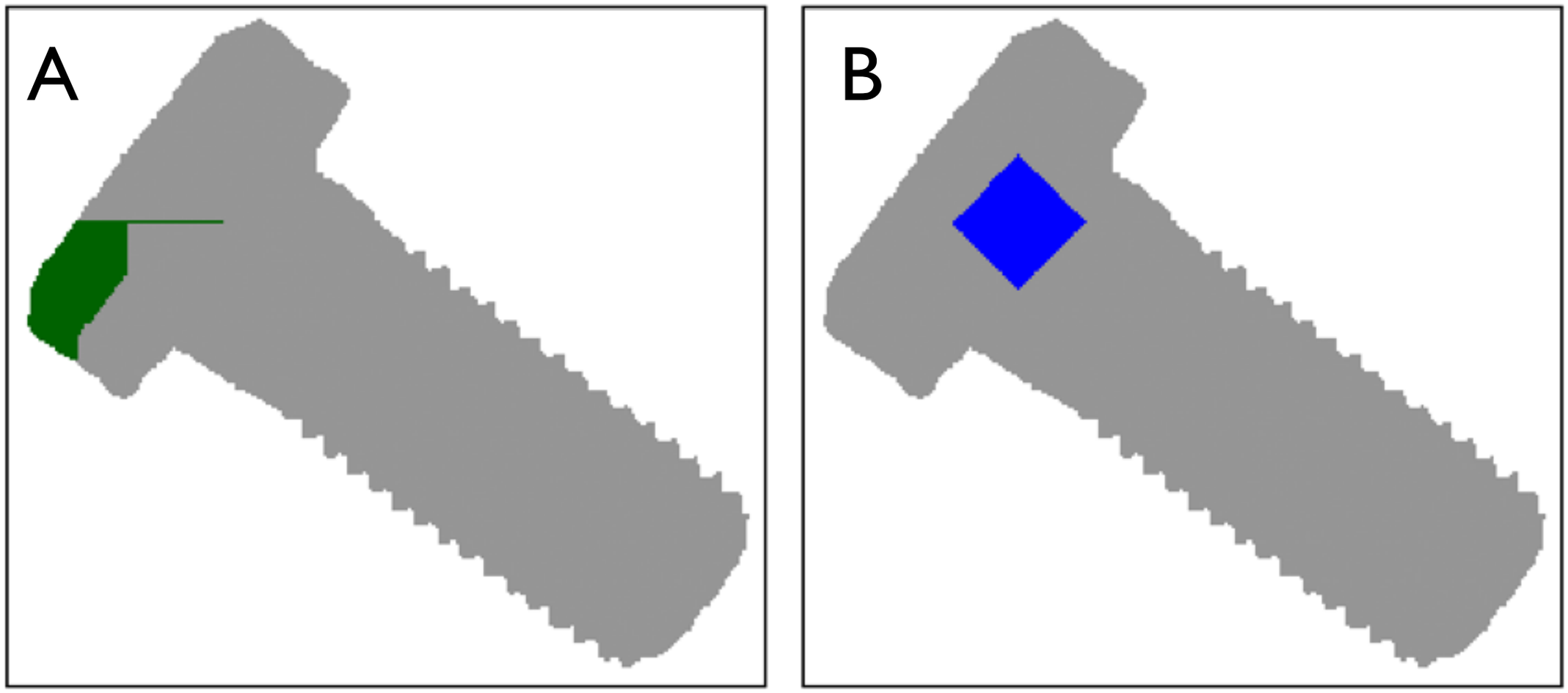
Example with starting seed



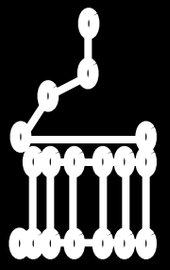
original image with seed



CQ: Which intermediate image below do you think reflects using depth-first search with a stack?



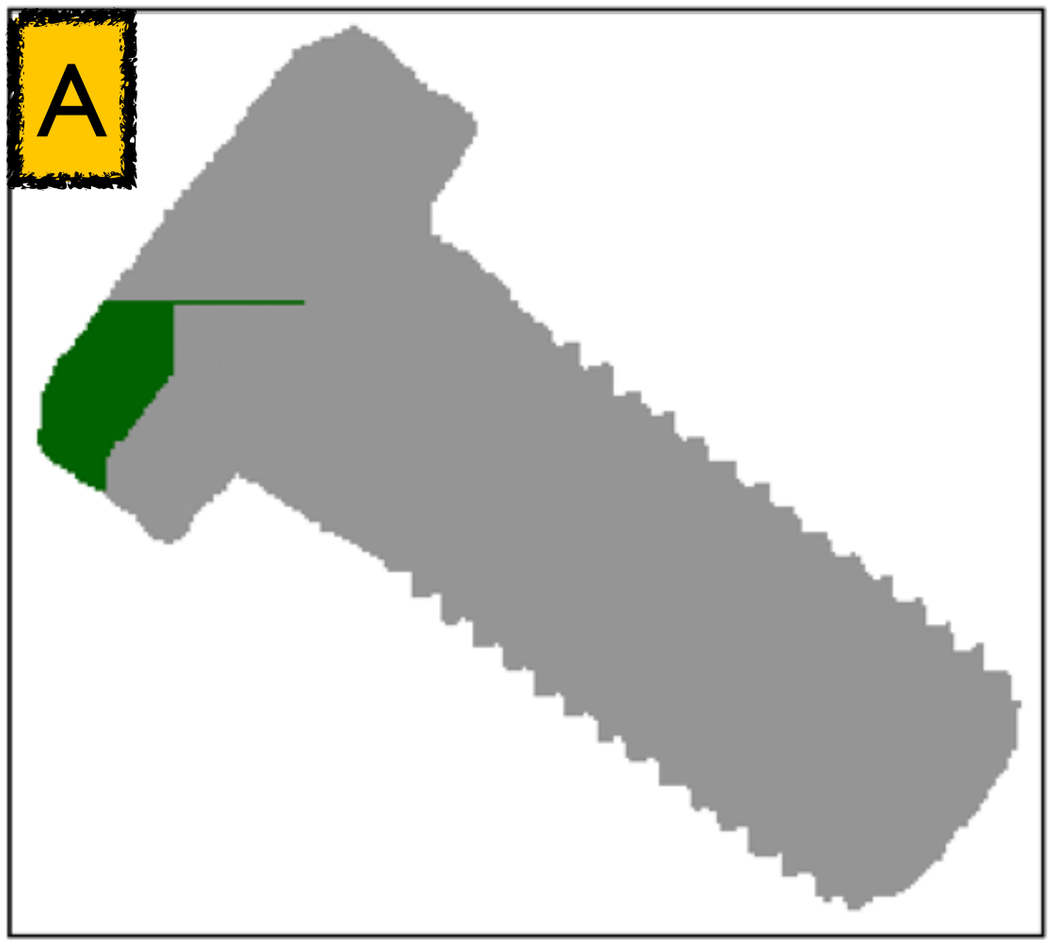
K = 1000 marked pixels

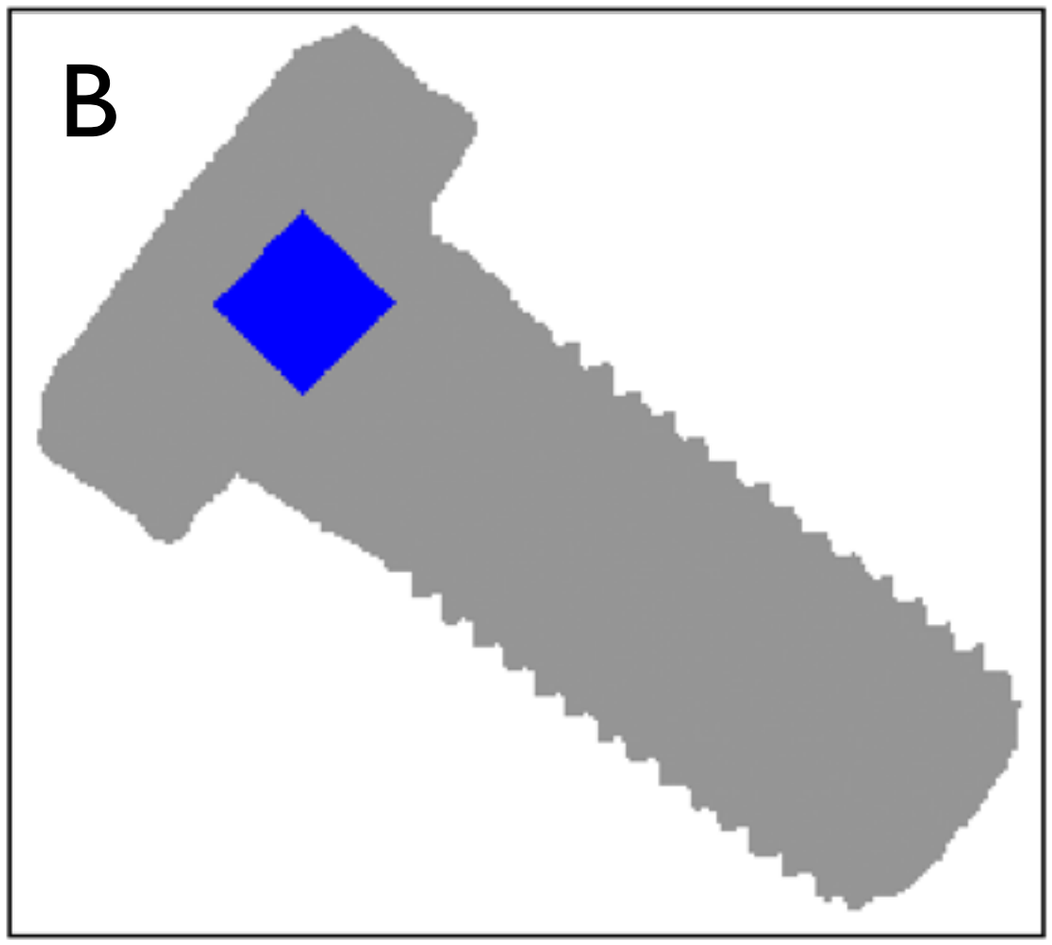


CQ: Which intermediate image below do you think reflects using depth-first search with a stack?

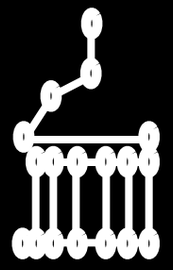
depth-first

breadth-first



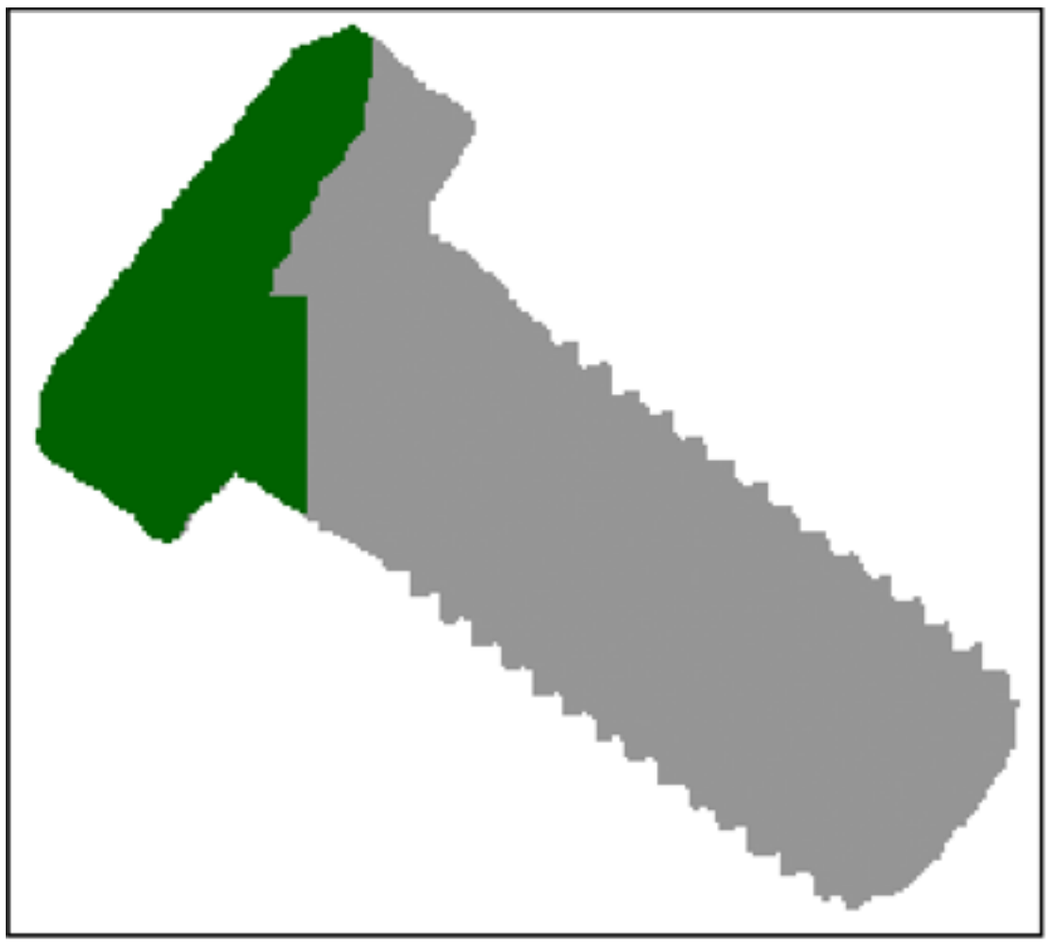


K = 1000 marked pixels

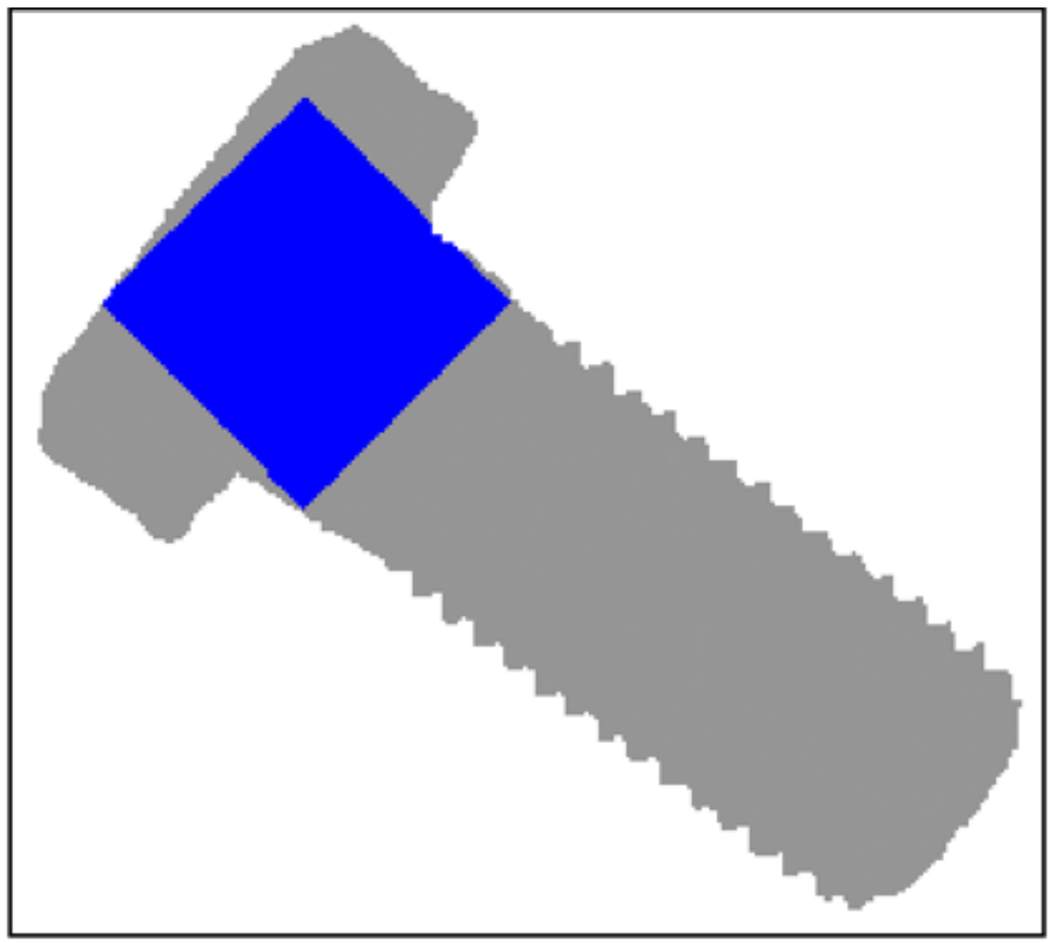


More iterations...

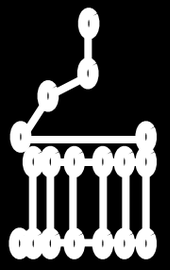
depth-first



breadth-first

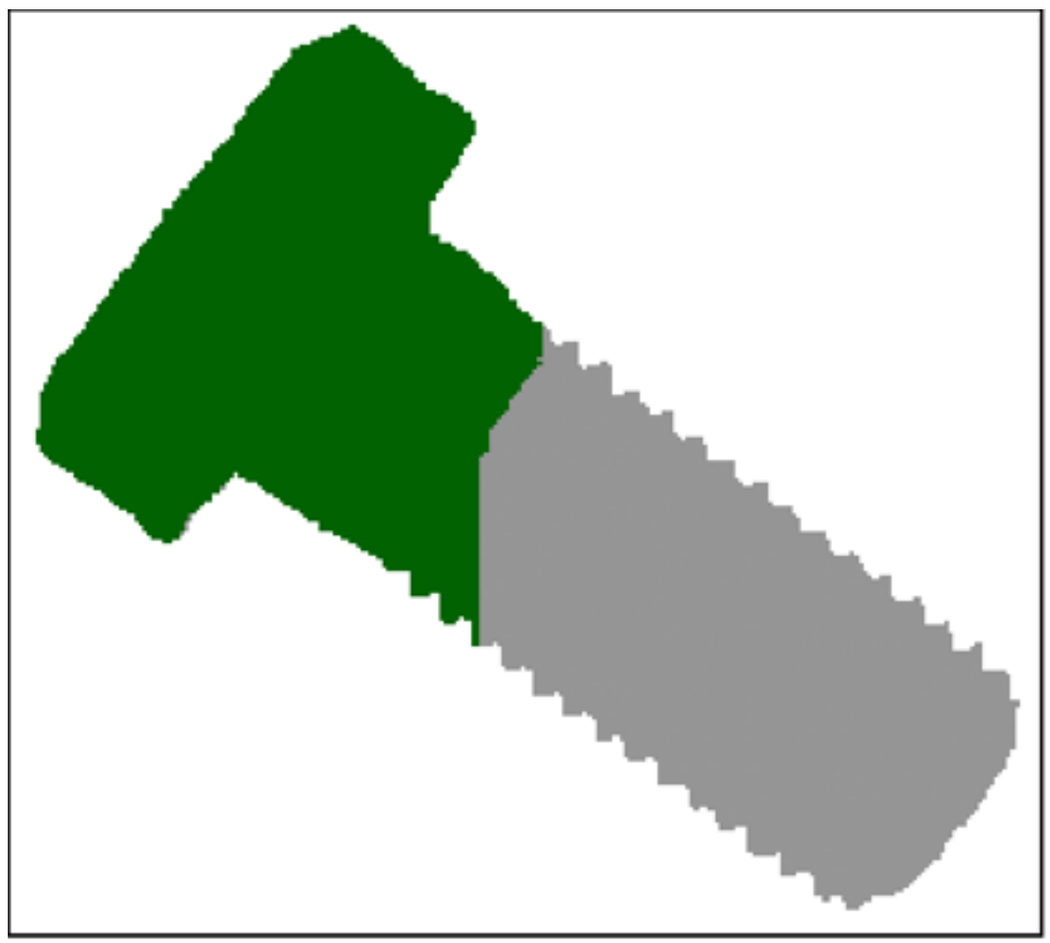


K = 5000 marked pixels

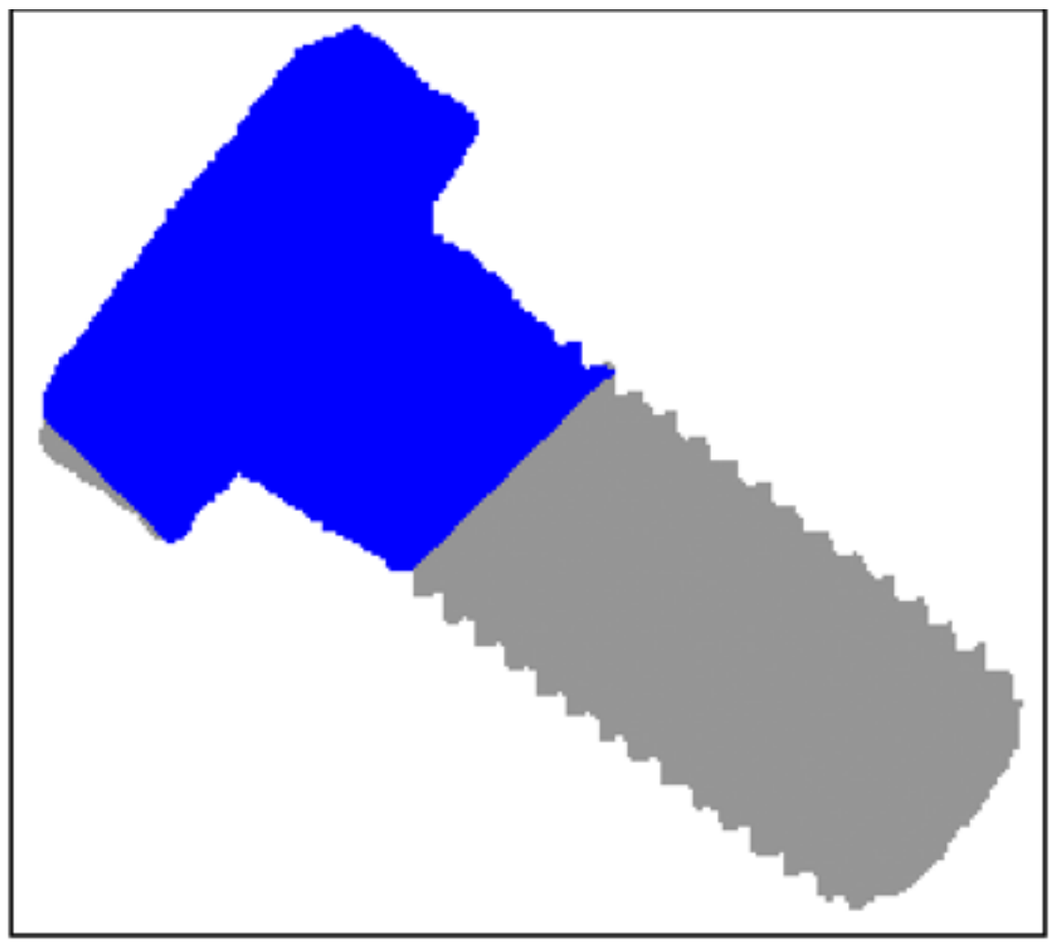


Even more iterations...

depth-first



breadth-first

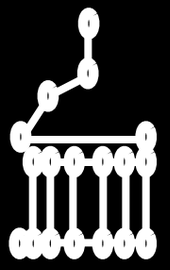


K = 10000 marked pixels

Mini-assignment: finishing the implementations of stack/queue operations

class/lec24ma/main.c

due F, 10/24, by 9:30 a.m.



Finishing the implementations of stack/queue operations (your mini-assignment)

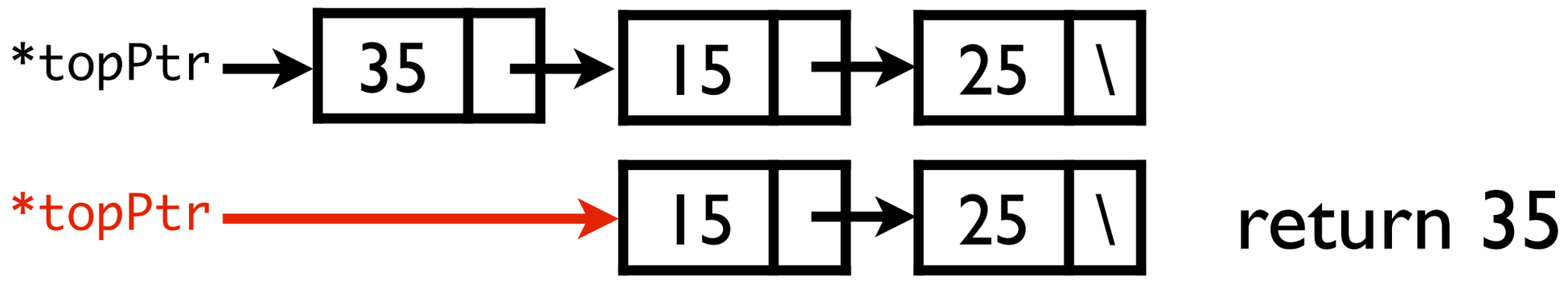
Delete first node from stack/queue and return value:

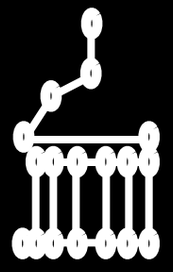
```
int stackPop(NodePtr *topPtr)

int queueDequeue(NodePtr *headPtr, NodePtr *tailPtr)
```

*(code is almost the same in both cases, but you also need to set *tailPtr to NULL if the queue becomes empty)*

stack case:





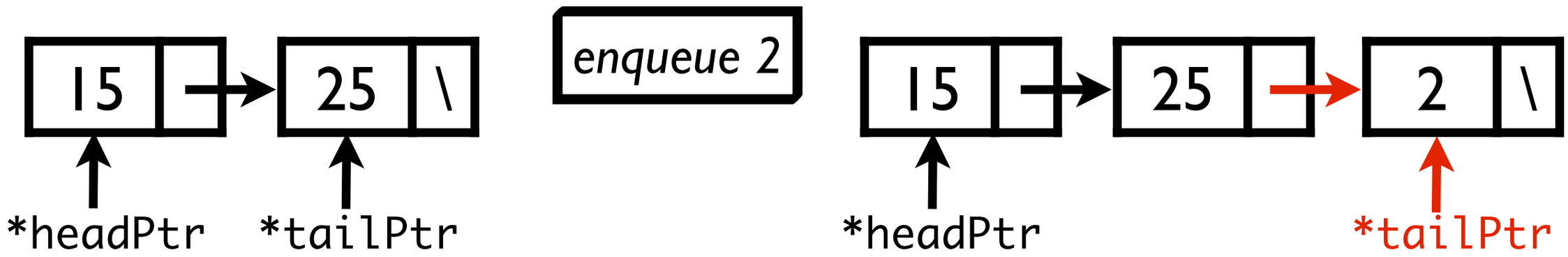
Finishing the implementations of stack/queue operations (your mini-assignment)

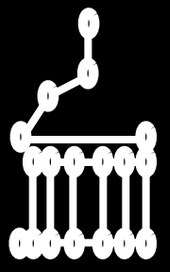
Add node to top of stack:

```
void stackPush(NodePtr *topPtr, int value)
```

Add node to tail of queue:

```
void queueEnqueue(NodePtr *headPtr, NodePtr *tailPtr, int value)
```





Finishing the implementations of stack/queue operations (your mini-assignment)

Delete first node from stack/queue and return value:

```
int stackPop(NodePtr *topPtr)

int queueDequeue(NodePtr *headPtr, NodePtr *tailPtr)
```

*(code is almost the same in both cases, but you will also need to set *tailPtr to NULL if the queue becomes empty)*

Add node to top of stack:

```
void stackPush(NodePtr *topPtr, int value)
```

Add node to tail of queue:

```
void queueEnqueue(NodePtr *headPtr, NodePtr *tailPtr, int value)
```