Stacks API and finding the convex hull

CS 146 - Spring 2017

Today

- Collision detection
- Convex hull
- Graham scan algorithm
- DFS traversal

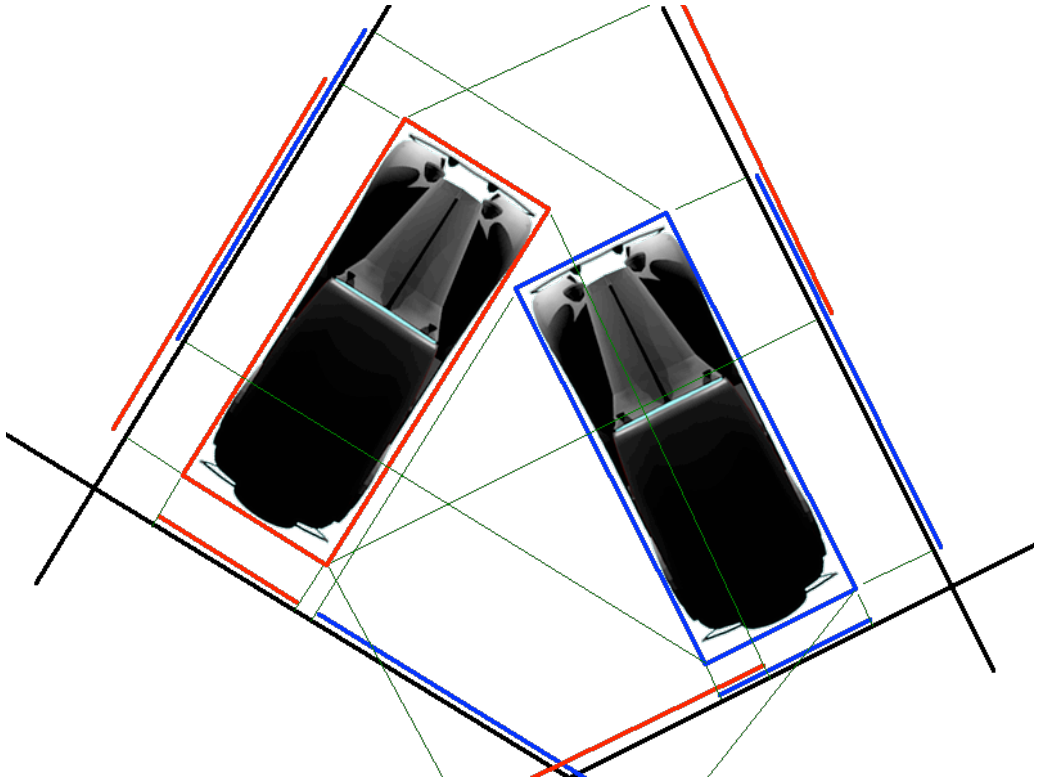
The stack API

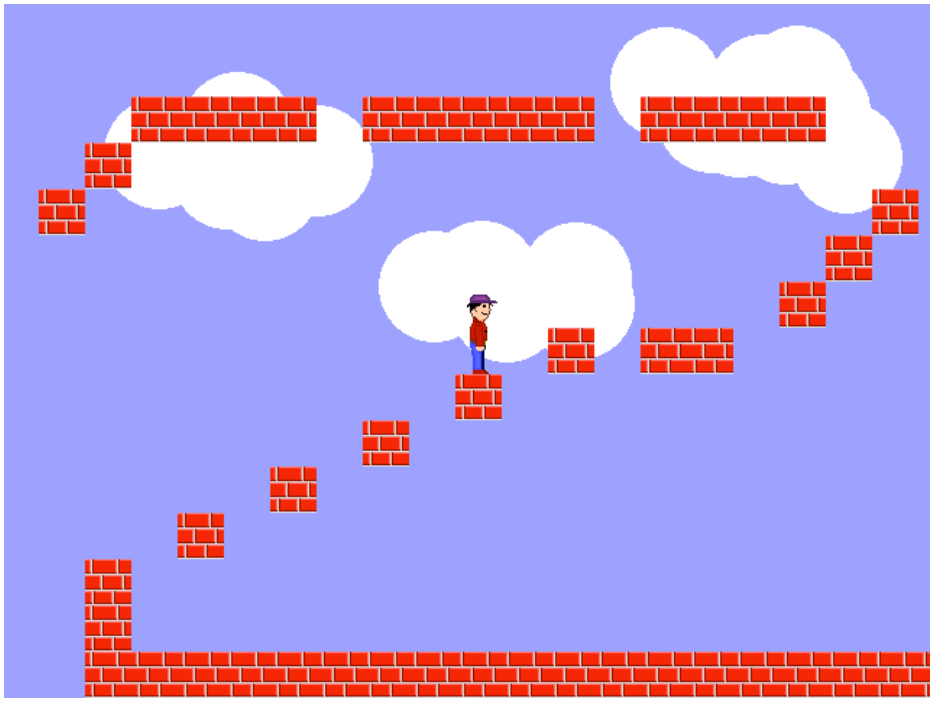
- push()
- pop() <- **no parameter**

What is a stack good for?

- implementing recursion -> call stack (seen before)
- Graham scan algorithm for finding the convex hull (today)
- implementing recursive solutions without recursive calls (next time)

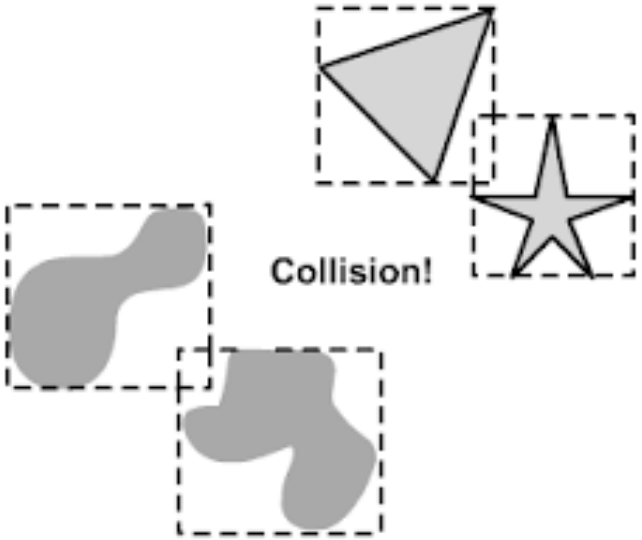
Collision detection in game programming

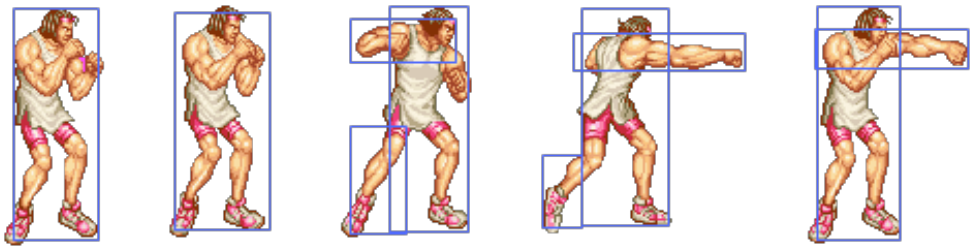




http://www.euclideanspace.com/threed/games/examples/cars/collisions/
http://www.parallelrealities.co.uk/2011/10/intermediate-game-tutorial-4-tile-based.html

Collision detection using a bounding box



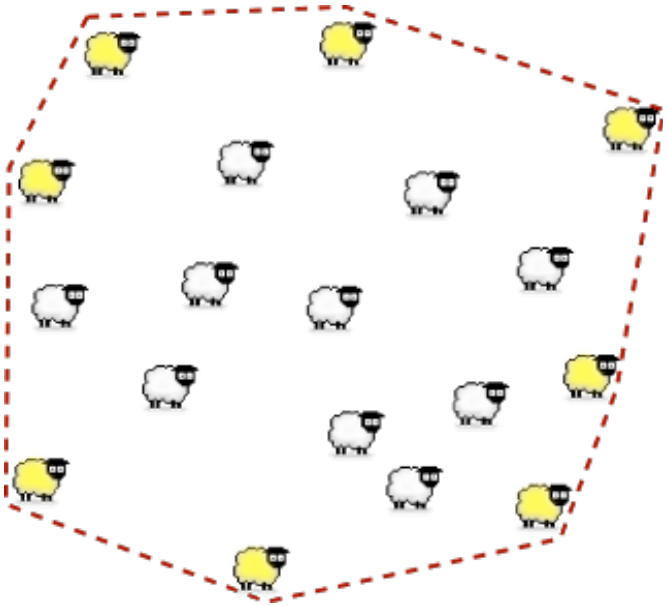


https://code.tutsplus.com/tutorials/quick-tip-collision-detection-between-circles--active-10523

http://www.dreamincode.net/forums/topic/180084-how-i-make-collision-detection-on-a-fight-game/

Convex hull of a set of points

A set of points
in the plane



the convex hull of this set of points is the **smallest convex polygon** that contains all these points

no caving in

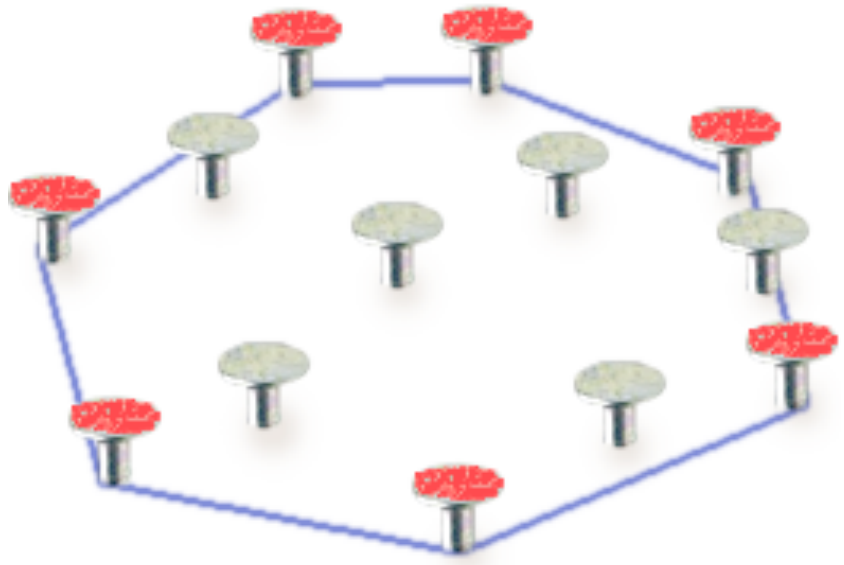
http://mindthenerd.blogspot.com/2012/05/fastest-convex-hull-algorithm-ever.html

what is the convex hull of this set of points?

https://github.com/AndriiHeonia/hull

https://github.com/AndriiHeonia/hull

intuition: the convex hull can be found by releasing a large rubber band around all the points

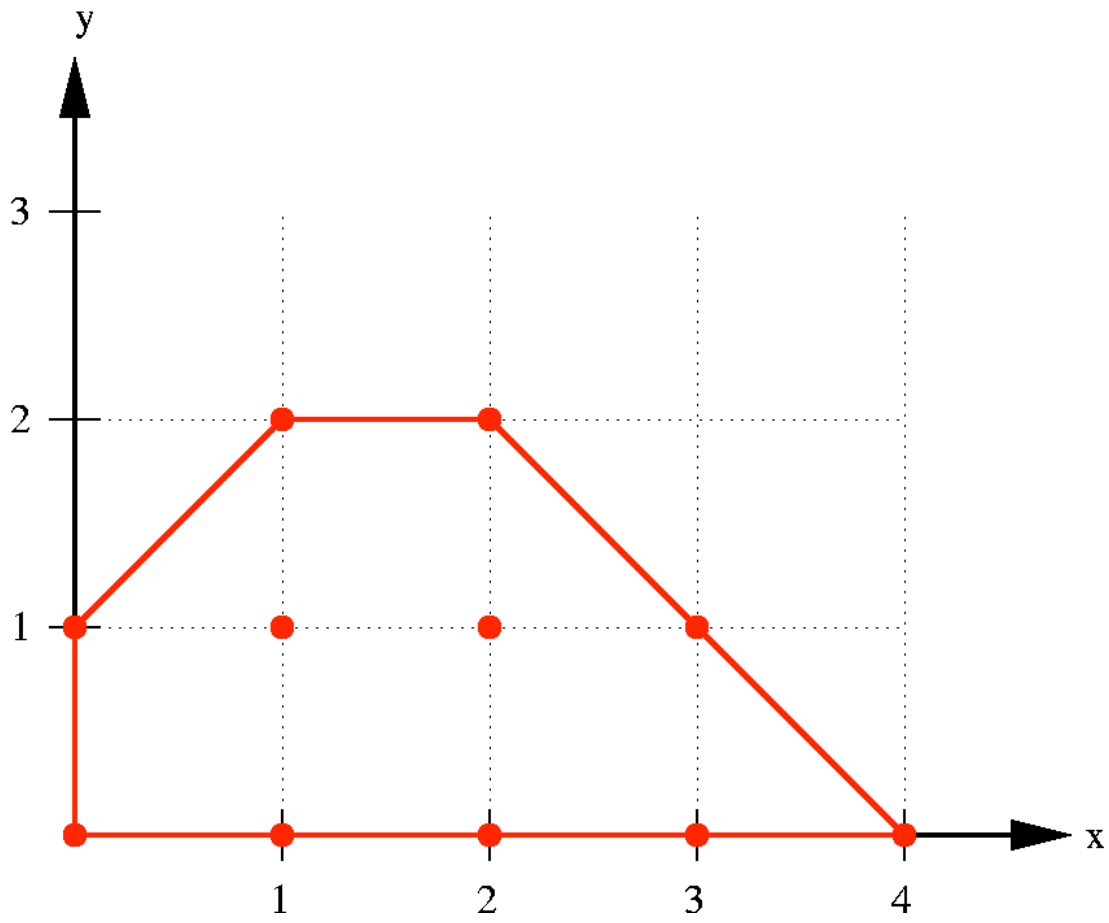


the rubber band + the area enclosed is the convex hull

https://kukuruku.co/post/building-a-minimal-convex-hull/

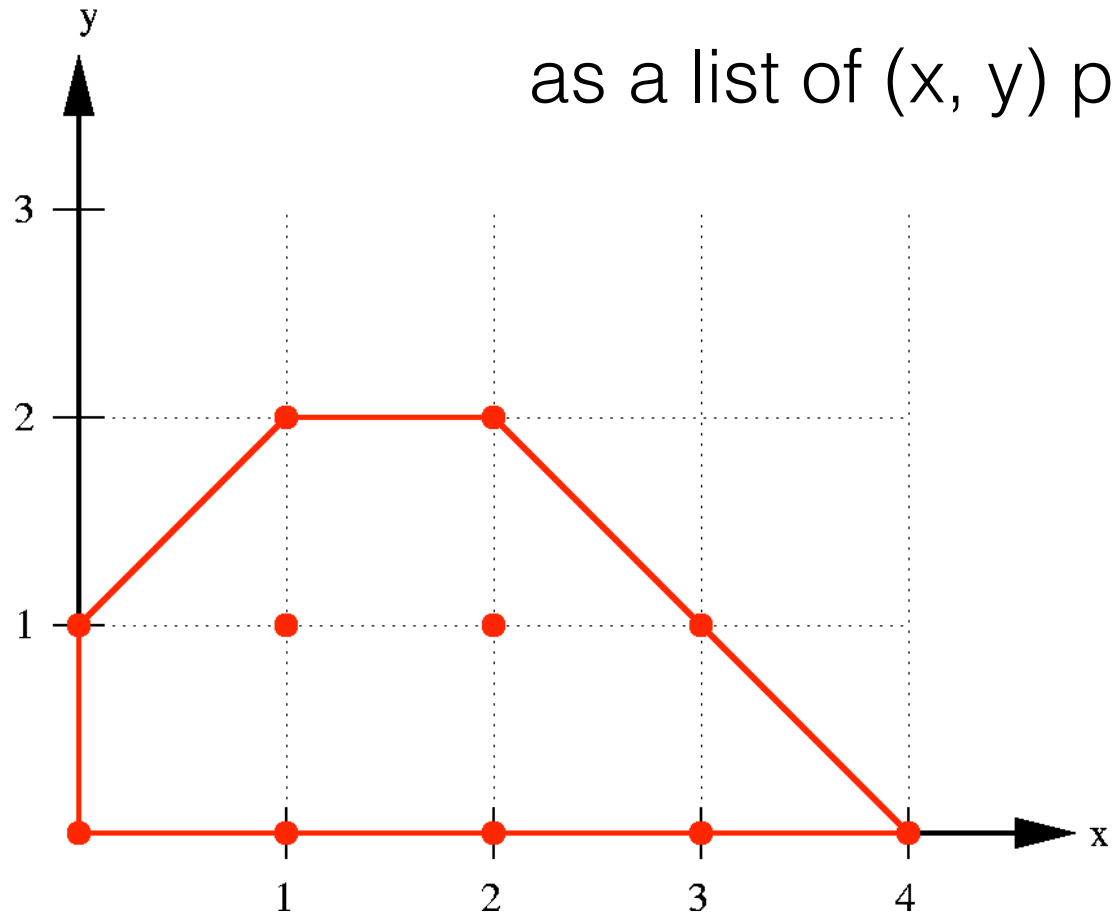
Find a set of 10 points whose convex hull
is a polygon with as few vertices as possible.

How do you represent a convex hull (in the plane) on a computer?



https://en.wikipedia.org/wiki/Convex_hull#/media/File:Convex_hull.png

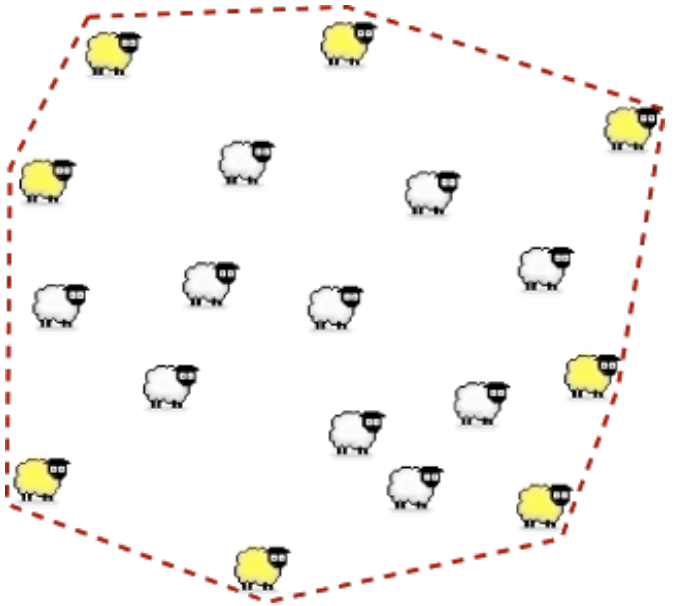
as a list of (x, y) pairs



it is understood the list “wraps around”

https://en.wikipedia.org/wiki/Convex_hull#/media/File:Convex_hull.png

Problem: finding the convex hull

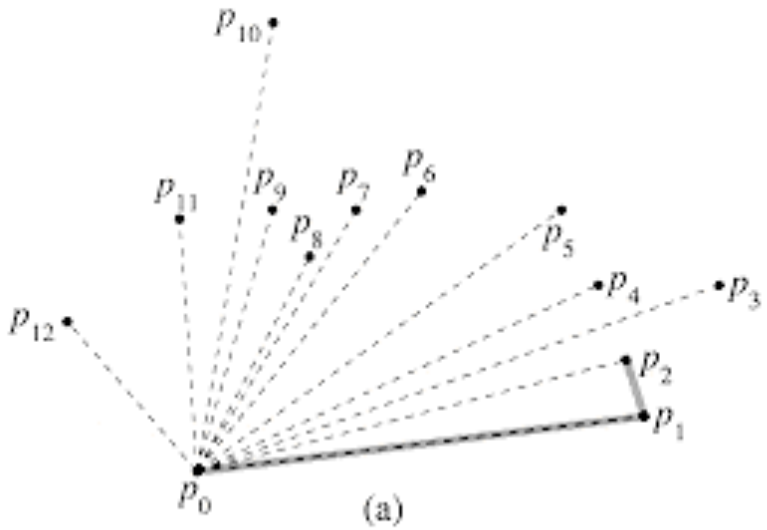


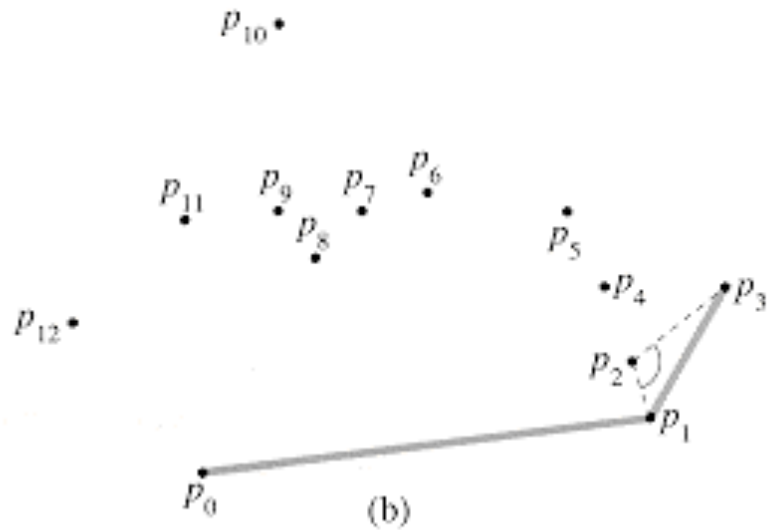
input: a set or list of points where the order is meaningless

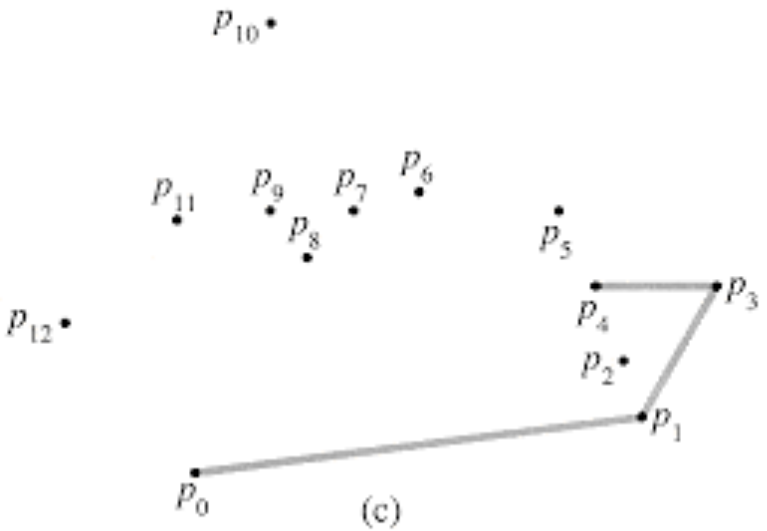
output: list of points where the order is meaningful

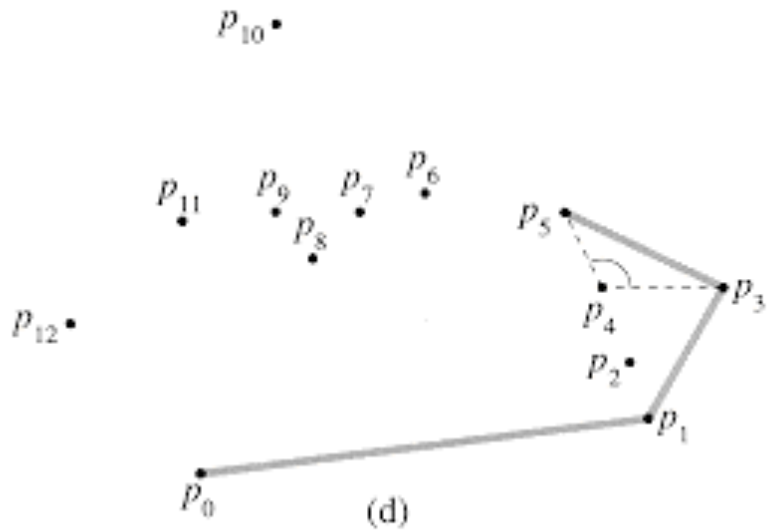
Graham's scan algorithm for finding the convex hull

- find the point **p0** with smallest y coordinate
- sort all other points by their angle counterclockwise around p0
- create stack, push p0, p1, p2
- for i = 3, 4, 5, …, n-1, n, 0
  - while (2nd to last pt on stack) -> (last pt) -> pi forms a right turn, pop()
  - push pi
- return the points that are on the stack

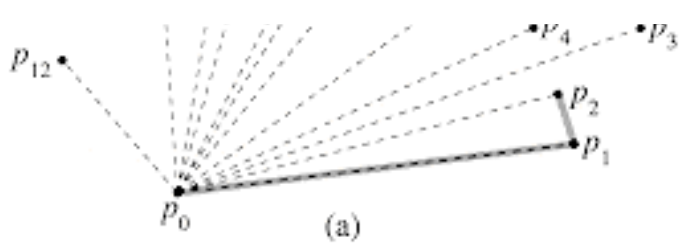


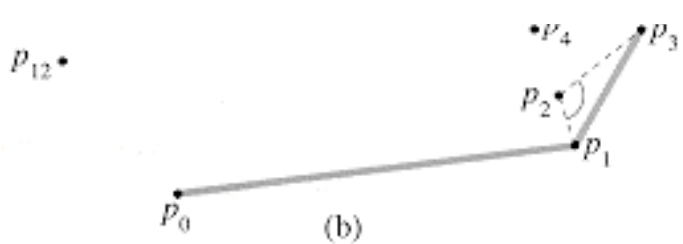


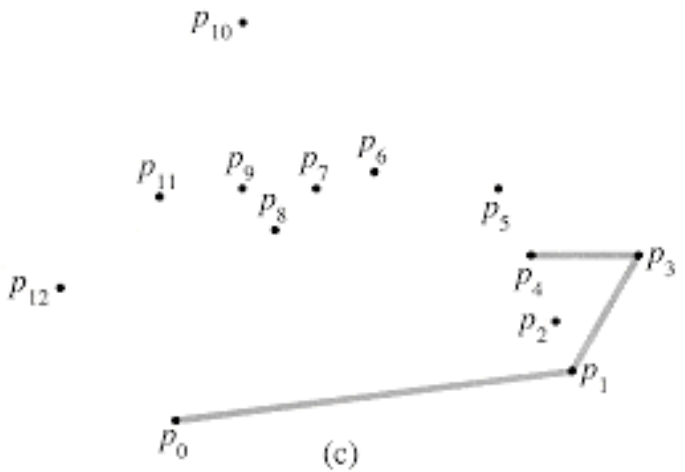


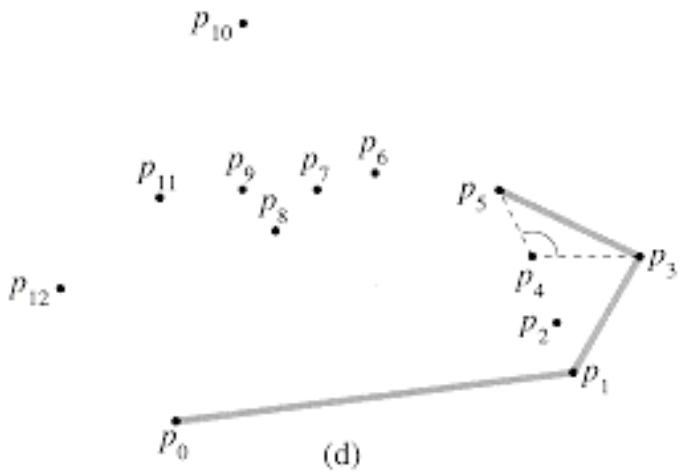


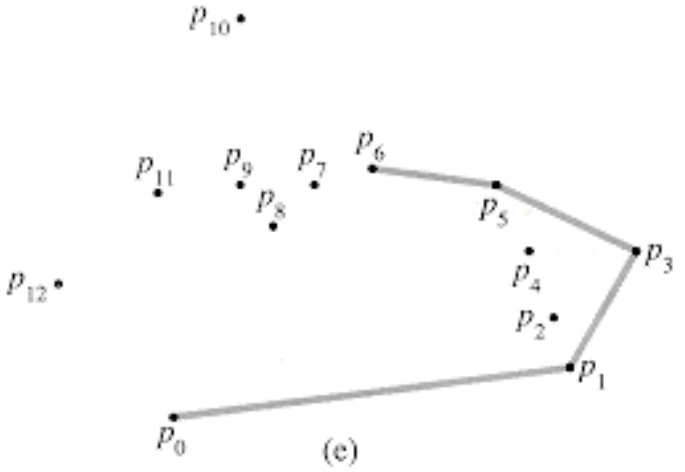
http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap35.htm

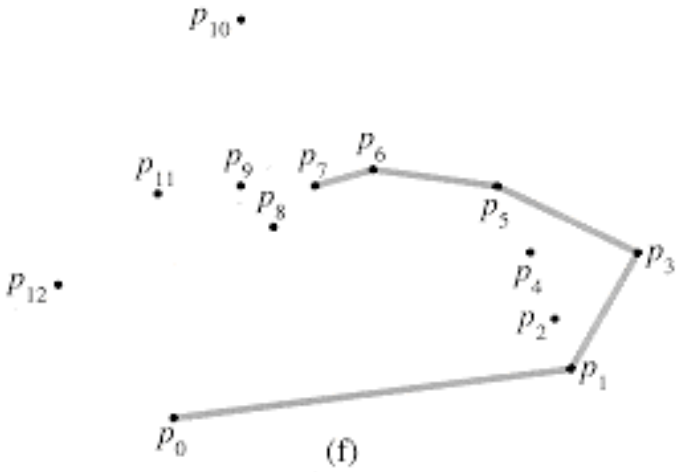




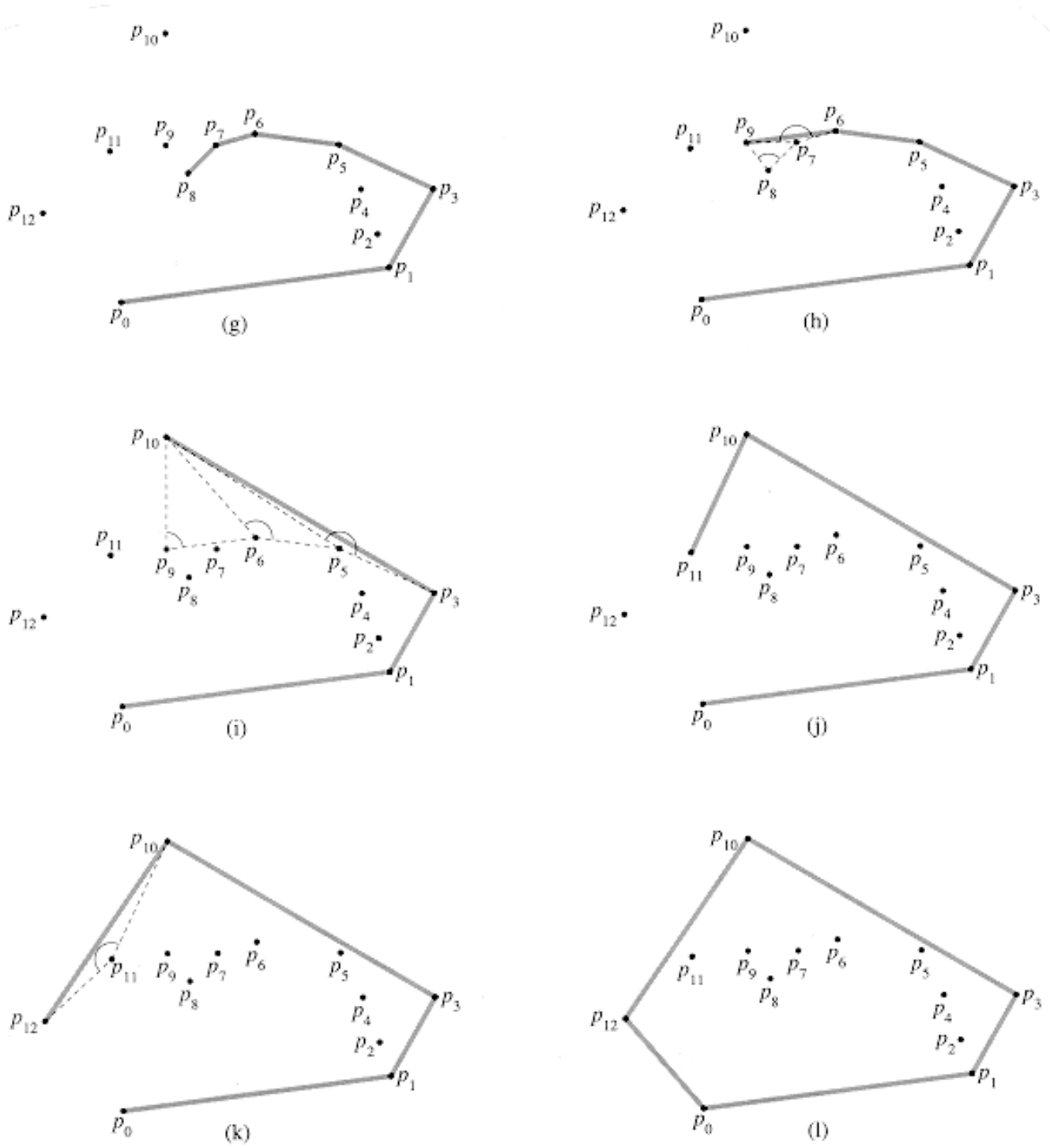








http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap35.htm



http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap35.htm

Graham's scan algorithm for finding the convex

- find the point **p0** with smallest y coordinate
- sort all other points by polar angle counterclockwise around p0
- create stack, push p0, p1, p2
- for i = 3, 4, 5, …, n-1, n, 0
  - while (2nd to last pt on stack) -> (last pt) -> pi forms a right turn, pop()
  - push pi
- return the points that are on the stack in that order

can be any point on the convex hull make sure it can be found quickly

loop invariant:
stack contains all points on convex hull processed so far

running time?

Graham's scan algorithm for finding the convex hull

- find the point **p0** with smallest y coordinate
- sort all other points by polar angle counterclock
- create stack, push p0, p1, p2
- for i = 3, 4, 5, …, n-1, n, 0
  - while (2nd to last pt on stack) -> (last pt) -> pi forms a right turn, pop()
  - push pi
- return the points that are on the stack in that order

linear (just scan)

n log n

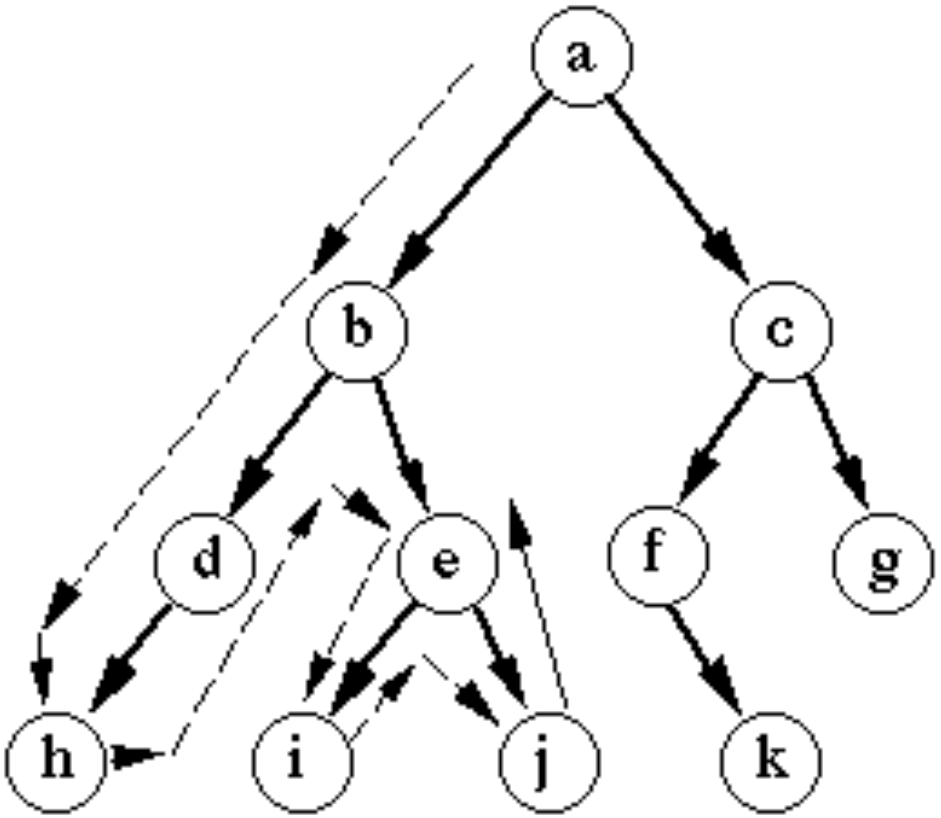
Theta(n): each point is pushed once onto stack and popped at most once

running time O(n log n)

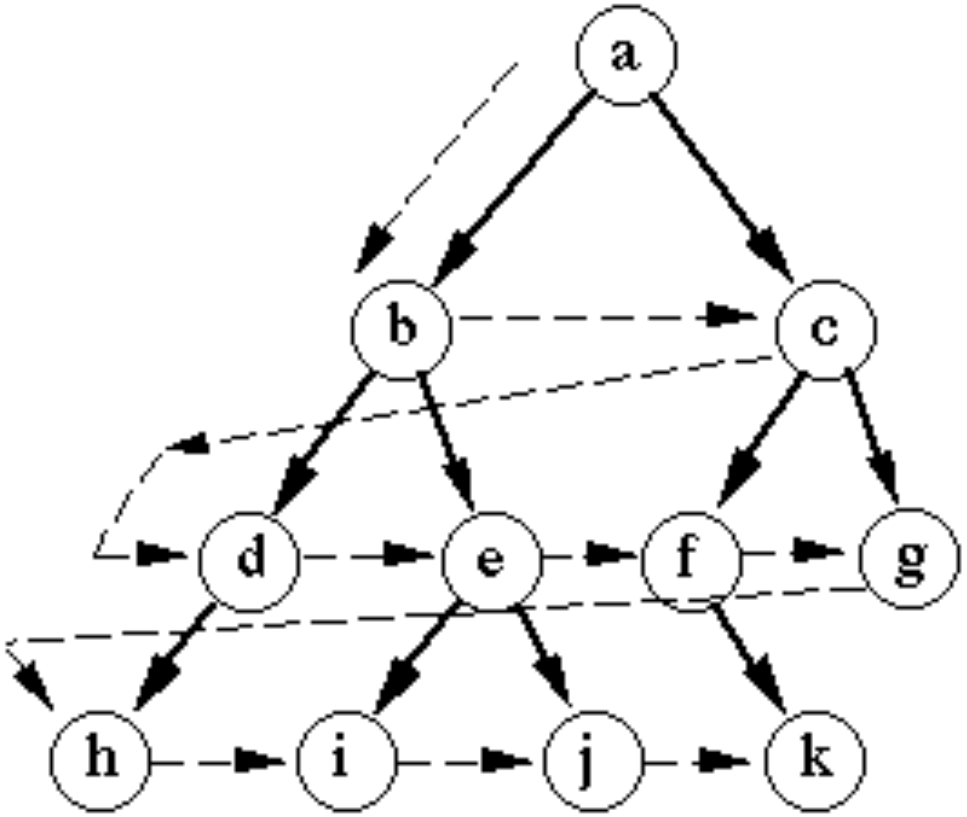
What about collision detection?

- Do 2 given shapes overlap?
- Do 2 given rectangles overlap? <- easy and O(1)
- Do 2 given convex shapes overlap? <- V-clip algorithm

Using a **stack** to traverse a graph.



Depth-first search



Breadth-first search

http://www.cse.unsw.edu.au/~billw/Justsearch.html

DFS on a tree is just a generalization of tree-traversal

```
preorder(v) {
    // do stuff on v
    if v has left child w
        preorder(w)
    if v has right child w
        preorder(w)
}
```

```
dfs(v) {
    // do stuff on v
    for every neighbor w of v
        dfs(w)
}
```

sometimes, it will also be useful to do stuff afterwards

DFS on a graph

```
Set visited = new Set();
dfs(v) {
    visited.add(v)
    // do stuff on v
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
}
```

observation: each node is “visited” (= input to a dfs call) once

Connection with textbook version

```
Set visited = new Set();

dfs(v) {
    visited.add(v)
    // do stuff on v
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
}
```

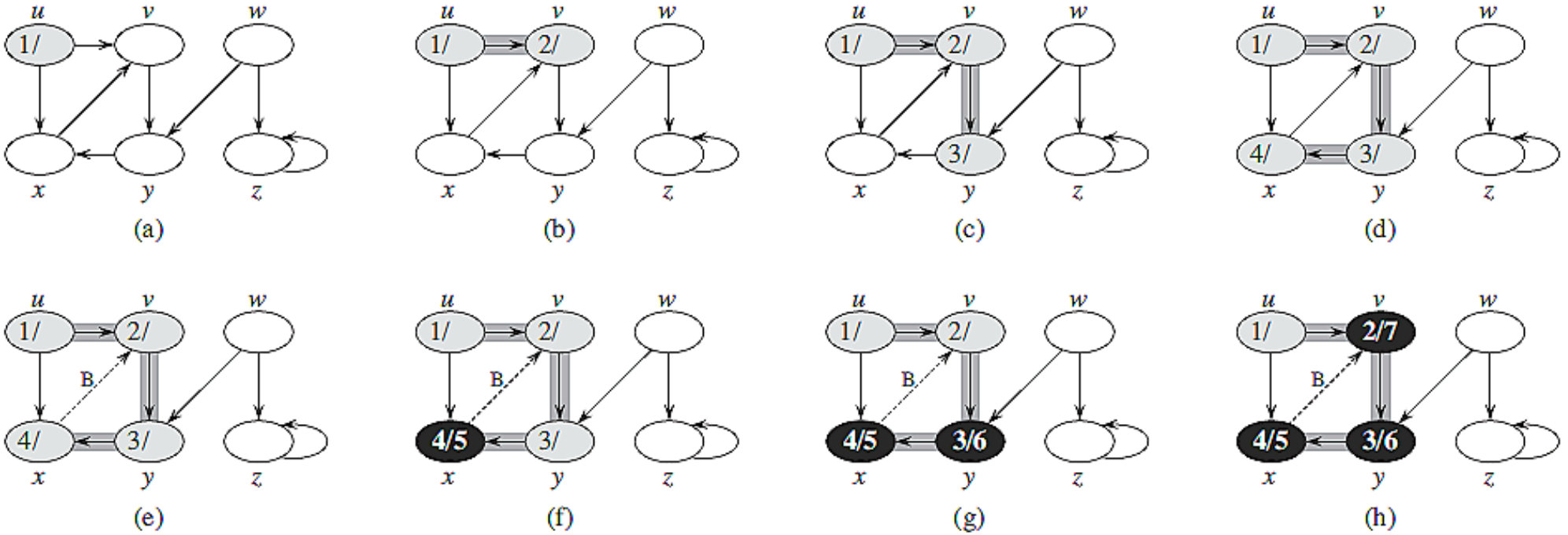
every vertex starts out **white (not visited)**

upon entering a recursive call, v turns **grey (visiting)**

upon exiting the recursive call, v turns **black (finished visiting)**

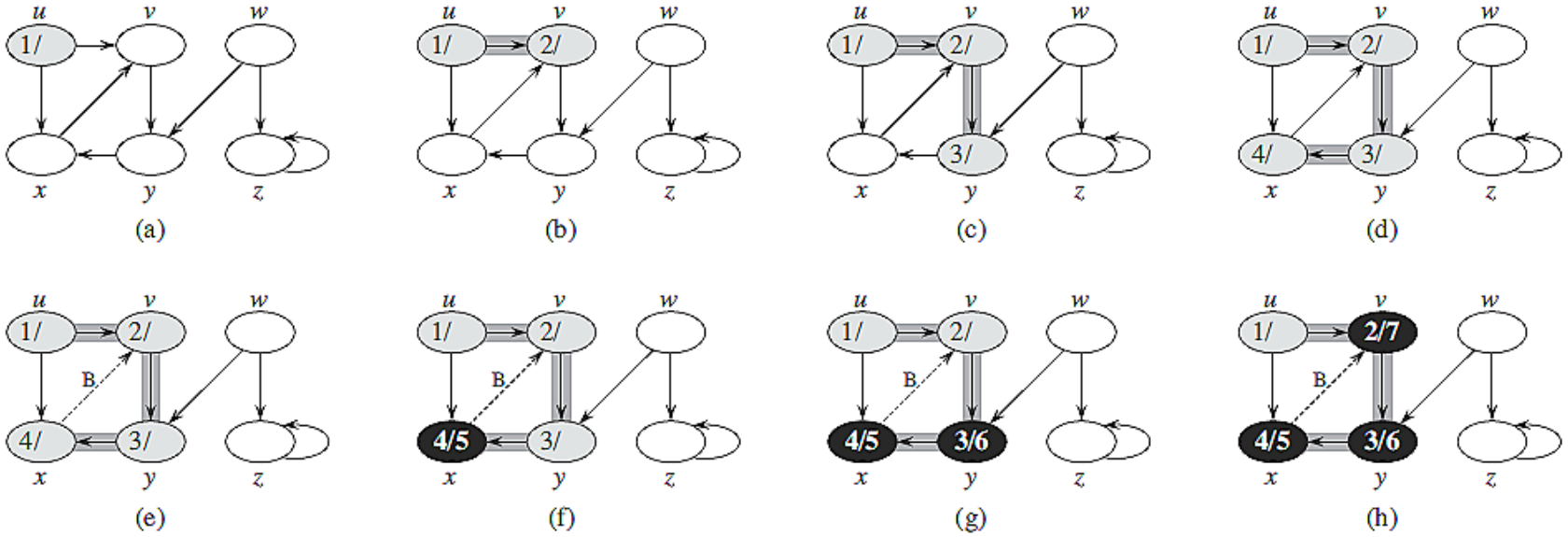
Note similarities with the lifecycle of a vertex in BFS

Example with timestamps



each vertex gets a **“visiting” timestamp** when first visited

each vertex gets a **“done visiting” timestamp** when finished




```
Set visited = new Set();
dfs(v) {
    visited.add(v)
    // do stuff on v
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
}
```

how would you modify dfs() to print timestamps…?

and ensure that every vertex gets a timestamp?

DFS for visit timestamp

```
Set visited = new Set();
time = 1


dfs(v) {
    visited.add(v)
    print("visiting v: " + time); time++
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
    print("done visiting v: " + time); time++
}
```

To ensure all vertices are visited (and get a timestamp)

```
Set visited = new Set();
time = 1

for each vertex v
    if (!visited.contains(v))
        dfs(v)

dfs(v) {
    visited.add(v)
    print("visiting v: " + time); time++
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
    print("done visiting v: " + time); time++
}
```

Time complexity of DFS

```
Set visited = new Set();

dfs(v) {
    visited.add(v)
    // do stuff on v
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
}
```

each vertex is input of at most 1 recursive call

each edge is explored at most once overall

as long as “do stuff” is O(1)
DFS is O(V + E)

Time complexity of DFS

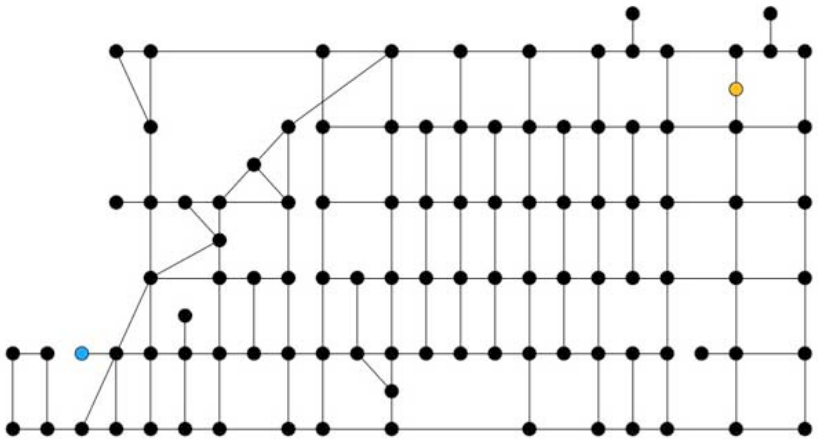
```
Set visited = new Set();

dfs(v) {
    visited.add(v)
    // do stuff with v
    for every neighbor w of v
        if (!visited.contains(w))
            dfs(w)
}
```

each vertex is input of
at most 1 recursive call

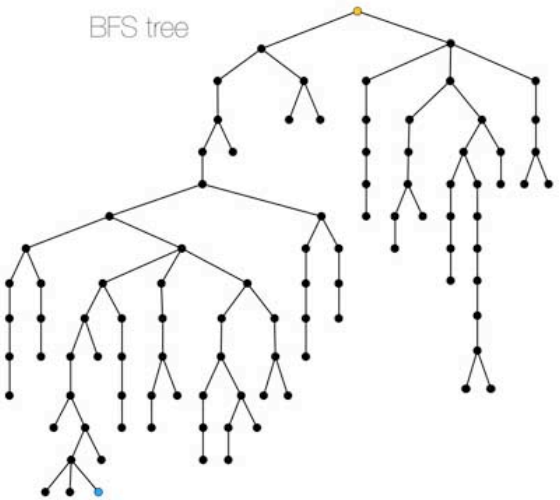
Note similarities with BFS/Dijkstra analysis despite DFS being a recursive algorithm

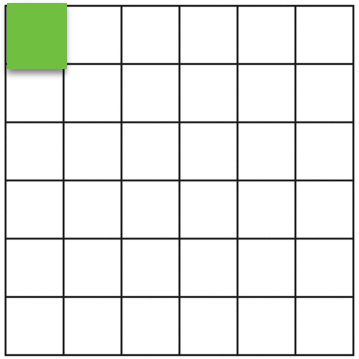
each edge is explored
at most once overall

as long as “do stuff” is O(1)
DFS is O(V + E)

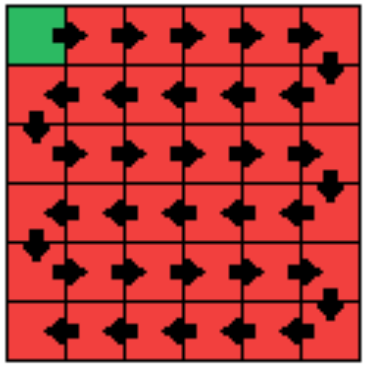


a graph to traverse

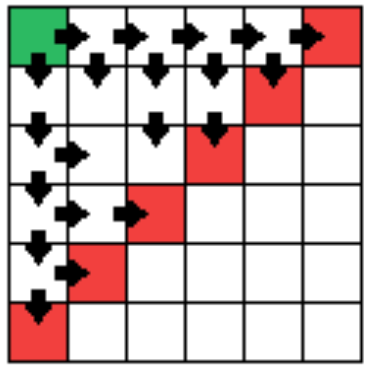




DFS



BFS



https://www.codefellows.org/blog/trees-as-graphs-vs-trees-as-data-structures/
http://stackoverflow.com/questions/20192445/which-procedure-we-can-use-for-maze-exploration-bfs-or-dfs