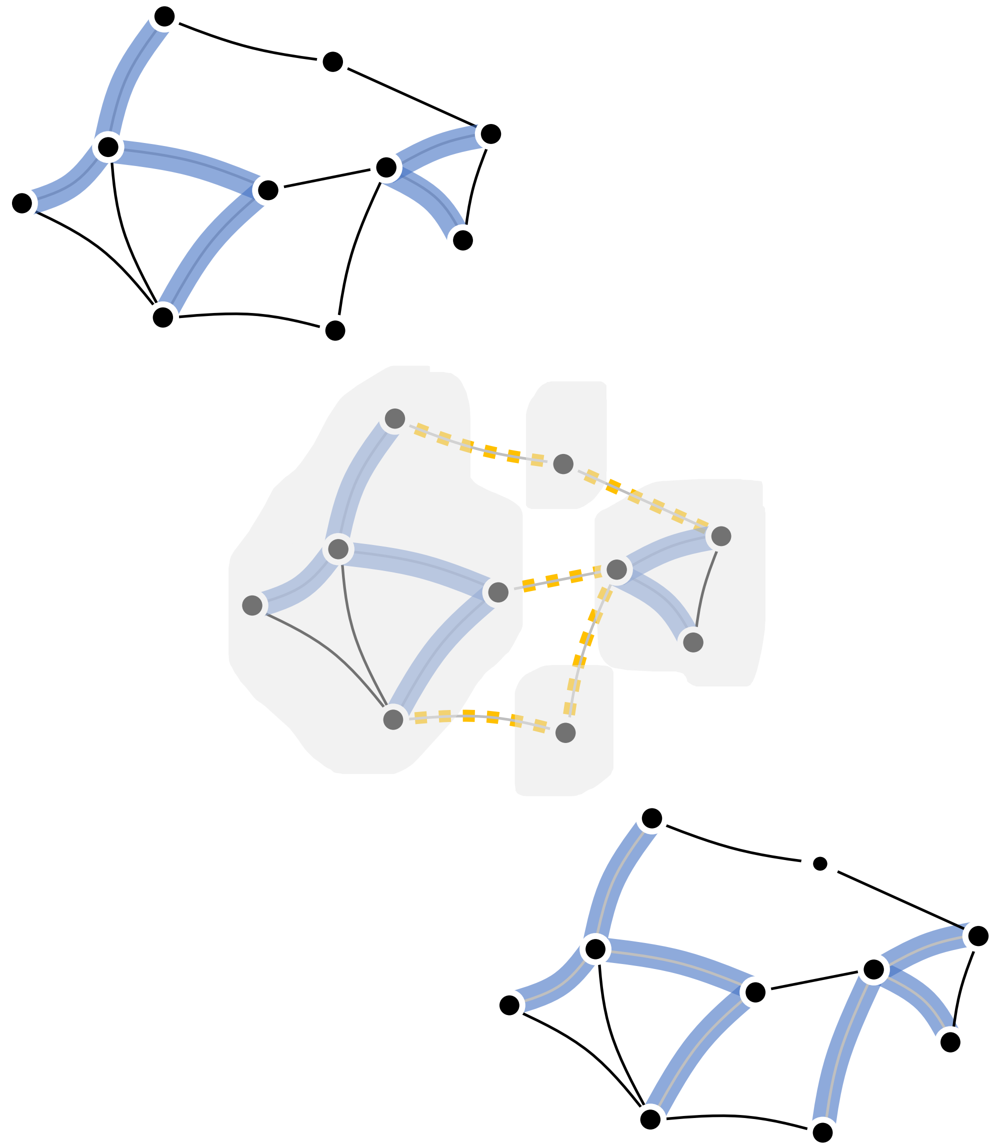
KRUSKAL'S ALGORITHM

Given a **forest** we've built so far,

1. look at all the edges that would join two fragments of the forest
2. pick the lowest-weight one and add it to the tree, thereby joining two fragments
3. *Assert: the* ***forest*** *we have so far is part of some minimum spanning tree*

Repeat until we have a spanning tree.



KRUSKAL'S ALGORITHM

Given a **forest** we've built so far,

1. look at all the edges that would join two fragments of the forest
2. pick the lowest-weight one and add it to the tree, thereby joining two fragments
3. *Assert: the **forest** we have so far is part of some minimum spanning tree*

Repeat until we have a spanning tree.

```
def kruskal(g):
    tree_edges = []
    partition = DisjointSet()
    for v in g.vertices:
        partition.addsingleton(v)
    edges = sorted(g.edges, sortkey = λ(u,v,weight): weight)

    for (u,v,edgeweight) in g.edges:
        p = partition.getsetwith(u)
        q = partition.getsetwith(v)
        if p != q:
            tree_edges.append((u,v))
            partition.merge(p, q)
```

Don't recompute these edges every iteration.

Just pre-sort the list of all edges, then ignore those that are within-fragment.

Total cost $O(V + E + E \log E)$

We're assuming a connected graph.

$E \geq V-1 \Rightarrow V \leq E+1$

$E \leq \frac{1}{2}V(V-1) \Rightarrow \log E \leq 2 \log V$

Total cost $O(E \log V)$

```
def kruskal(g):
    tree_edges = []
    partition = DisjointSet()
    for v in g.vertices:
        partition.addsingleton(v)
    edges = sorted(g.edges, sortkey = λ(u,v,weight): weight)

    for (u,v,edgeweight) in g.edges:
        p = partition.getsetwith(u)
        q = partition.getsetwith(v)
        if p != q:
            tree_edges.append((u,v))
            partition.merge(p, q)
```

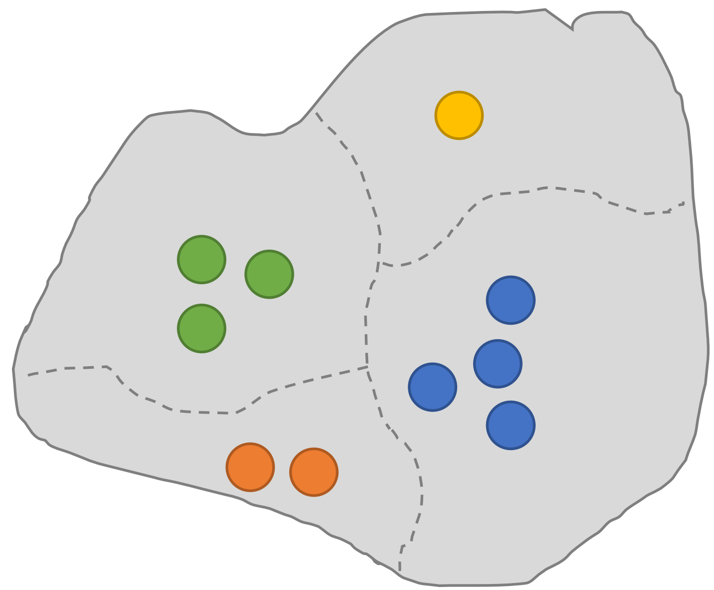
$O(V)$

$O(E \log E)$

$O(E)$

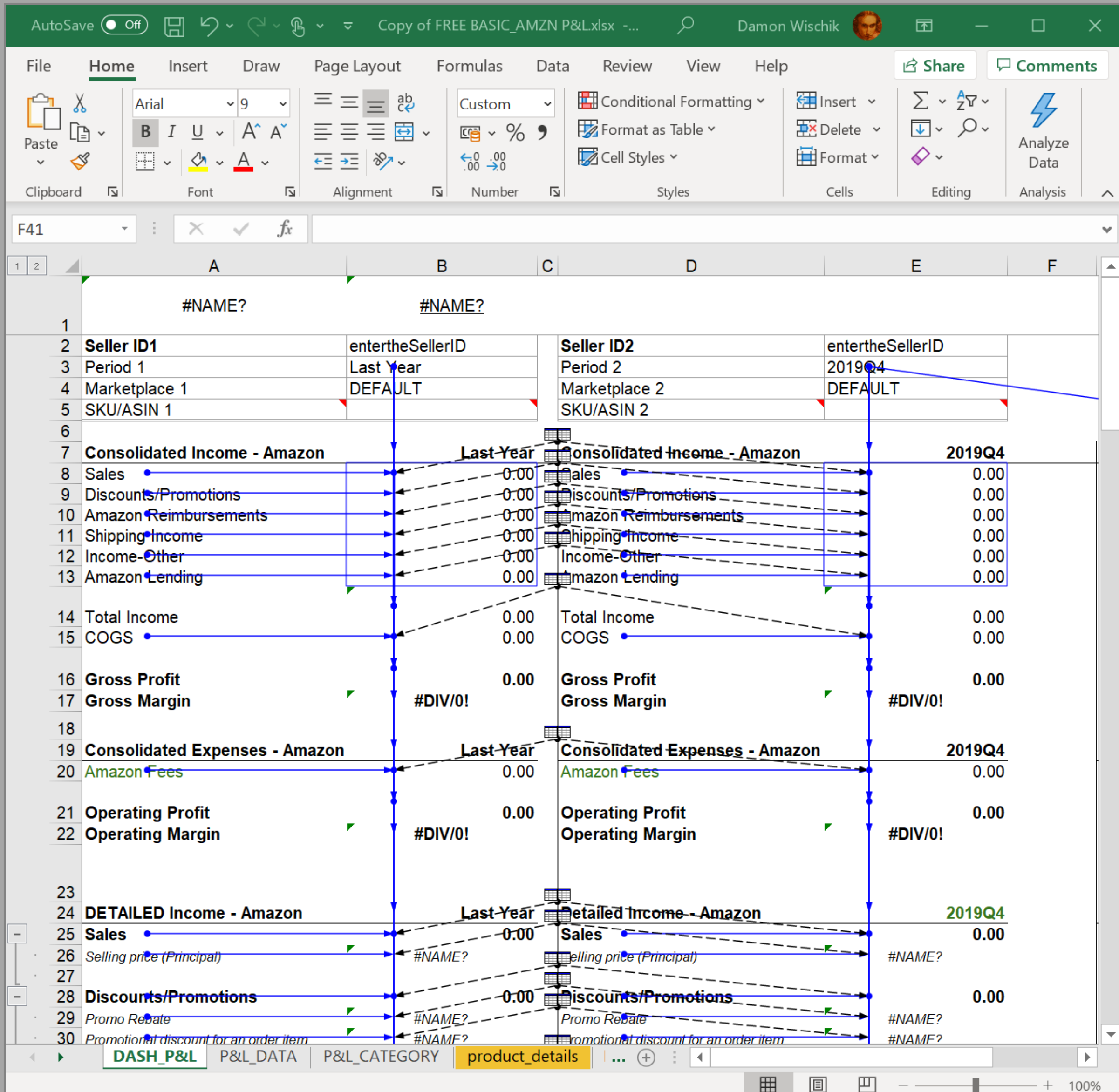
The abstract data type **DisjointSet** stores a collection of disjoint sets, and supports

- $O(1)$ ish `addsingleton(v)`
- $O(1)$ ish `p = getsetwith(v)`
- $O(1)$ ish `merge(p,q)`



SECTION 6.7

Topological sort



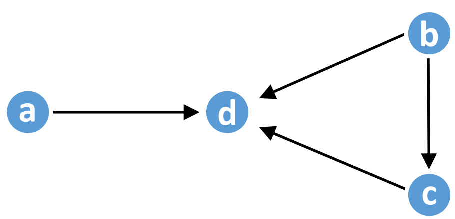
	A	B	D	E
1	#NAME?	#NAME?		
2	**Seller ID1**	entertheSellerID	**Seller ID2**	entertheSellerID
3	Period 1	Last Year	Period 2	2019Q4
4	Marketplace 1	DEFAULT	Marketplace 2	DEFAULT
5	SKU/ASIN 1		SKU/ASIN 2	
6				
7	**Consolidated Income - Amazon**	**Last Year**	**Consolidated Income - Amazon**	**2019Q4**
8	Sales	0.00	Sales	0.00
9	Discounts/Promotions	0.00	Discounts/Promotions	0.00
10	Amazon Reimbursements	0.00	Amazon Reimbursements	0.00
11	Shipping Income	0.00	Shipping Income	0.00
12	Income-Other	0.00	Income-Other	0.00
13	Amazon Lending	0.00	Amazon Lending	0.00
14	Total Income	0.00	Total Income	0.00
15	COGS	0.00	COGS	0.00
16	**Gross Profit**	**0.00**	**Gross Profit**	**0.00**
17	**Gross Margin**	**#DIV/0!**	**Gross Margin**	**#DIV/0!**
18				
19	**Consolidated Expenses - Amazon**	**Last Year**	**Consolidated Expenses - Amazon**	**2019Q4**
20	Amazon Fees	0.00	Amazon Fees	0.00
21	**Operating Profit**	**0.00**	**Operating Profit**	**0.00**
22	**Operating Margin**	**#DIV/0!**	**Operating Margin**	**#DIV/0!**
23				
24	**DETAILED Income - Amazon**	**Last Year**	**Detailed Income - Amazon**	**2019Q4**
25	**Sales**	**0.00**	**Sales**	**0.00**
26	*Selling price (Principal)*	*#NAME?*	*Selling price (Principal)*	*#NAME?*
27				
28	**Discounts/Promotions**	**0.00**	**Discounts/Promotions**	**0.00**
29	*Promo Rebate*	*#NAME?*	*Promo Rebate*	*#NAME?*
30	*Promotional discount for an order item*	*#NAME?*	*Promotional discount for an order item*	*#NAME?*

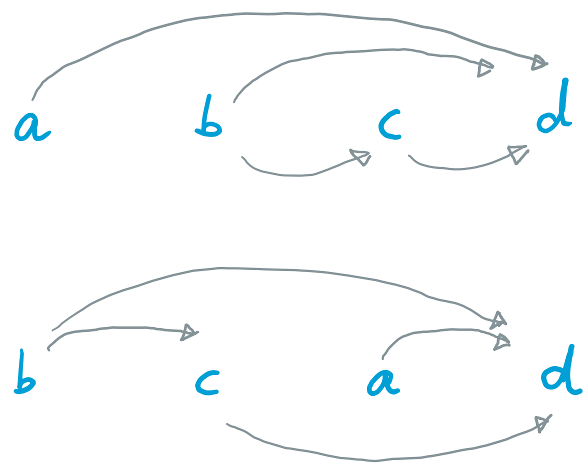
DEFINITION

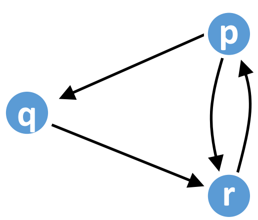
Given a directed graph, a **total ordering** is an ordering of the vertices such that if there is an edge $v \to u$ in the graph, then $v < u$ in the ordering.

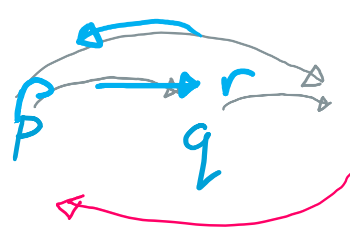
PROBLEM STATEMENT

Find a total ordering, if one exists.









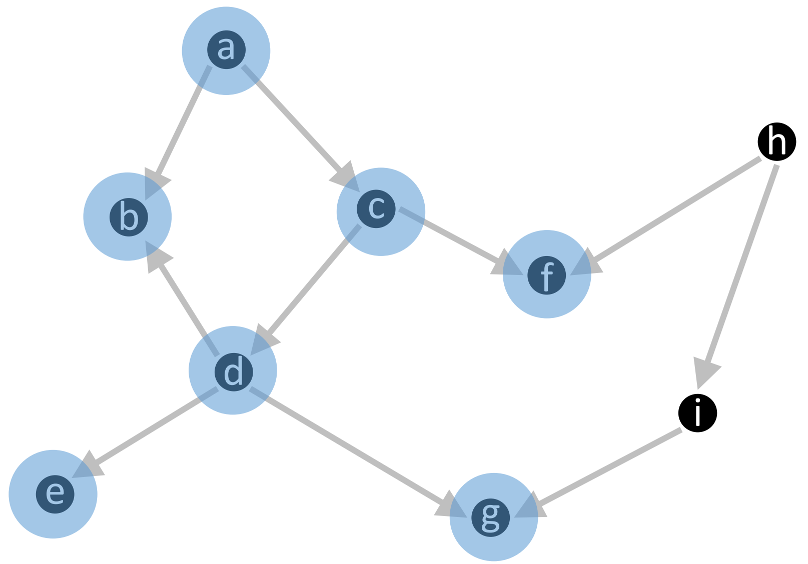
This graph has a cycle so there ~~is~~ no total order possible.

This graph has a cycle, so no total order is possible.

```
def dfs_recurse(g, s):
    for v in g.vertices:
        v.visited = False
    visit(s)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)
```

attempt 1: depth-first search

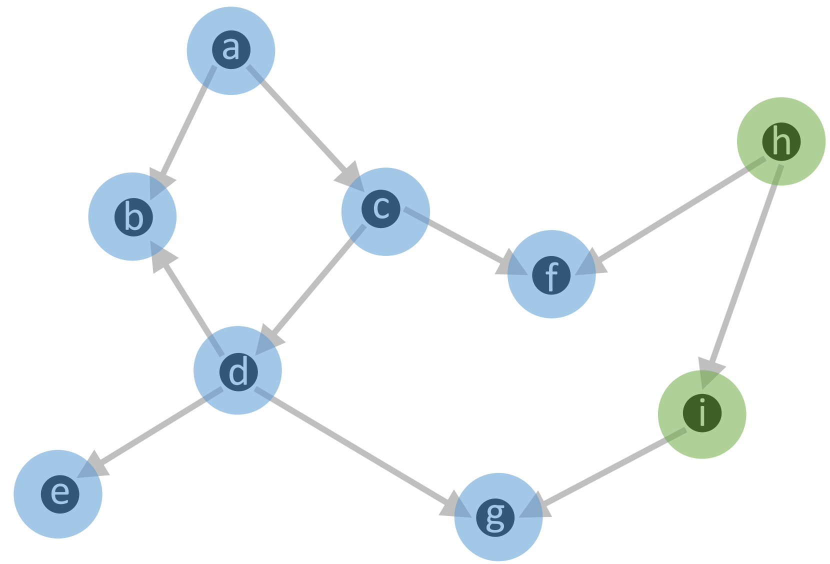


This might not even visit all vertices, so it might not produce a total order.

```python
def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)
```

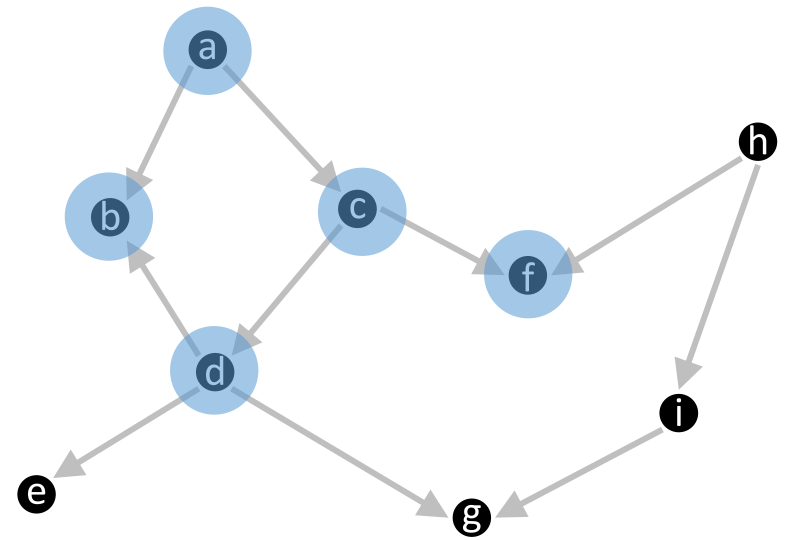
attempt 2: comprehensive depth-first search



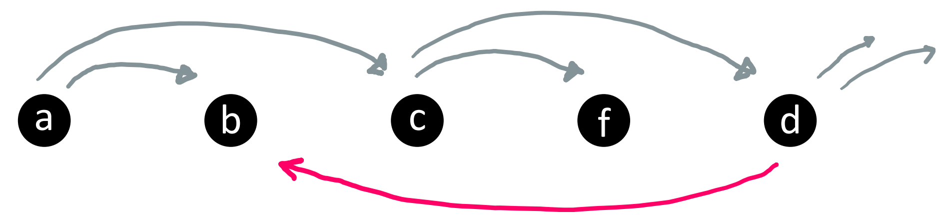
```python
def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)
```

attempt 2: comprehensive depth-first search



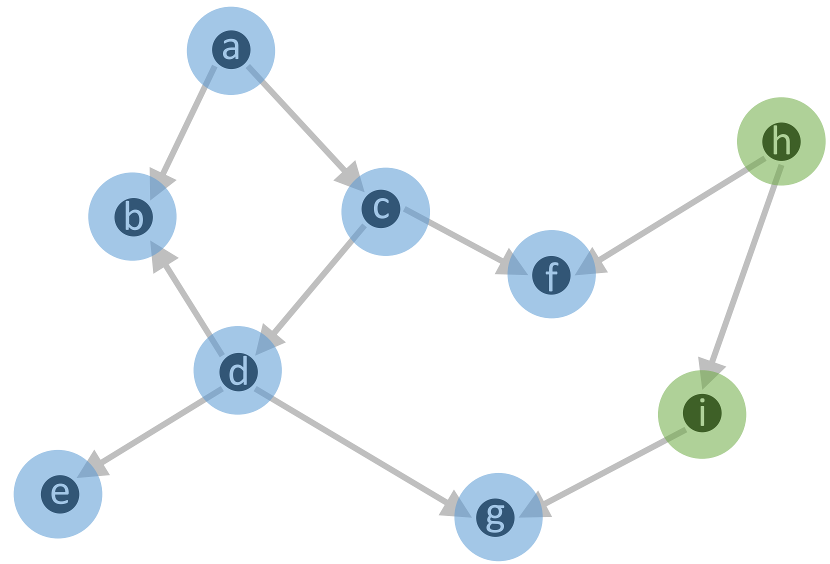
Some edges point backwards – not a total order.



```python
def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)
```

attempt 2: comprehensive depth-first search



it(d)

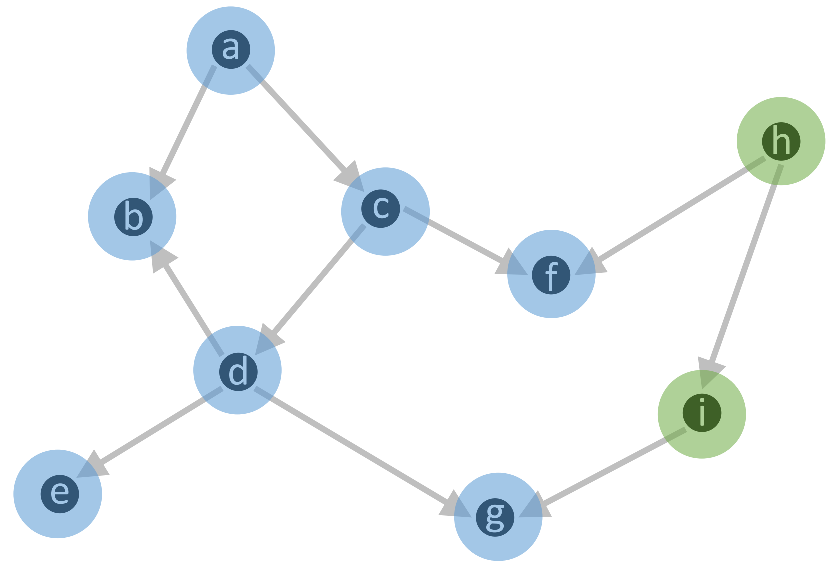
dfs_recurse_all()

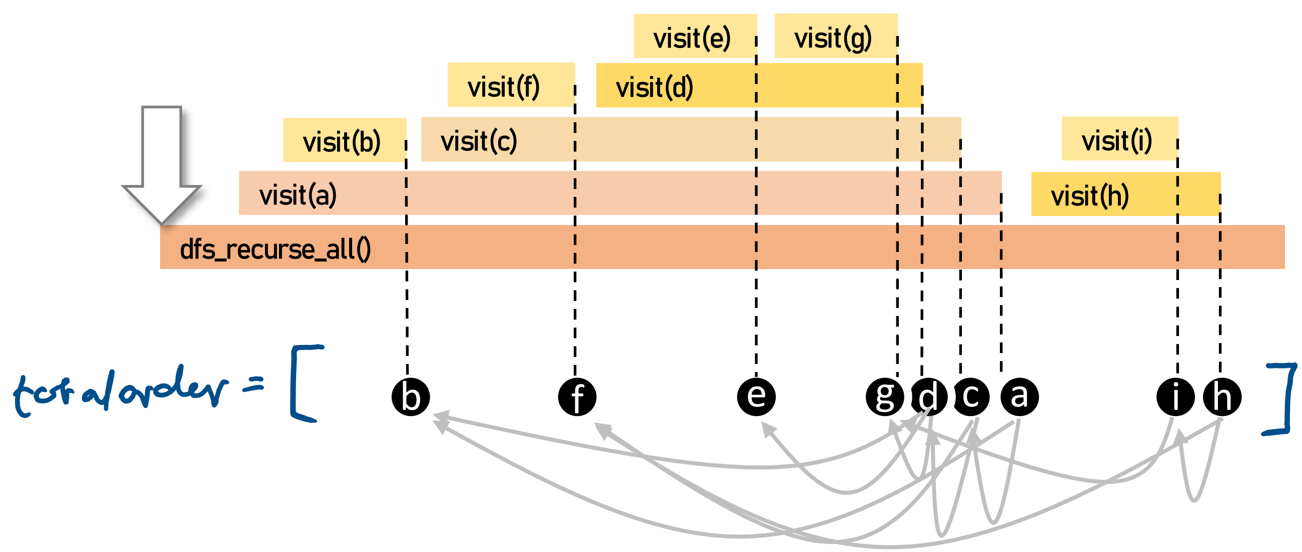
all of a's descendants

visit(a) returns

```python
def toposort(g):
    for v in g.vertices:
        v.visited = False
        # v.colour = 'white'
    totalorder = []
    for v in g.vertices:
        if not v.visited:
            visit(v, totalorder)
    return totalorder

def visit(v, totalorder):
    v.visited = True
    # v.colour = 'grey'
    for w in v.neighbours:
        if not w.visited:
            visit(w, totalorder)
    totalorder.append(v)
    # v.colour = 'black'
```





```
def toposort(g):
    for v in g.vertices:
        v.visited = False
        # v.colour = 'white'
    totalorder = []
    for v in g.vertices:
        if not v.visited:
            visit(v, totalorder)
    return totalorder

def visit(v, totalorder):
    v.visited = True
    # v.colour = 'grey'
    for w in v.neighbours:
        if not w.visited:
            visit(w, totalorder)
    totalorder.append(v)
    # v.colour = 'black'
```

Correctness theorem.

Given a DAG g, this algorithm produces a `totalorder` such that for every edge $v_1 \to v_2$, v_1 appears to the right of v_2 in `totalorder`.

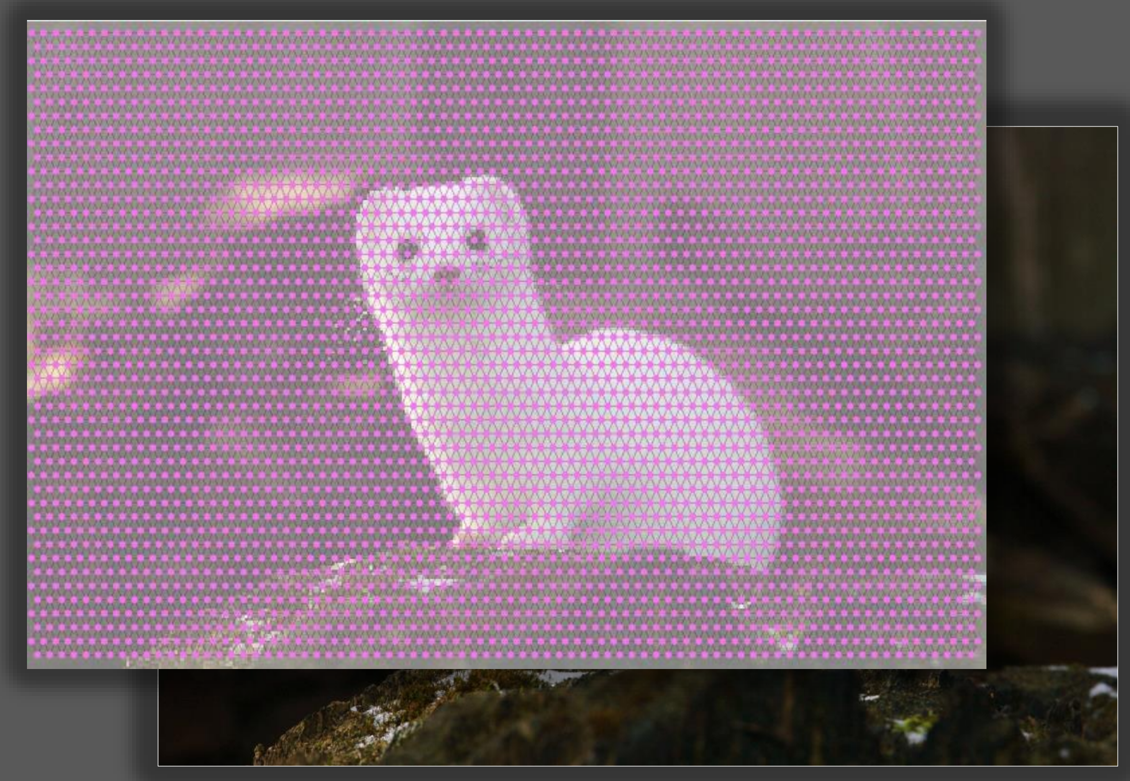
$O(V+E)$ running time, like DFS.

depth-first search
breadth-first search
Dijkstra's algorithm
Bellman-Ford algorithm
dynamic programming
Johnson's algorithm

Ford-Fulkerson algorithm
matchings
Prim's algorithm
Kruskal's algorithm
topological sort

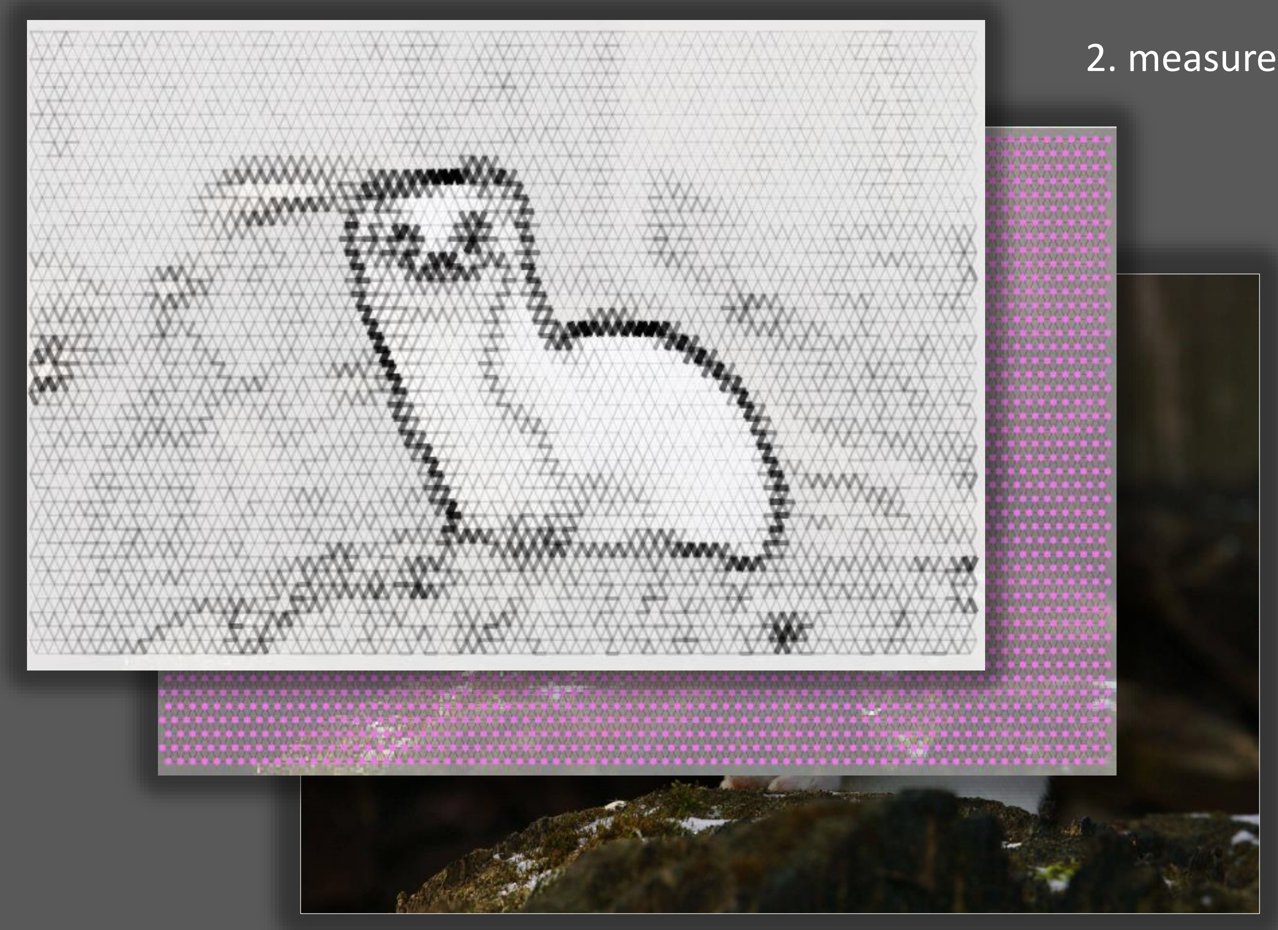
translation strategy

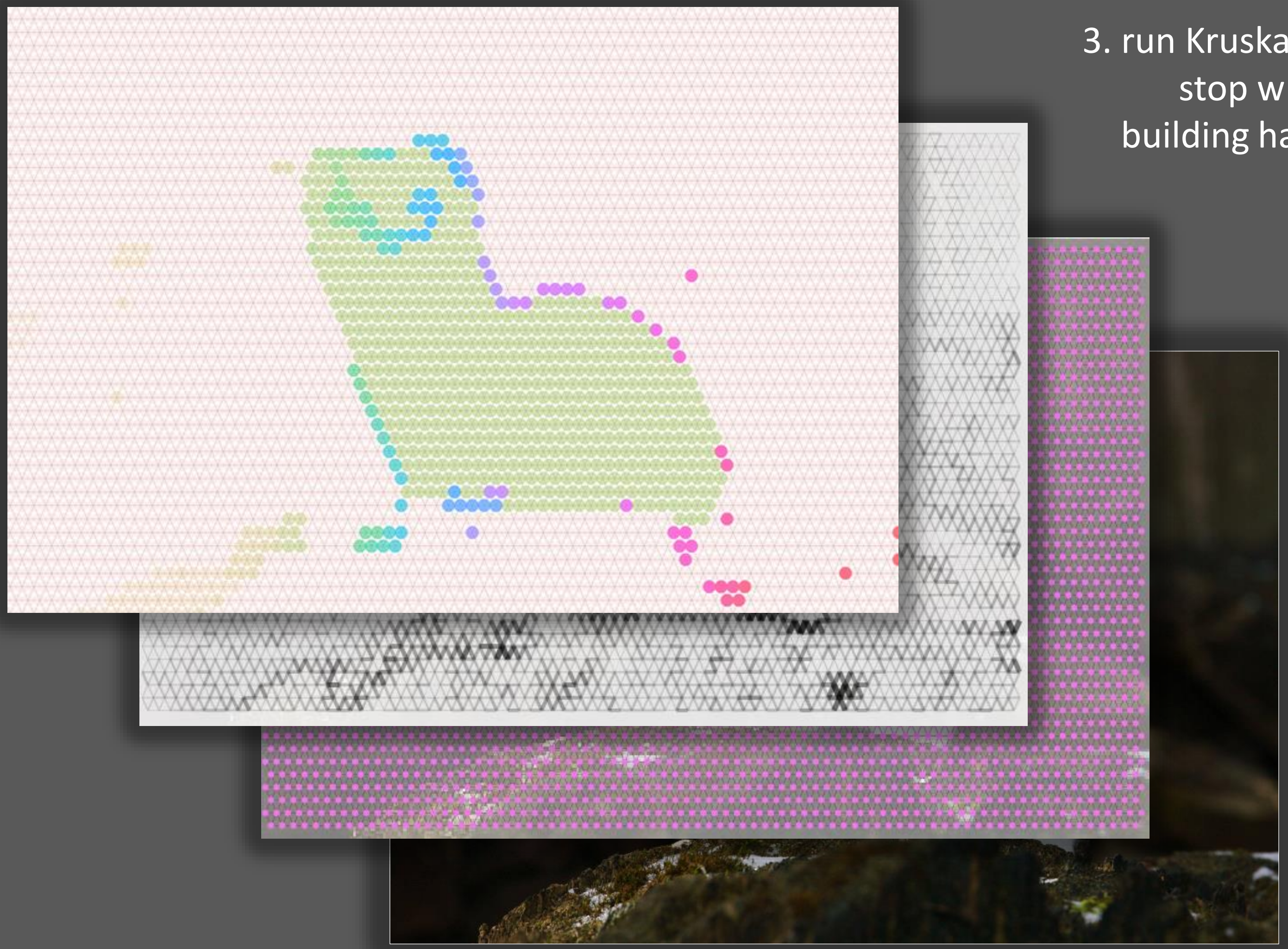
QUESTION. How might we segment this image into “handsome stoat” and “background”?



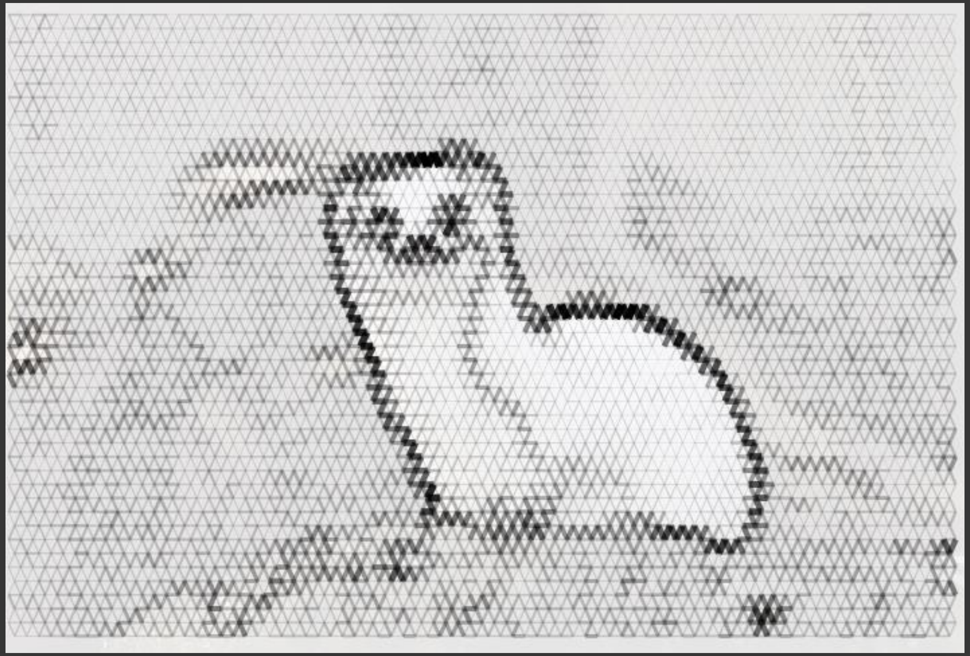
1. define a grid

2. measure dissimilarity along edges

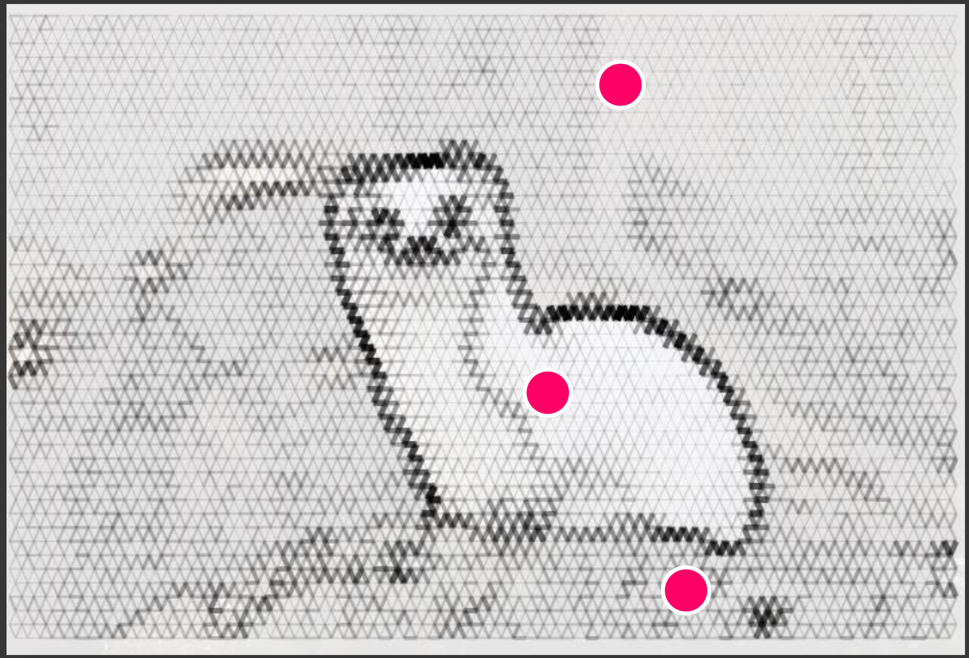




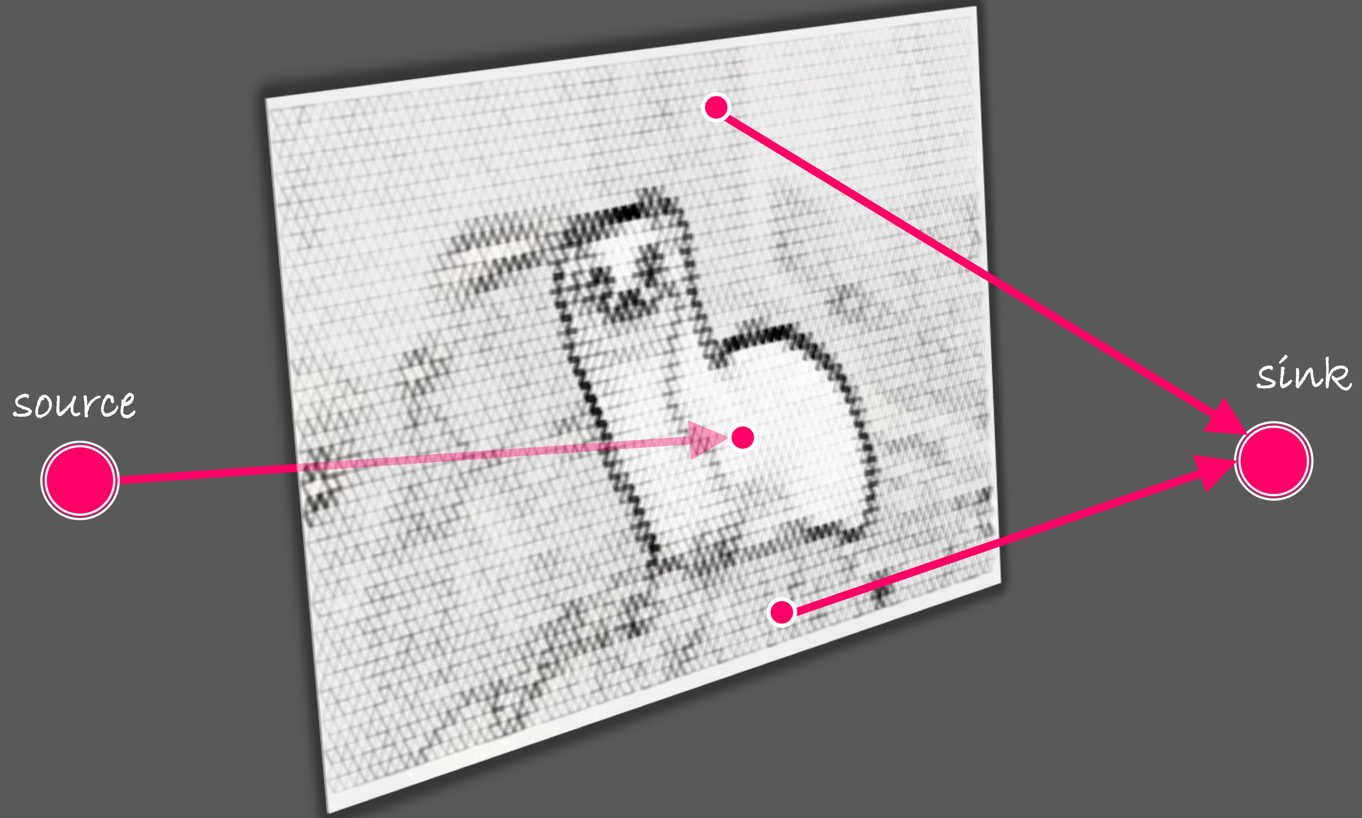
3. run Kruskal’s algorithm, and
stop when the forest it’s
building has just a few trees



2. measure dissimilarity along edges

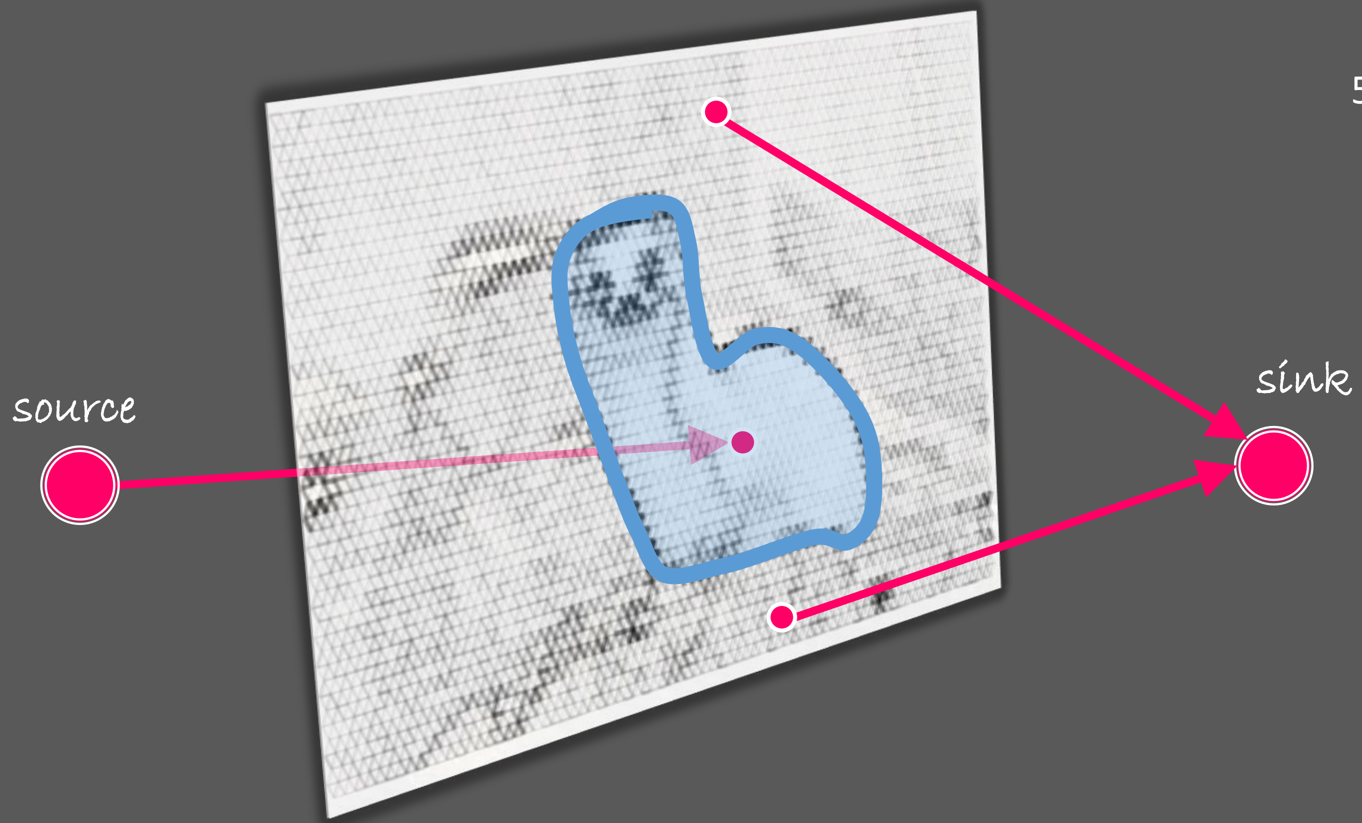


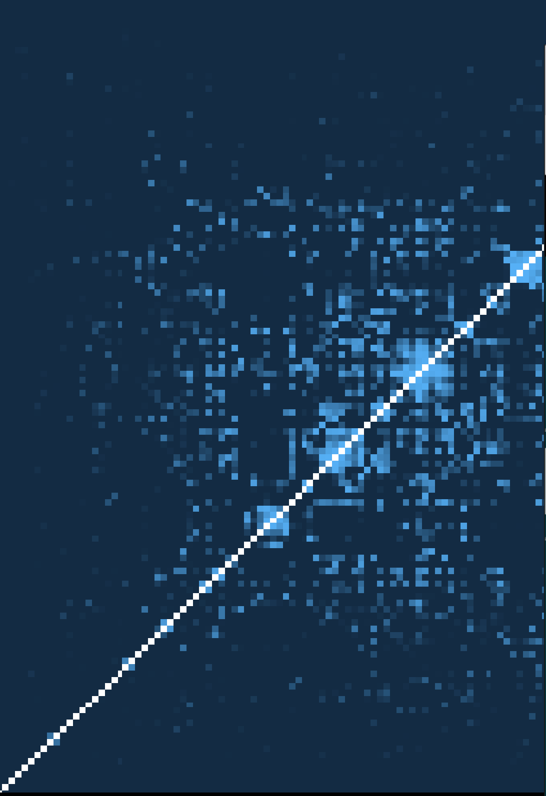
3. ask the user to label some “stoat” points and some “background” points



4. set up a flow network

5. find a minimum-capacity cut





Algorithms 2 2022-23 rank-sim S

vle.cam.ac.uk/mod/vpl/views/submissionslist.php?id=18069212&group=-1&...

UNIVERSITY OF CAMBRIDGE

		First name / Surname	Submitted on	Submissions	Grade	Evaluator	Evaluated on
1		Kevin Xie	Sunday, 5 March 2023, 5:06 PM	22	76.76 / 100.00 (76.75909009751997)	Automatic grade	Sunday, 5 March 2023, 5:07 PM
2		Matej Urban	Sunday, 5 March 2023, 5:33 PM	2	68.70 / 100.00 (68.69996961985618)	Automatic grade	Sunday, 5 March 2023, 5:33 PM
3		Milos Puric	Wednesday, 1 March 2023, 1:03 AM	10	65.41 / 100.00 (65.41160210284743)	Automatic grade	Wednesday, 1 March 2023, 1:03 AM
4		Katy Thackray	Friday, 3 March 2023, 2:24 PM	1	65.18 / 100.00 (65.18324383627447)	Automatic grade	Friday, 3 March 2023, 2:24 PM
5		Elizabeth Ho	Sunday, 5 March 2023, 3:59 PM	2	64.98 / 100.00 (64.97779892836748)	Automatic grade	Sunday, 5 March 2023, 3:59 PM
6		Paul DSouza	Sunday, 5 March 2023, 3:38 PM	5	0.00 / 100.00	Automatic grade	Sunday, 5 March 2023, 3:38 PM
7		George Ogden	Saturday, 4 March 2023, 1:45 PM	5	0.00 / 100.00	Automatic grade	Sunday, 5 March 2023, 9:12 AM