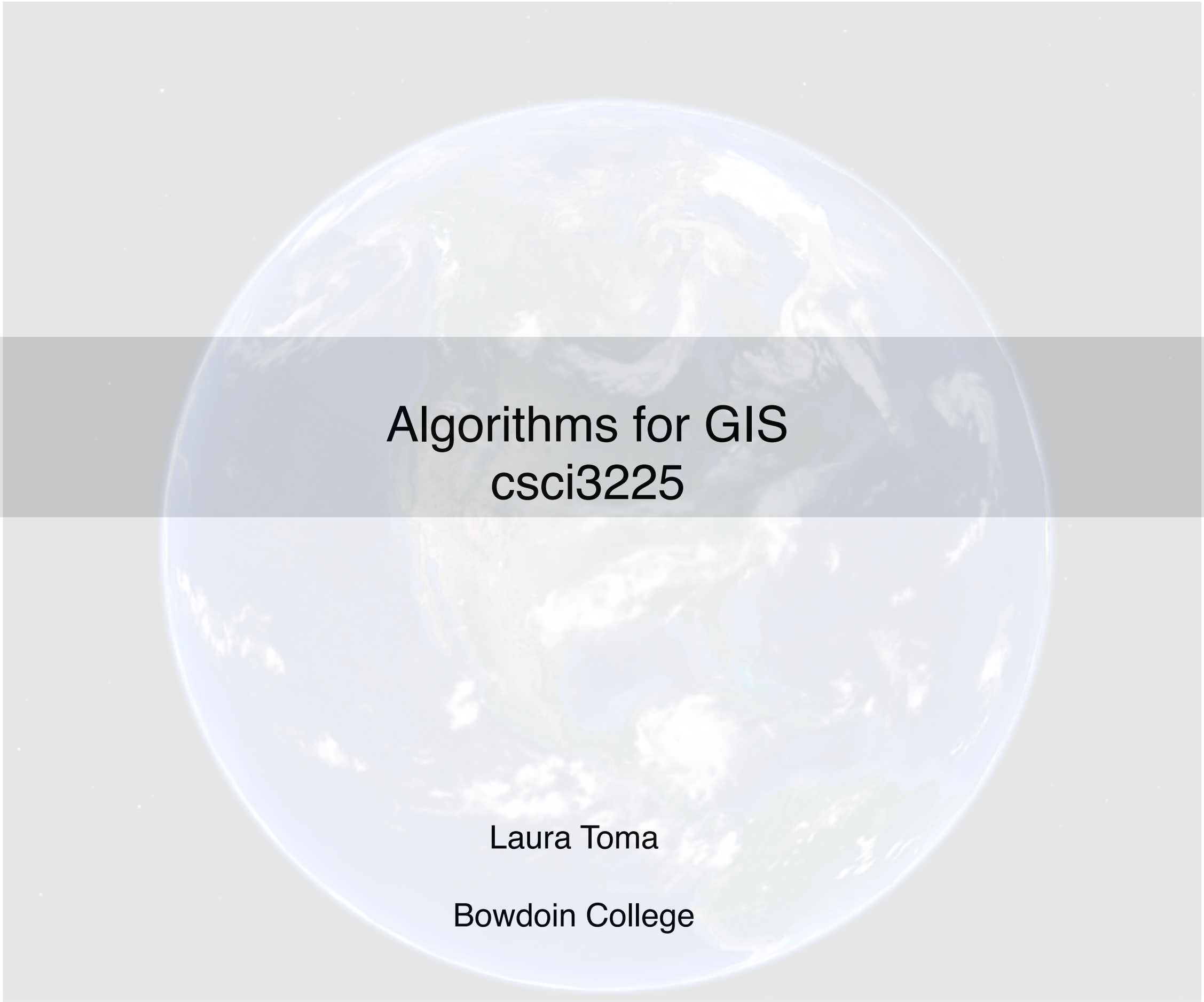


Algorithms for GIS
csci3225

Laura Toma

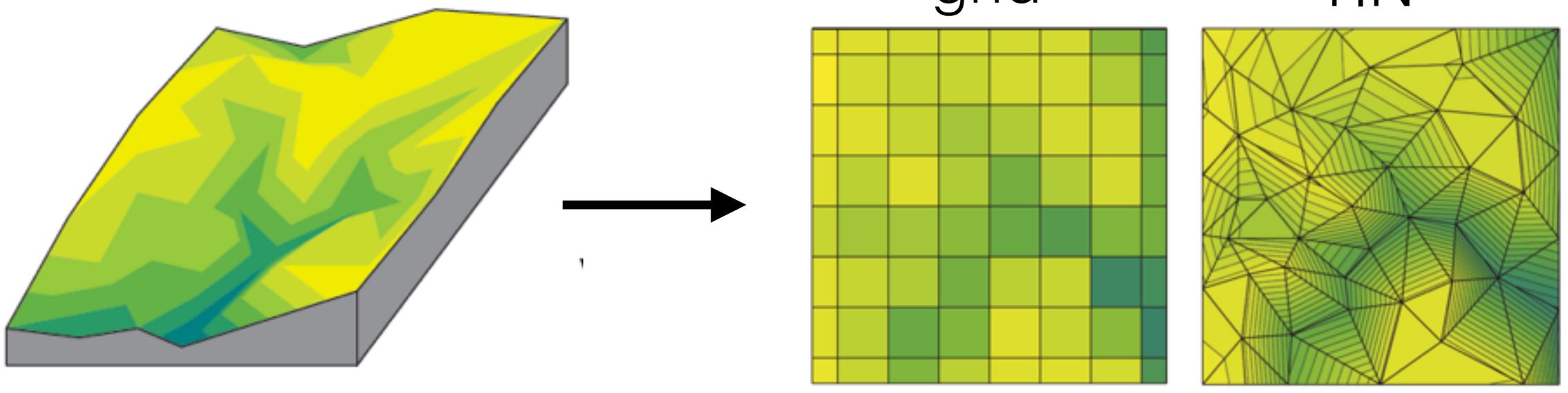
Bowdoin College

Flow on digital terrain models (I)

Flow

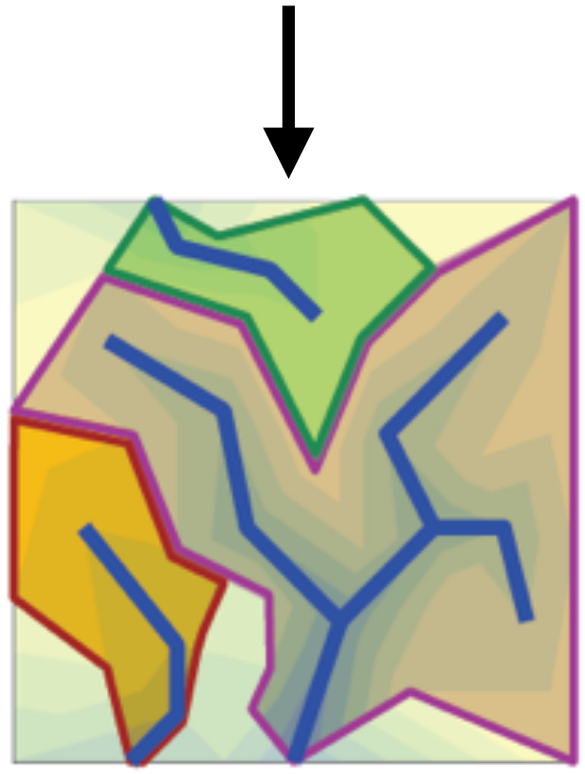
- Where does the water go when it rains?
- **Flooding**: What are the areas susceptible to flooding?
- **Sea level rise:** What will go under water when sea level rises by e.g. 10 ft?
- **River network:** Model and “compute” rivers on a terrain?
- **Catchments:** What area drains to a point?
- **Watersheds and watershed hierarchy:** What are the sub-basins of the Amazon?
- Many other processes can be modeled with flow
    - E.g.: Suppose a pollutant is spilled on a terrain: what area will be contaminated (when it rains)) ?

Flow on digital terrain models



Outline:

- Flow direction
- Flow accumulation
- Flat areas
- Catchments
- Watersheds and watershed hierarchy
- Sea-level rise

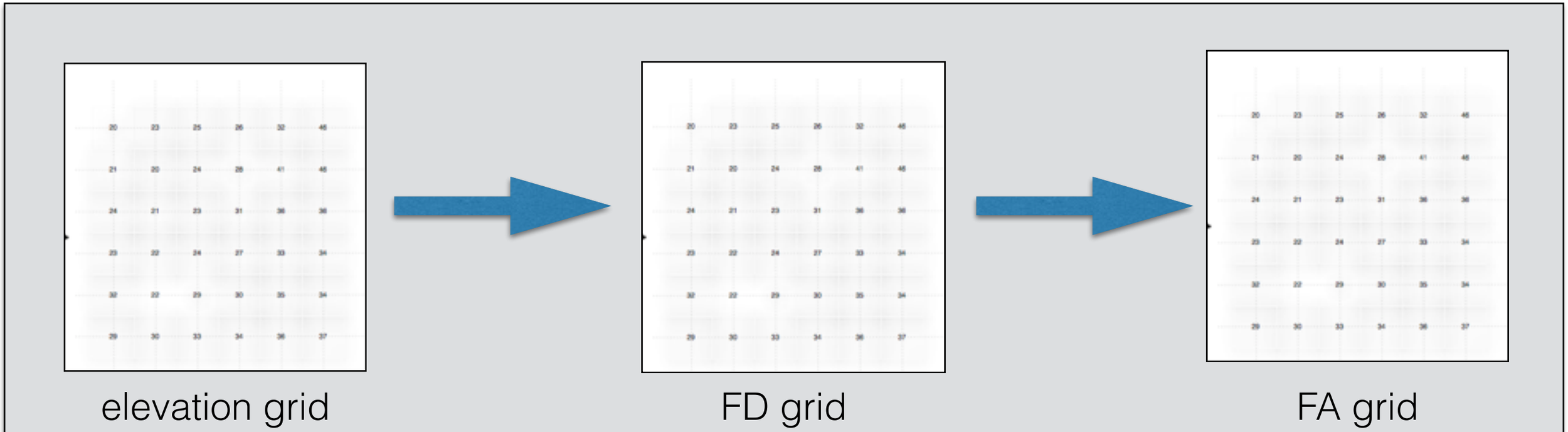


images from H. Haverkort

Flow on grid terrains

Modeled by two basic functions

- **Flow direction (FD):** the direction water flows at a point
- **Flow accumulation (FA):** total amount of water flowing through a point

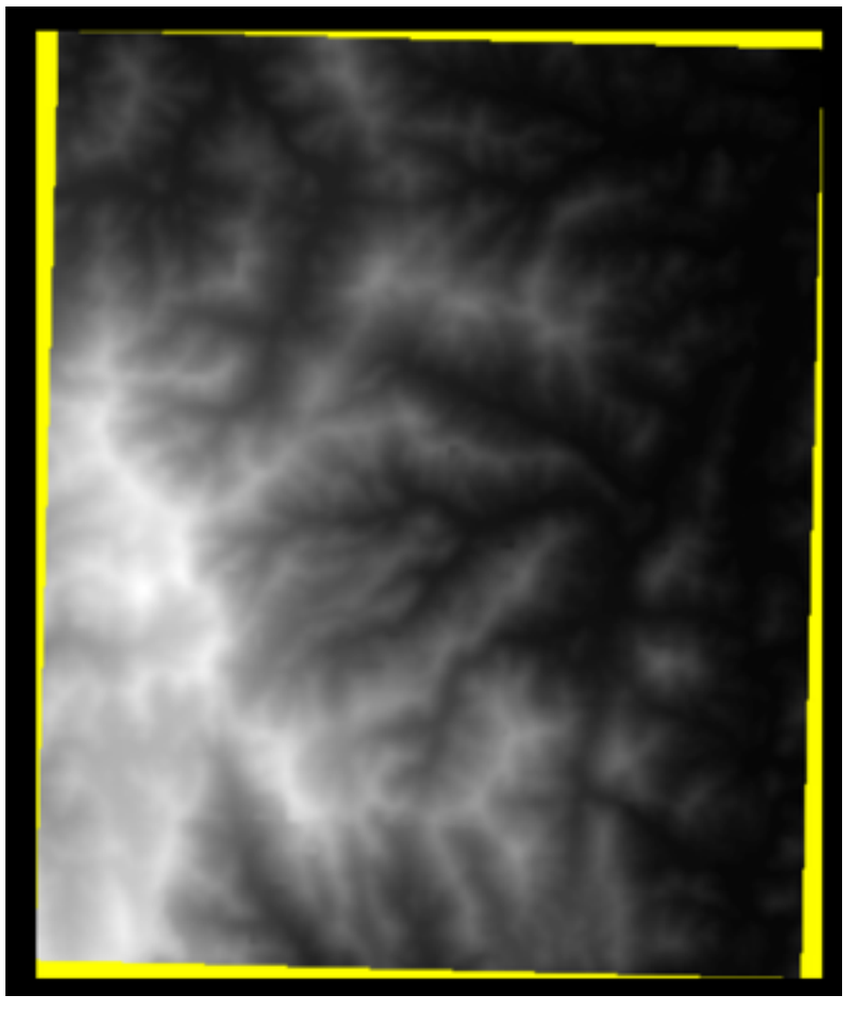


Flow on grid terrains

Modeled by two basic functions

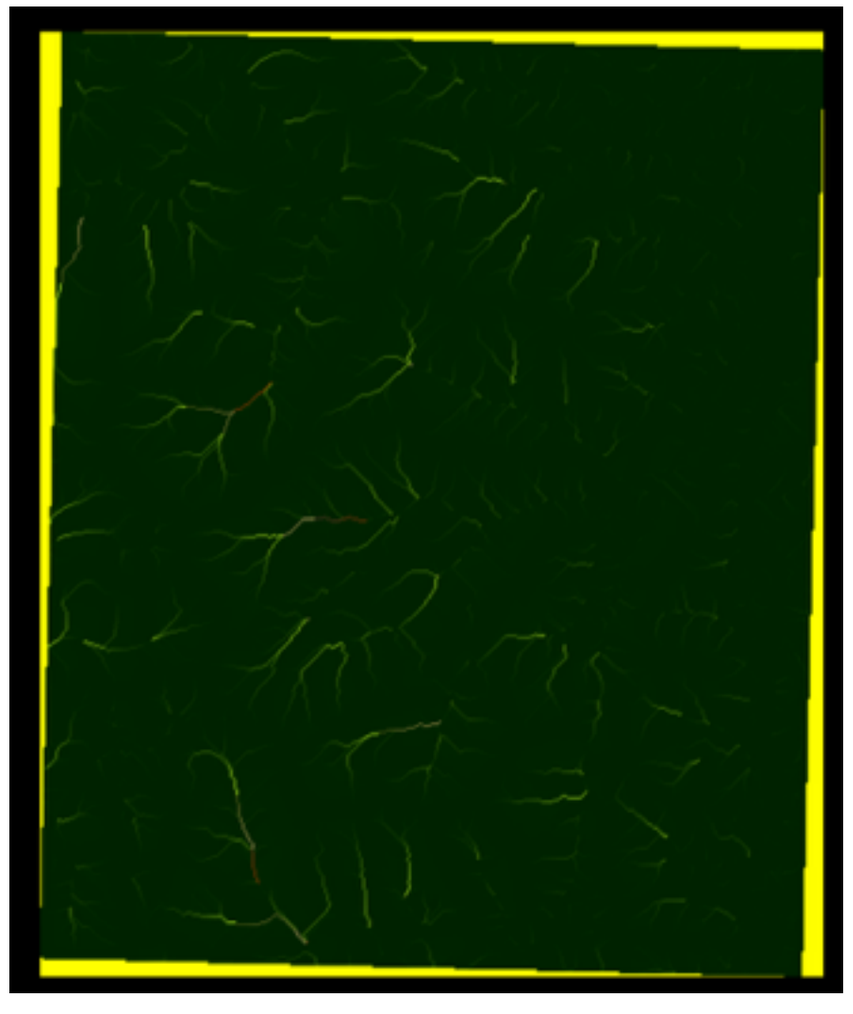
- **Flow direction (FD):** the direction water flows at a point
- **Flow accumulation (FA):** total amount of water flowing through a point

elev grid



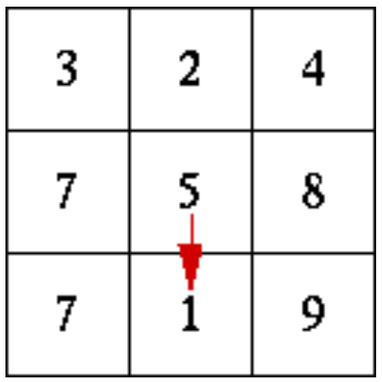
FD grid

FA grid



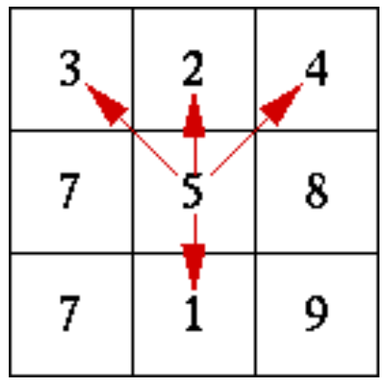
Flow direction (FD)

- FD(p) = the direction water flows at p
- Generally, FD is direction of gradient at p (direction of greatest decrease)
- FD can be approximated based on the neighborhood of p
- FD on grids:
    - discretized to eight directions (8 neighbors), multiple of 45^{o}



SFD: Single flow direction

(steepest downslope)



MFD: Multiple flow directions

(all downslope neighbors)

78	72	69	71	58	49
74	67	56	49	46	50
69	53	44	37	38	48
64	58	55	22	31	24
68	61	47	21	16	19
74	53	34	12	11	12

Elevation surface

2	2	2	4	4	8
2	2	2	4	4	8
1	1	2	4	8	4
128	128	1	2	4	8
2	2	1	4	4	4
1	1	1	1	4	16

Flow direction

Direction coding

The coding of the direction of flow

```
//return the flow direction of cell (r,c)

int flowdir(int r, int c) {

   …

}
```

Flat areas and pits ?? later

78	72	69	71	58	49
74	67	56	49	46	50
69	53	44	37	38	48
64	58	55	22	31	24
68	61	47	21	16	19
74	53	34	12	11	12

Elevation surface

2	2	2	4	4	8
2	2	2	4	4	8
1	1	2	4	8	4
128	128	1	2	4	8
2	2	1	4	4	4
1	1	1	1	4	16

Flow direction

Direction coding

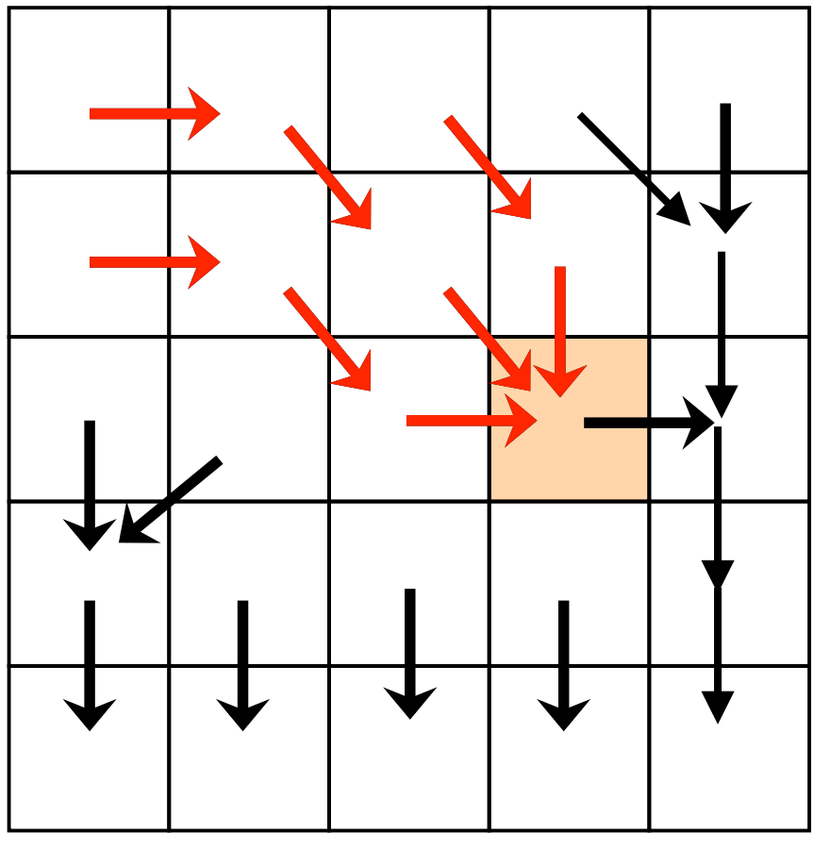
The coding of the direction of flow

```
//compute FD grid
for r = 0 to nrows
   for c = 0 to ncols
        FD[r][c] = flowdire(r,c)
```

O(n) time

Flow direction as a graph

- FD graph cannot have cycles (water goes downhill)
- For single flow direction (SFD): FD is a set of directed trees
- Each tree represents a separate “river tree”



Flow accumulation (FA)

- Assume each cell starts with 1 unit of water
- Assume each cell sends its initial as well as incoming water to the neighbor pointed to by its FD
- A(p) = how much water goes through cell p

Flow accumulation (FA)

- Assume each cell starts with 1 unit of water
- Assume each cell sends its initial as well as incoming water to the neighbor pointed to by its FD
- A(p) = how much water goes through cell p

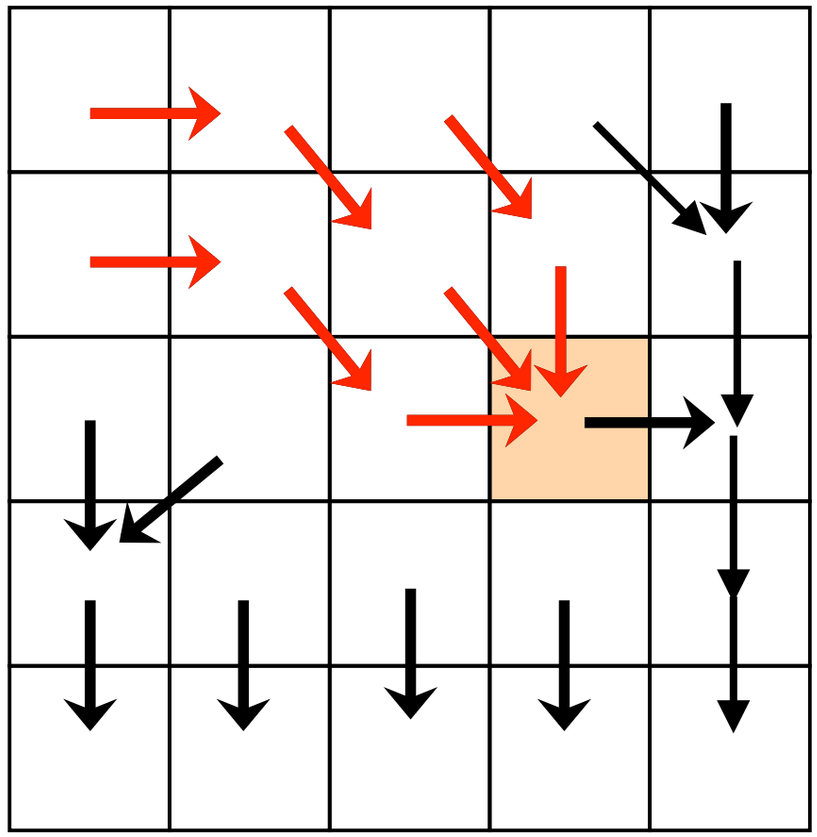
Show the rest of the FA grid

FD grid

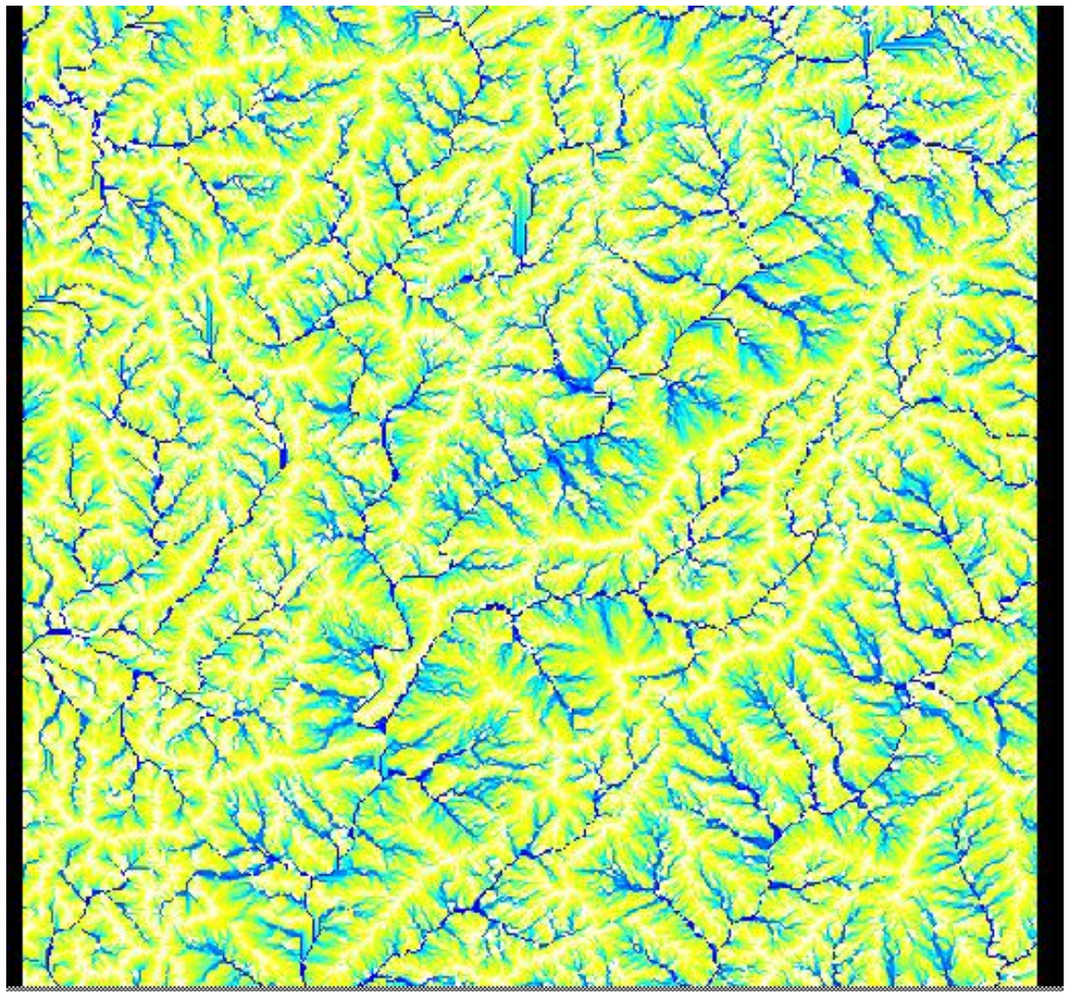
FA grid

Flow accumulation (FA)

- Points with small FA= ridges
- Points with high FA = channels (rivers)
- FA models rivers
  - set an arbitrary threshold t
  - cell c is on a river if FA(c) >= t

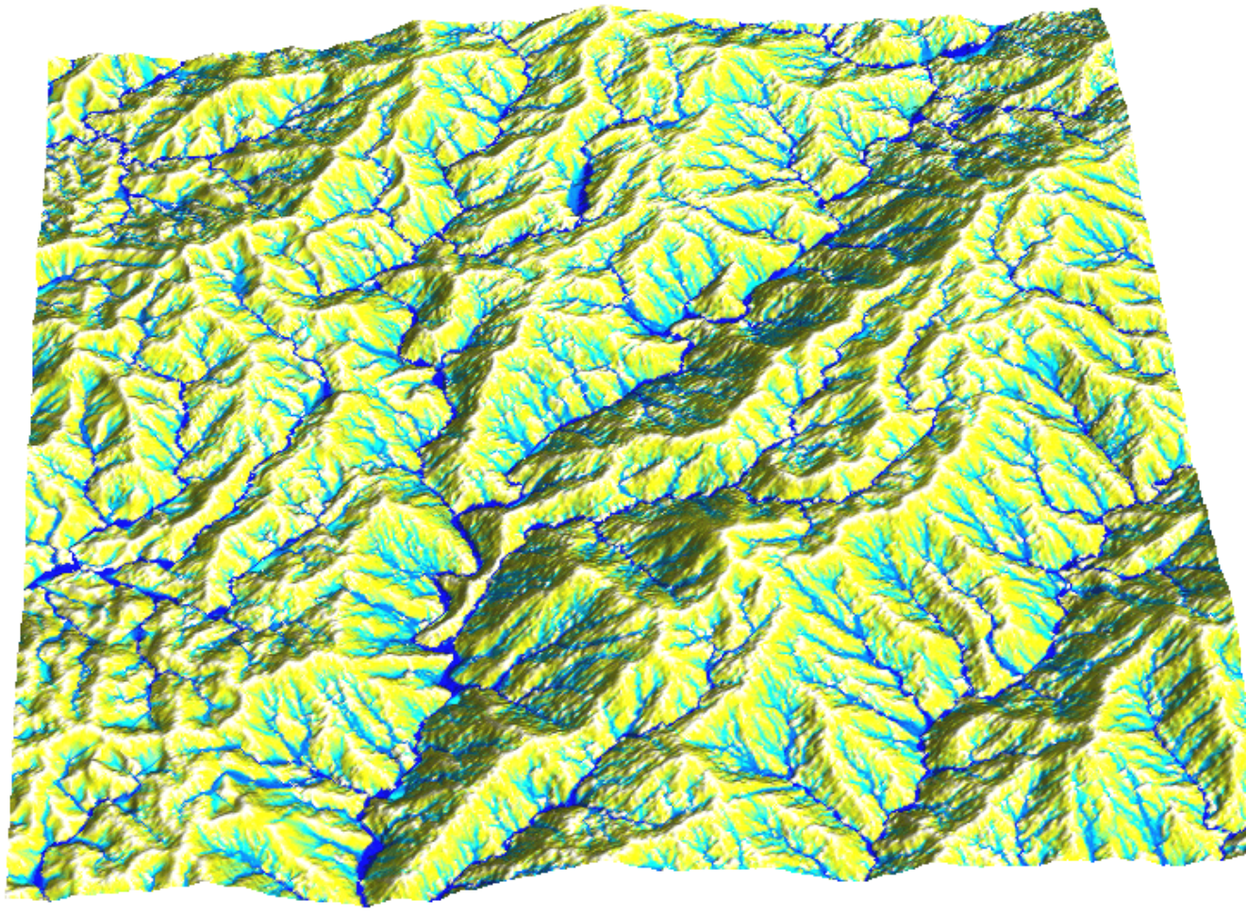


Flow accumulation (FA)



FA 2D view

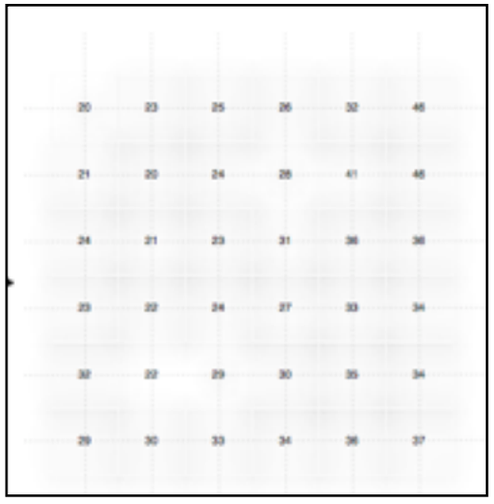
- high values: blue
- medium values: light blue
- low values: yellow



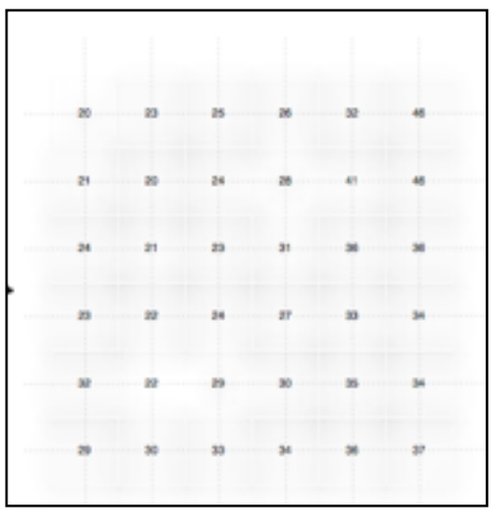
FA grid draped over elevation grid

Computing FA grid

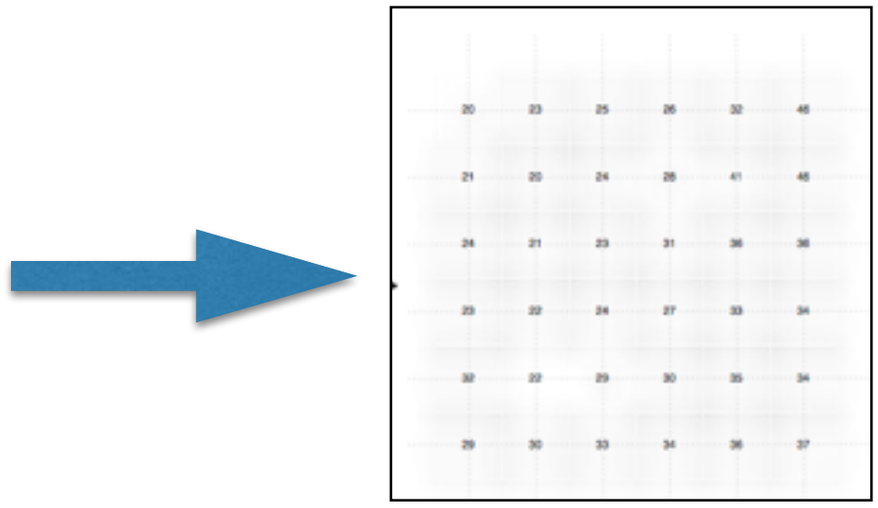
Given an elevation grid, and a FD grid, compute the FA grid



elevation grid



FD grid



FA grid

FD grid can be given explicitly or implicitly, via a function:

```
//return the flow direction of cell (r,c)
int flowdir(int r, int c)
```

32	64	128
16		1
8	4	2

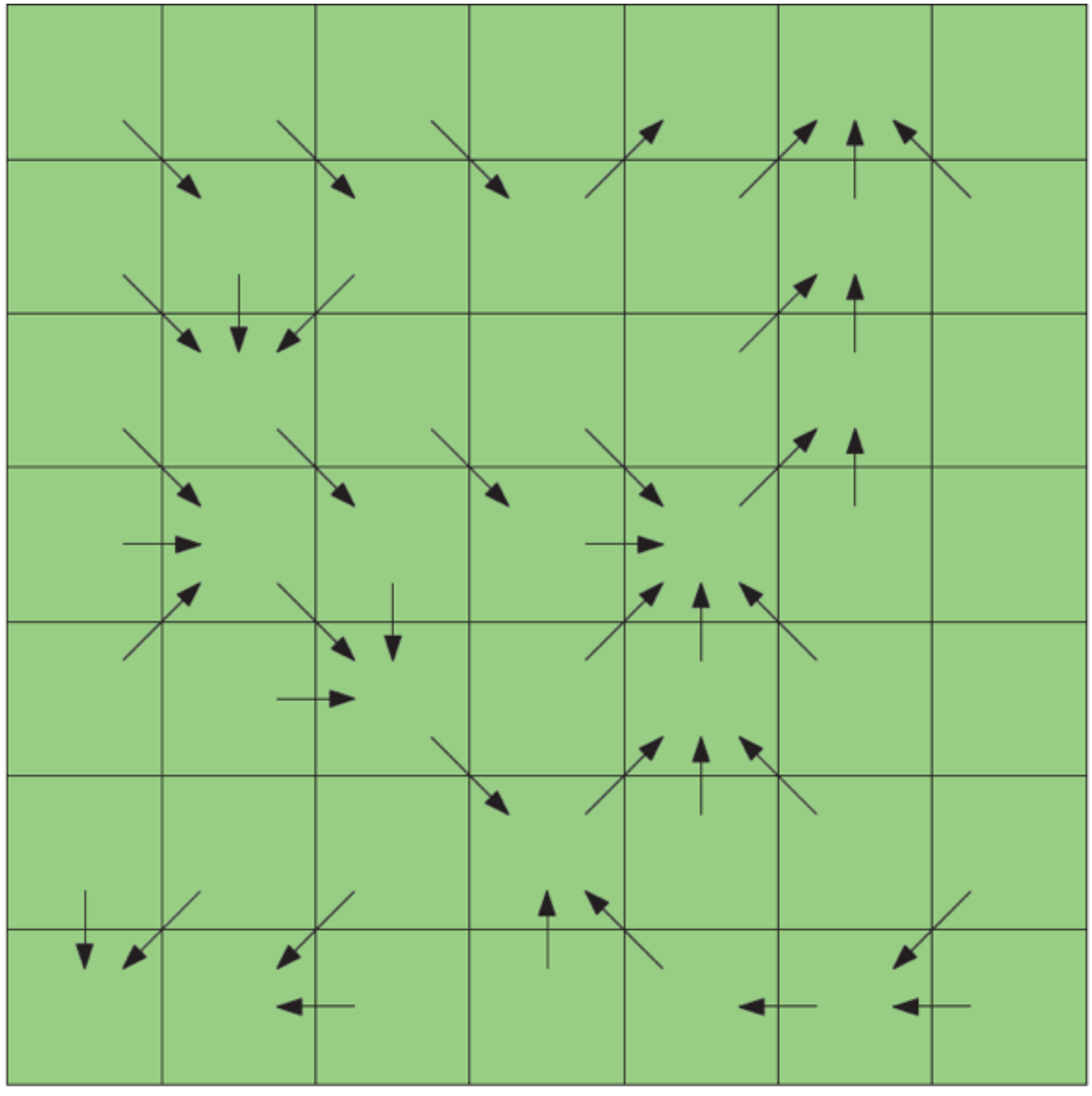
Computing FA grid

Given an elevation grid, and a FD grid, compute the FA grid

- Idea 1:
  - Scan row-by-row: for each cell, add +1 to all cells on its downstream path
- Idea 2:
  - Flow at cell c is the sum of the flows of the neighbors that flow into c
  - Use recursion
  - Do this for every cell
- Or..?

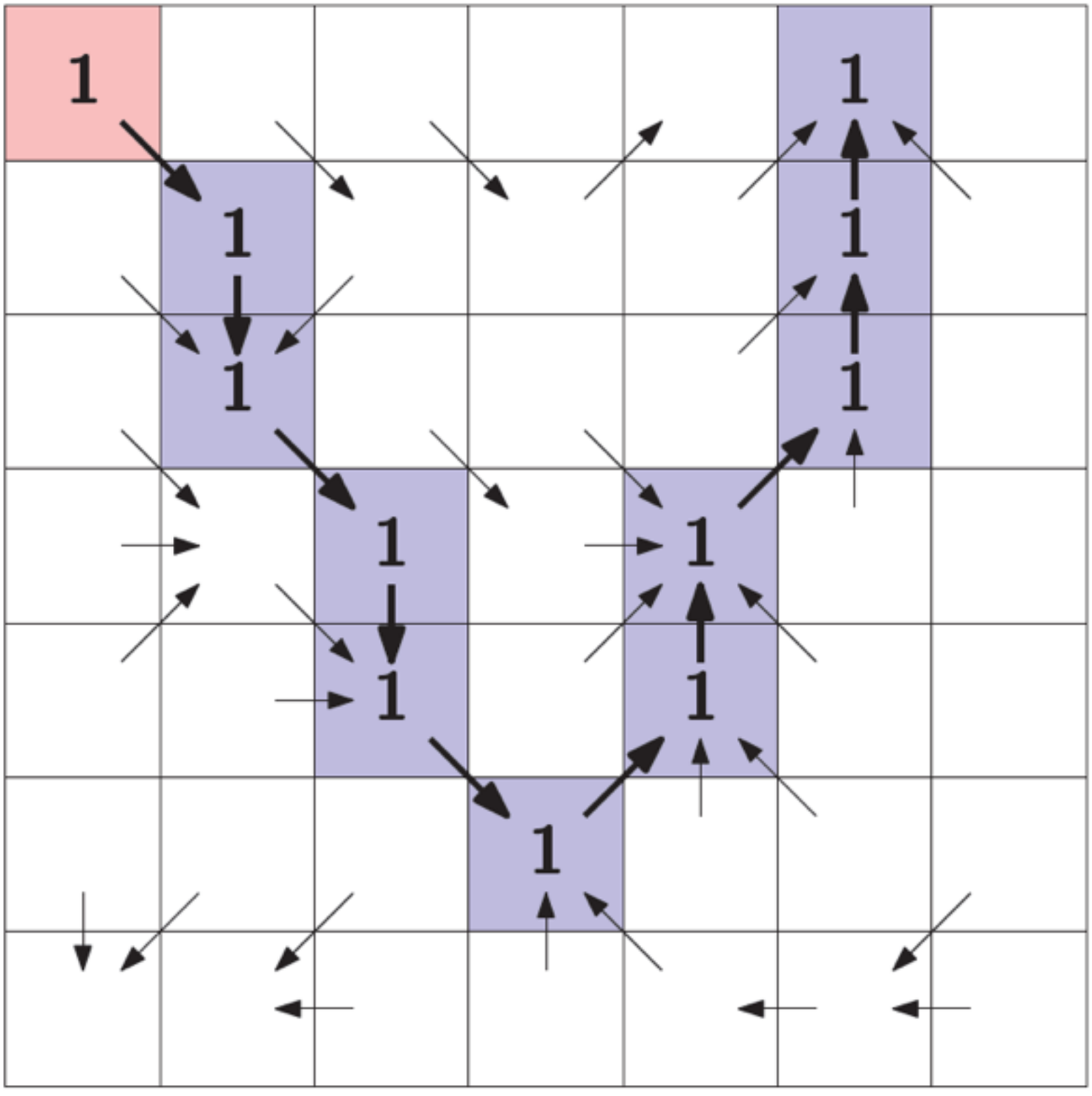
Not straightforward to analyze

Computing FA: naive algorithm (1)



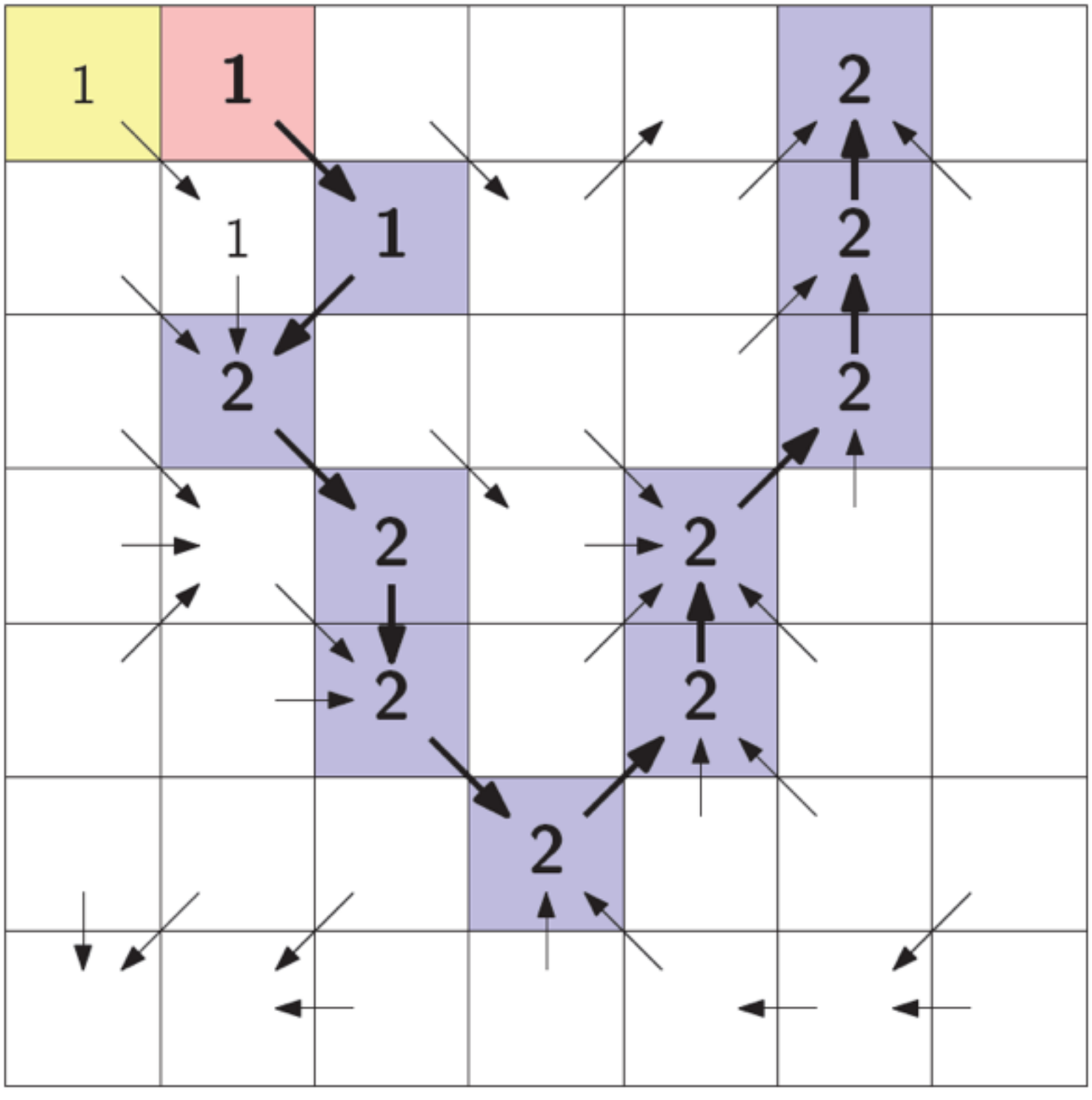
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



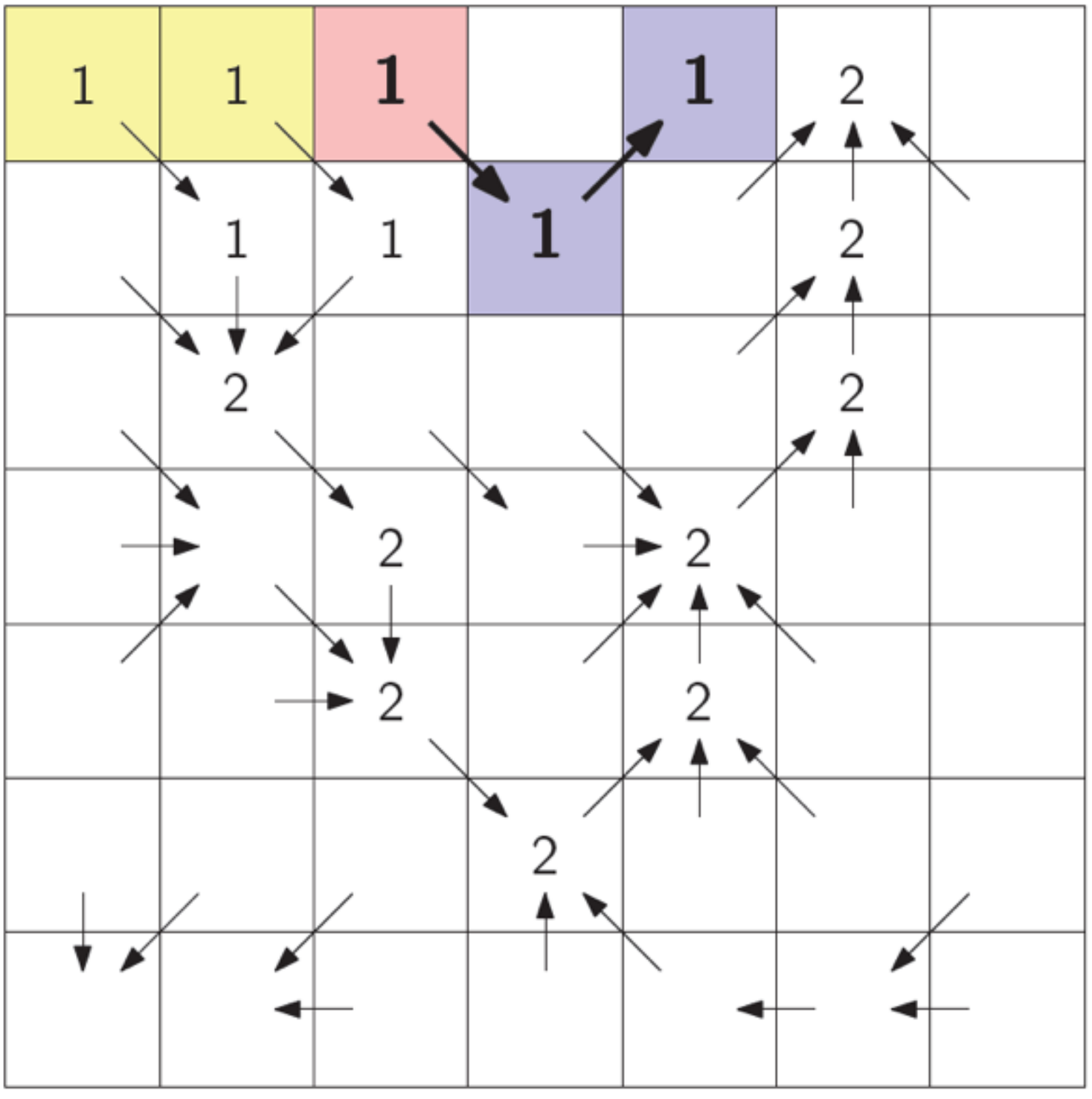
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



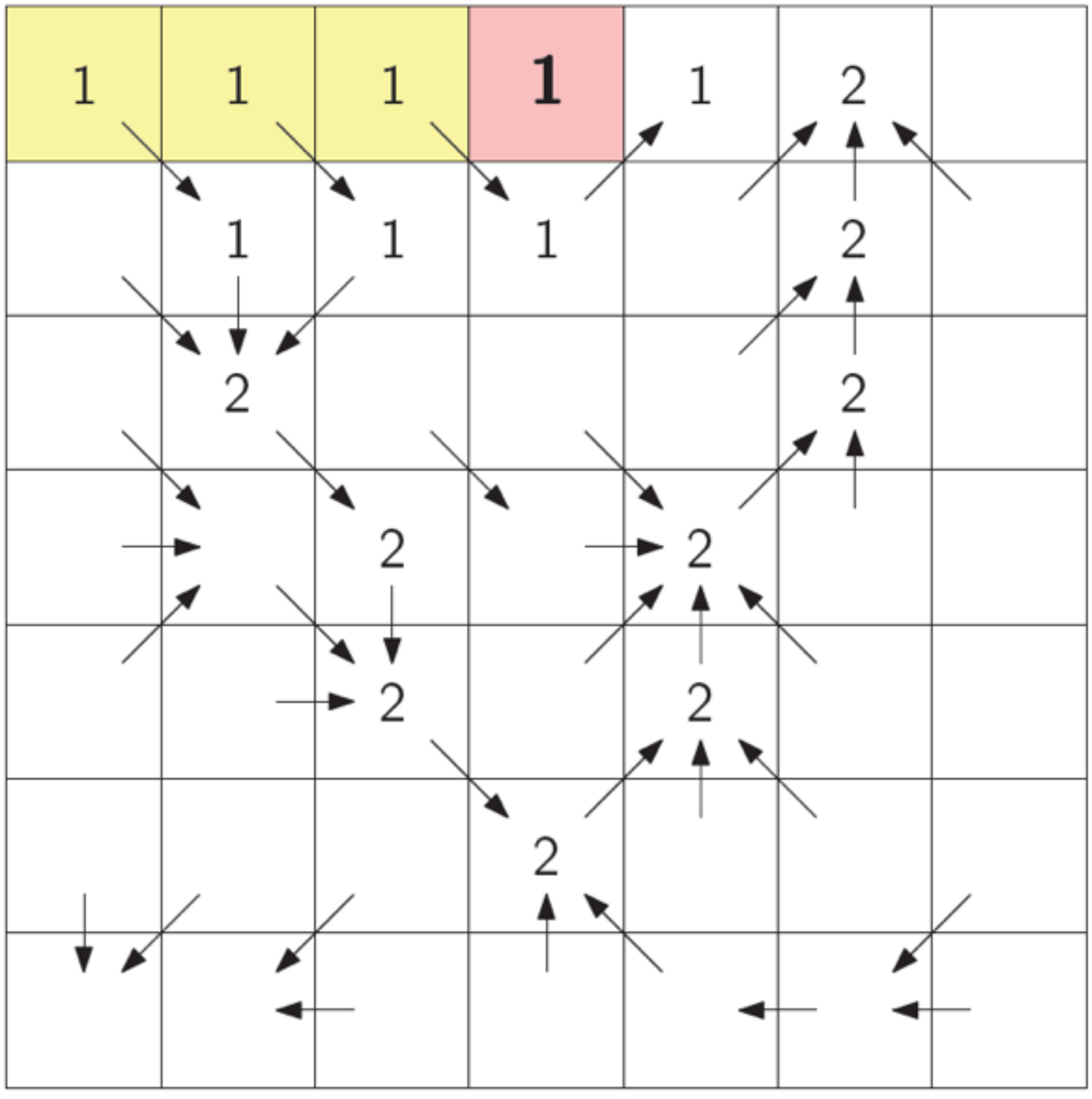
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



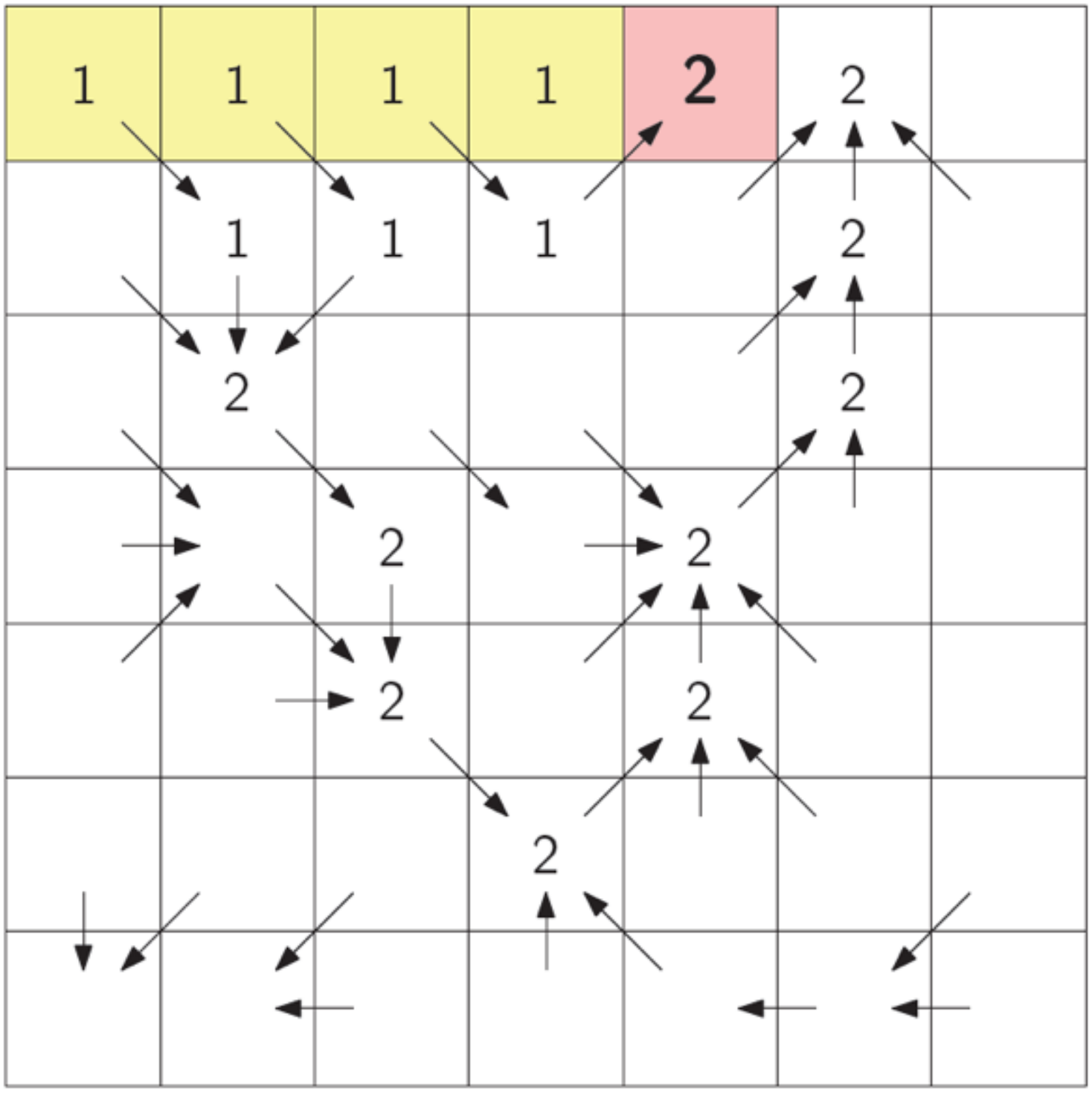
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



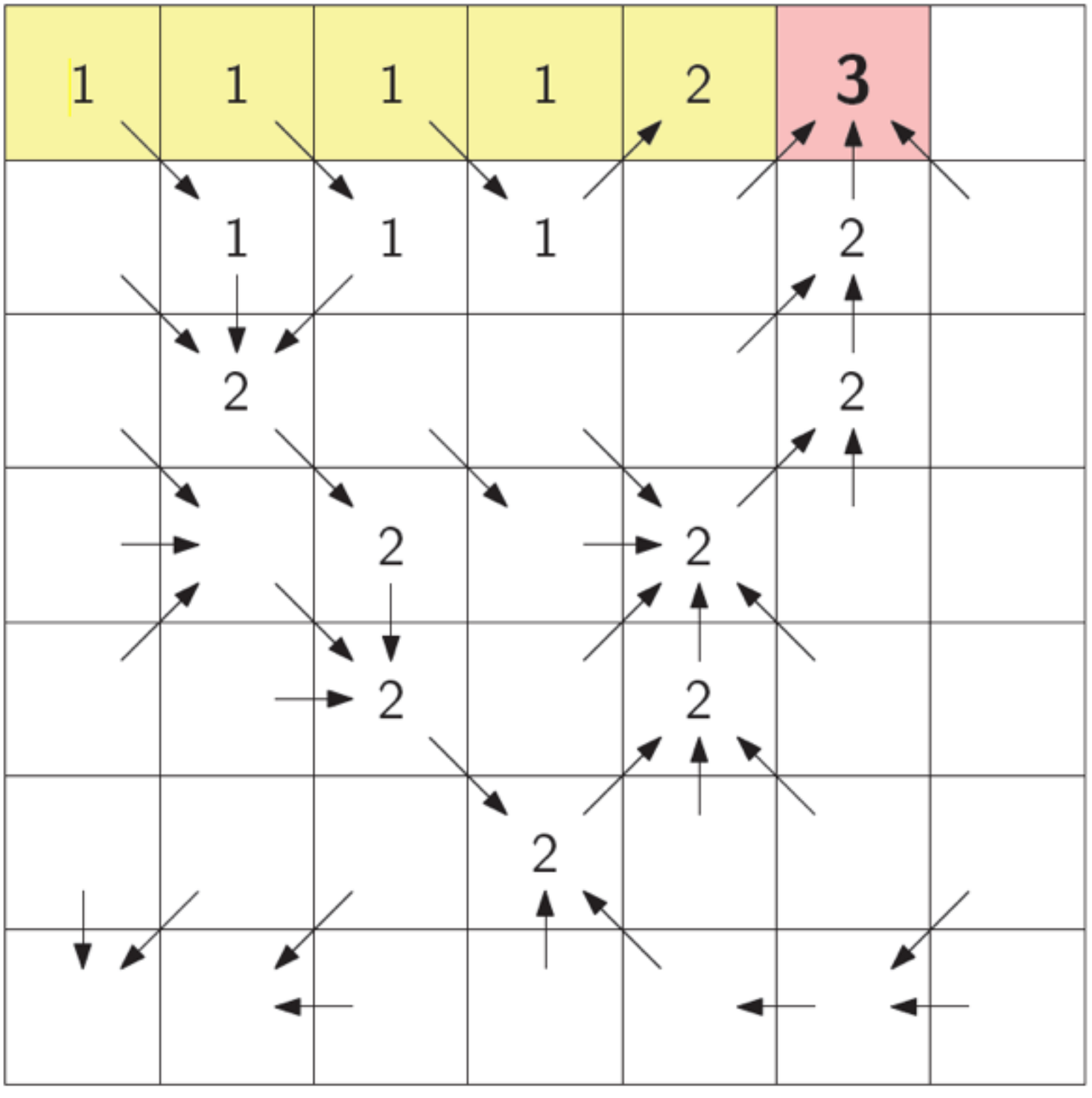
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



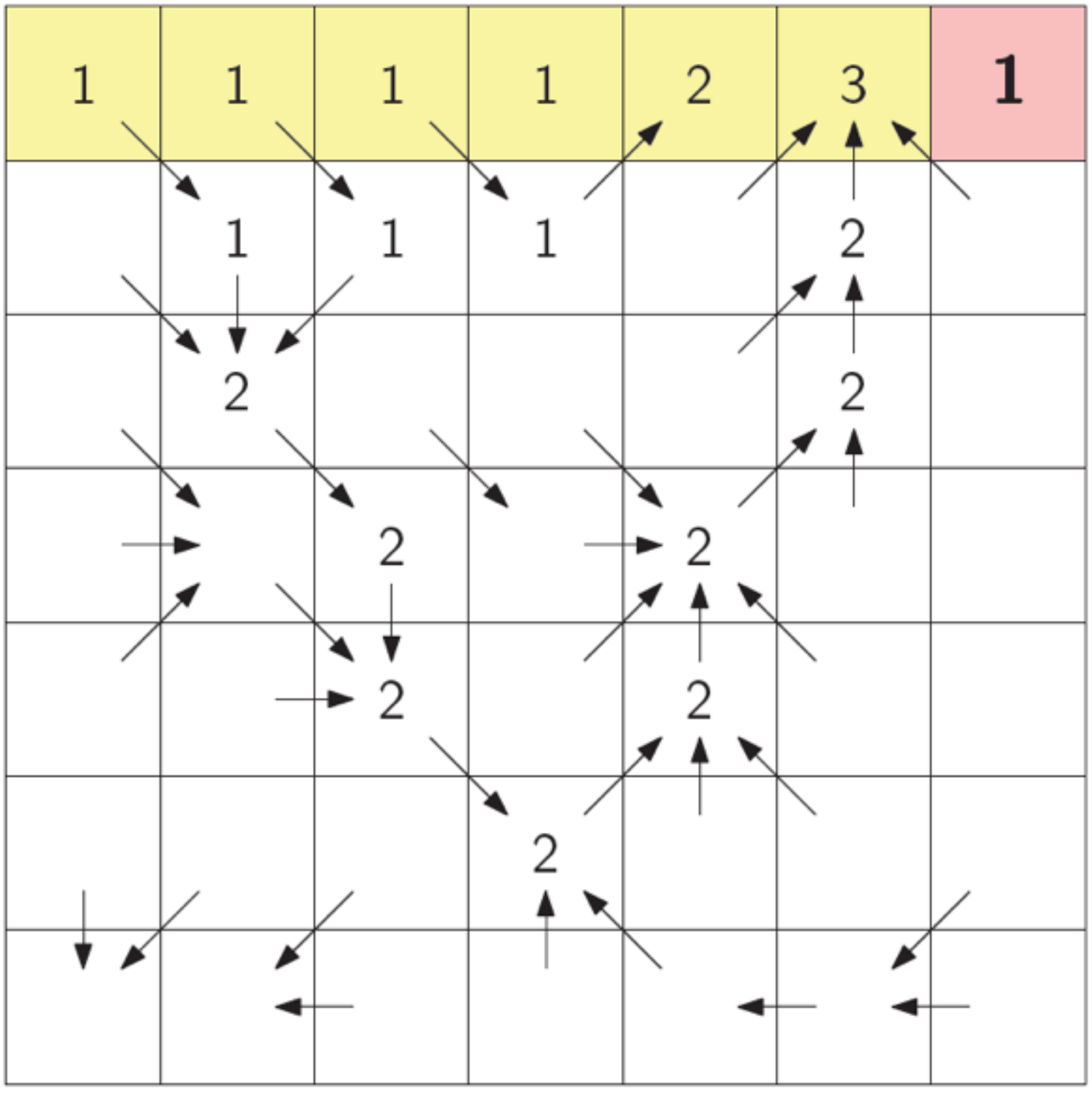
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



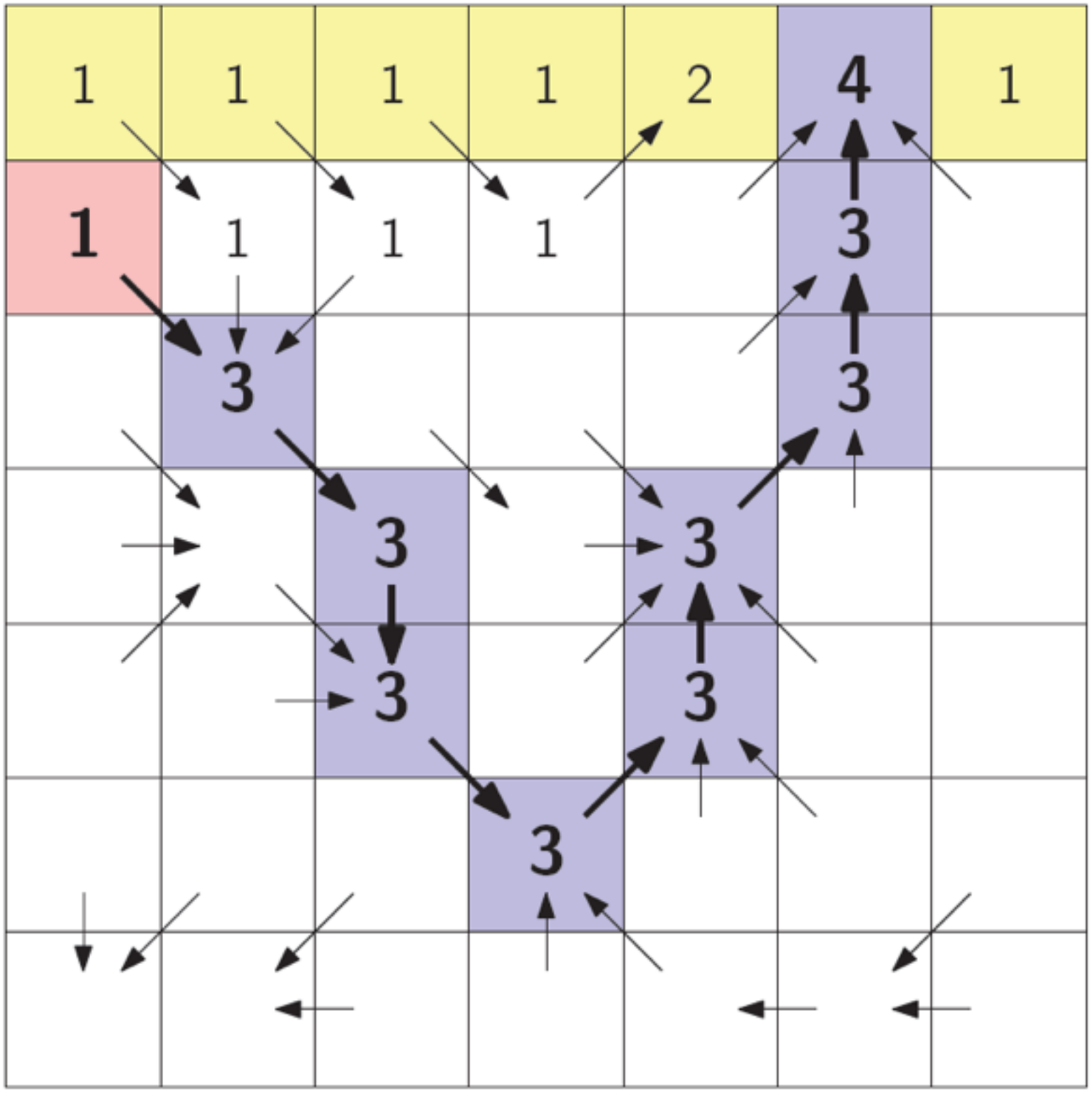
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



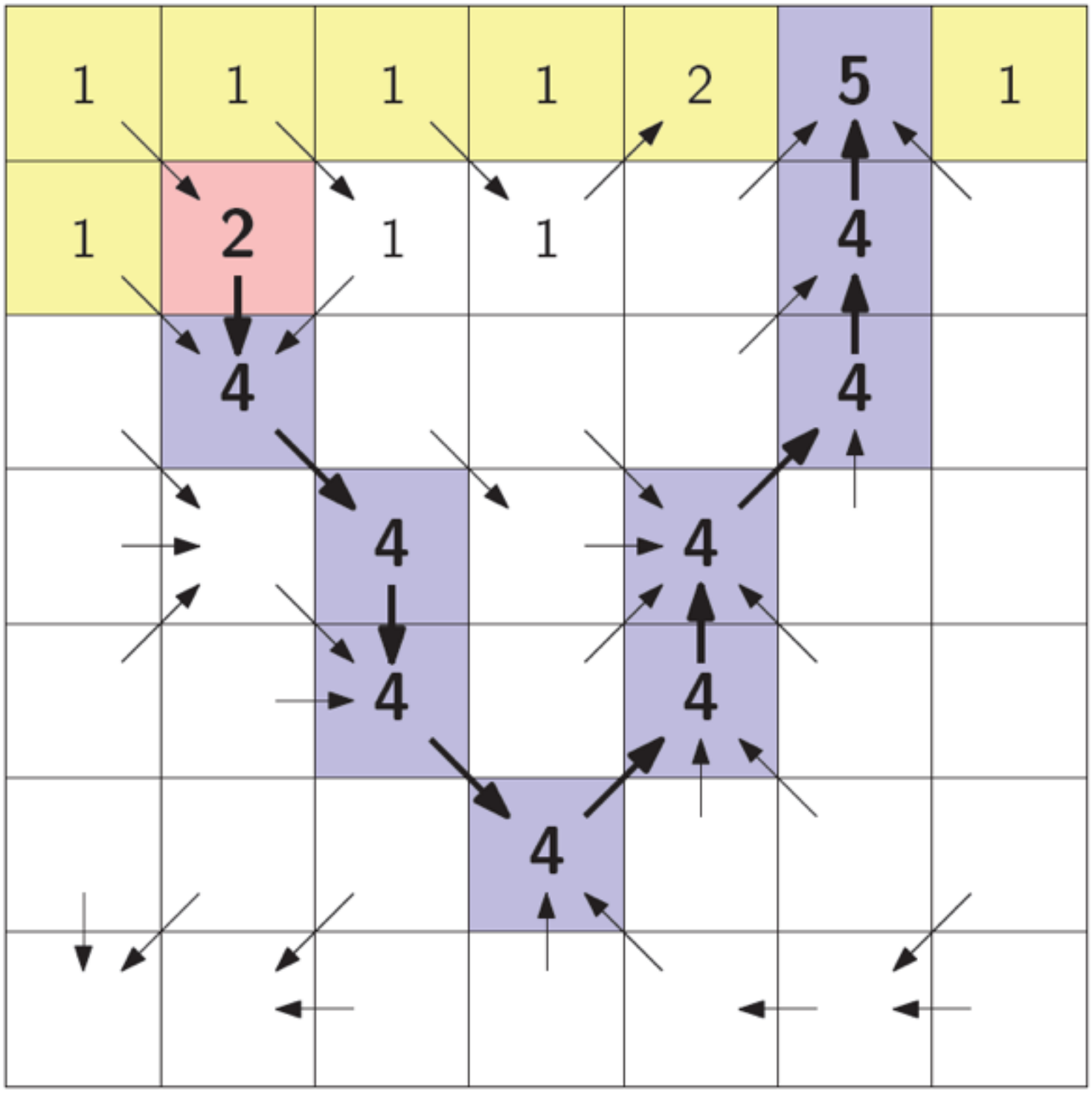
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



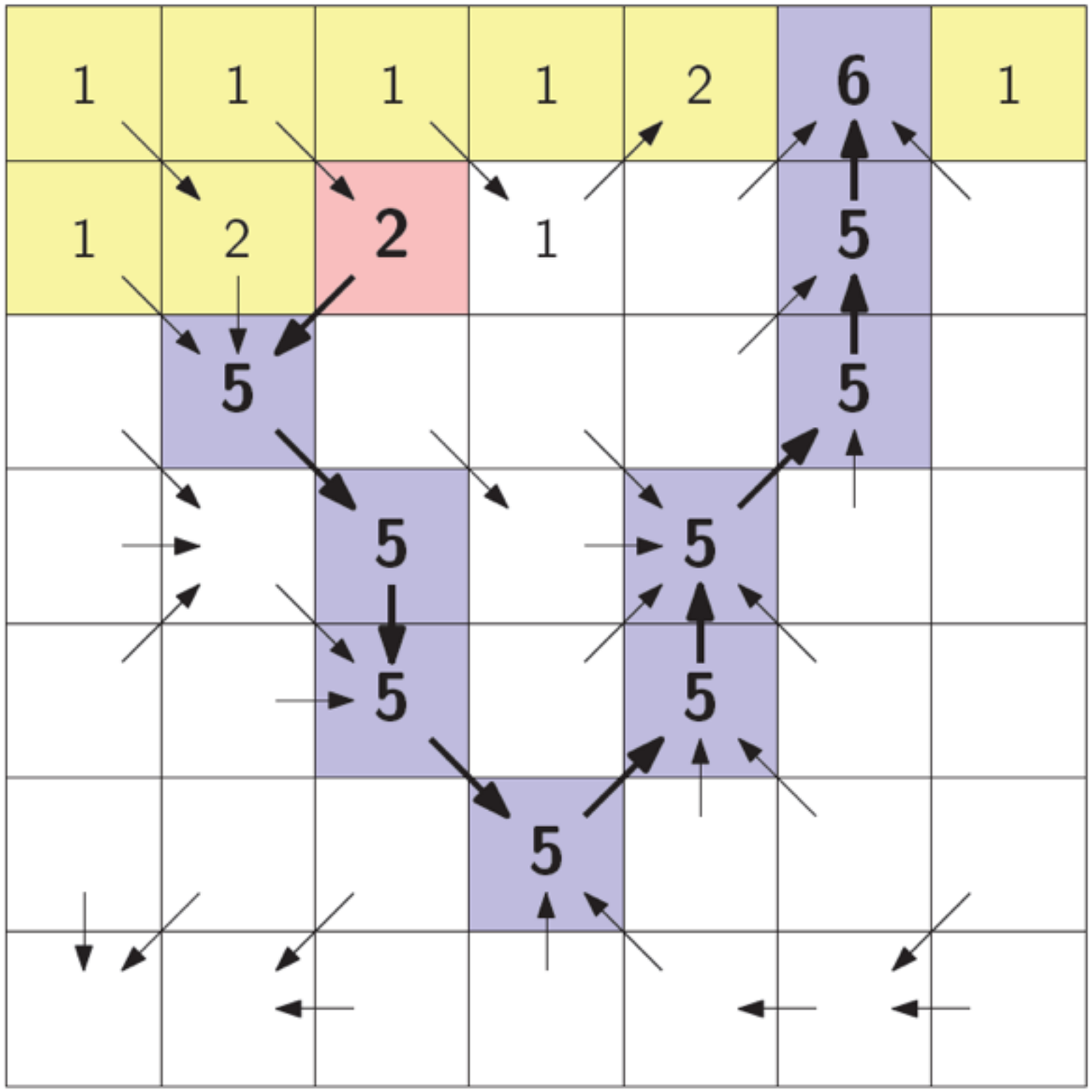
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



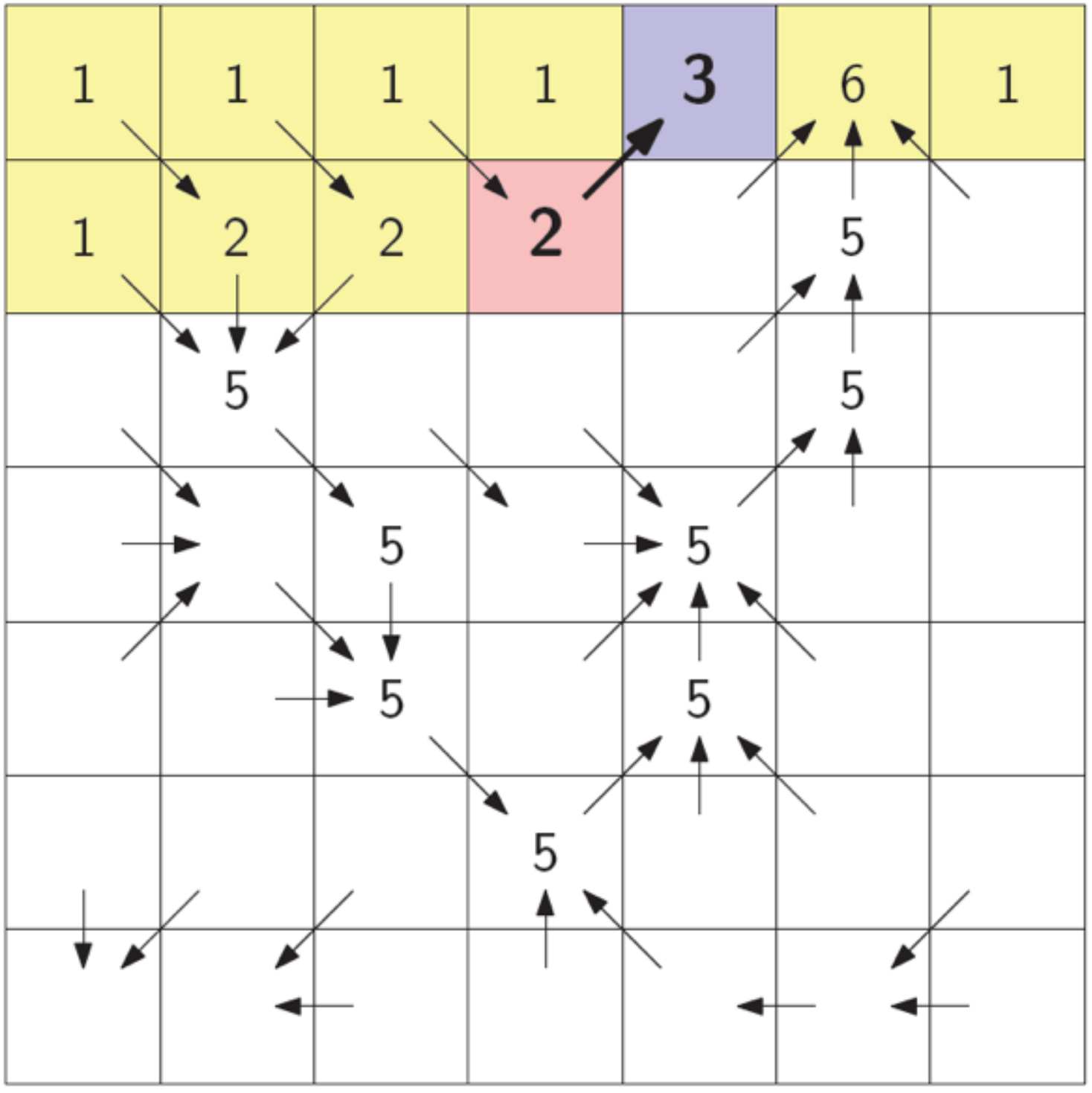
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



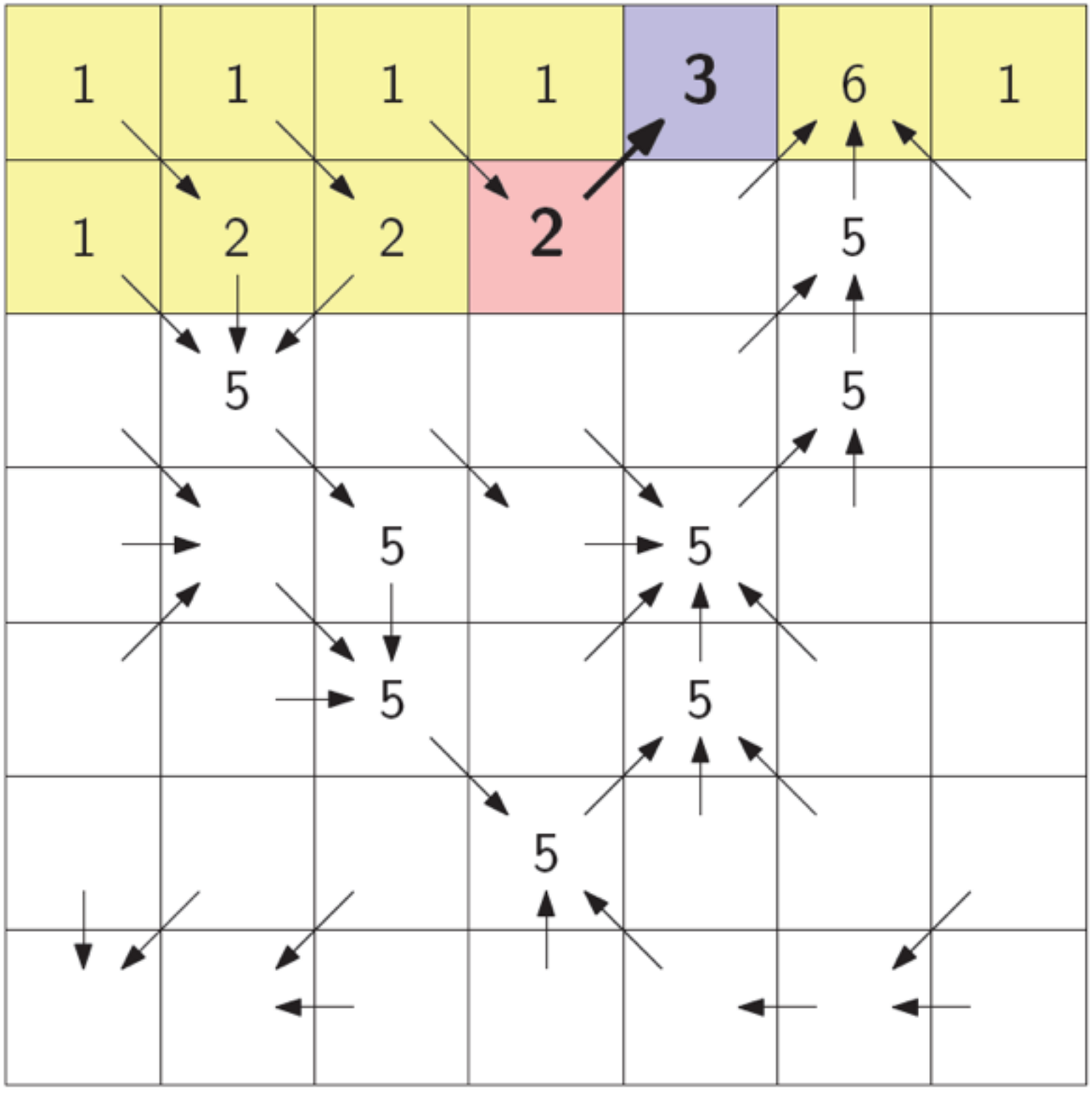
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



n = nb. of cells in the grid

worst-case running time Theta(n^2)

thanks!!! to H. Haverkort

Computing FA: naive algorithm (2)

```
for (i=0; i<nrows; i++)
   for (j=0; j<ncols; j++)
         flow[i][j] =compute_flow(i,j);
```

```
//return 1 if cell (a,b)'s FD points to cell (x,y)
// that is, if (a,b)'s FD points towards (x,y)
int flows_into(a,b,  x,y) {
        if (!inside_grid(a,b)) return 0;
        …
}
```

- Is this linear ??
- Worst case running time?
- What sort of FD graph would trigger worst-case?

```
//return the flow of cell (i,j)
int compute_flow(i,j) {
        assert(inside_grid(i,j));
        int f = 1;  //initial flow at (i,j)
        for (k=-1; k<= 1; k++) {
                for (l=-1; l<= 1; l++) {
                        if flows_into(i+k, j+l, i,j)
                                f +=  compute_flow(i+k, j+l);
                }//for k
        }//for l
        return f;
}
```

Computing FA: naive algorithm (2)

```
for (i=0; i<nrows; i++)
   for (j=0; j<ncols; j++)
         flow[i][j] =compute_flow(i,j);
```

```
//return 1 if cell (a,b)'s FD points to cell (x,y)
// that is, if (a,b)'s FD points towards (x,y)
int flows_into(a,b,  x,y) {
        if (!inside_grid(a,b)) return 0;
        …
}
```

worst-case running time Theta(n^2)

```
//return the flow of cell (i,j)
int compute_flow(i,j) {
        assert(inside_grid(i,j));
        int f = 0;  //initial flow at (i,j)
        for (k=-1; k<= 1; k++) {
                for (l=-1; l<= 1; l++) {
                        if flows_into(i+k, j+l, i,j)
                                f +=  compute_flow(i+k, j+l);
                }//for k
        }//for l
        return f;
}
```

Flow accumulation: faster?

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 1: naive algorithm + dynamic programming:
    - Use recursion, but once a value flow(i,j) is computed, store it in a table. This avoids re-computation.

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 1: naive algorithm + dynamic programming:
  - Use recursion, but once a value flow(i,j) is computed, store it in a table. This avoids re-computation.

```
for (i=0; i<nrows; i++)
   for (j=0; j<ncols; j++)
        flow[i][j] =compute_flow(i,j);
```

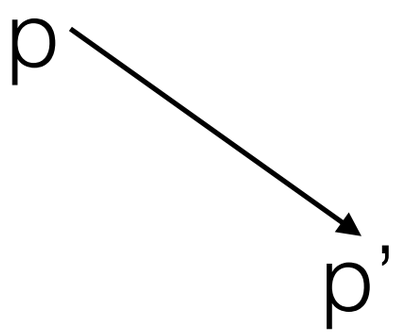
Analysis?

```
//return the flow of cell (i,j)
int compute_flow(i,j) {
        assert(inside_grid(i,j));
        if flow[i][j] != -1:  return flow[i][j]
        int f = 1;  //initial flow at (i,j)
        for (k=-1; k<= 1; k++) {
              for (l=-1; l<= 1; l++) {
                    if flows_into(i+k, j+l, i,j)
                          f +=  compute_flow(i+k, j+l);
              }//for k
        }//for l
        flow[i][j] = f
        return f;
}
```

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 2: Naive algorithm + avoid recursion

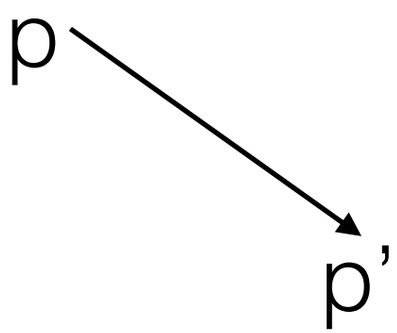


if p's FD points to p', then compute FA of **p before p'**

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 2: Naive algorithm + avoid recursion



if p’s FD points to p’, then compute FA of **p before p’**

Instead of row-major order, traverse points in **topological order**!

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 2: Naive algorithm + avoid recursion

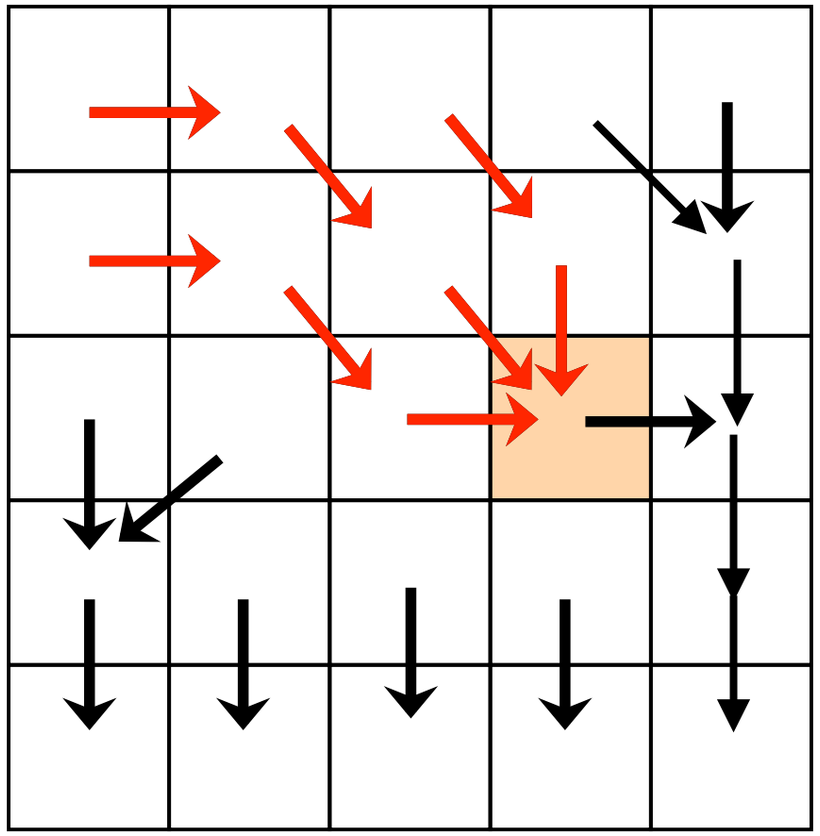
Initialize all cels to flow[i][j] = 1

For each cell (i,j) in **topological order**:

compute_flow(i,j)

```
//return the flow of cell (i,j)
void compute_flow(i,j) {
      assert(inside_grid(i,j));
      for (k=-1; k<= 1; k++) {
            for (l=-1; l<= 1; l++) {
                  if flows_into(i+k, j+l, i,j)
                        flow[i][j] +=  flow[i+k][j+l];
            }//for k
      }//for l
}
```

Analysis?



Compute a topological order, and number the cells in this order

Then traverse the cells in this order, and update their flow

n = nb. of cells in the grid

Flow accumulation: faster

- Fast algorithm 3: call copute_flow(i,j) only from points (i,j) that are outs of rivers (either flow off the grid, or are in pits and have no FD)

```
//return the flow of cell (i,j)
void compute_flow(i,j) {
        assert(inside_grid(i,j));
        for (k=-1; k<= 1; k++) {
                for (l=-1; l<= 1; l++) {
                        if flows_into(i+k, j+l, i,j)
                                flow[i][j] +=  flow[i+k][j+l];
                }//for k
        }//for l
}
```

Analysis?