Flow on terrains
(I)

Laura Toma

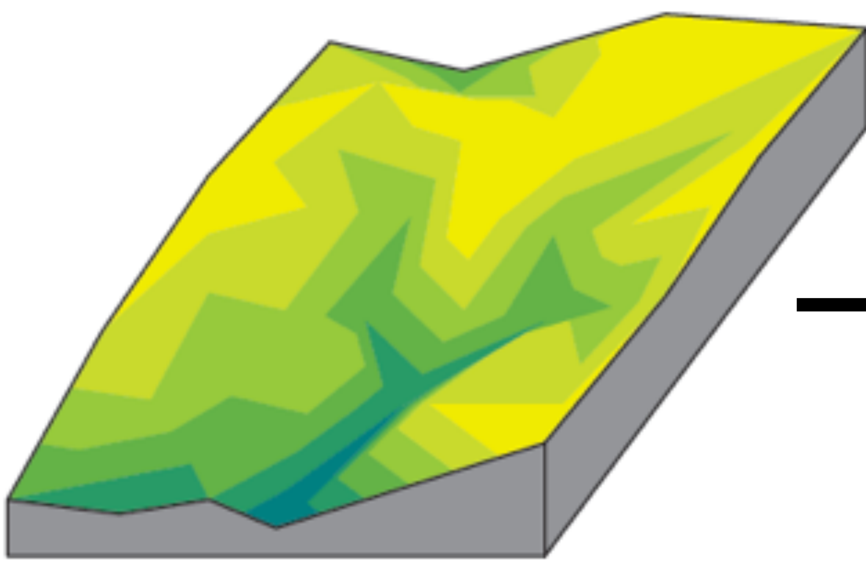
Bowdoin College

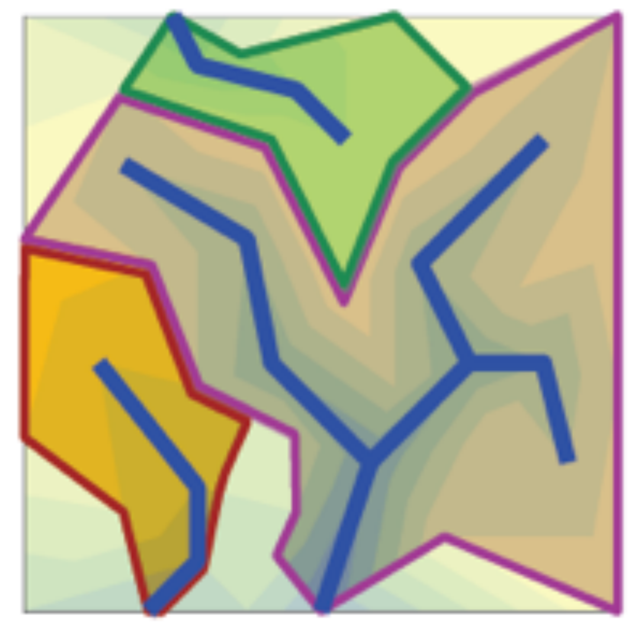
Overview

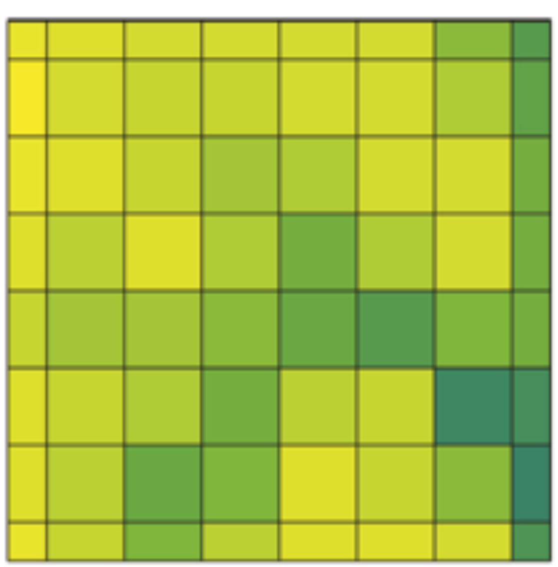
- Flow on grid terrains
    - Flow direction
    - Flow accumulation
    - Flat areas
    - Watersheds and watershed hierarchy

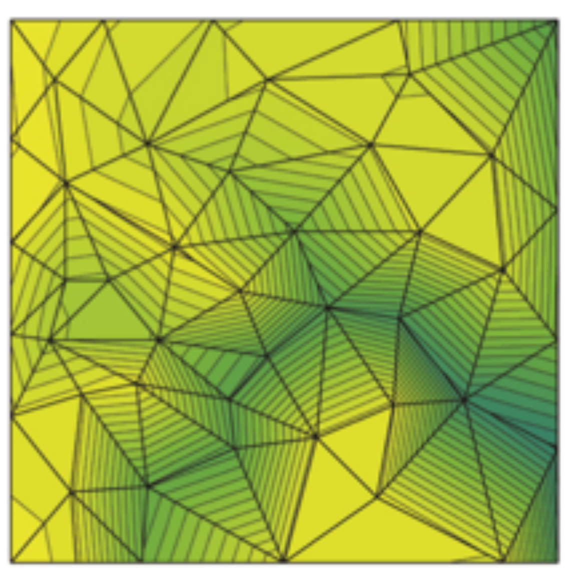
- Where does the water go when it rains?
- What will happen when it rains (a lot)?
- What are the areas susceptible to flooding?
- What areas will flood first?
- What parts of the world will go under water when sea level rises by e.g. 10 ft?
- River data is expensive to collect. Is it possible to model and automatically compute rivers on a terrain?
- What area drains to a point?
- Suppose someone spilled some pollutant)at this point on the terrain—-what area is contaminated when it rains ?
- … and many more.

Flow on digital terrain models









river network,
watersheds,
flooding,

.....

Big data

- Massive amounts of terrain data available
  - e.g. NASA SRTM, acquired 80% of Earth at 30m resolution. Total 5TB !!
  - USGS: most USA at 10m resolution
  - LIDAR data: 1m resolution

==> need efficient algorithms!!

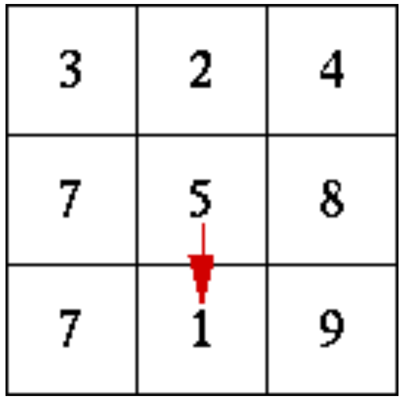
- Example:
  - Area if approx. 800 km x 800 km
  - Sampled at:
    - 100 resolution: 64 million points **(128MB)**
    - 30m resolution: 640 **(1.2GB)**
    - 10m resolution: 6400 = 6.4 billion **(12GB)**
    - 1m resolution: 600.4 billion **(1.2TB)**

Flow on grid terrains

- Modeled by two basic concepts
  - **Flow direction (FD)**
    - the direction water flows at a point
  - **Flow accumulation (FA)**
    - total amount of water flowing through a point
- Based on this can define
  - watersheds, drainage areas, river network, flooding
  - (Pfafstteter) river and watershed hierarchy

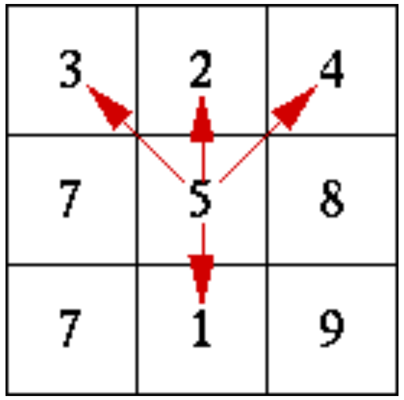
Flow direction (FD)

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



SFD: Single flow direction

(steepest downslope)

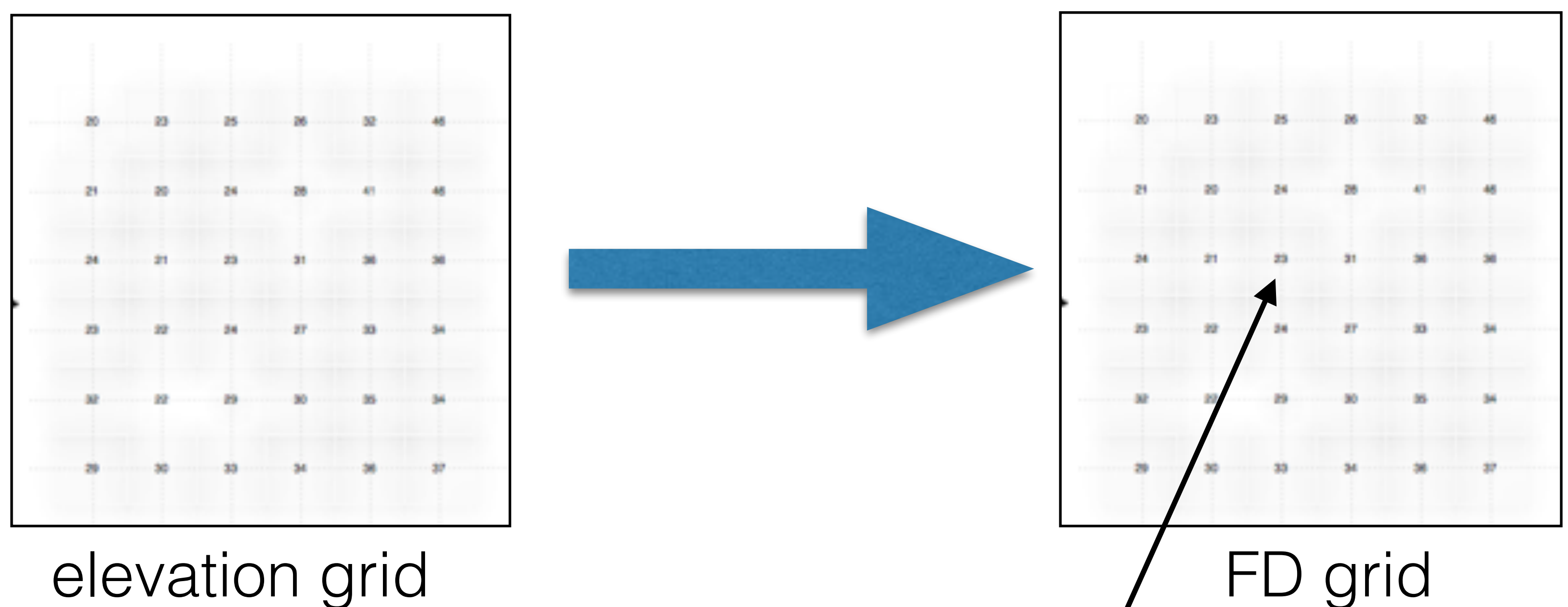


MFD: Multiple flow directions

(all downslope neighbors)

Flow direction

n = nb. of cells in the grid



point (i,j) in FD grid stores FD(i,j)
values usually coded as

32	64	128
16		1
8	4	2

- FD can be computed in O(n) time
- Issue: flat areas… later

Flow direction

n = nb. of cells in the grid

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

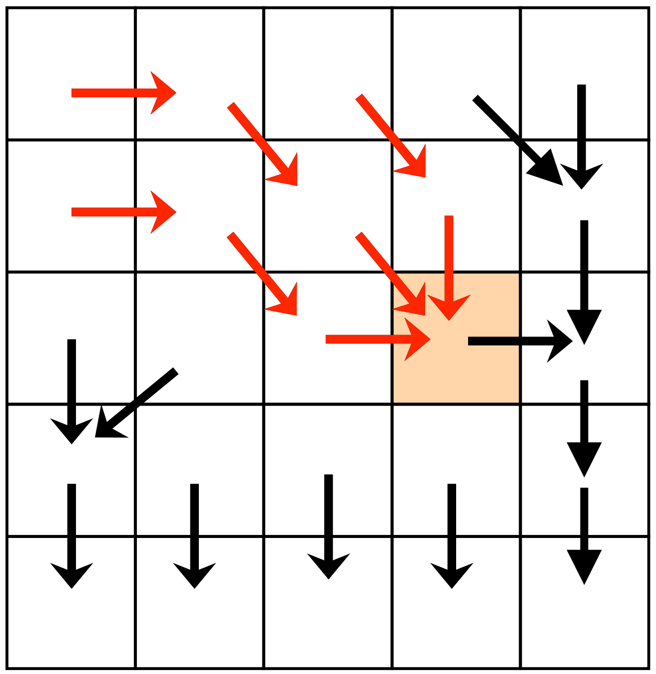
2	2	2	4	4	8
2	2	2	4	4	8
1	1	2	4	8	4
128	128	1	2	4	8
2	2	1	4	4	4
1	1	1	1	4	16

Flow direction

Direction coding

The coding of the direction of flow

Flow accumulation (FA)



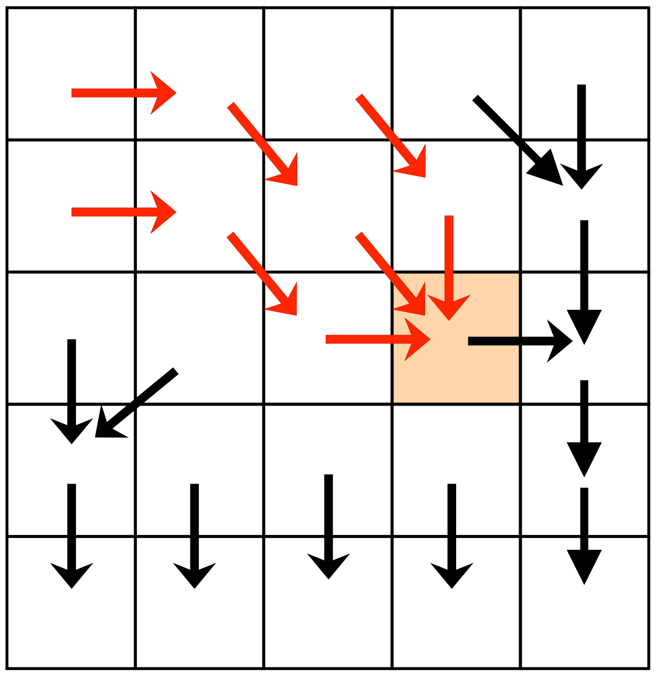
- FA(p) = how much water goes through point p

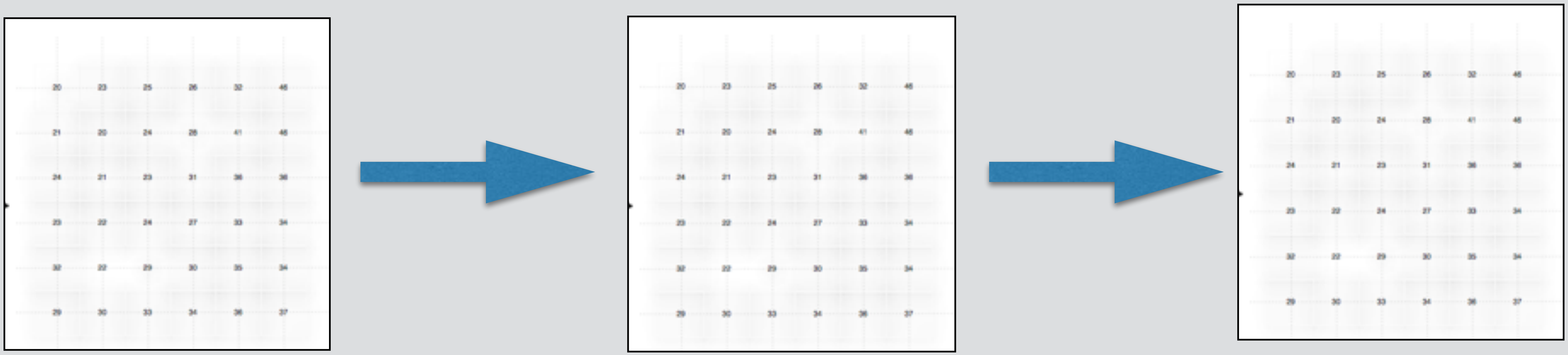
- FA grid:
  - Compute, for each cell (point) c, how much water passes through that cell.
  - Assume each cell starts with 1 unit of water
  - Assume each cell sends its initial as well as incoming water to the neighbor cell pointed to by its FD

Flow accumulation (FA)

- FA(p) = how much water goes through point p

- FA grid:
  - Compute, for each point/cell c, how much water passes through that cell.





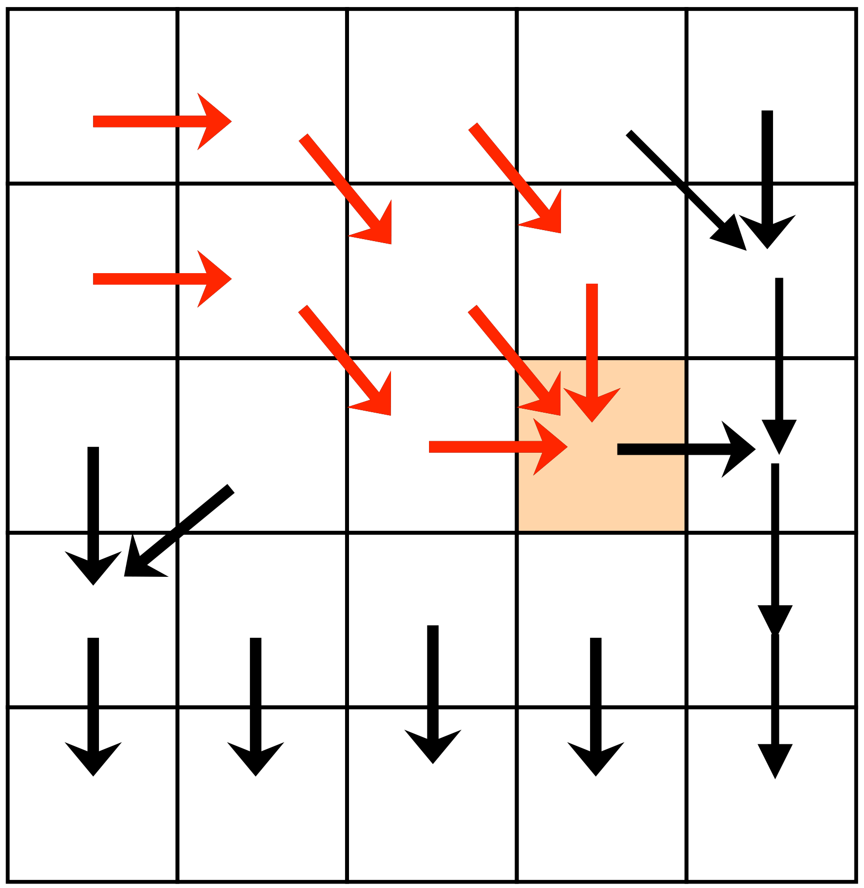
elevation grid

FD grid

FA grid

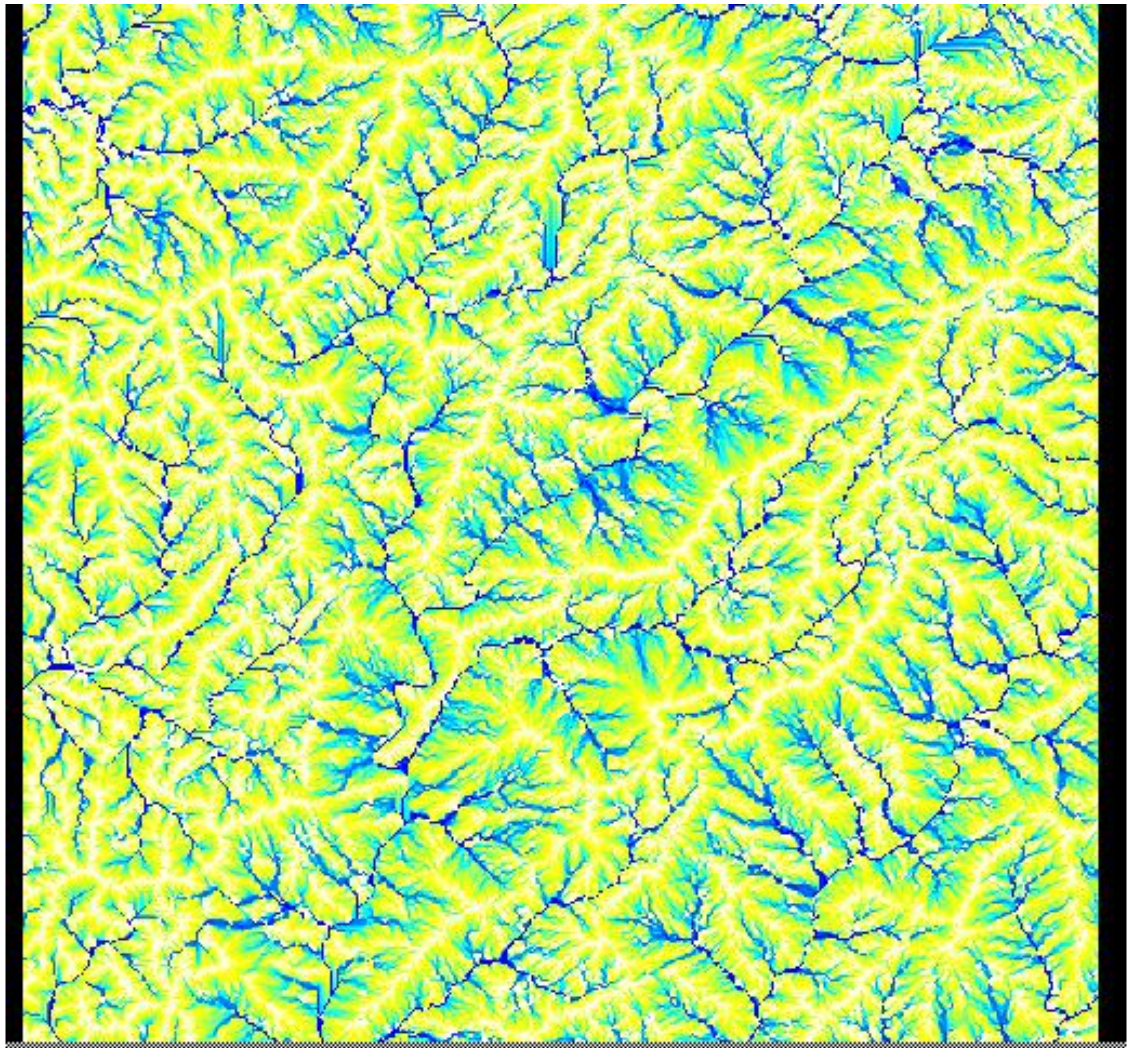
FD and FA

- Some observations
  - FD graph: forest of trees
  - each tree represents a separate “river tree”
  - points with small FA= ridges
  - points with high FA = channels (rivers)
  - FA: how many cells are upstream, or size of subtree of that cell, if viewing the tree upside down



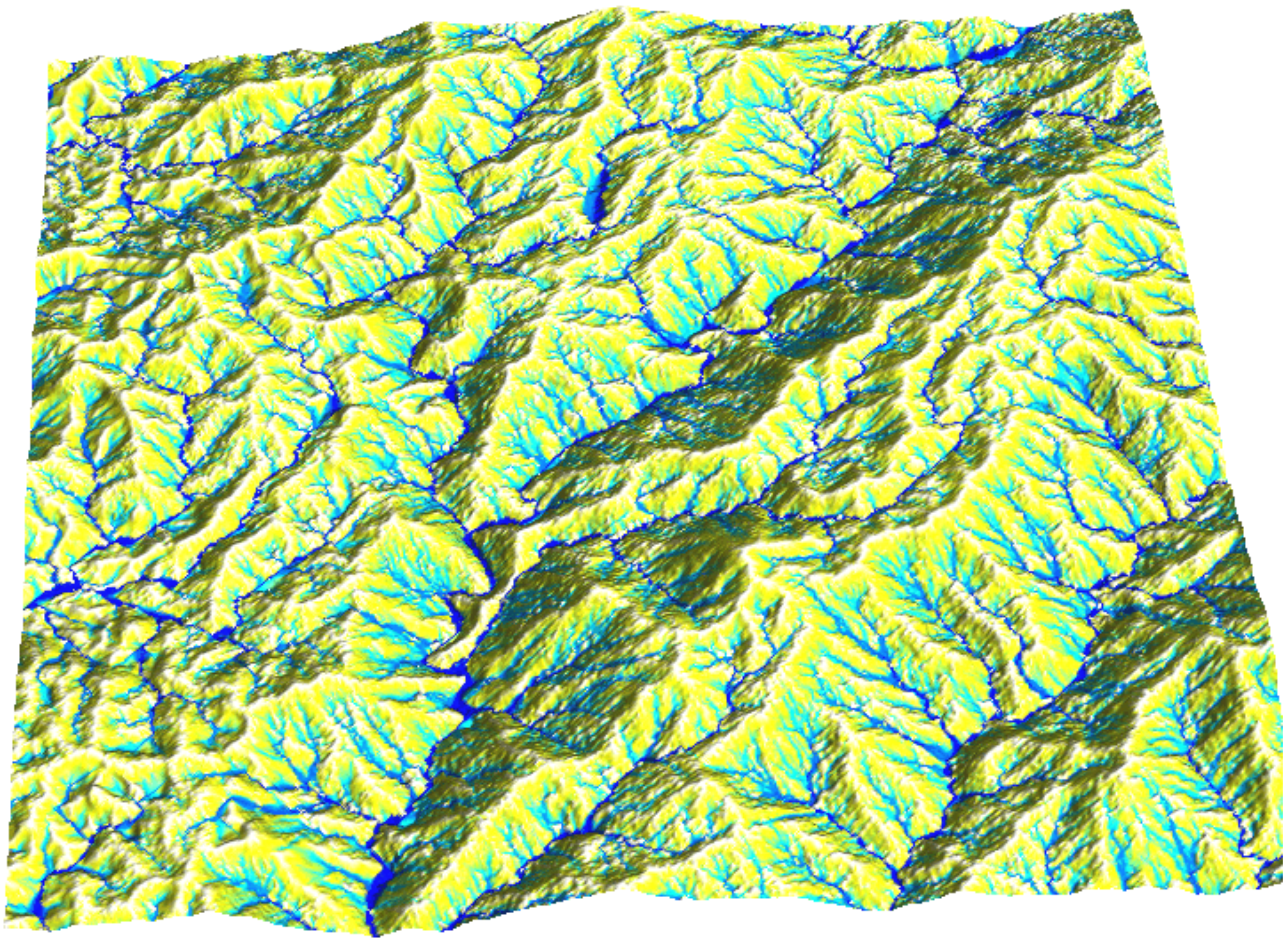
- FA models rivers!
  - set an arbitrary threshold t
  - cell c is on a river if FA(c) >= t

Flow accumulation



FA 2D view

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



FA grid draped over elevation grid

Computing FA: naive algorithms

- Idea 1:
    - Scan row-by-row: for each cell add +1 to flow of all cells along 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
- Other ideas?

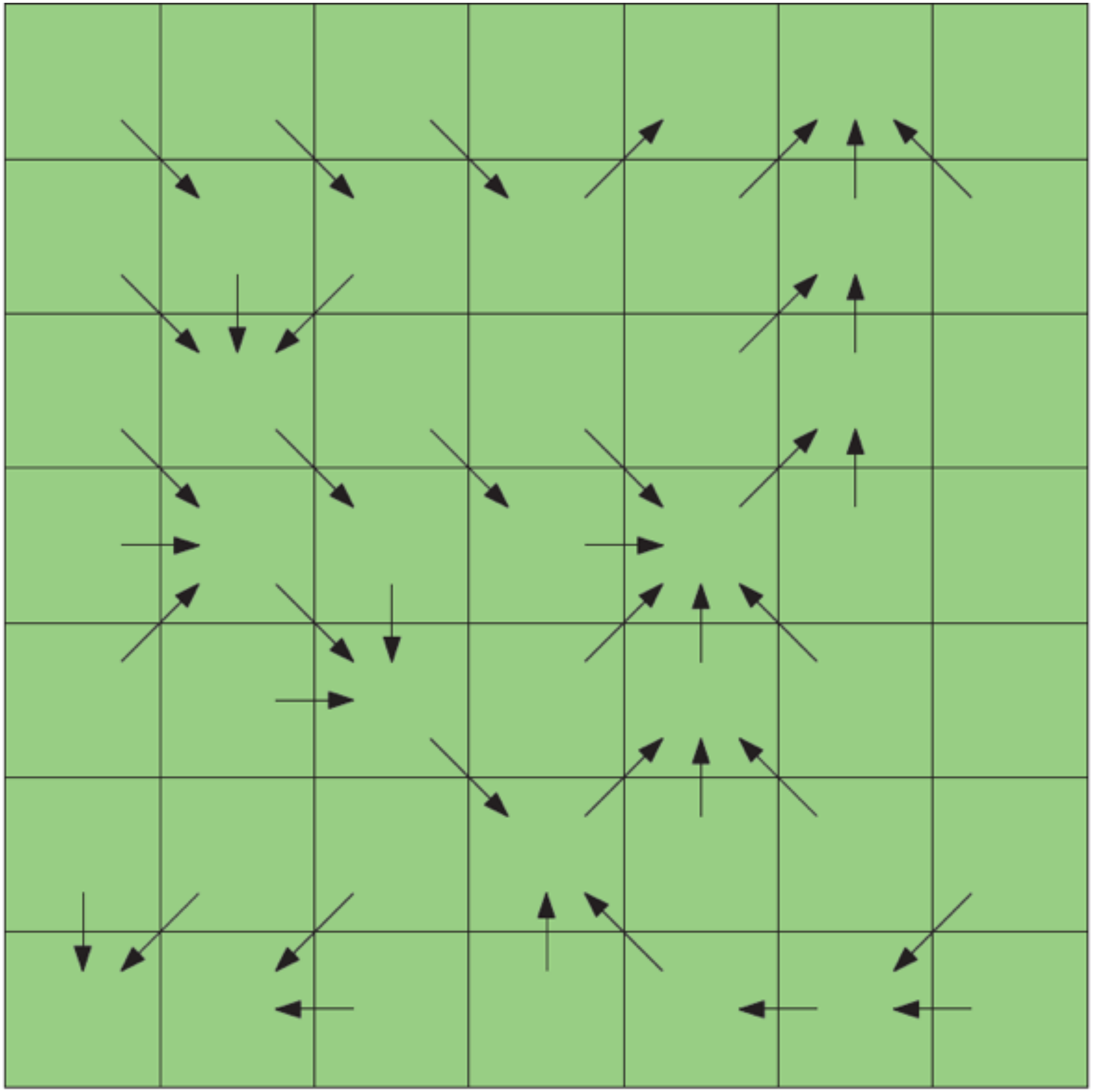
Computing FA: naive algorithms

- Idea 1:
    - Scan row-by-row: for each cell add +1 to flow of all cells along its downstream path
    - Analysis??

- 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
    - Analysis??

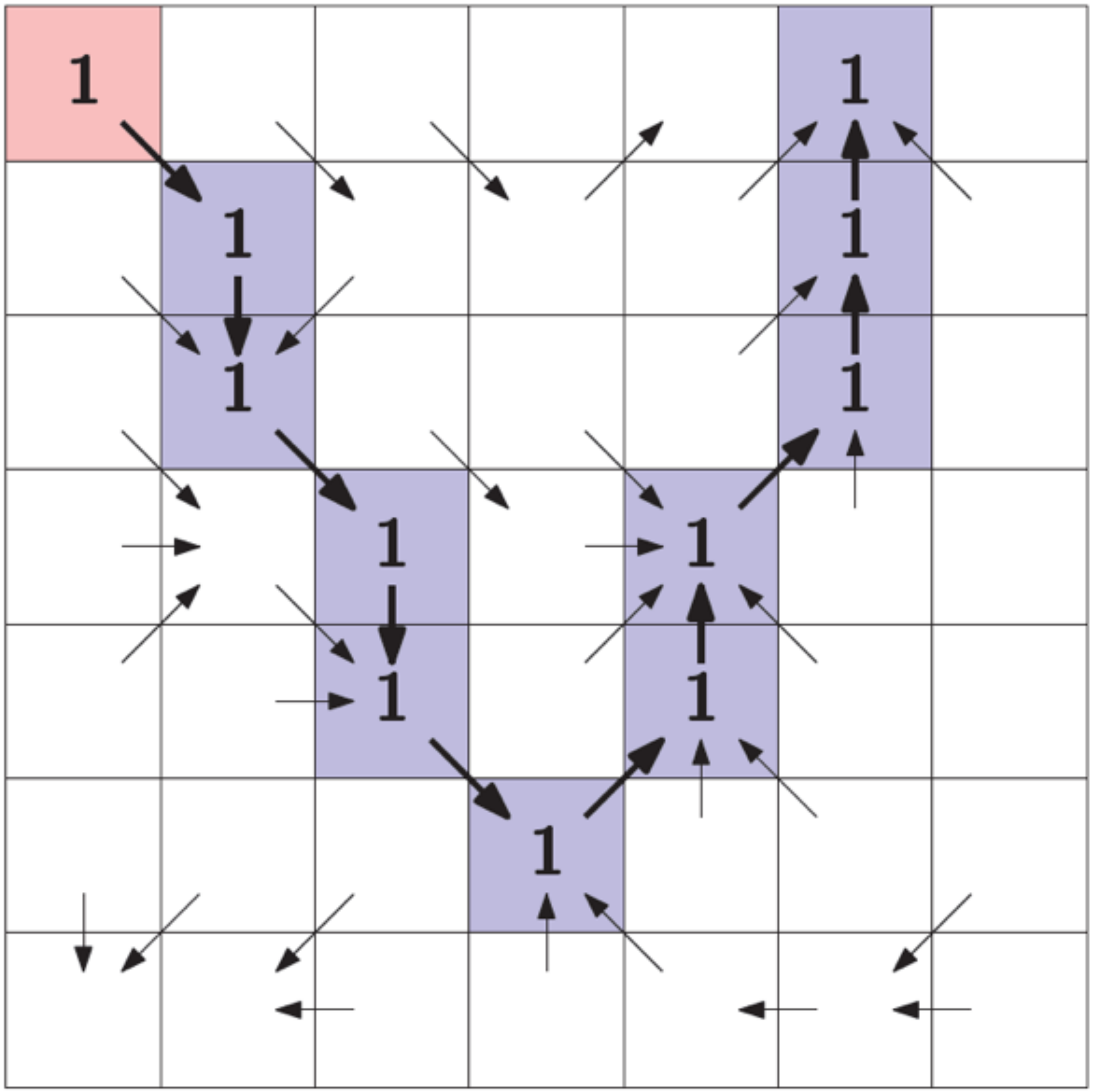
- Other ideas?
    - Analysis??

Computing FA: naive algorithm (1)



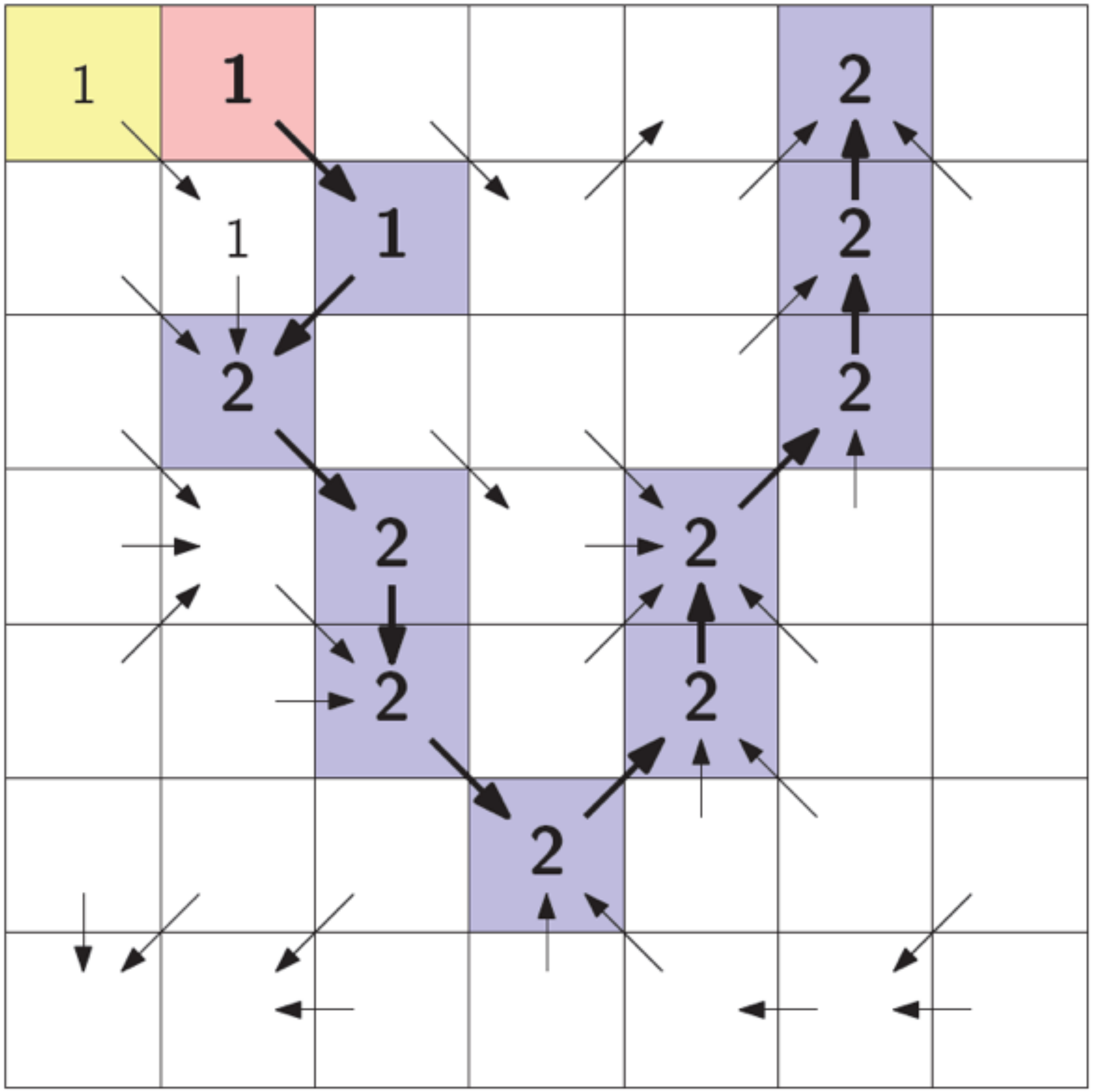
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



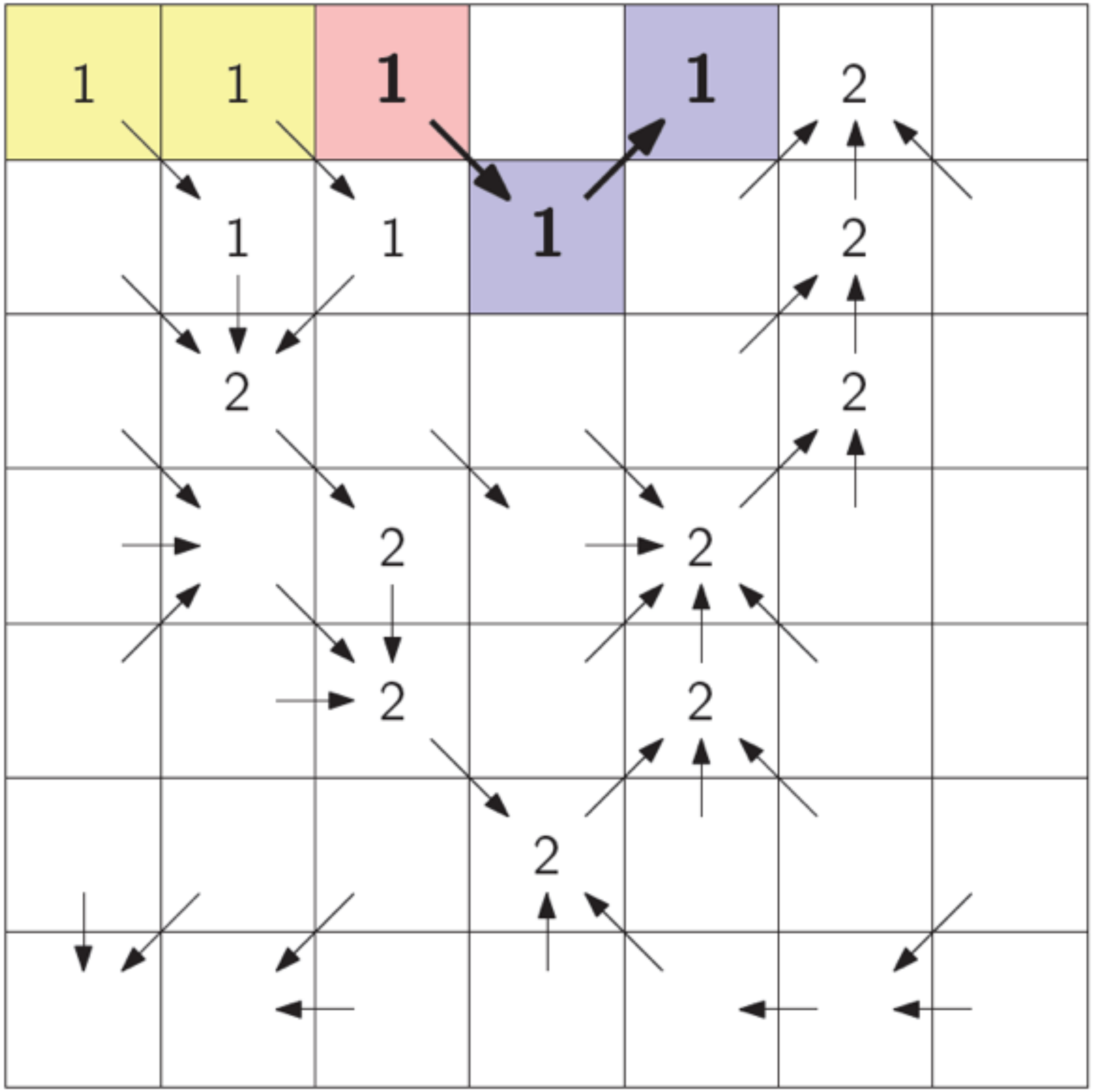
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



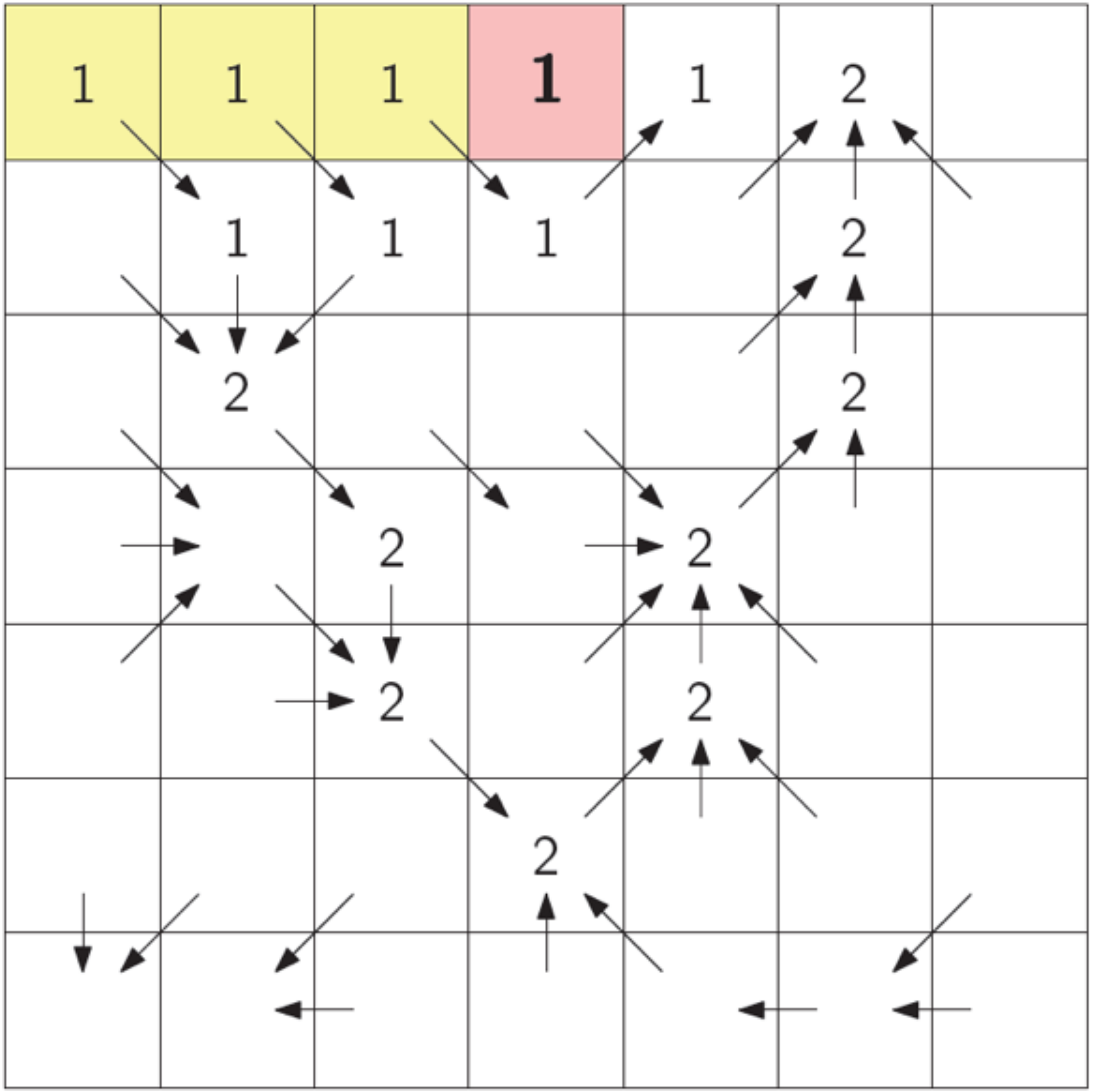
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



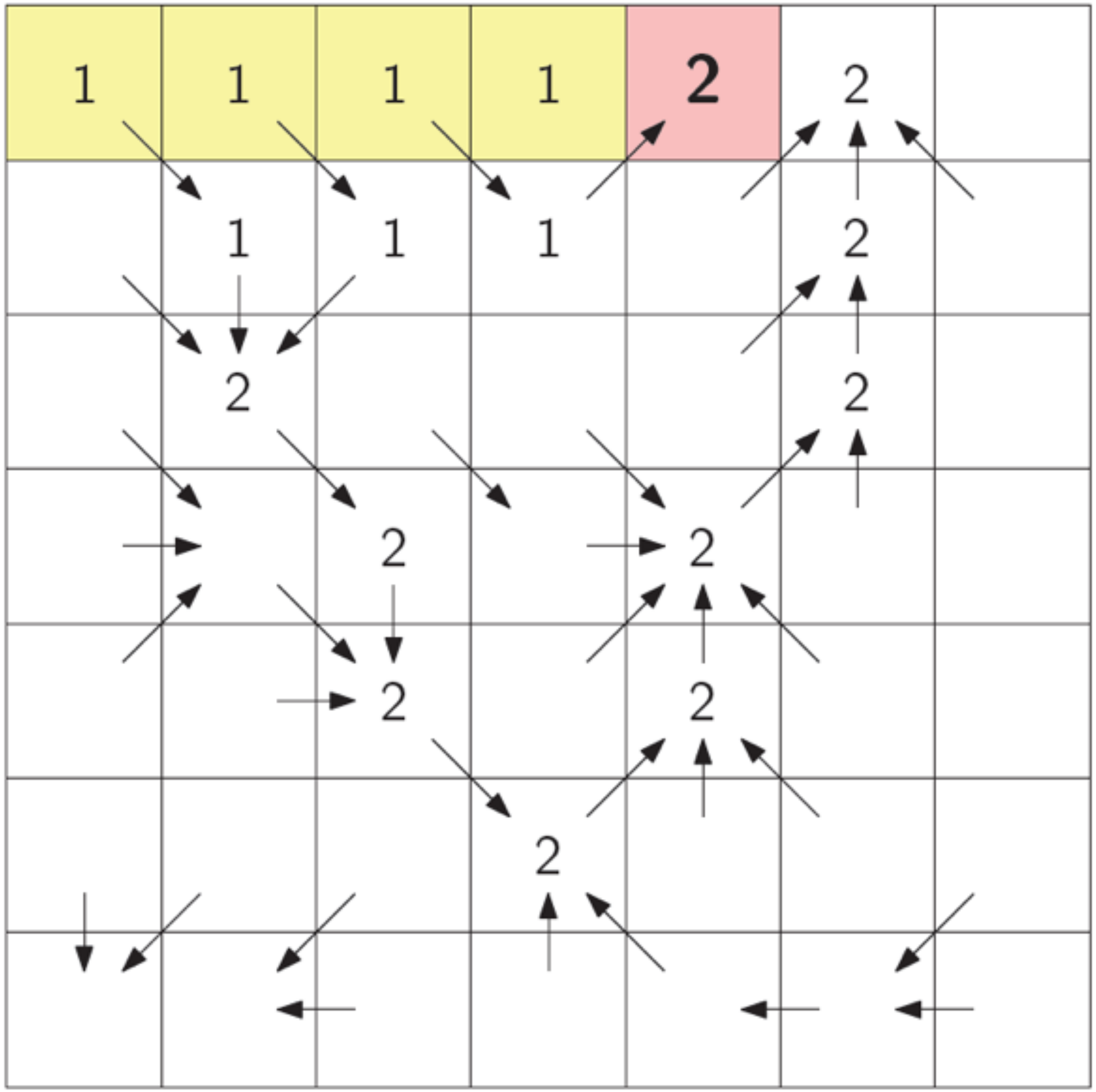
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



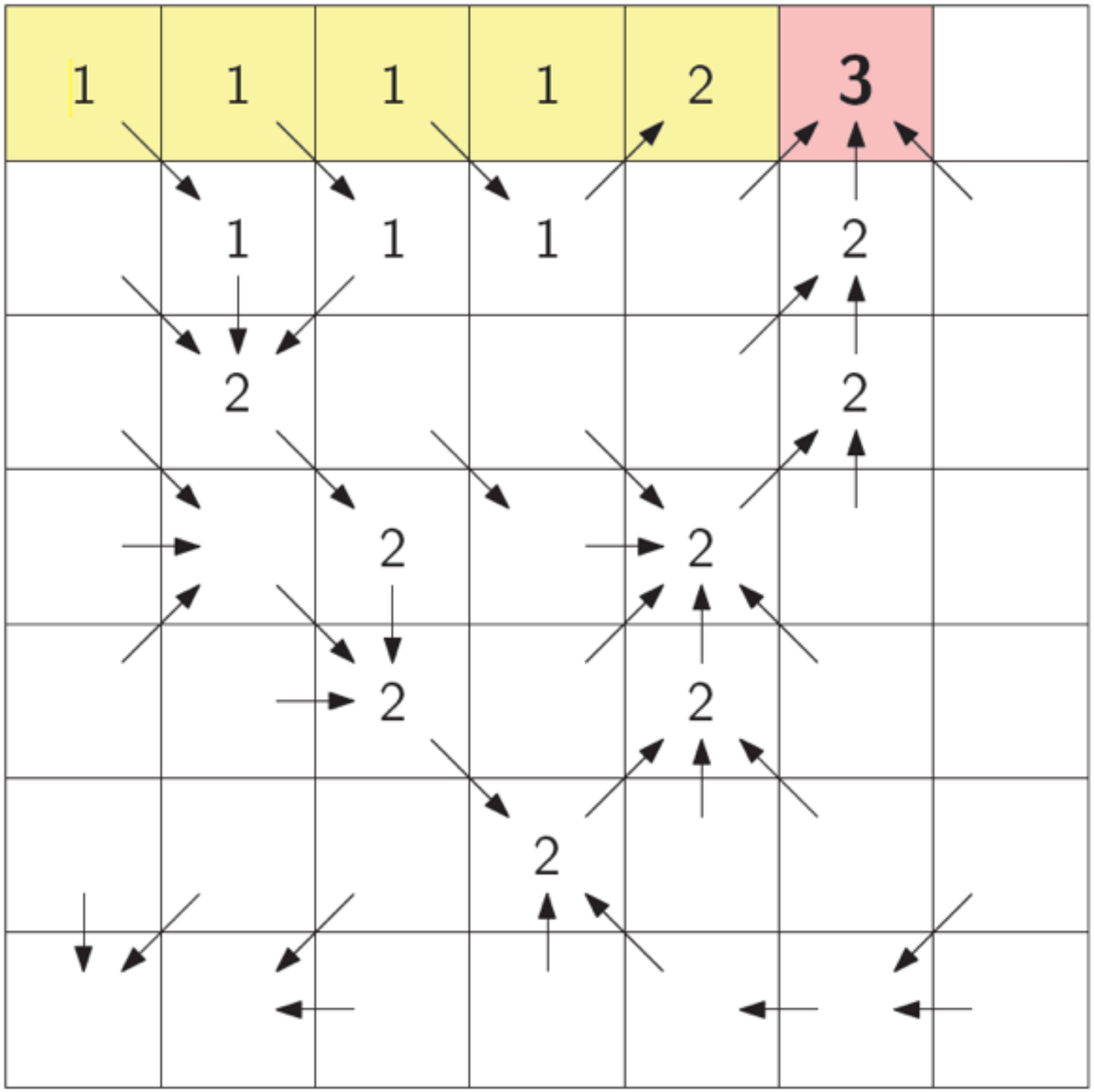
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



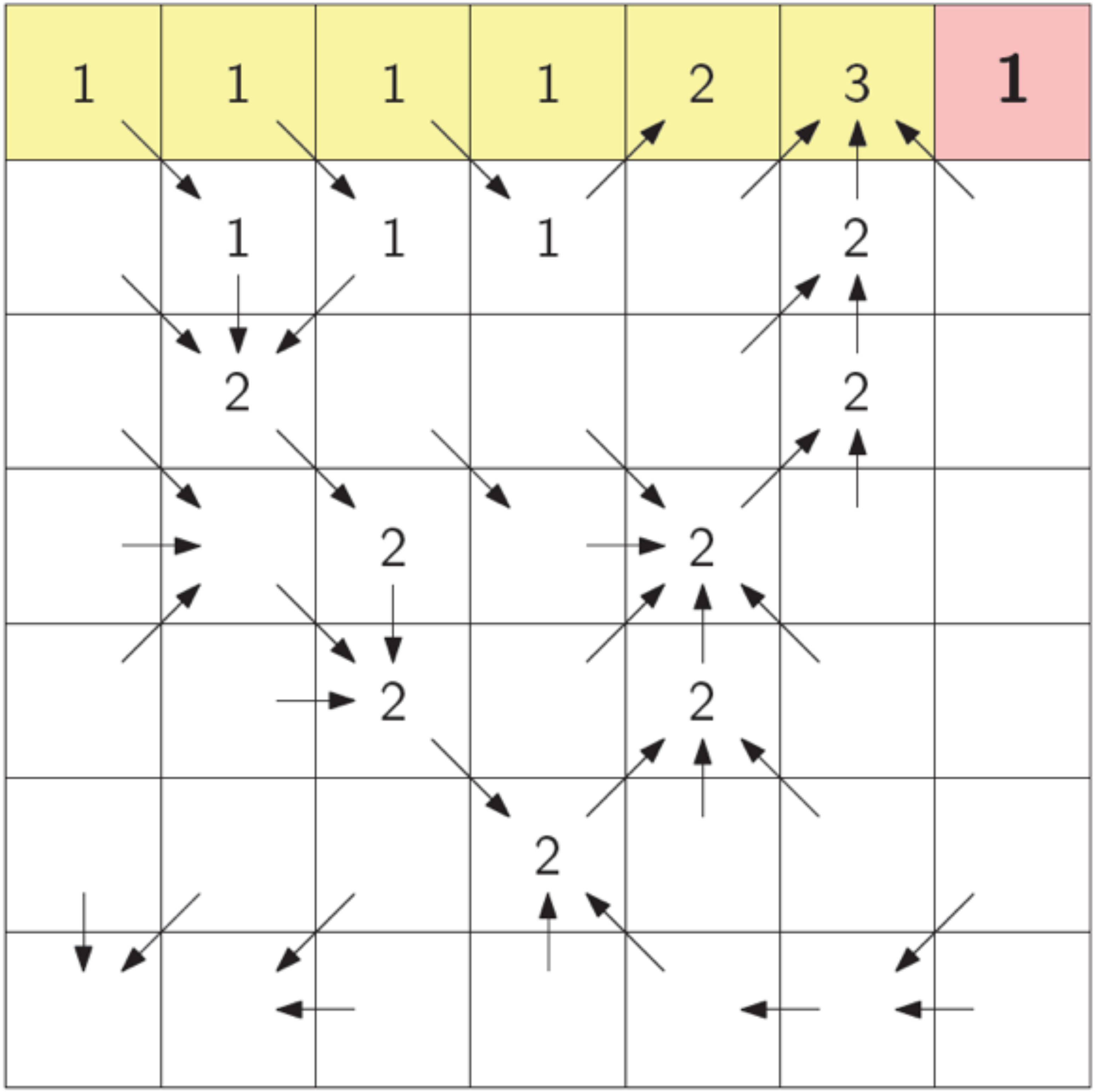
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



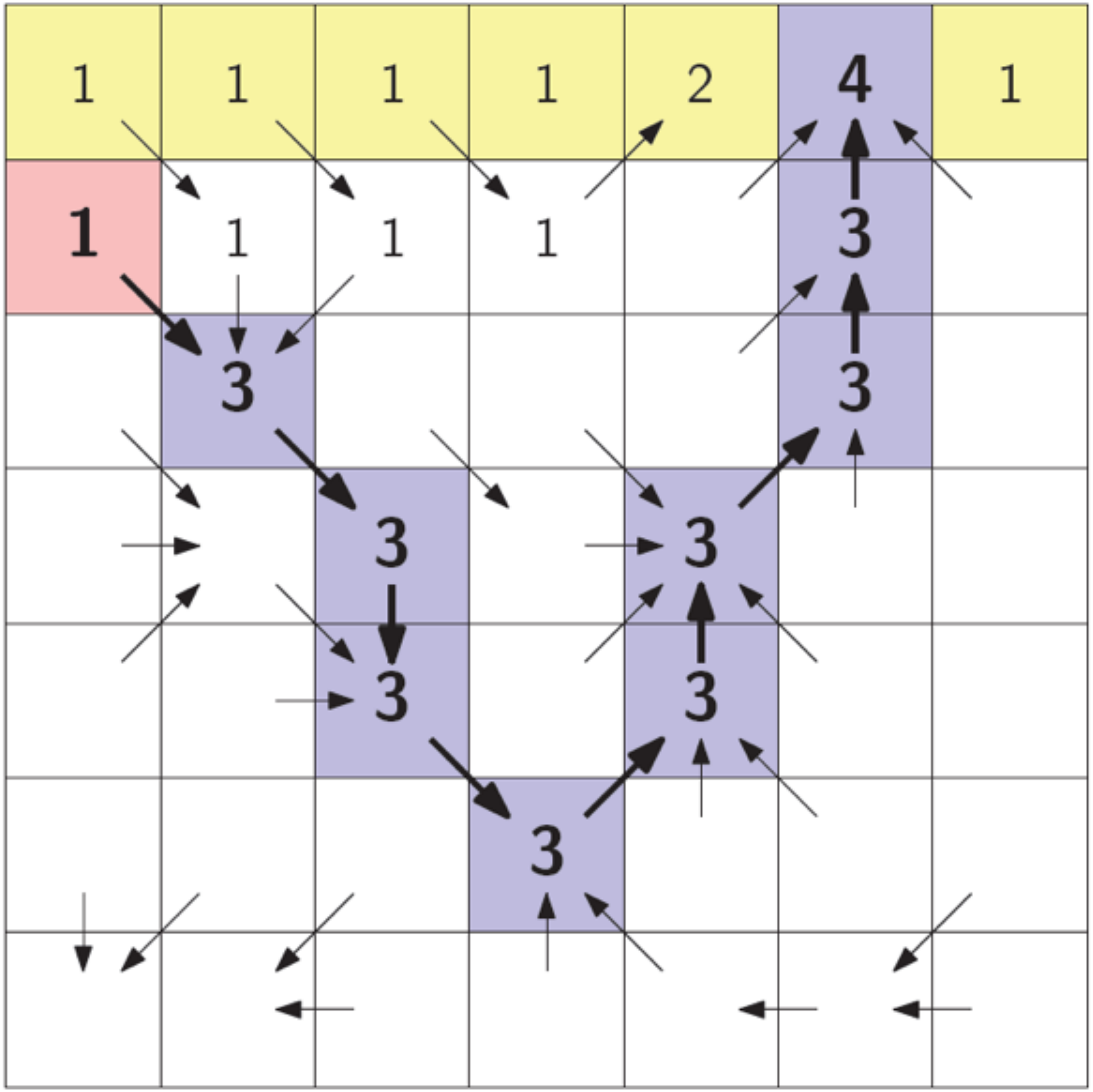
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



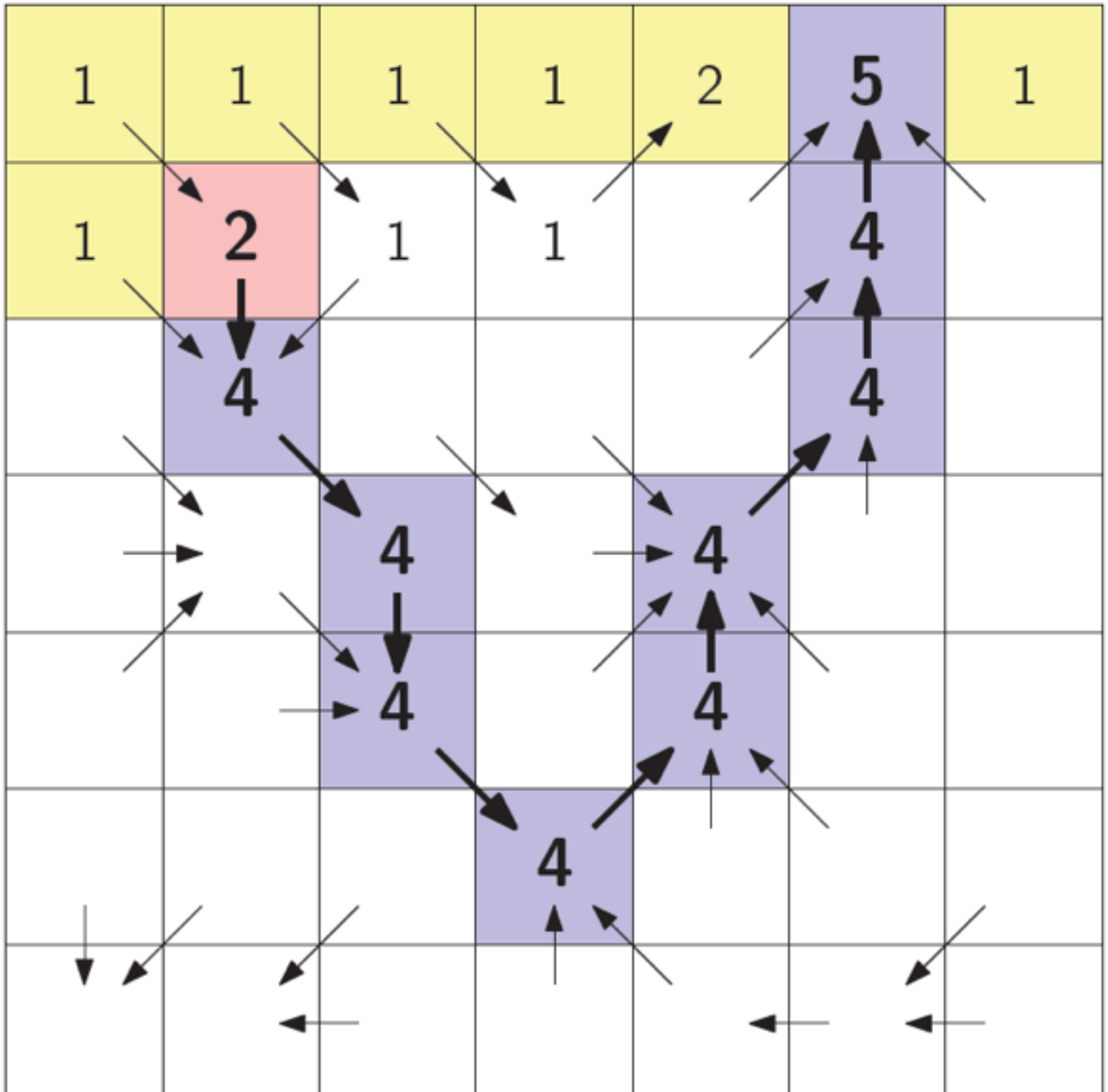
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



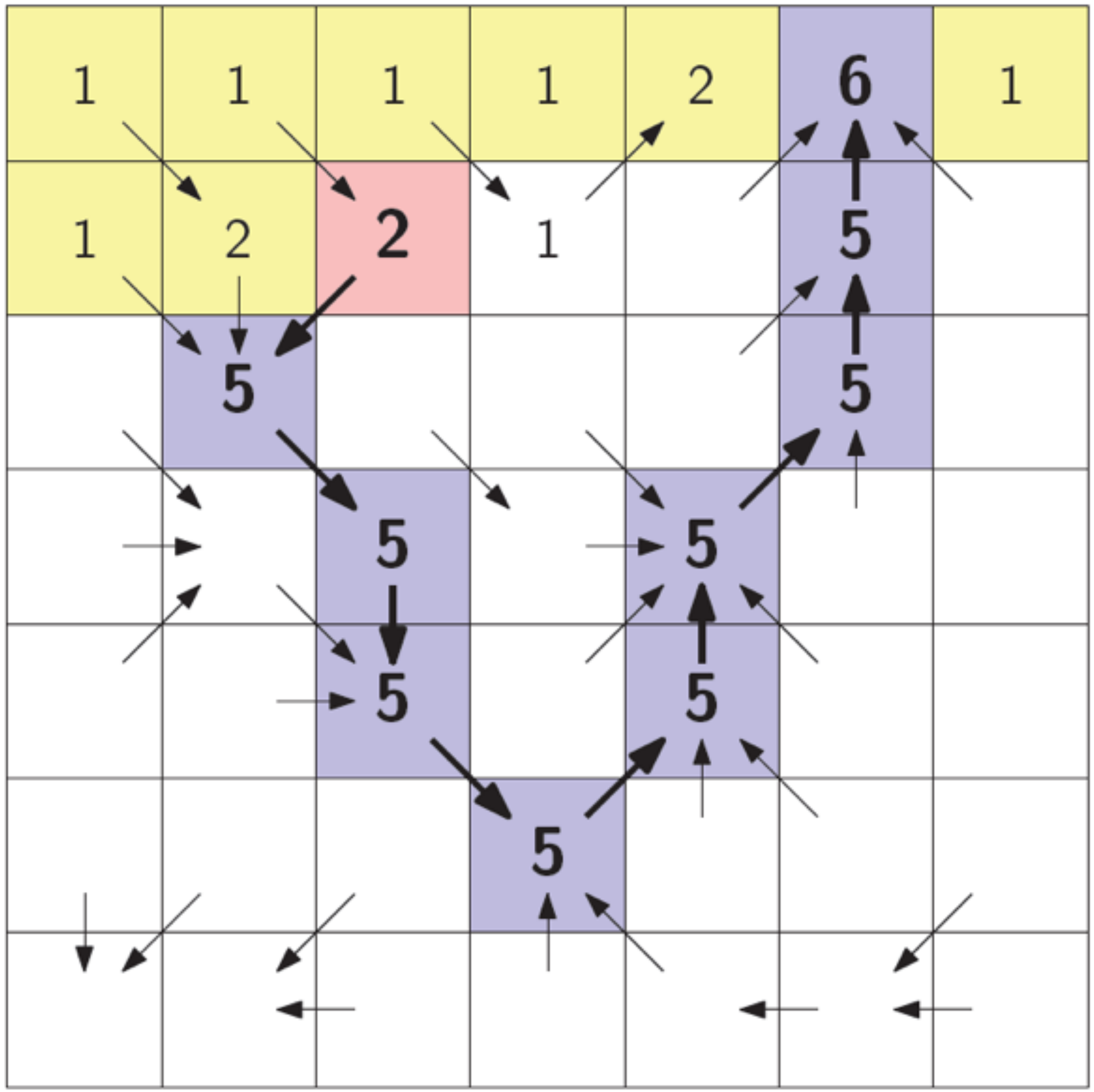
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



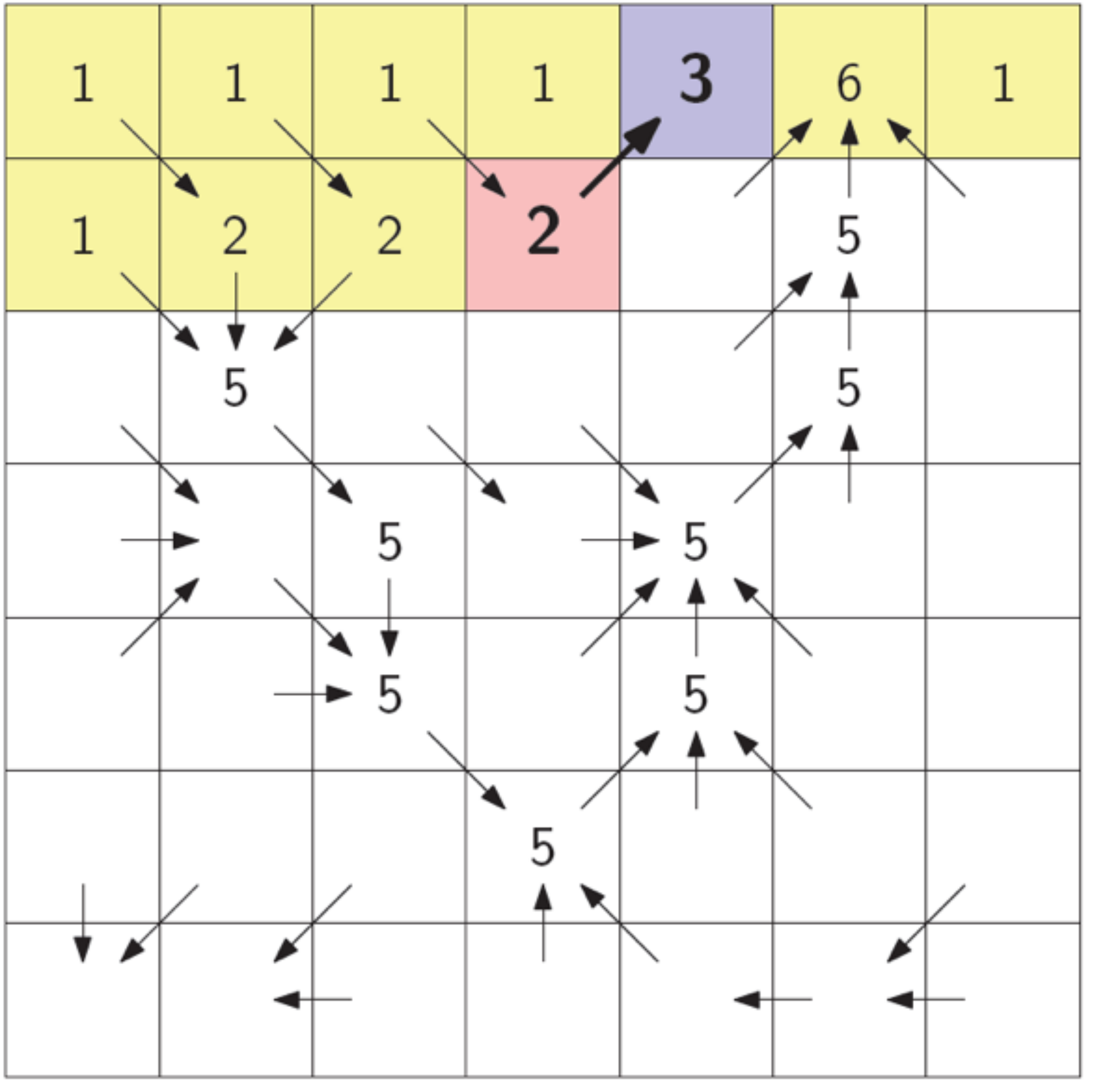
thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)



thanks!!! to H. Haverkort

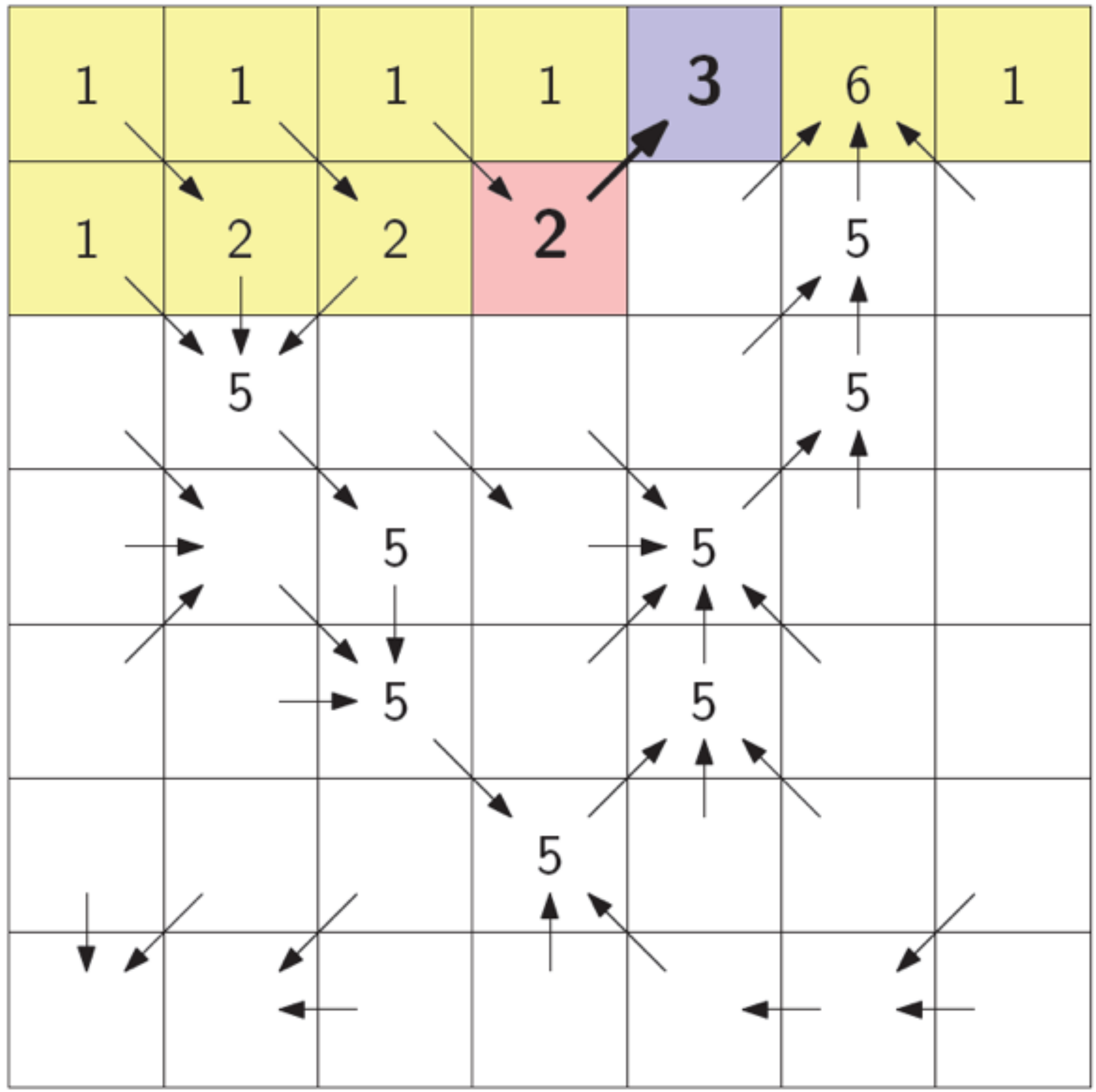
Computing FA: naive algorithm (1)



thanks!!! to H. Haverkort

Computing FA: naive algorithm (1)

n = nb. of cells in the grid



thanks!!! to H. Haverkort

worst-case running time Theta(n^2)

Computing FA: naive algorithm (2)

```
//return 1 if cell (a,b) flows into 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;
        ...
}
```

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

```
//return the flow of cell (i,j)
void 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;
}
```

Computing FA: naive algorithm (2)

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

- Questions:
  - What is the worst case running time?
    - Is it linear?
  - What sort of FD graph would trigger worst-case?

```
//return 1 if cell (a,b) flows into 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;
        …
}
```

```
//return the flow of cell (i,j)
void 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;
}
```

Computing FA: naive algorithm (2)

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

worst-case running time $\Theta(n^2)$

```
//return 1 if cell (a,b) flows into 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;
        …
}
```

```
//return the flow of cell (i,j)
void 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: smarter algorithms?

- Ideas?

Flow accumulation: smarter algorithms?

n = nb. of cells in the grid

- Use recursion, but once a value flow(i,j) is computed, store it in a table. This avoids re-computation.
    - dynamic programming!

- To completely avoid recursion, compute flow(i,j) in topological order of FD graph
    - topological order can be computed in linear time
    - or: sort by height, but that's O(n lg n)

- Analysis?
- Which one would you chose in practice?