6.006 Recitation

Build 2008.23

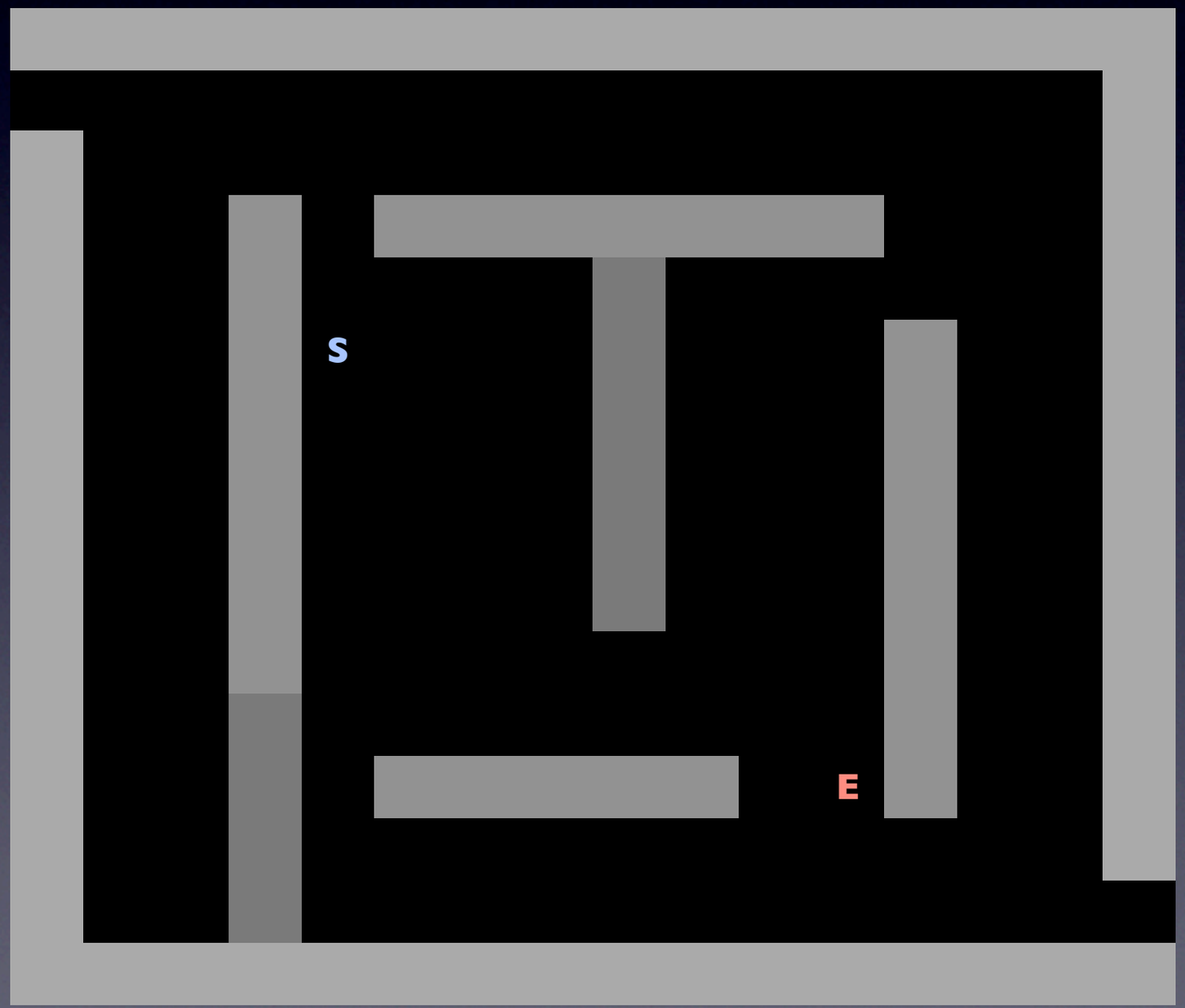
6.006 Proudly Presents

- Two-Way BFS
- Stable Sorting
- DRY

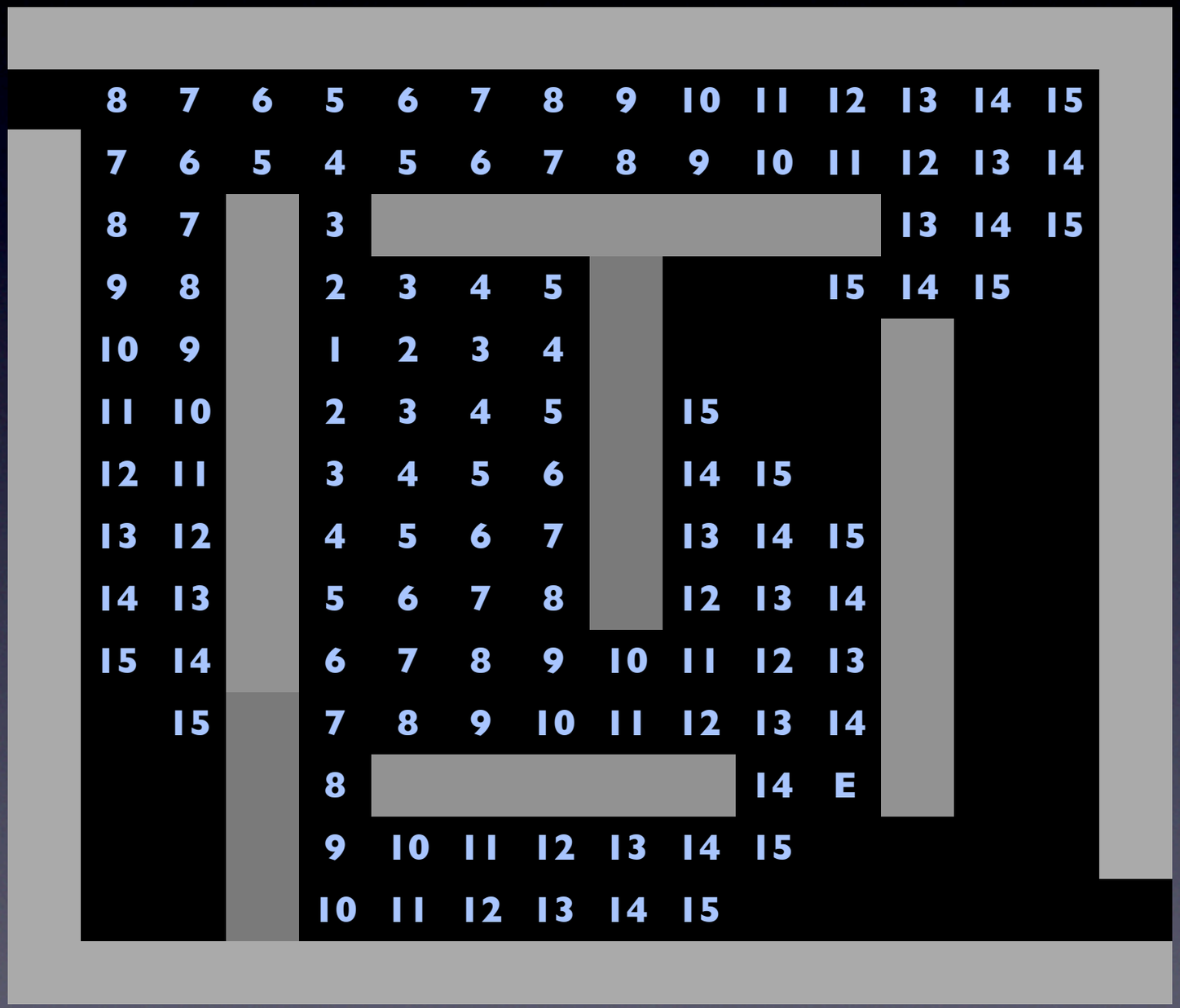
Two-Way BFS

Regular BFS	Two-Way BFS
Start: from source	Start: from both the source and the goal
End: reached a goal	End: a node is reached both from source and from goal
Works with multiple goals	Requires single goal

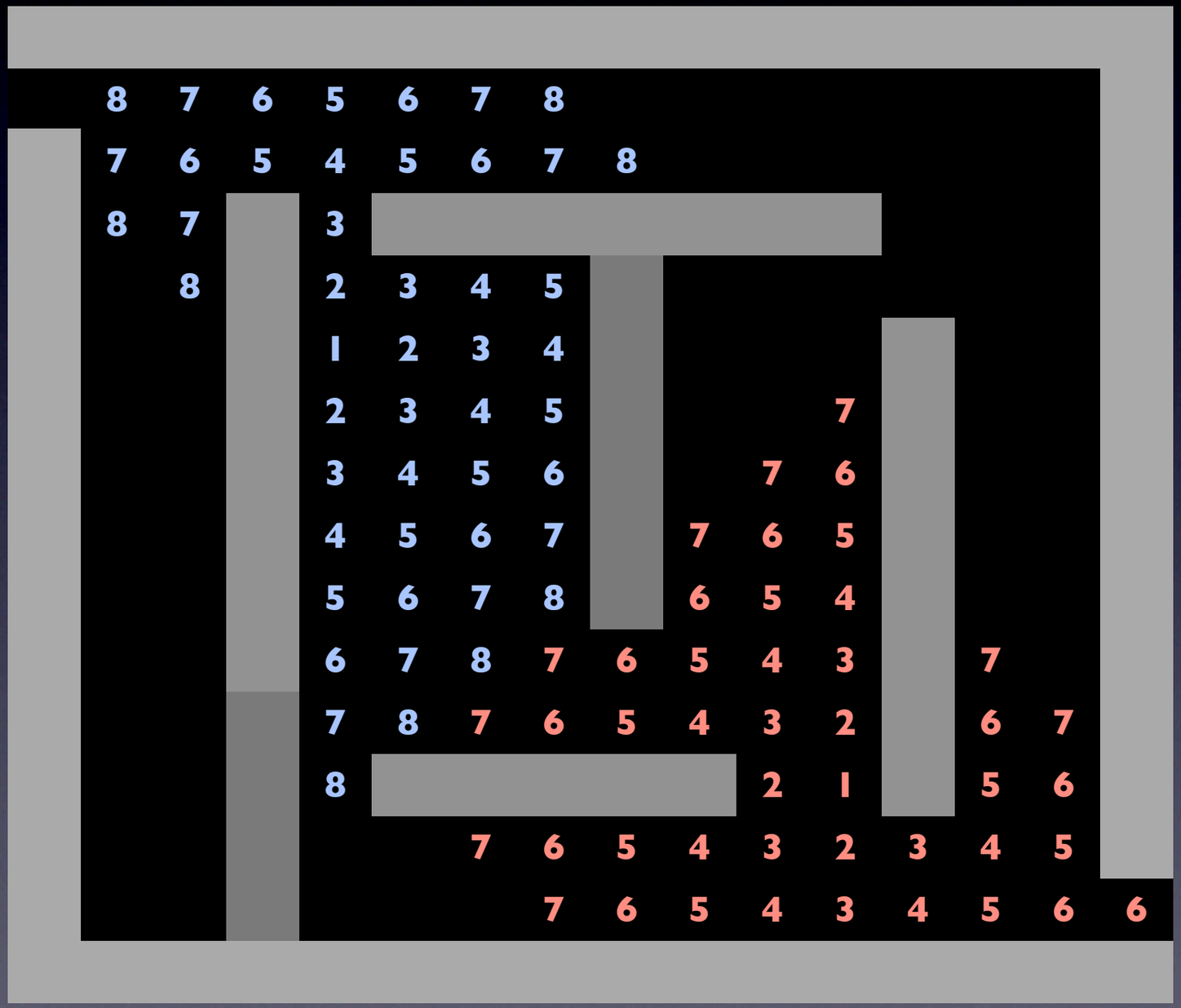
Poking Aftermath: (gasp) Meeting Her



BFS


8 7 6 5 6 7 8 9 10 11 12 13 14 15
7 6 5 4 5 6 7 8 9 10 11 12 13 14
8 7 3 13 14 15
9 8 2 3 4 5 15 14 15
10 9 1 2 3 4
11 10 2 3 4 5 15
12 11 3 4 5 6 14 15
13 12 4 5 6 7 13 14 15
14 13 5 6 7 8 12 13 14
15 14 6 7 8 9 10 11 12 13
15 7 8 9 10 11 12 13 14
8 14 E
9 10 11 12 13 14 15
10 11 12 13 14 15

Two-Way BFS


8 7 6 5 6 7 8
7 6 5 4 5 6 7 8
8 7 3
8 2 3 4 5
1 2 3 4
2 3 4 5 7
3 4 5 6 7 6
4 5 6 7 7 6 5
5 6 7 8 6 5 4
6 7 8 7 6 5 4 3 7
7 8 7 6 5 4 3 2 6 7
8 2 1 5 6
7 6 5 4 3 2 3 4 5
7 6 5 4 3 4 5 6 6

Two-Way BFS Implementation Talk

```
 1 def bfs(g, s):
 2     r = BFSResults()
 3     actives = deque()
 4     actives.append(s)
 5     r.parent[s] = None
 6     r.level[s] = 0
 7
 8     while len(actives):
 9         v = actives.popleft()
10         for n in g.neighbors(v):
11             if n not in r.parent:
12                 r.parent[n] = v
13                 r.level[n] = r.level[v] + 1
14                 actives.append(n)
15     return r
```

Stable Sorting

- Property of sorting algorithms
  - It's not Yet Another Sorting Algorithm
- Maintains the relative order of equal keys
- Desirable in some grand scheme of things (like Radix Sort)

Stable Sorting: Example

3	1	4’	1’	5	2	7	4	6	4”
1	1’	2	3	4’	4	4”	5	6	7
1’	1	2	3	4	4’	4”	5	6	7

Don’t Repeat Yourself (DRY)

- Code one decision in one place
  - No magic constants all over the code
  - Easy to change your mind (once you found the code for that decision, you don’t have to dig deeper)
- Useful every day, priceless in large systems
- Do use: functions, constants, local variables

```
 1 def detect_collisions(balls):
 2     set_of_collisions = set()
 3     x_cells = int((gas.world_max_x - gas.world_min_x) / 256) + 1
 4     y_cells = int((gas.world_max_y - gas.world_min_y) / 256) + 1
 5     grid = [[[] for i in range(x_cells)] for i in range(y_cells)]
 6     for b in balls:
 7         grid[int((b.x - gas.world_min_x) / 256)][int((b.y - gas.world_min_y) /
256)].append(b)
 8
 9     for xc in range(x_cells):
10         for yc in range(y_cells):
11             for xp in [-1, 0, 1]:
12                 for yp in [-1, 0, 1]:
13                     if xc + xp < 0 or xc + xp >= x_cells:
14                         continue
15                     if yc + yp < 0 or yc + yp >= y_cells:
16                         continue
17                     for b1 in grid[xc][yc]:
18                         for b2 in grid[xc + xp][yc + yp]:
19                             if b1.id < b2.id and gas.colliding(b1, b2):
20                                 set_of_collisions.add(gas.ball_pair(b1, b2))
21     return set_of_collisions
22
23 import gas
24 gas.detect_collisions = detect_collisions
25 if __name__ == "__main__":
26     gas.main()
```

Questions

Better have some!

v. Next

- Animated 2-way BFS would be nice
- 2-way BFS: consider building complete pseudocode collaboratively
- DRY: more examples
- This takes 35-40 minutes, not 20