6.006 Proudly Presents

- Two-Way BFS
- Stable Sorting
- DRY
## Two-Way BFS

<table>
<thead>
<tr>
<th></th>
<th>Regular BFS</th>
<th>Two-Way BFS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start:</strong></td>
<td>from source</td>
<td>from both the source and the goal</td>
</tr>
<tr>
<td><strong>End:</strong></td>
<td>reached a goal</td>
<td>a node is reached both from source and from goal</td>
</tr>
<tr>
<td><strong>Works with:</strong></td>
<td>multiple goals</td>
<td>single goal</td>
</tr>
</tbody>
</table>

Two-Way BFS works by starting the search from both the source and the goal, and it ends when a node is reached from both the source and the goal. It is useful when there are multiple goals, but requires a single goal for efficient search.
Poking Aftermath: (gasp) Meeting Her
Two-Way BFS
Two-Way BFS
Implementation Talk

def bfs(g, s):
  r = BFSResults()
  actives = deque()
  actives.append(s)
  r.parent[s] = None
  r.level[s] = 0

  while len(actives):
    v = actives.popleft()
    for n in g.neighbors(v):
      if n not in r.parent:
        r.parent[n] = v
        r.level[n] = r.level[v] + 1
        actives.append(n)

  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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>1’</th>
<th>2</th>
<th>3</th>
<th>4’</th>
<th>4</th>
<th>4’’</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>4’</td>
<td>1’</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>4’’</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1’</td>
<td>2</td>
<td>3</td>
<td>4’</td>
<td>4</td>
<td>4’’</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1’</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4’</td>
<td>4’’</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
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
```python
def detect_collisions(balls):
    set_of_collisions = set()
    x_cells = int((gas.world_max_x - gas.world_min_x) / 256) + 1
    y_cells = int((gas.world_max_y - gas.world_min_y) / 256) + 1
    grid = [[[] for i in range(x_cells)] for i in range(y_cells)]
    for b in balls:
        grid[int((b.x - gas.world_min_x) / 256)][int((b.y - gas.world_min_y) / 256)].append(b)

    for xc in range(x_cells):
        for yc in range(y_cells):
            for xp in [-1, 0, 1]:
                for yp in [-1, 0, 1]:
                    if xc + xp < 0 or xc + xp >= x_cells:
                        continue
                    if yc + yp < 0 or yc + yp >= y_cells:
                        continue
                    for b1 in grid[xc][yc]:
                        for b2 in grid[xc + xp][yc + yp]:
                            if b1.id < b2.id and gas.colliding(b1, b2):
                                set_of_collisions.add(gas.ball_pair(b1, b2))
    return set_of_collisions

import gas
gas.detect_collisions = detect_collisions
if __name__ == "__main__":
    gas.main()
```

Questions
Better have some!
• 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