Using statement-comments

// (b) Base case. For b[m..n] of size 0, do nothing. // For b[m..n] of size 1, store a new block ...

// (c) Store in k the smallest value that satisfies both of // the following two conditions: ...

// (d) Create two BoundingBoxes for the left and right // parts—split box along its longer side.

// (e) Recursively allocate nodes b[m..k] and b[k+1..n]...

Place these comments just before the statements that implement them, with a blank line after the implementation.

Using statement-comments

// (b) Base case. For b[m..n] of size 0, do nothing. // For b[m..n] of size 1, store a new block ...

if (m > n) return;
if (n == m) {
    Color color= new Color(0, 0, 127);
    b.get(m).block= new Block(bbox, color);
    return;
}

// (c) Store in k the smallest value that satisfies …
Wrapper2 wrapper= getSplit(b, m, n);
int k= wrapper.k;

Using statement-comments

// (d) Create two BoundingBoxes for the left and right // parts—split box along its longer side.
BoundingBox head= new BoundingBox(bbox);
BoundingBox tail= new BoundingBox(bbox);

if (...) {
    ...
} else {
    ...
}

// (e) Recursively allocate nodes b[m..k] and b[k+1..n]...
sliceAndDice(b, m, k, head, w, h);
sliceAndDice(b, k + 1, n, tail, w, h);

Can read at two levels. Read series of green statement-comments to see what is being done. Read the code under a statement-comment, to see how it is done.

Shortest Paths in Graphs

Problem of finding shortest (min-cost) path in a graph occurs often

- Find shortest route between Ithaca and West Lafayette, IN
- Result depends on notion of cost
  - Least mileage… or least time… or cheapest
  - Perhaps, expends the least power in the butterfly while flying fastest
  - Many “costs” can be represented as edge weights

Every time you use googlemaps to find directions you are using a shortest-path algorithm

Dijkstra’s shortest-path algorithm

Edsger Dijkstra, in an interview in 2010 (CACM):

... the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiance, and tired, we sat down on the cafe terrace to drink a cup of coffee, and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a 20-minute invention. [Took place in 1956]

Visit http://www.dijkstrascry.com for all sorts of information on Dijkstra and his contributions. As a historical record, this is a gold mine.
Dijkstra’s shortest-path algorithm

Dijkstra describes the algorithm in English:
- When he designed it in 1956 (he was 26 years old), most people were programming in assembly language!
- Only one high-level language: Fortran, developed by John Backus at IBM and not quite finished.
No theory of order-of-execution time — topic yet to be developed.
In paper, Dijkstra says, “my solution is preferred to another one … “the amount of work to be done seems considerably less.”


1968 NATO Conference on Software Engineering

- In Garmisch, Germany
- Academicians and industry people attended
- For first time, people admitted they did not know what they were doing when developing/testing software. Concepts, methodologies, tools were inadequate, missing
- The term software engineering was born at this conference.
- The NATO Software Engineering Conferences:
  http://homepages.cs.ncl.ac.uk/brian.randell/NATO/index.html
  Get a good sense of the times by reading these reports!

1968/69 NATO Conferences on Software Engineering

Beards

The reason why some people grow aggressive tufts of facial hair is that they do not like to show the chin that isn’t there.

a grook by Piet Hein

Editors of the proceedings

Edsger Dijkstra, Niklaus Wirth, Tony Hoare, David Gries

From Gries to Foster

Use googlemaps to find a bicycle route from Gries’s to Foster’s house.

Gives three routes for bicycles, depending on what is to be minimized.

Miles?
Driving time?
Use of highways?
Scenic routes?
Each intersection is a node of the graph, and each road between two intersections has a weight.

distance? time to traverse? ...

Fan out from the start node (kind of breadth-first search)

Settled set: Nodes whose shortest distance is known.

Frontier set: Nodes seen at least once but shortest distance not yet known.

Settled set: we know their shortest paths
Frontier set: We know some but not all information

Each iteration:
1. Move to the Settled set: a Frontier node with shortest distance from start node.
2. Add neighbors of the new Settled node to the Frontier set.

Fan out from start node. Recording shortest distance from start seen so far

Settled set:
Frontier set:
1 2
2. Add neighbors of new Settled node to Frontier
Shortest path?

1. Fan out from start node. Recording shortest distance from start seen so far
   - Settled set: 1
   - Frontier set: 2

2. Add neighbors of new Settled node to Frontier (there are none)

Shortest path?

1. Move to Settled set a Frontier node with shortest distance from start

Shortest path?

1. Move to Settled set a Frontier node with shortest distance from start

Dijkstra’s shortest path algorithm

The n (> 0) nodes of a graph numbered 0..n-1.

Each edge has a positive weight.

weight(v1, v2) is the weight of the edge from node v1 to v2.

Some node v be selected as the start node.

Calculate length of shortest path from v to each node.

Use an array L[0..n-1]: for each node w, store in L[w] the length of the shortest path from v to w.

\[
\begin{align*}
L[0] &= 2 \\
L[1] &= 5 \\
L[2] &= 6 \\
L[3] &= 7 \\
L[4] &= 0
\end{align*}
\]
Dijkstra’s shortest path algorithm

Develop algorithm, not just present it.

Need to show you the state of affairs — the relation among all variables — just before each node i is given its final value L[i].

This relation among the variables is an invariant, because it is always true.

Because each node i (except the first) is given its final value L[i] during an iteration of a loop, the invariant is called a loop invariant.

\[ L[0] = 2 \]
\[ L[1] = 5 \]
\[ L[2] = 6 \]
\[ L[3] = 7 \]
\[ L[4] = 0 \]

1. For a Settled node s, L[s] is length of shortest v \( \rightarrow s \) path.
2. All edges leaving S go to F.
3. For a Frontier node f, L[f] is length of shortest v \( \rightarrow f \) path using only red nodes (except for f).
4. For a Far-off node b, L[b] = \( \infty \).

\[ L[v] = 0, L[w] > 0 \text{ for } w \neq v \]

Theorem: For a node f in F with min L value, L[f] is shortest path length

Loopy question 1:
How does the loop start? What is done to truthify the invariant?

Loopy question 2:
How is progress toward termination accomplished?

Loopy question 3:
How is progress toward termination accomplished?
The algorithm

1. For s, L[s] is length of shortest v → s path.
2. Edges leaving S go to F.
3. For f, L[f] is length of shortest v → f path using red nodes (except for f).
4. For b, L[b] = ∞
5. L[v] = 0, L[w] > 0 for w ≠ v

Theorem: For a node f in F with min L value, L[f] is shortest path length.

Loopy question 4:
How is the invariant maintained?

Final algorithm

1. No need to implement S.
2. Implement F as a min-heap.
3. Instead of ∞, use Integer.MAX_VALUE.

Execution time

n nodes, reachable from v, e ≥ n-1 edges
n-1 ≤ e ≤ n^2

For all w, L[w] = ∞; L[v] = 0;
F = (v); S = {};
while F ≠ {} {
  f = node in F with min L value;
  Remove f from F, add it to S;
  for each edge (f,w) {
    if (L[w] is ∞) add w to F;
    if (L[f] + weight(f,w) < L[w])
      L[w] = L[f] + weight(f,w);
  }
}
Algorithm is finished

Loopy question 4:
How is the invariant maintained?

Final algorithm

1. No need to implement S.
2. Implement F as a min-heap.
3. Instead of ∞, use Integer.MAX_VALUE.