We give you class ArrayHeaps for a reason:

It shows the simplest way to write methods like bubble-up and bubble-down. It gives you a method to get the smaller child.

You can write A6 most easily by adapting the ArrayHeap methods to work in the new environment! Do the assignment without looking at ArrayHeap makes it MUCH harder!

Look at all the notes in the pinned Piazza note A6 FAQ before beginning —and then every other day to see whether new info has been added.

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 or the GPS system on your smartphone 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.
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.
  Get a good sense of the times by reading these reports!

1968/69 NATO Conferences on Software Engineering

Editors of the proceedings

Edsger Dijkstra   Niklaus Wirth   Tony Hoare   David Gries

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

From Gries to Tate

Googlemaps: find a route from Gries’s to Tate’s house.

Gives two routes
12 minutes, 7.3 miles
15 minutes, 6.6 miles

Shortest path?

Each intersection is a node of the graph, and each road between two intersections has a weight

distance?  
time to traverse?  
...

Shortest path?

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
Shortest path?

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.
2. Add neighbors of the new Settled node to the Frontier set.

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

Settled set:
Frontier set:

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

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

Settled set:
Frontier set:

2. Add neighbors of new Settled node to Frontier

Fan out from start, recording shortest distance seen so far

Settled set:
Frontier set:

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

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

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 \( f \)).

\[
\begin{align*}
L[0] &= 2 \\
L[1] &= 5 \\
L[2] &= 6 \\
L[3] &= 7 \\
L[4] &= 0 
\end{align*}
\]
For a Settled node s, L[s] is length of shortest v → r path.

All edges leaving S go to F.

For a Frontier node f, L[f] is length of shortest v → f path using only Settled nodes (except for f).

### Theorem
For a node f in F with minimum L value (over nodes in F), L[f] is the length of a shortest path from v to f.

Case 1: v is in S.
Case 2: v is in F. Note that L[v] is 0; it has minimum L value.

The algorithm

```plaintext
while F ≠ {} {
    f = node in F with min L value;
    Remove f from F, add it to S;
    for each edge (f, w) {
        if (w not in S or F) {
            L[w] = L[f] + wgt(f, w);
            add w to F;
        } else {
            break;
        }
    }
}
```

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

Loopy question 2:
When does loop stop? When is array L completely calculated?

Loopy question 3:
Progress toward termination?

Loopy question 4:
Maintain invariant?
The algorithm

1. For s, L[s] is length of shortest v \rightarrow s path.
2. Edges leaving S go to F.
3. For f, L[f] is length of shortest v \rightarrow f path using red nodes (except for f).

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

Loopy question 4: Maintain invariant?

S
F
Far off

S = \{ \}; F = \{ v \}; L[v] = 0; while F ≠ \{ f \} \{ 

f = node in F with min L value; Remove f from F, add it to S;
for each edge (f, w) \{ 

if (w not in S or F) 

L[w] = L[f] + wgt(f, w); add w to F;
} else 

if (L[f] + wgt(f, w) < L[w]) 

L[w] = L[f] + wgt(f, w);
}

Algorithm is finished!

S
F
Far off

S = \{ \}; F = \{ v \}; L[v] = 0; while F ≠ \{ f \} \{ 

f = node in F with min L value; Remove f from F, add it to S;
for each edge (f, w) \{ 

if (w not in S or F) 

L[w] = L[f] + wgt(f, w); add w to F;
} else 

if (L[f] + wgt(f, w) < L[w]) 

L[w] = L[f] + wgt(f, w);
}

Examples:

- For each statement, calculate the average TOTAL time it takes to execute it.
- \( F \) is evaluated \( n+1 \) times. \( O(n) \)
- \( w \) not in map is evaluated \( c \) times (once for each edge).
- It's true \( n-1 \) times
- It's false \( c \) times

For each statement, calculate the average TOTAL time it takes to execute it.

- \( F \) is evaluated \( n+1 \) times. \( O(n) \)
- \( w \) not in map is evaluated \( c \) times (once for each edge).
- It's true \( n-1 \) times
- It's false \( c \) times
\begin{align*}
&\text{n nodes, reachable from } v. \quad e \geq n - 1 \text{ edges.} \\
&n - 1 \leq e \leq n^2 \\
&F = \{ v \}; L[v] = 0; \text{ add } v \text{ to map} \quad O(1) \\
&\text{while } F \neq \{ \} \quad O(n) \\
&\text{if node in } F \text{ with min } L \text{ value; } \quad O(n) \\
&\text{Remove } f \text{ from } F; \quad O(n \log n) \\
&\text{for each edge } (f, w) \quad O(n + e) \\
&\quad \text{if } (w \text{ not in map}) \quad O(n + e) \\
&\quad \quad L[w] = L[f] + \text{wgt}(f, w); \quad O(n) \\
&\quad \quad \text{add } w \text{ to } F; \quad O(n \log n) \\
&\text{else} \quad O(n) \\
&\quad \quad \text{if } (L[f] + \text{wgt}(f, w) < L[w]) \quad O(n) \\
&\quad \quad \quad L[w] = L[f] + \text{wgt}(f, w); \quad O(n \log n) \\
&\quad \quad \text{Remove } f \text{ from } F; \quad O(n \log n) \\
&\text{Condition evaluated } n + 1 \text{ times.} \\
&\text{outer loop: } n \text{ iterations.} \\
&\text{Condition evaluated } n + 1 \text{ times.} \\
&\text{inner loop: } e \text{ iterations.} \\
&\text{Condition evaluated } n + e \text{ times.} \\
\end{align*}