Analysis and Design of Algorithms

Tutorial

Week 4

How to use invariant to show the correctness of an iterative algorithm?

To understand the correctness of an algorithm using an invariant, we need to show three things:

- **Initialization:** The invariant is true before the first iteration of the loop
- **Maintenance:** If the invariant is true before an iteration, it remains true before the next iteration
- **Termination:** When the algorithm terminates, the invariant provides a useful property for showing correctness.

Question 1

Dijkstra(G, s)

1. For all $u \in V \backslash \{s\}$, $d(u)=\infty$;

2. $d(s)=0$; $R=\{\}$;

3. While $R \neq V$

4.   pick $u \notin R$ with the smallest $d(u)$

5. $R = R \cup \{u\}$

6.   for all neighbor v of u,

7. $d(v) = \min\{ d(v), d(u)+w(u,v))\}$

G=(V,E) is an undirected graph.
s is the start node
Assume all edges in G are of positive weights.

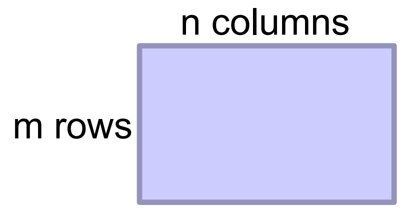
What is the invariant for the while loop?

Can you show that this algorithm correctly compute the shortest distance from s to all nodes?

The divide-and-conquer design paradigm (Recap)

1. Divide the problem (instance) into subproblems.
2. Conquer the subproblems by solving them recursively.
3. Combine subproblem solutions.

Question 2



Suppose we are given a 2D-array A of size m rows by n columns. An element in the array $A[i][j]$ is called a "peak" if it is **greater than or equal to** its adjacent neighbours (if they exist -- an element is always considered greater than or equal to non-existent elements). For example, the 8 in the middle is a peak in the following array.

$$\begin{array}{cccc} * & * & 5 & * \\ * & 8 & 8 & 3 \\ * & * & 2 & * \end{array}$$

Consider the following algorithm to return any "peak":

```
Find2DPeak(A):
    If A only has a column, return the maximal element of the column
    Otherwise:
        Select the middle column of the A
        Find the maximal element of the column
        If the maximal element is a peak, return that element
        Else
            p1 = Find2DPeak(right half of A excluding middle col)
            p2 = Find2DPeak(left half of A excluding middle col)
            If p1 or p2 is a peak, return either one, otherwise return None
```

What is the runtime of the algorithm?

- $\Theta(mn)$
- $\Theta(m \lg n)$
- $\Theta(\lg m \lg n)$
- $\Theta(m^2 \lg n)$

Design a faster Algorithm

Suppose we are given a 2D-array A of size m rows by n columns. An element in the array $A[i][j]$ is called a "peak" if it is **greater than or equal to** its adjacent neighbours (if they exist -- an element is always considered greater than or equal to non-existent elements). For example, the 8 in the middle is a peak in the following array.

*	*	5	*
*	8	8	3
*	*	2	*

Consider the following algorithm to return any "peak":

```
Find2DPeak(A):
    If A only has a column, return the maximal element of the column
    Otherwise:
        Select the middle column of the A
        Find the maximal element of the column
        If the maximal element is a peak, return that element
        Else
            p1 = Find2DPeak(right half of A excluding middle col)
            p2 = Find2DPeak(left half of A excluding middle col)
            If p1 or p2 is a peak, return either one, otherwise return None
```

1. Can we reduce the number of subproblems?
2. Can we reduce our search to only one half?

Question 3

Suppose we are given a 2D-array A of size *m* rows by *n* columns. An element in the array $A[i][j]$ is called a "peak" if it is **greater than or equal to** its adjacent neighbours (if they exist -- an element is always considered greater than or equal to non-existent elements). For example, the 8 in the middle is a peak in the following array.

$$\begin{matrix} * & * & 5 & * \\ * & 8 & 8 & 3 \\ * & * & 2 & * \end{matrix}$$

Let $A[i][j]$ be the largest element in column j. Assume that $A[i][j+1] \geq A[i][j]$. Argue that any peak in the the subarray $A[1..m][j+1..n]$ that is the largest element in its column is also a peak of the entire array A.

Question 4

Suppose we are given a 2D-array A of size *m* rows by *n* columns. An element in the array $A[i][j]$ is called a "peak" if it is **greater than or equal to** its adjacent neighbours (if they exist -- an element is always considered greater than or equal to non-existent elements). For example, the 8 in the middle is a peak in the following array.

$$\begin{array}{cccc} * & * & 5 & * \\ * & 8 & 8 & 3 \\ * & * & 2 & * \end{array}$$

Using the idea in Question 3, describe an algorithm that is asymptotically faster that the one given in Question 2. What is it's runtime?