Colorado State University

CS 320
Algorithms: Theory and Practice
PA1: Line-of-Sight

Sanjay Rajopadhye
Colorado State University

Aug 2023

Problem statement

Given

- an array, X[i,j] of the elevations of points in a (hilly) terrain, and
- information about where the sun currently is,
determine, for each point, whether it is sunlit or in the shade.

Also called the line-of-sight problem.

Imagine that you were positioned at the sun (beware Icarus) then which points in the hilly terrain would be in your line of sight and which would be hidden from view

Specification

Inputs:

- Y[i, j] is an n x n array of (floating point) numbers (in meters)
- The angle of elevation of the sun $\Theta \leq 90°$
- The angle of azimuth of the sun, Φ
- The horizontal distance (in meters) between adjacent points, d, (the resolution or scale of our data)

Output:

- S[i, j] an n x n array of Booleans:
  - If [i, j] is in the shade, S[i, j] is 1
  - Otherwise it is 0

Simplifying assumptions & conventions:

- The azimuth is due west, $\Phi = 270°$. So only points to the west (i.e., on the i^{th} row) can cast a shadow on [i, j]
- So, focus on just the i^{th} row of Y, which we re-name as R, a 1-dimensional array (an outer loop iterates over each row). This Simplifies notation/figures on next few slides

Algorithmic Approaches

Use predicate logic and some simple reasoning. And remember that we only look at the i^{th} row.

- A point at j is in the shade, if some point to its west casts a shadow on it, i.e.,

$$S[j] = \exists k: 0 \leq k < j, \quad \frac{R[k] - R[j]}{d(j-k)} > \tan \Theta$$

- First algorithm implements this as a loop (quadratic time per row)
- Second algorithm does an "early exit:" as soon as we find a point that puts j in the shade, we exit the loop
- Next, we improve the complexity. Change the existential ∃ to universal ∀ and use negation. Exploit a running max.

Prefix Computations

Some easy problems

- Add up n elements of an array $\Theta(n)$
- Max of all elements in an array $\Theta(n)$

```
r = 0
for i in range(length(X)):
  r += X[i]
```

```
r = 0; // minus infinity
for i in range(length(X)):
  r = max (r, X[i])
```

What if you wanted all *intermediate sums/maxima*

$$Y[i] = \sum_{j=0}^{i} X[j]$$

Lower bound? $\Omega(n)$

First (direct) algorithm? $O(n^2)$

Can do better? $O(n)$

```
for i in range(length(X)):
  Y[i] = 0
  for k in range(i)
    Y[i] += X[k]
```

```
Y[0] = X[0]
for i in range(1,length(X)):
    Y[i] = Y[i-1] + X[i]
```

Colorado State University

Running Max Improvement

Calculate the negation: j is sunny if

$$\forall k: 0 \le k < j, \qquad \frac{R[k] - R[j]}{h(j-k)} \le \tan\Theta$$

Move all terms involving j and k on opposite sides

$$\forall k: 0 \le k < j, \qquad R[j] + hj\tan\Theta \ge R[k] + hk\tan\Theta$$

LHS is independent of the quantified variable. Distribute it and use max (all elements in a set are less than some, value v if and only if the maximum element in the set is less than v

$$R[j] + hj\tan\Theta \ge \max_{0 \le k < j} R[k] + hk\tan\Theta$$

Calculate the RHS using the running max idea

Rules of the game

- You will write the program in python and check it in using the Checkin tab on ~cs320
- Automatic Grading issues:
  - Need to evaluate 150 programs
  - Correctness alone is not enough, your algorithm must exhibit the right asymptotic complexity – quadratic/cubic, and in some cases we must know the right constant factor.
  - We analyzed the complexity using the number of comparisons (running max), which can be implemented using an if-then-else
- Main challenge: how can a grading script count the number of times your program evaluates an if-then-else

Rules of the game

- We can't do it. Smart solutions welcome

Workaround

- Thou shalt not program with if-then-else
- Thou shalt use a special function that will be provided
- When that function is executed, it also updates a global counter (initialized to 0 before the grading script calls your function)