Alternate Analysis

Use amortized analysis on multiple calls to `push`, `pop`:

Cannot pop more elements than get pushed onto the stack.

After many pushes, a single pop has high **potential**.

Once that potential has been expended, it is not available for future `pop` operations.

The cost for m_1 pushes and m_2 pops:

$$m_1 + (m_2 + m_1) = O(m_1 + m_2)$$

Alternate Analysis
Use amortized analysis on multiple calls to push, pop:
Cannot pop more elements than get pushed onto the stack.
After many pushes, a single pop has high potential.
Once that potential has been expended, it is not available for future pop operations.
The cost for m_1 pushes and m_2 pops:
$m_1 + (m_2 + m_1) = O(m_1 + m_2)$

Actual number of (constant time) push calls + (Actual number of pop calls + Total potential for the pops)

CLR has an entire chapter on this – we won't go into this much, but we use Amortized Analysis implicitly sometimes.

Creative Design of Algorithms by Induction

Analogy: Induction $\leftrightarrow$ Algorithms

Begin with a problem:

- "Find a solution to problem Q."

Think of Q as a set containing an infinite number of **problem instances**.

Example: Sorting

- Q contains all finite sequences of integers.

Creative Design of Algorithms by Induction
Analogy: Induction $\leftrightarrow$ Algorithms
Begin with a problem:
▲ "Find a solution to problem Q."
Think of Q as a set containing an infinite number of problem instances.
Example: Sorting
▲ Q contains all finite sequences of integers.

Now that we have completed the tool review, we will do two things:

1. Survey algorithms in application areas
2. Try to understand how to create efficient algorithms

This chapter is about the second. The remaining chapters do the second in the context of the first.

I $\leftarrow$ A is reasonably obvious – we often use induction to prove that an algorithm is correct. The intellectual claim of Manber is that I $\rightarrow$ A gives insight into problem solving.

Solving Q

First step:

- Parameterize problem by size: $Q(n)$

Example: Sorting

- $Q(n)$ contains all sequences of n integers.

Q is now an infinite sequence of problems:

- $Q(1), Q(2), ..., Q(n)$

Algorithm: Solve for an instance in $Q(n)$ by solving instances in $Q(i), i < n$ and combining as necessary.

Solving Q
First step:
▸ Parameterize problem by size: $Q(n)$
Example: Sorting
▸ $Q(n)$ contains all sequences of n integers.
Q is now an infinite sequence of problems:
▸ $Q(1), Q(2), ..., Q(n)$
Algorithm: Solve for an instance in $Q(n)$ by solving instances in $Q(i), i < n$ and combining as necessary.

This is a "meta" algorithm – An algorithm for finding algorithms!

Induction

Goal: Prove that we can solve for an instance in $Q(n)$ by assuming we can solve instances in $Q(i), i < n$.

Don't forget the base cases!

Theorem: $\forall n \geq 1$, we can solve instances in $Q(n)$.

- This theorem embodies the **correctness** of the algorithm.

Since an induction proof is mechanistic, this should lead directly to an algorithm (recursive or iterative).

Just one (new) catch:

- Different inductive proofs are possible.
- We want the most **efficient** algorithm!

Induction
Goal: Prove that we can solve for an instance in $Q(n)$ by assuming we can solve instances in $Q(i), i < n$.
Don't forget the base cases!
Theorem: $\forall n \geq 1$, we can solve instances in $Q(n)$.
▸ This theorem embodies the correctness of the algorithm.
Since an induction proof is mechanistic, this should lead directly to an algorithm (recursive or iterative).
Just one (new) catch:
▸ Different inductive proofs are possible.
▸ We want the most efficient algorithm!

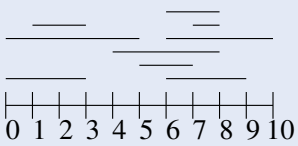
The goal is using Strong Induction.
Correctness is proved by induction.
Example: Sorting

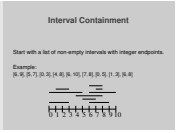
- Sort $n-1$ items, add nth item (insertion sort)
- Sort 2 sets of $n/2$, merge together (mergesort)
- Sort values $< x$ and $> x$ (quicksort)

Interval Containment

Start with a list of non-empty intervals with integer endpoints.

Example:
$[6,9], [5,7], [0,3], [4,8], [6,10], [7,8], [0,5], [1,3], [6,8]$



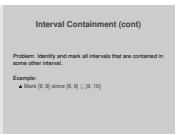


no notes

Interval Containment (cont)

Problem: Identify and mark all intervals that are contained in some other interval.

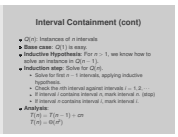
Example:
- Mark $[6,9]$ since $[6,9] \subseteq [6,10]$



$[5,7] \subseteq [4,8]$
$[0,3] \subseteq [0,5]$
$[7,8] \subseteq [6,10]$
$[1,3] \subseteq [0,5]$
$[6,8] \subseteq [6,10]$
$[6,9] \subseteq [6,10]$

Interval Containment (cont)

- $Q(n)$: Instances of n intervals
- **Base case**: $Q(1)$ is easy.
- **Inductive Hypothesis**: For $n > 1$, we know how to solve an instance in $Q(n-1)$.
- **Induction step**: Solve for $Q(n)$.
  - Solve for first $n-1$ intervals, applying inductive hypothesis.
  - Check the nth interval against intervals $i = 1, 2, \cdots$
  - If interval i contains interval n, mark interval n. (stop)
  - If interval n contains interval i, mark interval i.
- **Analysis**:
  $T(n) = T(n-1) + cn$
  $T(n) = \Theta(n^2)$



Base case: Nothing is contained

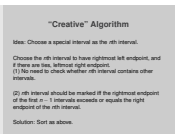
"Creative" Algorithm

Idea: Choose a special interval as the nth interval.

Choose the nth interval to have rightmost left endpoint, and if there are ties, leftmost right endpoint.
(1) No need to check whether nth interval contains other intervals.

(2) nth interval should be marked iff the rightmost endpoint of the first $n-1$ intervals exceeds or equals the right endpoint of the nth interval.

Solution: Sort as above.



In the example, the nth interval is $[7,8]$.
Every other interval has left endpoint to left, or right endpoint to right.
We must keep track of the current right-most endpont.

“Creative” Solution Induction

Induction Hypothesis: Can solve for $Q(n-1)$ AND interval n is the “rightmost” interval AND we know R (the rightmost endpoint encountered so far) for the first $n-1$ segments.

Induction Step: (to solve $Q(n)$)

- Sort by left endpoints
- Solve for first $n-1$ intervals recursively, remembering R.
- If the rightmost endpoint of nth interval is $\leq$ R, then mark the nth interval.
- Else R $\leftarrow$ right endpoint of nth interval.

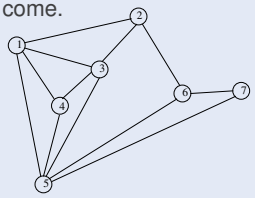
Analysis: $\Theta(n \log n) + \Theta(n)$.
Lesson: Preprocessing, often sorting, can help sometimes.

“Creative” Solution Induction
Induction Hypothesis: Can solve for $Q(n-1)$ AND interval n is the “rightmost” interval AND we know R (the rightmost endpoint encountered so far) for the first $n-1$ segments.
Induction Step: (to solve $Q(n)$)
Sort by left endpoints
Solve for first $n-1$ intervals recursively, remembering R.
If the rightmost endpoint of nth interval is $\leq$ R, then mark the nth interval.
Else R $\leftarrow$ right endpoint of nth interval.
Analysis: $\Theta(n \log n) + \Theta(n)$.
Lesson: Preprocessing, often sorting, can help sometimes.

We strengthened the induction hypothesis. In algorithms, this does cost something.
We must sort.
Analysis: Time for sort + constant time per interval.

Maximal Induced Subgraph

Problem: Given a graph $G = (V, E)$ and an integer k, find a maximal induced subgraph $H = (U, F)$ such that all vertices in H have degree $\geq k$.
Example: Scientists interacting at a conference. Each one will come only if k colleagues come, and they know in advance if somebody won't come.
Example: For $k = 3$.



Solution:

Maximal Induced Subgraph
Problem: Given a graph $G = (V, E)$ and an integer k, find a maximal induced subgraph $H = (U, F)$ such that all vertices in H have degree $\geq k$.
Example: Scientists interacting at a conference. Each one will come only if k colleagues come, and they know in advance if somebody won't come.
Example: For $k = 3$.
Solution:

Induced subgraph: U is a subset of V, F is a subset of E such that both ends of $e \in E$ are members of U.
Solution is: $U = \{1, 3, 4, 5\}$

Max Induced Subgraph Solution

$Q(s, k)$: Instances where $|V| = s$ and k is a fixed integer.

Theorem: $\forall s, k > 0$, we can solve an instance in $Q(s, k)$.

Analysis: Should be able to implement algorithm in time $\Theta(|V| + |E|)$.

Max Induced Subgraph Solution
$Q(s, k)$: Instances where $|V| = s$ and k is a fixed integer.
Theorem: $\forall s, k > 0$, we can solve an instance in $Q(s, k)$.
Analysis: Should be able to implement algorithm in time $\Theta(|V| + |E|)$.

Base Case: $s = 1$ H is the empty graph.
Induction Hypothesis: Assume $s > 1$. we can solve instances of $Q(s-1, k)$.
Induction Step: Show that we can solve an instance of $G(V, E)$ in $Q(s, k)$. Two cases:

(1) Every vertex in G has degree $\geq k$. $H = G$ is the only solution.
(2) Otherwise, let $v \in V$ have degree $< k$. $G - v$ is an instance of $Q(s-1, k)$ which we know how to solve.

By induction, the theorem follows.
Visit all edges to generate degree counts for the vertices. Any vertex with degree below k goes on a queue. Pull the vertices off the queue one by one, and reduce the degree of their neighbors. Add the neighbor to the queue if it drops below k.

Celebrity Problem

In a group of n people, a **celebrity** is somebody whom everybody knows, but who knows no one else.

Problem: If we can ask questions of the form “does person i know person j?” how many questions do we need to find a celebrity, if one exists?

How should we structure the information?

Celebrity Problem
In a group of n people, a celebrity is somebody whom everybody knows, but who knows no one else.
Problem: If we can ask questions of the form “does person i know person j?” how many questions do we need to find a celebrity, if one exists?
How should we structure the information?

no notes

Celebrity Problem (cont)

Formulate as an $n \times n$ boolean matrix M.
$M_{ij} = 1$ iff i knows j.

Example:
$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

A celebrity has all 0's in his row and all 1's in his column.

There can be at most one celebrity.

Clearly, $O(n^2)$ questions suffice. Can we do better?

Celebrity Problem (cont)
Formulate as an $n \times n$ boolean matrix M.
$M_{ij} = 1$ iff i knows j.
Example: $\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$
A celebrity has all 0's in his row and all 1's in his column.
There can be at most one celebrity.
Clearly, $O(n^2)$ questions suffice. Can we do better?

The celebrity in this example is 4.

Efficient Celebrity Algorithm

Appeal to induction:

- If we have an $n \times n$ matrix, how can we reduce it to an $(n-1) \times (n-1)$ matrix?

What are ways to select the n'th person?

Efficient Celebrity Algorithm
Appeal to induction:
- If we have an $n \times n$ matrix, how can we reduce it to an $(n-1) \times (n-1)$ matrix?
What are ways to select the n'th person?

This induction implies that we go backwards. Natural thing to try: pick arbitrary n'th person.
Assume that we can solve for $n-1$. What happens when we add nth person?

- Celebrity candidate in $n-1$ – just ask two questions.
- Celebrity is n – must check $2(n-1)$ positions. $O(n^2)$.
- No celebrity. Again, $O(n^2)$.

So we will have to look for something special. Who can we eliminate? There are only two choices: A celebrity or a non-celebrity. It doesn't make sense to eliminate a celebrity. Is there an easy way to guarentee that we eliminate a non-celeberity?

Efficient Celebrity Algorithm (cont)

Eliminate one person if he is a non-celebrity.

- Strike one row and one column.

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Does 1 know 3? No. 3 is a non-celebrity.
Does 2 know 5? Yes. 2 is a non-celebrity.
Observation: Each question eliminates one non-celebrity.

Efficient Celebrity Algorithm (cont)
Eliminate one person if he is a non-celebrity.
- Strike one row and one column.
$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$
Does 1 know 3? No. 3 is a non-celebrity.
Does 2 know 5? Yes. 2 is a non-celebrity.
Observation: Each question eliminates one non-celebrity.

no notes

Celebrity Algorithm

Algorithm:

1. Ask $n-1$ questions to eliminate $n-1$ non-celebrities. This leaves one candidate who might be a celebrity.
2. Ask $2(n-1)$ questions to check candidate.

Analysis:

- $\Theta(n)$ questions are asked.

Example:

- Does 1 know 2? No. Eliminate 2
- Does 1 know 3? No. Eliminate 3
- Does 1 know 4? Yes. Eliminate 1
- Does 4 know 5? No. Eliminate 5

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

4 remains as candidate.

Celebrity Algorithm
Algorithm:
1. Ask $n-1$ questions to eliminate $n-1$ non-celebrities. This leaves one candidate who might be a celebrity.
2. Ask $2(n-1)$ questions to check candidate.
Analysis:
- $\Theta(n)$ questions are asked.
Example:
- Does 1 know 2? No. Eliminate 2
- Does 1 know 3? No. Eliminate 3
- Does 1 know 4? Yes. Eliminate 1
- Does 4 know 5? No. Eliminate 5
$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$
4 remains as candidate.

no notes