Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 2

University of Southern California

Spring 2023

Review

Amortized Cost

Reading: chapters 1 & 2

Ch1: review questions

2. (T/F) Any function which is $\Omega(\log n)$ is also $\Omega(\log(\log n))$.

$\exists c_1, f(n) \geq c_1 \cdot \log n \geq c_2 \cdot \log(\log n)$

$\geq \exp$

$n \geq c_3 \cdot \log n$

3. (T/F) If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$.

$\exists c_1, c_2, \quad c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$

$g(n) \leq \frac{1}{c_1} \cdot f(n) \iff g(n) \geq \frac{1}{c_2} \cdot f(n)$

$\frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n) \Rightarrow g = \Theta(f)$

Ch1: exercises

4. Arrange the following functions

$4^{\log n}, \sqrt{\log n}, n^{\log\log n}, (\sqrt{2})^{\log n}, 2^{\sqrt{2\log n}}, n^{1/\log n}, (\log n)!$

in increasing order of growth rate with $g(n)$ following $f(n)$ in your list if and only if $f(n) = O(g(n))$.

$\sqrt{2\log n} \overset{?}{\leq} \sqrt{n}$ (is it true?)

apply log to both parts

$\sqrt{2\log n} \leq \log \sqrt{n}$

$\sqrt{2}\cdot\sqrt{\log n} \leq \frac{1}{2}\log n$ square both sides

$2\cdot\log n \leq \frac{1}{4}\log^2 n$, $n \to \infty$

Compare $n^{\log\log n}$ with $(\log n)!$

$(\log n)! = \left[x! = x(x-1)\cdots 2\cdot 1\right] =$

$= \log n \cdot (\log n - 1) \cdots (2) \cdot 1 =$

$\log_2 n = \frac{1}{\log_n 2}$

$\leq \log n \quad \leq \log n$

$! \leq \log n \cdot \log n \cdot \log n \cdot \ldots \cdot \log n =$

$= (\log n)^{\log n} = n^{\log(\log n)}$

$\log(\log n) = \left[\log n = t\right] = n^{\log t}$ (change the base)

$n^{\frac{\log_n t}{\log_n 2}} = \left(n^{\log_n t}\right)^{\frac{1}{\log_n 2}} = t^{\log n} = (\log n)^{\log n}$

Discussion Problem 1

Consider these two statements about a connected undirected graph with V vertices and E edges:

I. $O(V) = O(E)$

II. $O(E) = O(V^2)$ always true, complete graph

Mark all the correct choices below

(a) I and II are both false.

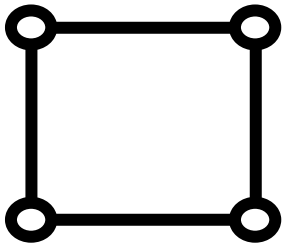
(b) Only I is true.

(c) Only II is true.

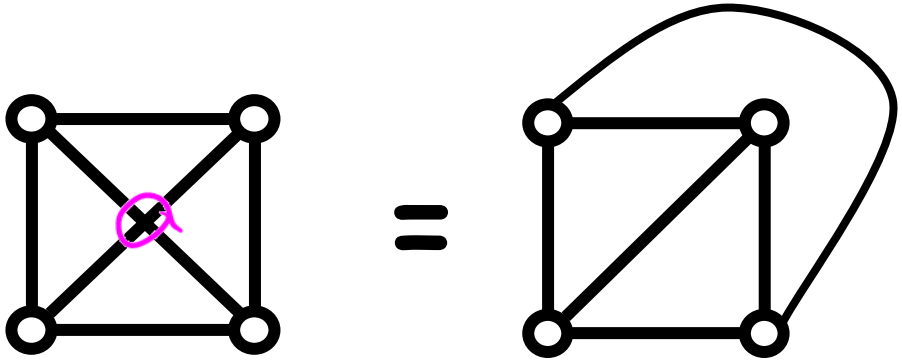
(d) I and II are both true.

Planar Graphs

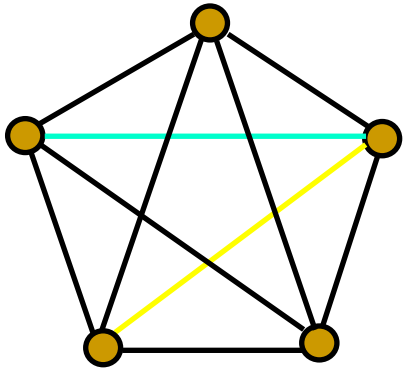
A graph is planar if it can be drawn in the plane without crossing edges

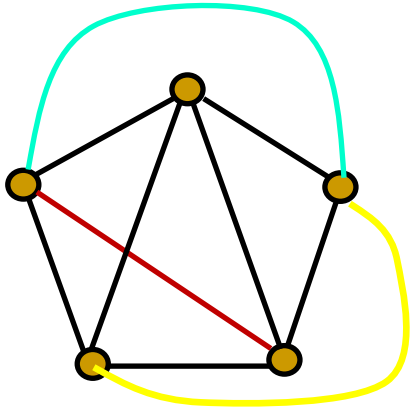


K_4 is planar



K_5 is not planar





Euler's Formula

Theorem. If G is a connected planar graph with V vertices, E edges and F faces, then $V - E + F = 2$.

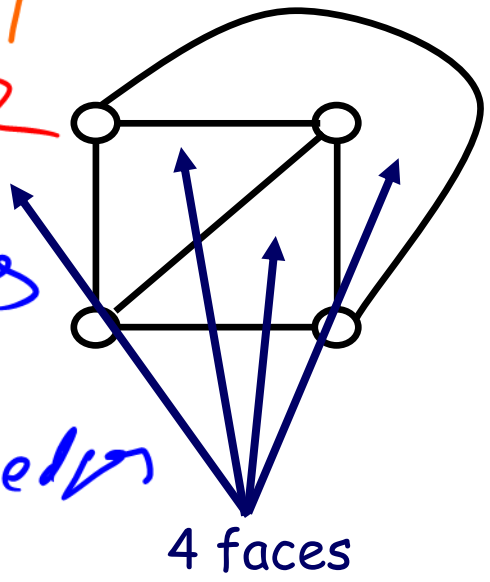
Proof by induction on edges

Base Case: $E = 1$ $\quad V = 2, F = 1$

$V - E + F = 2 - 1 + 1 = 2$

IH: Assume $V - E + F = 2$ for graphs with $E < m$ edges

IS: Prove $V - E + F = 2$ for graphs with $E = m$ edges



4 faces

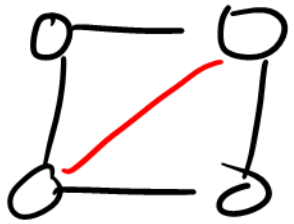
A planar graph when drawn in the plane, splits the plane into disjoint faces.

Proof of 1 S

Case 1: V unchanged

E ++

F ++



$V - E + F = 2$

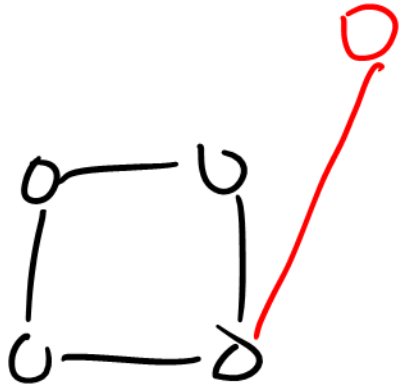
$V - (E++) + (F++) = 2$

Case 2:

V ++

E ++

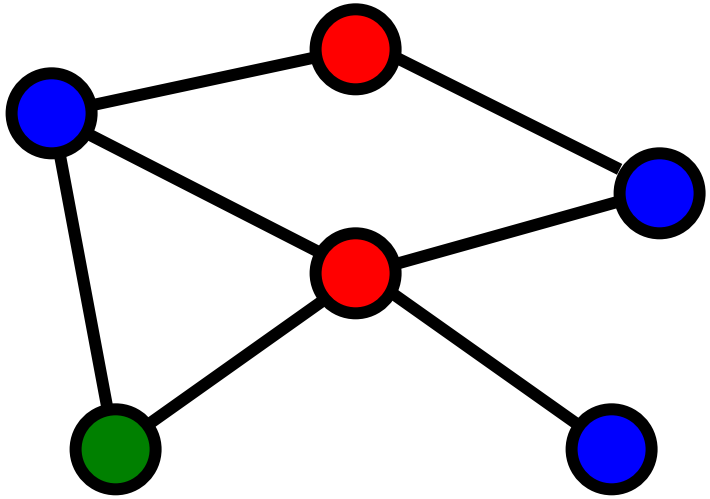
F unchanged



BREAK

Coloring Planar Graphs

A coloring of a graph is an assignment of a color to each vertex such that no neighboring vertices have the same color



4 Color Theorem (1976)

Theorem: *Any simple planar graph can be colored with less than or equal to 4 colors.*

It was proven in 1976 by K. Appel and W. Haken. They used a special-purpose computer program.

Since that time computer scientists have been working on developing a formal program proof of correctness. The idea is to write code that describes not only what the machine should do, but also why it should be doing it.

If a graph is NOT planar, the coloring problem is hard (NP-hard)

Amortized Analysis

In a sequence of operations, the worst-case does not occur often in each operation - some operations may be cheap, some may be expensive.

Therefore, a traditional worst-case per operation analysis can give overly *pessimistic* bound.

When same operation takes different times, how can we accurately calculate the runtime complexity?

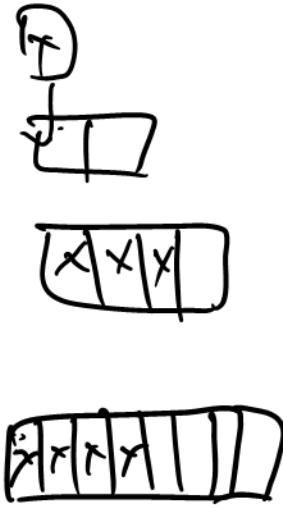
The Aggregate Method

The aggregate method computes the upper bound $T(n)$ on the total cost of n operations.

The amortized cost of an operation is given by $\frac{T(n)}{n}$

In this method each operation will get the same amortized cost, even if there are several types of operations in the sequence.

Unbounded Array



insert	old size	new size	# of copy
1	1	—	—
2		2	1
3	2	4	2
4	—	—	—
5	4	8	4
6	—	—	—
7	—	—	—
8	—	—	—
9	8	16	8

compute the total work:

inserts = 9

copy = 1+2+4+8 = 15

total = 9+15 = 24

AC of insertion is

$$\frac{\text{total cost}}{\#\text{ inserts}} = \frac{24}{9}$$

Unbounded Array

inserts $= 2^n + 1$

copy $= 1 + 2 + 4 + \ldots + 2^n = 2^{n+1} - 1$

Total cost: inserts + copies

$2^n + 1 + 2^{n+1} - 1 = 3 \cdot 2^n$

$$AC = \frac{\text{total cost}}{\#\text{inserts}} = \frac{3 \cdot 2^n}{2^n + 1}$$

$$AC = \lim_{n \to \infty} \frac{3 \cdot 2^n}{2^n + 1} = 3 = O(1)$$ constant time

Binary Counter

Given a binary number n with $\log(n)$ bits, stored as an array, where each entry $A[i]$ stores the i-th bit.

$n = 3$

The cost of incrementing a binary number is the number of bits flipped.

	flips
0 0 0	
0 0 1	1
0 1 0	2
0 1 1	1
1 0 0	3
1 0 1	1
1 1 0	2
1 1 1	1
0 0 0	3

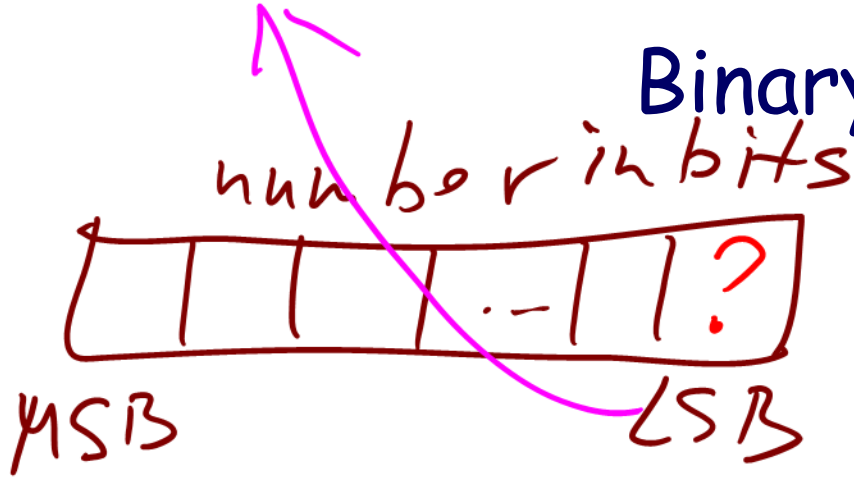
Worst-case $O(\log n)$

Best-case $O(1)$

average over a sequence of operations

$$AC = \frac{\text{total cost}}{n} = \frac{\#\text{ of flips}}{n}$$

Binary Counter



$$\sum_{k=0}^{\infty} \frac{1}{2^k} = O(1)$$

clue: look at the LSB

LSB flips n times

the previous bit $\frac{n}{2}$ times

next $\frac{n}{4}$ times

Count the total # of flips:

$$n + \frac{n}{2} + \frac{n}{4} + \dots + 2 = n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots + ?\right) \leq$$

$$\leq n\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) = n \cdot O(1) = O(n)$$

total cost: $O(n)$

# of flips: n

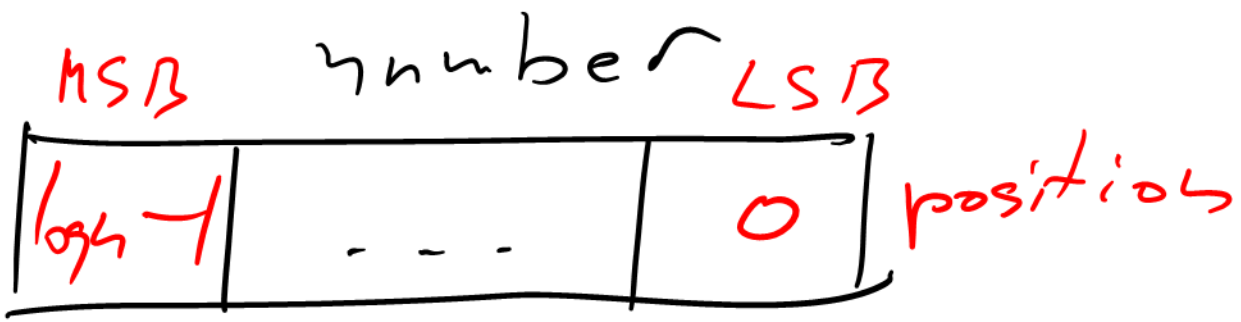
$$AC = \frac{\text{total}}{\#\text{flips}} \approx \frac{O(n)}{n} = O(1)$$

constant

BREAK

Discussion Problem 2

Another Binary Counter. Let us assume that the cost of a flip is 2^k to flip k-th bit. Flipping the lowest-order bit costs $2^0 = 1$, the next bit costs $2^1 = 2$, and so on. What is the amortized cost per increment? Use the aggregate method.



check
n = 3

flips cost

total cost = $n \times 1 + \frac{n}{2} \times 2 + \frac{n}{4} \times 4 + \ldots$

$= \underbrace{n + n + n + \ldots}_{\log n} = O(n \log n)$

$$AC = \frac{\text{total cost}}{\#\text{flips}} = \frac{O(n \log n)}{n} = O(\log n)$$

why a big-O notation?

Discussion Problem 3

Another Yet Binary Counter. Let us assume that the cost of a flip is (k+1) to flip k-th bit. Flipping the lowest-order bit costs 0 + 1 = 1, the next bit costs 1 + 1 = 2, the next bit costs 2 + 1 = 3, and so on. What is the amortized cost per operation for a sequence of *n* increments, starting from zero? What is the amortized cost per increment? Use the aggregate method.

Total cost =

$$n \times 1 + \frac{n}{2} \times 2 + \frac{n}{4} \times 3 + \frac{n}{8} \times 4 \leq$$

$$\leq n \cdot \sum_{k=0}^{\infty} \frac{k+1}{2^k} = O(n)$$

$$AC = \frac{\text{total cost}}{n} = \frac{O(n)}{n} = O(1)$$

The Accounting Method

The accounting method (or the banker's method) computes the individual cost of each operation.

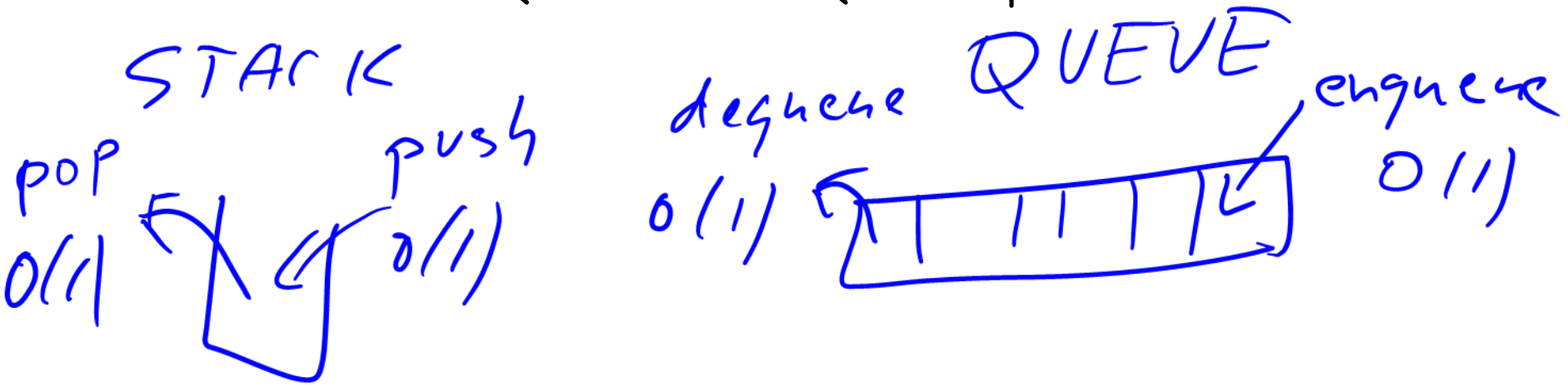
We assign different charges to each operation; some operations may charge more or less than they actually cost.

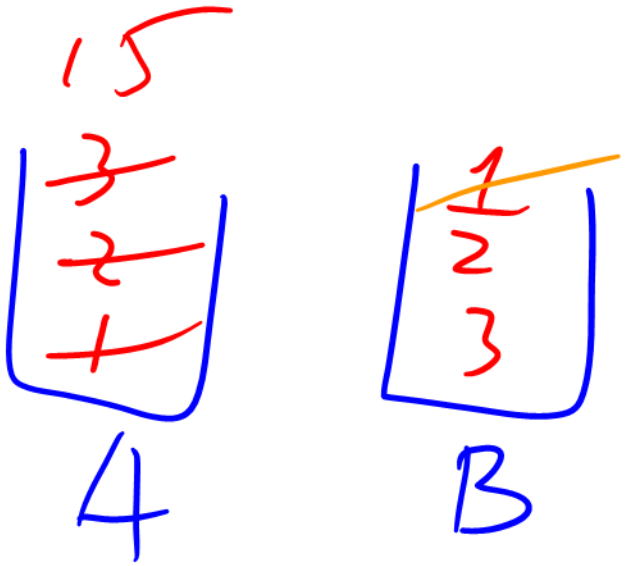
The amount we charge an operation is called its amortized cost.

Discussion Problem 4

You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE.

We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations?





enqueue: 1, 2, 3

A.push O(1)

dequeue:

loop A.pop
B.push

B.pop

worst-case

enqueue: O(1)

dequeue: O(4)

dequeue:

B.pop

enqueue: 15

A.push

Compute an amortized cost.

enqueue + dequeue = 4 tokens

① how many tokens for enqueue? 1 ~~3~~

② how many tokens for dequeue? ~~3~~ 1 ~~1~~

wrong ...

Example

enqueue: 1, 2, 3

Bank = ~~0~~ ~~3~~ 6

dequeue