Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 2

University of Southern California

Fall 2023

Review
Amortized Cost

Reading: chapters 1 & 2

Ch1: review questions

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

$\exists c,\ \underbrace{f(n) \geq c \cdot \log n \geq c_2 \cdot \log(\log n)}_{\text{exponentiate}}$?

$n \geq c_3 \cdot \log n$ True

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

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

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

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

Ch1: exercises

4. Arrange the following functions

5 2 7 4 3 1 6

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

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

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

?

?

$2^{\sqrt{2\log n}} \le \sqrt{n}$, apply log

$\sqrt{2\log n} \le \log\sqrt{n} \implies \sqrt{2}\sqrt{\log n} \le \frac{1}{2}\log n$

square both parts

$2\log n \le \frac{1}{4}\log^2 n \implies \boxed{8 \le \log n}$ True

$n \to \infty$

$$n^{\log\log n} = [\text{let } \log n = t] = n^{\log t}$$

change the base

$$= n^{\frac{\log_n t}{\log_n 2}} = t^{1/\log_n 2} = t^{\log n}$$

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

$$(\log n)! = \overbrace{\log n \cdot \underbrace{(\log n - 1)}_{\le \log n} \cdots \underbrace{2}_{\le \log n} \cdot 1}^{\log n \text{ terms}}$$

$$\le \log n \cdot \log n \cdots \log n \cdot \log n \;?$$

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

Discussion Problem 1

simple

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

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

II. $O(E) = O(V^2)$ $\exists c,\ E \le c \cdot V^2$

Mark all the correct choices below

(a) I and II are both false.

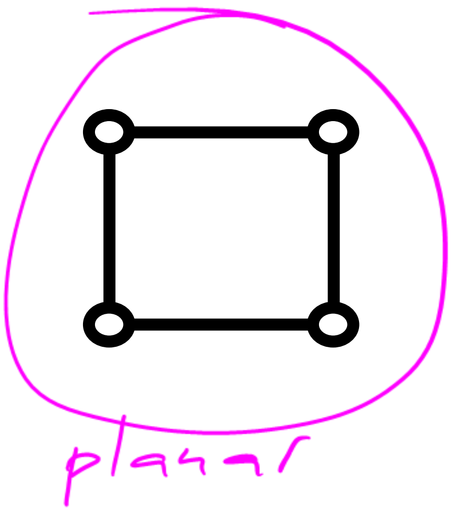
(b) Only I is true.

(c) Only II is true.

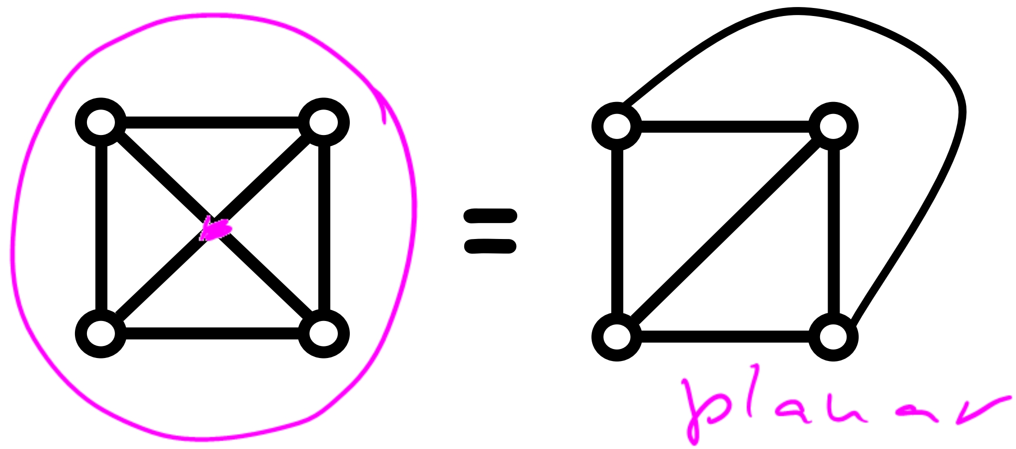
(d) I and II are both true.

Planar Graphs

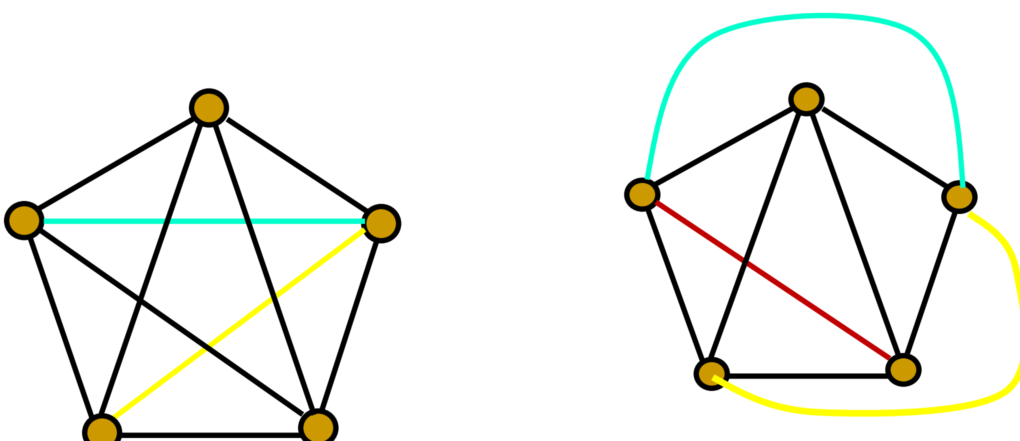
A graph is planar if it can be drawn in the plane without crossing edges



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 $V - E + F = 2$.

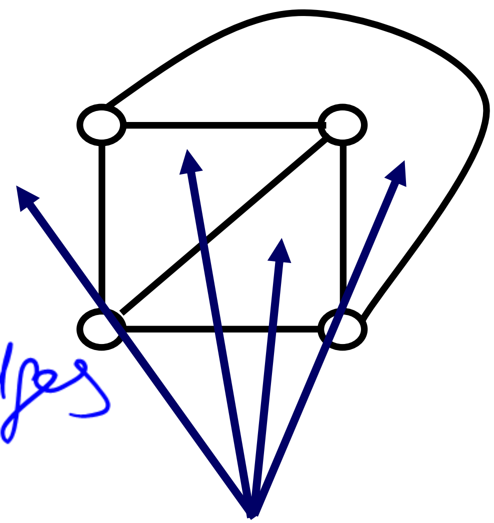
Proof. by induction on edges

Basecase: $E=0$, $V=1$, $F=1$

$E=1$, $V=2$, $F=1$

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

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



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

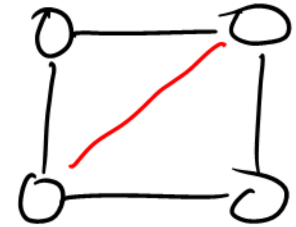
a) V - unchanged

E++

F++

$V - E + F = 2$

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



IH

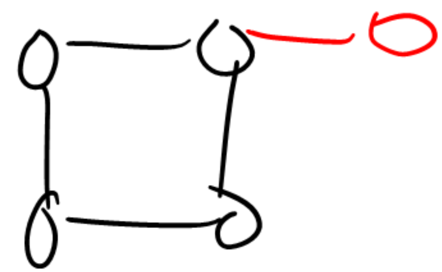
b) F - unchanged

E++

V++

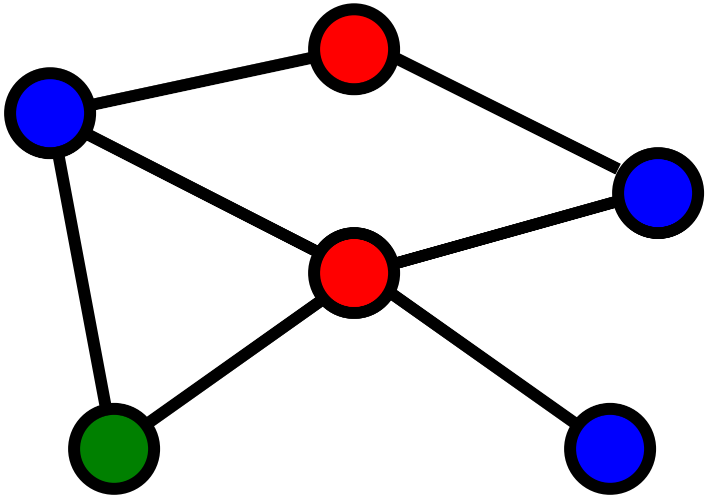
$V - E + F = 2$

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



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)

Break, 5 mins

Amortized Analysis

In a <u>sequence</u> 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?

Unbounded Array

Consider insertions into an array of size n:

some operations take O(n), others – O(1)

If the current array is full, the cost of insertion is linear;
if it is not full, insertion takes a constant time.

Amortized analysis is an alternative to the traditional worst-case analysis. Namely, we perform a worst-case analysis on a sequence of operations.

We will create a new data structure – unbounded array: when we need to insert into a full array, we allocate a new array twice as large and copy the elements we already have to the new array.

The Aggregate Method

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

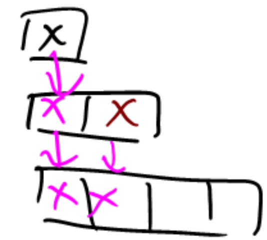
The amortized cost of an operation is given by $\frac{T(n)}{n}$

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	Copy
1	1	—	—
2	1	2	1
3	2	4	2
4	4	—	—
5	4	8	4
6	8	—	—
7	8	—	—
8	8	—	—
9	8	16	8

1.
2.
3.



4. $O(1)$
5. $O(n)$

Compute the total work

$\#$ inserts $= 9$

$\#$ copy $= 1+2+4+8 = 15$

$$AC = \frac{\text{total cost}}{\#\text{ inserts}} = \frac{15+9}{9}$$

input $\to \infty$

Unbounded Array

$\# \text{ inserts} = 2^n + 1$

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

Total cost : inserts + copy

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

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

/ constant

Unbounded Array: summary

The amortized cost of an operation is given by $\frac{T(n)}{n}$, where $T(n)$ is the upper bound on the total cost of n operations.

We considered unbounded array with a doubling-up resizing policy

Insertions:	1, 2, 3, 4, 5, 6, 7, 8, 9, …, 2^n+1
Insertion Cost:	1, 1, 1, 1, 1, 1, 1, 1, 1, …, 1
Copy Cost:	0, 1, 2, 0, 4, 0, 0, 0, 8, … , 2^n

We computed the average cost per insert: O(1)

It is important to realize that we achieve a great amortized cost just because we have implemented a clever resizing policy!

Binary Counter

$$\begin{array}{r} 001 \\ +\ \ \ 1 \\ \hline 010 \end{array}$$

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

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

	# of flips
000	1
001	2
010	1
011	3
100	1
101	2
110	1
111	3

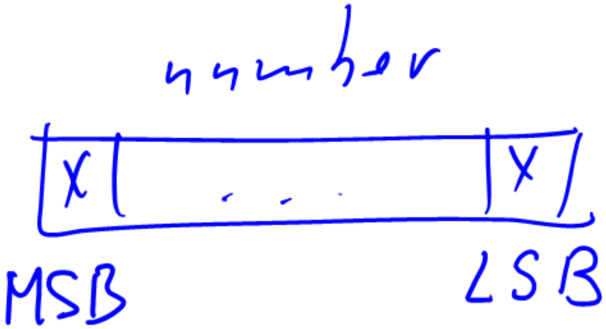
Worst-case $= O(\log n)$

Best-case $= O(1)$

Average over a sequence of increments

$$AC = \lim_{n \to \infty} \frac{\text{total cost}}{\#\text{ additions}} = \frac{14}{8}$$

Binary Counter



LSB changes n times

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

next $n/4$

MSB $= 1$

Total work: $n + \frac{n}{2} + \frac{n}{4} + \ldots + 1 =$

$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{n}\right) \leq n\Big(\underbrace{1 + \frac{1}{2} + \ldots}_{\text{infinite}}\Big) = O(n)$$

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

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.

MSB LSB

$\log n - 1$	...	1	0	index
$2^{\log n - 1}$	..	2^1	2^0	cost

Compute the total cost: $n \times 2^0 + \frac{n}{\cancel{2}} \times \cancel{2}^1 + \frac{n}{\cancel{4}} \times \cancel{2}^2 + \ldots$

$= \underbrace{n + n + n + \ldots}_{\log n}$

$= O(n \log n)$

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

Break 3 mins

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.

$\log n - 1$	...	1	0	index
$\log n$		2	1	cost

Total cost =

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

$$\le \sum_{k=0}^{\infty} n \times \frac{k+1}{2^k} = n \sum_{k=0}^{\infty} \frac{k+1}{2^k} \quad = O(1)$$

wolfram α

$$= O(n)$$

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

The Accounting Method

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

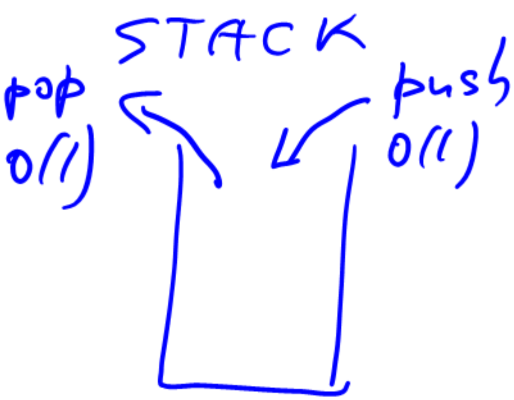
We assign different charges to each operation; some operations may charge more or less than they actually cost.

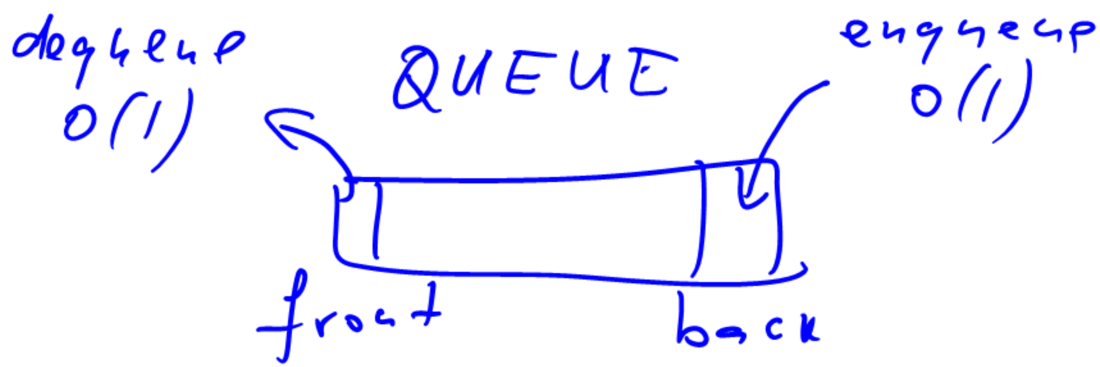
The amount we charge an operation is called its amortized cost.

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, 4

A.push $O(1)$

dequeue: if B is empty

loop $\left\{ \begin{array}{l} \text{A.pop} \\ \text{B.push} \end{array} \right.$ $O(n)$

dequeue: if B is not empty

B.pop $O(1)$

Case 1

~~enqueue — 1 tokens~~

~~dequeue — n tokens~~

wrong analysis

Case 2

enqueue — 3 tokens $O(1)$ constant

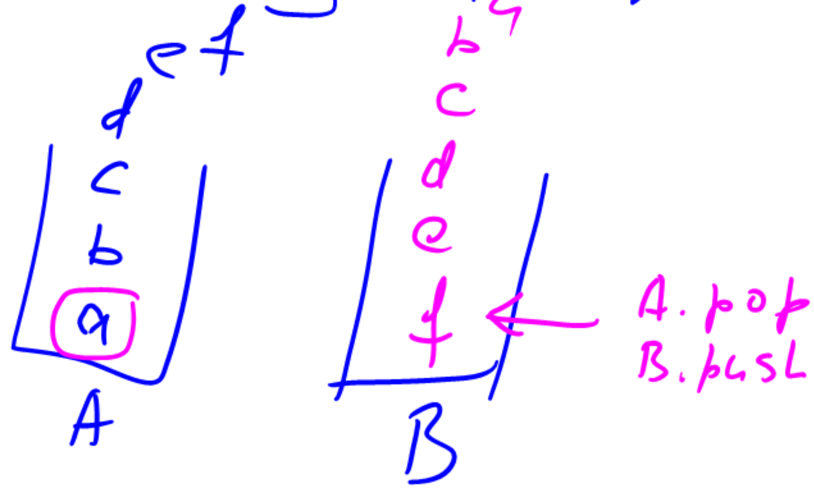
dequeue — 1 token $O(1)$ run time

3 enqueue: a, b, c, d, e, f] my sequence

1 dequeue

2+2+2+2+2+2=12

Bank



2 tokens

empty Bank