

Instructor: Antonio Miranda
Hill 363
Phone 57477
email: antonio.miranda@cs.rutgers.edu
Office hours: TBD

In this course we will study a variety of algorithms and analyze their complexity to gain insight into the principles and data-structures useful in algorithm design.

Textbook 1: *Algorithms* by Sanjoy Dasgupta, Christos Papadimitriou and Umesh Vazirani, McGraw-Hill.

Textbook 2: *Introduction to Algorithms* by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, 3rd Edition, MIT Press.

Students are responsible for knowing all the material that will be covered in class and is NOT in the book. Some lecture notes will be posted on Sakai.

Prerequisites:

Calculus and Discrete Mathematics.

Chapter 0 of Textbook 1, **Chapters 1, 2, and 3** of Textbook 2

Class Policy:

- **Exams.** The only acceptable reason for not attending an exam is a major (documented) medical emergency. NO make-ups will be given in any other case. A list of specific topics will be posted before each exam, to help you prepare.
- **Quizzes.** At the beginning of most lectures you will take a short quiz, usually it can be done in about 10 minutes.
- **Homeworks.**
 - Homeworks will be posted on Sakai.
 - Homework submissions MUST be handwritten.
 - All the pages must be scanned into ONE single file
 - The file must be uploaded to Sakai before the due date (time).
 - Make sure that your scanned any files are easily readable before submitting them.
 - NO late homeworks will be accepted

- **Regrading.** To report possible grading errors, attach a page describing the alleged error to the corresponding exam, homework, or project and submit it to the instructor or TA no later than one week after the date when the exam, homework or project was returned graded (to the class). An answer to a regrade may not be available until the end of the semester, so make copies of the materials given back for regrading. One week after the exam, homework or project was returned graded to the class, the grade becomes permanent and cannot be changed. The grade of the final exam becomes permanent one week after it is posted on Sakai.

- **Project.** The final project will be graded principally on functionality. In order to pass the course, a working programming project must be completed and handed in. Individual contributions to the project will be measured and taken into account, the instructor may request an oral examination to further evaluate a student's understanding of the material involved and the way in which the program works.

The only communication between teams should concern very general topics such as how to log in, how to install software and the like. Reusing software written by others or for other courses/projects is prohibited, unless approved by the instructor.

- **Grading.**

Homeworks	15%
quizzes	5%
2 Midterm exams	30%
Final exam	30%
Final project	20%

The grade assigned as final grade cannot be changed, even by doing additional work. In order to be fair to all students, any option to improve grades (if any) will be given to every student, NOT just to one particular student.

- **Academic Integrity.**

We take academic integrity quite seriously. Copying answers from any source including published solutions is considered academic dishonesty.

In case of learning disabilities, please provide verification from the College Coordinator. Also inform us at the beginning of the semester of any **planned absences** due to participation in professional events.

- **Sakai.**

Sakai will be used for weekly announcements related to quizzes, homeworks, midterms, final exam, etc. If you are duly registered for the class you will get emails alerting you of these periodic sakai announcements.

Topics:

Topic	Description
Complexity Measures	Methods for expressing and comparing complexity of algorithms
Searching and Sorting	Lower bounds for comparison-based sorting; counting sort and radix sort
Divide and Conquer	Fast integer multiplication, recurrences, the Master Theorem, mergesort, median and selection algorithms, quicksort, heapsort, fast matrix multiplication
Graph Search Algorithms	Graph representations, depth first search, strongly connected components, breadth first search and layered DAGs
Shortest Paths in digraphs	Single-source shortest paths for non-negative edge weights, priority queues and Dijkstra's, single source shortest paths for general weights
Greedy Algorithms	Spanning trees and cuts, union-find and path compression, minimum spanning tree algorithms, sample randomized algorithms
Dynamic Programming	Shortest paths in DAGs, longest increasing subsequence, string matching (approximate), integer and (0,1) knapsack problems, chain matrix multiplication, all pair shortest paths, independent sets
Network Flows	Max-flow, bipartite matching
Introduction to Linear Programming	Duality, simplex algorithm
NP-Completeness and reductions	
Coping with NP-Completeness	Approximation algorithms, fixed parameter tractability
Algorithm Sampler (if time allows)	Some more advanced topics of current interest like Page Rank, External Memory Algorithms, Streaming Algorithms, Parallel Algorithms, Distributed Algorithms, and Quantum Computing

Important Dates:

Exam 1	2/25/2019
Exam 2	4/1/2019
Final Exam	TBD

HOMEWORK 1 due date: Wednesday 2/30/19 (in class)

Solve the following exercises from Textbook 1:

- 2.1
 - 2.2
-

Quiz 1 Thursday 2/28/19

Those who want a SPN to enroll in this course must pass Quiz 1 with a grade of 60 or more

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

:

induction.

$$\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$$

$$S = 1 + a + a^2 + \dots + a^n$$

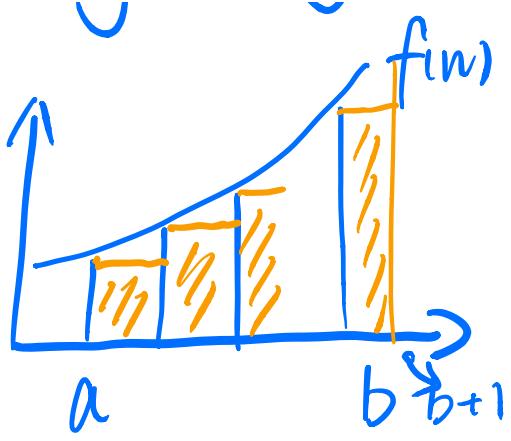
$$aS = a + a^2 + a^3 + \dots + a^{n+1}$$

$$aS - S = a^{n+1} - 1$$

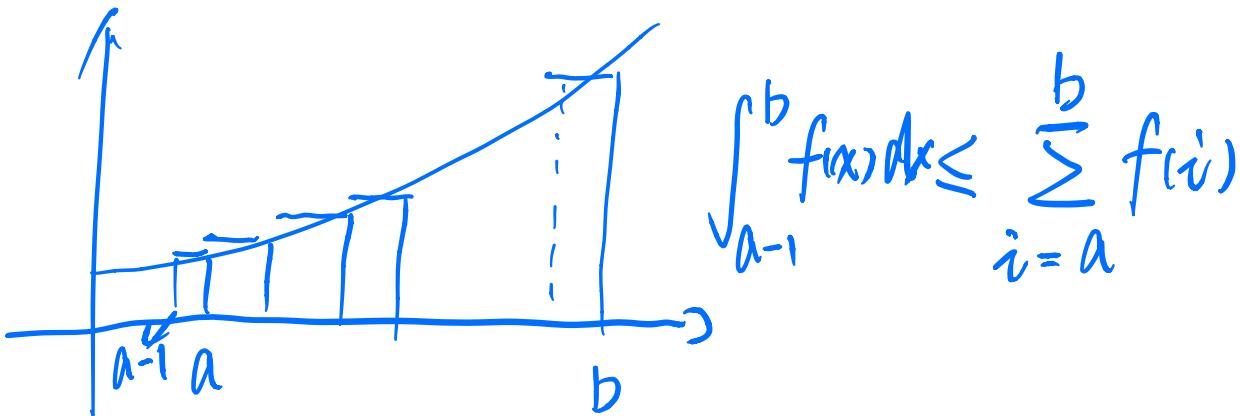
$$S(a-1) = a^{n+1} - 1$$

$$S = \frac{a^{n+1} - 1}{a - 1}$$

Using integrals.



$$\sum_{i=a}^b f(i) \leq \int_a^{b+1} f(x) dx$$



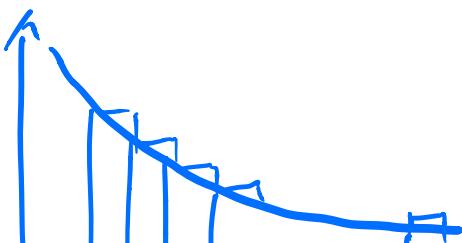
$$\int_{a-1}^b f(x) dx \leq \sum_{i=a}^b f(i)$$

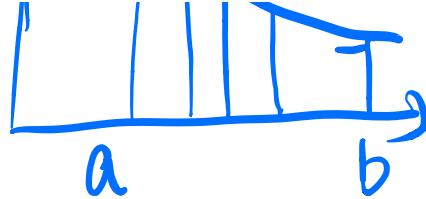
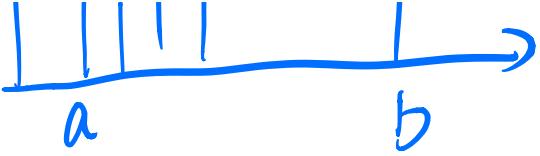
Example. find an upper bound.

$$\text{for } \sum_{i=1}^n \frac{1}{i} \leq \int_1^{n+1} \frac{dx}{x} = \ln x \Big|_1^{n+1} = \ln(n+1) \quad (\text{Harmonic}).$$

$\sum_{i=1}^n \frac{1}{i} \in O(\lg n)$. < careful it's decreasing. $O(\lg n)$.

Decreasing Function.





$$\int_a^{b+1} f(x) dx \leq \sum_{i=a}^b f(i) \leq \int_{a-1}^b f(x) dx$$

$$\sum_{i=1}^n \frac{1}{i} \leq \int_0^n \frac{dx}{x}$$

$$\downarrow = 1 + \sum_{i=2}^n \frac{1}{i} \leq 1 + \int_1^n \frac{dx}{x} = 1 + \ln x \Big|_1^n$$

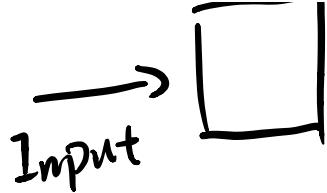
$$= 1 + \ln n \in O(\ln n)$$

Techiques

- Divide and Conquer
 - Greedy Alg.
 - Dynamic Programming..
-

1) divide input into smaller pieces.

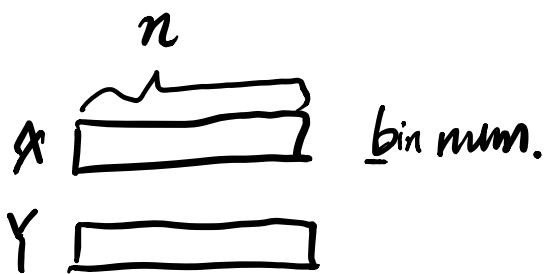
Problem: 2) Recursively solve the smaller pieces
 3) combine into solution



Example.

Binary Operations.

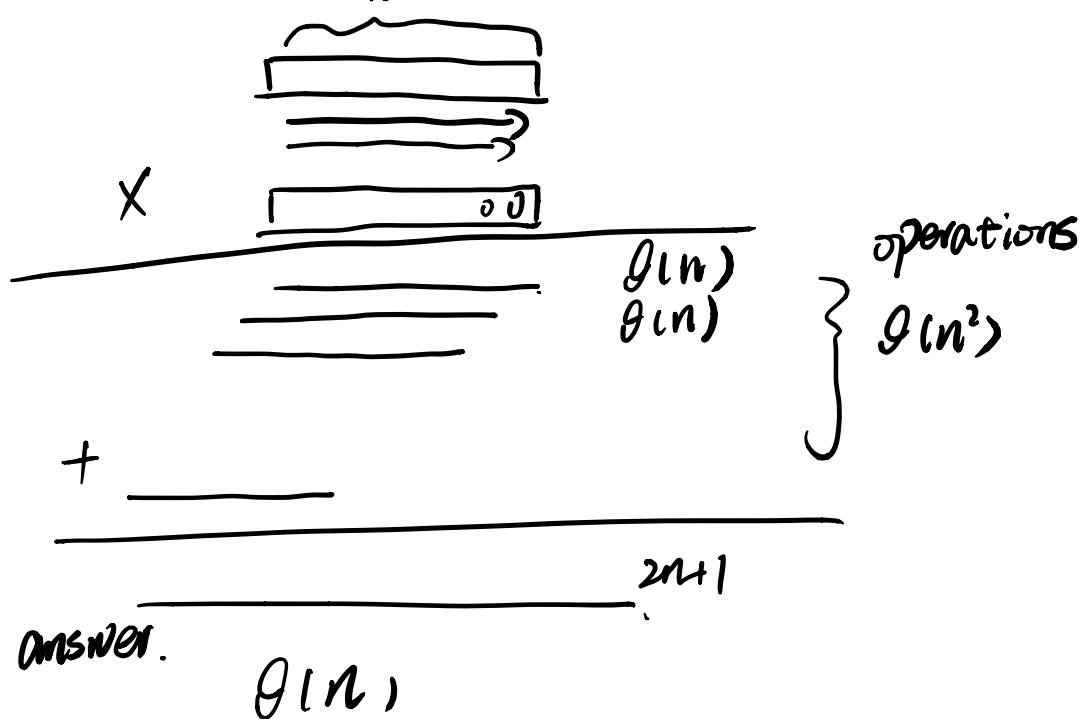
- Addition



$\Theta(n)$. ← Alg. $\Omega(n)$ ← problem.

- multiplication

every Alg. must
write the answer,



$\Theta(n^2)$ known Alg.

gap { $\Omega(n)$ for the problem.

↓ \rightarrow hard to move in this case.

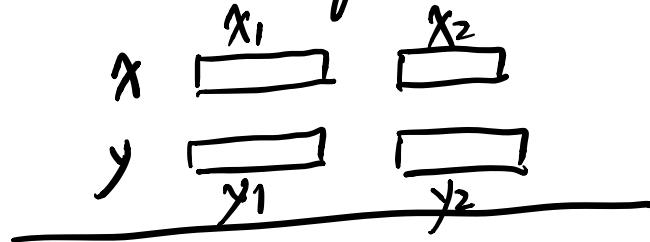
can do better.

improvement.

1) Better Alg. faster.

2) Better lower bound. (proof). faster.

Divide and conquer.



n is power of 2.

⊗ add 0 on left.

$$x = 2^{\frac{n}{2}} x_1 + x_2$$

$$y = 2^{\frac{n}{2}} y_1 + y_2$$

$$x \cdot y = 2^n x_1 y_1 + 2^{\frac{n}{2}} (x_1 y_2 + y_1 x_2) + x_2 y_2$$

↑ shift $\theta(n)$ ↑ $\theta(n)$.

Numbers of operations

$$T(n) = 4 T\left(\frac{n}{2}\right) + \theta(n) \rightarrow \text{recurrence relation.}$$

Iteration:

||

$$4T\left(\frac{n}{2}\right) + Cn.$$

$$\begin{aligned} T(n) &\stackrel{(1)}{=} 4 \left(4T\left(\frac{n}{2^2}\right) + \frac{Cn}{2} \right) + Cn \\ &= 4^2 T\left(\frac{n}{2^2}\right) + \frac{4}{2} Cn + Cn \end{aligned}$$

$$\begin{aligned} (2) \quad &= 4^2 \left(4T\left(\frac{n}{2^3}\right) + C\frac{n}{2^2} \right) + \frac{4}{2} Cn + Cn \\ &= 4^3 T\left(\frac{n}{2^3}\right) + \left(\frac{4}{2}\right)^2 Cn + \left(\frac{4}{2}\right) Cn + Cn \end{aligned}$$

repeat k times:

$$= 4^k T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \left(\frac{4}{2}\right)^i Cn$$

$\frac{n}{2^k} = 1 \Rightarrow k = \lg n$

$C \rightarrow \text{constant.}$

$$= 4^{\lg n} + Cn \sum_{i=0}^{\lg n - 1} 2^i = n^2 \in O(n^2)$$

$$4^{\lg n} = 4^{\frac{\log_4 n}{\log_4 2}} = (4^{\log_4 n})^{\frac{1}{\log_4 2}} = n^{\frac{1}{\log_4 2}} = n^{\log_2 4}$$

$= n^2$

$$(x_1 + x_2)(y_1 + y_2) = x_1 y_1 + x_2 y_1 + x_1 y_2 + x_2 y_2$$

$$xy = 2^n x_1 y_1 + 2^{n/2} (x_2 y_1 + x_1 y_2) + x_2 y_2$$

$$\underline{P_1 = x_1 y_1}$$

$$\underline{P_2 = x_2 y_2}$$

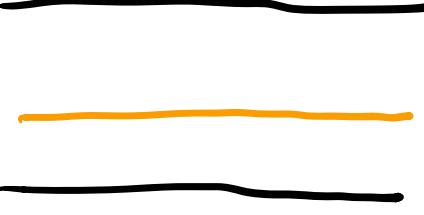
$$\underline{P_3 = (x_1 + x_2)(y_1 + y_2) - P_1 - P_2}$$

$$xy = 2^n P_1 + 2^{n/2} P_3 + P_2$$

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

$$3^{\lg n} + Cn \sum_{i=0}^{\lg n - 1} \left(\frac{3}{2}\right)^i \in \Theta(n^{1.59})$$

$$\Theta(n^{1.59})$$


 $\mathcal{O}(n^2)$
 $\mathcal{O}(n^{1.59})$
 $\Omega(n)$

 fast for human
 fast for computer.

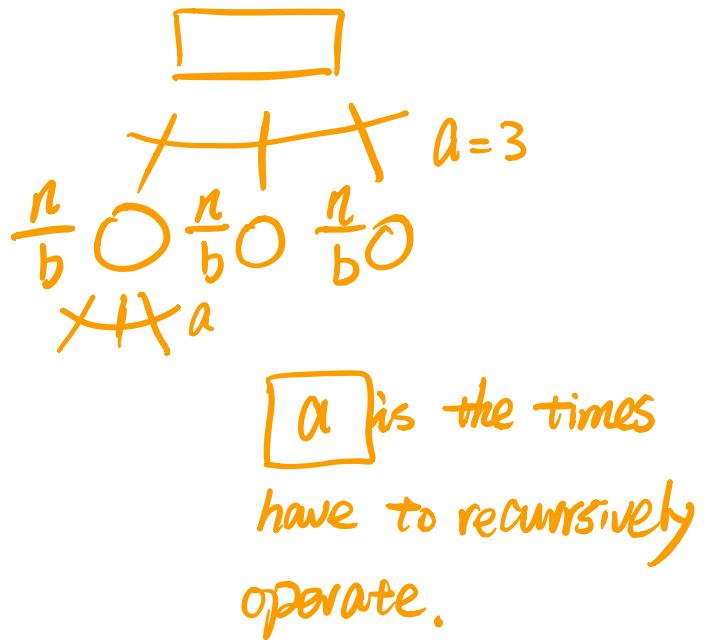
In general:

$$T(n) = \underbrace{\alpha T\left(\frac{n}{b}\right)}_{(3)} + f(n).$$

$\overbrace{\quad\quad\quad}$

Master theorem:

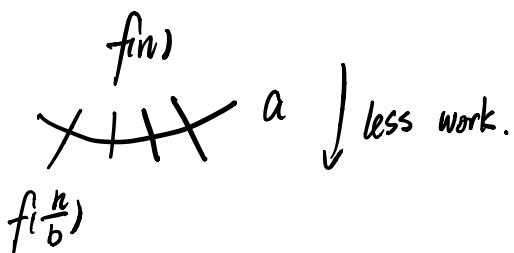
Compare
 $f(n)$ ----- $n^{\log_b a}$.



$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad f(n) > 0 \quad T(m) = 3T\left(\frac{m}{2}\right) + \underline{\Theta(m)}$$

$f(n)$ vs $n^{\log_b a}$ Master Theorem

- upper bound. (1) $f(n) \in O(n^{\log_b a - \varepsilon}) \Rightarrow T(n) \in \Theta(n^{\log_b a})$
- (2) $f(n) \in \Theta(n^{\log_b a}) = T(n) \in \Theta(n^{\log_b a} \lg n)$
- lower bound. (3) $f(n) \in \Omega(n^{\log_b a + \varepsilon}) \Rightarrow T(n) \in \Theta(f(n))$
- and $af\left(\frac{n}{b}\right) \leq Cf(n) \quad 0 < C < 1$
Regularity condition



Proof :

Iteration : $T(1) = 1$ (only affect cost const. anything works).

$$\begin{aligned} \boxed{1} \quad T(n) &= a(aT\left(\frac{n}{b}\right) + f\left(\frac{n}{b}\right)) + f(n) \\ &= a^2 T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n) \end{aligned}$$

$$\begin{aligned} \boxed{2} \quad T(n) &= a^2(aT\left(\frac{n}{b^3}\right) + f\left(\frac{n}{b^2}\right)) + f\left(\frac{n}{b}\right) + f(n) \\ &= a^3 T\left(\frac{n}{b^3}\right) + a^2 f\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n) \end{aligned}$$

$$\boxed{k-1} \quad = a^k T\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)$$

$$n = b^k \Rightarrow k = \lg_b n \quad a^{\log_b n} = n^{\log_b a}.$$

$$T(n) = n^{\log_b a} + \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) \quad \text{Try to find bound for this.}$$

CASE 1 :

$$\begin{aligned} f(n) &\leq C n^{\log_b a - \varepsilon} \\ \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) &\leq C \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon} \\ &= C n^{\log_b a - \varepsilon} \sum_{i=0}^{k-1} \left(\frac{a}{b^{\log_b a - \varepsilon}}\right)^i \\ &= C n^{\log_b a - \varepsilon} \sum_{i=0}^{k-1} (b^{-\varepsilon})^i \end{aligned}$$

$$\begin{aligned} b^{\log_b a - \varepsilon} &= b^{\log_b a} b^{-\varepsilon} \\ &= ab^{-\varepsilon} \end{aligned}$$

$$\begin{aligned}
 &= Cn^{\log_b a - \varepsilon} \frac{\sum_{i=0}^{k-1} b^i \log_b n}{b^\varepsilon - 1} \\
 &= Cn^{\log_b a - \varepsilon} \left(\frac{n^\varepsilon - 1}{b^\varepsilon - 1} \right) \\
 &= \underbrace{\frac{C}{b^\varepsilon - 1}}_{\text{constant}} n^{\log_b a} - \frac{C}{b^\varepsilon - 1} n^{\log_b a - \varepsilon}
 \end{aligned}$$

CASE 2.

$$f(n) \in \Theta(n^{\log_b a})$$

$$\begin{aligned}
 C_1 n^{\log_b a} &\leq f(n) \leq C_2 n^{\log_b a} \\
 \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) &\leq C_2 \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a} \\
 &= C_2 n^{\log_b a} \sum_{i=0}^{k-1} \left(\frac{a}{b^{\log_b a}}\right)^i = C_2 n^{\log_b a} \sum_{i=0}^{k-1} 1 = C_2 n^{\log_b a} \frac{k}{\log_b n}
 \end{aligned}$$

$$d_1 n^{\log_b a} \lg n \leq \sum \leq d_2 n^{\log_b a} \lg n$$

$$T(n) = n^{\log_b a} + \Theta(n^{\log_b a} \lg n) \quad \text{going faster}$$

$$T(n) \in \Theta(n^{\log_b a} \lg n)$$

CASE 3.

$$f(n) \in \Omega(n^{\log_b a + \varepsilon}) \text{ and } af\left(\frac{n}{b}\right) \leq C f(n), \quad 0 < C < 1.$$

$$\text{iterate. } a^i f\left(\frac{n}{b^i}\right) \leq C^i f(n)$$

$$T(n) \in \Theta(f(n))$$

$$\begin{aligned}
 T(n) &\leq \dots f(n) \\
 \sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) &\leq \sum_{i=0}^{k-1} C^i f(n) \\
 &= f(n) \sum_{i=0}^{k-1} C^i
 \end{aligned}$$

$$\begin{aligned}
 S &= 1 + C + C^2 + \dots \\
 CS &= C + C^2 + C^3 + \dots \\
 S - CS &= 1 \quad S = \frac{1}{1-C} \\
 \sum_{i=0}^{\infty} C^i &= \frac{1}{1-C} \quad \checkmark
 \end{aligned}$$

$\dots \sum_{i=0}^{\infty} C_i$ Costant.

$$\leq f(n) \sum_{i=0}^{\infty} C_i = C(f(n)).$$

$$T(n) \leq n^{\log_b a} + C(f(n))$$

$$\therefore T(n) \leq C(f(n))$$

the other side.

$$Cn^{\log_b a} \leq f(n). \quad \text{repeat like CASE 1.}$$

Big-O notation

positive function $f(n)$ $g(n)$
 $c > 0$ $n > 0$

$$n \geq n_0 \quad f(n) \leq c g(n)$$

$$f(n) = O(g(n))$$

$f(n)$ is upper bound by $g(n)$.

L'Hopital Rule

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)} = \begin{cases} \infty & f(n) \neq O(g(n)) \\ & g(n) = O(f(n)) \\ O & f(n) = O(g(n)) \\ & g(n) \neq O(f(n)) \\ C > 0. \text{ Constant.} & \\ \begin{cases} f(n) = O(g(n)) \\ g(n) = O(f(n)) \end{cases} & \\ f(n) = \Theta(g(n)) & \end{cases}$$

Divide & conquer Alg.

$S(n)$
 \downarrow
 $S(n_1), S(n_2), \dots, S(n_k)$ solve recursively

combine solutions.

input $S = \{a_1, a_2, \dots, a_n\}$

output $\min^n\{a_i\}$

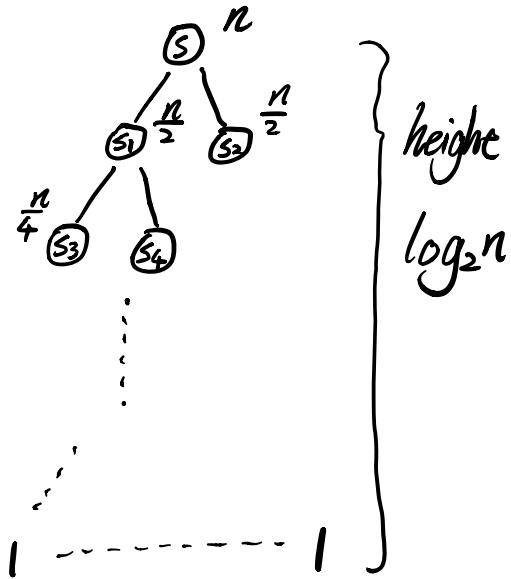
if $n = |S| \leq 1$, return S .

divide S into subsets, S_1, S_2 of equal size

scan S_1 to find $\min(S_1)$

scan S_2 to find $\min(S_2)$

return $\min\{\min(S_1), \min(S_2)\}$.



$$\begin{aligned} T(n) &= T(\frac{n}{2}) + T(\frac{n}{2}) + a \\ &= 2T(\frac{n}{2}) + a = \boxed{?} \end{aligned}$$

Master Theorem

Examples

$$1) T(n) = 3T\left(\frac{n}{2}\right) + n^2$$

$$f(n) = n^2 \quad n^{\log_2 3} = n^{\log_2 3} = n^{1.57...}$$

$f(n) \in \Omega(n^{1.57+\varepsilon})$ [CASE 3]

Regularity Condition

$$af\left(\frac{n}{b}\right) \leq c f(n)$$

$$0 < c < 1$$

$$3\left(\frac{n}{2}\right)^2 \leq cn^2$$

$$\frac{3}{4} \leq c$$

$$c = \frac{3}{4} \quad 0 < c < 1$$

regularity condition is satisfied.

$$T(n) \in \Theta(n^2)$$

$$2) T(n) = T\left(\frac{n}{2}\right) + 2^n$$

$$f(n) = 2^n \quad n^{\log_2 1} = n^0 = 1$$

$f(n) \in \Omega(1)$ [CASE 3]

$$\rightarrow RC: 2^{\frac{n}{2}} \leq c 2^n$$

$$2^{\frac{n}{2}-n} \leq c$$

$$\boxed{\frac{1}{2^{\frac{n}{2}}}} \leq c$$

$$\rightarrow n=1 \quad \frac{1}{2^{\frac{1}{2}}} \leq c$$

largest

$$\frac{1}{\sqrt{2}} \leq c$$

$$c = 0.7071 < 1 \quad RC \text{ satisfied.}$$

$$T(n) \in \Theta(2^n)$$

$$3) T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$$

$$f(n) = n \lg n \quad n^{\log_2 2} = n$$

$$f(n) = n^{\lg n} \in \Omega(n^{1+\varepsilon})$$

have to have a ε satisfy this situation

n^{ε} lower bound for $\lg n \times$

Not case 3 or 1 & 2

cannot use master theorem.

$$4) T(n) = 64T\left(\frac{n}{2}\right) - \underbrace{n^2 \lg n}_{f(n) < 0}$$

$f(n) < 0$ so can not use master theorem.

Sorting

list $A[1], A[2], A[3], \dots, A[n]$ \leftarrow Input

element

Compare $A[i] \stackrel{<}{=} A[j]$

$B[1], B[2], B[3], \dots, B[n]$ \leftarrow permutation output

Silly Sort

for each permutation verify $b[i] \leq b[j] \leq \dots \leq b[n]$
 \uparrow \downarrow if so return b .

Count operations. (comparisons)

$$\hookrightarrow T(n) = n! \Theta(n).$$

Divide & conquer

Divide

Recursively solve
combine.

① Merge sort

Easily

Carefully
(merge)

Quick Sort

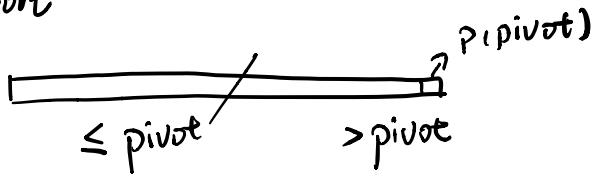
Carefully (partition)

Easily

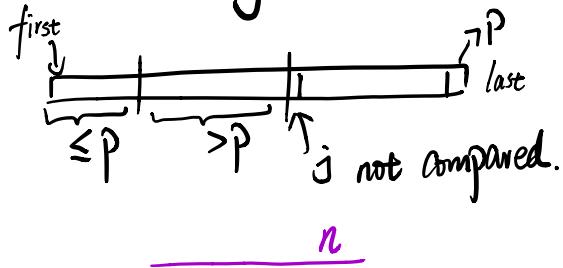


partition

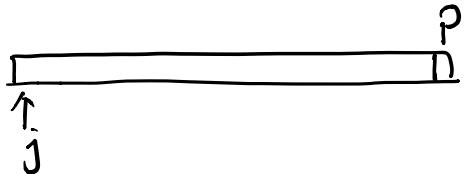
Quick Sort



while executing



initial



partition (a, first, last)

$p = a[\text{last}]$

$i = \text{first} - 1$

for $j = \text{first}$ to $\text{last} - 1$

if ($a[j] \leq p$)

$i++$

swap $a[i]$ with $a[j]$

$\mathcal{O}(n)$

$i++$

swap $a[i]$ with $a[\text{last}]$

return i .

QuickSort (a, first, last)

if $\text{first} < \text{last}$

$q = \text{partition}(a, \text{first}, \text{last})$

QuickSort (a, first, $q - 1$)

QuickSort (a, $q + 1$, last)



proper position

$$\text{Analysis: } T(n) = \Theta(n) + T(q) + T(n-q) \quad 1 \leq q \leq n$$

Comparision based sorting

Best Case: evenly $\Rightarrow n=2^q$

$$T(n) = \Theta(n) + 2T\left(\frac{n}{2}\right)$$

Master Theorem: $f(n) = n \quad n^{\log_2 2} = n$

CASE 2

$$f(n) \in \Theta(n)$$

$$T(n) \in \Theta(n \lg n)$$

Quick Sort

$$T(n) = T(q) + T(n-q) + \theta(n)$$

A horizontal line representing an array of length n . A point labeled p is marked on the line, with an arrow pointing to it from the text "P". Below the line, a bracket indicates the segment from index q to $n-1$.

Best $\Theta(\lg n)$

Worst $\Theta(n^2)$

Worst Case:

A horizontal line representing an array of length n . A point labeled p is marked at the far right end of the line. An arrow points to it from the text "P is the biggest". Below the line, a bracket indicates the segment from index $q=n-1$ to $n-1$.

$$T(n) = T(n-1) + T(0) + \theta(n) = T(n-1) + \theta(n)$$

$$T(n) = T(n-1) + Cn$$

$$\text{Iteration: } T(n) = T(n-2) + C(n-1) + Cn$$

$$= T(n-3) + C(n-2) + C(n-1) + Cn$$

:

$$= T(\underbrace{n-k}_{\substack{n-k=1 \\ k=n-1}}) + [Cn + C(n-1) + C(n-2) + \dots + C(n-k+1)]$$

$$\begin{aligned} n-k &= 1 \\ k &= n-1 \end{aligned}$$

$$= T(1) + C \underbrace{(2+3+4+\dots+n)}$$

$$\frac{n(n+1)}{2} - 1 \in \Theta(n^2)$$

Average Case

$\sum_{\text{All cases}} \text{Comparisons probability}$

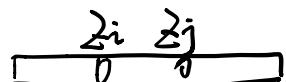
Randomized Partition(a , first, last)

$r = \text{random integer } \in [\text{first}, \text{last}]$

swap $a[r]$ with $a[\text{last}]$

return partition(a , first, last)

Sorted list:



first time the algorithm picks a pivot in the interval.

$j-i+1$

CASE 1: pivot is not $\in \{z_i, z_j\}$.

0 Comparisons

CASE 2: pivot is z_i

1 Comparison

CASE 3: pivot is z_j

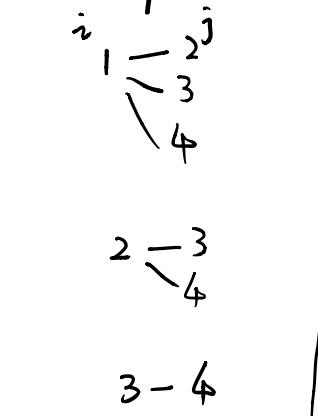
1 Comparison

give z_i and z_j .

$$\text{expected (average)} = 0 + 1 \cdot \frac{1}{j-i+1} + 1 \cdot \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

add overall possible pairs

Example $n=4$

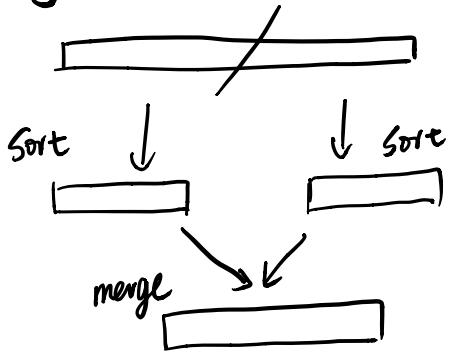


$$\begin{aligned}
 & \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \quad \text{if } j-i=k \\
 &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} \\
 &\leq 2 \sum_{i=1}^{n-1} \lg n = 2(n-1) \lg n \in O(n \lg n)
 \end{aligned}$$

Simplify to compute.

Average Case.

Merge Sort :



Merge(a, first, mid, last)

$i = \text{first}$

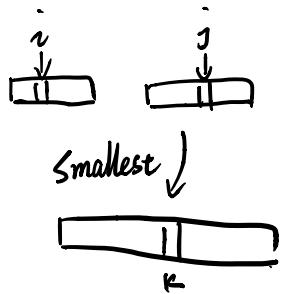
$j = \text{mid} + 1$

for $k = \text{first}$ to last

if ($i \leq \text{mid}$ and $j \leq \text{last}$)

if $a[i] < a[j]$

first mid last



MergeSort(a, first, last)

if first < last

mid = $\lfloor (first + last) / 2 \rfloor$

MergeSort(a, first, mid)

MergeSort(a, mid+1, last)

Merge(a, first, mid, last)

b[k] = a[i]

i++

else

b[k] = a[j]

j++

else if (i > mid)

b[k] = a[j]

j++

else

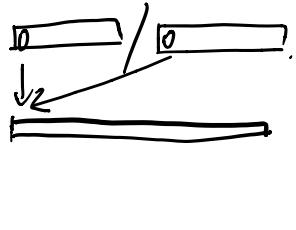
b[k] = a[i]

i++

copy b to a

(from first to last).

Merge Sort



Worst Case = n $LRRLRLLR$
 best case = $n/2$ Left < Right
 $\theta(n)$

$$T(n) = 2T(\frac{n}{2}) + \theta(n)$$

$$n^{\log_2^2} = n \quad n$$

$$T(n) \in \Theta(n \lg n).$$

upper bound.
 $O(n \lg n)$

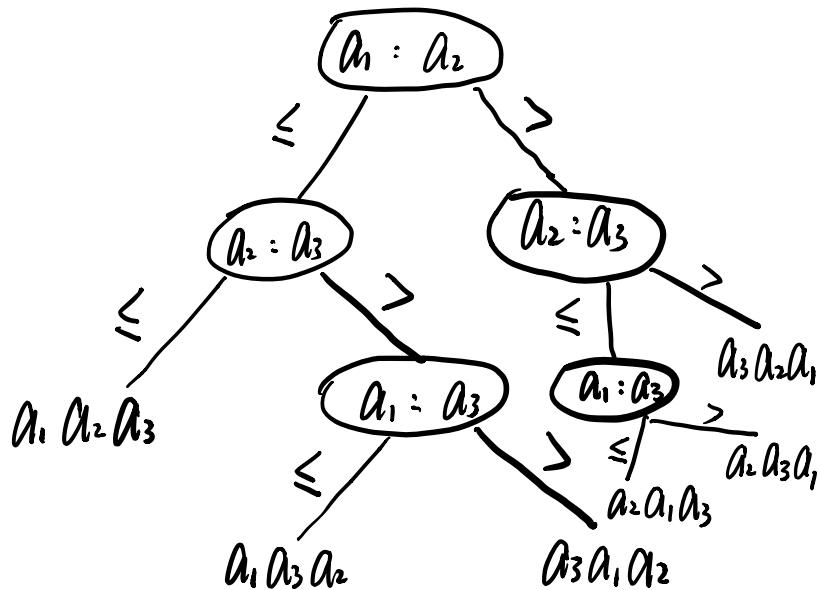
lower bound
 $\Omega(n \lg n)$

Comparison Decision Tree

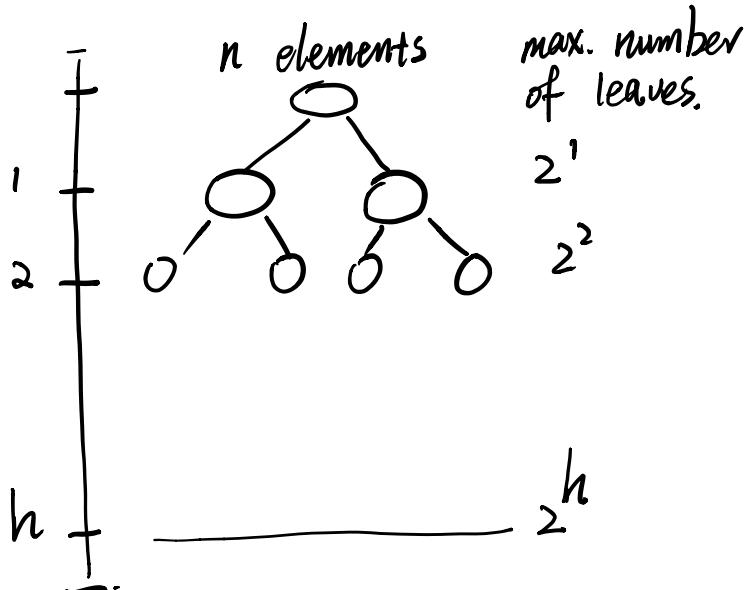
Example : a_1, a_2, a_3

any comparison-based sorting algorithm must do at least n comparisons.

each permutation is a leaf.



$$\text{permutation} = 3! = 6$$



$$T(n) \geq h \geq \lg l$$

$$l = n!$$

$$T(n) \geq \lg n!$$

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \leftarrow \text{Stirling Approx}$$

if l is the number of leaves

$$l \leq 2^h$$

$$\lg l \leq h$$

$$\lg n! \simeq \lg \sqrt{2\pi n} + n \lg \left(\frac{n}{e}\right)$$

$$= \lg \sqrt{2\pi} + \frac{1}{2} \lg n + n \lg n - n \lg e$$

$$= \lg \sqrt{2\pi} + \frac{1}{2} \lg n + n \lg n - n \lg e$$

$$\bar{T}(n) \in \Omega(n \lg n)$$

Any Sorting algorithm $\in \Omega(n \lg n)$
comparison based.

↑
Worst Case

Worst Case - mergesort

$\Omega(n \lg n) \cdot O(n \lg n) \leq \text{mergesort}$ is optimal.

Selection Problem

find the k -th smallest element

$$k=1 \quad \min \quad O(n)$$

$$k=n \quad \max \quad O(n)$$

$$k=\frac{n}{2} \quad \text{median}$$

↓
Sort a
return $a[k]$ $\mathcal{O}(n \lg n)$.

lower bound

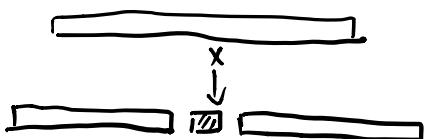
must compare all elements at least once: $\Omega(n)$.

$$\underline{\text{gap}} \quad \overbrace{11111}^{\mathcal{O}(n \lg n)} \quad \underline{\Omega(n)}$$

Divide & Conquer

$$(a_1, a_2, \dots, a_n)$$

$$K.$$



select(a , first, last, k)

$x = \text{partition}(a, \text{first}, \text{last})$

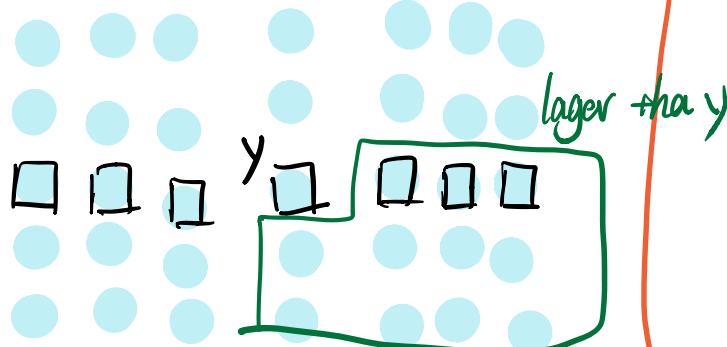
if $k=x$ return $a[x]$

else if $k < x$ return select(a , first, $x-1$, k)

else return select(a, x+1, last, k)

- use randomized partition
- average case $\theta(n)$
- worst case $\theta(n^2)$

Finding a good pivot
group of 5



~~0 | 0 | 0 |~~

find median

$$\text{Small} \geq \frac{1}{2} \cdot \frac{n}{5} \cdot 3$$

$$\text{Large} \leq \frac{7}{10}n$$

why 5? not 3?

- find the median of each group of 5 (constant time).
- find the median of the medians (recursively)
- partition using $a[y]$ as pivot.

\downarrow

$$T(n) = \theta(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right)$$

$$1.(a) \begin{cases} T(m, n) = T(k, \frac{n}{2}) + T(m-k, \frac{n}{2}) + a \cdot m & , n > 1 \\ T(m, n) = b \cdot m & , n \leq 1 \end{cases}$$

$T(m, n) = O(m \log n)$ *guess & verify*

$$\Rightarrow O(m \log n) \leq C \cdot m \log n + a \cdot m$$

$$\Rightarrow C \cdot k \log \frac{n}{2} + C(m-k) \log \frac{n}{2} \leq C \cdot m \log n$$

$$\Rightarrow Cm(\log n - \log_2 1) + a \cdot m \leq C \cdot m \log n$$

$$\Rightarrow Cm \log n - Cm + am \leq Cm \log n$$

$$\Rightarrow -Cm + am \leq 0$$

$$\Rightarrow a \leq C$$

$$(b) a_0 = 1 \quad a_1 = 2 \quad n > 1 \quad a_n = 6a_{n-2} - a_{n-1}$$

$$\Rightarrow a_n x^n = 6a_{n-2} x^n - a_{n-1} x^n$$

$$\sum_{n=2}^{\infty} a_n = \sum_{n=2}^{\infty} 6a_{n-2} x^n - \sum_{n=2}^{\infty} a_{n-1} x^n \quad \begin{cases} a_0 = 1 \\ a_1 = 2 \end{cases}$$

$$G(x) = \sum_{n=0}^{\infty} a_n x^n$$

$$G(x) = \sum_{n=2}^{\infty} a_n x^n + a_1 x + a_0$$

$$\Rightarrow G(x) - 2x - a_0 = 6x^2 \cdot G(x) - xG(x) + a_0 x$$

$$G(x) = \frac{3x+1}{-6x^2+x+1}$$

$$= \frac{1}{-2x+1}$$

$$= \sum_{n=0}^{\infty} a_n x^n \quad a_n = 2^n$$

2. $O((\log n)^3)$



if > 2 = recursively

else : terminate.

$$\begin{cases} T(n) = 3T\left(\frac{n}{3}\right) + \alpha(\log n)^3 & n > 2 \\ T(n) = b \text{ (constant)} & n \leq 2 \end{cases}$$

Master Theorem

CASE ONE.

3. First, find the middle row j of matrix

Linearly scan row j to find leftmost minimum element, x , and corresponding index, k , the leftmost min element for all rows having index less than j can only be exist at position $< k$.

Hence, we can recursively call the top-left part of $M[1 \dots (j-1), \dots k]$
 $M[(j+1) \dots n, k \dots m]$

Pseudo code

find Left Most Min Val (M)

if $n == 1$:

linear search for min value x .

return x

else :

$$j = \frac{n}{2}$$

linear search for row j of M , which is $x(j, k)$.

$$M_1 = M[1 \dots (j-1), 1 \dots k]$$

$$M_2 = M[(j+1) \dots n, k \dots m]$$

$I_1 = \text{find Left Most Min Val } (M_1)$

$I_2 = \text{find Left Most Min Val } (M_2)$

return x, I_1, I_2

$$2\overline{1}(\frac{n}{2})+n\;.$$

$$\mathcal{O}(m \log n)$$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + \alpha n, \alpha > 0$$

hope it is linear time. $O(n)$

definition $T(n) \leq Cn, C > 0$

$$\Rightarrow T\left(\frac{n}{5}\right) \leq C\frac{n}{5}$$

$$T\left(\frac{7}{10}n\right) \leq \frac{7}{10}Cn$$

$$\underbrace{\frac{Cn}{5} + \frac{7}{10}Cn}_{\text{need } \uparrow \uparrow} + \alpha n \leq Cn$$

need $\uparrow \uparrow$
to show.

$$2Cn + 7Cn + 10\alpha n \leq 10Cn$$

$$10\alpha \leq C$$

$$\therefore C \geq 10\alpha > 0$$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + (\alpha n)$$

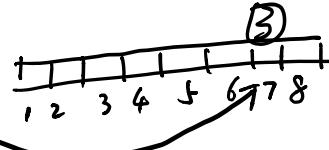
$\underbrace{T(n) \in O(n)}$ already linear time

$$T(n) \in O(n).$$

Sorting without comparison of elements.

Array = 2 5 3 1 2 3 1 3

Counter :	1	11	(2) - 2
	2	11	(2) - 4
	3	111	(3) - 7
	4		(0) - 7
	5	1	(1) - 8



Counting sort.

for $i=1$ to k

$$C[i] = 0$$

K

$$T(n) \in O(n+k).$$

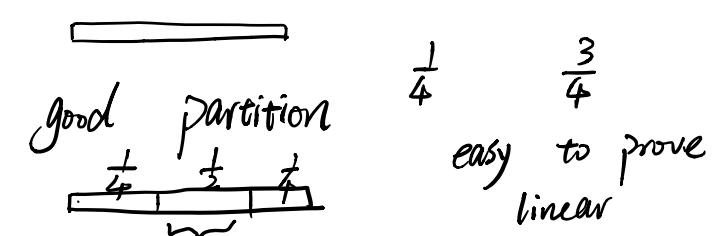
for $j=1$ to n

$$C[a[j]]++$$

n

for $i=2$ to k

sorts in linear time



probability of a good partition is $\frac{1}{2}$. $\rightarrow E = 1 + \frac{1}{2}E$

times to have a good result of flip coin. flip once coin

$$C[i] = C[i] + C[i-1] \quad k-1$$

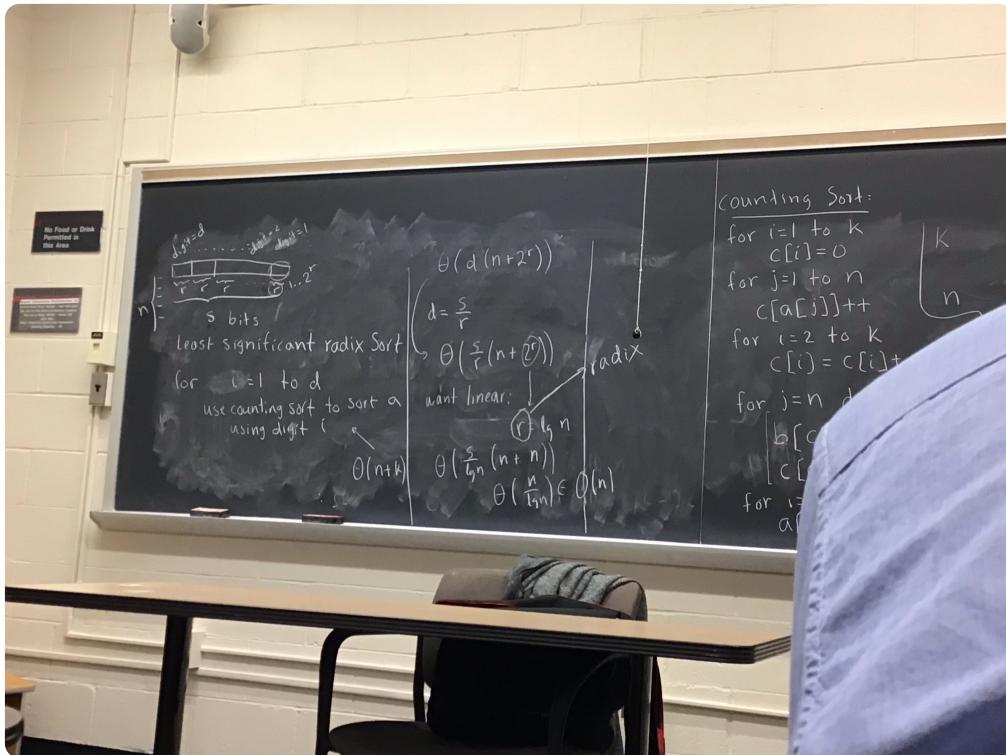
for $j=n$ down to 1

$$b[c[a[j]]] = a[j] \quad n$$

$$c[a[j]] --$$

for $i=1$ to n

$$a[i] = b[i]$$



Radix sort

Divide & Conquer

Matrix Multiplication

$$X^{(n \times n)} Y^{(n \times n)} = \left(\square \xrightarrow{\text{def}} \right)$$



need to compute n^2
 \sin^3)

lower bound:

At least write the produce

$\longrightarrow O(n^3)$

$\Omega(n^2)$

$$X = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

where $A \sim H$ are matrices

$$XY = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) \rightarrow T(n) \in \Theta(n^3)$$

Strassen's Algorithm.

$$P_1 = A(\bar{F} - H)$$

$$P_2 = (A+B)H$$

$$P_3 = (C+D)\bar{E}$$

$$P_4 = D(G - \bar{E})$$

$$P_5 = (A+D)(E+H)$$

$$XY = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$$

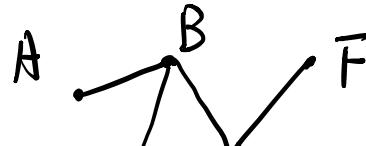
$$\rightarrow T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$T(n) \in \Theta(n^{2.81})$$

Graph Algorithm.

Graph = $G = (V, E)$

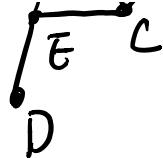
undirected graph



$$V = \{A, B, C, D, E, F\}$$

V = set of vertices

$E \subseteq V \times V$. set of edges.



$$E = \{(A,B), (B,C), (C,E), (E,D), (B,E), (C,F)\}$$

directed graph

Some definitions.

• Adjacent vertices

v is adjacent to $u \Leftrightarrow (u,v) \in E$

• u and v are neighbors $\Leftrightarrow (u,v) \in E$

• (undirected) degree : number of adjacent vertices.

• (directed) in-degree:

out-degree =

• $n = |V|$, $m = |E|$

• Complete graph: all possible edges
(undirected) $m = \frac{n(n-1)}{2}$

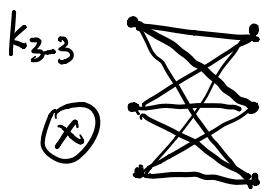
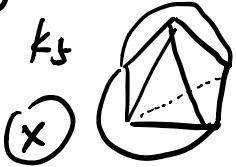
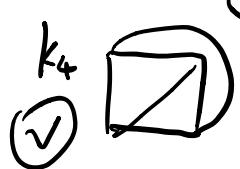
K_n — complete graph with n vertices

K_3

K_4

K_5

Embedding on a plane (without crossing edges)



↔ planarity problem.

Dense graphs : $m \in \Theta(n^2)$

Sparse graphs = $m \in \Theta(n)$

Simple graphs = not allowed :



(in this course)

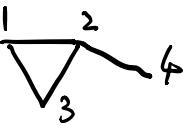
(multigraphs).

Data Structures.

Simplest : list of edges \times searching $\Theta(n^2)$

Adjacency Matrix :

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$



$$M_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{if } (i, j) \notin E \end{cases}$$

symmetric: undirected graphs.

How long to determine
if $(i, j) \in E$? $\Theta(1)$

list vertices adjacent to i : $\Theta(n)$.

Adjacency list :

(undirected) $1 - 2, 3$

how long to determine if $(i, j) \in E$?

$\Theta(n)$

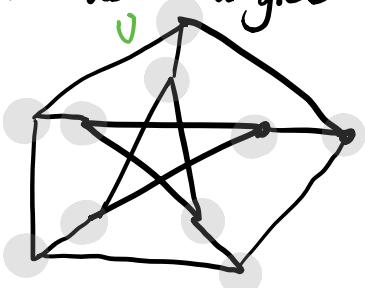
every edge $2 - 1, 3, 4$

appears $3 - 1, 2$

2 times. $4 - 2$

In a directed graph, each edge appears once. $\Theta(\max \underline{\text{adjacent vertices}})$, degree

Bounded degree = 3



Petersen Graph

Explore a graph.

Explore (v)

```

    | v. marked = True
    | for each  $(v, u) \in E$ 
    |   | if not u. marked
    |   |   | Explore(u)
  
```

Connected Graph.

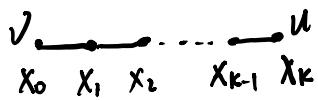
$\forall u, v \in V$, there is a path from u to v .

path from u to v

$$u = x_0, x_1, x_2 \dots x_r = v$$

Since $(x_{i-1}, x_i) \in E$ Length (number of edges) : k .

$$\forall i = 1 \dots k$$

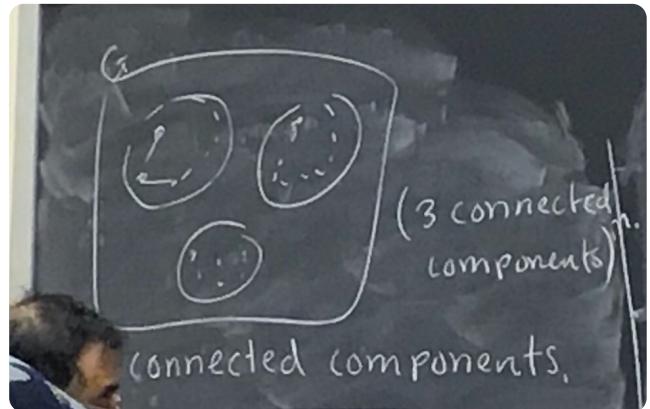


Algorithm to determine if a graph (undirected) is connected:

Connected(G)

```

    | Let  $v \in V$ 
    | Explore(v)
    | for each  $u \in V$ 
    |   | if  $u$ . marked = false
    |   |   | return false
    | return true
  
```



Algorithm to determine the number

of connected components.

Connected Components.

$\text{Explore}(v)$

$v.\text{marked} = \text{true}$

$\text{previsit}(v)$

for each (v, u)

if not $u.\text{marked}$

$\text{Explore}(u)$

$\text{postvisit}(v)$

Connected Component (G)

$\text{comp} = 0$

for each $v \in V$

if not $v.\text{marked}$

$\text{comp}++$

$\text{Explore}(v)$

return comp

→ Connected Graph: $\forall u, v \in V, \exists$ a path from u to v

Relation. u and v are related if \exists a path from u to v

Path relation is an equivalence relation.

⇒ partitions V into connected components.

Time complexity: $O(|V| + |E|)$ ← using adjacency list

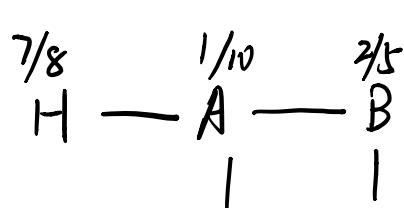
$O(n+m)$	sparse $O(n)$	adj. matrix $O(n^2)$
	dense $O(n^2)$	

Depth - First Search (DFS) Adj list processed in lexicographical order.

$\text{previsit}(v)$

$\text{Count}++$

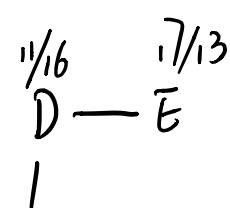
$v.\text{pre} = \text{Count}$



$\text{postvisit}(v)$

$\text{Count}++$

$v.\text{post} = \text{Count}$



$\text{DFS}(G)$

$\text{Count} = 0$

for each $v \in V$

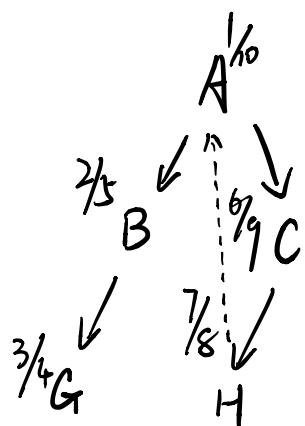
if not $v.\text{marked}()$

$\text{Explore}(v)$

$C_{6/9}$

$G_{3/4}$

$F_{14/15}$

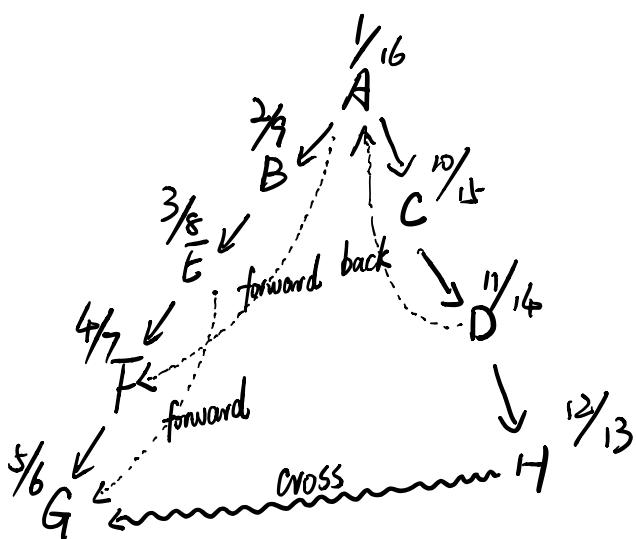
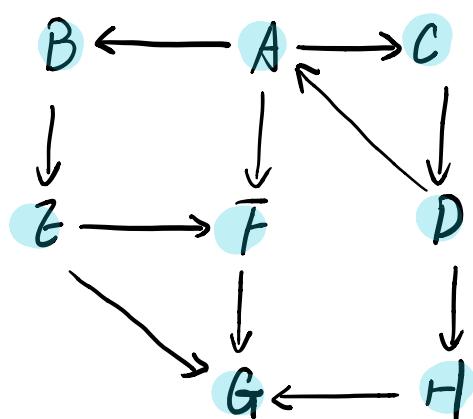


Tree edge (\downarrow) Cross edge (\rightarrow)

Back edge (\uparrow) / forward edge (\downarrow)

G_f has a cycle \Leftrightarrow DFS finds a back edge

path $v = x_0, x_1, \dots, x_n$



$u \rightarrow v$

$u.\text{pre} < v.\text{pre} < v.\text{post} < u.\text{post}$

$[$	u	$]$	v	$]$	v	$]$	u	$]$	tree/forward	$[[]]$	$u v v u$
u									back	$[[]]$	$v u u v$
									cross	$[] []$	$v v u u$

G_f has a cycle \Leftrightarrow DFS finds a back edge

Proof: G_f has a cycle (not repeating vertices)

(\Rightarrow): $x_0, x_1, \dots, x_j, x_{j+1}, \dots, x_k, \dots, x_0$

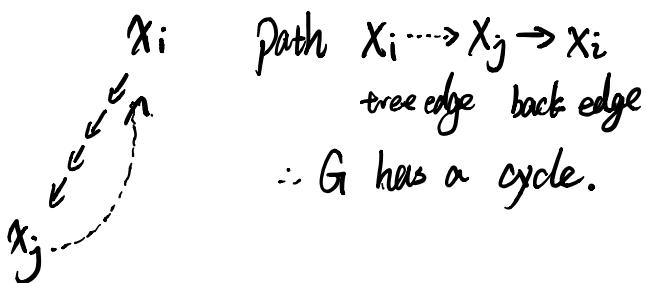
x_0 is first vertex in C seen by DFS.

\Rightarrow every vertex in the cycle is reachable from x_0

\Rightarrow down in the recursion.

$X_{k-1} \Rightarrow \text{edge}(X_{k-1}, X_0)$ is seen as a back edge.

(\Leftarrow) : If a back edge



If back edge \Rightarrow cycles

If no cycles \Rightarrow no back edges

Directed Acyclic Graph (DAG) \Leftarrow no back edge.

$\forall u \rightarrow v \quad v.\text{post} < u.\text{post}$

If a graph represents tasks (precedence)

we can find a linear order of the task

satisfying the precedence.



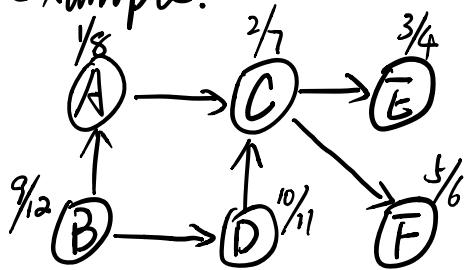
Topological Sorting

(Linearization).



Sort tasks in reverse post-visit number order.

Example.



linearization: B D A C F E
(post) 12 11 8 7 6 4

DAG $\xrightarrow{u} \xrightarrow{v}$

post $u >$ post v

DS:

Priority Queues

Sorted list

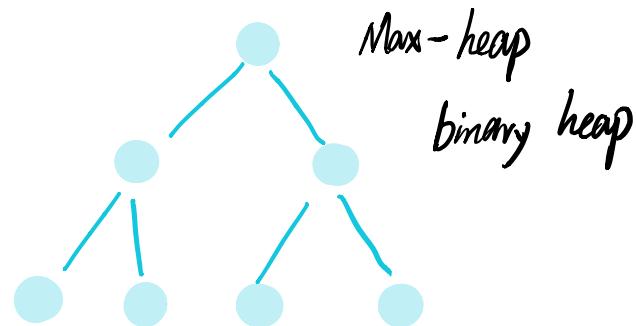
find position $\Theta(\lg n)$

insert $\Theta(n)$

Basic operations.

- Extract Max (min)
Not the same time.
- insert (x)
- build Queue (array a)
- increase key (i)
- decrease key (i)

Tree structure. (Max)



conditions for heap:-

- complete binary heap except last level
complete from left to right

Heap property =

value of a node is \geq the value of its children

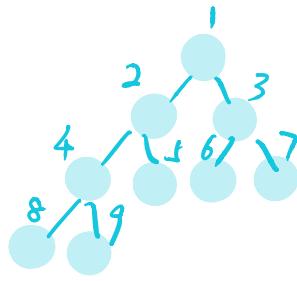
root has the max value.

heap: use an array

$$\text{left}(i) = 2i$$

$$\text{right}(i) = 2i+1$$

$$\text{parent}(i) = \lfloor i/2 \rfloor$$



Extract Max()

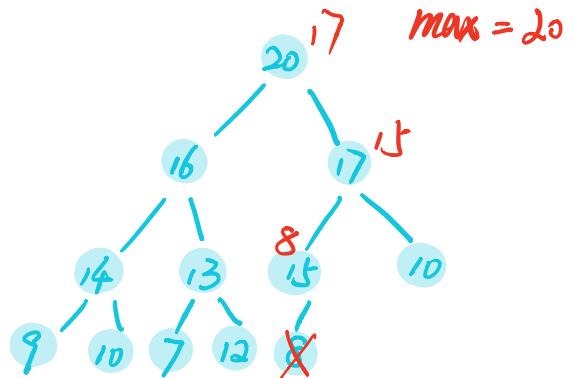
$$\max = a[1]$$

$$a[1] = a[n]$$

$$n-- \rightarrow O(\lg n)$$

Sift Down(i)

return max



Sift Down(i)

if ($2i \leq n$)

let $j = \max$ index of ($a[i], a[2i], a[2i+1]$)

if ($i \neq j$)

swap $a[i]$ with $a[j]$

Sift Down(j)

might not exist.

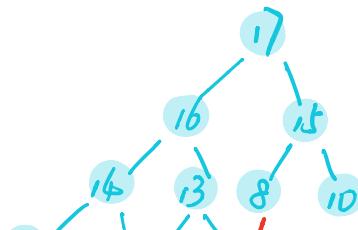
max levels of heap: $\lg n$

$O(\lg n)$ worst case.

insert(x)

$n++$

$$a[n] = x$$



Bubble Up (n) $\rightarrow O(lgn)$

9' 10' 7' 12' 19'

Bubble Up (i)

Bubble Up (i)

if $i > 1$

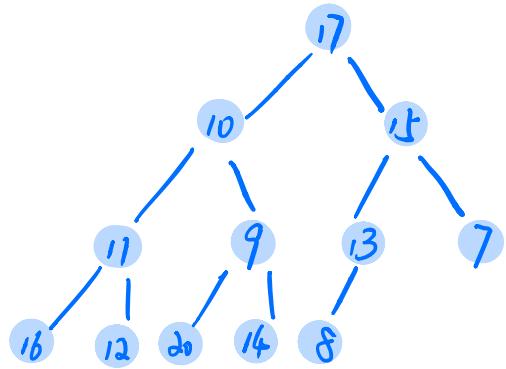
Let $j = \max$ index of $(a[i], a[\text{parent}(i)])$

if $|i=j|$

swap $a[i]$ with $a[\text{parent}(i)]$

Bubble Up ($\text{parent}(i)$)

17 10 15 11 9 13 7 16 12 20 14 8



Build Queue (a).

for $i = \lfloor \frac{n}{2} \rfloor$ down to 1

Sift Down (i)

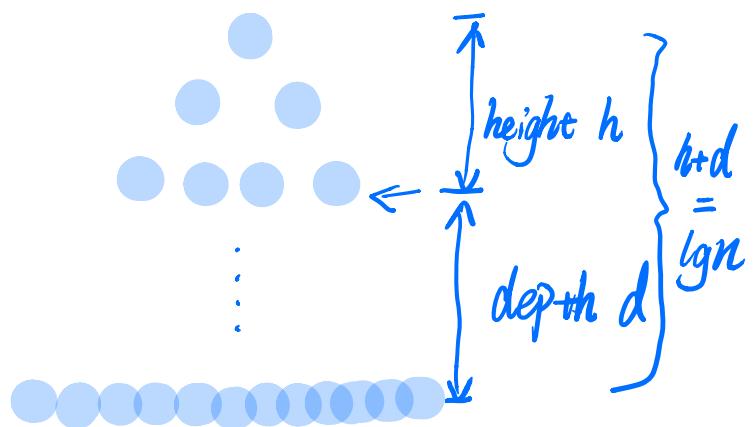
Analysis.

Priority Queue

`buildQueue (array a)`

for $i = \lfloor \frac{n}{2} \rfloor$ down to 1

`siftDown (i)`



$$\text{Worst case: } W(n) = \sum_{d=0}^{\log n - 1} n \frac{d}{2^d} = n \sum_{d=0}^{\log n - 1} \frac{d}{2^d}$$

we don't compute last level

$$\sum_{d=0}^{\infty} \frac{d}{2^d} = 0 + \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \dots = S$$

$$\frac{1}{2}S = 0 + \frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \dots$$

$$S - \frac{1}{2}S = \frac{1}{2}S = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots = 1$$

$$S = 2$$

$$W(n) \leq 2n \Rightarrow W(n) \in \Theta(n).$$

* number of nodes 2^h (height h).

* number of operations for a node at depth $d = d$

* number of nodes at depth of d

$$2^h = 2^{\lg n - d} = 2^{\lg n} 2^{-d}$$

$$= n \frac{1}{2^d}$$

* total operations at depth d

= nodes at depth d · operations for each nodes = $n \cdot \frac{1}{2^d} \cdot d = \frac{nd}{2^d}$

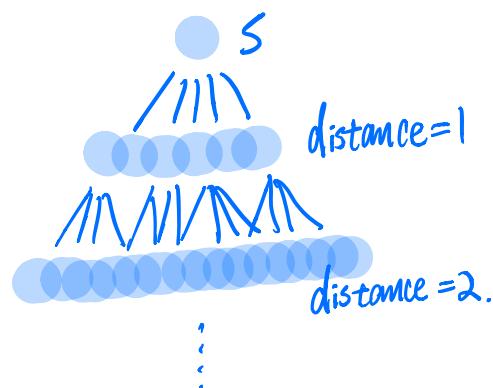
Shortest Paths. (undirected graphs)

Correctness (There is a point where a vertex v is at distance d

\uparrow
 v is in the queue).

$v.dist = \text{distance from } s \text{ to } v$

$v.pred = \text{keeps predecessor on the tree}$



`BFS (graph G)`

for each $v \in V$

$v.dist = \infty$

$\{O(1/v)\}$

Queue:

- `isEmpty()` Constant time
- `enqueue(x)`

$v.\text{pred} = \text{null}$

$S.\text{dist} = 0$

$Q.\text{enqueue}(S)$

while not $Q.\text{isEmpty}()$

$v = Q.\text{dequeue}()$

for each $(v, u) \in E$

if $u.\text{dist} = \infty$

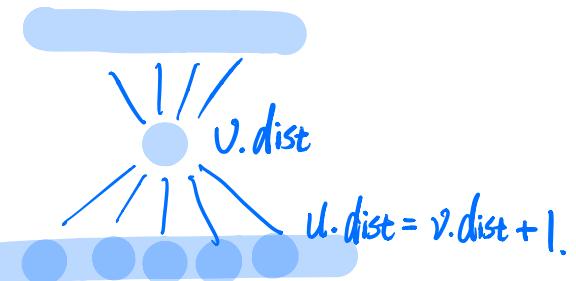
$u.\text{dist} = v.\text{dist} + 1$

$u.\text{pred} = v$

$Q.\text{enqueue}(u)$

• $\text{dequeue}()$

$O(|E|)$



↑ Analysis.

Worst case: $O(|V| + |E|)$

Weighted graph.

$G = (V, E), w: E \rightarrow R^+$

Assume a shortest path from s to v

$s \xrightarrow{\dots} x \xrightarrow{\dots} v$

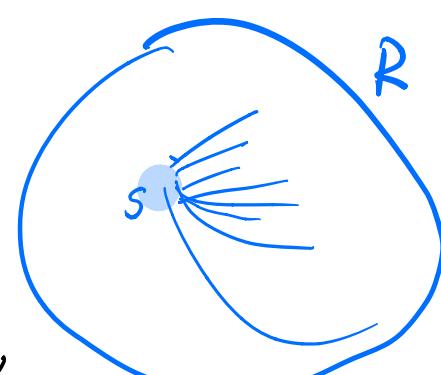
$\text{dis}(x) < \text{dis}(x) + w(x, v) = \text{dis}(v)$ $d(s, v)$: distance

$\Rightarrow \text{dis}(x) < \text{dis}(v)$ (shortest path) from
s to v.

\swarrow x must be in R.

\searrow otherwise x is closer from s than v
contradicts p.

satisfy



at point R is correctly computed
all shortest paths from s $\leq d$

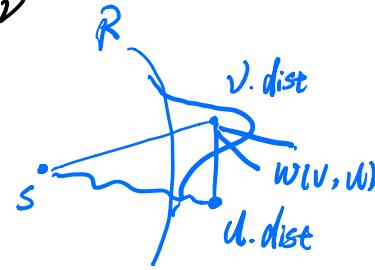
find v that we can add to R.

\nwarrow
closest vertex from s not in R.

v must be a one-edge extension of vertices in R.

claim that v is the one edge extension with minimum length.

otherwise $\exists x$ that is closer from s than v



Dijkstra l. Gr. S)

for each $v \in V$

$v.dist = \infty$

$v.pred = \text{null}$

$s.dist = 0$

$Q.\text{buildQueue}(V)$

while not $Q.\text{isEmpty}()$

$v = Q.\text{extractMin}()$

for each $(v, u) \in E$

if $v.dist + w(v, u) < u.dist$

$u.dist = v.dist + w(v, u)$

$u.pred = v$

$Q.\text{decreaseKey}(u) \leftarrow \text{bubble up } g(|V|)$

Dijkstras Alg.

for each $v \in V$

$$\begin{cases} v.dist = \infty \\ v.prev = \text{null} \end{cases}$$

$|V|$

$s.dist = 0$

$Q.\text{buildQueue}(v)$

$|V|$

while not $Q.\text{isEmpty}()$

$|V| \lg |V|$

$v = Q.\text{extractMin}()$

for each $(v, w) \in E$

if $v.dist + w(v, w) < u.dist$

$|E| \lg |V|$

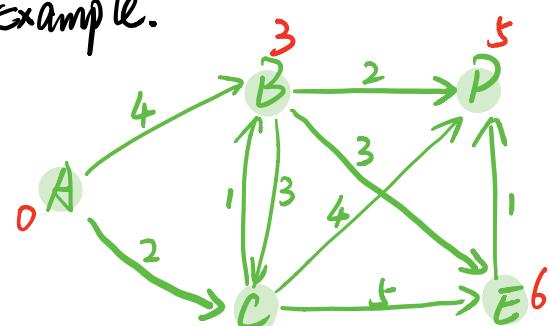
$u.dist = v.dist + w(v, w)$

$u.prev = v$

$Q.\text{decreaseKey}(w)$

$O((|V|+|E|) \lg |V|)$

Example.



starting from A.
 $S = A$
 dist/pre

	A	B	C	D	E
0/	∞ /				
A	$4/A$	$2/A$	∞ /	∞ /	∞ /
C	$3/C$		$6/C$	$7/C$	
B			$5/B$	$6/B$	

A
C
B
D
E

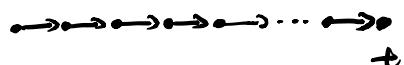
$6/B$ no out-way from
D. copy others.

$\text{update}(v, u)$

if $v.dist + w(v, u) < u.dist$

$u.dist = v.dist + w(v, u)$

shortest path



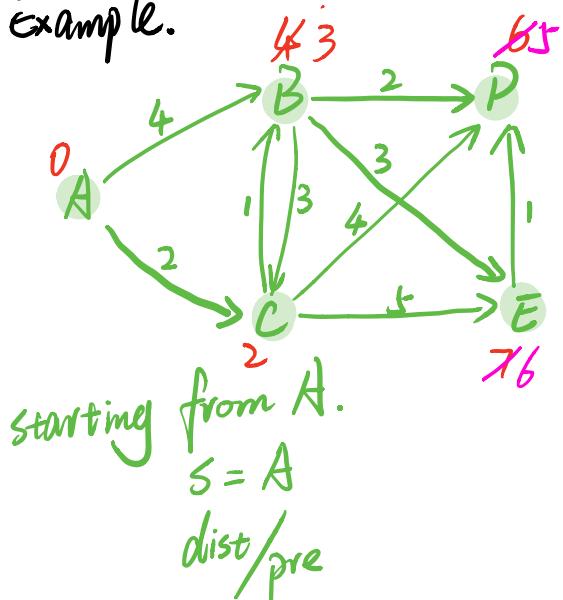
s member of edges: $|V|-1$

Bellman - Ford:

repeat $|V|-1$ times:

for each $(v, u) \in E$
update (v, u)

Example.



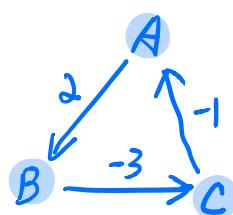
update.

	1 time	2 time	---
AB	$B \rightarrow 4$	$B -$	
AC	$C \rightarrow 2$	$C -$	
BD	$D \rightarrow 6$	$D \rightarrow 5$	
CD	$D -$	$D -$	
CE	$E \rightarrow 7$	$E -$	
BC	$C -$	$C -$	
CB	$B \rightarrow 3$	$B -$	
ED	$D -$	$D -$	
BE	$E -$	$E \rightarrow 6$	

spacial situation :

solution: check on more

time ($|V|$ times) and if



value change, then there are negative cycle ↑.

Directed Graph.

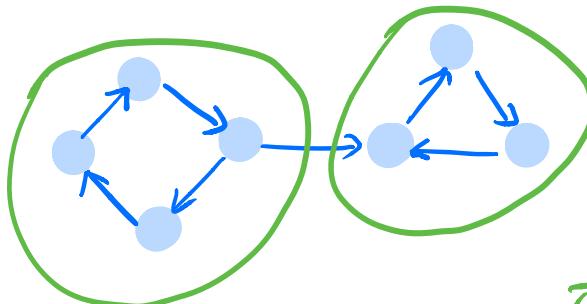
$\forall u, v \in V$. there is a path from u to v and there is a path from v to u .

Strongly connected \uparrow

u, v are related. if f^T

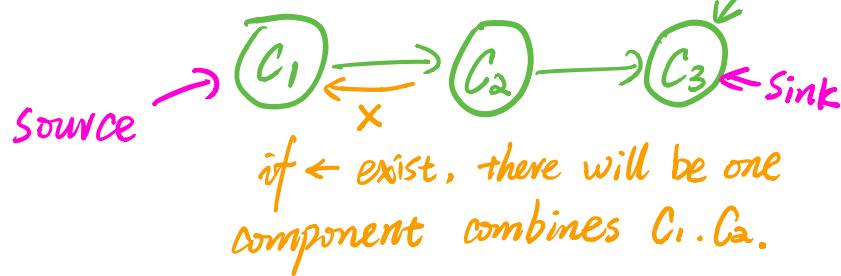
It satisfies : Equivalence Relation.

Partitions the set of vertices into equivalence classes



strongly connected
components (SCC)

C_4



find the SCCs of a G

Count = 0

repeat.

 | find a vertex v

 | in a sink SCC of G

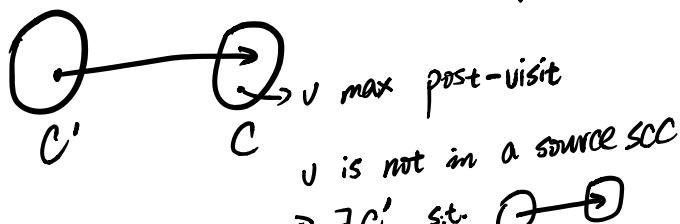
 | Count ++

 | explore (v)

 | $G_r = G - \text{marked vertices}$

until V is empty

We need to show that the vertex with maximum post-visit number is in a source SCC. (below is proof. assuming it's true).



CASES:

(1) vertex in C is visited first

$\Rightarrow v$ visited first

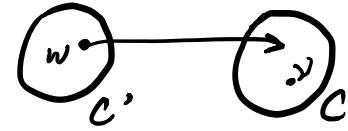
post-visit numbers in

$C' >$ post-visit number of v

$\Rightarrow v$ post was not the max

(2) visit a vertex in C' before C

\Rightarrow visit



after all of C will be visited

\Rightarrow before backtracking from w
DFS visits all of C including v

$\Rightarrow w.\text{post} > v.\text{post}$

Reverse of G (same vertex, reverse edges).

SCC

run dfs on G^R

unmark all v

$sComp = 0$

for each vertex v

in reverse post-visit number order

if not v .marked

$sComp++$

explore(w)

$O(|E| + |V|)$

Disjoint Sets:

$\{1\} \quad \{2\} \quad \{3\} \quad \{4\} \quad \{5\} \quad \{6\} \Rightarrow n \text{ elements}$

union(1, 4)

$\{1, 4\} \quad \{2\} \quad \{3\} \quad \{5\} \quad \{6\}$

union (5, 6)

{1, 4} {2} {3} {5, 6}

union (2, 4)

{1, 2, 4} {3} {5, 6}

find(2) = find(3) ? no

find(1) = find(4) ? yes



union & find program.

makeSet(x)

x.parent = x.

union (1, 4)



union (5, 6)

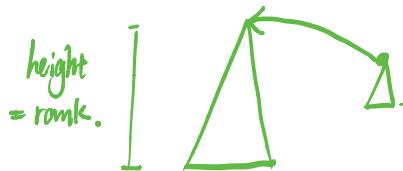


union (4, 2)

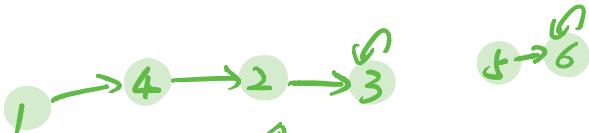


find(1) = find(3) not true.

find(1) = find(4) true.



union (1, 3)



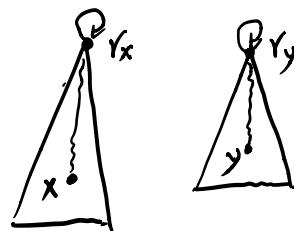
might take $O(n)$ to find.

(worst case).

Union by rank.

(make smaller (rank))

point to larger (rank))



Union (x, y).

$r_x = \text{find}(x)$

$r_y = \text{find}(y)$

if $r_y.\text{rank} < r_x.\text{rank}$

$r_y.\text{parent} = r_x$

else.

$r_x.\text{parent} = r_y$

if $r_x.\text{rank} = r_y.\text{rank}$



$r_y.rank++$

Union by rank.

find(x)

if $x \neq x.parent$

return find(x.parent).

return X.

union(4, 2)

union(1, 3)

union(5, 1)



Every subtree with $\underset{\text{(of root)}}{\text{rank}} = k$ has at least 2^k nodes.

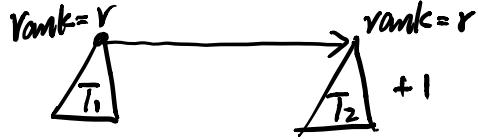
by induction:



Base $k=0$. $2^0=1$ at least 1 node.

I.H. assume for a fixed r. $\text{rank} = r$. nodes $\geq 2^r$

I.S. $\text{rank} = r+1$. rank incremented only when



by I.H. $\text{nodes}(T_1) \geq 2^r$

$\text{nodes}(T_2) \geq 2^r$

$\text{nodes}(T_1 \cup T_2) \geq 2^r + 2^r = 2^{r+1}$

makeSet(x)

x.parent = x

x.rank = 0

find(x)

if $x \neq x.\text{parent}$

x.parent = return find(x.parent).

return x.parent.

with path compression.



Union(x, y)

$\lg n \left\{ \begin{array}{l} r_x = \text{find}(x) \\ r_y = \text{find}(y) \end{array} \right.$

if $r_x \neq r_y$

if $r_x.\text{rank} > r_y.\text{rank}$

$r_y.\text{parent} = r_x$

else

$r_x.\text{parent} = y$

if $r_x.\text{rank} = r_y.\text{rank}$

$r_y.\text{rank}++$

find $\in O(\lg n)$

union $\in O(\lg n)$

unions
find

$(m+n) \lg^* n$

node at rank = k



rank = k

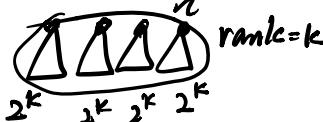
nodes in subtree $\geq 2^k$

Upper bound on number of nodes
rank = k



rank = k

$1 \leq \text{nodes at rank } k \leq \frac{n}{2^k}$



$1 \leq \frac{n}{2^k}$

$2^k \geq n \quad k \leq \lg n$

↑
rank.

Greedy \longrightarrow Try to locally minimize / maximize
Minimum Spanning Tree

goal: global optimum

Given a graph:

Tree: graph . acyclic , connected ,
undirected)

cut property).

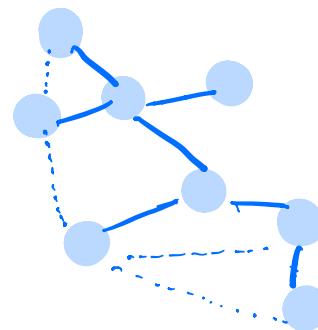
with a weight function:

$w: E \rightarrow \mathbb{R}$

suspense that at some point

we want to find a spanning tree

"we make build a "subtree" X (a subgraph of G that contains V).
S.t. X is an subtree of some
MST T .



to find next edge to add:
 \rightarrow partition $S, V-S = A$. put

st. no edge in X of minimum weight $w(T) = \sum_{e \in T} w(e)$ minimum
gose from , M Vmilk view

Claim: let e be an edge of minimum weight that crosses the
cut $\Rightarrow X \cup \{e\}$ belongs to same mst of G

Prim (G)

for each $v \in V$ $\begin{cases} v.\text{cost} = \infty \\ v.\text{pred} = \text{null} \end{cases}$

pick $S \subseteq V$

$S.\text{cost} = 0$

Q. makeQueue(V)

while not Q.isEmpty()

$v = Q.\text{extractMinl})$

for each $(v, w) \in E$

if $w(v, w) < v.\text{cost}$

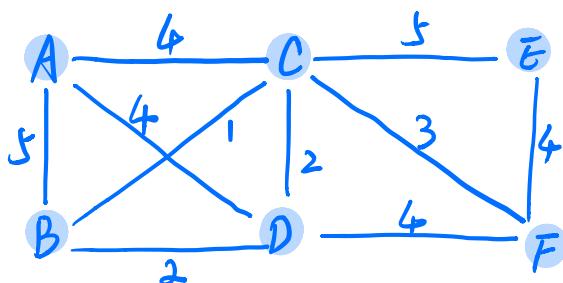
$v.\text{cost} = w(v, w)$

$v.\text{pred} = w$

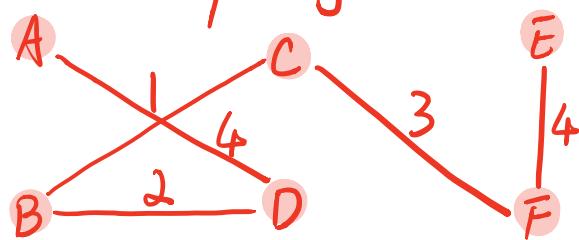
Q.decreasekey(w)

$O((|V| + |E|) \lg |V|)$

Example.



minimal spanning tree.



total weight 14.

Kruskal

for each $v \in V$

makeSet(v)

$O(|V|)$

sort E by weight $O(|E| \lg |V|)$

for each $(u, v) \in E$ in ascending order

if $\text{find}(u) \neq \text{find}(v)$ $|E| \lg^* |V|$

$X = X \cup \{(u, v)\}$

$\text{union}(u, v)$ $(|V|-1) \lg^* |V|$

return X

$O(\underline{|E| \lg |V|} + (|V| + |E|) \lg^* |V|)$

↓
linear

↓
almost constant.

↓
 $O(|V| + |E|)$

A	B	C	D	E	F
0/-	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$	$\infty/-$
A	$5/A$	$6/A$	$4/A$	$\infty/-$	$\infty/-$
D	$2/D$	$2/D$	$\infty/-$	$4/D$	
B		$1/B$	$\infty/-$	$4/D$	
C			$5/C$	$3/C$	
F				$4/F$	
E					

Greedy Algorithm.

file compression.
(text)

symbols (a, b, c, \dots)
ASCII $\xrightarrow{8\text{ bits}}$
 n symbols $\xrightarrow{\text{length} = 8n}$
fixed length code

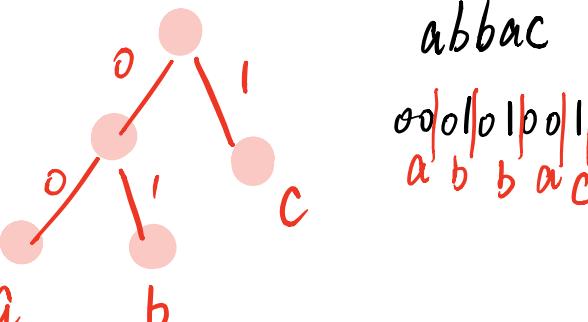
variable length encoding

a	101	$101 <_{bc}$
b	10	
c	1	

frequency table

1	a	$f[1]$
2	b	$f[2]$
3	c	$f[3]$
:	:	:

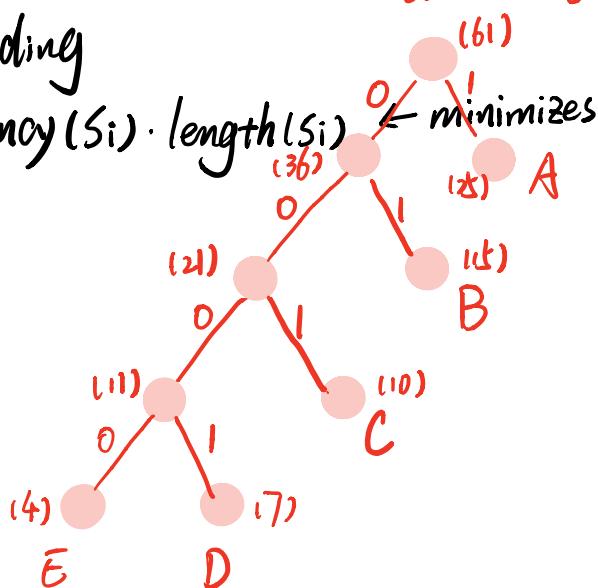
symbols are in the leaves

abbac

 abbac
 00|01|01|00||
 a b b a c

optimal encoding

$\sum_i \text{frequency}(S_i) \cdot \text{length}(S_i)$ \leftarrow minimizes

A-25
 B-15
 C-10
 D-7
 E-4



A 1
 B 01
 C 001
 D 0001
 E 0000

Huffman ($f[1], f[2], \dots, f[n]$)

"local optimal thought"

build n nodes $N = \{(i, f[i])\}$

Q.buildQueue(N)

while Q.size > 1

$O(n)$ | $u = Q.\text{extractMinl}$
 $v = Q.\text{extractMinl}$

lgn | Create node $w = (-, u.f + v.f)$
| set u and v to be children of w .

| Q.insert(w)

return Q.extractMin()

$O(n \lg n)$

Dynamic Programming

$$\text{fib}(0) = 1$$

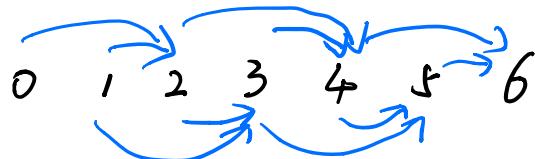
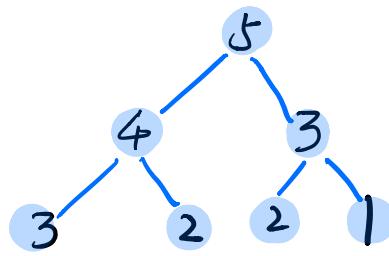
if ...

$$\text{fib}(1) = 1$$

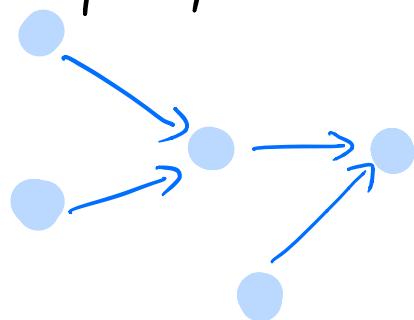
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

$$\text{fib}(2) = 2$$

:



DAG of subproblems



$\text{fib}(0) = 1$
 $\text{fib}(1) = 1$
 for i to n :
 $\text{fib}(i) = \text{fib}(i-1) + \text{fib}(i-2)$

Store them.

- find an order to traverse the DAG.

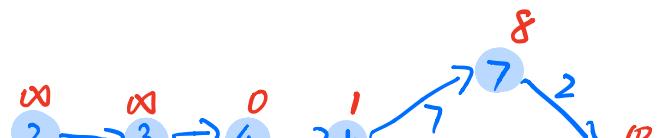
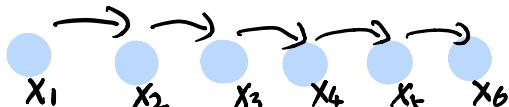
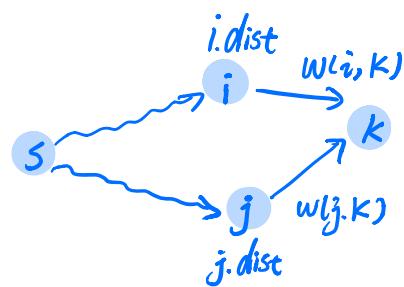
memorization

recursive algorithm.

immediately before doing recursive call check if the solution is stored. else use the recursive call and store the solution

Find shortest paths in a DAG (single source)

$$k.\text{dist} = \min_{(j,k) \in E} \{ j.\text{dist} + w(j,k) \}$$



linearize
Process them in

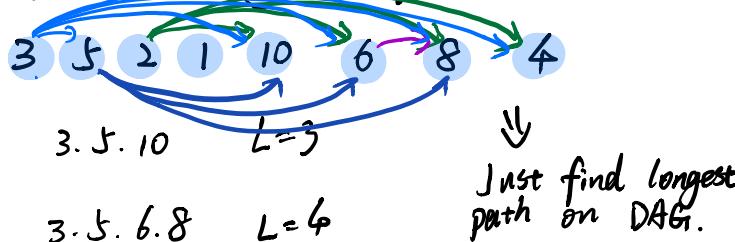
$\{x_1\}$

$\{x_1, x_2\}$

$\{x_1, x_2, x_3\}$

:

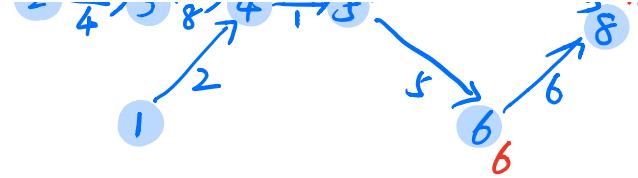
Longest increasing subsequence



if $u < v \Rightarrow (u, v) \in E$

in the order given

$$w(u, v) = 1$$



Algorithm

for all $v \in V$:

$$v.dist = \infty$$

$$s.dist = 0$$

compute a linearization $O(|V| + |E|)$

for each $v \in V$ in the order of the linearization

$$d = \min_{(u, v)} \{u.dist + w(u, v)\}$$

$$v.dist = \min \{v.dist + d\}$$

$$\text{total: } O(|V| + |E|)$$

Edit distance

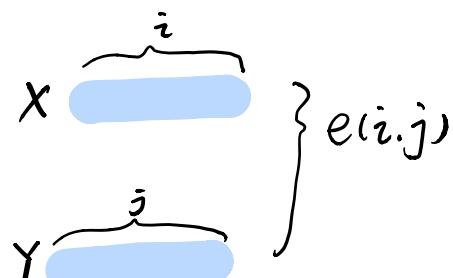
\rightarrow (min)

given two words how many changes to transfer one to the other?

$X = \underline{\text{E}} \text{ X} \underline{\text{T}} \underline{\text{I}} \text{ N} \text{ C} \text{ T} \text{ I} \text{ O} \text{ N}$
 $Y = \underline{\text{I}} \text{ N} \underline{\text{T}} \underline{\text{E}} \text{ N} \text{T} \text{ I} \text{ O} \text{ N}$
 $\quad \quad \quad \underline{\text{E}} \text{ X} \underline{\text{T}} \underline{\text{I}} \text{ N} \text{ C} \text{ T} \text{ I} \text{ O} \text{ N}$
 $\quad \quad \quad \underline{\text{I}} \text{ N} \underline{\text{T}} \underline{\text{E}} \text{ N} \text{-} \text{T} \text{ I} \text{ O} \text{ N}$

$$e = 8$$

$$e = 4$$



$X \quad \underline{\quad \quad \quad} \quad X[i] \quad e(i,j) = e(i-1, j) + 1$

$X \quad \underline{\quad \quad \quad} \quad - \quad e(i,j) = e(i, j-1) + 1$

X $X[i]$
 Y $Y[j]$ $e(i,j) = e(i-1,j-1)$
 $+ \text{diff}(i,j)$

$$e(i,j) = \min \{ e(i-1,j), e(i,j-1), e(i-1,j-1) + \text{diff}(i,j) \}$$

$$\text{diff}(i,j) = \begin{cases} 0 & \text{if } X[i] = Y[j] \\ 1 & \text{if } X[i] \neq Y[j] \end{cases}$$

for $i=0$ to n :

$$e(i,0) = i$$

for $j=0$ to m :

$$e(0,j) = j$$

for $i=1$ to n :

for $j=1$ to m :

$$e(i,j) = \min \{ e(i-1,j), e(i,j-1), e(i-1,j-1) + \text{diff}(i,j) \}$$

$O(mn)$

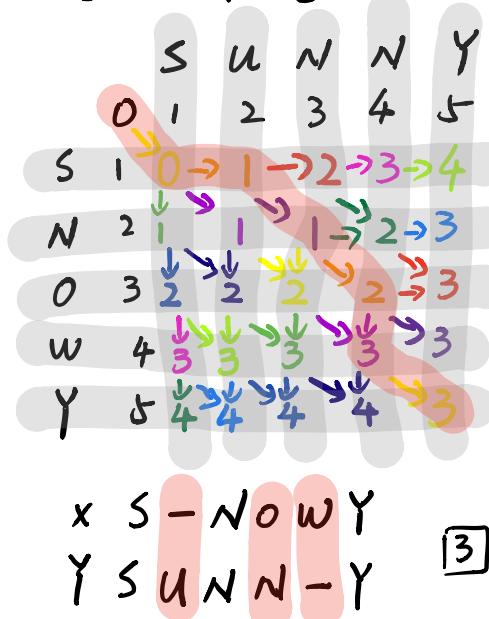
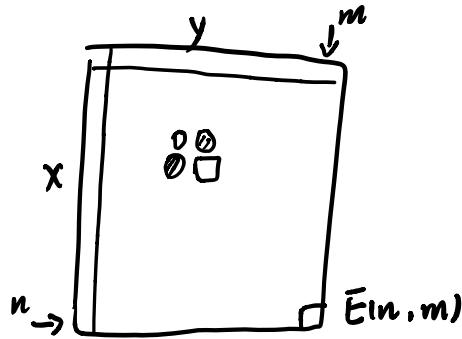
Chain Matrix Multiplication

A 5×20	B 20×1	C 1×10	D 10×100
----------------------	----------------------	----------------------	------------------------

$$\begin{array}{c}
 A \times B = \left(\begin{array}{c} \text{ } \\ \text{ } \\ \text{ } \\ \text{ } \\ \text{ } \end{array} \right) \\
 n \times r \quad r \times m \quad \underbrace{\quad}_{n \times m} \quad O(nmr)
 \end{array}$$

Edit distance.

$$E(i, j) = \min \{ E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + \text{diff}(i, j) \}.$$



- ① divide into subproblems
- ② combine subproblems to solve problem
- ③ order (DAG)

Chain Matrix Multiplication

$$(A_1, A_{i+1} \dots A_k)(A_{k+1} \dots A_j)$$

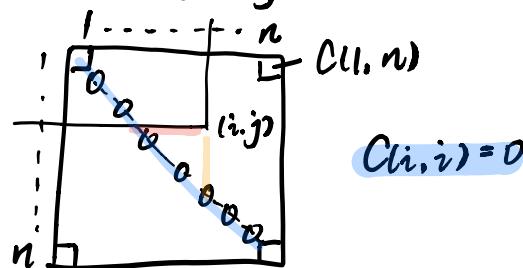
$$A_0 \quad m_0 \times m_1$$

$$A_1 \quad m_1 \times m_2$$

$$\vdots \quad \vdots$$

$$A_n \quad m_n \times m_{n+1}$$

$$C(i, j) = \min_{i \leq k < j} \{ C(i, j) + C(k+1, j) + m_{i-1}m_km_j \}$$



Chain Matrix Multiplication (m_0, ..., m_n)

for $i=1$ to n

$$C(i, i) = 0$$

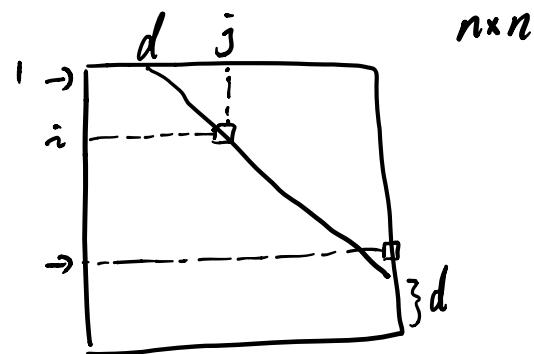
for $d=2$ to n

for $i=1$ to $n-d+1$

$$j = d+i-1$$

$$C(i, j) = \infty$$

for $k=i$ to $j-1$



$$v = C(i, k) + C(k+1, j) + m_{i-1} m_k m_j$$

if $v < C(i, j)$

$$C(i, j) = v$$

$$R(i, j) = k$$

$O(n^3)$

$$A_1. 50 \times 20 \quad A_2. 20 \times 1 \quad A_3. 1 \times 10 \quad A_4. 10 \times 100$$

$$\min: k=1 \quad 0 + 3000 + 50 \cdot 20 \cdot 100 = 10300$$

$$k=2 \quad 1000 + 1000 + 50 \cdot 1 \cdot 100 = 10000$$

$$k=3 \quad 1500 + 0 + 50 \cdot 10 \cdot 100 = 51500$$

$$(A_1 A_2)(A_3 A_4)$$

50	20	10	100
1	2	3	4
20	C	0 1000 1500 7000	
1	2	0 200 3000	
10	3	0 1000	
100	4	0	

1	2	3
2	2	
3		

All pairs shortest paths

$$M_k(i, j) = \begin{cases} w(i, j) & \text{if } (i, j) \in E \\ 0 & \text{if } i=j \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \end{cases}$$

for $k=1$ to n

compute all pairs of shortest paths going from i to j using $1, 2, \dots, k-1$ as intermediate vertices

compute i to j using $1 \dots k$

for $i=1$ to n

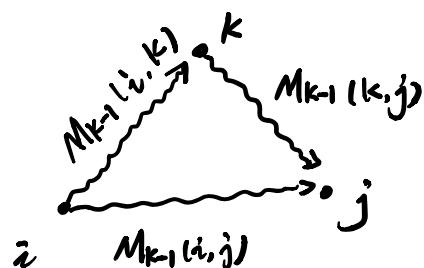
for $j=1$ to n

$$v = M_{k-1}(i, k) + M_{k-1}(k, j)$$

if $(v < M_{k-1}(i, j))$

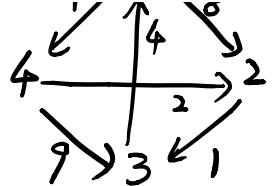
$$M_k(i, j) = v$$

$\pi(i, j) = \pi(k, j)$ π is the matrix of parents

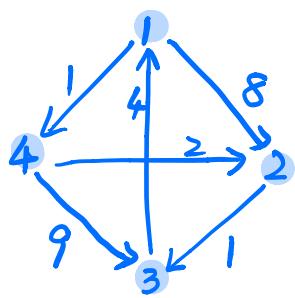


$$M_k(i, j) = \min \{M_{k-1}(i, k) + M_{k-1}(k, j), M_{k-1}(i, j)\}$$

1	2	3	4



1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0



Floyd-warshall

$$M_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & - & 1 \\ 2 & - & 0 & 1 & - \\ 3 & 4 & - & 0 & - \\ 4 & - & 2 & 9 & 0 \end{array}$$

$$M_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & - & 1 \\ 2 & - & 0 & 1 & - \\ 3 & 4 & 12 & 0 & 5 \\ 4 & - & 2 & 9 & 0 \end{array}$$

check $M_1(m, n)$ with $M_1(m, 1) + M_1(1, n)$. choose smaller one.

$$\begin{aligned} M_k &: M_k(m, n) \\ &\text{vs} \\ M_k(m, k) + M_k(k, n) \end{aligned}$$

$$M_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & - & 0 & 1 & - \\ 3 & 4 & 12 & 0 & 5 \\ 4 & - & 2 & 3 & 0 \end{array}$$

complexity: $O(n^3)$

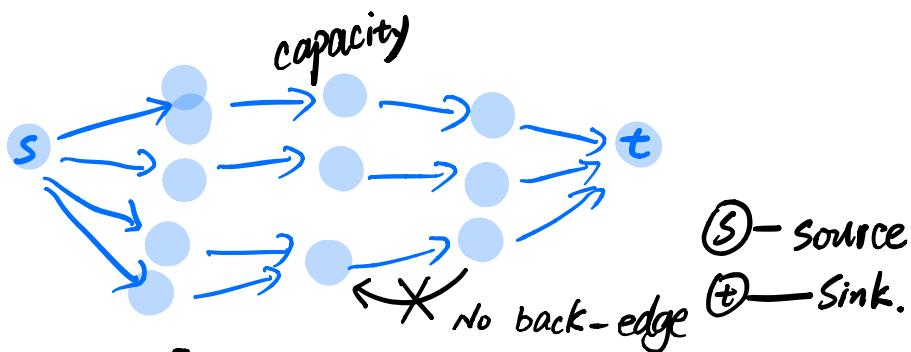
Bellman-Ford: $O(|E||V|)$ for each
Floyd-warshall is better.

$$M_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 12 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

$$M_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 4 & 0 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 7 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

$$3,2 \rightarrow 3,4 + 4,2$$

Max-Flow Problem.



$$G = (V, \bar{E})$$

$\nexists u, v \in V$, such $(u, v) \in \bar{E}$ and $(v, u) \in \bar{E}$

s — no in edges

t — no out edges

Capacity: $C: \bar{E} \rightarrow \mathbb{R}^+$

Goal: find a flow

$$f: \bar{E} \rightarrow \mathbb{R}^+$$

value of the flow.

$$\sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$$

$$1) f(u, v) \leq C(u, v)$$

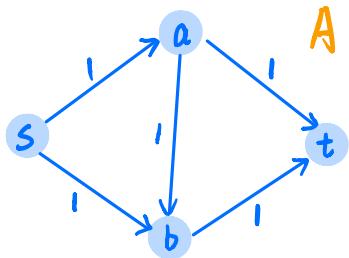
$$\forall (u, v) \in E$$

2) flow conservation

$$\forall v \in V - \{s, t\}$$

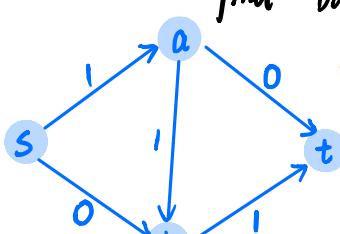
$$f \xrightarrow{s} ; \xleftarrow{v} f \quad \text{in} == \text{out}$$

Example.



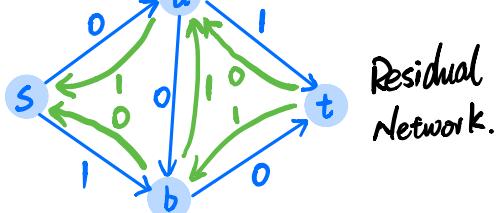
Algorithm:

find augmenting path.
 $p = s \xrightarrow{a} b \xrightarrow{t}$



find bottleneck capacity (= 1)

$A - B$: compute residual capacity



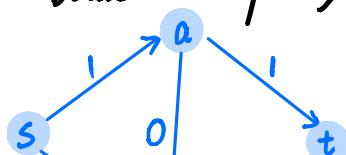
Residual Network.

$$C^*(u, v) = \begin{cases} C(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in \bar{E} \end{cases}$$

Augmenting path

$$p = s \xrightarrow{b} a \xrightarrow{t}$$

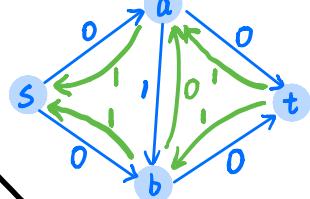
bottleneck capacity = 1



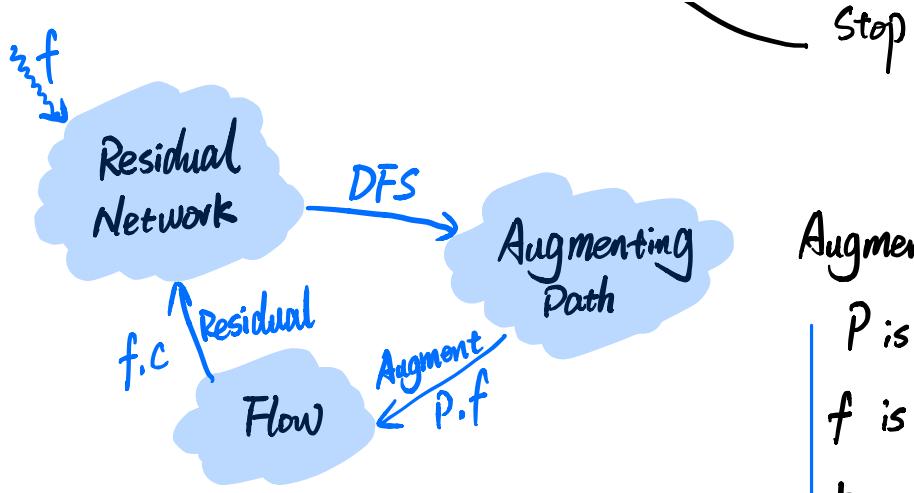
claim
this

is
max.

→ Repeat: Residual network.



find augmenting path?
No.



Residual (C, f)

C is the original capacities

f is a flow

for each $(u, v) \in E$

$$C^f(u, v) = C(u, v) - f(u, v)$$

$$C^f(v, u) = f(u, v)$$

return C^f

Augment (P, f, C^f)

P is an augmenting path

f is a flow

$$b = \min_{i, j \in P} \{C^f(i, j)\}$$

for each $(i, j) \in P$

if $(i, j) \in \bar{E}$

$$f(i, j) = f(i, j) + b$$

else

$$f(i, j) = f(i, j) - b$$

return f

Ford and Fulkerson ($G = (V, \bar{E})$, $C: \bar{E} \rightarrow \mathbb{R}^+$)

$\forall (u, v) \in \bar{E}$

$$f(u, v) = 0$$

$C^f = \text{Residual}(C, f)$

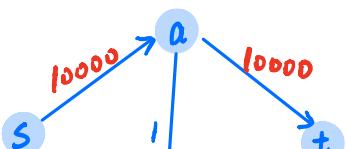
while there is an augmenting path P on C^f

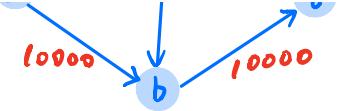
$f = \text{Augment}(P, f, C^f)$

$C^f = \text{Residual}(C, f)$

return f

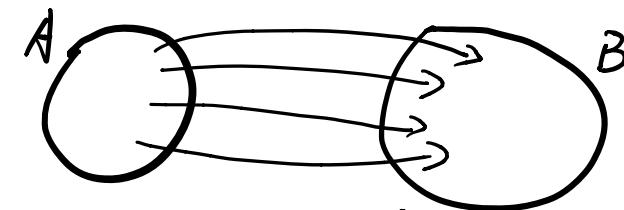
Bad situation





Ford & Fulkerson

- Flow:
- $f(u, v) \leq C(u, v)$
 - flow conservation
flow in = flow out
 $\Rightarrow \bullet \Rightarrow$



$$\sum_{\substack{a \in A \\ b \in B \\ (a, b) \in E}} f(a, b) = \sum_{a \in A} f_a$$

$$Cf(u, v) = \begin{cases} C(u, v) - f(u, v) & \text{if } (u, v) \in E \\ f(u, v) & \text{if } (u, v) \notin E \end{cases}$$

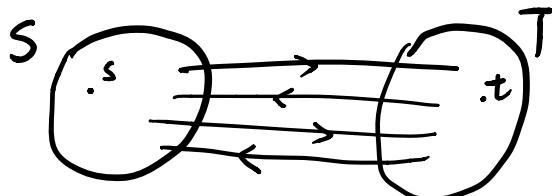
$$\sum_{v \in B} f = \sum_{\substack{b \in B \\ (v, b) \in E}} f(v, b)$$

Def:

cut (S, T)

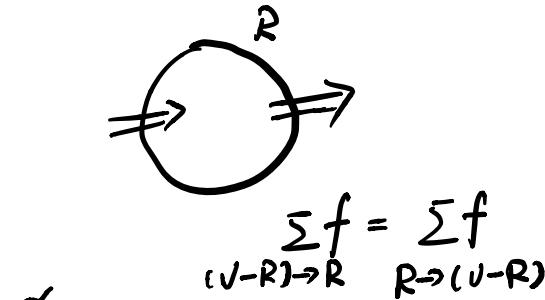
- partition: $S, \bar{T} \subseteq V$. $S \cup T = V$. $S \cap T = \emptyset$

$s \in S$. $t \in \bar{T}$



Def:

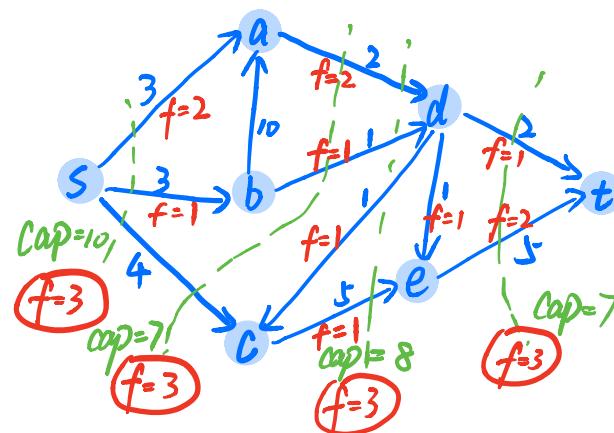
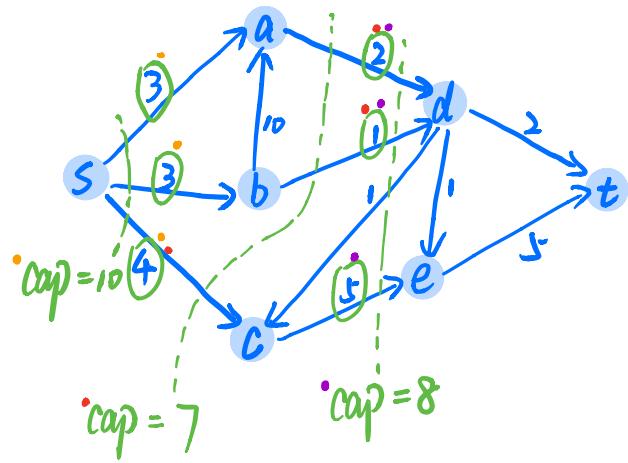
$$\text{flow}(S, \bar{T}) = \sum_{S \rightarrow T} f - \sum_{T \rightarrow S} f$$



Def: capacity of a cut

$$\text{cap}(S, \bar{T}) = \sum_{S \rightarrow T} C$$

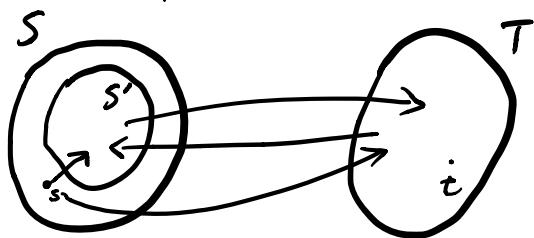
From S to T , only consider one direction



$$\nu(f) = \sum_{S \rightarrow V} f$$

Theorem Let S, \bar{T} be a cut

$$v(f) = \text{flow}(S, T)$$



$$S' = S - \{s\}$$

Since $s, t \notin S'$

flow into $S' = \text{flow out of } S'$

$$\sum_{s \rightarrow S'} f + \sum_{T \rightarrow S'} f = \sum_{S' \rightarrow T} f$$

$$\Rightarrow \sum_{S \rightarrow S'} f = \sum_{S' \rightarrow T} f - \sum_{T \rightarrow S'} f$$

$$\text{flow}(S, T) = \sum_{S \rightarrow T} f + \sum_{S' \rightarrow T} f - \sum_{T \rightarrow S'} f$$

$$= \sum_{S \rightarrow T} f + \sum_{S \rightarrow S'} f$$

$$= \sum_{S \rightarrow V} f = v(f)$$

$$\text{flow}(S, T) = v(f)$$

Theorem $v(f) \leq \text{cap}(S, T)$

$$v(f) = \text{flow}(S, T)$$

$$= \sum_{S \rightarrow T} f - \sum_{T \rightarrow S} f \leq \sum_{S \rightarrow T} f \leq \sum_{S \rightarrow T} C = \text{cap}(S, T)$$

F&F returns the $\max^{(f^*)}$ Flow

i.e. \exists flow f

$$v(f) \leq v(f^*)$$

we know that $C^* = 0$ for all edges
(in Residual Network) crossing the cut

$$S^* \Rightarrow T^* \quad C^*(u, v) = 0 =$$

$$(u, v) \in E \Rightarrow 0 = C(u, v) - f(u, v)$$

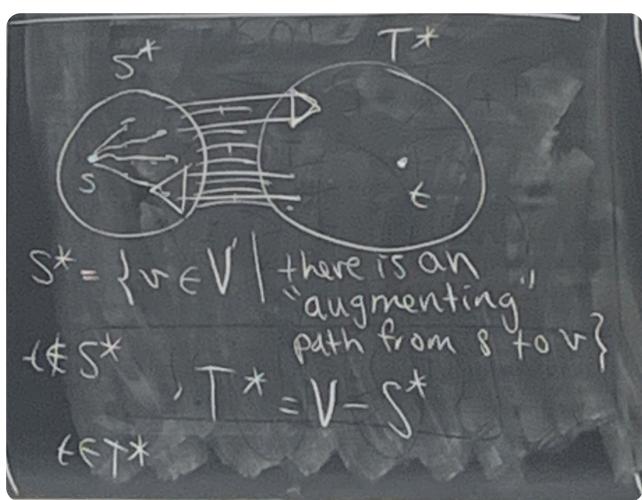
$$\Rightarrow f(u, v) = C(u, v)$$

$$S^* \Leftarrow T^* \quad C^*(u, v) = 0 = f(u, v)$$

$$f(S^*, T^*) = \sum_{S^* \rightarrow T^*} f - \sum_{T^* \rightarrow S^*} f \xrightarrow{\rightarrow 0}$$

$$\sum_{S^* \rightarrow T^*} C$$

$$= \text{cap}(S^*, T^*)$$



$$v(f) \leq \text{cap}(s^*, T^*) = \text{flow}(s^*, T^*) \\ = v(f^*)$$

Augment updates flow

Residual ... computes C^f

initialize $O(|E|)$

while augmenting path

 | Augment $O(|E|)$

 | Residual $O(|E|)$

$v(f^*) \cdot |E|$

use shortest path augmenting ($BFS \rightarrow$) $|V||E|$

total: $O(|V||E|^2)$

Edmonds - Karp

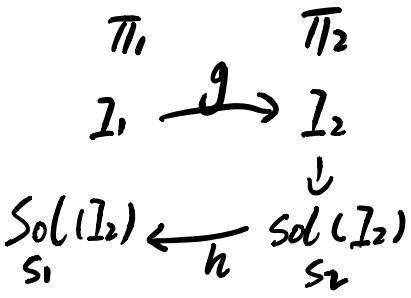
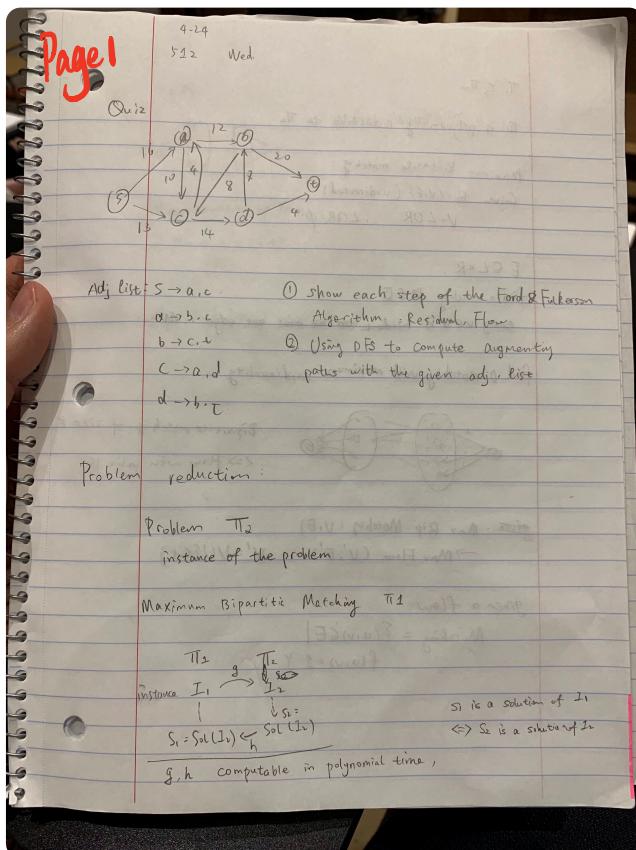
Problem Reduction

Problem Π_2 (maxflow)

instance of the problem

Maximum Bipartition Matching

$\vdash \Pi_1$

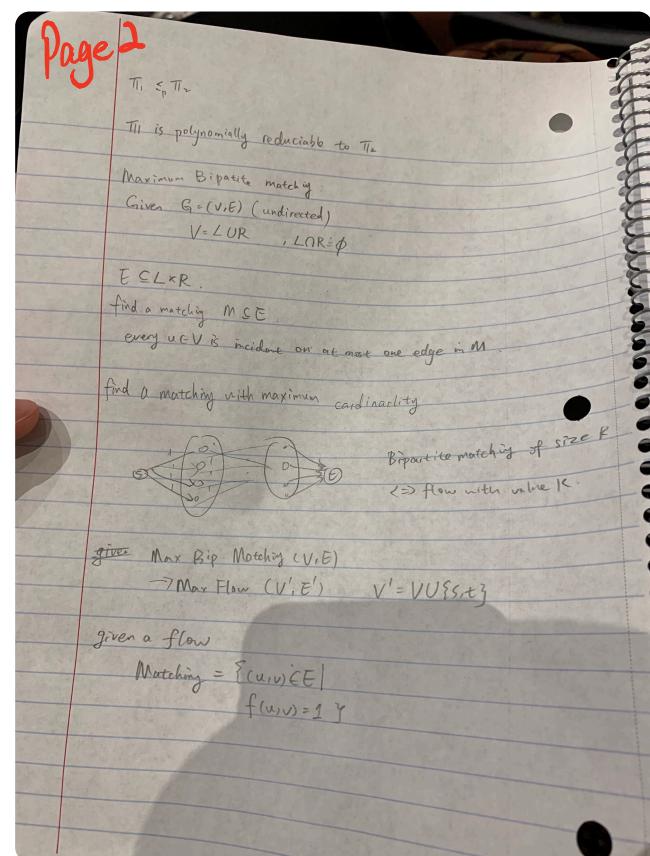
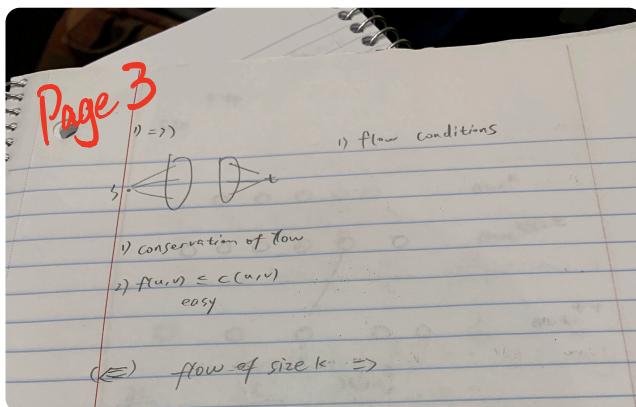


g,h Computable in polynomial time.

S_1 is a solution of I_1 ,



S_2 is a solution of I_2



Alphabet

Σ = set of symbols

$$\Sigma = \{0, 1\} \quad \Sigma' = \{a, b, c, d, \dots, z\}$$

language : L = set of strings over an alphabet

$$\Sigma = \{0, 1, 2, 3, \dots, 9\}$$

$L_{\text{prime}} = \{ x \text{ strings over } \Sigma \mid x \text{ is prime} \}$

Regular operations.

$$1) L_1 \cup L_2 \quad 2) L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\} \quad \text{example. } L_1 = \{aab, ba\} \\ (concatenation) \quad L_2 = \{aba, aa\}$$

3) Kleene Star

$$L^* = \bigcup_{n=0}^{\infty} L^n + \Sigma^* = \text{all possible strings over } \Sigma$$

$$L_1 L_2 = \{aababa, aabaa, baaba, baaa\}$$

Example.

$$\Sigma = \{0, 1\}$$

$\star \epsilon \in L^0$
 \nwarrow empty string

$$\Sigma^* = \Sigma^\circ \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

$$= \{\varepsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \dots$$

$$L_{\text{prime}} \quad \Sigma = \{0, 1, 2, 3, \dots, 9\}$$

Problem Prime

input $x \in \Sigma^*$ Question: $x \in L_{\text{prime}}$

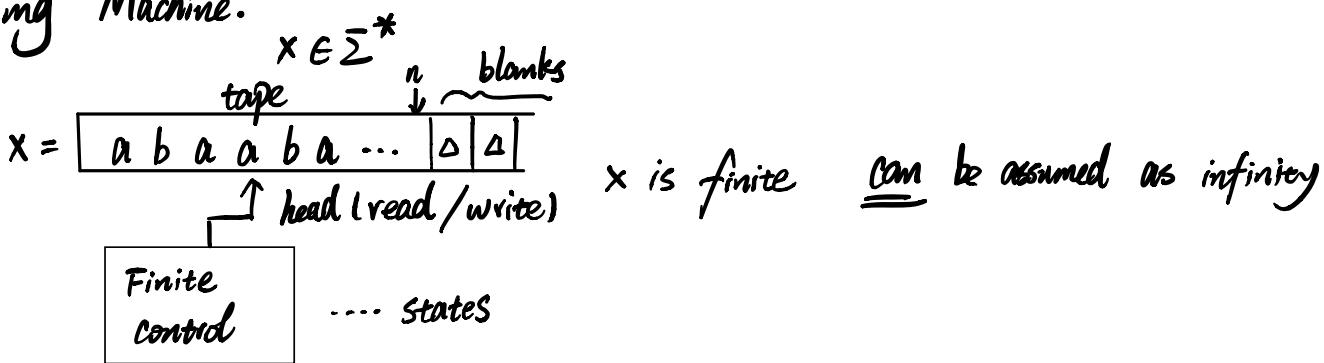
Machine :



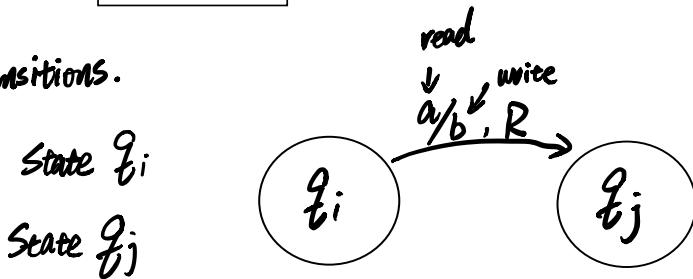
L' prime = language accepted by M_p

L' is the language of M_2 ; $L' = L(M_2)$

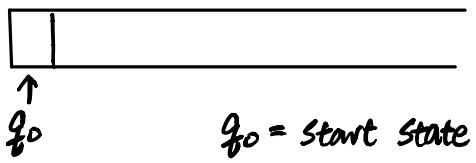
Turing Machine.



Transitions.



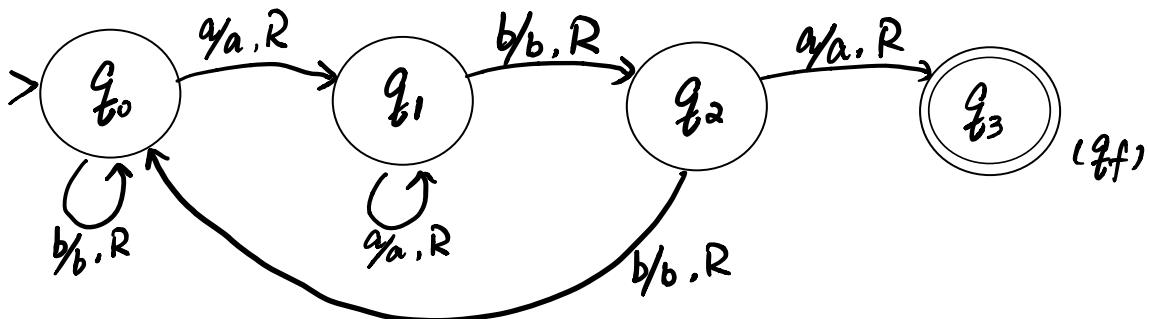
Starting configuration



final / accepting state q_f (one) sometimes can be multiple

$$L_1 = \{x \in \Sigma^* \mid x \text{ contains } aba\}$$

Transition Diagram.



Instantaneous configurations:

aabab yield

$q_0 a a b a b \Delta \vdash$

$aq_1ababa \leftarrow aaq_1bab\Delta$
 $\leftarrow aabq_2ab\Delta \leftarrow aabaq_3b\Delta$
Accept

Church-Turing thesis

Given any algorithm \exists a TM that simulates it.

Universal TM(Turing Machine)

Takes an input the encoding of another TM and simulates it.

Formal Definition of a TM (Deterministic TM)

$$M(Q, \Sigma, \Gamma, \delta, q_0, q_f)$$

Q : set of states (finite)

Σ : input alphabet

Γ : tape alphabet ($\Sigma \subseteq \Gamma$)

δ : transition function $\delta: Q \times \Gamma \rightarrow Q \times \Sigma \times \{L, R\}$

q_0 : start state set of all possible pairs

q_f : final state

Types of problems

- Decision Problems

 - Yes , . No

- Optimization Problems

 - find max or min.

 - (max-flow) \leftarrow find value

- Search Problems

 - find the flow st.

 - its value is maximum.

Decision Problems

P : A problem Π is in $P \Leftrightarrow \exists$ a DTM
s.t. M accepts Π

$$L(M) = \Pi \text{ in Polytime}$$

\uparrow can be decided by DTM in polynomial time

Deterministic (DTM):

$$\delta(q_i, a) = (\dots)$$

$$\delta: Q \times I \rightarrow Q \times I \times \{R, L\}$$

Non-deterministic:

$$\delta(q_i, a) = \{(\dots), (\dots), (\dots)\}$$

$$\delta: Q \times I \rightarrow P(Q \times I \times \{R, L\})$$

\uparrow
Power set

Sample: SAT

(Satisfiability)

Given $\phi(x_1, x_2, x_3, \dots, x_n)$

x_i are boolean variables

\exists a truth assignment to x_{n-1}, x_n s.t. $\phi(\dots) = T$

$$\rightarrow \phi = (P \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

is ϕ SAT? $P=T$ $q=T$ $r=T \rightarrow$ certificate (C)

SAT $\in P$. but verified in Polynomial time.

\exists a DTM M that verifies $(\text{Ins}(\text{SAT}), C)$ in poly-time.

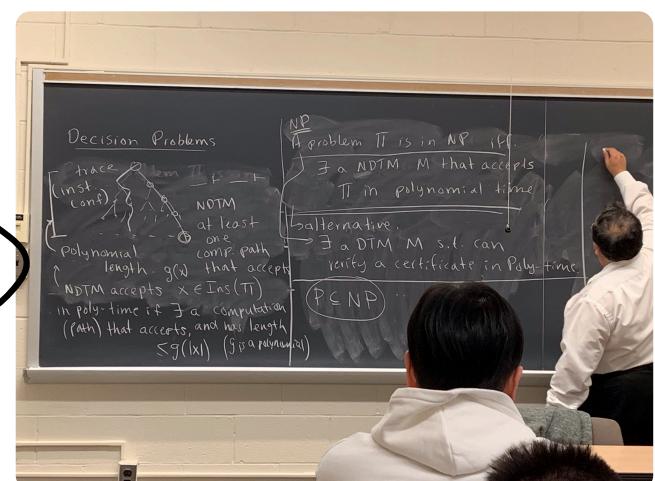
Polynomial Reducibility

$\Pi_1 \quad \Pi_2$

$$\begin{aligned} I_1 = \text{Ins}(\Pi_1) &\xrightarrow{f} I_2 = f(I_1) = \text{Ins}(\Pi_2) \\ I_1 \in \Pi_1 &\Leftrightarrow I_2 \in \Pi_2 \end{aligned}$$

$\Pi_1 \leq_p \Pi_2$ iff \exists a polynomially computable function f s.t.

$$g(I_1)$$



Theorem

If $\pi_1 \leq_p \pi_2$ and $\pi_2 \in P \Rightarrow \pi_1 \in P$

proof: 1) given $I_1 \in \text{Ins}(\pi_1)$. $f(\pi_1) = I_2 = \text{Ins}(\pi_2)$

$$|I_2| = g(|I_1|)$$

2) solve I_2 in poly time

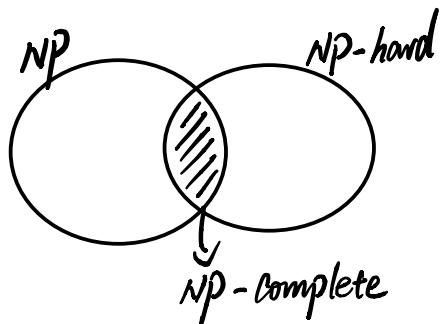
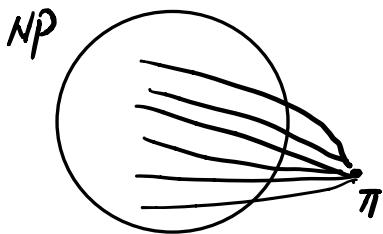
$$\text{time} = h(|I_2|) \quad \stackrel{(h)}{\uparrow}$$

(so we solved I_1)

$$\text{time} = h(g(|I_1|)) \Rightarrow \pi_1 \in P$$

Def: NP-hard

iff $\forall \pi_x \in NP, \pi_x \leq_p \pi$



NP-complete:

- 1) NP
- 2) NP-hard.

suppose that $\exists \pi' \in NP\text{-complete}$ and $\pi' \in P$

$\Rightarrow \pi' \in NP\text{-hard}$. $\forall \pi_x \in NP, \pi_x \leq_p \pi'$ and $\pi' \in P$

$\therefore \forall \pi_x \in NP \Rightarrow \pi_x \in P$

$$NP \subseteq P$$

we knew $P \subseteq NP$

$$\therefore P = NP$$





CS512 LECTURE NOTES - LECTURE 1

1 Algorithms

An Algorithm is:

- sequence of steps
- is not ambiguous
- process to solve a problem
- takes an input
- produces an output
- Terminates in a finite amount of time

Examples:

1. Sorting Algorithm

Input: finite array of numbers

Output: sorted finite array of numbers

2. Shortest path

Input: Graph (V, E) , v_1, v_2

Output: $v_1 = x_1, x_2, x_3, x_4, \dots, x_n = v_2$ (shortest path)

3. Prime

Input: $n \in \mathbb{N}$

Output: True or False (prime or not prime)

Problems for which the output is either True or False are called decision problems. (Recall DFA, PDA, Turing Machines from CS205, they either accept or reject).

DFA. Deterministic Finite Automaton

有穷自动机. 通过 event 和 state 得到下一个 state

即 $event + state = nextstate$.

系统中有多个节点，通过传递进入的 event，来确定走哪个路由至另一个节点，而节点是有限的。

Why "Computer" algorithms?

Algorithms have existed since ancient times. Remember Euclid's GCD algorithm, generation of all primes less than a fixed given integer n , etc.

However, due to the speed with which a computer can execute the algorithms, most algorithms are intended to be used by a computer.

Analysis of Algorithms

There are three aspect that are important to analyze:

- Correctness: Formal proof that the algorithm halts and produces the correct output given an input.
- Time complexity:
 - Measures the time it takes to solve a given problem.
 - Is used to compare one algorithm to another one.
 - Is used to determine if we have the best possible algorithm for a problem (optimal) or if there is room for improvement.
- Space complexity: Measures the amount of memory used by an algorithm.

1.1 Measuring execution time

- Not useful to measure the actual time (in seconds) since we might be using different computers with different speeds.
- The basic idea is to count operations
- The running time is:
 1. Proportional to the number of operations
 2. A function of the length of the input

NOTICE: The actual running time is a function of the input, not only of the input length. In order to simplify our analysis we use three possible cases:

★ **Best case:** Among all possible input instances of the problem (P_n) with length n , we compute the minimum execution time:

$$B(n) = \min\{time(I) | I \in P_n\}$$

★ **Average case:** Among all possible input instances of the problem (P_n) with length n , we compute the average execution time:

$$A(n) = \sum_{i \in P_n} \{probability(I)time(I)\}$$

★ **Worst case:** Among all possible input instances of the problem (P_n) with length n , we compute the maximum execution time:

$$W(n) = \max\{time(I) | I \in P_n\}$$

1.2 Big Oh and Big Omega

The functions that we are interested in satisfy:

1. Defined over the set of natural numbers
2. Are positive non-decreasing

Given a function $g : \mathbb{N} \rightarrow \mathbb{R}^+$ we can classify other functions according to the following definitions:

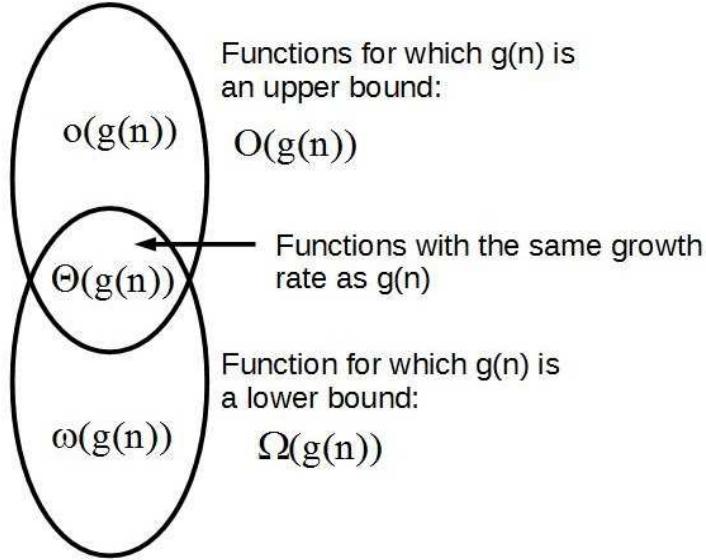
- Functions $f(n)$ for which $g(n)$ is an upper bound (i.e. functions that grow no faster than $g(n)$)

$$O(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ | f(n) \leq cg(n) \text{ for some } c > 0\}$$

- Functions $f(n)$ for which $g(n)$ is a lower bound (i.e. functions that grow at least as fast as $g(n)$)

$$\Omega(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ | f(n) \geq cg(n) \text{ for some } c > 0\}$$

1.3 Little o and Little ω



Notice that

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

So we can define

$$\Theta(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid c_1g(n) \leq f(n) \leq c_2g(n) \text{ for some } c_1 > 0, c_2 > 0\}$$

where $f(n)$ is "sandwiched" between $c_1g(n)$ and $c_2g(n)$

In order to show that we can use these sets to classify functions according to their rate of growth, we will show that the relation defined by Θ is an equivalence relation, i.e. it is reflexive, symmetric and transitive.

1.4 Θ is an equivalence relation

Reflexive:

Clearly $f(n)$ satisfies $c_1f(n) \leq f(n) \leq c_2f(n)$, therefore $f(n) \in \Theta(f(n))$.

Symmetric:

Let $f(n) \in \Theta(g(n)) \Rightarrow c_1g(n) \leq f(n) \leq c_2g(n)$.

From the first inequality we have that:

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

And from the second inequality we have that:

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

Let $c'_1 = \frac{1}{c_2}$, and $c'_2 = \frac{1}{c_1}$, we have that:

$$c'_1 f(n) \leq g(n) \leq c'_2 f(n)$$

Transitive:

Assume that $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$. Using the definition of Θ we have that:

$$\begin{aligned} c_1 g(n) &\leq f(n) \leq c_2 g(n) \\ c_3 h(n) &\leq g(n) \leq c_4 h(n) \end{aligned}$$

From the first inequality we have:

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

Combining it with the third inequality we have:

$$c_3 h(n) \leq g(n) \leq \frac{1}{c_1} f(n) \quad (5)$$

From the second inequality we have:

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

Combining it with the fourth inequality we have:

$$\frac{1}{c_2} f(n) \leq g(n) \leq c_4 h(n) \quad (6)$$

If we let $c'_1 = c_1 c_3$, $c'_2 = c_2 c_4$, from (5) and (6) we get:

$$c'_1 h(n) \leq f(n) \leq c'_2 h(n)$$

Therefore, $f(n) \in \Theta(h(n))$

So we have that we can use $\Theta(g(n))$ to classify functions according to their asymptotic growth rate into equivalence classes.

1.5 Using limits

Since we are interested in the asymptotic behavior of the functions, it is sometimes easier to use limits to determine their growth rate. It is easy to see that:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} c & \Rightarrow f(n) \in \Theta(g(n)) \\ 0 & \Rightarrow f(n) \in o(g(n)) \\ \infty & \Rightarrow f(n) \in \omega(g(n)) \end{cases}$$

CS512 LECTURE NOTES - LECTURE 2

Example:

Show that $3n^2 - 2n + 7 \in \Theta(n^2)$.

1. Using the definition (we will show $3n^2 - 2n + 7 \in O(n^2)$, the proof that $3n^2 - 2n + 7 \in \Omega(n^2)$ is similar).

We have to show that $\exists c > 0$ such that

$$3n^2 - 2n + 7 \leq c \Rightarrow 3 - \frac{2}{n} + \frac{7}{n^2} \leq c$$

Notice that

$$3 - \frac{2}{n} + \frac{7}{n^2} \leq 3 + \frac{7}{n^2} \leq 3 + 7 = 10$$

Since the largest possible value of $7/n^2$ when n is a positive integer is obtained when $n = 1$.

Therefore $c = 10$

2. Using limits:

$$\lim_{n \rightarrow \infty} \frac{3n^2 - 2n + 7}{n^2} = \lim_{n \rightarrow \infty} \left(3 - \frac{2}{n} + \frac{7}{n^2}\right) = 3$$

Since the limit is equal to a constant, then $3n^2 - 2n + 7 \in O(n^2)$.

1 Common functions

function name	order
constant	$\Theta(1)$
logarithmic	$\Theta(\lg n)$
polylogarithmic	$\Theta(\lg^k(n)) \ k \geq 0$
linear	$\Theta(n)$
quadratic	$\Theta(n^2)$
polynomial	$\Theta(n^k) \ k \geq 0$
exponential	$\Theta(b^n) \ b > 1$
factorial	$\Theta(n!)$

This functions are written in ascending order (remember that o , ω , O , Ω , and Θ induce a poset).

We can show that the function in each row is in little o of the function in the next row.

2 Order of growth of common functions

We can show that the function in each row is in little o of the function in the next row, so we have that:

1. $1 \in o(\lg n)$
2. $\lg n \in o(\lg^k n); \ k > 1$
3. $\lg^k n \in o(n); \ k \geq 0$
4. $n \in o(n^2)$
5. $n^2 \in o(n^k); \ k > 2$
6. $n^k \in o(b^n); \ k \geq 0, \ b > 1$
7. $b^n \in o(n!); \ b > 1$

We now use limits to show each one.

1. $1 \in o(\lg n)$

$$\lim_{n \rightarrow \infty} \frac{1}{\lg n} = 0$$

2. $\lg n \in o(\lg^k n); k > 1$

$$\lim_{n \rightarrow \infty} \frac{\lg n}{\lg^k n} = \lim_{n \rightarrow \infty} \frac{1}{\lg^{k-1} n} = 0$$

because $k - 1 > 0$

3. $\lg^k n \in o(n); k \geq 0$

$$\lim_{n \rightarrow \infty} \frac{\lg^k n}{n}$$

L'Hopital 1 time:

$$= \lim_{n \rightarrow \infty} k \lg^{k-1} n \left(\frac{1}{n \ln 2} \right) = \frac{k}{\ln 2} \lim_{n \rightarrow \infty} \frac{\lg^{k-1} n}{n}$$

L'Hopital 2 times:

$$= \frac{k(k-1)}{\ln^2 2} \lim_{n \rightarrow \infty} \frac{\lg^{k-2} n}{n}$$

L'Hopital 3 times:

$$= \frac{k(k-1)(k-2)}{\ln^3 2} \lim_{n \rightarrow \infty} \frac{\lg^{k-3} n}{n}$$

L'Hopital $\lceil k \rceil$ times:

$$= \frac{k(k-1)(k-2)\dots(k-\lceil k \rceil+1)}{\ln^{\lceil k \rceil} 2} \lim_{n \rightarrow \infty} \frac{\lg^{k-\lceil k \rceil} n}{n}$$

Let

$$c = \frac{k(k-1)(k-2)\dots(k-\lceil k \rceil+1)}{\ln^{\lceil k \rceil} 2}$$

CASE 1: k is integer, then $k = \lceil k \rceil$, $k - \lceil k \rceil = 0$, and $k - \lceil k \rceil + 1 = 1$. Therefore $c > 0$ and

$$\lim_{n \rightarrow \infty} \frac{\lg^k n}{n} = c \lim_{n \rightarrow \infty} \frac{1}{n} = 0$$

CASE 2: k is not an integer, then $k - \lceil k \rceil < 0$, $\lceil k \rceil - k > 0$, and $\lceil k \rceil - k + 1 > 0$. Clearly $c > 0$ and

$$\lim_{n \rightarrow \infty} \frac{\lg^k n}{n} = c \lim_{n \rightarrow \infty} \frac{1}{n \lg^{\lceil k \rceil - k} n} = 0$$

4. $n \in o(n^2)$

$$\lim_{n \rightarrow \infty} \frac{n}{n^2} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0$$

5. $n^2 \in o(n^k)$; $k > 2$

$$\lim_{n \rightarrow \infty} \frac{n^2}{n^k} = \lim_{n \rightarrow \infty} \frac{1}{n^{k-2}} = 0$$

because $k - 2 > 0$

6. $n^k \in o(b^n)$; $k \geq 0$, $b > 1$

$$\lim_{n \rightarrow \infty} \frac{n^k}{b^n}$$

L'Hopital 1 time:

$$= \frac{k}{\ln b} \lim_{n \rightarrow \infty} \frac{n^{k-1}}{b^n}$$

L'Hopital 2 times:

$$= \frac{k(k-1)}{\ln^2 b} \lim_{n \rightarrow \infty} \frac{n^{k-2}}{b^n}$$

L'Hopital 3 times:

$$= \frac{k(k-1)(k-2)}{\ln^3 b} \lim_{n \rightarrow \infty} \frac{n^{k-3}}{b^n}$$

L'Hopital $\lceil k \rceil$ times:

$$= \frac{k(k-1)(k-2) \dots (k - \lceil k \rceil + 1)}{\ln^{\lceil k \rceil} b} \lim_{n \rightarrow \infty} \frac{n^{k - \lceil k \rceil}}{b^n}$$

Let

$$c = \frac{k(k-1)(k-2)\dots(k-\lceil k \rceil + 1)}{\ln^{\lceil k \rceil} b}$$

CASE 1: k is integer, then $k = \lceil k \rceil$, $k - \lceil k \rceil = 0$, and $k - \lceil k \rceil + 1 = 1$. Therefore $c > 0$ and

$$\lim_{n \rightarrow \infty} \frac{n^k}{b^n} = c \lim_{n \rightarrow \infty} \frac{1}{b^n} = 0$$

since $b > 1$

CASE 2: k is not an integer, then $k - \lceil k \rceil < 0$, $\lceil k \rceil - k > 0$, and $\lceil k \rceil - k + 1 > 0$. Clearly $c > 0$ and

$$\lim_{n \rightarrow \infty} \frac{n^k}{b^n} = c \lim_{n \rightarrow \infty} \frac{1}{b^n n^{\lceil k \rceil - k}} = 0$$

7. $b^n \in o(n!)$; $b > 1$

It is possible to show in general that this case is true, but it requires the use of the Γ function to approximate the factorial, which leads to the Stirling approximation (see your textbook for the formula). We will show a weaker case:

8. $2^n \in O(n!)$

To show this we need to show that $\exists c > 0$ such that

$$2^n \leq cn!$$

Let $c = 2$, and notice that:

$$cn! = (2)(1)(2)(3)(4)\dots(n) = (2)(2)(3)(4)\dots(n)$$

Since each factor is ≥ 2 and there are n factors,

$$2^n \leq cn!$$

2.1 Generalization of the factorial function

It is possible to generalize the factorial function to all real numbers by using the following integral:

$$\text{fact}(n) = \int_0^\infty x^n e^{-x} dx$$

We can easily show that this function is equal to $n!$ where $n \in \mathbb{N}$. This can be done by induction if we prove the following two conditions:

- $\text{fact}(n) = n \text{ fact}(n - 1)$
- $\text{fact}(0) = 1$

To show $\text{fact}(n) = n \text{ fact}(n - 1)$ we can try to solve the integral by parts: $u = x^n$, $du = nx^{n-1}dx$, $dv = e^{-x}$, $v = -e^{-x}$. So we have:

$$\begin{aligned} \text{fact}(n) &= -x^n e^{-x} \Big|_0^\infty + \int_0^\infty e^{-x} nx^{n-1} dx \\ &= n \int_0^\infty x^{n-1} e^{-x} dx \end{aligned}$$

Since $x^n e^{-x} = \frac{x^n}{e^x} = 0$ when $x \rightarrow \infty$ because e^x grows faster than x^n (you can prove it using limits too!). And since

$$\int_0^\infty x^{n-1} e^{-x} dx = \text{fact}(n - 1)$$

We get the desired result: $\text{fact}(n) = n \text{fact}(n - 1)$

Now we only have to show that $\text{fact}(0) = 1$. Using the definition of fact we have that:

$$\begin{aligned} \text{fact}(0) &= \int_0^\infty x^0 e^x dx \\ &= -e^x \Big|_0^\infty \\ &= 0 + 1 = 1 \end{aligned}$$

The function that we discussed is called the Gamma Function: $\Gamma(z) = \text{fact}(z - 1)$. With a variable change $x = ny$ we can obtain

an integral for which a closed-form approximation can be given by using Laplace's method, and we get:

$$n! \simeq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

This formula is called the *Stirling Approximation* which can be used to show the growth rate of $n!$ with respect to n^n