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 dully 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

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

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

$$\vdots$$

induction.

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

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

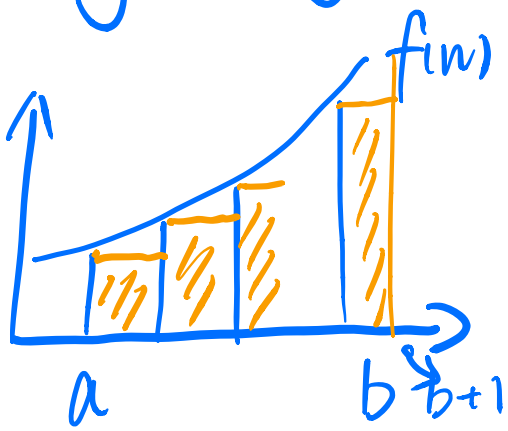
$$aS = a + a^2 + a^3 + \cdots a^{n+1}$$

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

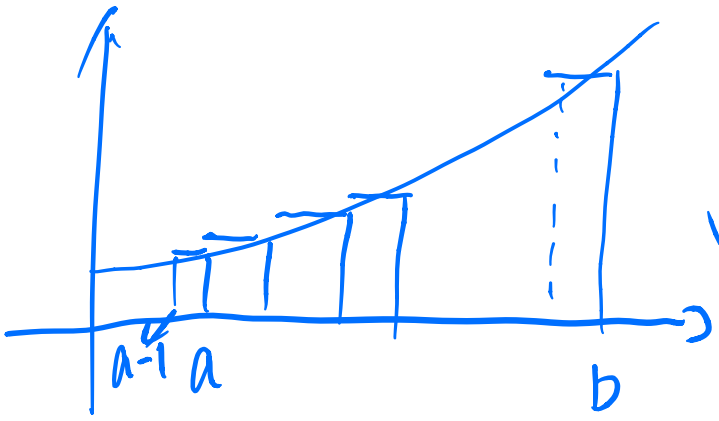
$$S(a-1) = a^{n+1} - 1$$

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

Using integrals.



$$\sum_{i=a}^{b} f(i) \le \int_{a}^{b+1} f(x)\,dx$$



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

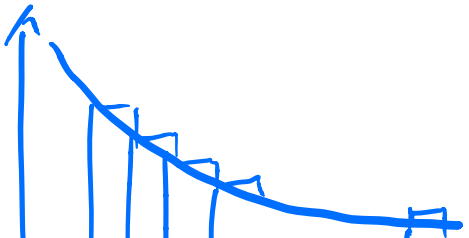
Example. find an upper bound.

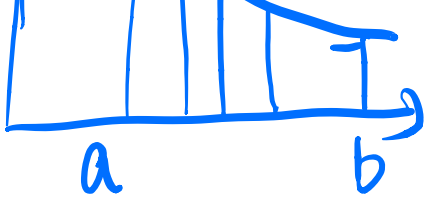
$$\text{for } \sum_{i=1}^{n} \frac{1}{i} \le \int_{1}^{n+1} \frac{dx}{x} = \ln x \Big|_{1}^{n+1} = \ln(n+1) \in \theta(\lg n)$$

(Harmonic).

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

Decreasing Function.





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

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

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

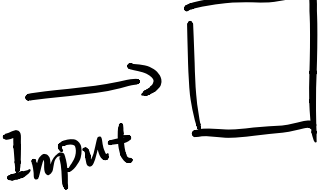
$$= 1 + \ln n \in O(\lg n)$$

Techiques

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

Problem:

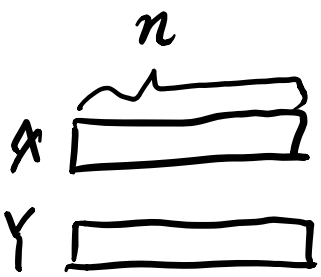
1) divide input into smaller pieces.
2) Reawsively solve the smaller pieces
3) combine into solution



Example.

Binary Operations.

- Addition

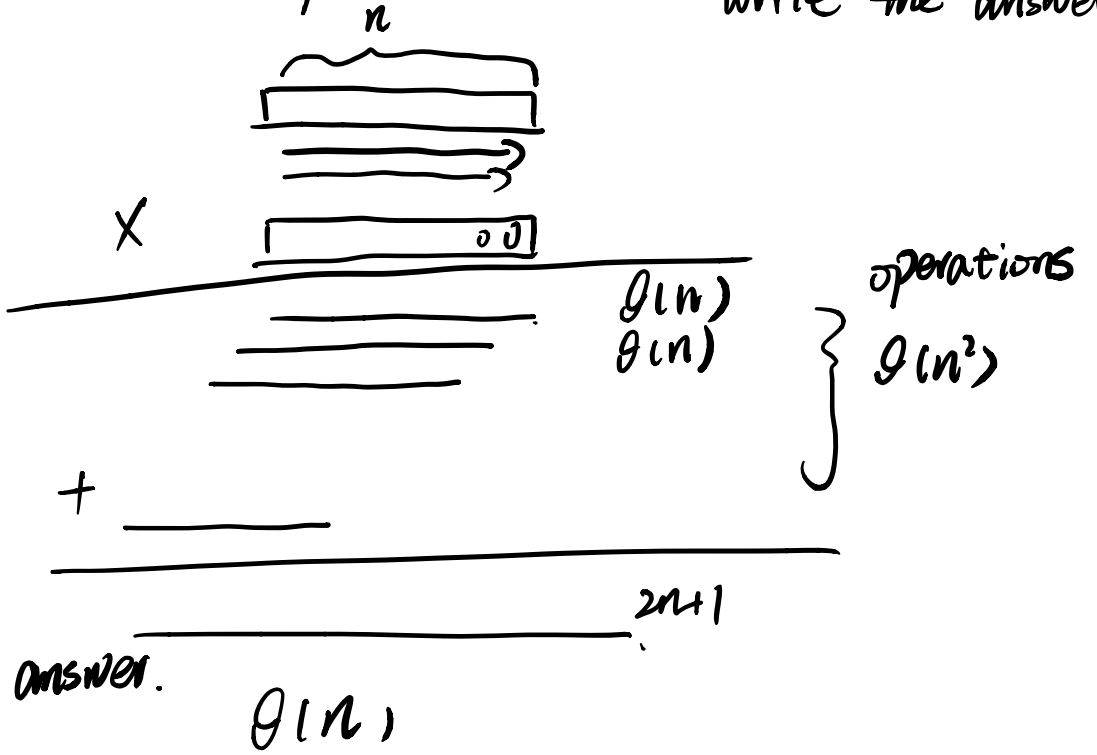


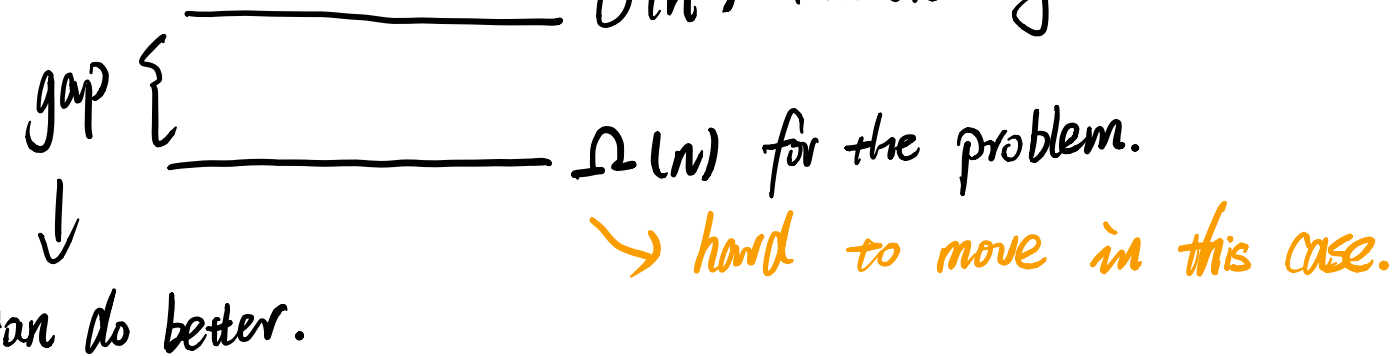
bin num.

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

every Alg. must write the answer.

- multiplication



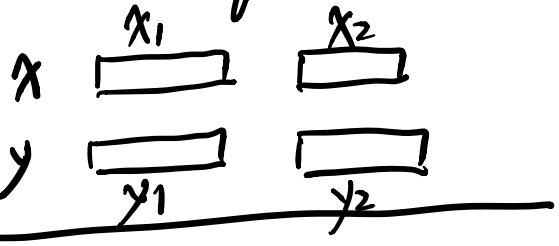


improvement.

1) Better Alg. faster.

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

Divide and conquer.



n is power of 2.

⊗ add 0 on left.

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

$$y = 2^{n/2} y_1 + y_2$$

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

↑ shift $\theta(n)$ ↑ $\theta(n)$.

Numbers of operations

$T(n) = 4T(\frac{n}{2}) + \theta(n)$ → recurrence relation.

Iteration:

$4T(\frac{n}{2}) + cn.$

① $T(n) = 4(4T(\frac{n}{2^2}) + \frac{cn}{2}) + cn$

$= 4^2 T(\frac{n}{2^2}) + \frac{4}{2} cn + cn$

② $= 4^2(4T(\frac{n}{2^3}) + c\frac{n}{2^2}) + \frac{4}{2} cn + cn$

$= 4^3 T(\frac{n}{2^3}) + (\frac{4}{2})^2 cn + (\frac{4}{2}) cn + cn$

repeat k times:

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

$T(1) = 1$

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

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

$C \rightarrow$ Constant.

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

no improvement!

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

$$= n^{\lg 4}$$

$$= n^2$$

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

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

$$P_1 = x_1 y_1$$

$$P_2 = x_2 y_2$$

$$P_3 = (x_1 + x_2)(y_1 + y_2) - P_1 - P_2$$

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

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

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

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

$O(n^2)$

$O(n^{1.59})$

$\Omega(n)$

× fast for human

✓ fast for computer.

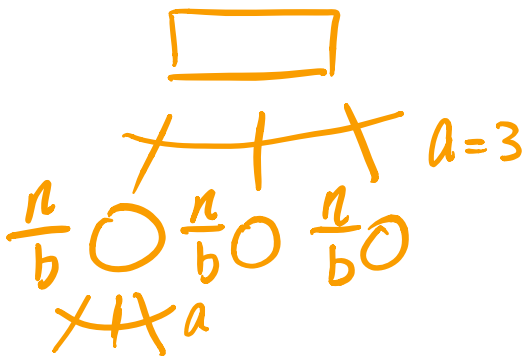
In general:

$$T(n) = \underset{(3)}{a}T\left(\frac{n}{\underset{(2)}{b}}\right) + f(n).$$

Master theorem:

compare

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



a is the times have to recursively operate.

$T(n) = aT(\frac{n}{b}) + f(n)$ $\quad f(n) > 0$ $\quad T(n) = 3T(\frac{n}{2}) + \underbrace{\Theta(n)}_{cn.}$

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

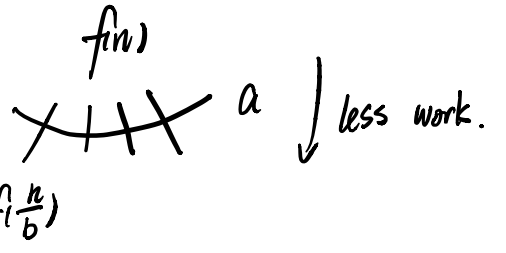
upper bound. (1) $f(n) \in O(n^{\log_b a - \varepsilon}) \Rightarrow T_n \in \Theta(n^{\log_b a})$

(2) $f(n) \in \Theta(n^{\log_b a}) \Rightarrow T_n \in \Theta(n^{\log_b a} \lg n^0)$

lower bound. (3) $f(n) \in \Omega(n^{\log_b a + \varepsilon}) \Rightarrow T(n) \in \Theta(f(n))$

and $af(\frac{n}{b}) \le cf(n)$ $\quad 0 < c < 1$

Regularity condition



Proof:

Iteration: $T(1) = 1$ (only affect constant, anything works).

1. $T(n) = a(aT(\frac{n}{b^2}) + f(\frac{n}{b})) + f(n)$
   $= a^2 T(\frac{n}{b^2}) + af(\frac{n}{b}) + f(n)$
2. $T(n) = a^2(aT(\frac{n}{b^3}) + f(\frac{n}{b^2})) + af(\frac{n}{b}) + f(n)$
   $= a^3 T(\frac{n}{b^3}) + a^2 f(\frac{n}{b^2}) + af(\frac{n}{b}) + f(n)$

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

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

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

CASE 1:

$f(n) \le C n^{\log_b a - \varepsilon}$; $C > 0$

$\sum_{i=0}^{k-1} a^i f(\frac{n}{b^i}) \le C \sum_{i=0}^{k-1} a^i (\frac{n}{b^i})^{\log_b a - \varepsilon}$

$= Cn^{\log_b a - \varepsilon} \sum_{i=0}^{k-1} (\frac{a}{b^{\log_b a - \varepsilon}})^i$

$= Cn^{\log_b a - \varepsilon} \sum_{i=0}^{k-1} (b^\varepsilon)^i$

$b^{\log_b a - \varepsilon} = b^{\log_b a} b^{-\varepsilon}$

$= a b^{-\varepsilon}$

$$= cn^{\log_b a - \varepsilon} \frac{b^{\varepsilon \log_b n} - 1}{b^\varepsilon - 1}$$

$$= cn^{\log_b a - \varepsilon} \left(\frac{n^\varepsilon - 1}{b^\varepsilon - 1} \right)$$

$$= \frac{c}{b^\varepsilon - 1} n^{\log_b a} - \frac{c}{b^\varepsilon - 1} n^{\log_b a - \varepsilon}$$

constant

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

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

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) \in \Theta(n^{\log_b a})$$

CASE 2.

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

$$c_1 n^{\log_b a} \le f(n) \le c_2 n^{\log_b a}$$

$$\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) \le c_2 \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^{\log_b a}$$

$$= c_2 n^{\log_b a} \sum_{i=0}^{k-1} \left(\frac{a}{b^{\log_b a}}\right)^i = c_2 n^{\log_b a} \sum_{i=0}^{k-1} 1 = c_2 n^{\log_b a} k$$

$k \downarrow \log_b n$

$$= c_2 n^{\log_b a} \frac{\lg n}{\lg b} = \frac{c_2}{\lg b} n^{\log_b a} \lg n$$

constant

$$d_1 n^{\log_b a} \lg n \le \Sigma \le d_2 n^{\log_b a} \lg n$$

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

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

CASE 3.

$f(n) \in \Omega(n^{\log_b a + \varepsilon})$ and $af\left(\frac{n}{b}\right) \le cf(n)$, $0 < c < 1$.

iterate. $a^i f\left(\frac{n}{b^i}\right) \le c^i f(n)$

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

$$T(n) \le \_\_\_ f(n)$$

$$\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) \le \sum_{i=0}^{k-1} c^i f(n)$$

$$= f(n) \sum_{i=0}^{k-1} c^i \quad 0 < c < 1$$

$$S = 1 + c + c^2 + \cdots$$

$$cS = \quad c + c^2 + c^3 + \cdots$$

$$S - cS = 1 \qquad S = \frac{1}{1-c}$$

$$\sum_{i=0}^{\infty} c^i = \frac{1}{1-c} \swarrow$$

Constant.

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

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

$$\therefore T(n) \leq cf(n)$$

~~the other side.~~

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

Big-O notation

positive function $f(n)$ $g(n)$

$c > 0$ $n > 0$

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

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

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

L Hopital Rule

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)} =$$

- ∞ $f(n) \neq O(g(n))$, $g(n) = O(f(n))$
- 0 $f(n) = O(g(n))$, $g(n) \neq O(f(n))$
- $c > 0$. constant. $\begin{cases} f(n) = O(g(n)) \\ g(n) = O(f(n)) \end{cases}$ $f(n) = \Theta(g(n))$

Divide & conquer Alg.

$S(n)$

↓

$S(n_1) \cdot S(n_2) \cdots S(n_k)$ solve recursively

Combine solutions.

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

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

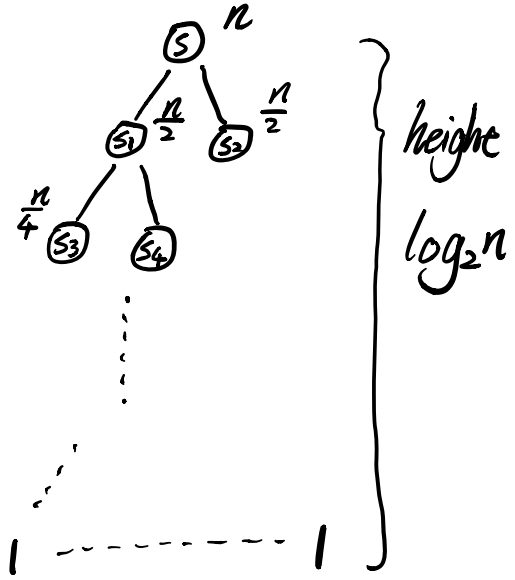
if $n = |S| \le 1$, return S.

devide S into subsets. S_1. S_2 of equal site

scan S_1 to find $\min(S_1)$

scan S_2 to find $\min(S_2)$

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



$$T(n) = T(\tfrac{n}{2}) + T(\tfrac{n}{2}) + a$$

$$= 2T(\tfrac{n}{2}) + a = \boxed{?}$$

Master Theorem

Examples

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

$f(n) = n^2 \qquad n^{\log_b a} = n^{\log_2 3} = n^{1.59\ldots}$

$f(n) \in \Omega(n^{1.59+\varepsilon})$ [CASE 3]

↳ Regularity condition

$af(\frac{n}{b}) \le cf(n)$ $\qquad$ $3(\frac{n}{2})^2 \le cn^2$

$0 < c < 1$ $\qquad$ $\frac{3}{4} \le c$

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

regularity condition is satisfied.

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

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

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

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

→ RC: $2^{\frac{n}{2}} \le c\,2^n$

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

$\boxed{\frac{1}{2^{\frac{n}{2}}}} \le c$ → largest → $n=1$ $\quad \frac{1}{2^{\frac{1}{2}}} \le c$

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

$c = 0.7071 < 1$ RC satisfied.

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

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

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

$f(n) = n\lg n \in \Omega(n^{1+\varepsilon})$ → have to have a ε satisfy this situation

n^2 lower bound for $\lg n$ ×

Not case 3 or 1&2

cannot use master theorem.

4) $T(n) = 64T(\frac{n}{2}) - n^2 \lg n$

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

Sorting

list $A[1], A[2], A[3] \cdots A[n]$ ← Input

element

compare $A[i] \overset{\leq}{\geq} A[j]$

$B[1], B[2], B[3] \cdots B[n]$ ← permutation output

Silly sort

for each permutation verify $b[i] \leq b[j] \leq \cdots \leq b[n]$

$\theta(n)$ ↑

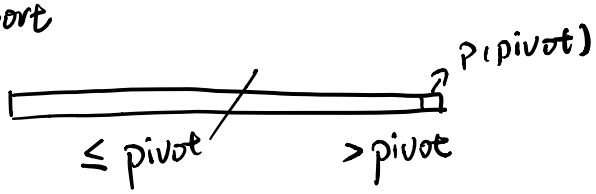
if so return b.

count operations. (comparisons)

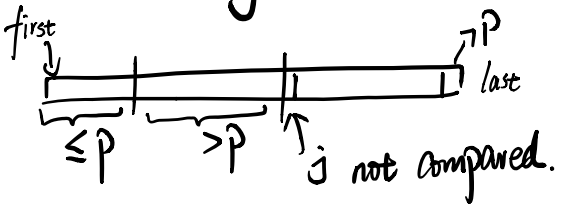
↳ $T(n) = n! \theta(n)$.

Divide & conquer	① Merge sort	Quick Sort
Divede	Easily	Carefully (partition)
Recursively solve	Carefully (merge)	
combine.		Easily

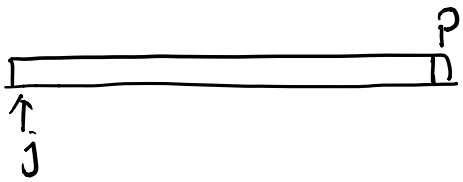
Quick Sort



while executing



initial

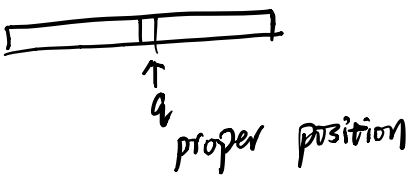


n

```
partition (a, first, last)
    p = a[last]
    i = first - 1
    for j = first to last - 1
        if (a[j] ≤ p)
            i++
            swap a[i] with a[j]
    i++
    swap a[i] with a[last]
    return i.
```

O(n)

```
QuickSort (a, first, last)
    if first < last
        q = partition (a, first, last)
        QuickSort (a, first, q-1)
        QuickSort (a, q+1, last)
```



Analysis: $T(n) = \theta(n) + T(q) + T(n-q)$ $\quad 1 \le q \le n$

Comparision based sorting

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

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

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

$f(n) \in \theta(n)$

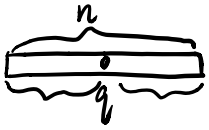
$T(n) \in \theta(n \lg n)$

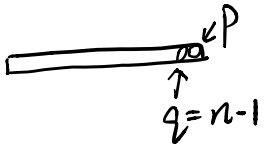
Quick Sort

Best $\Theta(n \lg n)$

Worst $\Theta(n^2)$

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



Worst case:  (p is the biggest).

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

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

Iteration: $T(n) = T(n-2) + c(n-1) + cn$

$$= T(n-3) + c(n-2) + c(n-1) + cn$$

$$\vdots$$

$$= T(\underbrace{n-k}) + [cn + c(n-1) + c(n-2) + \cdots + c(n-k+1)]$$

$n-k=1$

$k=n-1$

$$= T(1) + c(\underbrace{2+3+4+\cdots+n}_{\frac{n(n+1)}{2}-1}) \in \Theta(n^2)$$

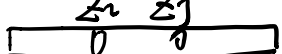
Average Case

$$\sum_{\text{All cases}} \text{Comparisions} \cdot \text{probability}$$

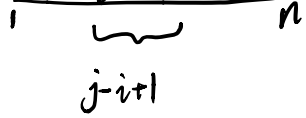
Randomized Partition (a, first, last)

- r = random integer $\in$ [first, last]
- swap a[r] with a[last]
- return partition (a, first, last)

Sorted list:



first time the algorithm picks a pivot in the interval.



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

0 comparisions

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 $n=4$

i

1 — 2, 3, 4

2 — 3, 4

3 — 4

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \quad \text{if } j-i=k$$

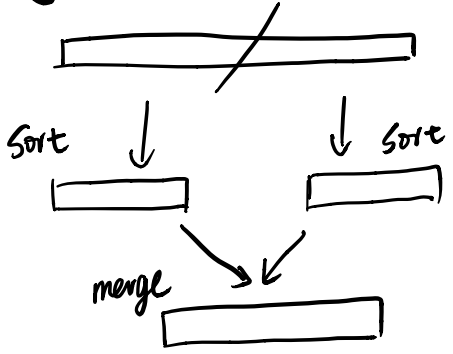
$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

Simplify to compute.

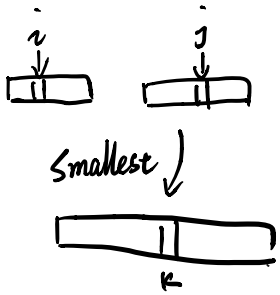
$$\leq 2 \sum_{i=1}^{n-1} \lg n = 2(n-1)\lg n \in O(n \lg n)$$

Average Case.

Merge Sort:



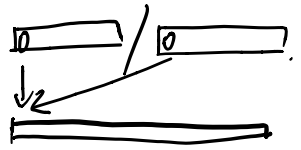

```
Merge(a, first, mid, last)
  i = first
  j = mid+1
  for k = first to last
      if (i ≤ mid and j ≤ last)
          if a[i] < a[j]
```




```
MergeSort (a, first, last)
  if first < last
    mid = ⌊(first + last)/2⌋
    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 LRLRLR.

best case = $n/2$ Left < Right

$\theta(n)$

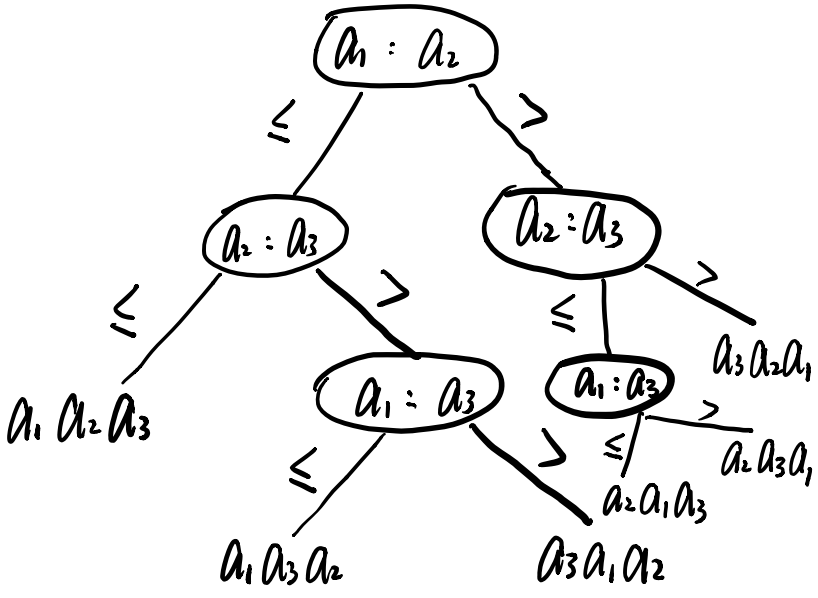
$T(n) = 2T(\frac{n}{2}) + \theta(n)$ ——— upper bound. $O(n \lg n)$

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

$T(n) \in \theta(n \lg n)$. ——— lower bound $\Omega(\ )$

Comparison Decision Tree

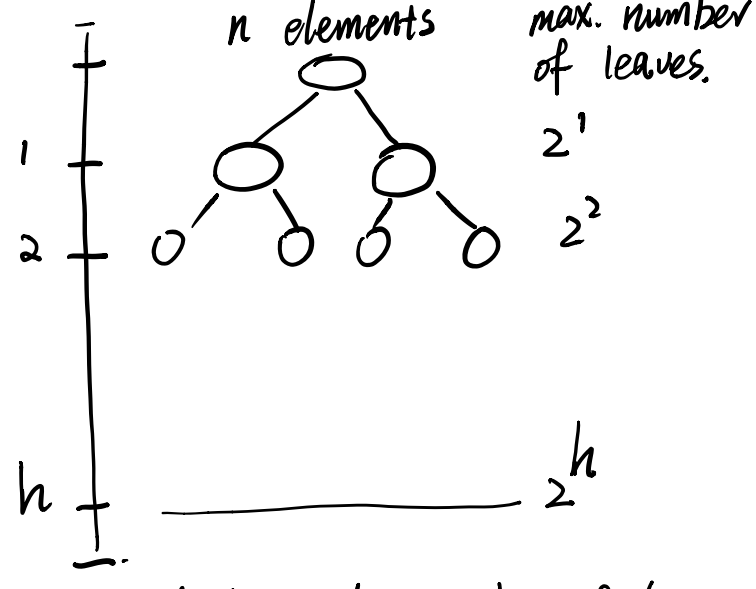
Example : a_1, a_2, a_3



any comparison-based sorting algorithm must do at least n comparisons. $T(n)$

each permutation is a leaf.

permutation = $3! = 6$



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

$l = n!$

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

$n! \simeq \sqrt{2\pi n}\,(\frac{n}{e})^n$ ← stiring Approx

if l is the number of leaves

$l \le 2^h$

$\lg l \le h$

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

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

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

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

Any sorting algorithm $\in \Omega(n\lg n)$

comparison based.

⇑ worst case

Worst case — mergesort

$\Omega(n\lg n)$. $O(n\lg n)$ ∈ mergesort is optimal.

Selection Problem

find the k-th smallest element

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

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

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

↓

Sort a
return a[k] $\Theta(n\lg n)$.

lower bound

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

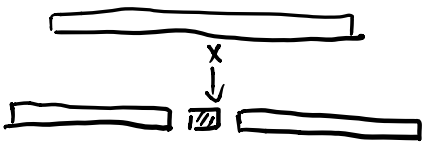
gap $O(n\lg n)$

$\Omega(n)$

Divide & Conquer

$(a_1, a_2, \dots a_n)$

k.



select (a, first, last, k)

x = partition (a, first, 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)

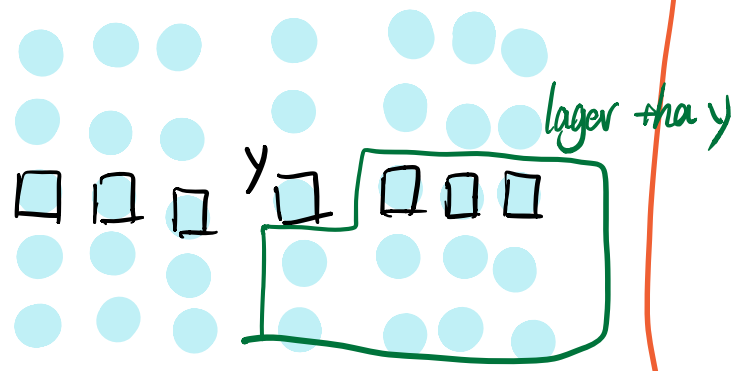
— use randomized partition

average case $\theta(n)$

worst case $\theta(n^2)$

Finding a good pivot

group of 5



find median

why 5? not 3?

— find the median of each group of 5 (constant time).

— find the median of the medians (recursively) ↓ y

— partition using a[y] as pivot. ↓ x

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

large $\leq \frac{7}{10} n$

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

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

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

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

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

$\Rightarrow C m (\log n - \log 2) + a \cdot m \le C \cdot m \log n$ ($\log 2 \hookrightarrow 1$)

$\Rightarrow C m \log n - C m + a m \le C m \log n$

$\Rightarrow -C m + a m \le 0$

$\Rightarrow a \le C$

$O(n)$

$O(\log n)$ most common.

$O(n \log n)$ most common.

$O(n^2)$

$O(\sqrt{n})$ rare.

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

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

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

$\begin{cases} a_0 = 1 \\ a_1 = 2 \end{cases}$

$G(x) = \sum_{n=0}^{\infty} a_n x^n$ $\quad G(x) = \sum_{n=2}^{\infty} a_n x^n + a_1 x + a_0$

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

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

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

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

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

$s_1 \quad s_2 \quad s_3$

if > 2 : recursively

else : terminate.

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

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 $M[1 \cdots (j-1), \cdots k]$ $M[(j+1) \cdots n, k \cdots m]$

Pseudo code

```
findLeftMostMinVal(M)
    if n == 1:
        linear search for min value x.
        return x
    else:
        j = n/2
        linear search for row j of M, which is x(j,k).
        M1 = M[1 ··· (j-1), 1 ··· k]
        M2 = M[(j+1) ··· n, k ··· m]
        I1 = findLeftMostMinVal(M1)
        I2 = findLeftMostMinVal(M2)
    return x, I1, I2
```

$2T(\frac{n}{2}) + n$.

$O(m \log n)$

$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + an, a>0$

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

Definition $T(n) \le cn, c>0$

$\Rightarrow T(\frac{n}{5}) \le c\frac{n}{5}$

$T(\frac{7}{10}n) \le \frac{7}{10}cn$

$\frac{cn}{5} + \frac{7}{10}cn + an \le cn$

need to show.

$\Downarrow$

$2cn + 7cn + 10an \le 10cn$

$10a \le c$

$\therefore c \ge 10a > 0$

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

$T(n) = T(\frac{n}{5}) + T(\frac{7}{10}n) + an$ → already linear time

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

good partition

$\frac{1}{4}$ $\frac{3}{4}$

easy to prove linear

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

E: times to have a good result of flip coin.

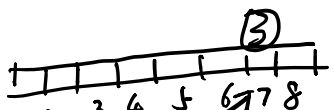
flip once coin

Sorting without comparison of elements.

Array = 2 5 3 1 2 3 1 3

Counter:

1	II	② — 2
2	II	② — 4
3	III	③ — 7
4		⓪ — 7
5	I	① — 8



Counting sort.

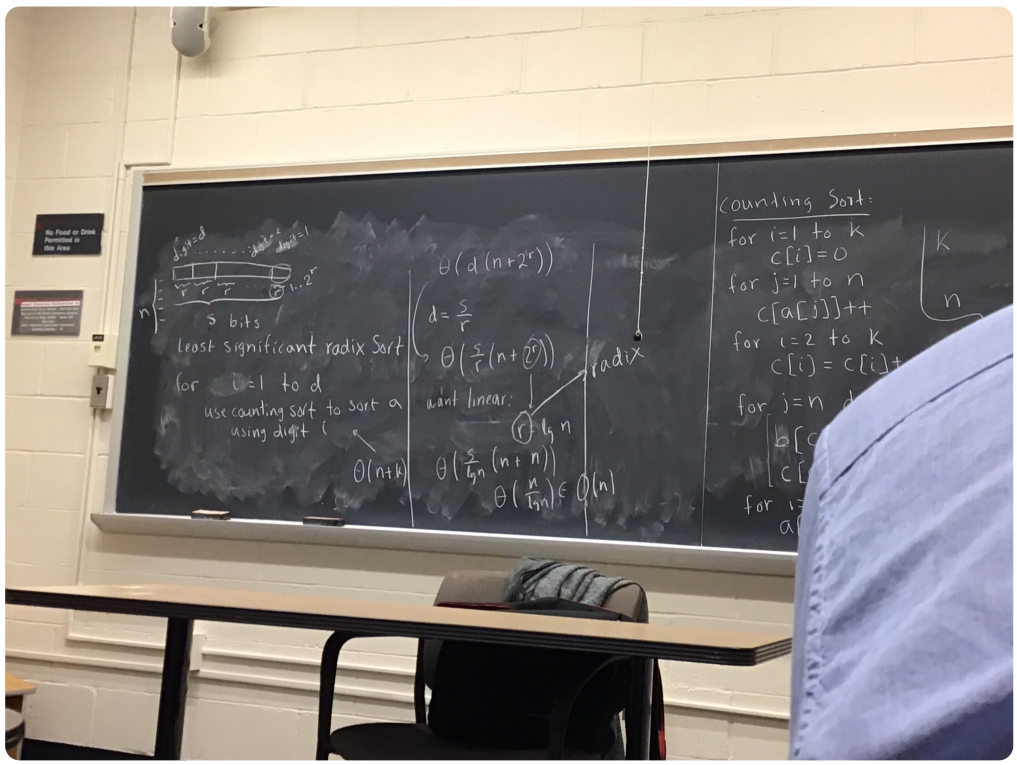
```
for i=1 to k
    C[i] = 0                k
for j=1 to n
    c[a[j]] ++              n
for i=2 to k
```

Time complex (count assignment).

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

Sorts in linear time

```
    C[i] = C[i] + C[i-1]        k-1
for j=n downto 1
    b[c[a[j]]] = a[j]           n
     C[a[j]] --
for i=1 to n                    n
    a[i] = b[i]
```



Radix sort

Divide & Conquer

Matrix Multiplication

$X_{(n\times n)}$ $Y_{(n\times n)}$ = (□ → $\theta(n)$)

need to compute n^2

$\theta(n^3)$

lower bound:

At least write the produce

—— $O(n^3)$

—— $\Omega(n^2)$

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

where $A \sim H$ are matries

$$XY = \begin{pmatrix} A\bar{E}+BG & A\bar{F}+BH \\ C\bar{E}+DG & C\bar{F}+DH \end{pmatrix}$$

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

Strassen's Algorithm.

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

$P_2 = (A+B)H$

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

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

$P_5 = (A+D)(\bar{E}+H)$

$P_6 = (B-D)(G+H)$

$P_7 = (A-C)(\bar{E}+\bar{F})$

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

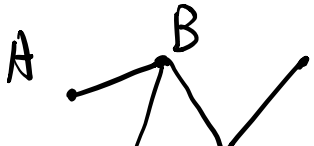
$\rightarrow T(n) = 7T(\frac{n}{2}) + \theta(n^2)$

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

Graph Algorithm.

Graph: $G = (V, E)$

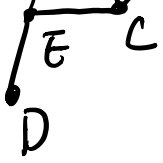
undirected graph



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

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

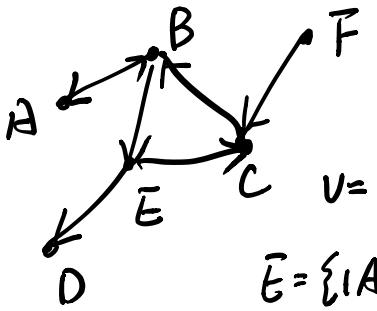
Some definitions.

- Adjecent 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) $m = \frac{n(n-1)}{2}$



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

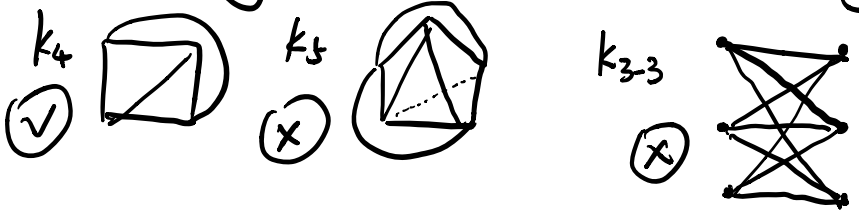
$E = \{(A,B)(B,A)(BE)(EC)(FC)(ED)(CB)\}$

K_n — complete graph with n vertices

K_3 △ K_4 ⊠ K_5

Embedding on a plane (without crossing edges)

K_4 ✓ K_5 ✗ $K_{3,3}$ ✗



← planarity problem.

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

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

Simple graphs: not allowed: ○ ⬭

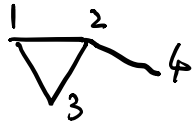
(in this course)

(multigraphs).

Data Structures.

Simplest : list of edges × searching $\Theta(m^2)$

Adjacency Matrix :



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

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

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

↑ Symmetic : undirected graphs.

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

Adjacency list :

(undirected) 1 — 2. 3

every edge 2 — 1. 3. 4

appears 3 — 1. 2

2 times. 4 — 2

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

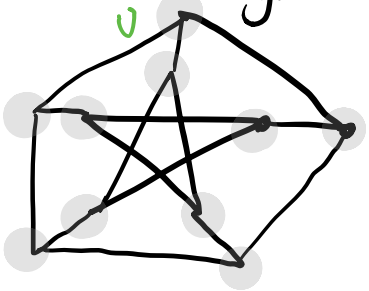
$\Theta(n)$

Θ(1 max adjacent vertices).

degree

In a directed graph, each edge appears once.

Bounded degree = 3



petersen Graph

↑

Explore a graph.

Explore (v)

```
    v.marked = True
    for each (v,u) ∈ E
        if not u.marked
            Explore(u)
```

Connected Graph.

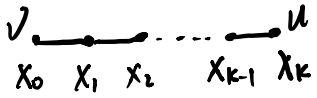
$\forall\ u, v \in V$, there is a path from u to v.

path from u to v

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

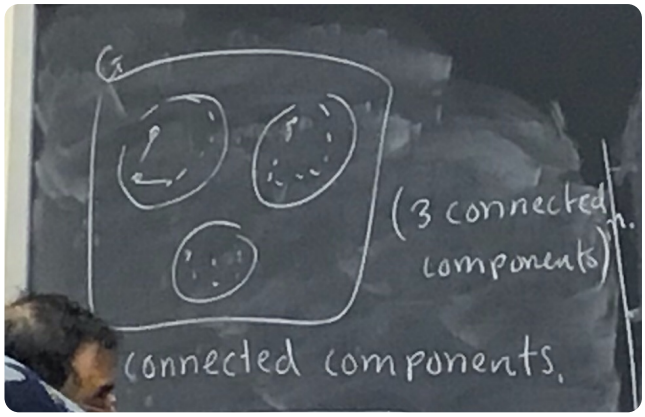
s.t $(x_{i-1}, x_i) \in E$ Length (number of edges) : k.

$\forall i = 1 \cdots k$



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

```
Connected(G)
    Let v ∈ V
    Explore(v)
    for each u ∈ V
        if u.marked = false
            return false
    return true
```



Algorithm to determine the number

of connected components.

Connected components.

```
Explore(v)
  v.marked = true
  previsit(v)
  for each(v, u)
      if not u.marked
          Explore(u)
  postvisit(v)
```

```
connectedComponent(G)
  comp = 0
  for each v ∈ V
      if not v.marked
          comp++
          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 $O(n^2)$

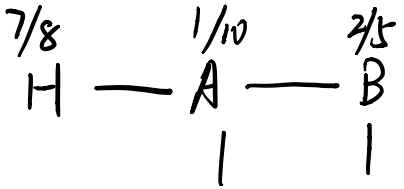
Depth-First Search (DFS)

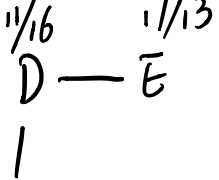
Adj list processed in lexicographical order.

```
previsit(v)
  count++
  v.pre = count
```

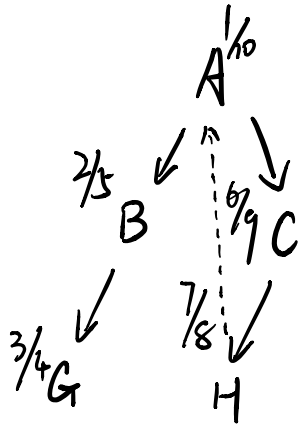
```
postvisit(v)
  count++
  v.post = count
```

```
DFS(G)
  count = 0
  for each v ∈ V
      if not v.marked()
          Explore(v)
```





C 6/9 G 3/4 F 14/15



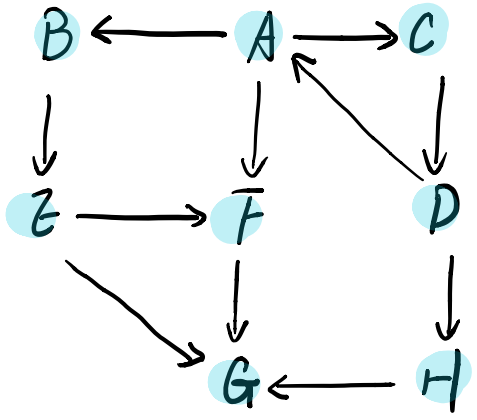
Tree edge (↓) Cross edge (⇝)

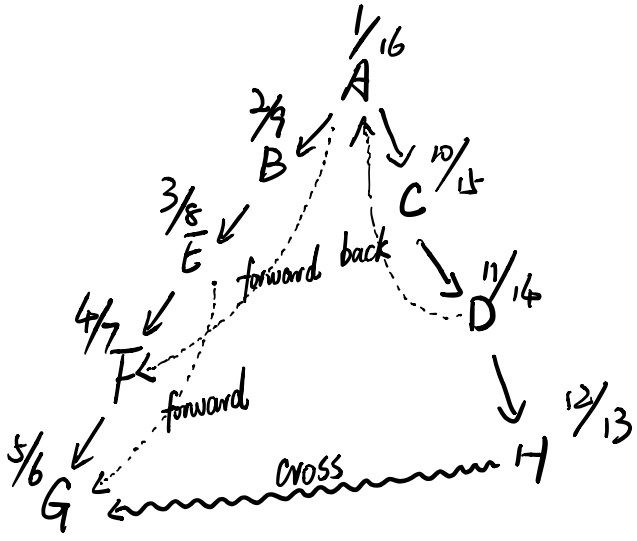
Back edge (↑) / forward edge (↓)

G_T has a cycle ⇔ DFS finds a back edge

↙

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





$u \to v$

$u.pre < v.pre < v.post < u.post$

[[ ]] tree/forward [[]] (u v v u)

back [[]] (v u u v)

cross [ ] [ ] (v v u u)

G_T has a cycle ⇐⇒ DFS find a back edge

Proof : G_T has a cycle (not repeating vertices)

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

x_0 is first vertex in C seen by DFS.

⇒ every vertex in the cycle is reachable from x_0

⇒ down in the recursion.

X_{k-1} ⇒ edge (X_{k-1}, X_0) is seen as a back edge.

($\Leftarrow$): If a back edge

path $X_i \dashrightarrow X_j \rightarrow X_i$

tree edge back edge

∴ G has a cycle.

If back edge ⇒ cycles

if no cycles ⇒ no back edges

Directed Acycle Graph (DAG) ← no back edge.

$\forall\ u \longrightarrow v$ $v.post < u.post$

If a graph represents tasks (precedence)

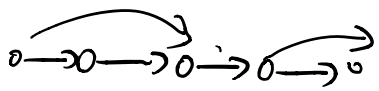
we can find a linear order of the task

satisfying the precedence.

↗

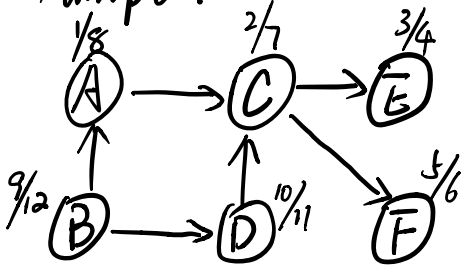
Topological Sorting

(Linearization).



Sort tasks in reverse post-visit number order.

Example.



BADCEF linearization

DAG $u \to v$

post u > post v

linearization: B D A C F E

(post) 12 11 8 7 6 4

DS:

Priority Queues

sorted list

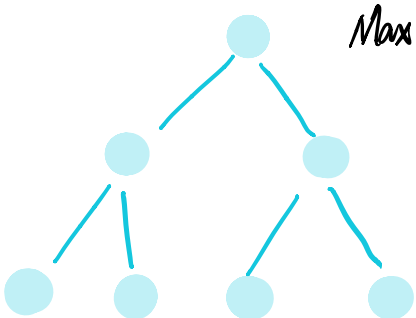
find position $\theta(\lg n)$

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)



Max-heap

binary heap

conditions for heap:

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

Heap property:

value of a node is ≥ the value of its children

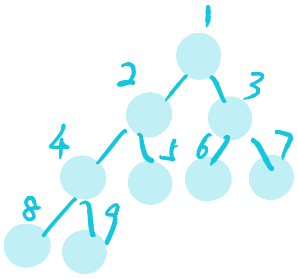
root has the max value.

heap: use an array

left(i) = 2i

right(i) = 2i+1

parent(i) = ⌊i/2⌋



ExtractMax()

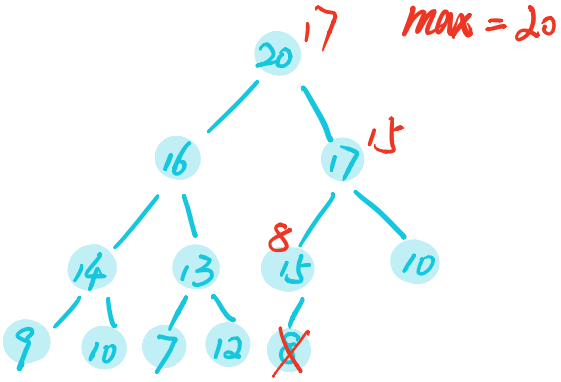
max = a[1]

a[1] = a[n]

n-- → O(lgn)

SiftDown(1)

return max



SiftDown(i)

if(2i ≤ n)

might not exist.

let j = maximum index of (a[i] . a[2i] . a[2i+1])

if(i ≠ j)

swap a[i] with a[j]

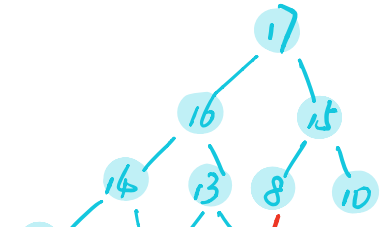
SiftDown(j)

max levels of heap: lgn → O(lgn) worst case.

insert(x)

n++

a[n] = x



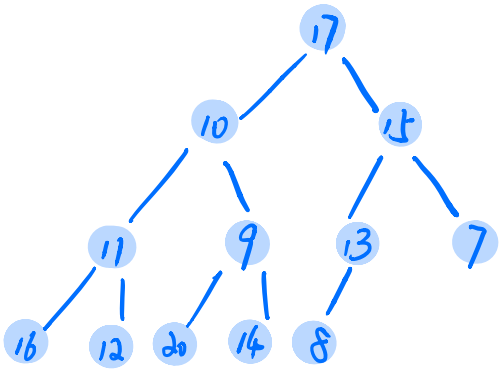
Bubble Up (n) → O(lgn)

Bubble Up(i)

```
if i > 1
    Let j = maxmum index of (a[i], a[parent(i)])
    if (i=j)
        swap a[i] with a[parent(i)]
        Bubble Up (parent(i))
```

Bubble Up(i)

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



Build Queue (a).

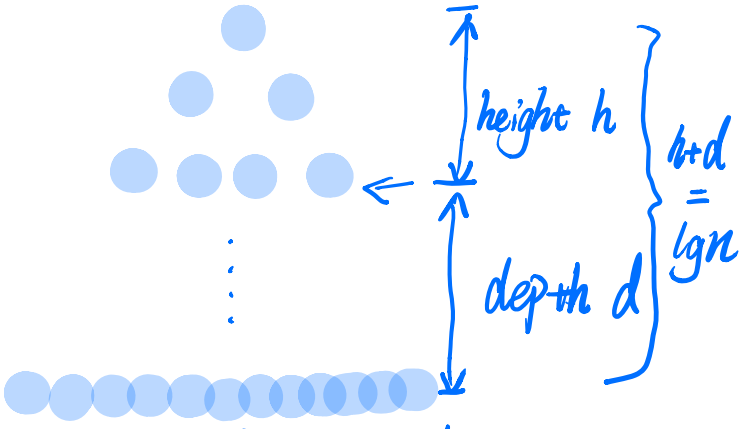
for $i = \lfloor \frac{n}{2} \rfloor$ downto 1

Sift Down (i)

Analysis.

Priority Queue

```
buildQueue(array a)
    for i = ⌊n/2⌋ down to 1
        siftDown(i)
```



worst case: $W(n) = \sum_{d=0}^{\lg n - 1} n\frac{d}{2^d} = n\sum_{d=0}^{\lg n - 1}\frac{d}{2^d}$

we don't compute last level

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

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

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

$S = 2$

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

* number of nodes 2^h (height h).
* number of operations for a node at depth d $= d$
* number of nodes at depth of d

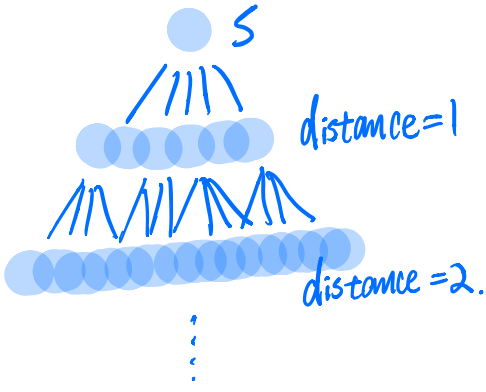
$2^h = 2^{\lg n - d} = 2^{\lg n}2^{-d}$

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

* 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 ⇕ v is in the queue).



$v.dist$ = distance from s to v

$v.pred$ = keeps predecessor on the tree

```
BFS(graph G)
    for each v ∈ V
        v.dist = ∞          } O(|V|)
    s.dist = 0
    Q = new Queue()
    Q.enqueue(s)
    while Q.isEmpty() == false
        u = Q.dequeue()
        for each v ∈ u.adj
            if v.dist == ∞
                v.dist = u.dist + 1
                v.pred = u
                Q.enqueue(v)
```

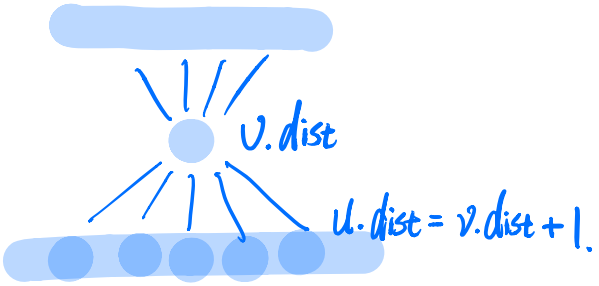
Queue:

- isEmpty() constant time
- enqueue(x)
- dequeue()

```
    v.pred = null
s.dist = 0
Q.enqueue(s)
while not Q.isEmpty()
    v = Q.dequeue()
    for each (v,u) ∈ E
        if u.dist = ∞
            u.dist = v.dist + 1
            u.pred = v
            Q.enqueue(u)
```

$O(|E|)$

- dequeue()



↑ Analysis.

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

Weighted graph.

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

Assume a shortest path from s to v

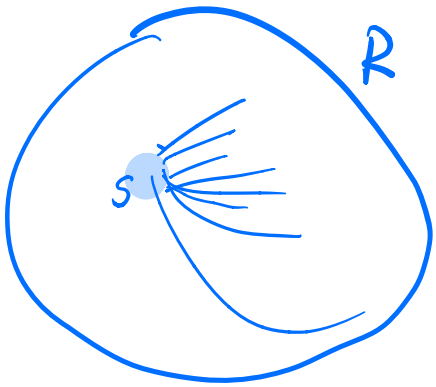
$s \rightsquigarrow x \to v$

$d(s,x) < d(s,x) + w(x,v) = d(s,v)$ $d(s,v)$: distance (shortest path) from s to v.

$\Rightarrow d(s,x) < d(s,v)$

x must be in R.

otherwise x is closer from s than v contradicts p.



satisfy

At point R is correctly computed all shortest paths from $s \le d$

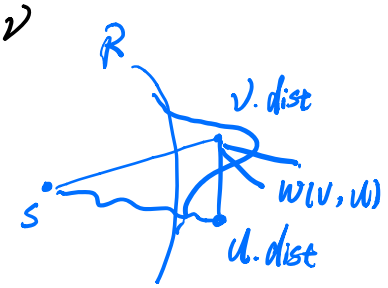
find v that we can add to R.

⇑ 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



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

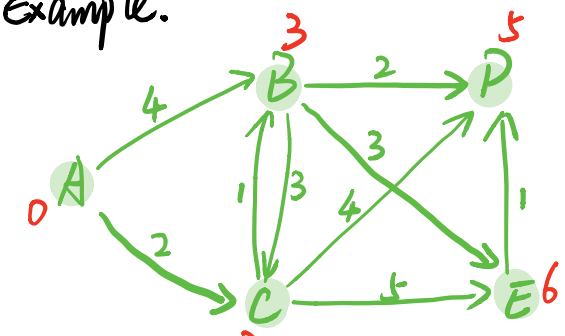
```
Dijkstra (l. G. S)
    for each v ∈ V
        V. dist = ∞
        V. pred = null
    S. dist = 0
    Q. build Queue (V)
    while not Q.isEmpty( )
        v = Q. extract Min( )
        for each (v, u) ∈ E
            if v.dist + w(v, u) < u. dist
                u.dist = v.dist + w(v, u)
                u. pred = v
                Q. decrease key (u) ← bubble up  lg(|V|)
```

Dijkstras Alg.

```
for each v ∈ V { v.dist = ∞
                 v.prev = null        |V|

s.dist = 0
Q.buildQueue(V)                       |V|
while not Q.isEmpty()                 |V|lg|V|
    v = Q.extractMin()
    for each (v, u) ∈ E
        if v.dist + w(v, u) < u.dist  |E|lg|V|
            u.dist = v.dist + w(v,u)
            u.prev = v
            Q.decreasekey(u)          O((|V|+|E|)lg|V|)
```

Example.



starting from A.

S = A

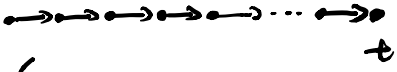
dist/pre

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

no out-way from D. copy others.

```
update(v, u)
if v.dist + w(v, u) < u.dist
    u.dist = v.dist + w(v, u)
```

shortest path



member of edges : $|V|-1$

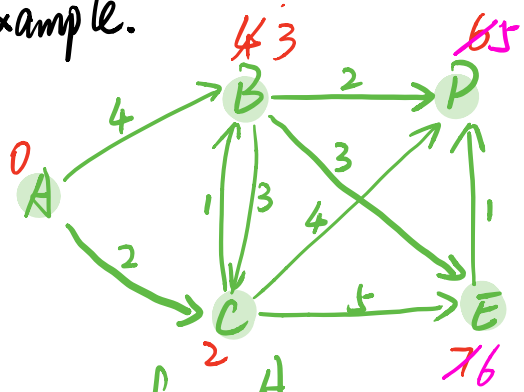
Bellman-Ford:

repeat $|V|-1$ times:

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

update (v, u)

Example.



starting from A.

$S = A$

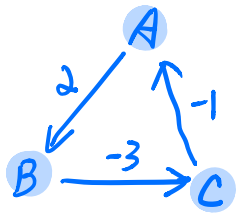
dist/pre

update.

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

Special situation :

solution : check on more time ($|V|$ times) and if value change. then there are negative cycle↑.



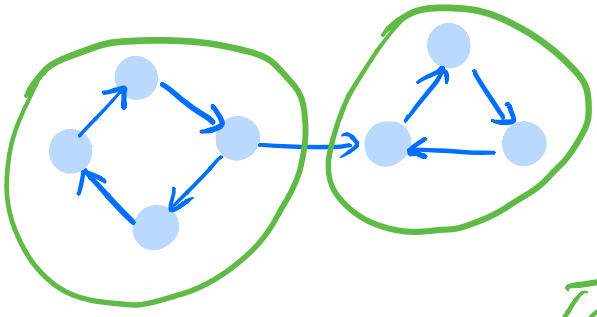
Directed Graph.

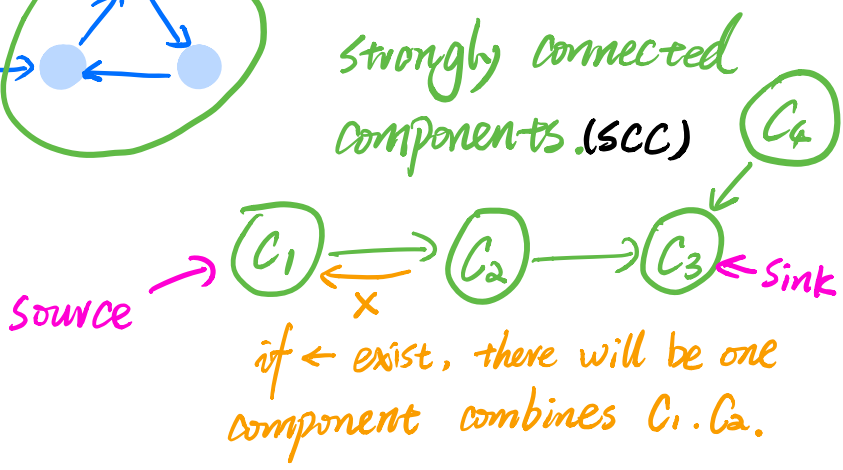
$\forall u, v \in V$. there is a path from u to v and there is a path from v to u.

Strongly connected ↑ u, v are related. if f ↑

It satisfies : Equivalence Relation.

Partitions the set of vertices into equivalence classes

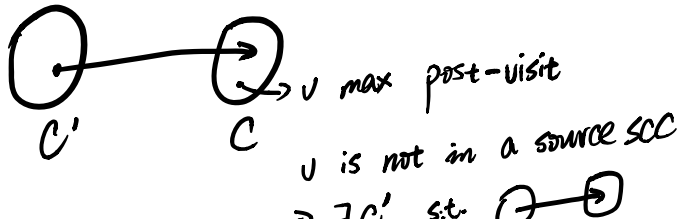




find the SCCs of a G

```
count = 0
repeat.
    find a vertex v
        in a sink SCC of G
    count ++
    explore(v)
    G = G - 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

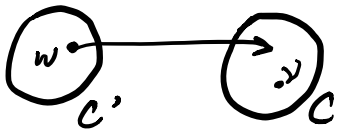
⇒ v visited first

post-visit numbers in C' > post-visit number of v

⇒ v post was not the max

(2) visit a vertex in C' before C

⇒ visit



after all of C will be visited

⇒ before backtracking from w

DFS visits all of C including v

⇒ w.post > v.post

SCC

Reverse of G (same vertice, reverse edges).

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 (v)

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

Disjoint Sets:

{1} {2} {3} {4} {5} {6} ⇒ n elements

union (1, 4)

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

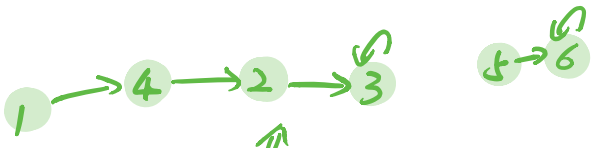
x.parent = x.

union (1.4)

union (5.6)

union (4, 2)

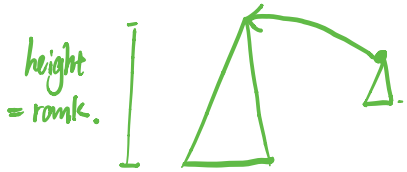
union (1, 3)



might take O(n) to find.
(worst case).

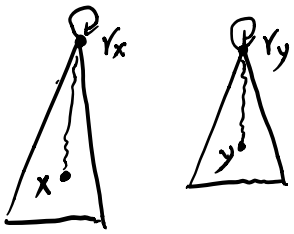
find (2) = find (3) not true.

find (1) = find (4) true.



Union by rank.

(make smaller (rank)
point to larger (rank))



union (x, y).

```
rx = find (x)
ry = find (y)
if ry.rank < rx.rank
    ry.parent = rx
else.
    rx.parent = ry
    if rx.rank = ry.rank
```

ry.rank++

union by rank.

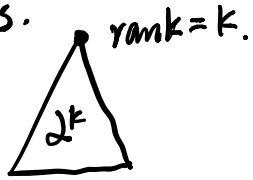
union(4, 2)

union(1, 3)

union(5, 1)

```
find(x)
  if x ≠ x.parent
    return find(x.parent).
  return x.
```

Every subtree with rank=k (of root) has at lease 2^k nodes.



by induction:

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

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

I.S. rank = r+1. rank incremented only when

by IH. nodes(T_1) $\geq 2^r$

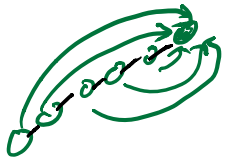
nodes(T_2) $\geq 2^r$

nodes(T_1, uT_2) $\geq 2^r + 2^r = 2^{r+1}$

```
makeSet(x)
    x.parent = x
    x.rank = 0
```

```
find(x)
    if x ≠ x.parent
        x.parent = return find(x.parent).
    return x.parent.
```

with path compression.



```
union(x, y)
    r_x = find(x)
    r_y = find(y)
    if r_x ≠ r_y
        if r_x.rank > r_y.rank
            r_y.parent = r_x
        else
            r_x.parent = y
            if r_x.rank = r_y.rank
                r_y.rank++
```

lgn

find ∈ O(lgn)

union ∈ O(lgn)

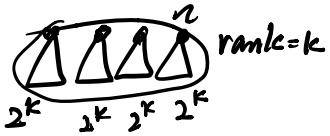
node at rank = k

nodes in subtree ≥ 2^k

Upper bound on number of nodes

rank = k

1 ≤ nodes at rank k ≤ $\frac{n}{2^k}$



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

$2^k \ge n$ $\quad k \le \lg n$

↑ rank.

unions
find
↘
$(m+n)\lg^* n$

Greedy ⟶ Try to locally minimize / maximize

Minimum Spanning Tree

goal: global optimum

Given a graph:

Tree: graph. acyclic, connected, undirected)

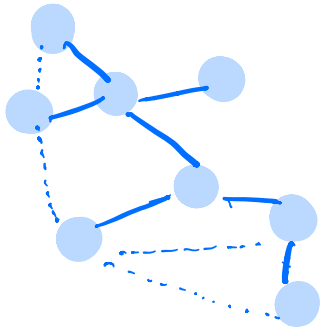
with a weigh function:

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

cut property.

we want to find a spining tree

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



to find next edge to add.
→ partition S, V−S = A. put
St. no edge in X of minimum weighe $w(T) = \sum_{e \in T} w(e)$ minimum
gose from , M V mill view

Chaim: Let e be an edge of minimum weight that crosses the cut ⇒ X ∪ {e} belongs to same muse of G

Prim(G)

```
for each v ∈ V { v.cost = ∞
                 v.pred = null
pick s ∈ V
s.cost = 0
Q.makeQueue(V)
while not Q.isEmpty()
    v = Q.extractMin()
    for each (v, u) ∈ E
        if w(v, u) < u.cost
            u.cost = w(v, u)
            u.pred = v
            Q.decreasekey(u)
```

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

Kruskal

```
for each v ∈ V                          O(|V|)
    makeset(v)
sort E by weight                        O(|E| lg |V|)
for each (u, v) ∈ E in ascending order
    if find(u) ≠ find(v)                |E| lg*|V|
        X = X ∪ {(u, v)}
        union(u, v)                     (|V|-1) lg*|V|
return X
```

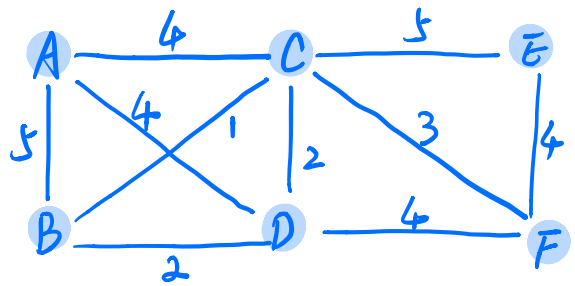
$O(|E|\lg|V| + (|V|+|E|)\lg^*|V|)$

$\lg|V|$: linear; $\lg^*|V|$: almost constant.

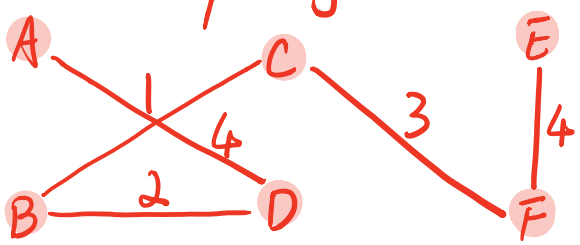
$\Downarrow$

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

Example.



minimal spinning tree.



total weight 14.

	A	B	C	D	E	F
	0/–	∞/–	∞/–	∞/–	∞/–	∞/–
A		5/A	6/A	4/A	∞/–	∞/–
D		2/D	2/D		∞/–	4/D
B			1/B		∞/–	4/D
C					5/C	3/C
F					4/F	
E						

Greedy Algorithm.

file compression.

(text)

symbols (a, b, c, ...)

ASCII — 8 bits

n symbols length = 8n

fixed length code

frequency table

i	a	f[i]
2	b	f[2]
3	C	f[3]
⋮	⋮	⋮

---

variable length encoding

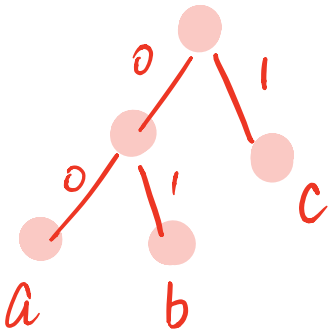
a 101

b 10

C 1

101 < a bC

symbols are in the leaves



abbac

00|0|1|0|0|1

a b b a C

optimal encoding

$\sum_i$ frequency(Si) · length(Si) ← minimizes

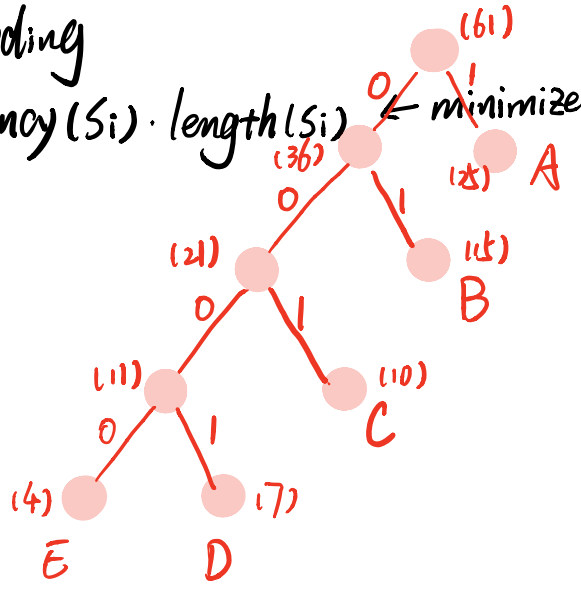
a-2步

b-1步

C-10

D-7

E-4



A 1

B 01

C 001

D 0001

E 0000

Huffman (f[1] . f[2] . ... f[n])

"local optimal thought"

build n nodes N = {i, f[i]}

Q. buildQueue (N)

O(n)

while Q.size > 1

lgn

u = Q.extractMin()

v = Q.extractMin()

create node $w = (-, u.f + v.f)$

set u and v to be children of w.

Q. insert(w)

lgn

return Q. extractMin()

$O(n \lg n)$

Dynamic Programming

$fibo(0) = 1$

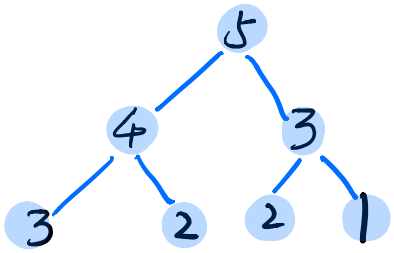
$fibo(1) = 1$

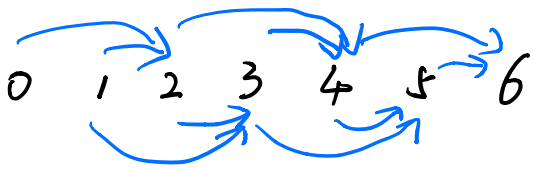
$fibo(2) = 2$

⋮

if ...

$fibo(n) = fibo(n-1) + fibo(n-2)$





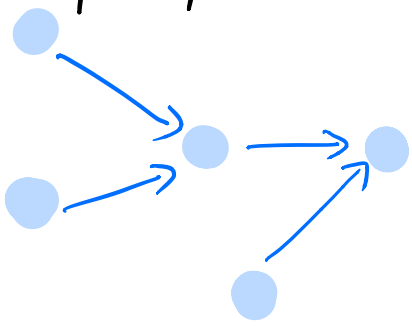
$fib(0) = 1$

$fib(1) = 1$

for 2 to n:

$fib(i) = fib(i-1) + fib(i-2)$

DAG of subproblems



Store them.

- find an order to traverse the DAG.

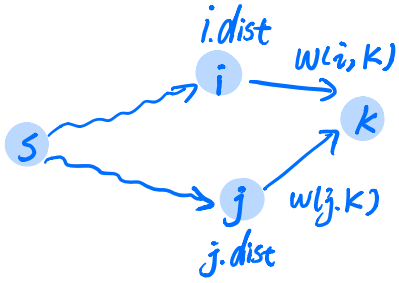
memorization

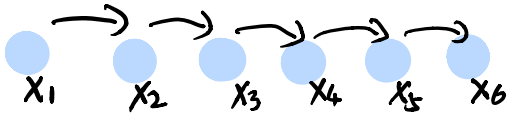
recursive algorithm.

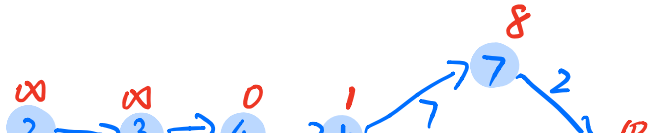
immediately before doing recursive call check if the solution is stored. else use the recursive call and store the solution

Find shortest paths in a DAG (single source)

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







linearize

Process them in

$\{x_1\}$

$\{x_1, x_2\}$

$\{x_1, x_2, x_3\}$

⋮

Algorithm

for all $v \in V$:

$v.dist = \infty$

$s.dist = 0$

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

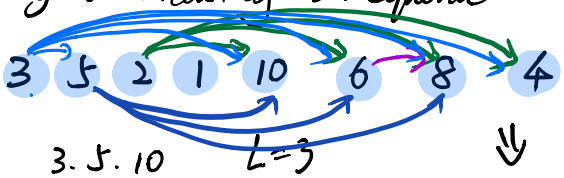
for each $v \in V$ in the order of the linearization

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

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

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

Longest increasing subsequence



3. 5. 10 L=3

3. 5. 6. 8 L=4

⇓ Just find longest path on DAG.

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

in the order given

$w(u,v) = 1$

Edit distance

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

X = E X T I N C T I O N

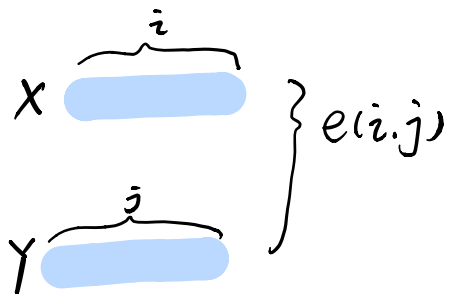
Y = I N T E N T I O N

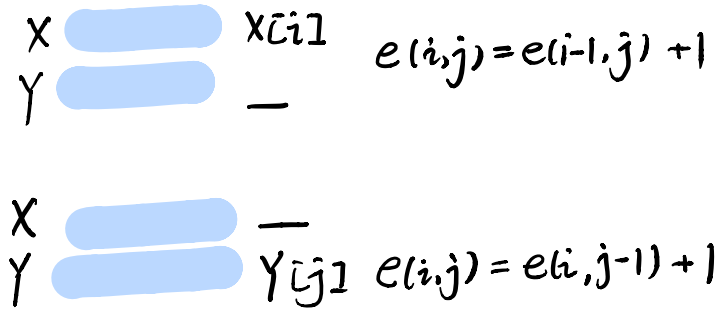
e = 8

E X T I N C T I O N

I N T E N _ T I O N

e = 4





X [redacted] X[i]

Y [redacted] Y[j]

$e(i,j) = e(i-1,j-1) + diff(i,j)$

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

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

```
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) + diff(i,j)}
```

$O(mn)$

Chain Matrix Multiplication

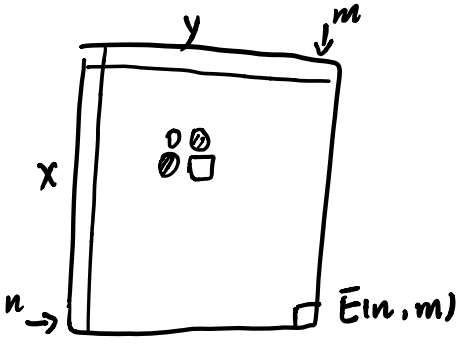
A 50×20 B 20×1 C 1×10 D 10×100

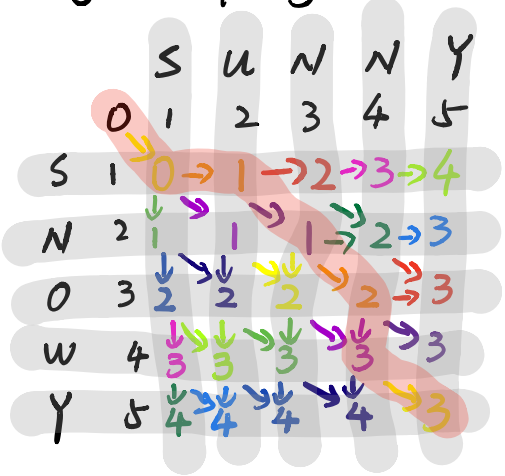
$A_{n\times r} \times B_{r\times m} = (\square)_{n\times m}$ $O(r)$

$O(nmr)$

Edit distance.

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





x S - N O W Y
Y S U N N - Y

3

① divide into subproblems

② combine subproblems to solve problem

③ Order (DAG)

Chain Matrix Multiplication

$(A_i, A_{i+1} \cdots A_k)(A_{k+1} \cdots A_j)$

A_0 $m_0 \times m_1$

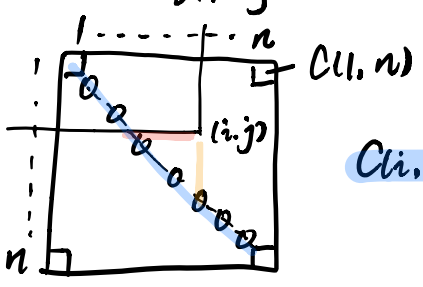
A_1 $m_1 \times m_2$

$\vdots$

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

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

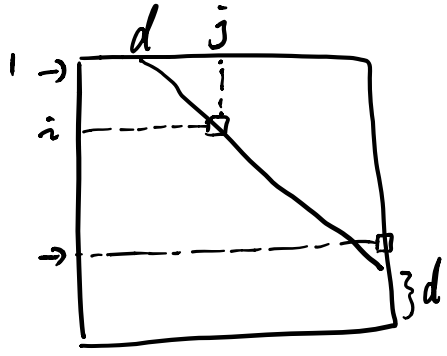
$i \le k < j$



$C(i,i) = 0$

Chain Matrix Multiplication $(m_0, \cdots m_n)$

```
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) = ∞
        for k=i to j-1
```



$n \times n$

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

if $v < C(i,j)$

$C(i,j) = v$

$R(i,j) = k$

$O(n^3)$

A_1 50×20 A_2 20×1 A_3 1×10 A_4 10×100

min: $k=1$ $0 + 3000 + 50 \cdot 20 \cdot 100 = 103000$

$k=2$ $1000 + 1000 + 50 \cdot 1 \cdot 100 = 7000$

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

$(A_1 A_2)(A_3 A_4)$

50 20 1 10 100

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

C

R =

	1	2	3
		2	2
		2	2
			3

All pairs shortest paths

$$M_0(i,j) = \begin{cases} w(i,j) & \text{if } (i,j) \in \bar{E} \\ 0 & \text{if } i=j \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin \bar{E} \end{cases}$$

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

→ compute i to j using $1 \ldots k$

for $k=1$ to n

for $i=1$ to n

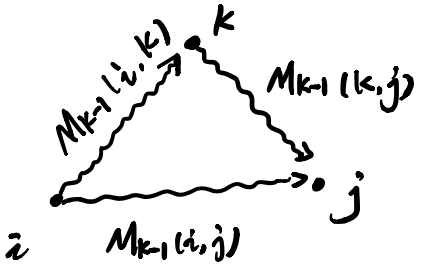
for $j=1$ to n

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

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

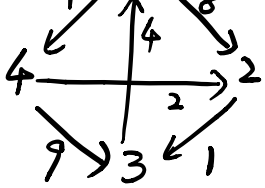
$M_k(i,j) = v$

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

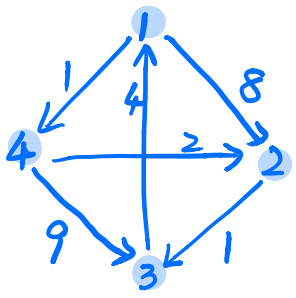


$M_k(i,j) = \min\{M_{k-1}(i,k) + M_{k-1}(k,j), M_{k-1}(i,j)\}$

	1	2	3	4



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



Floyd-warshall

M_0 =

	1	2	3	4
1	0	8	–	1
2	–	0	1	–
3	4	–	0	–
4	–	2	9	0

M_1 =

	1	2	3	4
1	0	8	–	1
2	–	0	1	–
3	4	12	0	5
4	–	2	9	0

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

M_k: $M_k(m,n)$ vs $M_k(m,k) + M_k(k,n)$

M_2 =

0	8	9	1
–	0	1	–
4	12	0	5
–	2	3	0

complexity: $O(n^3)$

Bellman-Ford: $O(|E||V|)$ foreach

Floyd-warshell is better.

M_3 =

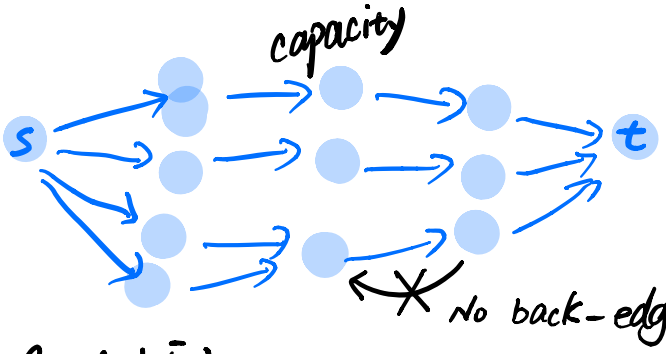
0	8	9	1
5	0	1	6
4	12	0	5
7	2	3	0

M_4 =

0	3	4	1
5	0	1	6
4	7	0	5
7	2	3	0

3,2 → 3,4 + 4,2

Max-Flow Problem.



(s) – source

(t) – sink.

$G = (V, E)$

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

s — No in edges

t — No out edges

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

Goal: find a flow

$f: E \to \mathbb{R}^+$

value of the flow.

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

1) $f(u,v) \le c(u,v)$

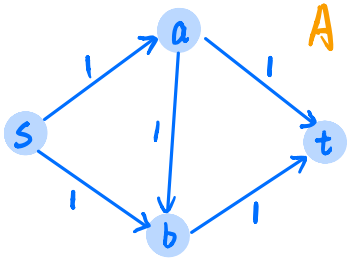
$\forall (u,v) \in E$

2) flow conservation

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

$f \rightrightarrows v \rightrightarrows f$ in == out

Example.



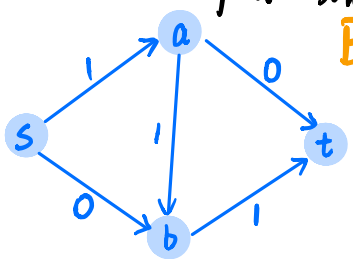
A

Algorithm:

find augmenting path.

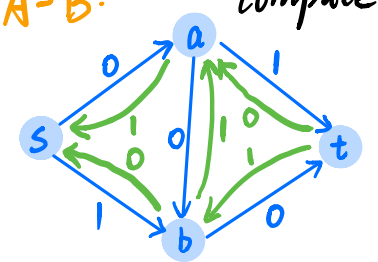
$p = sabt$

find bottleneck capacity (= 1)



B

A - B: compute residual capacity



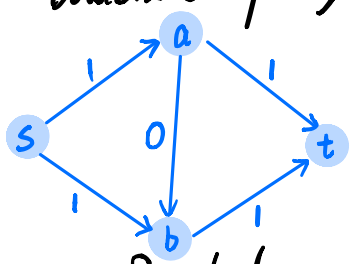
Residual Network.

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

Augmenting path

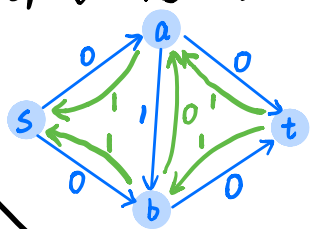
$p = sbat$

bottleneck capacity = 1



claim this is max.

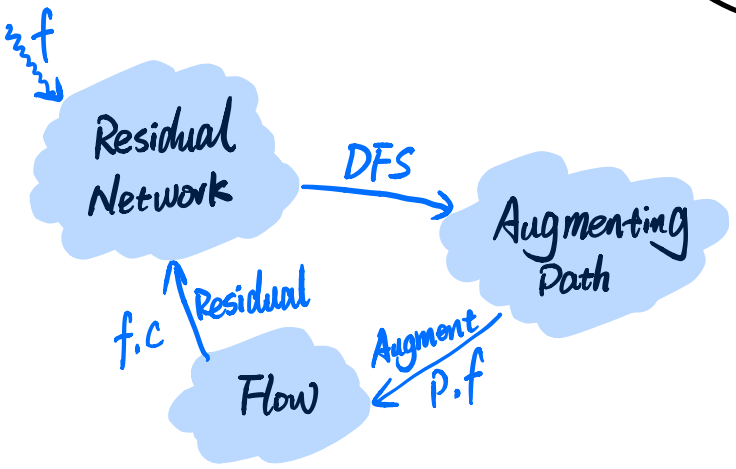
→ Repeat: Residual network.



find augmenting path ?

No.

Stop



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

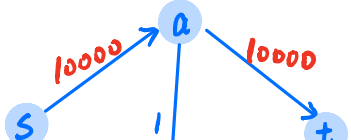
Augment (p, f, c^f)

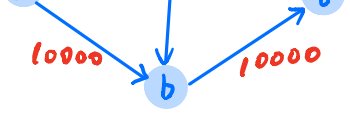
| P is an augmenting path
| f is a flow
| $b = \min_{i,j \in P} \{ c^f(i,j) \}$
| for each $(i,j) \in P$
| | if $(i,j) \in E$
| | | $f(i,j) = f(i,j) + b$
| | else
| | | $f(i,j) = f(i,j) - b$
| return f

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

| $\forall (u,v) \in E$
| | $f(u,v) = 0$
| c^f = Residual (c, f)
| while there is an augmenting path P on c^f
| | f = Augment (p, f, c^f)
| | c^f = Recidual (c, f)
| return f

Bad situation




10000
10000

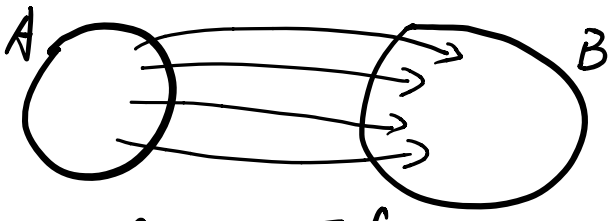
Ford & Fulkerson

Flow: a) $f(u,v) \le C(u,v)$

b) flow conservation

flow in = flow out

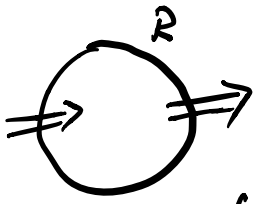
$\Rightarrow \cdot \Rightarrow$



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

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

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



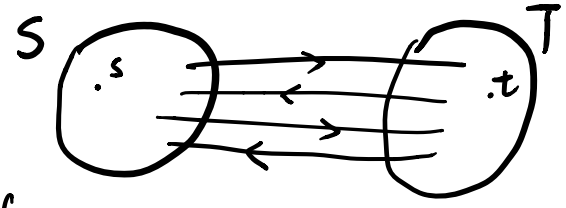
$$\sum_{(V-R)\to R} f = \sum_{R\to(V-R)} f$$

Def:

cut(S, T)

· partition: $S \cap T \subseteq V$. $S \cup T = V$. $S \cap T = \emptyset$

$s \in S$. $t \in T$



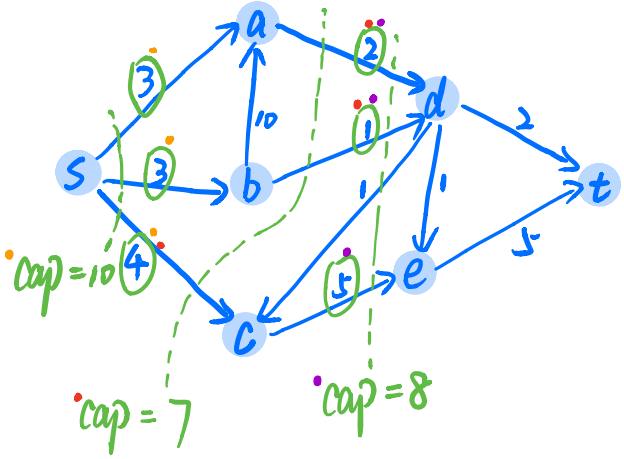
Def:

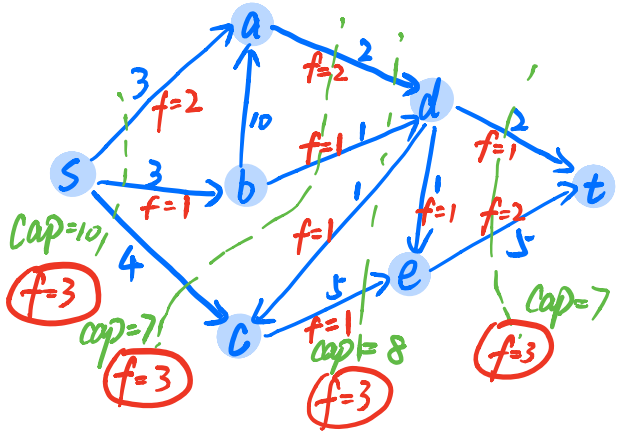
$$\text{flow}(S,T) = \sum_{S\to T} f - \sum_{T\to S} f$$

Def: capacity of a cut

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

From S to T. only consider one direction

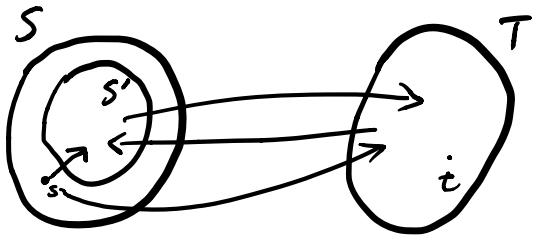




$$v(f) = \sum_{S\to V} f$$

Theorem Let S, T be a cut

$v(f) = flow(s, \bar{T})$



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

Since $s.t \notin S'$

flow into S' = flow out of S'

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

$$\Rightarrow \sum_{s \to S'} f = \sum_{S' \to \bar{T}} f - \sum_{T \to S'} f$$

$$flow(s, \bar{T}) = \sum_{s \to \bar{T}} f + \sum_{S' \to \bar{T}} f - \sum_{T \to S'} f$$

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

$$= \sum_{s \to v} f = v(f)$$

$$flow(s, \bar{T}) = v(f)$$

Theorem $v(f) \le cap(s, \bar{T})$

$$v(f) = flow(s, \bar{T})$$

$$= \sum_{s \to \bar{T}} f - \sum_{T \to s} f \le \sum_{s \to \bar{T}} f \le \sum_{s \to \bar{T}} C = cap(s, T)$$

F&F returns the (f^*) max Flow

ie. $\forall$ flow f

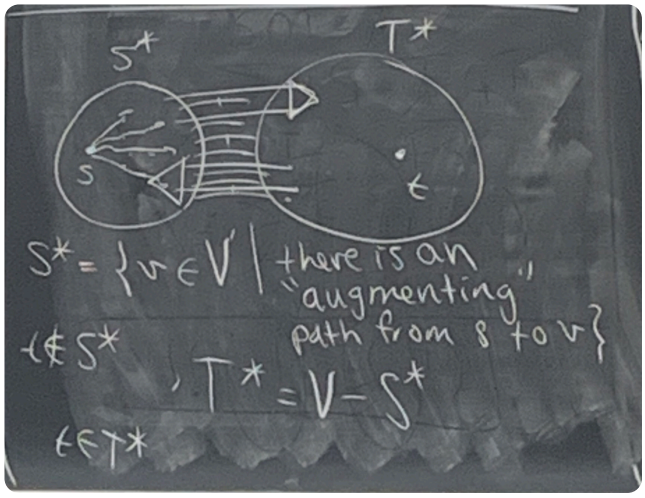
$$v(f) \le v(f^*)$$

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

$C_f(u, v) = 0 =$

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

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



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

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

$$= \sum_{S^* \to T^*} C$$

$$= cap(S^*, T^*)$$

$$v(f) \le cap(S^*, T^*) = flow(S^*, T^*)$$
$$= v(f^*)$$

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

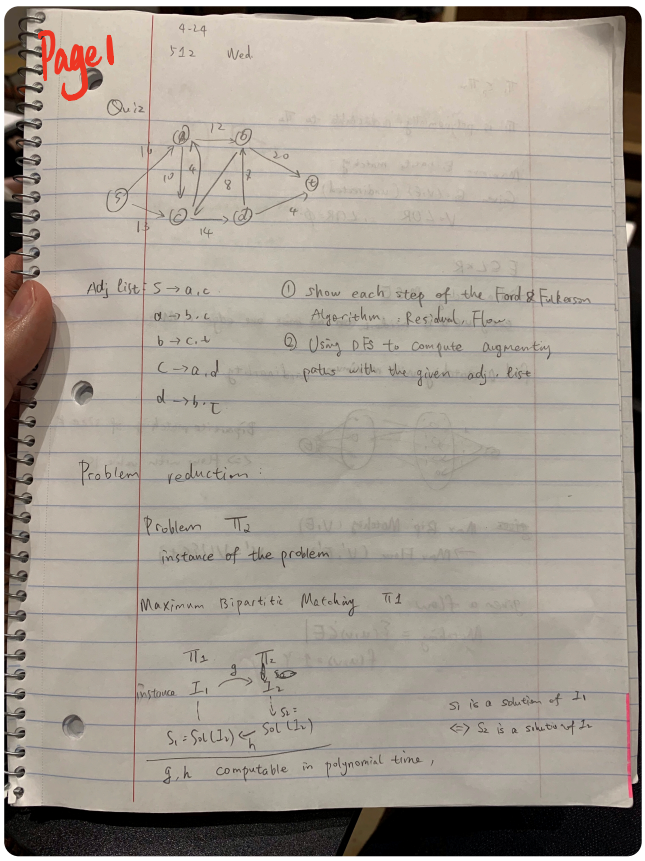
Problem Reduction

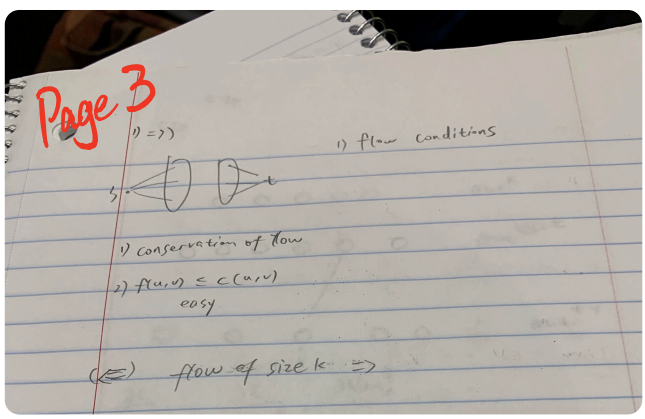
Problem Π_2 (maxflow)

instance of the problem

Maximum Bipartition Matching

Π_1





Π_1 Π_2

$I_1 \xrightarrow{g} I_2$

$Sol(I_2) \xrightarrow{h} Sol(I_2)$

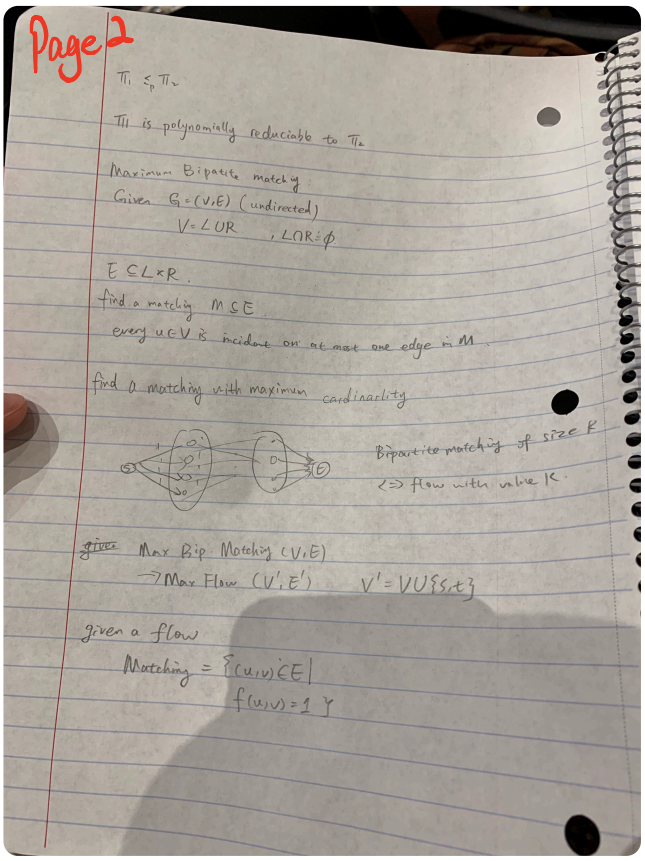
S_1 S_2

g, h computable in polynomial time.

S_1 is a solution of I_1

$\Updownarrow$

S_2 is a solution of I_2



Alphabet

Σ = set of symbols

$\Sigma = \{0,1\}$ $\Sigma = \{a,b,c,d,\cdots,z\}$

language: L = set of strings over an alphabet

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

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

Regular operations.

1) $L_1 \cup L_2$ 2) $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$ (concatenation)

example. $L_1 = \{aab, ba\}$

$L_2 = \{aba, aa\}$

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

3) kleene Star

$L^* = \bigcup_{n=0}^{\infty} L^n$ ← Σ^* = all possible strings over Σ

↳ $L^n = L^{n-1} L$

☆ $\varepsilon \in L^0$ ↖ empty string

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

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$

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

L_{prime} $\Sigma = \{0,1,2,3,\cdots 9\}$

Problem Prime

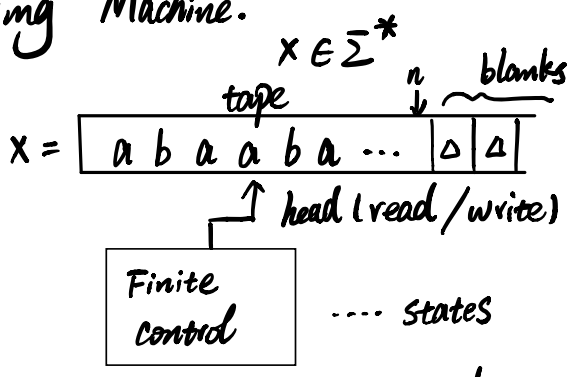
input $x \in \Sigma^*$ Question: $x \in L_{prime}$

Machine:

L_{prime} = language accepted by M_P

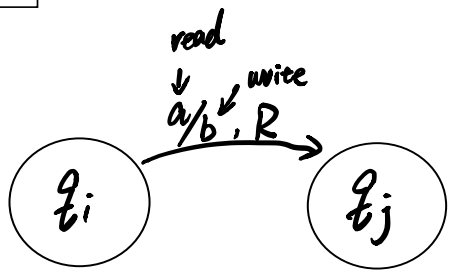
L_{prime} is the language of M_P; $L_{prime} = L(M_P)$

Turing Machine.



x is finite can be assumed as infinite

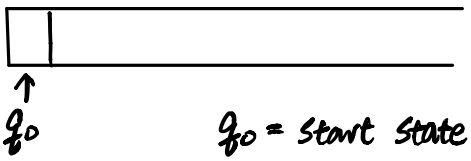
Transitions.



State q_i

State q_j

Starting configuration

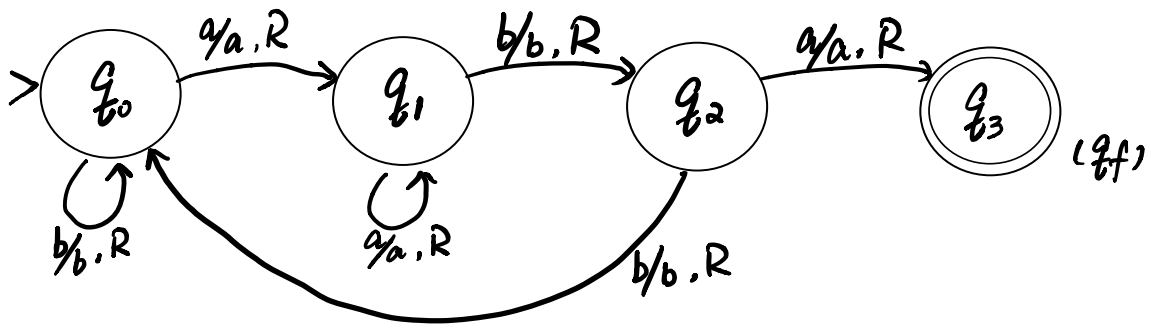


q_0 = start state

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

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

Transition Diagram.



Instaneous configurations:

aabab yield

q aababΔ $\vdash$

$aq_1abab\Delta \vdash aaq_1bab\Delta$

$\vdash aabq_2ab\Delta \vdash aabaq_3b\Delta$

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 Defination 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 $\quad \delta: Q \times \Gamma \to Q \times \Sigma \times \{L, R\}$

($Q \times \Gamma$: set of all possible pairs)

q_0 : start state

q_f : final state

Types of problems

- Decision Problems
  - Yes , • No
- Optimization Problems
  find max or min.
  (max-flow) ← 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

$L(M) = \Pi$ in Poly-time

↑ can be decided by DTM in polynomial time

Deterministic (DTM):

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

$\delta: Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$

Non-deterministic:

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

$\delta: Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{R, L\})$

↑ 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. $\phi(\dots) = T$

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

is ϕ SAT? $p = T$ $q = T$ $r = T$ → certificate (C)

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

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

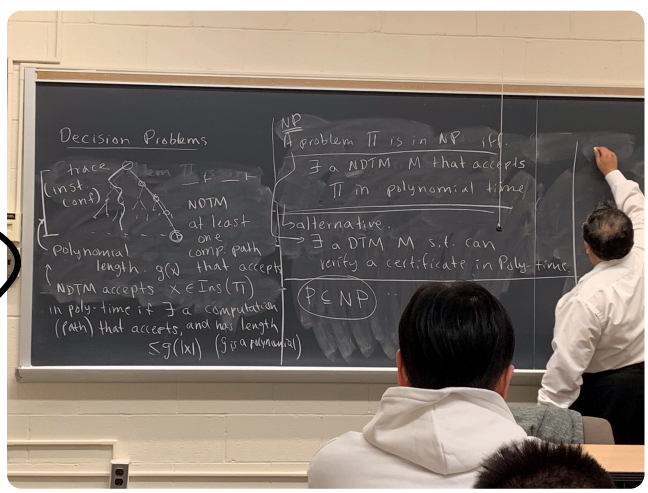
Polynomial Reducibility

Π_1 Π_2

$I_1 = Ins(\Pi_1) \xrightarrow{f} I_2 = f(I_1) = Ins(\Pi_2)$

$I_1 \in \Pi_1 \iff I_2 \in \Pi_2$

$\Pi_1 \leq_P \Pi_2$ 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 $I_1 \in Ins(\pi_1)$. $f(\pi_1) = I_2 = Ins(\pi_2)$

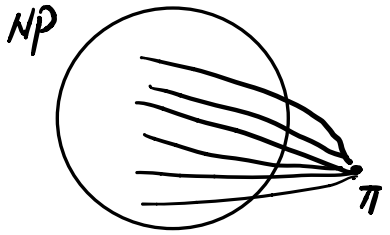
$|I_2| = g(|I_1|)$

2) solve I_2 in poly time (h)

time $= h(|I_2|)$

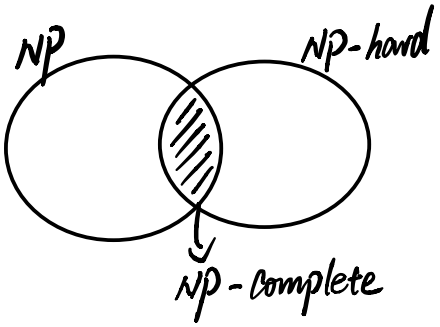
(so we solved I_1)

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 $P \subseteq NP$

$\therefore P = NP$

π_1 . NDTM instance (M, x) $\xrightarrow{f}$ boolean formula ϕ (CSAT)

M accepts x $\Leftrightarrow$ ϕ is sat

$x \in \pi_1$

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), v1,v2
   Output: v1=x1,x2,x3,x4,...,xn=v2 (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 natrual numbers
2. Are positive non-decreasing

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

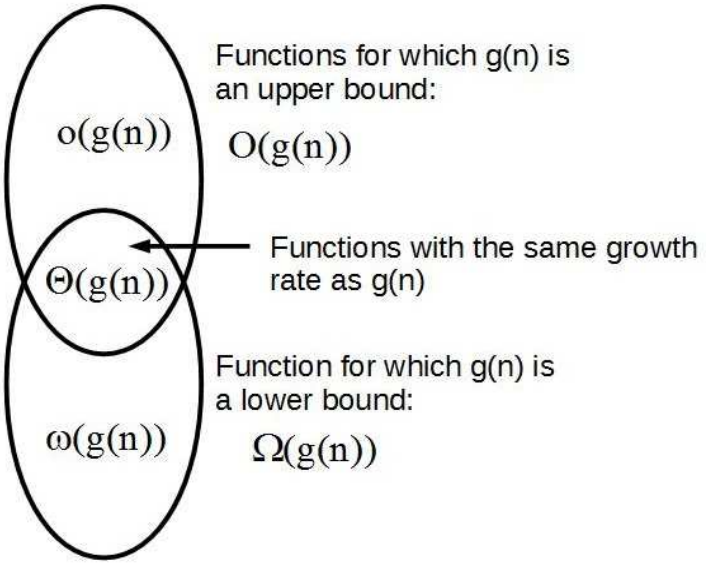
- Functions $f(n)$ for which $g(n)$ is an upper bound (i.e. functions that grow no faster than $g(n)$)

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

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

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

1.3 Little o and Little ω



Notice that

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

So we can define

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

where $f(n)$ is "sandwiched" between $c_1 g(n)$ and $c_2 g(n)$

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 $c_1 f(n) \leq f(n) \leq c_2 f(n)$, therefore $f(n) \in \Theta(f(n))$.

Symmetric:
Let $f(n) \in \Theta(g(n)) \Rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n)$.

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:

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

From the first inequality we have:

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

Combining it with the third inequality we have:

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

From the second inequality we have:

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

Combining it with the fourth inequality we have:

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

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:

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

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:

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

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

We now use limits to show each one.

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

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

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

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

because $k - 1 > 0$

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

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

L'Hopital 1 time:

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

L'Hopital 2 times:

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

L'Hopital 3 times:

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

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

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

Let

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

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

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

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

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

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

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

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

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

because $k - 2 > 0$

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

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

L'Hopital 1 time:

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

L'Hopital 2 times:

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

L'Hopital 3 times:

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

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

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

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

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

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

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

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:

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

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: $u = x^n$, $du = nx^{n-1}\text{dx}$, $dv = e^{-x}$, $v = -e^{-x}$. So we have:

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

Since $x^n e^{-x} = \frac{x^n}{e^x} = 0$ when $x \to \infty$ 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:

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

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