TOPICS:
  1. Background
     - Asymptotics
  2. Sorting
  3. Trees
  4. Dynamic Programming
  5. Graphs
  6. Parallel Algorithms
  7. NP-Complementeness
  8. Approximation Algorithms

Asymptotics:

\( \Theta(g(n)) = \{ f(n) : \exists \text{ constants } c, C, \text{ s.t. } \Theta \leq g(n) \leq \Theta, \Theta > 0 \} \)

\( \Omega(g(n)) = \{ f(n) \text{ s.t. } \exists \text{ constants } c, C, \text{ s.t. } \Omega \leq f(n) \leq \Omega, \Omega > 0 \} \)

\( o(g(n)) = \{ f(n) \text{ s.t. } \exists n = \infty \text{ s.t. } o \leq f(n) \leq g(n), \Theta > 0 \} \)

\( \omega(g(n)) = \text{ similar, check text} \)
Ex 1:
\[ 5n^3 = o(n^3) \not= o(n^3) \]
\[ 5n^3 = o(n^3) = o(n^3) \]

Ex 2:
Prove: \[ \sum_{i=1}^{n} \hat{e}^k = \Theta(n^{k+1}) \]

1) \[ O(n^{k+1}) \]
\[ \sum_{i=1}^{n} \hat{e}^k \leq \sum_{i=1}^{n} n^k = n - n^k = n^{k+1} \]

2) \[ \Omega(n^{k+1}) \]
\[ 2 \sum_{i=1}^{n} \hat{e}^k = \sum_{i=1}^{n} (\hat{e}^k + (n-\hat{e}+1)^k) \]
\[ \geq \sum_{i=1}^{n} \left( \frac{1}{2} \right)^k = \left( \frac{1}{2} \right)^{k+1} n^{k+1} \]
\[ = \Omega(n^{k+1}) \]

Ex 3: Show \[ \frac{1}{2} n^2 - 3n = \Theta(n^2) \]

\[ O(n^2) \]
\[ \frac{1}{2} n^2 - 3n \leq Cn^2 \]
\[ \frac{1}{2} n^2 - \frac{3n}{n} \leq C \]
\[ C = \frac{1}{2} \]

\[ \Omega(n^2) \]
\[ C n^2 \leq \frac{1}{2} n^2 - 3n \]
\[ C_1 \leq \frac{1}{2} n^2 - 3n \]
Some Properties of Asymptotics

\[ n^a \in O(n^b) \iff a \leq b \]
\[ n^a \in O(n^b) \iff a < b \]
\[ \log_b n \in O(n^a) \iff b^a = n \]
\[ \log_a n \in O(\log_b n) \iff a, b \]
\[ C^a \in O(d^a) \iff c \leq d \]

\[
\begin{align*}
\text{If } f(n) &\in O(f'(n)) \quad \& \quad g(n) \in O(g'(n)) \quad \text{then} \\
\quad f(n)g(n) &\in O(f'(n)g'(n)) \\
\quad f(n) + g(n) &\in O(\max(f(n), g(n)))
\end{align*}
\]

Best / Worst / Average \quad \{ \text{Probabilistic Analyses} \}

\quad \{ \text{Randomized Algorithm} \}

\quad \{ \text{Monte-Carlo} \quad \text{Las Vegas} \}

\quad \{ \text{(don't lie)} \quad \text{(lie)} \}

\text{Induction:}

\quad \text{Based on well ordering of numbers}

\quad P(1) \land P(k) \to P(k+1)
Inductive Bases
Inductive Hypotheses
Inductive Steps

Prove: \[ 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \]

Bases: \[ 1 = \frac{1(1+1)}{2} \]

Hypotheses: \[ 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \]

Inductive: \[ 1 + 2 + \cdots + (n-1) + n = \frac{n(n-1)}{2} + \frac{2n}{2} \]
\[ = \frac{n(n+1)}{2} \]

Ex: Prove any \( 2^n \times 2^n \) grid can be tiled with L-shaped (3 block) tiles leaving only one tile empty.

Bases

Hypotheses

\( 2^n \times 2^n \)

Can be tiled
Recurrences:

**Merge Sort**

\[ A = \text{MergeSort}(A[1..\lfloor n/2 \rfloor]) \]
\[ B = \text{MergeSort}(A[\lfloor n/2 \rfloor + 1..n]) \]
\[ C = \text{MergeSort}(A[2..n]) \]
\[ \text{Merge}(B, C) \]

\[ B = O(n) \quad C = O(n) \]

**Recurrences**

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n) \]

1. **Substitution (Induction)**

   **Guess** \( T(n) = cn \log n \)

   **Hypothesis**:
   \[ T(n/2) \leq C \frac{n}{2} \log \frac{n}{2} \quad \text{true } (\cdots \frac{n}{2} \]

   **Step**:
   \[ T(n) \leq 2C \cdot \frac{n}{2} \log \left(\frac{n}{2}\right) + O(n) \]
   \[ = Cn \log \left(\frac{n}{2}\right) + O(n) \]
   \[ = Cn \log n - cn + O(n) \]
   \[ = Cn \log n - n(c - c') \leq Cn \log n \]

   \( C \) can be large enough to dominate \( c' \)

   \[ T(1) = 1 \log 1 = 0 \]

   **Base**
   \[ n = 2 \quad T(2) = 4 \quad T(3) = 5 \]

   \[ T(2) \leq 2 \log 2 = 2c, \text{ I need } c \geq 2 \]
Errors:
\[ T(n) = O(n) = c \cdot n \]
\[ T(n) \leq 2 \cdot c \cdot \frac{n}{2} + O(n) = cn + O(n) \]
\[ = (c + c') n = c'' n \]

Inductive Assumption: \[ T(n/2) \leq c^{n/2} \]

Recursion Tree
\[ T(n) = T(n/4) + T(2n/3) + n \]

Time = \[ n \cdot h \]
\[ = n \log_{3/2} n = O(n \log n) \]

How about \[ T(n) = T(\frac{n}{16}) + T(\frac{9n}{10}) + n \]
\[ = O(n \log n) \]
ECE1762 Algorithm LEC02

Cont. Recursion

Renaming Variable

\[ T(n) = 2T(\sqrt{n}) + \log n \quad m = \log n \]
\[ T(2^m) = 2T(2^m) + m \]
\[ S(m) = 2S\left(\frac{m}{2}\right) + m = O(m \log m) \]
\[ = O(\log n \log \log n) \]

Iteration Method

Read Text

Master Theorem:

Let \( a > 1 \), \( b > 1 \) and \( f(n) = \) function

Recurrence

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]
has solution

- IF \( f(n) = O\left(n^{\log_b a - \epsilon}\right) \) for \( \epsilon > 0 \). Then \( T(n) = \Theta(n^{\log_b a}) \)
- IF \( f(n) = \Theta(n^{\log_b a}) \) then \( T(n) = \Theta(n^{\log_b a \log n}) \)
- IF \( f(n) = \Omega\left(n^{\log_b a + \epsilon}\right) \) for \( \epsilon > 0 \), \( af(n/b) \leq cf(n) \) for \( c < 1 \)
then \( T(n) = \Theta(f(n)) \)

Ex \( T(n) = T\left(\frac{2n}{3}\right) + 1 \)
\[ a = 1 \quad b = \frac{3}{2} \quad f(n) = 1 \]
\[ \log_{3/2} 1 = \varnothing \]
\[ \therefore = \Theta(\log n) \]
\[ T(n) = 3 \cdot T\left(\frac{n}{4}\right) + n \log n \]

\[ f(n) = n \log n, \ a = 3, \ b = 4 \]

\[ n \log n = 2^{n \log_2 3 + 0.2} \]

\[ T(n) = \Theta(n \log n) \]

\[ 2^\frac{n}{\log 4} < cn \log n, \text{ true for } c > \frac{3}{4} \]

Some useful formula:

\[ n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n \geq 1 \end{cases} \]

\[ \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \leq n! \leq e^{\theta(n)^{\frac{1}{2}}} \]

Fibonacci:

\[ F_n = F_{n-1} + F_{n-2} \quad F_0 = 0 \quad F_1 = 1 \]

\[ F_n = \frac{\phi^n - \overline{\phi}^n}{\sqrt{5}} \quad (\text{induction}) \]

\[ \phi = \frac{1 + \sqrt{5}}{2} \quad \overline{\phi} = \frac{1 - \sqrt{5}}{2} \quad \text{Golden Conjugate} \]

\[ \sum_{k=1}^{n} k = 1 + 2 + \cdots + k = \frac{n(n+1)}{2} = \Theta(n^2) \]

\[ \sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^k = \frac{x^{n+1} - 1}{x - 1} \]

\[ (x+r)^n = \sum_{i=0}^{n} \binom{n}{i} x^i r^{n-i} \]

\[ \sum_{r=0}^{n} x^r = \frac{1}{1-x} \quad \text{when } |x| < 1 \]
\[
\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad \sum_{k=1}^{\infty} x^k = \frac{1}{1-x} \quad \sum kx^k = \frac{x}{(1-x)^2}
\]

\[
\sum_{i=1}^{n} a_i - a_i^{-1} = a_n - a_1 \quad \sum_{i=1}^{n-1} \frac{1}{a_i(a+1)} = 1 - \frac{1}{n}
\]

**Logarithm**

\[a = b^c \quad \log_b a = c\]

Properties:

\[\log (ab) = \log a + \log b\]

\[\log a^n = n \log a\]

\[\log \frac{1}{a} = -\log a\]

\[\log_b \frac{a}{c} = \log_b a - \log_b c\]

\[a^{\log_b a} = n\]

\[\log^{(c)} n = \log \log \ldots \log n\]

\[\log^* n = \min \{c : \log^{(c)} n \leq 13\}\]

\[\log^* 2^{16} = 1 + \log^* 16 = 1 + 1 + \log^* 4 = 4\]

\[\log^* 2^{14} = 5\]

\[f : A \rightarrow B\]

\[\text{domain} \rightarrow \text{range}\]

- 1-1 Function: \(f(a) \neq f(a')\) for \(a \neq a'\)
- Onto: Every element in range is a map
- Bijection: 1-1 & Onto
Binary $R \subseteq A \times A$

Reflexive if $aRa$

Symmetric $aRb \Rightarrow bRa$

Transitive $aRb \land bRc \Rightarrow aRc$

\{ equivalence relationship \}

(all three)

$[a] = \{ b : aRb \}$

$[a]$ equivalence class of an element w.r.t $R$

antisymmetric: if $aRb \neq bRa \Rightarrow aRa$

partial order: reflexive, antisymmetric & transitive

A partial order $R$ is called total order iff $\forall a, b \in U$

we have $aRb$ or $bRa$

Graphs: $G = (V, E)$

- directed or undirected

- weighted $G$ (cost, profit, ...)

- path, simple path (no edge repetition)

- connected & unconnected

- induced subgraph

- clique (complete graph) All vertices connected
Adjacency Matrix

\[
\begin{bmatrix}
A & B & C & D \\
\times & v & v & x \\
\times & v & v & x \\
x & v & v & v \\
x & v & v & v \\
\end{bmatrix}
\]

Adjacency List

- C → D
- D → C → E → F
- E → F → D
- F → D → E

Sparse graph \( E \ll O(n^2) \)

Time: \( O(n) \)
Space: \( O(n^2) \)

Tree = connected, undirected, acyclic graph

For a tree, equivalent statement

1) If two vertices are connected by a simple path
2) Removing any edge disconnects the tree
3) A connected & \( |E| = |V| - 1 \)
4) A is acyclic and \( |E| = |V| - 1 \)
5) Acyclic and any new edge addition creates a cycle.
Permutations & Combinations

Rule of Product: If we go event A that can happen in \( n \)-ways, event B that can happen in \( m \)-ways there are \( n \times m \) ways both events can happen.

Rule of Sum: If we go event A that can happen in \( n \)-ways, event B that can happen in \( m \)-ways there are \( n + m \) ways A or B can happen.

Permutation: \( nPr = \frac{n!}{(n-r)!} = P(n, r) \)

If \( n \) objects are \( q_1 \) 1st kind, \( q_2 \) 2nd kind, \( \ldots \), \( q_r \) \( r \)th kind

Ex: Prove \( (k!)! \) is divisible by \( (k!)^{(k-1)!} \)

Assume you get

\[
\frac{(k!)!}{k! \cdot k! \cdot \ldots \cdot k!} = \frac{(k!)!}{(k!)^{(k-1)!}} \cdot \text{QED}
\]
Ex: Among 1..10¹⁰, how many contain digit 1

⇒ How many does not contain 1?

(10¹⁰ - 1)

⇒ How many contain 1

10¹⁰ - (10⁹ - 1)

Combinations: Some definition as Permutation except order doesn’t matter.

\[ C(n, r) = \frac{P(n, r)}{r!} = \frac{n!}{r!(n-r)!} = \binom{n}{r} \] a binomial coefficient.

Ex: How many diagonals a decagon has?

Review on Probability:

\( S = \) Sample Space
\( \emptyset \leq P(A) \leq 1 \)
\( \sum P(\text{Event}) = 1 \)
\[ P(\text{Event}) = \frac{1}{151} \] uniform distribution

Ex. A fair coin

\( S = \{H, T\} \) of not biased \( P(H) = P(T) \)

\[ P(A \cup B) = P(A) + P(B) - P(A \cap B) \]

Conditional Probability

\[ P(A|B) = \frac{P(A \cap B)}{P(B)} \] of \( A \& B \) are independent

\[ P(A \cap B) = P(A)P(B) \]
\[
\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}
\]

Discrete Random Variables

Maps Events to Real Numbers Allow Us to Generate Distributions.

Define Event

\[
X = x \text{ such as } \forall s \in S: X(s) = x
\]

\[
\Pr(X = x) = \sum_{s \in S: X(s) = x} \Pr(s)
\]

Expected Value

\[
\mathbb{E}[X] = \sum_{x} x \Pr(X = x)
\]

On Some $\mathcal{G}$, r.v. $X, Y$

\[
\Pr(X = x | Y = y) = \frac{\Pr(X = x \land Y = y)}{\Pr(Y = y)}
\]

Properties of Expected Value

\[
\mathbb{E}(X + Y) = \mathbb{E}(X) + \mathbb{E}(Y)
\]

\[
\mathbb{E}(aX) = a \mathbb{E}(X)
\]

\[
\mathbb{E}(XY) = \mathbb{E}[X] \mathbb{E}[Y] \quad x, y \text{ independent}
\]

\[
\operatorname{Var}(X) = \mathbb{E}[X - \mathbb{E}[X]^2]
\]

\[
= \mathbb{E}(X^2) - \mathbb{E}^2(X)
\]
Bernoulli Trial

An experiment with two outcomes \( S, F \)

\[ P(S) = p \quad P(F) = 1 - p = q \]

Geometric Distribution: How many (expected) times we need to wait to get a success.

\[ Pr(X = k) = q^{k-1} \cdot p \]

\[ E[X] = \sum_{k=1}^{\infty} k \cdot q^{k-1} = \frac{p}{q} \sum_{k=1}^{\infty} kq^{k} = \frac{q}{(1-q)^2} = \frac{1}{p} \]

Binomial Distribution: How many successes over \( n \)-trials?

\[ Pr(X = k) = \binom{n}{k} p^k q^{n-k} = b(n, k, p) \]

\[ E[X] = \sum_{k=0}^{n} k \cdot b(n, k, p) \]

\[ = \sum_{k=0}^{n} k \binom{n}{k} p^k q^{n-k} \]

\[ = np \sum_{k=1}^{n} \binom{n-1}{k-1} p^{k-1} q^{n-k} \]

\[ = np \sum_{k=0}^{n-1} \binom{n-1}{k} p^k q^{n-1-k} \]

\[ = np \]
HEAP SORT:
- In place

- Tree is just indicative, not used in practice:
  HEAP: a binary tree that has two properties:

  - Heap Shape Property

  complete except lower level where leaves were pushed to the left

  - Heap Order Property

  key (parent) > key (children)

```
  50
     |
  40  60
      |
  15  56
     |
  7   14
     |
  3   9
```

- Bubble Up
  compare (i) with \( \lfloor i/2 \rfloor \)
  exchange of larger \( \{ O(n) \)
Build - heap

bubble down (c)

replace w/ larger

call yourself recursively if you exchange an that element

Lemma: There are at most

\[
\left\lceil \frac{n}{2^h+1} \right\rceil
\]

notes at height \( h \) in a heap with \( n \) nodes

Time to build a heap

\[
T(\text{build-heap}) = \sum_{R=0}^{h} \frac{n}{2^R} \leq n \sum_{R=0}^{\infty} \frac{1}{2^R+1} = n \frac{1/2}{1/4} = 2n = O(n)
\]

\[
\sum_{r=0}^{\infty} r x^r = \frac{x}{(1-x)^2} \quad |x| < 1
\]

Extract Max

\( O(\log n) \): replace one right most leaf, bubble down new root

Heap Sort

Extract - max (A)

\(|A| = |A| - 1\)

repeat until \(|A| = 1\)

\( O(n \log n) \)

Min/Max Heaps

Priority Queue
QS (A, l, r)
  if l < r
    q = RATION(A, l, r)
    QS(A, l, q)
    QS(A,q+1, r)

PARTITION (A, l, r)
  k = l
  p\text{\textbf{r\textbf{o\textbf{t}}} = A[l][0]
  for i = l+1 \ldots r
    if A[i][0] \leq \text{p\textbf{r\textbf{o\textbf{t}}}}
      k = k+1
      Swap (A[i], A[k])
  Swap (A[l], A[k])
  return k

Worst Case (informal)
\begin{align*}
T(\theta) &= T(n-1) + \theta(n) \\
&= T(n-2) + \theta(n-1) + \theta(n) \\
&= \sum_{i=1}^{n} \theta(n^2)
\end{align*}

Best Case (informal)
\begin{align*}
T(n) &= 2T(n/2) + \theta(n) = \cdots = \theta(n \log n)
\end{align*}

Average Case
\begin{align*}
T(n) &= T(\frac{n}{q}) + T(\frac{b}{q}) + \theta(n) = \theta(n \log n)
\end{align*}
Randomized Partition \((A, l, r)\)

Randomly Swap \(A(l) \leftrightarrow A(\text{random} \cdot r)\)

Return Partition \((A, l, r)\)

Worst Case (Formal) both deterministic & non-deterministic

\[
T(n) = \max_{1 \leq q \leq n-1} \left( T(q) + T(n-q) \right) + \Theta(n)
\]

Solve by substitution, we guess \(O(n^2)\)

\[
T(n) \leq \max_{1 \leq q \leq n-1} \left( cq^2 + c(n-q)^2 \right) + \Theta(n)
\]

\[
= c \max_{1 \leq q \leq n-1} \left( q^2 + (n-q)^2 \right) + \Theta(n)
\]

\[
= c \max_{q=1 \text{ or } n-1} \left( q^2 + (n-q)^2 \right) + \Theta(n)
\]

\[
q = 1 \quad \text{to} \quad q = n-1
\]

\[
= c \left( 1 + (n-1)^2 \right) + \Theta(n)
\]

\[
= cn^2 - 2c(n-1) + \Theta(n) \leq Cn^2
\]

Can always choose \(c > c'\) or \(\Theta(n)\)

Average Case

\[
T(n) = \frac{1}{n} \left( T(1) + T(n-1) \right) + \frac{1}{n} \sum_{q=1}^{n-1} \left( T(q) - T(n-q) \right) + \Theta(n)
\]

\[
= \Theta(n) + \frac{1}{n} \sum_{q=1}^{n-1} \left( T(q) - T(n-q) \right)
\]
\[ \frac{1}{n} \sum_{q=1}^{n-1} T(q) + T(n-q) + \Theta(n) \]
\[ = \frac{2}{n} \sum_{k=1}^{n-1} T(k) + \Theta(n) \]

*Lemma:*
\[ \sum_{k=1}^{n-1} k \log k = \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \]

Assume \( T(n) = an \log n + b \)

\[ T(n) \leq \frac{2}{n} \sum_{k=1}^{n-1} a k \log k + b + \Theta(n) \]
\[ = \frac{2a}{n} \sum_{k=1}^{n-1} k \log k + \frac{2b}{n} (n-1) + \Theta(n) \]
\[ = an \log n + b + \Theta(n) + b - \frac{a}{n} n \leq an \log n + b \]

*Difference Between*

**Theorem:** Usually requires huge proofs.

**Lemma:** Small "Theorem" used to prove Theorem.

**Corollary:** Result of Theorem

**Axioms:** Can not be proved. Take it as a truth.

**Proof:** \[ \sum_{k=1}^{n-1} k \log k \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \]

\[ \sum_{k=1}^{n/2-1} k \log k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log k \]
\[ \leq (\log \frac{n}{2}) \sum_{k=1}^{n/2-1} k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \log k \]
\[ \leq (\log n) \sum_{k=1}^{n/2} k - \sum_{k=1}^{n/2} k + \log n \sum_{k=\lceil n/2 \rceil}^{n-1} \]

LEC05 Page 3 of 6
\[ \sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2} k \leq \frac{n}{2} n (n-1) \log n - \frac{1}{8} n^2 \]

Next week: Lower Bound on Comparison-based Sorting \( R(n \log n) \)

Bubble Sort:

for \( i = 1 \rightarrow n \) do
  for \( j = 1 \rightarrow n \) do
    if \( A(i) > A(j) \)
      Swap \( A(i) \& A(j) \)

RADIX Sort: A stable sort algorithm

Digits sorting within a range \([0..k]\)

\( n \) is the \# of sorting

\( d \) is the \# of digits

RS \((A, d)\)

LSB to MSB

\( \rightarrow \) stable sort \( A \) on digit \( i \)

\( \rightarrow \) insertion sort

\[
\begin{array}{cccc}
531 & 581 & 521 & 181 \\
424 & 181 & 424 & 424 \\
181 & 521 & 531 & 521 \\
732 & 732 & 732 & 831 \\
521 & 424 & 181 & 732
\end{array}
\]
Counting Sort: Stable but not in-place

A \[1\ldots n\] B \[1\ldots n\] C \[1\ldots \text{range} J\]

C[\text{\texttt{\texttt{int} \texttt{J}}}] = \text{\texttt{int}}

For \(i = 1 \ldots \text{length}(A)\)

\[\text{C}[	ext{A}[i]] = \text{C}[	ext{A}[i]] + 1\]

For \(i = 2 \ldots k\)

\[\text{C}[i] = \text{C}[i] + \text{C}[i-1]\]

For \(j = \text{length}[A] \text{ down to 1 do}\)

\[B[C[A(j)]] = A[j]\]

\[\text{C}[A(j)] = \text{C}[A(j)]-1 \text{ total time } O(n+k)\]

\[
\begin{array}{c}
A \[4\ 3\ 5\ 2\ 1\ 4\ 3\] \\
1\ 2\ 3\ 4\ 5 \\
C \[1\ 1\ 2\ 2\ 1\] \text{ loop \#1} \\
C \[1\ 2\ 4\ 6\ 7\] \text{ loop \#2} \\
B \[1\ 2\ 3\ 3\ 4\ 4\ 5\]
\end{array}
\]

Bucket Sort:

Assume you have \(n\) \(\text{\texttt{\texttt{int}}}s\) uniformly distributed between a range \([0 \ldots k]\)

\(16\text{\\#s} \ 0 \ldots 99\) - Create an array \(B\) of \(n\)-slots
- Use sort $E(c)$ with bubble sort
- "Thread" slots together

Assume $n_i$ elements in bucket $i$

$E(\text{sort one bucket}) = E(O(n_i^2)) = O(E(n_i^2))$

Time:

$$\sum_{i=0}^{n-1} E(0(n_i^2)) = O(\sum_{i=0}^{n-1} E(n_i^2)) = 2n - \frac{1}{n} \cdot n = O(n)$$

$$\left(1 - \frac{1}{n}\right) + \frac{1}{n} = 2 - \frac{1}{n}$$

$p = \frac{1}{n}$

Binomial Distribution

$E[x] = n \cdot p = 1$

$\text{Var}[x] = npq = n \cdot \frac{1}{n} \left(1 - \frac{1}{n}\right)$

$= 1 - \frac{1}{n}$
Lower Bound on Sorting

Any comparison based sorting algorithm for $n$-elements of unrestricted range takes $\Omega(n \log n)$ time.

Insertion Sort

5 3 1 2

c.e. Decision Tree for insertion sort of 3 elements
we have \( n! \) possible outcomes of sorted sequences given \( n \)-element

\[
2^n = \text{max of leaves for binary tree} 
\]

\[
h! = \left(\frac{n}{e}\right)^n 
\]

\[
2^h \geq n! \iff h \geq \log n! \geq (\log \left(\frac{n}{e}\right))^n 
\]

\[
\Rightarrow h \geq n \log \frac{n}{e} = n \log n - n \log e = \Omega(n \log n) 
\]
Selecting $k^{th}$ largest

Finding largest in an array

1. Compare and Swap

2. Comparator Tree

- if known max $\frac{n}{2} - 1$ to find min
- $2^{nd}$ max logn additional comp.
- $(n-1) + \log n$ comp for $k^{th}$ max.

To find largest $O(n)$

Finding $k^{th}$-max

- Expected $O(n)$ algorithm
- Worst case $O(n)$ algorithm

Random Select $(A, p, r, \hat{c})$

if $p = r$ then return $A(p)$

$q = \text{rand - partition}(A, p, r)$

$k = q - p + 1$

if $\hat{c} \leq k$ then

Random Select($A, p, q, \hat{c}$)

else

Random Select($A, q + 1, r, \hat{c} - k$)
Efficiency:

\[ T(n) = \frac{1}{n} \sum \left( \max(1, n - 3) + \sum_{k=1}^{n-1} \left( \max(k, n-k) \right) \right) + O(n) \]

\[ \leq \frac{1}{n} \sum_{k=n/2}^{n} T(k) + O(n) \]

Assume \( T(n') \leq cn' \), \( n' < n \)

Prove it works for \( n \)

\[ T(n) \leq \frac{2c}{n} \sum_{k= \lceil n/2 \rceil}^{n} k + O(n) = \frac{2c}{n} \left[ \sum_{k=1}^{n} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \right] + O(n) \]

\[ = \frac{2c}{n} \left( \frac{1}{2} (n-1)n - \frac{1}{2} \left( \frac{n}{2} - 1 \right) \frac{n}{2} \right) + O(n) \]

\[ \leq c \cdot n \]
Worst Case $O(n)$

1) Partition array in $\frac{n}{5}$ groups of 5 elements
2) Find the median of each group (will insertion sort)
3) Recursively find median of all medians, call it $x$
4) Partition original array around $x$
5) Recursively call step 1 on the right side of partition

\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
\[
\begin{array}{c}
5 \\
\downarrow \\
5 \\
\end{array}
\]
At every iterations of the algorithm

At least half of the groups constitute 3 elements except last group & group that contains median of all medians.

\[
3 \left( \frac{1}{2} \right) \left( \frac{n}{5} \right) - 2 \geq \frac{3n}{10} - 6
\]

At step 5 we will need at most $\frac{7n}{10} + 6$ elements
\[ T(n) = \begin{cases} \theta(n) & \text{if } n < 140 \\ T\left(\frac{n}{8}\right) + T\left(\frac{7n}{10} + 6\right) + o(n) & \text{if } n \geq 140 \end{cases} \]

\[ T(n) \leq C \left[ \frac{n^7}{8} \right] + C \left( \frac{7n}{10} + 6 \right) + an \]

\[ \leq \frac{cn}{5} + C + \frac{7cn}{10} + 6c + an \]

\[ = \frac{acn}{10} + 7c + an \]

\[ = cn + \left( -\frac{cn}{10} + 7n + an \right) \leq cn \]

\[ \text{as negative} \]

\[ -\frac{cn}{10} + 7c + an \leq 0 \]

\[ c \geq 10a \left( \frac{n}{n-70} \right) \leq 2 \]

\[ \theta \text{ if } c \geq 20a \text{ makes the quantity negative} \]

QED \text{ (Quod Erat Demonstrandum)}
Binary Search Trees

- Binary Tree
  - key
  - left subtree < parent < right subtree

Sort: In order \( O(n) \)
Search: prope the root
  go left or right among comparison \( O(h) \)
Insert: Assume unique key
  Search and insert as leaf \( O(h) \)
Max/Min: Most right or most left
  in order
Successor: element visited after in inorder traversal \( O(1) \)
Delete: 1) Search for it
  2) a. leaf \( \rightarrow \) delete
     b. 1 child \( \rightarrow \) replace with immediate child
     c. 2 children \( \rightarrow \) replace with largest on the left
        or smallest on the right \( O(h) \)
\[ h = O(n) \text{ worst case} \]

A balanced Binary Tree \[ h = O(\log n) \]

R-B trees, AVL trees, B/B⁻¹ trees, 2/3 trees

Splay Trees

How long does it take to build a BST?

\[ n \log n \quad n^2 \text{ (worst)} \]

Theorem: if keys are selected from a uniform distribution, then the expected number of comparisons is \( O(n \log n) \).

Let keys be \( a_1, a_2, \ldots, a_n \) & their sorted sequence be \( b_1, b_2, \ldots, b_n \).

\[ \mathcal{T}(n) = \frac{1}{n} \sum_{j=1}^{n} \left[ (j-1) + \mathcal{T}(j-1) + \mathcal{T}(n-j+1) \right] \Rightarrow O(n \log n) \]
Most ops depend \( O(h) \), for BSTs \( \log n \leq h \leq n \)

"Balanced Trees" = guarantee \( h = O(\log n) \)

**Red-Black Tree Properties:**
- A BST
- Every node is red or black
- The root is always black
- A red node needs have a black child
- Each path from root to a leaf has the same # of black nodes

**Def:** Black height \( bh(x) = \) # black nodes of any path from \( x \) to a leaf excluding \( x \).

**THM:** A RB-tree with \( n \)-internal nodes of has

\[ h \leq 2 \log(n+1) = O(\log n) \]

**Lemma:** A subtree rooted at \( x \) has at least \( 2^{bh(x)} - 1 \) internal nodes.

**Proof:** Induction on \( h \) of tree

- \( h = 0 \Rightarrow bh(x) = 0 \Rightarrow 2^0 - 1 = 0 \) (internal nodes)
- \( h \) depends on color of \( x \) and is \( 2^{bh(x)} - 1 \) or \( 2 - 1 \)
\[
\#\text{nodes of subtree } = (2^{bh(x)+1} - 1)H(2^{bh(x)} - 1) + 2 = 2^{bh(x)} > 2^{bh(x)-1}
\]

**THM** A RB-tree with \( n \) internal nodes of has

\[
\log n \leq 2 \log (n+1) = O(\log n)
\]

**Proof**

Lemma \( \Rightarrow \) \( n \geq 2^{bh(\text{root})} - 1 \geq 2^{bh} - 1 \)

\[
\log n \leq 2 \log (n+1) = O(\log n)
\]

**Rotations in BSTs**

\[
\begin{align*}
    &\text{right rotation} \\
&a \\&b \\
&x \\
\text{left rotation} \\
&c \\
&x \\
&y
\end{align*}
\]

Rotations preserve BST-order property

**Insert = Search and Insert:** rotate to fix color

**Case 1:**

- y sibling of \( p(x) \) as red
  - a) \( x \) \( \in \) \( R(p(x)) \)
  - b) \( x \) \( \in \) \( L(p(x)) \)

LEC07 Page 2 of 7
Case 2: Sibling $y$ of $p(x)$ is black. $x \leftarrow L(p(x))$

$\text{Case 2: Sibling } y \text{ of } p(x) \text{ is black } x \leftarrow L(p(x))$

Hashing: Dictionary Operations Insert/Delete/Search $\approx O(1)$

$U = \text{SIN}$

One way: Place them on array $O(1)$ times

Bad News: XXX XXX XXX

$10^9 \rightarrow$ storage

means 96% of space wasted

Motivation: want a D.I. that achieves $\approx O(1)$ time but uses $\approx O(n)$ space as well
n-elements to store
m-entries long

Usually \( m > n \) \( \frac{n}{m} = \alpha = \text{load factor} \)

When \( h(x) = h(y) \) for \( A \neq B \) then we have a collision

1) resolution by chaining
2) open addressing

Simple Uniform Hashing

If \( h \) hashes a key/ probability \( \frac{1}{m} \) to any slot, then the expected \# of probes to search is \( O(1 + \alpha) = O(1 + \frac{n}{m}) \)

\[
\# \text{probes} = (\text{time to calculate } h) + (\text{time to search end}) = O(1) + \frac{1}{\alpha} \sum_{i=1}^{\alpha} \frac{i-1}{m}
\]

\[
= O(1) + \frac{n}{m} + \frac{1}{\alpha m} \sum_{i=1}^{\alpha} i - 1 = O(1) + \frac{(\alpha - 1)}{\alpha} \cdot \frac{1}{m}
\]

\[
= O(1) + \frac{1}{2} - \frac{1}{2} m = O(1 + \alpha)
\]
Different Types of Hash Functions

Division Method
\[ h(k) = k \mod m \]
bad value \( m = 2^p \quad p \in \mathbb{Z} \)

Multiplication Method
\[ h(k) = \lfloor m \cdot (k \cdot A \mod 1) \rfloor \quad 0 < A < 1 \]
when \( m = 2^p \)

Good Value \( A \)
\[ -\phi = \frac{\sqrt{5} - 1}{2} \quad \text{(inverse golden conjugate)} \]

Universal Hashing

There are \( m^n \) ways to create a hash function.
Which one is good? It is the one that hashes two distinct keys \( x \neq y \) to same bucket \( w \) with probability \( 1/m \).

We call \( H \) a family of universal hash functions if the number of hash functions from \( H \) that wrap two distinct keys into the same bucket is \( |H|/m \).

\[ \Pr (h(x) = h(y)) = \frac{|H|/m}{|H|} = 1/m \]
Performance:

\[ C_{xy} = r \cdot v = \begin{cases} 1 & \text{if } x \neq y \text{ collide} \\ 0 & \text{otherwise} \end{cases} \]

\[ E \left[ n(\frac{1}{m}) \right] = \frac{1}{m} \]

\[ E \left[ \# \text{collisions} \right] = \sum_{y \neq x} E\left( C_{xy} \right) = \frac{n-1}{m} = \alpha = O(1) \quad x \neq y \]

Lemma 1: If \( \gcd(a,n) = 1 \) then

\[ a \cdot x \equiv b \pmod{n} \iff (ax) \mod{n} = b \mod{n} \]

has a unique solution for \( x \)

Ex:

\[ 8 \cdot x \equiv 13 \pmod{15} \quad x = 1 \]
\[ 7 \cdot x \equiv 11 \pmod{5} \quad x = 3 \]

How to create \( H \) ? Pick \( m \)-prime

Decompose every key \( x \) into \( r \)-pieces

\[ x = \langle x_0, x_1, \ldots, x_{r-1} \rangle \]

\[ \Theta x_e \leq m \]

Let \( a = \langle a_0, a_1, \ldots, a_r \rangle \)

where

\[ \Theta a_e \text{ is selected randomly from } \langle 0, 1, 2, \ldots, n-1 \rangle \]
Define \( h_a(x) = \left( \sum_{i=0}^{r} a_i x_i \right) \mod m \)

\( H = h_a \) the choice of \( a \), \( |H| = m^{r+1} \)

This is a universal hash function seed. Why? Pick distinct key \( x \& y \) and they are different because \( x \neq y \). Let \( h(a) = h_d(y) \)

\[
\sum_{i=0}^{r} a_i x_i = \sum_{i=0}^{r} a_i y_i \mod m \leq a_o (x_0 - y_0) = -\sum_{i=1}^{r} a_i (x_i - y_i) \mod m
\]

\[
\frac{m^r}{m^{r+1}} = \frac{1}{m} \text{ hash function collide } x \& y
\]

QED
Amortized Analysis

Bound defined over a sequence of n-ops

STACK
push, pop, multipop (=empty) n ops on total

pop = push = O(1)
multipop = O(k) = O(n)
max: O(n^2) cycles

Bet - Counter

Increment
\[ \hat{c} = \emptyset \]
while \( \hat{c} < \text{length}(A) \)
and \( A[\hat{c}] = 1 \)
\[ A(\hat{c}) = \emptyset \]
\( \hat{c}++ \)
if \( \hat{c} < \text{length}(A) \)
\[ A(\hat{c}) = 1 \]
Aggregate / Stack

\[ \text{pop} = O(1) \quad > O(n) \]
\[ \text{push} = O(1) \quad > O(n) \]
\[ n_1 + n_2 + n_3 + \ldots + n_c = O(n) \]

\( \text{total} \quad O(n) \quad \text{or} \quad \frac{O(n)}{n} \quad \text{on average} \)

Aggregate / Counter

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & \vdots \\
0 & 1 & 1 & 0 & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}
\]

\[
\sum_{i=0}^{A} \frac{n}{2^i} \leq \sum_{i=0}^{B} \frac{n}{2^i} = n \sum_{i=0}^{B} \frac{1}{2^i} = n \frac{1}{2^{i+1}} = O(n)
\]

Accounting Method

Every op charges a fixed $\$
- Some pay for op
- Some are left as credit on ADT to pay for future ops
- You can never run on negative credit
Accounting Method/Stack

<table>
<thead>
<tr>
<th>Actual Cost</th>
<th>Amortized Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>push 1</td>
<td>$2</td>
</tr>
<tr>
<td>pop 1</td>
<td>$0</td>
</tr>
<tr>
<td>$\text{m-pop}$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

$O(1) / \text{op}$

Every time we flip $\varphi \leftarrow 1$

$\$1 \text{ for } \varphi \leftarrow 1 \text{ and } \$1 \text{ for } i \Rightarrow \varphi$

$n$-increment cost is $2n$

or

$O(1) / \text{per increment}$

Potential ADT has \text{"potential" and associate op-}i \text{ with actual cost } \varphi \text{ & amortized cost } \hat{C}$

$$\hat{C} = C + \sum_{i} ^{\varphi} \left[ \varphi_{i} - \varphi_{i-1} \right] \Rightarrow \hat{C} = C + \Delta \varphi$$

Total Cost = $\sum_{i} ^{\hat{C}} \hat{C} = \sum C_{i} + \Delta \varphi$

$\Delta \varphi = \varphi_{m} - \varphi_{0}$ \geq \varphi

Then $\hat{C}_{i}$ is an upper bound to actual cost
Define potential \( \phi_c = \# \) elements in stack after op-c

Potential / Counter \( \phi_c = \# 1 \) bits in counter.

Assume \( c^{th} \) increment resets \( t_i \) bits. Its actual cost is \( t_i + 1 \)

Now many bits you get after \( c^{th} \) increment = \( b_i \)

\[ b_i \leq b_{i-1} - t_i + 1 \]

\[ \Delta \phi = \phi_c - \phi_c = b_{i-1} - t_i + 1 = -t_i + 1 \]

\[ \Delta t = t_i + 1 + (-t_i + 1) = 2 \]

A very interesting topic on Hashing (Dynamic hashing table)

You don't know how many elements you hash a priori

\[ n \leq m \quad \text{load factor} \quad \lambda = \frac{n}{m} \leq 1 \]

Insert: Start w/table of size 1, when add 2\textsuperscript{nd} elem. make its size 2, 4, 8, 16
contract in 1/2 when 1/4 of table is filled

$2

$2

A

$2

$2

A | B


SPLAY TREES (Sleator & Tarjan, 1983)

If known frequencies are off-line then can use dynamic programming to

\[ \frac{w_i}{w} \]

Theoretical Optimal

SPLAY TREES

access \( \sum \frac{w_i}{w} \log \frac{w_i}{w} \)

SPLAY(x) /* rotates till node becomes tree root */

while (x) not root

if \( p(x) = \text{root} \)

\{ 2zc \}

if both x & p(x) are left or right children

\{ 2zc \}

else

\{ 2zc - 2Ag \}
Insert: Like BST, splay (new leaf)
Search: =/= , splay (reached element)
Delete: =/= , splay p(x) O(log n)
Join: Splay tree (A) ≤ x < splay tree (B)
Split: Splay on x & split as

O(log n)
Define \( \text{weight}(X) = \# \text{external nodes under } X \) \\
\( \text{rank}(X) = \lceil \log \text{wt}(X) \rceil \) \\
Credit Invariant: Every node will rank \( (X) \) $s on it.

CLAIM: Every zig, zag-zag, or zig-zag operation needs at most \( 3(\text{nr}(X) - \text{or}(X)) \) $s except zig that may need $1 additional

\[ \text{} \]

Time: If claim is true, then \( O(\log n) \) $s are enough for a splay

let the change of ranks be \( \text{rank}_k \) (when root), \( \text{rank}_{k-1}, \ldots \)

\( \ldots, \text{rank}_0 \) (start)

Because of claim: 
\[
1 + 3(\text{rank}_k - \text{rank}_{k-1}) + \\
3(\text{rank}_{k-1} - \text{rank}_{k-2}) + \\
\ldots + \\
3(\text{rank}_1 - \text{rank}_0)
\]

\[
= 1 + 3(\text{rank}_k - \text{rank}_0)
\]

\[
\leq 1 + 3(\log n - \phi)
\]

\[
= O(\log n)
\]

Exercise: SPLAY a linked list and observe how the tree balance
Need to prove the claim

Only \( g/p/x \) change ranks. To restore credit invariant we need

\[
nr(g) + nr(p) + nr(x) - or(x) - or(g) - or(p)
\]

\[
\begin{align*}
nr(g) & \leq nr(x) \\
nr(p) & \leq nr(x) \\
or(p) & \geq or(x) \\
nr(x) & = or(g)
\end{align*}
\]

we need \( 2(nr(x) - or(x)) \) how much \$/s need

If \( nr(x) = or(x) \) \( \Rightarrow \) no new \$/s allocated to pay for rotation

\( \Rightarrow \) > \( \frac{1}{2} \) tree nodes under \( X \) (C&D)

\( \Rightarrow \) < \( \frac{1}{2} \) are on A&B

\( \Rightarrow \) \( g \) drops on rank to pay for rotation
Analysis same as before except case
\[ nr(x) = or(x) \]

More than \( \frac{1}{2} \) children under B & C

Case 1: if (i.e.) B \( \gg \) C
then \( p \) decreases in rank

Case 2: B \( \approx \) C
then \( p \) and \( g \) decrease in rank to pay for rotations

OR(x) \( \leq \) NR(x)
OL(p) \( \geq \) nr(p)

we can deposit \( nr(x) - or(x) \)
on x to restore the invariant
The remaining two ports pay for rotation.

If \( r(x) = o(x) \) then use the extra $1 to pay for rotation (single case).
ECE 1762 Algorithm LEC10

Dynamic Programming

"Divide & Conquer"

Characteristics

- Overlapping Sub-Problems
  (Similar to Greedy)
- Memorization

Example: find $F(6)$

```
F(6)  
/    
\   / 
F(5) F(4) result
  /  
F(4) F(3)  
  /  /  
F(3) F(2)F(2) F(1)
  /  /  
F(2)  
```

Example: DNA pattern matching
Matrix Multiplication (parenthesization)

\[
\begin{bmatrix}
A_1 & A_2 & A_3
\end{bmatrix}
\]

10x100 100x5 5x50

\[\text{need } nxm \times m \times k \text{ scalar muls}\]

Given a set of matrices \(A_1, A_2 \ldots A_n\) to be multiplied in that order. Find the parenthesization that minimizes the \# of scalar muls.

Naïve Approach

\[
P(n) = \sum_{k=1}^{n-1} P(k)P(n-k) = \mathcal{O}(n^{3.5})
\]

Find \(A_1 \cdot A_2 \cdot \ldots \cdot A_j\)

\[
m \cdot \sum_{i=1}^{j} \emptyset
\]

\[
m \cdot \min_{i < k < j} \left( m \cdot \sum_{k=i+1}^{j} P_i P_k P_j \right) \text{ if } i < j
\]
Dynamaxc

\[ O(n^2) \times O(n) = O(n^3) \]

square per square
LONGEST COMMON SUBSEQUENCE

Given two strings

\[ X = x_1 x_2 \ldots x_m \] \[ Y = y_1 y_2 \ldots y_n \]

and you want to find longest sequence of not necessarily consecutive characters but in that order

spring temp
pioneer

If \( m \leq n \)

\[ n \cdot 2^m \quad O(n \cdot 2^m) \text{ Very Bad} \]

THM: Let \( Z = z_1 z_2 z_3 \ldots z_k \) be a LCS of \( X \) and \( Y \)

A) if \( x_m = y_n \) then \( Z_{k-1} \) is LCS of \( X_{m-1} \) and \( Y_{n-1} \)

B) if \( x_m \neq y_n \) and \( z_k \neq x_m \) then

\[ Z \text{ is a LCS of } X_{m+1} \text{ and } Y_n \]

C) if \( y_n \neq x_m \) and \( z_k \neq y_n \) then

\[ Z \text{ is LCS of } X_m \text{ and } Y_{n-1} \]
Proof for A)

Assume A) is not true

\[ |Z_{k}^{'} U Z_k | > |Z_{k-1} U Z_k| \]

LCS For \( X \neq Y \) longer than \( Z \), a contradiction!

\[
C(\vec{x},\vec{y}) = \begin{cases} 
\emptyset & \text{if } \vec{x} = \emptyset \text{ or } \vec{y} = \emptyset \\
C[\vec{x} - 1, \vec{y} - 1] + 1 & \text{if } \vec{x} \neq \emptyset, \vec{y} \neq \emptyset, X = Y \\
\max \{ C[\vec{x} - 1, \vec{y}], C[\vec{x}, \vec{y} - 1] \} & \text{if } \vec{x} \neq \vec{y} 
\end{cases}
\]

Example:

\[
\begin{array}{cccccccccccc}
\text{sub} & a & m & p & u & t & a & t & e & c & o & n \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
4 & 3 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
\end{array}
\]
Binomial Coefficient \[ C(n, k) = \binom{n}{k} \]

\[ C(n, 0) = 1 \quad \& \quad C(n, n) = 1 \] 

Use a table to memorize.

Optimal Polygon Triangulation

You are given a convex polygon with vertices \( v_0, v_1, \ldots, v_n \) where \( v_0 = v_n \) and you want to divide into triangles using chords minimizing the Euclidean Distance.

\[
\min \sum \omega(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i| \]

of all triangles

Matrix Multiplication = TRIANGULATION

Table \( O(n^2) \) Time \( O(n^3) \)

\[
t_{i \leq j} = \begin{cases} \emptyset & \text{if } i = j \\ \mathbf{\nabla} & \text{if } i < j \end{cases}
\]

weight of optimal triangulation

For vertex

\[
\{ \begin{array}{ll} t_{i \leq j} = t_{i \leq k} + t_{k + 1, j} + \omega(\Delta v_i v_k v_j) & \text{if } i < j \\ \end{array} \}
\]
GREEDY ALGORITHM

- Overlapping Optimal Sub Problems

- Greedy Principle: at every step, do greedy selection to move to the next step

Notes: Good for minimization and maximization problems
Based on Theory of Methods
Good Approximate solutions to NP Complete Problems

Optimal Class Scheduling Problem

1. A single room and a set of classes to schedule
Maximize the # of classes to be scheduled.

Algorithm: Sort by finish time, select the one finish the earliest. O(nlogn)

Proof for optimality: Assume a contradiction that greedy
C₁, C₂, ..., Cₙ is not optimal but g₁, g₂, ..., gₖ is optimal

\[ C₁, C₂, \ldots, Cₙ \rightarrow \]
\[ g₁, g₂, \ldots, gₙ, \ldots, gₖ \]
Example: A thief that enters a store with \( n \) items. Item \( i \) cost \( c_i \) and weight \( w_i \). The thief can take a maximum of \( w \).

1. 0-1 version, take or not take [dynamic]
2. Functional, can take a partition [greedy]

Problem: maximize the profit

\[
C[l, w] = \begin{cases} 
\emptyset & \text{if } w < 0 \\
C[l-1, w] & \text{if } w \geq w \\
\max(C[l-1, w], C[l, w - w_i] + v_i) & \text{if } w_i \leq w
\end{cases}
\]

Assume the item is sorted in values

\( O(n) \)
Huffman encoding for data compression

A file with frequency of characters: \( T(B) = f(c) \)

<table>
<thead>
<tr>
<th>f(c)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>48%</td>
<td>13%</td>
<td>12%</td>
<td>16%</td>
<td>9%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Greedy Principle: Assume optimal tree where the two lowest frequency keys \( x \) \& \( y \) are the leaves in the deepest of the tree.

Optimal Substructure: Suppose you get an optimal tree, building on top of it gives another optimal tree.
Lemma 2.1 Suppose $C$ is the optimal code for $S$, $p_1, p_2$ and $l_1, l_2$ are the probabilities and code lengths of messages $s_1$ and $s_2$, respectively. Then $p_1 > p_2 \Rightarrow l_1 \leq l_2$.

Proof. Suppose $p_1 > p_2$ and $l_1 > l_2$, we swap the code words for $s_1$ and $s_2$, and get a new code $C'$. The length of $C'$ is $l_a(C') = l_a(C) + p_1(l_2 - l_1) + p_2(l_1 - l_2) = l_a(C) + (p_1 - p_2)(l_2 - l_1) < l_a(C)$. This contradicts the optimality of code $C$. $\blacksquare$

Lemma 2.2 Without loss of generality, the two messages of smallest probability occur as siblings in the code tree for an optimal code.

Proof. Given the code tree for an optimal code, we will show that it can always be modified without increasing the average code length so that the two smallest probability nodes are siblings. From Lemma 2.1, the smallest probability node must occur at the largest depth in the code tree. Note that the sibling of this node is also at the same depth. Now the sibling can be swapped with the second smallest probability node to obtain a code tree of the desired structure. This transformation does not increase the average code length. $\blacksquare$
Graph Algorithms

\[ G = (V, E) \]
\( V \): vectors  \( E \): edges

\[ E = O(V^2) \] worst case clique

Breadth First Search

Time \( O(V+E) \)

Depth First Search

Discovery / Finish

Use a stack

1. Create Depth First Forest

Time \( O(V+E) \)

Edge classifications on a DFS:

- Tree edges
- Back edges: to an ancestor
- Fund edges: to a descendant
- Cross edges: between different trees
**DFS Algorithm**

```plaintext
preorder(node v) {
    visit(v);
    for each child w of v
        preorder(w);
}
```

```plaintext
dfs(vertex v) {
    visit(v);
    for each neighbor w of v
        if w is unvisited
            dfs(w);
        add edge vw to tree T

}
```

**Theorem:** In the DFS Forest of an undirected graph every edge is a tree edge or a back edge.

- **Forward edges**
- **Back edges**
- **Cross edges**

**DFS Parenttheses THM (Well Parenthesized)**

```
  s 10 11 t 16
  2 3 6 7 8
  14 15
  4 5
```

Two edges are either disjoint or contained fully.

Every two vertices $u, v$ you have

```
Ld(u), f(u), Iu (u), f(u),
```

are either completely disjoint or one contained within the other.
Assume $d[Q] < d[V]$ compare

$d[V] \neq F(Q)$

$\geq$

**TOPOLOGICAL SORT**

Given a DAG find a total order of the vertices

Directed Acyclic Graph

Note: topological sort is not unique
Strongly Connected Components

In a directed graph, a SCC is a subset \( V' \subseteq V \) of \( V \) such that for every \( u, v \in V' \), there exists a path \( u \sim v \sim u \).

Algorithm:
- Run DFS
- Transpose \( A \Rightarrow A^T \)
- Run DFS \((A^T)\) but pick vertices in decreasing finish time
- Each Forest Tree, \( G \) is a SCC

Running time \( O(V+E) \) with adjacency matrix
Minimum Spanning Tree

Given a weighted \( G(V,E) \) with weights on edges.

Find a tree that minimizes \( \sum_{(u,v) \in T} w(u,v) \)

MST is not unique - weights can be gain, costs, distance etc.

Given a partial MST \( T_p \), a safe edge is an edge that can be added to \( T_p \) and give a "bigger" partial MST.

We define a cut \((U_1,U_2)\) on \( G = (V,E) \) to be partition of the graph s.t. \( U \cup U_2 = V \) and \( U \cap U_2 \neq \emptyset \).
Given a cut, an edge \((a, v)\) crosses the cut if \(a \in V_1\) and \(v \in V_2\) (or vice versa).

An edge crossing a cut is called light if it has the minimum weight along all edges crossing the cut.

**Theorem:** Given a cut, a light edge is a safe edge.

Let \(A\) be part of MST and consider the cut \(V_1 = A + \bar{V}_2 = U - A\).

Let \((u, v)\) be a light edge crossing the cut. Assume towards a contradiction it is not a safe edge but \((x, y)\) is, where weight \(w(u, v) < w(x, y)\).

Add to Final MST \((u, v)\) creating a cycle, and break cycle by removing \((x, y)\). You get a new tree with lower weight, contradicting that the original was part of MST.

*Proof (Greedy)*

- Insert every key in queue \(Q\) with \(\infty\) weight
- Decrease \((Q, \text{root}, \varnothing)\)
- while (Q ≠ ∅)
  Extract Min(Q) → q
  if v adjacent
    if w(q, v) < key(v)
      decrease (q, v, w(q, v))

Can use a heap to implement, running time

\[ E \log_2 u + u \log_2 u = E \log_2 u \]

with Fibonacci heap \( O((u \log_2 u) + E) \)

**Single Source Shortest Path**

Given a directed \( G = (V, E) \) with weight function

\[ S(u, v) = \begin{cases} 
\min \{ w(p) : u \xrightarrow{p} v \ \text{if} \ p \in E \} \\
\infty & \text{otherwise}
\end{cases} \]

\begin{itemize}
  
Notes:
  
- Numerate value: weights, cost, profit
- Can have negative weights but not cycle

\end{itemize}
Dijkstra: only positive weights (Greedy)
Bellman-Ford: works for negative weights & detects negative cycles

SSSP: Single Source Single Destination
Single destination SPs, all pair SPs (Dynamic Programming)

\[ \delta(u, v) = \text{SP } u \xrightarrow{\text{SP}} v \]

The algorithms will always keep track an estimate \( \delta(u, v) \) of SP. Always \( \delta(v) > \delta(u, v) \) and they will keep improving the estimate.

RELAX \((u, v, w(u, v))\)

If \( \delta(v) > \delta(u) + w(u, v) \)

\[ \delta(v) = \delta(u) + w(u, v) \]
\[ \pi(v) = u \]

Initially

\[ \delta(v) = \infty \]
(optimal substructure)

Every sub-path of a shortest path (SP) is also a SP

(Triangle Inequality) For relaxing edge \((u, v)\)

\[ S(s, u) + w(u, v) + d[v] \leq d[u] + w(u, v) \]

(Upper Bound) As you relax edges always \(d[v] \geq S(s, u)\)
and once \(d[v] = S(s, u)\) it changes

(No Path) if \(S[s, v] = \infty\) then \(d[v] = \infty\)

(Convergence) if \(S \not\supseteq Q \rightarrow u \in Q\) as a SP and \(d[v] = S(s, v)\)
after relax \((u, v)\) you get \(d[v] = S(s, u)\)

Path Relaxation:

Let \(p = \langle u_0, u_1, \ldots, u_k \rangle\) be a SP. If we relax \(\langle u_0, u_1, \rangle, \langle u_1, u_2 \rangle \ldots \langle u_{k-1}, u_k \rangle\) to that order even mixed with other relaxations in between, at the end we get

\[ d[u_k] = S(s, u_k) \]
Suppose \( u \rightarrow x \) not SP, but there is another one \( \rightarrow \text{cut & } \text{paste, you get a shorter SP} \)
\( u \rightarrow u \) a contradiction

Let \( u \) be the first vertex where relaxing \((u, v)\) makes
\( d[u] < s(s, v) \)
\[ d[u] < s(s, v) \leq s(s, u) + \omega(u, v) \leq d[u] + \omega[u, v] = d[u] \]

A contradiction

No path: \( d[u] \geq s(s, v) \)

Convergence: \( d[u] \leq d[u] + \omega(u, v) \)
\[ = s(s, v) \text{ by optimal substructure} \]

\( d[u] \geq s(s, v) \neq d[u] = s(s, v) \)

Run induction over the index can prove

\( u_0 \rightarrow u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4 \cdots u_k \)
Bellman-Ford Algorithm works on negative weights and negative cycles

for $i \in [1, |V|-1]$

for all edges $(u, v) \in E$

 relax$(u, v)$

if $d[S] > d[S] + w(u, v)$ return false/negative cycle

Proof for correctness: if no negative cycle

- Find SPs due to path relaxation

- $d[S] \leq S(s, u) \leq S(s, u) + w(u, v) = d[u] + w(u, v)$

if negative cycle $v_0, v_1, \ldots, v_k$ then exist

Contradiction

$d[v_1] \leq d[v_0] + w(v_0, v_1)$

$d[v_2] \leq d[v_1] + w(v_1, v_2)$

$\vdots$

$d[v_k] = d[v_0] + w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_k-1, v_k)$

$0 \leq \sum_{i=0}^{k-1} w(v_i, v_{i+1})$ no negative cycle
Dijkstra / non-negative weights / generation of BFS to weights

\[ S = \emptyset; \quad Q = V \text{ (priority queue)} \]

while \( Q \neq \emptyset \)

\[ u = \text{extract-min}(Q) \]

\[ S = S \cup \{u\} \]

\[ O(\log V) \]

\( u \) vertex adj to a

\[ \text{relax}(u, v) \]

Correctness: Need to show that when \( u \) is added into \( S \), \( d[u] = s(u) \)

By contradiction, let \( u \) be the first vertex where \( d[u] \neq s(u) \)

and also assume \( s(s, u) \neq \infty \Rightarrow \exists \text{sp from } s \xrightarrow{sp} u \)

Let \( y \) be the first on the sp
to \( u \) which is not in \( S \)

Let \( x \) be the vertex in \( S \) where \( x \rightarrow y \)

Decompose \( s \xrightarrow{sp} u \Rightarrow s \xrightarrow{sp} p \rightarrow y \xrightarrow{sp} x \)

Claim \( d[y] = s(s, y) \) when \( u \) added into \( S \) since \( u \) is the first added in \( S \) where \( d[u] \neq s(u) \)

For \( x \) it holds, that is \( d[x] = s(s, x) \) and by convergence, it holds for \( y \), that is \( d[y] = s(s, y) \)

we got \( d[y] = s(s, y) \leq s(s, u) \leq d[u] \)

upper bound
But you are outside $s$ and $u$ is selected

$\Rightarrow d[u] < d[y]$

$\Rightarrow d[w] = d[y]$

$\Rightarrow 8(s, u) = d[w]$

Single Source DAs

- Topologies Sort Vertices $O(V+E)$
- Relax Edges in that topological sort order
- Correctness due to path relaxation
Difference Constraints

\[ A_{mn} \quad B_{nx1} \quad C_{nx1} \]

\[
\begin{bmatrix}
  a_{11} & \cdots & a_{1n} \\
  \vdots & \ddots & \vdots \\
  a_{m1} & \cdots & a_{mn}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\leq
\begin{bmatrix}
  b_1 \\
  \vdots \\
  b_n
\end{bmatrix}
\]

want a solution for \( \bar{X} \) that minimizes or maximizes objectives function

\[
\begin{bmatrix}
  c_1 \\
  \vdots \\
  c_n
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}
\]

- Ellipsoid
  - Poly But runs slow

- Simplex
  - Exp But Fast

has one 1
one -1

- \( b \) unknown introduce a vertex
- for \( x - y < b \) introduce \( y \rightarrow x \)

- Introduce a pseudo source with edges pointing to all vertices w/ weight zero

- Run Bellman-Ford
  - SPs are the solutions
- If neg cycle \( \Rightarrow \# \text{ no feasible solution} \)

Example

\[
\begin{align*}
    x_1 - x_2 &\leq 5 \\
x_1 - x_3 &\leq 6 \\
x_2 - x_4 &\leq -1 \\
x_3 - x_4 &\leq -2 \\
x_4 - x_1 &\leq -3
\end{align*}
\]

- \( \# \text{ neg cycle} \Rightarrow \# \text{ gives correct answer} \)

\[
\begin{align*}
    s(s, y) &\leq s(s, x) + \omega(x, y) \\
y &\leq x + b \\
y - x &\leq b
\end{align*}
\]

- \( \exists \text{ neg cycle} \Rightarrow \# \text{ no feasible solution} \)

\[
\begin{align*}
    x_0 - x_1 &\leq b_1 \\
x_1 - x_2 &\leq b_2 \\
&\vdots \\
x_{k-1} - x_0 &\leq b_k
\end{align*}
\]

\[
\phi \leq \Sigma \text{ cycle} \]

A contradiction
All Pairs Shortest Paths

Suppose weights are stored in a matrix \( D^{(0)} \)

\[
D^{(k)} = \text{distance from every vertex by following at most } k\text{-edges}
\]

We would like to compute \( D^{(n-1)} \)

\[
D^{(n-1)}_{i,j} \rightarrow \text{the distance} = m_{\text{min}}
\]

the shortest distance between vertices \((i,j)\) w/ at most \( m \) edges

\[
m \leq n - 1
\]

\[
= m_{\text{min}} \left\{ d^{(m-1)}_{i,k} + w(k,j) \right\} \text{ because } w_{ij} \neq 0
\]
For \( \ell = 1 \ldots n \)
For \( j = 1 \ldots n \)
\( d_{\ell j} = \infty \)
For \( k = 1 \ldots n \)
\( d_{\ell j} = \min \{ d_{\ell k} + w_{kj} \} \)

Need run these \((n-1)\) times to get \( D^{(1)} \ldots D^{(n-1)} \)
Total \( O(U^4) \)

Matrix: \( \text{mat} (A, B) \rightarrow C \)
For \( \ell = 1 \ldots n \)
For \( j = 1 \ldots n \)
\( C_{\ell j} = 0 \)
For \( k = 1 \ldots n \)
\( C_{\ell j} = C_{\ell j} + A_{\ell k}B_{kj} \)

Strassen: \( O(n^{2.81}) \)

The best performance so far for
\( D^{(1)} \rightarrow D^{(2)} \rightarrow \ldots \rightarrow D^{(n)} \)

Need run this \((n-1)\) times to get
\( D^{(1)} \ldots D^{(n-1)} \)
Total \( O(U \cdot U^{2.81}) \rightarrow O(U^{2.81} \cdot \log U) \)
Maximum Flow

\[ \text{Observation} \]
- $\theta$ vertex - $\{s, t\}$

\[ \Sigma \text{Flow Outgoing} \]
- II

\[ \Sigma \text{Flow Incoming} \]

Defn: Directed weighted graph with two special vertices, a source and a sink $t$. Weights are edge capacities.

Imagine: Edges are pipes & tunnel through them you can transfer fluid from source

Some Definitions
- Directed, connected, positively weighted graph $G$ where weights denote capacities $(u, v)$
If \((u, v) \in A\) then assume it does exist with capacity zero we have source/sink flow network

**Capacity constraint:** \(\forall (u, v) \in E \ f(u, v) \leq C(u, v)\)

**Symmetry:** \(\forall (u, v) \in E \ we \ have \ F(u, v) = -F(v, u)\)

**Flow Conservation:** \(\forall u \in S, t \in T \ we \ get \ \sum_{v \in V} f(u, v) = \phi\)

no leakage, net flow \((u, v) = F(u, v)\)

**Problem:** Maximize \(f = \sum_{s,t} f(s, t)\)

**Observation:** Multi-source and Multi-sink

![Diagram](attachment:image.png)

add pseudo \(s\) and \(t\) with \(\infty\) capacity

**Def:** define flow between sets of vertices as we did between vertices

\[F(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)\]
Lemma

\[ F(X, Y) = \emptyset \]

\[ F(\bar{X}, Y) = -F(Y, \bar{X}) \]

\[ F(X \cup \bar{X}, \omega) = f(\bar{X}, \omega) + F(\bar{X}, \bar{\omega}) \]

\[ F(\omega, \bar{X} \cup \bar{Y}) = f(\bar{\omega}, \bar{X}) + F(\bar{\omega}, \bar{Y}) \]

Ex: \[ F(1) = F(s, \bar{v}) = F(u, t) \]

"All unit leaving source they do go to sink"

\[ F(1) = F(s, u) = F(u, v) - F(u - s, v) \]

\[ = F(\bar{u}, v) + F(u, u - s) \]

\[ = F(u, t) - F(u, u - s - t) \]

\[ = F(u, t) + F(u - s - t, v) \]

\[ = F(u, t) \]
Residual Capacity of edge \( C_{ef}(u,v) \)

\[ C_{ef}(u,v) = C(u,v) - F(u,v) \]

Residual Network \( G_{ef}(V, E_{ef}) \)

A graph on some set of vertices, with edge-set

\[ E_{ef} = \{ (u,v) \in E : \text{if } C(u,v) > \emptyset \} \]

Augmenting Path in \( G_{ef} \): a simple path \( s \rightarrow t \) on \( G_{ef} \)

Residual Capacity of an augmenting path.

\[ G^*(P) = \min \{ G_{ef}(u,v) \text{ for } (u,v) \in P \} \]

Lemma:

If \( F \) is a flow on \( G_{ef} \) then

\[ (F+F')(u,v) = F(u,v)+F'(u,v) \] is a flow on \( G \)

with value \( |F+F'| = |F| + |F'| \)
Skew Symmetry: \((F+F')(u,v) = F(u,v) + F'(u,v)\)
\[= -F(u,v) - F'(u,v) = -(F+F')(u,v)\]

Capacity Constraints: \(F'(u,v) \leq C_f(u,v)\)
\[(F+F')(u,v) = F(u,v) + F'(u,v) \leq F(u,v) + C_f(u,v)\]
\[= F(u,v) + C(u,v) - F(u,v) = C(u,v)\]

Flow Conservation
\[\sum_{u \in V} (F+F')(u,v) = \sum_{u \in V} F(u,v) + \sum_{u \in V} F'(u,v)\]
\[= 0\]

Size of New Flow: \(\sum F(s,v) + \sum F'(s,v) = |F| + |F'|\)

Ford - Fulkerson
Initialise \(F(u,v) = 0\)
repeat
- Compute \(G_F\)
- Find an augmenting graph \(PA\) in \(G_F\)
- Add the residual capacity of \(PA\) in residual Flow
- Until \exists\ augmenting path in \(G_F\)
Max Flow: Directed Weighted $C(u, v) =$ capacity graph of no $(u, v) \in E \Rightarrow C(u, v) = \emptyset$

Capacity Constraint $f(u, v) \leq C(u, v)$

Skew Symmetry $f(u, v) = -f(v, u)$

Flow Conservation $\sum_{v \in V} f(u, v) = \emptyset \quad u \in \{s, t\}$

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

$|F| = 23$

Capacity of Residual Edge

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

Residual Network

$G_f(U, E_f) \quad G_f(u, v) > \emptyset$
Augmenting Path $P$: A sample path $s \xrightarrow{\pi} t$ on $G_F$

Residual Capacity $P$:
$C_F(P) = \min \{ C_F(u, v) : (u, v) \text{ edge on } P \}$

**Lemma:** Let $P$ path on $G_F$. Then

$F_p(u, v) = \begin{cases} C_F(P) & \text{if } (u, v) \in P \\ -C_F(P) & \text{if } (v, u) \in P \\ 0 & \text{otherwise} \end{cases}$

is a flow where $|F| = |C_F(P)| > 0$

To Prove: Prove the three properties one by one

**Corollary:** $|F| + |F_p| = |F_{new}| > |F|$
1) If edge \((u, v)\) set \(f(u, v) = f(v, u) = 0\)

2) while \(\exists\) augmenting path in \(G_f\) : \(S \rightarrow T\) \(\rightarrow O(E)\) with BFS

3) \(C(p) = \min f(u, v) : (u, v) \in p\)

4) For every \((u, v) \in p\) do
   \[
   F(u, v) = F(u, v) + C(p)
   \]
   \[
   F(v, u) = -F(u, v)
   \]

Proof of Correctness

Lemma: Let \(f\) be a Flow and \((S, T)\) a cut. The \(F(S, T) = |F|\)
\(F(S, T) = F(S, U) - F(S, S) = F(S, U) - F(S, S)\)
\(= F(S, U) = |F|\)

Corollary: \(\exists\) Flow \(f\): \(|F| \leq C(S, T)\) for any cut \((S, T)\)

\(|F| = F(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = C(S, T)\)

MAX FLOW MIN CUT THEOREM

If flow on \(G\), following statements are equivalent
1) For a max-flow in G

2) Gf has no augmenting path

3) |F| = c(S, T) for some cut (S, T)

1) \Rightarrow 2) because you can add it & increase value of flow

2) \Rightarrow 3) Since Gf has no AP, let S = \{vertices reachable on Gf from s\}

Clearly (S, V-S-T) is a cut.

Now Every (u, v) st. u \in S \& v \in T

we get \ \ F(u, v) = c(u, v) \iff \Sigma F(u, v) = \Sigma c(u, v)

\iff |F| = F(S, T) = c(S, T)

3) \Rightarrow 1) By definition |F| \leq c(S, T) by def, because its equal of has to be maximum.

Lemma: Value MAX_FLOW = CAPACITY of MIN-CUT

|F| \leq c_1 \leq c_2 \leq c_3 \ldots \leq c_k

Edmonds-Karp

In (6.e.2) select the shortest AP (CUE-E) of minimum length
Lemma: if you run E + K algorithm, the SP $S_F(s, u)$ to every vertex $u \in G_F$ monotonically increases.

Proof:
Let $u$ be the first that between Flow Augmentations, $F \& F'$, it decreases, that is, $S_F'(s, u) < S_F(s, u)$, and wlog assume this is the one closest to $s$ that the decrease happens.

Consider SP on $G_F': s \rightarrow u \rightarrow v$
We know $S_F'(s, u) > S_F(s, u)$

CASE: $F(u, v) \leq C(u, v)$ on $G_F \Rightarrow G_F$ contains $u \rightarrow v$
edge $e \in E_F: S(s, u) < S_F(s, u) + 1 \leq S_F'(s, u) + 1$

$= S_F'(s, u)$

$\Rightarrow$ Contradiction !

CASE: $F(u, v) = C(u, v) \Rightarrow u \rightarrow v \notin G_F$ but $u \rightarrow v \in G_F'$

$\Rightarrow AP_F$ traverses

$\Rightarrow S_F(s, u) = S_F(s, u) - 1 \leq S_F'(s, u) - 1$

$= S_F'(s, u) - 2$ a contradiction
THM (Correctness & Run Time)

Total # of Flow Augmentations is $O(\text{UE})$

Proof: Given Augmenting Path, call the edge on path that determines capacity as critical edge. Suppose $u \rightarrow v$ is critical edge. We show that for two consecutive time $u$ becomes critical the SP to $u$ increases by 2. If this is true, every left side on edge can become critical $U/2 = O(v)$ times $\Rightarrow$ you got $O(E)$ edges $\Rightarrow O(\text{UE})$ iterations.

when $u \rightarrow v$ becomes critical you got

$S_F(s, u) = S_F(s, u) + 1$

& reverse direction. For $v \rightarrow u$ become critical again, some Flow $F'$ must be pushed through $v \rightarrow u$ later on & before.

$S_{F'}(s, u) = S_{F'}(s, u) + 1 \leq S_F(s, u)$

$\Rightarrow S_{F'}(s, u) > S_F(s, u) + 1 = S_F(s, u) + 2$
BIPARTITE MATCHING MAXIMUM

Matching ≤ E

\[ \text{St } \theta u \in V \text{ at most one edge point from } M \text{ exists} \]

Maximal A-1 C-3

Maximum B-1 C-2 E-3

How could we solve the problem using Max Flow?

Lemma:

a) \( M = \text{matching } \Rightarrow \exists \text{ flow w/ } |F| = |M| \)

b) F in Gnew \( \Rightarrow \exists \text{ matching } M \text{ w/ } |F| = |M| \)

a) is trivial to prove

b) define

\[ M = \{ (u, v) : u \in L \& \ v \in R \} \]
$M = F(L, R) = F(L, V) - F(L, L) - F(L, T) - F(L, S)$

$= -F(L, S)$

$= F(S, L) = |F|$
PARALLEL ALGORITHMS

**P**RAM Model

\[ \begin{array}{cccc}
P_0 \\
\vdots \\
P_{(n-1)} \\
\end{array} \rightarrow \text{RAM} \]

- EREW -
- ERCW
- CREW
- CRCW

Total Time \( T(n) \)

\# processors \( P(n) \)

\[ \text{Cost}(n) = T(n) \times P(n) \]

\[ \text{Work} \leq \text{Cost}(n) \]

The following statements are equivalent

1) The algorithm requires \( T(n) \) time with \( P(n) \) procs
2) Has cost \( C(n) \) and time \( T(n) \)
3) time \( O\left(\frac{\text{Cost}(n)}{p}\right) \) for any \( \# \) of procs \( p \leq P(n) \)
4) Also time \( O\left(\frac{\text{Cost}(n)}{p} + T(n)\right) \) \( \Theta \# \) procs \( p \)

1) \( \Rightarrow \) 2) by definition

2) \( \Rightarrow \) 3) Every step \( \frac{P(n)}{p} \cdot T(n) = \frac{P(n)}{p} \cdot \frac{\text{Cost}(n)}{P(n)} = \frac{\text{Cost}(n)}{p} \)

3) \( \Rightarrow \) 4) \( p \leq P(n) \) then 3) applies

\[ p > P(n) \text{ time } T(n) \]
4) ⇒ 1) \( P(n) = p \)

**Optimal:**

\[ P(n) - T(n) = \text{cost} (n) = \Theta (T_{\text{seq}} (n)) \]

**Optimal Speed Up:**

\[ P(n) = \frac{T_{\text{seq}} (n)}{T(n)} \]

**Lemma:** Suppose optimal algorithm w/ \( P(n) \) procs, we can create a new algorithm with \( p < P(n) \) procs

\[ T_p (n) = \frac{\text{cost} (n)}{p} \] which is also optimal.

The new algorithm \( p \frac{\text{cost} (n)}{p} = \text{cost} (n) = \Theta (T_{\text{seq}} (n)) \)

**Strongly Optimal For Problem:** The less time among all optimal algorithms.

**NC-Class “efficiently parallelizable problems”**

\( O(\log^{O(1)} n) \) time w/ \( O(n^{O(1)}) \) procs

**Brent’s Thm:** Assume PRAM algorithm w/ work \( (n) \) & time \( (n) \)

If at each step \( c \)

- Constant time to identify the \# of ps needed.
to allocate the processors for those ops.

Then we can run the algorithm in \( O(T(n) + \frac{\text{work}(n)}{p}) \) time using \( p \) processors.

\[
T(n) + \frac{\text{work}(n)}{p} = T(n)
\]

Example: \( \log n \) time with \( O(n) \) work \( O(n) \) proc.

\[
T(\text{seq}) = O(n)
\]

\[
\frac{n}{\log n} \ \text{procs} \times (\log n = O(n))
\]

Proof: Let \( \text{work}_i \) = amount of work at step \( i \):

\[
\sum_{i=1}^{T(n)} \left( \frac{\text{work}_i}{p} + 1 \right) = O(T(n) + \frac{\text{work}(n)}{p})
\]
Prefix Sums

\[\begin{array}{cccccccc}
5 & 3 & 2 & 4 & 1 & 9 & 8 & 5 \\
\end{array}\]
\[\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{array}\]

1. \(T_{seq} = O(n)\)

2. Sub-Optimal:
   - \(O(n)\) procs
   - \(O(\log n)\)
   - Cost = \(O(n \log n)\)
   - Work \(\sum_{k=1}^{n} \frac{n}{2^k} \leq 2n = O(n)\)

Can still do it in \(O(\log n)\) time with

\[
\frac{\text{work}(n)}{T(n)} = \frac{n}{\log n} \text{ procs.}
\]

\(\log n\) time \(\log n / \log n\) procs

Break original array in \(\log n\) pieces of \(\log n\) elements each. Allocate \(\log n / \log n\) procs to find sum in \(\log n\) time. Use sub-optimal algorithm on \(\log n / \log n\) partial sums.

\(\text{time} (\log (\log n)) = O(\log n - \log \log n) = O(\log n)\)
4.2 Finding the Max in $O(1)$ time

Recall that the common CRCW PRAM allows several processors to write to the same memory location, as long as they all write the same value.

**Theorem 1** We can find the max of $n$ elements in $O(1)$ time on a common CRCW PRAM with $n^2$ processors.

To show this, assume that we have $n^2/2$ processors, and let $m$ be an auxiliary boolean array of size $n$.

**Step 1.** For $i = 1, \ldots, m$, set $m[i] := $ true.

**Step 2.** For all $i$ and $j$ in parallel, where $1 \leq i < j \leq n$, if $A[i] < A[j]$, set $m[i] := $ false.

Now $m[i]$ is true exactly when $A[i] \leq A[j]$ for all $j$, i.e. when $A[i]$ holds a copy of the max value.

**Step 3.** For $i = 1, \ldots, n$, if $m[i] = $ true then set $\text{max} = A[i]$.

Even if several locations hold a copy of the same value, all values written to $\text{max}$ will be the same.

This algorithm runs on a CRCW PRAM with $n^2/2$ processors in $O(1)$ time. However, this algorithm is not optimal (since the problem can be solved sequentially in $O(n)$ time). Valiant proved that any $n$-processor CRCW PRAM algorithm requires $\Omega(\log \log n)$ time to find the max of $n$ elements. In fact, as we will see, $O(\log \log n)$ time is sufficient as well.

Notice that this discussion also shows that the CRCW model is strictly more powerful than the EREW and CREW models.
4.5 Accelerating cascades

We conclude our presentation with the presentation of a general technique that can be often used to improve the performance of a parallel algorithm. In our case, accelerating cascades produces a strongly optimal algorithm for the problem of finding the maximum.

The idea behind *accelerating cascades* is as follows: you have several algorithms to solve a problem, each with a different time/processor requirement. Typically, you have slower algorithms that are optimal, and faster algorithms that are non-optimal because they require too many processors. Each of these algorithms uses a sequence of steps to reduce the problem to a similar problem of smaller size. Start with the optimal but slower algorithm, and reduce the size of the problem. Then use a faster algorithm to reduce the size even more, and continue like this. In many cases, this will allow you to derive an algorithm which is both fast and closer to optimal.

For example, to find the max of elements we already described a couple of algorithms with different time/processor requirements:

1. Solve the problem optimally (sequentially) in $O(n)$ time with 1 processor.
2. Solve the problem optimally in $O(\log n)$ time with $O(n/\log n)$ processors using the balanced binary tree scheme (pairwise comparisons). Each step of this algorithm reduces the size of the problem by $\frac{1}{2}$.
3. Solve the problem in $O(\log \log n)$ time with $O(n)$ processors (Valiant’s algorithm).

We will now combine these algorithms to get an optimal $O(\log \log n)$ time one that has only $O(n)$ cost (i.e. uses $O(n/\log \log n)$ processors).

Here are two ways that you can do it:

1. Use (2) and (3). First we apply $\log \log n$ steps of the binary tree algorithm (2). There is $O(n)$ work here and, since each step reduces the problem size by $\frac{1}{2}$, it reduces the problem to one of size $N = n/2^{\log \log n} = n/\log n$. Now apply Valiant’s algorithm (3) to these elements: this requires $O(\log \log N) = O(\log \log n)$ time and work

   $$O(N \log \log N) = O\left(\frac{n}{\log \log n} \cdot \log \log \left(\frac{n}{\log n}\right)\right) = O(n).$$

   So the total time is $O(\log \log n)$ and work is $O(n)$. By Brent’s Theorem, this can be done with $O(n/\log \log n)$ processors.

2. Similarly, we can use (1) and (3) and $n/\log \log n$ processors. Partition the array into $n/\log \log n$ blocks of size $\log \log n$ each. Assign one processor to each block and (using (1)) each processor computes the max of its block of elements. This reduces the problem to one of size $n/\log \log n$ in $\log \log n$ steps. Now apply Valiant’s algorithm to these elements and find the max in $O(\log \log n)$ steps using only $n/\log \log n$ processors.

It can be shown [1] that any $n$-processor CRCW PRAM algorithm that uses only comparisons on elements of the array requires $O(\log \log n)$ time to find the max of $n$ elements. Therefore, both algorithms described above are strongly optimal comparison based parallel algorithms for this problem.
Intro. Theory of Computation

Computability & Complexity

\[ f(x) = -4x^3 + x^2 + 4x + 7 \] does there a solution?

Machine \iff Algorithm

Deal with languages from now on.

Problem = Sequence of 0's & 1's

Finite Automata

Regular Expression over a Finite Alphabet \( \Sigma \) is defined recursively as follows:

- The \( \varepsilon \) is a regular expression (ree)
- \( \emptyset \) is a re of \( r \) \( \in \) re
- If \( r \) & \( s \) are re's then \( r \cdot s \) (concatenation) is re
- If \( r \) & \( s \) are re's then \( r \cdot s \) (\( r^* \) or \( s^* \)) is re
- If \( r \) \( \in \) re, then \( r^* = \ldots \cdot r \cdot r \cdot (kleene star) \)

\[ r^0 = \varepsilon \] 20 times
$L(r) = \{ w : \text{strings produced by } r \}$

**Example:**

$L(011), (01)3 = 000, 01, 11, 103$

$L(011)3 = 0, 1, 01, 10, 11, \ldots \}$ = $\{ \text{any binary \# including empty strings} \}$

$L(011)*1 = \{ \text{any binary \# that ends with 1} \}$

digit $\Rightarrow 0, 1, 9$

letter $\Rightarrow a, z, A, Z$

symbol $\Rightarrow -, -, ^$ Af

$\text{letter} \mid \text{digit} \mid \text{symbol} \mid \text{letter}^* = \text{typical programming variables}$

Given a string $w$ we want to build a machine that recognizes whether $w \in L$?

$w \rightarrow \text{Machine L} \xrightarrow{\text{Yes}} w \in L$

$w \rightarrow \text{Machine L} \xrightarrow{\text{No}} w \notin L$
Deterministic Finite Automata

Σ: alphabet  \( K: \) finite set of states
s: start state  \( F: \) set of final states
\( \delta: \) transition function  \( \delta: K \times \Sigma \rightarrow K \)

Example:

\[ \Sigma = \{ a, b \}, K = \{ q_0, q_5 \}, S = \{ q_0, q_5 \}, F = \{ q_0, q_5 \} \]

<table>
<thead>
<tr>
<th>Present</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

Transition Table

Transition Diagram

\[ L(DFA) = \{ w : \text{any string } a_n b_m a^* \text{ with even number of } b's \} \]

\[ a^*(a^* b a^* b)^* a^* \]
\[ L = \{ w : \text{have at least one 1 and there is an even } \# \text{ of 0's following the last 1} \} \]

\[ L = \{ w : \text{sum of digits mod 3} = 0 \} \]

**Theorem:** The set of regular languages are those recognized by a DFA

\[ \text{DFA} \equiv \text{re} \]

**Example:** Compiler

<table>
<thead>
<tr>
<th>Program</th>
<th>DFA</th>
<th>PDA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lexical Analysis → Syntax Analysis → ...</td>
<td></td>
</tr>
</tbody>
</table>

Non-deterministic Machine

\[ \Sigma: \text{alphabet} \quad K: \text{finite set of states} \]

\[ s: \text{start state} \quad F: \text{set of final states} \]

\[ \Delta: \text{transition relation} \quad \Delta \subseteq K \times \Sigma \times F \times K \]
Non-determinism

\[ \Sigma = \{a, b, 5\} \]
\[ k = 5, 0, 1, 2, 3 \]
\[ 0 = 503 \]
\[ F = 925 \]

\[ (a \cup b)^* \]

\[ \text{Deffinition: No transistion from every symbol} \]
- \[ \varepsilon \] moves
- randomness

NFA are used to understand a problem. They are not real machine.

**THM:** The set of regular languages are those recognized by a DFA. NFA \( \equiv \) DFA \( \equiv \) r.e.

\[ L = \{0^n 5^n : n \geq 0\} \]
\[ n \leq k \text{ fixed} \]
\[ O(2^n) \text{ states} \]

\[ L = \{ww : w \in \{0, 1, 5\}^*\} \]

\[ L = \{w : w \text{ has equal \# of 0s \& 1s}\} \]

\[ L = \{ww^R : w^R : w \in \{0, 1, 5\}^*\} \]
Pushdown Automata (PDA)

```
\[
\text{Input Tape} \quad \downarrow \\
\text{Finite Control (DFA)} \quad \rightarrow \\
\text{Stack (Memory)}
\]
```

Context Free Grammars

- **U**: set of terminal & non-terminal symbols
- **T**: set of terminal symbols
- **P**: productions \(S \rightarrow (U-T)^*\) written as \(U-T \rightarrow U^*\)
- **S**: start symbol \( \subseteq U-T\)

Example:

- **U**: \(9, E, \text{num}, +, -, (, ) , \text{O}, \text{I} \ldots, \text{9, 5}\)
- **S**: \(TE3\)

- **P**: 
  \[E \rightarrow E+E \quad E \rightarrow E-E \quad E \rightarrow E \text{ (E)}\]
  \[E \rightarrow \text{num} \quad E \rightarrow -E \]
  \[\text{num} \rightarrow 0 \ldots \quad \text{num} \rightarrow 9\]
  
  \[E \rightarrow E+E \rightarrow -E+E \rightarrow -E+E -E \rightarrow -E+\text{num} -E \rightarrow -\text{num} + \text{num} -E \rightarrow \ldots \rightarrow -5 + 3 -2\]

- **L**: \( \subseteq \) a simple calculator's
1 \( S \rightarrow 0S11E \rightarrow 10^11^n : n \geq 3 \)
2 \( S \rightarrow aSaaSb1E \)
   \( S \rightarrow aSa \rightarrow aaSaa \rightarrow \ldots \)

**Theorem:** CFL \( \leftrightarrow \) PDA

**What about?**

\[ L = \{ \omega \omega : \omega \in \{0, 1\}^* \} \]

\[ L = \{ 0^i2^i2^j : \beta \leq i \leq j \leq k \} \]

*Cannot be recognized by PDA.*

---

**TURING Machine**

\[ \begin{array}{c}
\text{INPUT TAPE} \\
\text{S/I TAPE} \\
\end{array} \]

- Read a character
- May write on R/W Tape
- Move the heads left/right on the R/W and/or Input Tape

---

**Halting Problem:**

Given a machine and an input, will the machine ever terminate on this input?
Example of a TM that recognizes palindromes

i.e. Rose to vote son

Algorithm:

1) Copy input to r/w tape
2) Move input head to beginning
3) Move read head to end
4) Starting moving heads step-by-step declaring accept or reject

States = 9
   q_{start}, q_{copy}, q_{left}, q_{test}, q_{halt}

q_{start} = move head to left, switch to copy
q_{copy} = copy symbol of non-blank from input to r/w tape q_{left} at end
q_{left} = move head to left
q_{test} = it will move heads, opposite directions to check similar chars, declare accept/reject and move to q_{halt}
q_{left} = end
Computability & Complexity

- TM ≡ Algorithm ≡ Program
- Problem → 01001... → Yes → \( w \in L \)
  \[ x \in L \]
  \[ TM_x \]

- Optimization vs Decision Problem

Example: Max Flow in what is Max Flow in a graph? / Optimization

\[ \text{Max Flow}_{\text{opt}} = \text{Does } G \text{ have a Flow of size } k? \]

Not Flow k and
Not \( G \)
\[ \varepsilon = \varepsilon_0, 1^* \]

A TM decides a problem if after a finite of \( t \) of steps returns \( w \) answer yes or no
A TM accepts a problem if when it returns \( w \) yes or no the answer is correct, but as long as machine runs you don't know whether it will terminate or not.

Some things to prove

1. If \( L \) is decided, then \( \overline{L} \) is also decided

\[
P\text{co-NP} = P\text{as closed under complementation}
\]

\[
\begin{array}{c}
\text{TM}_L \\
\text{TM}_L \\
\text{TM}_L
\end{array}
\]

2. \( \exists \) problem that are undecidable

Halting: Is there a TM that can decide whether, given any machine \( M \) & input \( x \), \( (M, x) \), \( M \) has its on \( x \)?

Proof: Diagonalization
Complexity Class $P$

$P \equiv \{ L \in \mathbb{S}, \exists^* M : \text{TM that decides} \ L \text{ in poly-time} \}$

Thm: $P = \{ L \in \mathbb{S}, \exists^* M : \text{TM to accept} \ L \text{ in poly} O(n^c) \text{ time} \}$

Verification: Given an instance to a problem, and a candidate solution to problem, how "easy" is to verify if it is indeed a solution?

Algorithm $A$ verifies language $L$ if given instance $L$ there exists a certificate (witness) $y$ st. $A(x,y) = 1$ or $0$ (candidate solution)

Complexity Class $NP$

$NP \equiv \{ L \in \mathbb{S}, \exists^* \text{ certificate } y = O(|x|^c) \text{ and poly-time algorithm } A \text{ st. } A(x,y) = 1 \}$

Thm: $P \subseteq NP$

$NP \supseteq P$
\[ \text{co-NP} = \{ I \in \text{NP} \text{ then } \overline{I} \in \text{co-NP} \} = \{ L \in \Sigma^* : \exists y = O(1|x|^c) \in \text{poly} \ A(x,y) = \emptyset \} \]

**HAM-CYCLE (NP Class)**

Given a connected, undirected graph a simple cycle that traverses all vertices?

\[ \text{NP} \subset \text{co-NP} \]

\[ \text{NP} \cap \text{co-NP} = \text{P} \]

\[ \text{NP} = \text{co-NP} \]
Reducibility (informal)

Problem A can be reduced to problem B if there exists a translation algorithm that maps every instance of A into an instance of B, and these mapped instances back to instances of A.

Polynomial Time Reducibility

We say that $L$ is poly-reducible to $L'$ if there exists a function $f(x)$ such that:

- $x \in L$ if and only if $f(x) \in L'$

This is denoted as $L \leq_p L'$.

Theorem: If $L_1 \leq_p L_2$ and $L_2 \in P$, then $L_1 \in P$.
Language \( L \in \text{NP-Complete (NPC)} \) iff

- \( L \in \text{NP} \) (takes poly-time to verify)
- \( \exists L' \in \text{NP}, \ L' \leq_p L \) (NP-hard)

**THM:** \( \text{if } \exists L \in \text{NPC } \& \ 2 \in \text{P } \Rightarrow \text{P=NP} \)

**THM:** \( \text{NP=co-NP iff } \exists L \in \text{NPC st } \overline{L} \in \text{NP} \)

\( \Rightarrow \) Easy

\( \Rightarrow \) Let \( L \in \text{NPC st } \overline{L} \in \text{NP} \)

Peek any \( L' \in \text{NP} \)

\( L' \leq_p \overline{L} \) \( \Leftrightarrow \overline{L} \leq_p L \)

\( \Rightarrow \overline{L} \in \text{NP because } \overline{L} \in \text{NP} \)

\( \in \text{co-NP} \)

**NP-Complete Methodology**

To prove that \( L \in \text{NPC, do the following} \)

- Show that \( L \in \text{NP} \) (usually easy to show)
- Peek any known \( L' \in \text{NPC} \) and show \( L' \leq_p L \)
- anybody \( \leq_p L' \leq_p L \)
Cook's Thm (1971) Circuit - SAT \( \in \) NPC

Given circuit w/ OCN AND/OR/NOT gates & single output. Find an input vector that makes output "1".

(A) Circuit - SAT \( \in \) NP
(B) \( \exists \) NP problem \( \leq_p \) Circuit - SAT

Outline of Proof:

\[
\text{Circuit - SAT} \downarrow
\text{Formula - SAT} \downarrow
\text{3-SAT} \downarrow
\text{CLIQUE} \downarrow \text{Vertex Cover}
\]

\[
\text{HAM-CYCLE} \downarrow \text{travelling salesman}
\]
Formula-SAT: Given a formula $\phi$ with Boolean variables & connectives $\land, \lor, \neg, \iff, -, \rightarrow$ and $O(n)$ symbols all together, is there an assignment to the variables that make $\phi = 1$?

$$\phi = ((x \lor y) \rightarrow z) \iff (a \land b \land c \rightarrow d)$$

A) Show Formula SAT $\in$ NP

verification in poly time verified

B) Circuit-SAT can reduce to Formula-SAT

Circuit-SAT $\leq_p$ Formula SAT

Good enough to show an example

\[ \phi = x_9 \land ((x_1 \land x_2) \iff x_4) \land (x_5 \iff x_4) \land (x_6 \iff x_4) \land \\
((x_6 \lor x_3) \iff x_7) \land (x_8 \iff \neg x_5) \land (x_9 \iff (x_8 \land x_7)) \]
To prove language $L$ is NPC
- Show that $L \in$ NP (verifed in poly-time)
- Peck known NPC language $L'$ and show that $L' \leq_p L$ (NP-hard)

3-SAT

Given a set of clauses with three variables which is a conjunction of disjunctions. Find a satisfying assignment

$$\emptyset = \left( x_1 \lor x_2 \lor \overline{x}_3 \right) \land \left( x_4 \lor \overline{x}_1 \lor \overline{x}_3 \right) \land \left( \overline{x}_2 \lor x_3 \lor \overline{x}_5 \right) \land \ldots$$

2-SAT = poly time

CNF = Conjunctive Normal Form

Verification

QBF: PSPACE $\leq$ UP
Proof:

A) $\text{3-SAT} \in \text{NP}$ easy

B) $\text{Formula-SAT} \leq_p \text{3-SAT}$

Given a Formula $\phi$ I will create a 3-SAT instance st.

\[
\phi = (\neg (x_1 \rightarrow x_2) \lor \neg (\neg x_1 \leftrightarrow x_3) \lor x_4) \lor \neg x_2
\]

1) Build a parse tree for formula

\[
\phi = y_1 \lor (y_1 \leftrightarrow (y_2 \leftrightarrow \neg x_2)) \lor (y_2 \leftrightarrow (y_3 \lor x_4)) \lor (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \lor (y_4 \leftrightarrow (y_5 \lor x_4)) \lor (y_5 \leftrightarrow \neg y_6) \lor (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))
\]

replace with clauses
2) Build Characteristic

Function \( y_1 \) & Find Maxterms

<table>
<thead>
<tr>
<th>( y_1 ), ( y_2 ), ( x_2 )</th>
<th>( y_1 \Leftrightarrow (y_2 \land \neg x_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0</td>
<td>1</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>1</td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>( \phi )</td>
</tr>
<tr>
<td>0, 1, 1</td>
<td>0</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>0</td>
</tr>
<tr>
<td>1, 0, 1</td>
<td>0</td>
</tr>
<tr>
<td>1, 1, 0</td>
<td>1</td>
</tr>
<tr>
<td>1, 1, 1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \bar{\phi}_{\text{part}} = (\bar{y}_1 \land y_2 \land \bar{x}_2) \lor (y_1 \land \bar{y}_2 \land \bar{x}_2) \lor (y_1 \land y_2 \land x_2) \lor (y_1 \land y_2 \land \bar{x}_2) \]

\[ \overline{\phi}_{\text{part}} = (y_1 \land \bar{y}_2 \land \bar{x}_2) \land (\bar{y}_1 \land y_2 \land \bar{x}_2) \land \cdots \]

\( (x, y) = (x \lor y \lor \bar{p}) \lor (x \lor y \lor \bar{p}) \)

- You have to show reduction
- You have to show solution to \( L' \) \( \Leftrightarrow \) solution to \( L \)
- You have to show reduction takes polynomial time
CLIQUE (Complete Graph)

Decesion Problem

Given a graph G does it have a clique of size k?

A) CLIQUE ∈ NPC (Easy)
B) 3-SAT ≤p CLIQUE

\[ \phi = (x_1 \lor \bar{x}_2 \lor \bar{x}_3)^{\wedge} (\bar{x}_1 \lor x_2 \lor x_3)^{\wedge} (x_1 \lor x_2 \lor \bar{x}_3)^{\wedge} \]

ϕ has a solution \iff G has clique of size (\# clauses)

1) Introduce a vertex θ literal & "group" according to clauses
2) Connect variables between different groups of not complementing each other
**VERTEX COVER**: Given a graph, a VC is a set of vertices such that every graph edge is adjacent to at least one vertex from the cover.

Decision Problem: does G have VC of size k?

A) $VC \in NP$ (Easy)

B) $CLIQUE \leq_P VC$

\[ G \quad \overline{G} \]

\[ \begin{array}{c}
\text{CLAIM: } G \text{ has a clique of size } k \text{ off} \\
\overline{G} \text{ a VC of size } n-k \\
\text{The reduction is poly-time}
\end{array} \]
Assume Ham Cycle for the following Problem 6

Travelling Salesman Problem

Given a complete graph undirected w/ weights what's the lowest cost simple cycle?

Decision: Does G have a TSP of weight w?

A) TSP ∈ NP (early)

B) HAM CYCLE ≤p TSP

1) Given G to find a Ham Cycle, introduce the remaining edges to make Gnew a complete graph:

In $E_{new} = \begin{cases} 
\text{weight } \phi \text{ of edge existed in } G \\
\text{weight 1 of edge new introduced}
\end{cases}$

Does Gnew have TSP of weight $\phi$?
Remind: To prove L \in \text{NPc}

A) Show L \in \text{NPc}

B) Pick know L' \in \text{NPc} \& show L' \leq_p L

\text{HAM-CYCLE: Does there exist a simple cycle to traverse all vertices?}

A) Easy

B) 3-SAT \leq_p \text{HAM CYCLE}

does can never traverse all three cuter bi edges & traverse all inner ones
APPROXIMATION ALGORITHMS

Approximate solution w/ error bound $p(n)$

$$\max \left\{ \frac{C}{C_{\text{opt}}}, \frac{C_{\text{opt}}}{C} \right\} \leq p(n)$$

minimization \hspace{1cm} maximization

error bound

$$\left| \frac{C - C_{\text{opt}}}{C_{\text{opt}}} \right| \leq E(n)$$

$E(n) \leq p(n) - 1$

Different approximation

- Fixed error bound
- depends on $n$
- trade off
  - time vs error

(1992) Approximating CLIQUE is NPC
Approximation Vertex Cover

1. \( C = \emptyset, \quad E' = E \)
2. while \( E' \neq \emptyset \) do
   - randomly pick \((u,v)\) edge
   - \( C = C \cup \{u,v\} \)
   - \( E' = E' - \{\text{edges adjacent to vertices } u, v\} \)

\[ |C| < 2|C_{\text{opt}}| \]

the UC discovered is never twice bigger than the optimal (minimal) one

Proof Let \( A \) be set of edges packed at line 3 we have \( C = 2 \cdot A \) by construction.

Also \( A \leq C_{\text{opt}} \)

\[ \therefore C \leq 2C_{\text{opt}} \]

Run-time \( O(E) \)
Approximating TSP

- Assume complete graph w/ no negative weights
- Assume weights on graph obey triangle inequality

\[ w(A, B) \leq w(A, C) + w(C, B) \]

TSP on Euclidean Space

1. Select arbitrary root
2. Grow an MST
3. Let \( L \) preorder of MST
4. Return a ham-cycle that resembles preorder

Preorder:
\[ a \ b \ c \ h \ d \ e \ f \ g \]

Let \( C(H) = \sum C(u, v) \) on Ham

\[ C(H) \leq 2 \cdot C(\text{Hopt}) \]

Proof: \( C(\text{MST}) \leq C(\text{Hopt}) \)

Let \( w \) be preorder walk in step 3

\[ C(w) \leq 2C(\text{MST}) \]

cont. next pg
\[ C(H) \leq C(\omega) \text{ because of triangle inequality} \]

\[
\therefore C(H) \leq 2 C(H_{\text{opt}})
\]

**THM:** If we drop triangle inequality from requirement of poly algorithm to approximate TSP, unless \( P=NP \)

**Proof:** By contradiction, if it exists algorithm to approx TSP we solve HAM-CYCLE in poly time!

Let poly approx. with ratio bound \( \rho \) to approx. TSP.

Given \( G \) to find HAM CYCLE introduce remaining edges with weights as follows:

\[
(u, v) = \begin{cases} 
1 & \text{if } (u, v) \in E \\
\rho |u| + 1 & \text{if } (u, v) \notin E 
\end{cases}
\]