Proof

Steps
- Assumptions
- Facts
- Axioms
- Derived statements
- Hypothesis

Proof by Contrapositive
- to prove $p \to q$
- $\Leftrightarrow$
- $\neg q \to \neg p$

Proof by Contradiction
- p is true
- $\Leftrightarrow$
- $(\neg p \to \text{false})$ is true

Proof by Cases
- Divide domain into subsets + Discuss every cases

Proof By Construction
- Proof that $(\forall x) P(x)$

Mathematical Induction
- Basic
  - 1) Basis - $P(0)$ is true
  - 2) Induction Hypothesis - Assume $P(k)$ is true
  - 3) proof of Induction step - use $P(k)$ to prove $P(k+1)$ is true

Strong Principle of Mathematical Induction
- 1) Basis
  - $P(1)$ is true
- set a range $n_0 \leq n \leq k$ for P to be true
- 2) Induction Hypothesis
- 3) Proof of Induction step prove $P(k+1)$

Minimal Counterexample Principle
1) Assume P is false
2) set the range of input which P is false to be $n = k$
3) k is the smallest value for $P(k)$ to be false
4) $P(k-1)$ is true
5) prove $P(k)$ is true based on $P(k-1)$

Theory of Computation

Theory of Algorithm
- how problems can be solved

Applicability of Algorithm
- resource estimation
- correctness proof

Theoretical Analysis

Applicable Domain

Optimal Algorithm: if all algorithm is $\Omega(f(n))$
algorithm with $O(f(n))$ is optimal

Algorithmic Thinking
- think in terms of algorithm

Computational Complexity
- resources required

* Halting Problem
- given a program and input, will it end?
- can't be solved by computer

Space Complexity $S(n)$

$S(n) = O(T(n))$

Time Complexity $T(n)$

Computability Theory
- what problems can be solved
- mathematical logic

Computation Models
- λ-Calculus
- Recursive Functions
- Turing Machines
- Unlimited Register Machine
- Post Systems

* Church-Turing Thesis
- all Models have the same class of computable functions

- Highest Frequency Elementary Operations
Basic Operations
* algorithm
computer quality
code quality
machine independent
relative estimate
large input
asymptotic running time
Time Complexity
Factors → Input size
- operation with O(k)
Elementary operation
Estimation
Order of growth
Type of Analysis
*default - upper bound worst case
- lower bound best case
average case - expectation value
amortized analysis - average cost - no probability
Counting Iterations
Relations
ordered $<$
$f < g$
$\Updownarrow$
$f(n) = o(g(n))$
equivalence R
fRg
$\Updownarrow$
$f(n) = \Theta(g(n))$
Notations
$O(n)$
$f(n) = O(g(n))$
$\Updownarrow$
$f(n) \leq g(n)$
$\Omega(n)$
$f(n) = \Omega(g(n))$
$\Updownarrow$
$f(n) \geq g(n)$
$\Theta(n)$
$f(n) = \Theta(g(n))$
$\Updownarrow$
$f(n) = g(n)$
$o(n)$
$f(n) = o(g(n))$
$\Updownarrow$
$f(n) < g(n)$
$\omega(n)$
$f(n) = \omega(g(n))$
$\Updownarrow$
$f(n) > g(n)$

Search and Sort

Search

- Linear search
  - space $O(1)$
  - Average $= \frac{n+1}{2} \approx O(n)$
  - worst $O(n)$
  - Best $\Omega(1)$
- Binary Search
  - space $O(1)$
  - Best $\Omega(1)$
  - worst $O(\log n)$
  - Average k iterations

$$\sum_{i=1}^{n} Pr = \frac{1}{n}\sum_{i=1}^{n} i$$

$$= \frac{1}{n} \cdot \frac{n(n+1)}{2}$$

$$= \frac{n+1}{2} \approx O(n)$$

$n = 2^k - 1$

$k = \log(n+1)$

$$\sum_{i=1}^{n} Pr = \frac{1}{n}\sum_{i=1}^{k} i \times 2^{i-1}$$

$$= \frac{(n+1)}{n}\log(n+1) - 1$$

$O(\log n)$

Sort

- Recusive sort
  - Merge sort
    - $n = 2^k$
    - outer while $k = \log n$ times
    - in jth, $2^{k-j} = \frac{n}{2^j}$ pair of arrays (2^{j-1})

$$\sum_{j=1}^{\log n} \frac{n}{2^j} \times 2^j = n \log n$$

no. of comparison

least $\sum_{j=1}^{k} \left(\frac{n}{2^j}\right) 2^{j-1} = \frac{n \log n}{2}$

most $\sum_{j=1}^{k} \left(\frac{n}{2^j}\right)(2^j - 1) = n \log n - n + 1$

$\Omega(n \log n)$ best

$O(n^2)$ worst

  - merge 2 sorted lists
    - $\Omega(\min\{m, n\})$
    - $O(m + n - 1)$
- Linear sort
  - space: $O(1)$
  - Insertion sort
    - Average

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

$$= \frac{n^2}{4} + \frac{3n}{4} - \sum_{i=1}^{n} \frac{1}{i}$$

$O(n^2)$

no. of comparison

$$\frac{i-1}{i} + \sum_{j=2}^{i} \frac{i-j+1}{2}$$

$$= \frac{i-1}{i} + \sum_{j=1}^{i-1} \frac{j}{i}$$

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

    - Worst

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

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

$O(n^2)$

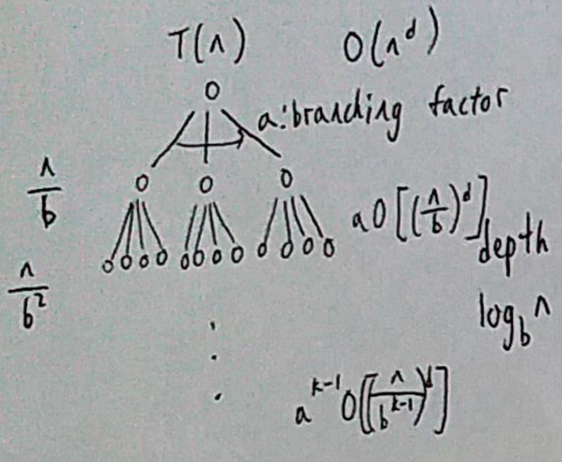
    - Best $O(n)$
  - Bubble sort → space: $O(1)$
    - Best / Average / Worst

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

- Selection sort → space $O(1)$
  - Best / Average / Worst

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

$O(n^2)$



size 1 0000 ... 0000

width $a^{\log_b n} = n^{\log_b a}$

$n = b^{k-1}$

$k = \log_b n + 1$

In j-th level

$a^{j-1} \times O\left(\frac{n}{b^{j-1}}\right)^d$

$= O(n^d) \times \left(\frac{a}{b^d}\right)^{j-1}$

Total work done

$\sum_{j=0}^{\log_b n} \left(O(n^d) \times \left(\frac{a}{b^d}\right)^j \right)$

$= O(n^d) \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$

Master Theorem ← Divide and Conquer → Steps → 1) Break p into subproblems (same type)
→ 2) recursively solve
→ 3) combine

$T(n) = a\,T(\lceil n/b \rceil) + O(n^d)$

$a > 0,\ b > 1,\ d \geq 0$

$$T(n) = \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

✓ $d > \log_b a$

$O(n^d) \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j \leq O(n^d) \frac{1}{1 - \frac{a}{b^d}}$

$= O(n^d)$

↓ $\frac{a}{b^d} = 1$

$d = \log_b a$

$O(n^d) \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$

$= O(n^d)(\log_b n + 1)$

$= O(n^d \log n)$

→ $\frac{a}{b^d} > 1 = d < \log_b a$

$O(n^d) \sum_{j=0}^{\log_b a} \left(\frac{a}{b^d}\right)^j$

$= O(n^d) \sum_{j=0}^{\log_b a} \left(\frac{a}{b^d}\right)^{\log_b n} \cdot \left(\frac{b^d}{a}\right)^j$

$\leq O(n^d) \frac{n^{\log_b a}}{n^d} \cdot \frac{1}{1 - \frac{b^d}{a}}$

$= O(n^{\log_b a})$

Sorting Network

- A comparison network which output sequence is monotonically increasing for every input

comparison network

- perform n-comparison simultaneously

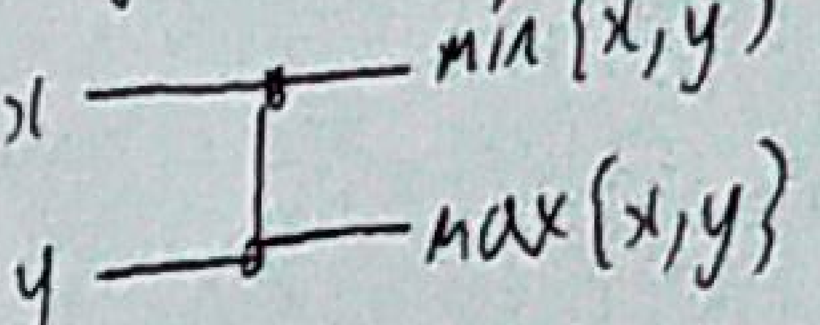
wires
- transmits a value from place to place

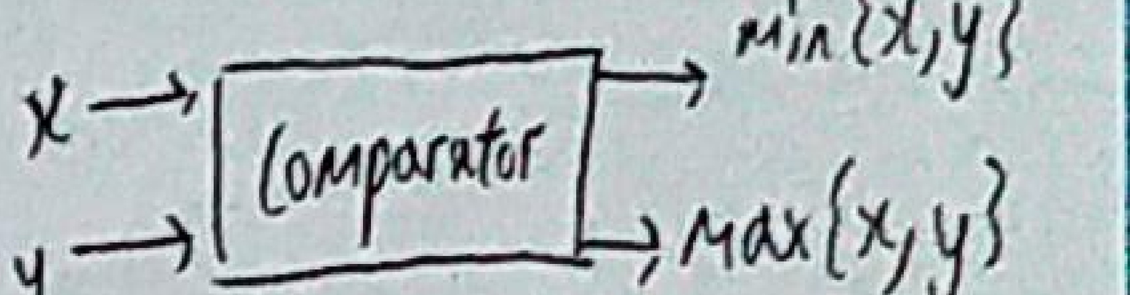
* connect an output to another input

- n input wires + n output wires
- Interconnections are acyclic

comparators

logical representation





O(1) time

d initial = 0

depth = max{dx, dy} + 1

Performance Analysis

$D(n) = \begin{cases} 0, & n=1 \\ D(n/2) + O(\log n), & n \geq 1 \end{cases}$

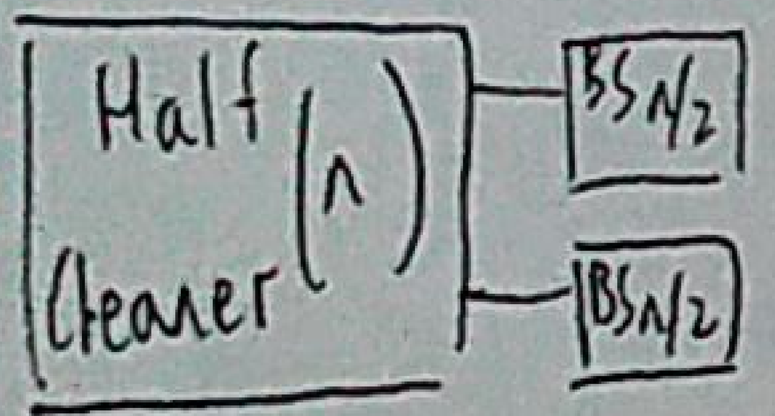
$D(n) = O(\log^2 n)$

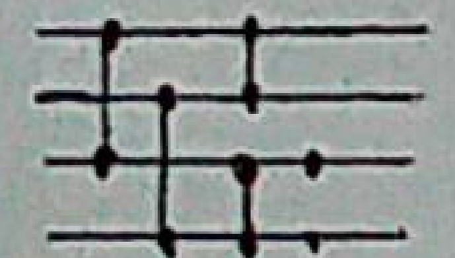
Construction

1) Bitonic sorter

双调

- sort bitonic sequence
  * mono ↑ then ↓ or circularly shifted





$D(n) = \begin{cases} 0, & n=1 \\ D(n/2)+1, & n=2^k, k \geq 1 \end{cases}$

* $O(\log n)$

Half-cleaner

- d = 1
- i compared with $i + \frac{n}{2}$

$i = 1, 2, \cdots, \frac{n}{2}$

Lemma: input 0, 1 (bitonic)

output: top half and bottom half are bitonic

- elements in top half ≤ elements in bottom half
- at least one half is clean

2) Merger

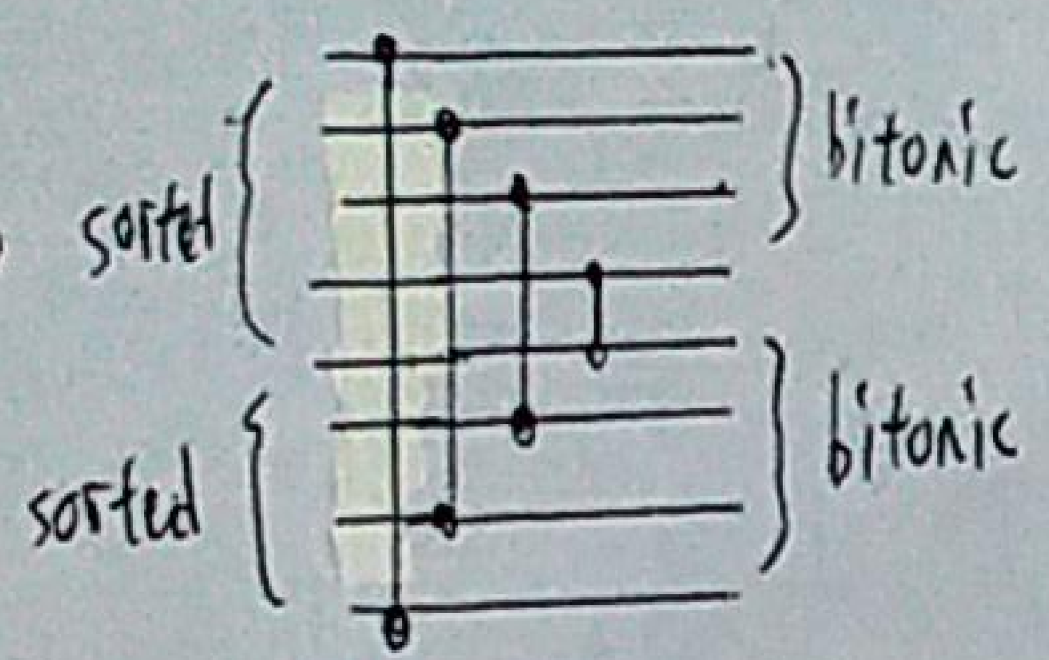
$\langle a_1, a_2, \cdots, a_{n/2} \rangle$

$\langle a_{n/2+1}, a_{n/2+2}, \cdots, a_n \rangle$

⇓

$\langle a_1, a_2, \cdots, a_{n/2}, a_n, a_{n-1}, \cdots a_{n/2+1} \rangle$

compare i with n − i + 1



3) Sorter

Sorter(n/2) + Sorter(n/2) + Merger(n)

zero-one principle

- if a sorting network works correctly with input drawn from {0,1}, it works on any input

→ Domain conversion Lemma
- If a comparison network a → b, for any monotonically function f,

f(a) → f(b)

Proof → Basis step

x, y → min{x,y}, max{x,y}

when apply f(x), f(y)

output: min{f(x), f(y)} = f(min{x,y})

max{f(x), f(y)} = f(max{x,y})

Stronger Statement
- if a wire assume a_i when input is a

f(a) → f(a_i)

Basis: wire at depth 0

f(a) → f(a_i)

Induction

d ≥ 1 wire is output of d comparator, input wire ≤ d

a_i, a_j → f(a_i), f(a_j)

- output wires of comparator

f(min{a_i, a_j})

f(max{a_i, a_j})

lemma proved

If comparison network with n inputs sort all 2^n possible sequences of 0, 1, it sorts all no. correctly

Proof

Suppose $\langle a_1, a_2, \cdots, a_n \rangle$ not correctly sorted

$a_i < a_j$ but a_j in front a_i

Let $f(x) = \begin{cases} 0, & x \leq a_i \\ 1, & x > a_i \end{cases}$

f(a_j) in front f(a_i)

but

f(a_j) = 1, f(a_i) = 0

zero-one sequence sorted wrong, contradiction

Greedy is optimal

Let S^* be fewest inversions

swapping inversion do not increase lateness, so S^* can be swapped to S

Let j and i be inversed

swap $j, i \to i, j$

l_i: before l_i': after

$l_k' = l_k \ \forall k \neq i, j$

$l_i' \leq l_i$

If j is late

$l_j' = f_j' - d_j$

$= f_i - d_j$

$\leq f_i - d_i$

$\leq l_i$

Correctness proof

Reduce Optimal solution

Greedy: no idle time

Optimal → no idle time

Greedy: no inversion

Optimal: inversion must be consecutive

Earliest deadline first

↑ $O(n \log n)$

Job j need t_j time, due d_j

start s_j, $f_j = s_j + t_j$

Lateness $= l_j = \max(0, f_j - d_j)$

Scheduling to Minimize Lateness

Sort by start time $O(n \log n)$

if not compatible, add classroom

Correctness Proof

greedy = depth

$OPT \geq$ depth

$OPT \geq$ greedy

$\therefore$ Greedy = OPT

Greedy is optimal

Depth: maximum lectures at a time

No. classroom $\geq$ depth

Greedy Algorithms → Basic Methodology

→ Interval Partitioning

-Lecture j start s_j finish f_j

Goal: find min no. classroom for all lectures (no 2 lectures in 1 room)

Interval scheduling

-job j start s_j finish f_j

-compatible: j_i and j_j don't overlap

Goal: find maximum subset of compatible jobs

* Sort jobs in finish time $O(n \log n)$

↓

Correctness analysis

Theorem: Greedy is optimal

Proof: Assume greedy not optimal

Let $i_1, i_2, \cdots, i_k$ = greedy $\quad i_1 = j_1, \cdots i_r = j_r$

$j_1, j_2, \cdots j_m$ = optimal $\quad r$: largest possible

$f_{i_{r+1}} \leq f_{j_{r+1}} \Rightarrow$ contradicts r being largest

Optimization Problem

-Given problem P with domain X

↓ minimization

↓ maximization

General Template

- consider $x_i \in X$ of P
- choose locally best choice

Strategies

↓ Exchange argument

↓ transform solution

↓ greedy

↓ Structural

-find a bound

↓

-prove every step within bound

↓ Greedy stays ahead

-show each step of greedy

↓

at least as good as any algorithm

$m \geq \max_{x_i \in S} c(x_i)$

$c^*(x_i) = m - c(x_i)$

to minimize, reweighting

A_G : greedy
A^* : optimal

$\frac{c(A^*)}{\rho} \leq c(A_G) \leq c(A^*)$

$1 \leq \frac{c(A^*)}{c(A_G)} \leq \max_{F \subseteq S} \frac{v(F)}{w(F)}$

- every maximal independent subset of S
Basis

$c : S \to R^+ + (S, C)$
Weighted Independent System

$v(F) = \max \{|I| \mid I \text{ is an independent subset of } F\}$

$u(F) = \min \{|I| \mid I \text{ is a maximal independent subset of } F\}$

$F \subseteq S$
Independent system (S, C)

decreasing
1) sort in order of $c(x_i)$
2) for loop : if $A \cup \{x_i\} \in C$
$A \leftarrow A \cup \{x_i\}$

template ← Greedy-MAX algorithm

time complexity
$O(n \log n + nf(n))$
↓ sorting
↓ time to check $A \cup \{x\} \in C$

Theorem ← Matroid → Definition
$M = (E, I)$

Extension
- x is extension if $x \notin I$
$I \cup \{x\}$ is independent

maximal
- $F \subseteq S$, $I \subseteq F$ is maximal in F if I has no extension in F

collection of independent sets

ground set (collection of elements)

properties

exchange property
$A \in I, B \in I$ and $|B| > |A|$
⇓
$b \in B \setminus A$ and $A \cup \{b\} \in I$
take any element from bigger set and add to smaller set to get larger independent set

$\phi \in I$

hereditary
$A \in I$ and $B \subset A$
⇓
$B \in I$
subset of independent set is independent

(S, C) is matroid
↕
for any $c(x)$, Greedy-MAX is optimal

If (S, C, c) is matroid
↓
Greedy-MAX is optimal

(S, C) is matroid
↕
$F \subseteq S$, $u(F) = v(F)$
⇓
all maximal independent subsets in a matroid have same size $u(F) = v(F)$

Independent System
↓ finite sets
↓ C collection of subsets of S

if $A \subset B, B \in C$
⇓
$A \in C$ (hereditary)
↘ subset

Dynamic Programming

- Break up problems into overlapping subproblems, and build up solutions to larger and larger sub-problems

Techniques
- dynamic programming over interval
- adding new variable
- multi-way choice
- binary choice

Recipe
1) Characterize structure of problem
2) Recursively define value of optimal solution
3) Compute value of optimal solution
4) Construct optimal solution from computed information

Tactics
- Brute force
  - directly recursive
    - repeated subproblem
    - exponential algorithm
- Memoization
  - store subproblem solution in memory
  - $O(n)$ if pre-sorted
- Bottom-Up
  - use given recurrence relation to calculate result from base case

Linear Programming

- Both objective function and constraints are linear

→ optimal solution
- on vertex of feasible region

→ Definitions
- Linear constraints form boundary as polyhedron (with hyperplanes)
- vertex: point of hyperplanes meet
- neighbors: 2 vertices adjacent on polyhedron

→ Algorithm
- Simplex Algorithm → steps
  1) start at a vertex
  2) compare objective value with neighbors
  3) move to neighbor that improves, step 2
  4) if no improving neighbor, stop
  - *Boundary
  - exponential in worst case
- Interior point algorithms
  1) LP → slack form
  2) obtain basic solution (by set all nonbasic variables = 0)
     - nonbasic → -in the objective
     - basic → -slack variables
  3) selecting nonbasic variable with positive k → increase value of xe without (x constraint)
  4) pivoting - exchange nonbasic ↔ basic
  5) repeat 2 → 4 until all negative k

→ Duality
- Primal form

$$\max \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \le b_i, \forall i \quad x_j \ge 0, \forall j$$

- Dual form

$$\min \sum_{i=1}^{m} b_i y_i \quad \text{s.t.} \sum_{i=1}^{m} a_{ij} y_i \ge c_j, \forall j \quad y_i \ge 0, \forall i$$

- Weak Theorem: x be any feasible solution to primal, y be any feasible solution to dual $\sum_{j=1}^{n} c_j x_j \le \sum_{i=1}^{m} b_i y_i$
- Strong Theorem: $\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i \iff x, y$ are optimal

→ Standard form

$$\max \sum_{j=1}^{n} c_j x_j \quad \text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \le b_i, i = 1, 2, \dots, m \quad x_j \ge 0, j = 1, 2, \dots, n$$

- Matrix-Vector form: $\max c^T x$ s.t. $Ax \le b$, $x \ge 0$

→ Slack form
- some constraints are equality constraints

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \rightarrow \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i, s_i \ge 0$$

s_i: slack variables

→ variants
- $\min \to \max$: $\min \sum_{j=1} c_j x_j \to \max -\sum_{j=1}^{n} c_j x_j$
- variables without constraints: $x_2^+, x_2^- \leftarrow x$; $x_2 = x_2^+ - x_2^-$; $x_2^+ \ge 0$; $x_2^- \le 0$
- $\ge$: $\sum_{j=1}^{n} a_{ij} x_j \ge b_i \rightarrow -\sum_{j=1}^{n} a_{ij} x_j \le -b_i$
- equality constraint: $\sum_{j=1}^{n} a_{ij} x_j = b_i = \begin{cases} \sum_{j=1}^{n} a_{ij} x_j \le b_i \\ \sum_{j=1}^{n} a_{ij} x_j \ge b_i \end{cases}$

$\hat{c}_i = c_i + \Phi(s_i) - \Phi(s_{i-1})$ $\quad \Phi(s): s \to R$ ← state

$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(s_n) - \Phi(s_0)$

1) use potential function to describe state of operations

Potential function method

Amortized Analysis
- spread out expensive operations to majority cheap operations
- find a tighter upper bound

↓ in worst case

↘ no probability involved

Accounting Method → 1) determine amortized cost for each operation $\hat{c}_i$

2) when $\hat{c}_i > c_i$, store credit for future use

3) when $\hat{c}_i < c_i$, use stored credit

* $\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i$ credit never goes negative!

Aggregate Analysis → 2) average cost per operation $= \frac{T(n)}{n}$

↓

1) find an upper bound $T(n)$ on total cost

Graph Theory

Shortest Path

def: $G=(V,E)$, $w: E \to R$

weight of $P = v_1 \to \dots \to v_k$

$$w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

shortest path from u to v

↓

path of min weight from u to v

shortest path weight $= d(u,v) = \min\{w(P)\}$

Single source SP

- from $s \in V$ to all $v \in V$, find $d(s,v)$

Properties

Triangle inequality

$d(v_1, v_2) \leq$

$d(v_1, v_3) + d(v_3, v_2)$

OPT substructure

- subpath of shortest path is shortest path

If negative weight cycle exists, SP may not exists

Dijkstra

non-negative weight

- maintain set S of known SP vertices
- add $v \in V-S$ to S with min distance
- update distances

Depth First Search DFS

Undirected Graphs

```
for each v ∈ V do
    visited(v) = false
for each v ∈ V do
    if not VISITED(v)
        explore(G, v)
```

↓

run time

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

Breadth First Search BFS

```
for each v ∈ V
    dist(u) = ∞
dist(s) = 0
Q = [s]
while Q is not empty
    u = eject(Q)
    for each edge(u, v) ∈ E
        if dist(v) = ∞
            Inject(Q, v)
            dist(v) = dist(u) + 1
```

Directed Acyclic Graph DAG

- directed graph without cycle

- Run DFS on G^R
- Run DFS on G, in order of decreasing POST no.

Type of Edges

Tree edges: part of DFS/BFS forest

Forward edges

- → nonchild descendant

Backedges

→ ancestor

Cross edges

→ neither descendant nor ancestor, explored

Detect negative cycle in $O(mn)$

- add node s, connect with 0-cost edge to all $v \in V$
- check $OPT(n,u) = OPT(n-1,u)$ $\forall u$

↓ no

cycle from $sp(s,u)$

Lemma

$\forall v$, if $OPT(n,v) = OPT(n-1,v)$

↓

no negative cycles

$\exists w$, $OPT(n,v) < OPT(n-1,v)$

↑

$P(s,v)$ has negative cycle

Dynamic Programming Bellman-Ford

space: $O(m+n)$

time: $O(mn)$ worst case

↑ negative weight

capacity scaling ← choose good augment path

- choose max bottleneck path

$O(m^2 \log C)$

↑ improvement

3) repeat until stuck

↑

Steps → 1) set all $f(e)=0$ → 2) find path $(s \to t) \in G_f$, augment flow along P, delete smallest edge

↑

Ford-Fulkerson algo → Terms → Bottleneck capacity = min residual capacity

→ Augmenting path : path $(s \to t)$ in G_f

run time $O(m n C)$ ← max capacity

↓ use BFS/DFS to find augment path

(A, B) be any cut

$$v(f) = \sum_{e \text{ out } A} f(e) - \sum_{e \text{ in } A} f(e)$$

Network Flow

- $G = (V, E)$, source $s \in V$, sink $t \in V$
- all nodes are reachable from s, no parallel edges
- capacity $c(e) > 0$, for $\forall e \in E$

→ Residual Graph

- capacity → forward edge $c_f(e) = c(e) - f(e)$
- ↓ backward edge $c_f(e^R) = f(e)$

↓

Flow

Def s-t flow f

1) capacity: $\forall e \in E$
   - $0 \le f(e) \le c(e)$
2) conservation: for $v \in V - \{s, t\}$
   - $\sum_{e \text{ in}} f(e) = \sum_{e \text{ out}} f(e)$

$$v(f) = \sum_{e \text{ out } s} f(e) - \sum_{e \text{ in } s} f(e)$$

↘

Cuts

s-t cut: a partition (A, B) of V with $s \in A$, $t \in B$

↓ capacity

$= cap(A, B) = \sum_{e \text{ out}} c(e)$

Theorem ←

- Strong Duality ← f is max flow ↔ no augmenting path
- weak duality $v(f) \le cap(A, B)$
- optimality if $v(f) = cap(A, B)$

↓

f = max flow + (A, B) = min cut

Turing Machine

Components
- finite set of states
- reading head
- cells
  - store symbols
- finite set of instructions
  - finite set of symbols
    $\{\triangleright, \triangleleft\} \cup \{s_1, \dots s_n\} \cup \{\square\}$
  - read s_j with state q_i $\longrightarrow$ state $q_i \to q_k$, $s_j \to s_k$, head $\to$ L/R/S $\to$ stay (left / right)
  - $\langle q_i, s_j \rangle \longrightarrow \langle q_k, s_k, L/R/S \rangle$

Deterministic TM
- function set has at most 1 instruction for each combination of symbol and state

Non-deterministic TM
- function set has more than 1 instruction for some combination of symbol and state

TM-computable function
- $M(a_1, \dots, a_n) \downarrow b$
- $f(a_1, \dots, a_n) = b$

other domains
D = object domain
coding of D = effective injection
$\alpha : D \to N$
$d \in D$ is coded by $\alpha(d)$
$f : D \to D$ extends to $f^* : N \to N$
$f^* = \alpha \circ f \circ \alpha^{-1}$
if f^* computable
f is computable

Algorithm
- input $\longrightarrow$ output
- finite set of instructions
- finite no. of steps

$\delta : Q \times T^k \to Q \times T^{k-1} \times (L/R/S)^k$

$\langle \Gamma, Q, \delta \rangle$
- finite set of symbols (alphabet)
- set of transition function
- q_{start}
- q_{halt}
- finite set of states

multi-tape

single-tape

all alphabet can be represented by only $\{0, 1, \triangleright, \triangleleft\}$

$\Sigma = \{a_1, \dots, a_L\}$

Σ^* (Kleene Star) = set of all strings/words from Σ

sequence of symbols $\in \Sigma$

Theory

multi tape vs single-tape

unidirectional tape vs bidirectional tape

$f : \{0,1\}^* \to \{0,1\}^*$ is computable in $T(n)$ time by bidirectional tape

$\downarrow$

it is computable in $4\,T(n)$ time by unidirectional tape

$f : \{0,1\}^* \to \{0,1\}^*$ is computable in $T(n)$ time by using Γ

$\downarrow$

it is computable in $4 \log |\Gamma| T(n)$ time by using $\{0, 1, \triangleright, \triangleleft\}$

$f : \{0,1\}^* \to \{0,1\}^*$ is computable in $T(n)$ time using k tapes

$\downarrow$

it is computable in $5k\,T(n)^2$ time by using 1 tape

Y is poly $\longleftrightarrow$ P=NP

↑

Y is NPC

↑

1) show Y is NP

Theorem $\rightarrow$ $W \leq_p X \leq_p Y$

$W \leq_p Y$

2) Choose a NPC X

Steps

3) prove $X \leq_p Y$

NP-Complete

- Y is NPC if

1) it is NP

2) $\forall X$ in NP, $X \leq_p Y$

special-case $\rightarrow$ general case

encoding with gadgets

Strategies

Simple Equivalence

- prove $X \equiv_p Y$

search problem $\leq_p$ decision problem

Self-Reducibility

$\forall x \in X \Rightarrow \exists y \in Y$

Cook $X \leq_p Y$

Nondeterministic Polynomial-time Problem

$\rightarrow$ Reduction $\rightarrow$ Definition

X P-reduces to Y if

↙ use poly-no. of steps + ↘ poly-no. of calls to oracle that solves Y

$X \leq_p Y + Y \leq_p X$

↓

$X \equiv_p Y$

Establish Equivalence

Purpose

Definition

↙ certifier (algo)

- algorithm C(s,t) for

$\forall s \in X \longleftrightarrow \exists$ string $\rightarrow C(s,t)$=yes

↘ certificate (input)

- input t such that ←

* certificate of NP can be detected by DTM in polynomial time

Karp (transform)

$\forall x \in X$ yes $\rightarrow \exists y \in Y$ yes

y yes $\rightarrow$ x yes

↓

Design algorithm x is yes $\Leftrightarrow$ y is yes

$X \leq_p Y$

+

Y in poly-time

↓

X in poly-time

Establish Intractability

$X \leq_p Y$

+

X is not poly

↓

Y is not poly