Week 1 NP-complete Problems and Reduction

Definitions : ① P : set of all (decision) problems solvable in polynomial time

: means can be solved in $O(n^k)$ time where $k > 0$ is some constant independent of n.

② Decision Problems : ⇒ Yes or No Question

- Given a list of numbers check if in order / if a given number is largest
- Given a weighted graph, what is the MST ?

Ex1 : How to convert sorting into a yes / no problem?

⇒ Given a list of n numbers : is it sorted ?

$O(n)$: $1 \le i \le n$ is $A[i] \le A[i+1]$?

sort : $O(n \lg n)$

Ex2 : How to convert shortest path into a yes / no problem?

Input : weighted graph , start vertex : S finish vertex : T

Goal : length of shortest path $s \to t$.

convert : Add a parameter : length: L

is there a path $S \to t$ of length L .

Ex3 : How to convert graph coloring into a yes / no problem?

Given a graph, how many colors to color it with a minimum number of colors?

"color" : adjacent vertices must have different colors

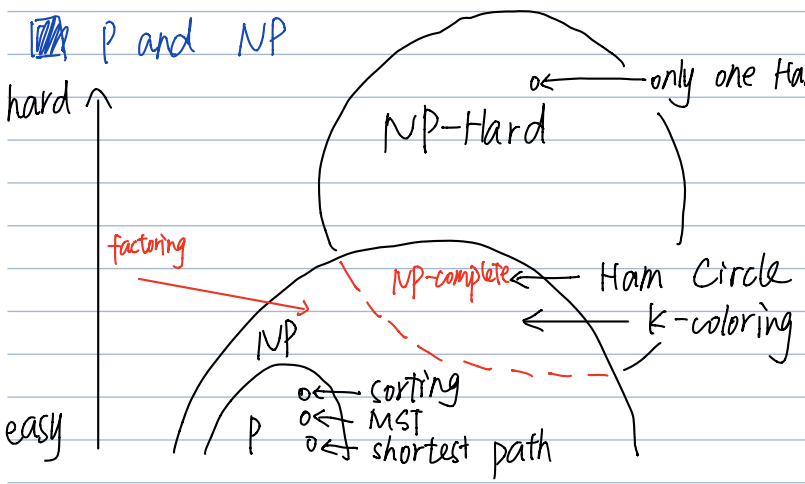
More specific : Given a graph , can graph be colored with 3 colors?

Can this problem solve in polynomial time? 4 5 6

Ex4 : How to convert Hamilton cycle into a yes / no problem? ⇒ $O(V!)$ (all permutation)

Euler cycle (NP complete) ⇒ even degree ⇒ $O(n)$

P and NP



Non-deterministic Algorithm
Intuitively can do things in parallel as long as resulting connected by OR's "V" all done in same time.

Example : $(x \vee y \vee z) \wedge (x \vee \bar{y}) \wedge (u \vee \bar{y} \vee z)$ n = # variables, M = # clauses

Problem : Is there a way to assign T/F to n variables to make Formula True ? (2^n : Truth table)

2^n processors in parallel : $\_ \vee \_ \vee \_ \vee \_ \vee \dots$ one of them be true ⇒ true

② NP: set of all (decision) problems whose solutions can be verified in polynomial time. i.e. a problem A is in NP if there is a polynomial verification ALG V that takes input an instance X of problem A and a proved solution S and outputs true IFF S is a solution to instance A $V(X,S)=1$

proved solution 潜在的解决方案

[e.g. find Ham cycle ⇒ V: check Ham cycle $O(n)$]

④ NP-complete

● NP-Hard:
X is NP-hard if every decision problem in NP reduction to X.

NP-complete 1-st problem: SAT (cook 1971).
A tedious, difficult, detailed proof

Any problem in NP could be translated into a boolean formular
ALG → Boolean Circuit → Boolean Formular
⇑
Took arbitrary algorithm and showed how to convert to a boolean formular

NPC: X is NP-complete if
① X is in NP
② For every decision problem D in NP $D \leq_p X$.
(X is NP-Hard)

⑤ Reduction
Example: Maximum Bipartite Matching (MBM)

MBM ≤ "Max Flow"
X ⇑ f(x)
"Reduces to"
Reduction: $O(V^2)$
Max-flow is at least as hard or harder than MBM

$A \leq B$ if B solve in polynomial then A solve in polynomial
so "reduction" can not more than O(B)

Property: New input f(x) to max-flow should be solvable if and only if the original input to MBM is solvable
↑ size of maximum bipartite matching
↑ if 1 : value of max flow

Reduction: Bipartite Graph: $G_1 = (X \cup Y, E)$
⇓
G': $V' = X \cup y \cup$ s.t
① directed edges $x \to y$
② add new edges $s \to$ each x to E'
③ add new edges each $y \to t$ to E'
④ capacity function: $C \to V' \times V' = \{0, 1\}$
every edge in E', $c(e)=1$
⑤ F-F (G')

↳ How we do reduce efficiently? ⇒ adj matrix $O(V^2)$

Cook: every problem in NP reduceds to SAT, i.e. SAT is NP-complete (Because SAT is in NP)

⇒ $SAT \leq_p 3SAT$

SAT: Given a boolean formula in n variables $f(x_1, \dots, x_n)$ is there an assignment of truth values (T/F, 1/0) to the variables so that the fomular evaluates to true?

Boolean Formular in CNF. n = # variables, m = # clauses. length of claus $\leq n$

3SAT: sample problem with n=3

EX. SAT: $(x) \wedge (y \vee \bar{x} \vee w \vee u) \wedge (\bar{x} \vee y \vee z) \wedge (x \vee \bar{z})$

1- clauses with 1 variable
(x): use auxiliary variables a, b
↳
$(x \vee a \vee b) \wedge (x \vee a \vee \bar{b}) \wedge (x \vee \bar{a} \vee b) \wedge (x \vee \bar{a} \vee \bar{b}) \Longleftrightarrow x$

2. clauses with 2 variables
$(x \vee \bar{z})$: use auxiliary variable c
$(x \vee \bar{z} \vee c) \wedge (x \vee \bar{z} \vee \bar{c}) \Longleftrightarrow x \vee \bar{z}$

3. clauses with 3 variable
$(\bar{x} \vee y \vee z) \Rightarrow$ just copy

4 clauses with more than 3 variables
$(y \vee \bar{x} \vee w \vee u)$: at least one variables is true
use auxiliary variable d.

$(y \vee \bar{x} \vee d) \wedge (\bar{d} \vee w \vee u)$

5. what we do if more than 4?
$x \vee \bar{y} \vee \bar{w} \vee \bar{z} \vee u \vee m$ use $d_1, d_2, \dots$ 6 ⇒ 4

$(x \vee \bar{y} \vee d_1) \wedge (\bar{d_1} \vee \bar{w} \vee d_2) \wedge (\bar{d_2} \vee \bar{z} \vee d_3) \wedge (\bar{d_3} \vee u \vee m)$

$SAT \leq_p 3SAT \Rightarrow$ 3SAT is NP-complete

↓

Reduction: $O(m \cdot (n-2)) = O(mn)$
polynomial time

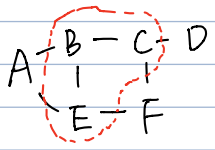
Does 3SAT ≤ SAT Yeah! it already is!

Harder Reduction

$A \to B \iff \bar{B} \to \bar{A}$

3SAT ≤p vertex cover ⇒

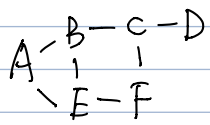
DEF: a vertex cover of G=(V,E) is set S⊆V such that if (u,v) ∈E then u∈S or v∈S



{B, C, E} is a vertex cover of size 3

最小点覆盖: Minimum vertex cover

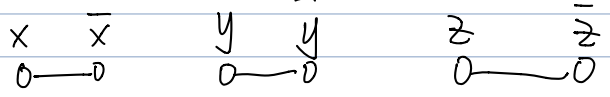
DEF: Independent set
a set S' ∈ V such that if (u,v) ∈ E, then u,v are not both in S'.
{A, D, F} is an Independent Set



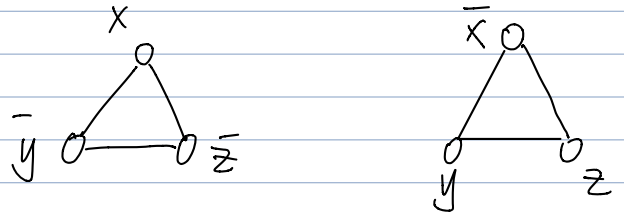
↳ 3SAT n = # variables m = # clauses

$(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z)$

∴ n=3 m=2 ∴ ⇒ x, y, z

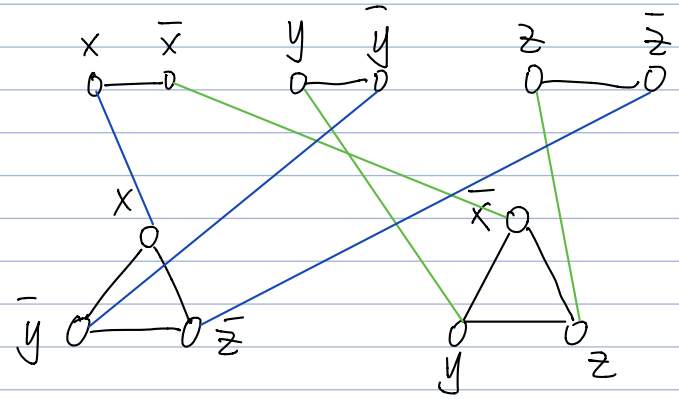


variable gadget



clauses gadget

$(x \vee \bar{y} \vee \bar{z})$ $(\bar{x} \vee y \vee z)$



覆盖一条边 覆盖一个三角形至少2个点

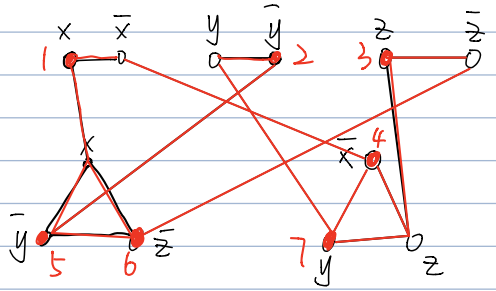
$k = n + 2m$

(min vertex cover)

$= 3 + 2 \times 2 = 7$

if we can find a min vertex cover with n+2m node in below graph

⇒ There is a solution to original SAT

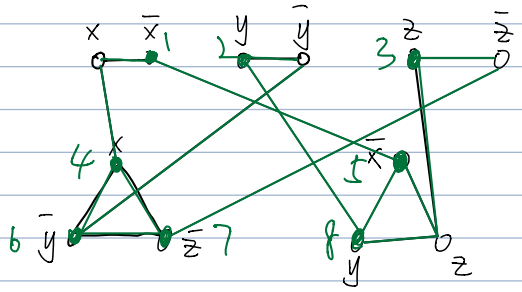


⇒ we know $x \wedge \bar{y} \wedge z = 1$ is a solution;

start with it, check the min vertex cover

⇒ see. we finish vetex cover with

$7 = 2 \times 2 + 3$ vertex



⇒ we also know $\bar{x} \wedge y \wedge z = 1$ is not a solution; start with it see what we get

⇒ we need at least 8 vertex to finish the cover.

⇒ Until Here: We show 3SAT $\leq_p$ vertex cover with k.

But we have not shown ⇒ vertex cover $\leq_p$ 3SAT with k

Why?

3SAT is NPC { 1. X in NP 2. X is NP-Hard
For all decision problems D in NP, $D \leq_p X$
so we need to show vertex cover $\leq_p$ 3SAT

Week 2. Approximation For NP-Hard Problems

{ vertex cover
  TSP
  set cover

- Design an algorithm
- Analyze the algorithm
  - Bound Running Time : Big O notation (↑ poly time)
  - Prove Correctness (↑ correct on all input)
- Goal:
  1. Hard-Problem (NP-complete)
  2. Fast Algorithm (poly-time)
  3. Exact Solution (correct)

  } => insist on ⇝ Relax Correctness

Unless P=NP, cannot achieve all three at same time

Approximation Bound optimization problems

- a set of valid outputs (Solutions), the cost/value $c(x)$ of each output X
  whether to minimize or maximize cost

Ex. vertex cover : minimum size of vertice
clique : maximum size of vertice
TSP : tour of min weight

- Optimal vs. Approx ALGs for optimization problems

Optimal ALG (usual notion of "correct")

compute for every input, a best valid output (OPT)

$$C(OPT) = \min\{c(x) : \text{All valid input } x\}$$
$$= \max\{c(x) : \text{All valid input } x\}$$

Ex. For TSP: $C(OPT)$ = tour length / weight (Not running time)

ρ: "rho" => approximation ratio

DEF: An ALG is a "ρ-approximation" ALG if on every input it computes a valid output APX whose cost/value is within A $\rho > 1$ factor of OPT.

For min problem: $C(OPT) \le C(APX) \le \rho \cdot C(OPT)$

For max problem: $\frac{1}{\rho} C(OPT) \le C(APX) \le C(OPT)$

ρ can be:
$\rho = O(n^c)$
$\rho = O(\lg n)$
$\rho = O(1)$

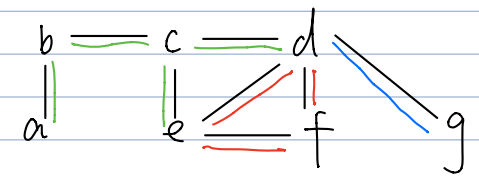
$\rho = 1$ means $\Rightarrow$ $C(OPT) = C(APX)$
so: ρ more closer to 1, more better
But we never know $C(OPT)$ $\Rightarrow$ we need another thing to compare.

Vertex Cover Problem

- Input: undirected graph $G = (V, E)$
- Output: a set $C \subseteq V$ of min size $\Rightarrow$ such that every edge in E has at least one endpoint in C.
- Object: minimize C.

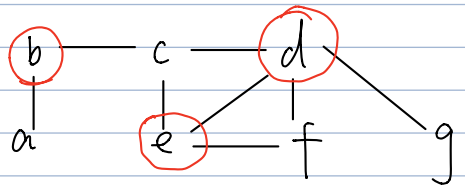
```
Approx-Vertex-Cover(G)   // Adj List format
01  C = empty
02  E' = E_G             // mark all edges as uncovered        O(V+E)
03  while E' ≠ ∅         // while there still exists an uncovered edge e
04      Let (u,v) be arbitrary edge of E'
05      C = C ∪ {u,v}    // add both u,v to E
06      remove from E' every edge incident to either u or v
07  return C             // Appropriate vertex cover
```

E.X.



(b,c) → add to C
(e,f) → add to C
(d,g) → add to C

⇒ Approx Vertex Cover C = {b, c, e, f, d, g}
C(APX) = 6



OPT: b, d, e
C(OPT) = 3

→ |C| = 6
|OPT| = 3

Theorem: Appr-Vertex-Cover is a 2-approximation ALG
Proof: output C is a vertex-cover of G. NTS: $\rho = 2$

let A := set of edges picked on line 4
To cover edges in A: any vertex cover in particular OPT must include at least one endpoint of each edge in A.

A: 没有共享节点的边集 | No two edges in A share an endpoint, all other edges incident to its endpoint and removed fore E' in line 6.

对于没有共享节点的n条边, 在OPT中也至少需要n个点 | No two edges in A are covered by the same vertex from OPT vertex cover
so $|OPT| \geq |A|$
This is a lower bound on OPT V.C.

对于没有共享节点的n条边, 我们加入了2n个点. | Further. each excution of line 4 Picks and edge None of whose endpoints already in C

$\therefore |C| \le 2|A|$. in fact $|C| = 2|A|$

Combine : $|C| = 2|A| \le 2|OPT.V.C|$.

$\therefore \frac{|C|}{|OPT|} \le 2 = \rho ho$

What if we do it greedy ? constantly remove vertex with largest degree

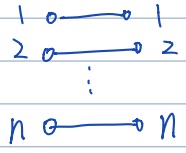
```
Greedy-vertex-cover
01  C := ∅
02  E0 = E
03  while Ei ≠ ∅
04      i = i+1
05      Vi = vertex in Ei-1 with max degree
06      di = deg_Gi-1(Vi)
07      Ei = Ei-1 - Vi
08      C = C ∪ {Vi}
09  return C
```

$\rho = O(\lg V)$

Recall : Matching : a matching M for a graph $G = (V, E)$ is a subset $M \subseteq E$ such that each vertex is incident to at most one edge in M.
Maximal : if not possible to add more edges in M.
$\therefore$ APPROX-Vertex-Cover : |A| is a matching ☆

Can we decrease ρ 2 → 1.999...? or smaller ? No !

Because we can find an example, that ρ must be 2.



$|C| = 2n$

$|OPT| = n$

we can do better than 2. But the upper bound must be 2

Currently Best know approx for ρ is $2 - O(\frac{1}{\sqrt{\lg V}}) = 2 - O(1)$ //cite : Halperin 2000

TSP (Traveling salesman Problem)

Input: Complete undirected graph $G = (V, E)$ with weight function : $w: E \to \mathbb{R}^*$ $|V| = n$
Output: Hamilton cycle H in G
Objective: Minimize $w(H)$

Ham Cycle $\le_p$ TSP (用会也可解决左边问题) Ham Cycle is NP-Hard $\Rightarrow$ TSP is NP-Hard

HamCycle: $G = (V, E)$ Arbitrary (no need to be complete)
Undirected
Unweighted

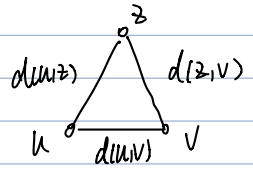
let $w(e) = \begin{cases} 1 & \text{if } e \in E \text{ in } G \\ 2 & \text{o/w} \end{cases}$

∴ if there is a Ham Cycle in G. Then there must be a Hamcycle in K_n with weight exactly n. (o/w. it will weight $\geq n+1$)

Extra Condition: (Strong Restriction)

Distance Function $d: E \to \mathbb{R}^{\geq 0}$ satisfy triangle :

$\forall u,v,z \in V.\ d(u,v) \leq d(u,z) + d(z,v)$



$d(u,u) = 0 \quad \forall u \in V$

$d(u,v) = d(v,u) \quad \forall u,v \in V$

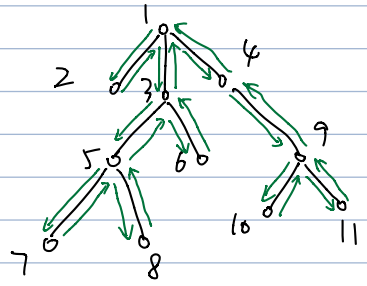
$d((x_1,y_1),(x_2,y_2)) = \sqrt{(x_1-x_2)^2 + (y_1-y_2)^2} \Rightarrow$ Euclidean metric

IDEA: Approximate Optimal TSP tour using MST's and graphs obey Δ-inequality

1. weight of MST is lower bound on cost of optimal tour
   remove an edge from a TSP tour get a path P is a spanning tree

   TSP cost $\geq$ Path P cost $\geq$ MST cost

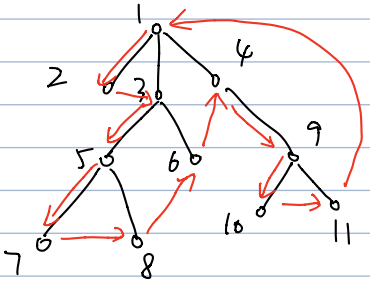
2. use MST to build a TSP Tour.
   consider a DFS traversal of MST



we use each edge twice
⇒ call it : twice-around tour W
Tour W weight = 2 × weight(MST)

2TSP cost $\geq$ 2MST cost = Tour W cost

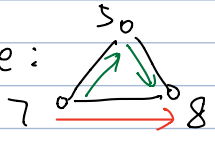
3. Remove extra vertices
   Take a shortest path to next unvisited vertex



1-2-3-5-7-8-6-4-9-10-11-1

Tour H

4. See:



Due to the triangle restriction

we can say 7→8 less than 7→5→8

∴ Tour W cost $\geq$ final tour H cost (APX)

$$\frac{2\,TSP}{2|OPT|} \geq \frac{Tour\ W}{|APX|} \geq \frac{Tour\ H}{|APX|} \qquad \rho = 2 \geq \frac{|APX|}{|OPT|}$$

Pseud code

Approx-TSP-Tour(G, d) ← $\triangle$-inequality

```
01  T = MST-Prim(G, d, s)                    // O(V^2)
02  W = DFS(T)  // tour traversal of T starting at S   // O(V)
03  H = W
04  for each vertex v in H (in traversal order)
05      if v has already been visited                  // O(V)
06          then delete v from H
07  return H        // add up weight: weight of APPROX TOUR
```

$\Rightarrow O(V^2)$

Homework: Christofides $\frac{3}{2}$ Appr to TSP

Set Cover Problem

Set system (X, F) $X = \{x_1, x_2, \cdots, x_n\}$ Finite Set, $|X| = n$, called universe

$F = \{S_1, S_2, \cdots S_m\}$ Family of subsets of X, such that every element of X belong to at least one set in F.

Undirected Graph: $G = (V, E)$

V: Universe, X, wireless

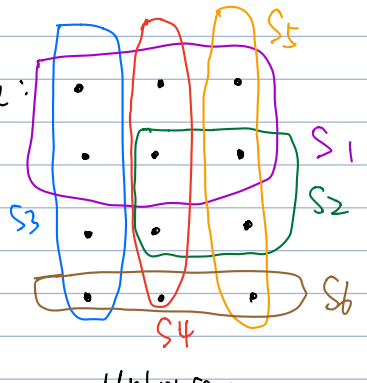
E: subset of, cardinality 2, 集合中元素的个数

DEF: A cover of F is a subcollection of sets whose union comprises X: $\bigcup_{S_i \in F} S_i = X$

Set Cover Problem: Given a set system (X, F) $F = \{S_1, \cdots, S_m\}$ Compute a set C of minimum cardinality such that $\bigcup_{S_i \in C} S_i = X$

NP-Complete

Example:



Universe •

X: set of black dots

F: $\{S_1, S_2, \cdots S_6\}$

Set cover of size 3: S_2, S_4, S_5 [exact cover]

Approx : Greedy ⇒ ① Pick the set that covers the most element
② Throw out all elements covered
③ Repeat until no element left

This algorithm has approx $\rho = O(\ln n)$

```
Approx-set-cover (X, F)      // F = {S1, ..., Sm}   |X| = n
01  U = X                    // U stores uncovered element
02  C = ∅                    // C stores set of cover
03  while U ≠ ∅ :
04      select Si in F that maximizes |Si ∩ U|
05      U = U - Si           // remove element of Si from U
06      C = C ∪ {Si}         // add Si to C
07  return C                 // sub family of F that covers X.
```

Running Time: $O(n^2 m)$

$|X| = 32$ ↘ Case that APX-ALG will always make mistakes

OPT (S7, S8) APX-ALG (S1, S2, S3, S4, S5, S6) ⇒ $|OPT| = 2$
$|C| = 6$

Theorem: Suppose $|X| = n$ if OPT uses k sets, greedy APX ALG find a solution with at most $O(k \ln k)$ set.

Proof: Since OPT uses k sets, there must be some set that covers at least $\frac{1}{k}$ of the elements.

Greedy choses set that covers the most, so it covers at least $\frac{n}{k}$ ∴ after first iteration at most $n(1-\frac{1}{k})$ left.

Again, OPT uses k sets there must be a set that covers $\frac{1}{k}$ of the remainder. Again Greedy chooses set that covers most elements. after 2-nd iteration there are $n(1-\frac{1}{k})^2$ left at most.

Generally. after t iterations at most $n \cdot (1-\frac{1}{k})^t$ elements left.
after $t = k \ln n$ iterations there are at most $n \underbrace{(1-\frac{1}{k})^{k \ln n}}_{< \frac{1}{e}}$ left

$n((1-\frac{1}{e})^{k \ln n} < n(\frac{1}{e})^{\ln n} = 1$ element left

which means we are done.

Week 3 Randomize Algorithms

Can flip coins pick a random bit or random element as a basic step.

Two Type

- Monte Carlo Algorithm : for every input correct with high probability
- Las Vegas Algorithm : for every input, always correct

String Equality

Multiple copies of huge dataset : n-bit string for some very large n

Problem : Alice (Processor A) : $a_1, a_2, \cdots a_n$
Bob (Processor B) : $b_1, b_2, \cdots b_n$
Are string equal ? with as few communications as possible

Deterministic Algorithm : Alice transmits n bits to Bob
Bob checked. send Alice answer
Requires n+1 bit communication.

Randomized Algorithm : $O(\log n)$ bits of communication.

A Field : a set of element : additions, multiplycation operations. satisfy commutative (交換律), associative (結合律), distributive laws. (分配律)

each element in field has additive and multiplicative identity additive and multiplicative inverse . EX : $\mathbb{R}, \mathbb{Q}$

A Finite Field : has a finite # of elements.

Ex. field of integers mod P, P a prime ; two integers a and b are equal iff they have same remainder when divided by P:
$a \equiv b \pmod p$ Z_p has p elements $\{0, 1, \cdots, p-1\}$

Randomized Algorithm : n-bits in server A : $a_1, a_2, \cdots a_n$
n-bits in server B : $b_1, b_2, \cdots b_n$.

Consider polynomials $A(x) = \sum_{i=0}^{n} a_i x^i$

$B(x) = \sum_{i=0}^{n} b_i x^i$

Consider a polynomial in Z_p : $A(x) = \sum_{i=1}^{n} a_i x^i \bmod p$

a root or zero of a polynomial is a value of X for which $A(x) = 0$

Theorem: a polynomial $A(x)$ of degree n has at most n zeros ⇐ using Horner's Rule 秦九韶算法

Algorithm: ① Alice and Bob agree on a large enough field F
(Modular arithmetic over a large enough prime p)

② Alice picks a random element $r \in F$ evaluates $A(r)$, send r and $A(r)$ to Bob

③ Bob computes $B(r)$ check whether $A(r) = B(r)$

Return { No, $A(r) \neq B(r)$; yes, $A(r) = B(r)$

cost: # bits (digits) communicated; cost of local computation by either Alice and Bob is ignored

Analyze success probability

Analysis: If string same, ALG always return yes

If Return No, ALG will never make mistakes

But the ALG makes a mistake when two strings are not same and it returns yes

Upper Bound this error probability:

$A(x) \neq B(x)$ but chosen r that $A(r) = B(r)$

⇒ Happens when r is a root of $(A-B)(x)$: poly of degree at most n

⇒ There are at most n roots
error probability is at most $\frac{n}{|F|} = \frac{n}{p}$

if choose $|F| = 1000n$. error probability is at most 0.001

$|F| = 1010n^2$ error probability is at most $\frac{0.001}{n}$

Repeated Trails: k-trails, independent [further decrease error rate]

let ϵ: Margin of error
T_i: R.V. indicating result of i-th Trail (out of k)

Constraint: $\epsilon \geq Pr[T_1 = 0 \text{ and } T_2 = 0 \cdots \text{ and } T_k = 0]$

$\geq (\frac{n}{p})^k$

Solving: $(\frac{n}{p})^k \leq \epsilon \Rightarrow k \geq \frac{\lg \epsilon}{\lg(\frac{n}{p})}$ $\because n < p \Rightarrow \lg(\frac{n}{p}) < 0$

Pseudo Code: Return whether $A(x) \equiv B(x) \bmod (p)$ $\forall x \in \{0,1,\cdots,p-1\} = Z_p$
with correction rate at least $1-\epsilon$

Poly-Equal ($a[0\cdots n]$ $b[0\cdots n]$, p, ϵ)

Using Horner's Rule
秦九韶算法

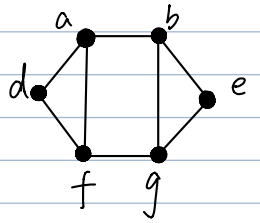
```
01  k = ⌈ lg ε / lg(1/p) ⌉                    θ(1)
02  for i = 1 to k                            ]
03      r = random(0, p-1)                    θ(1)   θ(n)
04      A' = A(r)                             ]
05      B' = B(r)                             ]
06      if A' - B' ≠ 0 (mod p)                θ(1)
07          return no
08  return yes
```

$$T(n) = \frac{\theta(n \cdot \lceil \lg \epsilon / \lg(p/n) \rceil)}{\theta(1)}$$

Karger's Algorithm

Minimum Cut Problem : $G = (V, E)$ connected, undirected, unweighted graph

- A cut in G is a subset of edges $F \subseteq E$ whose remove disconnected G.
- A minimum cut is a cut of minimum size



mincut size = 2

3 different min cuts $\{(d,a),(d,f)\}$; $\{(b,e),(e,g)\}$; $\{(a,b),(f,g)\}$.

- Properties of min cut:

① Claim 1: The size of a min cut is at most the min degree of any vertex in graph

Proof: if v has min degree. the cut $\{\{v\}, V-\{v\}\}$ has size $\deg(v)$
since min cut no larger than any cut in the graph
$\text{min-cut-size} \le \deg(v)$

② claim 2: in graph with n vertices if there is a min-cut of size k, G must have at least $\frac{nk}{2}$ edges

Proof: if there is a min cut of size k. every vertex must have degree at least k. by hand shake theorem $|E| = \frac{\sum \deg(v)}{2} \ge \frac{nk}{2}$

- Min-cut-problem : Find a cut of min size in G

Simple idea : mincut → maxflow

Also a min $s \to t$ cut for some $s,t \in V$
compute min $s \to t$ cut for <u>all</u> pairs $s,t \in V$ and take the min one

n^2 calls of maxflow

Deterministic : $\binom{n}{2}$ maxflow $\Rightarrow O(V^3E)$ Slow

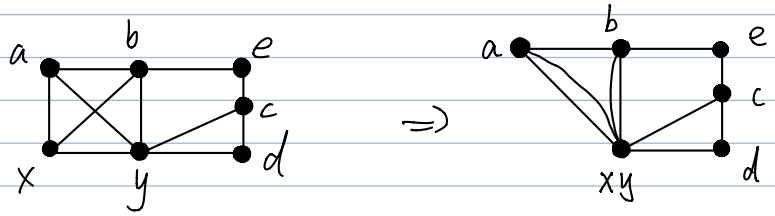
Randomized ALG: Karger : Random Contraction

$\Rightarrow$ Improving Running Time to $\tilde{O}(V^2)$ faster than max flow $O(V^2 \log^3 V) = \tilde{O}(V^2)$

Algorithm: Repeat
pick a random edge e
contract 2 endpoint of e
until only 2 vertices left
By "contracting" an edge, identify the endpoints as a single vertex

Edge Contraction allow parallel edges

Given an edge (x,y) in a multigraph G, contract x and y as follows
- Delete all edges between x and y
- Replace x and y with a new "supernode" xy
- replace all edges incident to x or y with edges incident to supernode xy
- Remove loops

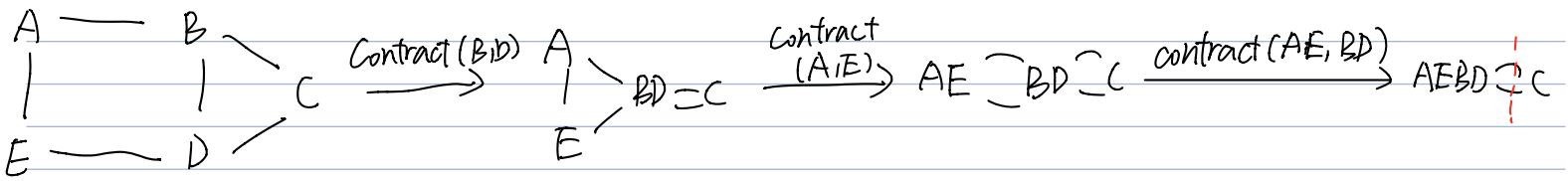


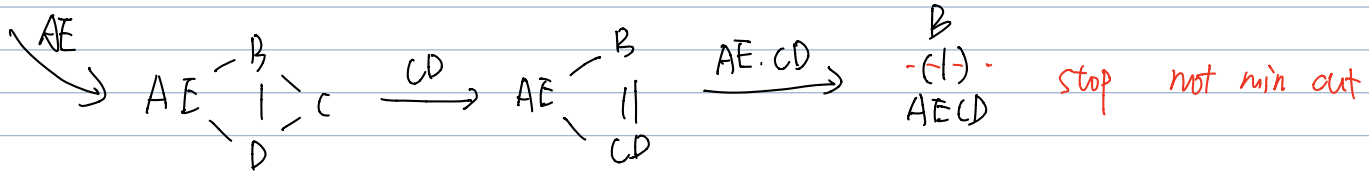
Karger's ALG
1. pick edge (x,y) at random in G
2. contract edge (x,y)
3. if there are more than 2 vertices $\Rightarrow$ go back to 1.
4. else output edges remaining as the cut

Contract $(G,(u,v))$
01 replace u and v by a new vertex uv
02 delete all edges (u,v)
03 for every edge (w,u) replace it by (w,uv)
04 for every edge (w,v) replace it by (w,uv)

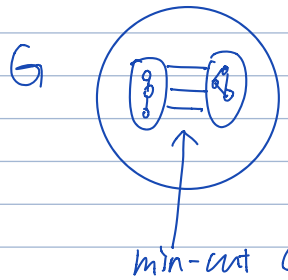
Karger (G)
01 for $i=1$ to $n-2$
02 pick an edge (u,v) in G at random $\theta(1)$
03 $G = $ contract $(G,(u,v))$ $\theta(V)$
04 return edges between 2 vertices that remain

$\Rightarrow \theta(V^2)$





Correctness Idea



Key idea: ALG1 should never contract one of the min-cut edges

Don't pick edge in C

Performance Analysis: Pr(ALG1's correct) ≥ Pr(C output at end)

= Pr(no edge of C contracted)

Theorem: Karger's ALG1 output a min-cut with probability at least $\frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \geq \frac{1}{n^2}$

where n = # of vertices

Proof: There may be many min-cuts in G. Fix some min cut C let C have k edges.

Lemma 1: So long as none of the edges in cut C are contracted, C remains a cut

Proof: Initially all vertices are in left or right side of C, contracting a edge that does not cross cut can only connect vertices on same side of cut

Lemma 2: Any cut in new graph is also a cut in original graph

Proof: The min degree in all intemediate graphs is at least k.

O/W $\rightarrow\leftarrow$ min size k

Lemma 3: When have $n-i$ vertices remaining (for some $i \geq 0$) the remaining are at least $\frac{(n-i)}{2}k$ edges in current graph.

Proof $\Rightarrow$: Denote edges contracted by $e_1, e_2, \cdots e_{n-2}$, C survives if these edges not in cut

① Prob C surive 1-st round $(e_1 \notin C)$

$$1 - \frac{k}{\#\text{edges}} \geq 1 - \frac{k}{\frac{nk}{2}} = \frac{n-2}{2}$$

⋮

① Prob C survive after i-st round $(e_1, e_2, \cdots e_i \notin C)$

$$1 - \frac{k}{\#\text{edges}} \geq 1 - \frac{k}{\frac{(n-i)k}{2}} = 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}$$

$$\text{Prob}(C \text{ survives}) = \text{Prob}(e_1 \notin C, e_2 \notin C, \cdots e_{n-2} \notin C)$$

$$= \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{3}\right)$$

$$= \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \times \cdots \times \frac{3}{5} \times \frac{2}{4} \times \frac{1}{3} = \frac{2}{n(n-1)}$$

$$\geq \frac{1}{n^2}$$

How can we boosting success probability? Repeated Trails

Prob (single run of KARGER will return a wrong answer :

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

Run karger t times independently and output smallest cut found:

Prob that all t runs failed: $\left(1-\frac{2}{n(n-1)}\right)^t$

Using Useful Inequality $1+x \leq e^x \quad \forall x \in R$

$$\Rightarrow \left(1-\frac{2}{n(n-1)}\right)^t \leq e^{-t\cdot\frac{2}{n(n-1)}}$$

If run karger $\frac{n(n-1)}{2}$ times, prob of failure is

$$\left(1-\frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}} \leq \frac{1}{e}$$

if run karger $\frac{n(n-1)}{2}\cdot \ln n$ times, prob of failure

$$\left(1-\frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}\cdot \ln n} \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n} \quad (\text{small failure})$$

$\Rightarrow$ Run Time $O(n^4 \log n)$

Week 4 Randomize Approximation Algorithm

Max-3-SAT

3-SAT (NP-complete)
Input: a set of m clauses, each with exactly 3 distinct variables, over a set $X=\{x_1,\dots,x_n\}$ of n boolean variables.

No clause contains a variable and its negation

Output: Is there a satisfying TVA?

Max-3-SAT
Input: Same e.g. $C_i = (x_{i1} \vee \bar{x}_{i2} \vee x_{i3})$
Output: a satisfying TVA that maximizes number of clauses satisfied
⇒ Also NP-Hard

Randomized Approximation ALG

IDEA: Try the all-true and all-false assignment
⇒ every clause is satisfied by at least one of them

∴ The best of the 2 assignment satisfies at least half the clauses and gives a 2-approximation

Randomizing - Pesudocode

01 for each variable x_i
02 set its value to 0/1 by fliping a coin
suppose coin is fair

⇒ what is the expected # clauses satisfied by such a random assignment?

⇒ for any clause C_i, the probability that it is not satisfied is exactly $(\frac{1}{2})^3$. ∴ C_i is satisfied with prob $= 1-\frac{1}{8}=\frac{7}{8}$

Let z_i be indicator R.V. such that $z_i = \begin{cases} 1 & \text{if } C_i \text{ satisfied} \\ 0 & \text{o/w} \end{cases}$

$z :=$ total # of satisfied clause

$$z = \sum_{i=1}^{n} z_i$$

$$\mathbb{E}(z) = \mathbb{E}\left(\sum_{i=1}^{m} z_i\right) = \sum_{j=1}^{m} \mathbb{E}(z_i) = \frac{7}{8}m$$

(Theorem. we don't care independency

Key Ideas: $\mathbb{E}(z) = \frac{7}{8}m$ represent a kind of average i.e. not all the z_i's can be less than $\mathbb{E}(z) = \frac{7}{8}$

∴ There has to be some assignment to the variables that achieves it.

Theorem: For any instanse of 3-SAT, there exists a truth assignment that satisfy at least a $\frac{7}{8}$ fraction of all the clauses

Proof: $\mathbb{E}(z) = \frac{7}{8}m$. and the R.V. $z \geq \mathbb{E}(z)$ some of the time.

ERDÖS: probabilistic method

Clearly m is an upper bound on # satisfied clauses

∴ appro ratio is at most $m / \frac{7}{8}m = \frac{8}{7}$

Approximation factor $\frac{7}{8}$ of optimal

Balls and Bins (CLRS 5.4.2)

n balls, m bins

Each ball throw to a random (uniformly) independently

Expected # balls in a bin

let B_{ij} be independent R.V.

$$B_{ij} = \begin{cases} 1 & \text{if ball } j \text{ in Bin } i \\ 0 & \text{o/w} \end{cases}$$

then $\mathbb{E}[\#\text{ balls in bin } i] = \mathbb{E}\left[\sum_{j=1}^{n} B_{ij}\right] = \sum_{j=1}^{n} \mathbb{E}(B_{ij})$

$$= \sum_{j=1}^{n} \Pr(\text{ball } j \text{ in bin } i)$$

$$= \sum_{j=1}^{n} \frac{1}{m} = \frac{n}{m}$$

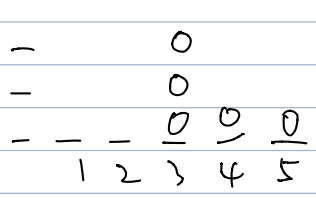
In particular, while $m = n$ expected # $= 1$

Do we expected that most bins have about one ball most of the time?

5 balls, 5 bins

small prize: if 4 balls in row or 4 balls in column

2 prize: if 5 balls in a row or 5 balls in a column



Binomial

expected # empty bins

Let y_i be indicator R.V.

$$y_i = \begin{cases} 1 & \text{if bin is empty} \\ 0 & \text{o/w} \end{cases}$$

Then: $\mathbb{E}[y_i] = \Pr(\text{bin } i \text{ is empty}) = (1-\frac{1}{m})^n \approx e^{-\frac{n}{m}}$ ← must be missed by all n balls

($\because 1-x \le e^{-x}$ and $1-x \approx e^{-x}$ for small x)

so. $\mathbb{E}[\#\text{ empty bins}] = \mathbb{E}[\sum_{i=1}^{m} y_i] = m \cdot e^{-\frac{n}{m}}$

when $n=m$. Expecte to see an $\frac{1}{e}$ fraction of the bins are empty

⇒ famous (coupon collectors problem)

Balls and bins (CLRS 5.4.2)

⇓

Maximum Load ⇒ Hashing, Addressing, collision

Maximum Load

Birthday Paradox:

for what n do expect to see 2 balls in a bin? ⇒ a "collision" occurs

Birthday Paradox where $m=365$ (ignore feb 29)

Prob of no collision in 1-st n balls:

$$1 \cdot (1-\frac{1}{m}) \cdot (1-\frac{2}{m}) \cdots (1-\frac{n-1}{m}) \le e^{-\frac{1}{m}} \cdot e^{-\frac{2}{m}} \cdot \cdots \cdot e^{-\frac{n-1}{2m}} = e^{-\frac{n(n-1)}{2m}} \approx e^{-\frac{n^2}{2m}}$$

This prob is less than $\frac{1}{2}$ when $n = \sqrt{2m\ln 2}$

for $m=365$ when $n \ge 22.49$ the prob that maxload is at least 2 is at least $\frac{1}{2}$

Summarize: expect to see a collision with $n = \Theta(\sqrt{m})$

⇒ $n^2 \approx m$. some collisions occur.

Hashing

sequence of insert, search, delete operations

Goal: Do all operations in $O(1)$ time

Formal Setup: keys come from some large universe U

set $S \subseteq U$ of actual keys stored : dynamic

$|S| = n \quad n \ll |U|$

Perform insert, search, delete operations

Table T of size m and hash function : $h: U \to \{0, 1, \ldots, m-1\}$

Given key $x \in U$, store it table position $T[h(x)]$

Two key questions

How compute h? h is Pseudo-Random : $h(x)$ equally likely to be any of $0, 1, \ldots, m-1$ independent of h for other keys.

How resolve collision?

collision : $h(x) = h(y)$ for 2 different keys $x \neq y$.

Chaining : each entry in table T is a Linked list.

open-addression : No linked lists : all keys stored in table T.

Chaining

operation : search, insert, delete

$\Rightarrow$ compute index $i = h(x)$

search : walk down linked list at $T[i]$
insert : insert x at head of $T[i]$
delete : perform deletion on list $T[i]$

1. keys spread out
2. $m = O(n)$
3. function h should be fast to compute. Assume $O(1)$

Average length of lists:

Let n_i = length of list $T[i]$ ($\sum_{i=0}^{m-1} n_i = n$)

Running Time is $O(\max_i n_i)$ (worst case)

length of list should be $\theta(n)$ (bad news)

switch to expected cost of operation let $\alpha = \frac{n}{m}$ = "load factor"
= average length of list

Running Times : $O(1+\alpha) = O(1)$ if $m = O(n)$

How to compute a hash function?

① Division method $h(x) = x \bmod m$ $\quad m$ = size of table

deterministic hash function. only Randomness assuming

Fact: put data is not random

EX. if m prime for all integers a and x

$h(x + am) = h(x)$

All such integer multiples of m collide with certainty

→ conclusion: Hash function must be random

② Universal Hashing: incorporates randomness into construction of hash function

DEF: a randomize ALG H for constructing hash functions $h: U \to \{0, 1, \dots, m-1\}$ is universal if for all $x \neq y$ in U

$$\Pr_{h \in H}(h(x) = h(y)) \leq \frac{1}{m}$$

Set H of such hash functions a universal hash family if procedure "choose h from H" is universal.

Theorem: if H is universal, then for any set $S \subseteq U$, $|S| = n$. for any $x \in U$ if $h \in H$ the expected # collisions between x and all other items in S is at most n/m

Proof: Each $y \in S$ $(y \neq x)$ has at most a $\frac{1}{m}$ chance of colliding with x by def of universal. So

Let: $C_{xy} = \begin{cases} 1 & \text{if } x,y \text{ collide} \\ 0 & \text{o/w} \end{cases}$

Let $C_x :=$ total # collision for x. so $\quad C_x = \sum_{\substack{y \in S \\ y \neq x}} C_{xy}$

$$\mathbb{E}(C_{xy}) = \Pr(x,y \text{ collide}) \leq \frac{1}{m}$$

$$\mathbb{E}(C_x) \leq \frac{n}{m}$$

Corollary: expected time to search / insert x with universal hash function

= expected time to look up a given key x.

= O (1 + expected #keys in x's linked list)

= O (1 + expected # keys that collide with x)

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

$= O(1+\alpha)$. $\alpha = \frac{n}{m}$ = load factor

$= O(1)$ if $m = \Theta(n)$

Theorem: Universal family of hash functions exist. (CLRS Theorem 11.5)

Example: $h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$

for any integers $a \in \mathbb{Z}_p^+ = \{1, 2, \ldots, p-1\}$ $b \in \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$

a, b Random chosen at set up, $0 \le a, b < p$ $(a \neq 0)$

p: large prime

$\Pr_{a,b}(h(x) = h(y)) \le \frac{1}{m}$ for all $x \neq y$.

Theorem: Hab is universal

Proof: fix 4 integers $r, s, x, y \in \mathbb{Z}$. such that $x \neq y$. $r \neq s$.

Linear system: $ax + b \equiv r \pmod p$

$ay + b \equiv s \pmod p$

has a unique solution $a, b \in \mathbb{Z}_p$ with $a \neq 0$, w.z.

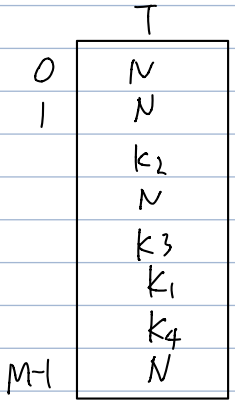
$a = (r-s)(x-y)^{-1} \bmod p$ ← Mult inverse of $(x-y)$

$b = (sx - ry)(x-y)^{-1} \bmod p$

Open Addressing For Collision Resolution

No linked lists: All keys stored in table T

Each T[i] is either NIL (empty) or a key



So: $n \le m$ $\alpha \le 1$ $(\alpha = \frac{n}{m})$

Table-doubly is necessary

Typically $\alpha \le \frac{1}{2}$

(python resizes at $\alpha = \frac{2}{3}$)

Hash function

Replace $h(X)$ with $h(x, i)$ for $i = 0, \ldots, m-1$

given a probe sequence $h(x,0)$ $h(x,1), \ldots h(x, m-1)$

for key x. probe sequence is a permutation of $0, 1, \ldots, m-1$ specifying a sequence of slot positions to be probed in search

```
Search (T, x)
01 for i=0 to m-1
02     j = h(x, i)
03     if T[j] == x
04         return j
05     else if T[j] == nil
06         return nil   // not in table
   else (T[j] ≠ x & Tj ≠ x)
       pass
```

Time: $O(\alpha)$

```
Insert (T, x)
01 for i=0 to m-1
02     j = h(x, i)
03     if T[j] == nil
04         T[j] = x
05         return j   // return # of slot that get key x.
06 return full
```

Deletion Tricky:

- find key
- replaced by "deleted"
- skip over "deleted" in search
- but use "deleted" in insert

Set $T[h(k,i)] = nil$

may cause search fail

Probe Strategies

- Linear probing:

$$h(x,i) = (h(x,0) + i) \bmod m$$

- Double Hashing

$$h(x,i) = (h_1(x) + i\, h_2(x)) \bmod m$$

h_1, h_2: 2 different hash functions

Bloom Filters

Setting: huge universe U of possible elements (e.g. password)

maintain a subset of $S \subseteq U$ where $|S| \leq n$

using an array $B[0, 1, \cdots, m-1]$ where $m = |B|$, $m = cn$ for $c \geq 1$

operations: insert(x): add x to S
query(x): is $x \in S$?

| want faster queries
small space

deletion not considered

What if willing to trade an occasional unsuccessful search in exchange for size and speed, nearly all cases?

IDEA: use Probabilistic data structure : Bloom filter

Bloom filter { : Bit vector of length m
. a set of k hash functions that returns value between 0 ~ m-1

k = # of hash functions

hash functions $h_1, h_2, \cdots, h_k : U \rightarrow \{0, 1, 2, \cdots, m-1\}$

↳ need to be random and mutually independent

- Initial Bloom Filter zeroed out
  initialize B to all 0

- Insert(X)

```
01 for i = 1 to k
02     f = h[i]
03     B[f(x)] = 1
```

EX: 0000 0000 000

Insert password1
hash1(PW1) = 3
hash2(PW2) = 10
hash3(PW3) = 8

0001 0000 101

Insert password2
$h_1(PW_2) = 5$
$h_2(PW_2) = 8$
$h_3(PW_2) = 1$

0101 0100 101

▲ contains PW1 and PW2

so we can't delete key but in password scenario we don't need delete

Query (x)

```
01  for i=1 to k
02      f=h(i)
03      if B[f(x)]==0
04          return false
05  // if reached here all B[f1(x), f2(x), .., fk(x)]=1
06  return True
```

Note: If $x \in S$. Always return "Yes"

But if $x \notin S$ Also might return "Yes" [all the bits could be covered by other elements in the set]

How to minimize False Positive?
What is prob of False Positive?

$$\Pr(\text{all } nk \text{ balls miss bin } b) = \left(1-\frac{1}{m}\right)^{nk} \simeq e^{-\frac{nk}{m}}$$

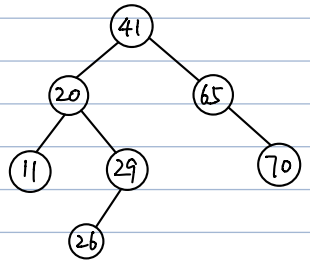
$$\Pr(\text{False Positive for } x \notin S) = \left(1-e^{-\frac{nk}{m}}\right)^{k}$$

Week 5 Balanced Search Trees (B-tree)

(nothing but Binary Search Tree)

Quick Review On BST.

- Rooted Binary Tree
- each node x : { key[x], left[x], right[x], p[x] (parent)



- any node x: for all nodes y in left subtree of X $key[y] \leq key[x]$
  for all nodes y in right subtree of X $key[y] \geq key[x]$

- operations: search (T, k)
  insert (T, k)
  delete (T, k)
  minimum (x)
  maximum (x)
  successor (x) ⇒ smallest key > key[x]
  precessor (x) ⇒ largest key < key[x]

- Analysis: operation all O(h) where h = height of tree

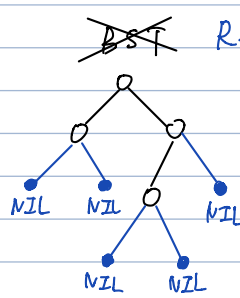
- Minimum Running Time:
  Problem: worst case : $\Theta(n)$ No better than Linked List

  Solution: Guarantee Small Height (balanced)
  ↓
  complete binary tree : $O(\lg n)$

  Method: Restructure Tree if necessary: insert / delete change the structure of tree

Red-Black Tree (CLRS 13.1)

- Add a color field (one bit) to each node of BST
- RB Tree Properties
  ① every node is red / black
  ② root and leaves (nils) are black
  ③ every red node has a black parent
  ④ all simple paths from node X to a descendant leaf of X have the same # of black nodes = Black height of X

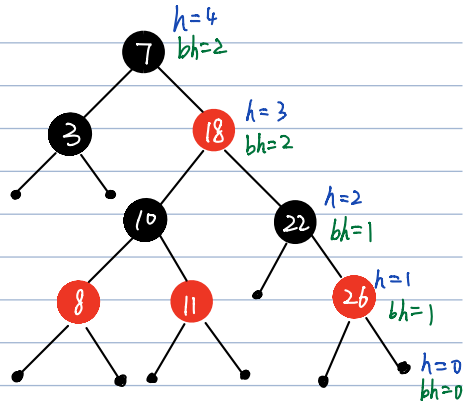


all leaves has zero children, all the internal nodes has 2 children

Black Height: bh(x) = # black nodes on path to a leaf, not containing X

(internal nodes)

Theorem: a RB tree with n keys has height $h \le 2\lg(n+1)$



Lemma 1: any node with height h has black height $\ge \frac{h}{2}$

proof: height is the length of a longest path to a leaf
by property ③, $\le \frac{h}{2}$ red nodes on the path
$\therefore \ge \frac{h}{2}$ are black

Lemma 2: for any node X, the subtree rooted at X contains $\ge 2^{bh(x)}-1$ internal nodes.

proof: By induction on height on X.

Basis: $h=0 \Rightarrow x$ leaf $\Rightarrow bh(x)=0 \Rightarrow 2^{bh(x)}-1 = 1-1=0$ ✓
subtree at x contains 0 internal nodes ✓

Inductive Step: let height of x be h and $bh(x)=b$

Some child of x has height $h-1$ and black height b or $b-1$ (if child is black)
By IH, each child has $\ge 2^{bh(x)-1}-1$ internal nodes

$\therefore$ subtree rooted at x contains $\ge 2\cdot(2^{bh(x)-1}-1)+1 = 2^{bh(x)}-1$
(+1 for itself)

Proof for theorem: let h and b be height and black-height, resp, of the root

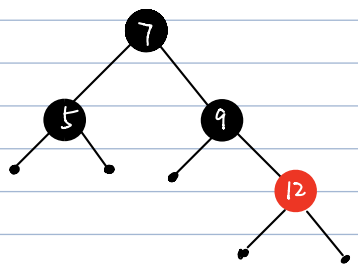
$$n \ge 2^b - 1 \ge 2^{\frac{h}{2}} - 1$$

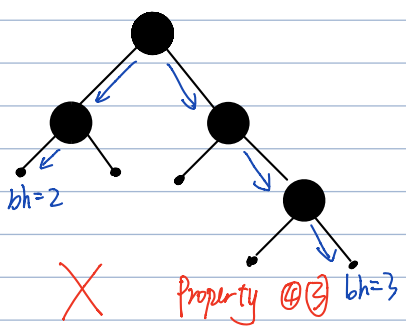
(lemma 2, lemma 1)

$$n+1 \ge 2^{\frac{h}{2}}$$

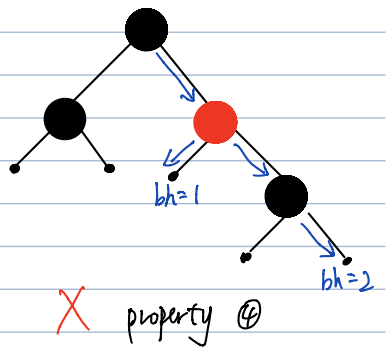
$$\lg(n+1) \ge \frac{h}{2} \qquad h \le 2\lg(n+1)$$

Problem of insertion into RB trees

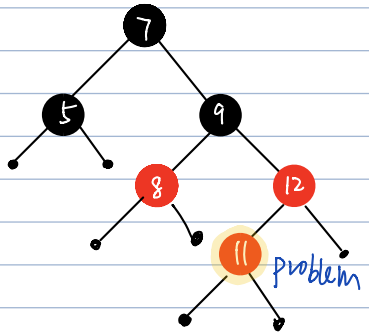




✗ Property ④⑤

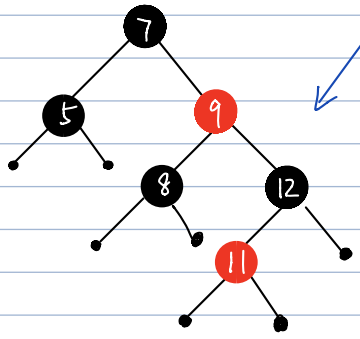


✗ property ④



- insert(8) ⇒ left for 9: color red
- insert(11) ⇒ left for 12 ⇒ color red ⇒ X ③
  color black ⇒ X ④

recolor the tree

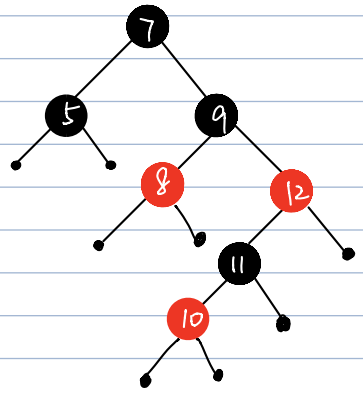


insert(10) : left child of 11

← recolor? Not enough

why $bh(7) = \begin{cases} 2 \\ 3 \end{cases}$ (7→5→nil) (7-9-12-11-10-nil)

X property ④



Goal: Restructe Tree in O(lgn) time

Insert Operation

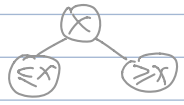
IDEA: insert x using BST-TREE-INSERT(x)

- color X red
- parent might be red ⇒ violate ③
- move violation up tree by recoloring ← 一直把这种冲突推到根对面上层再处理
- because tree imbalance can't satisfy ④ (black-height) without violation ③ (recoloring)
- must change tree structure in O(lgn) time — Goal

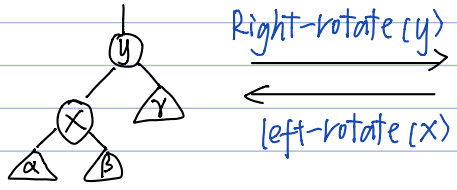
(Week 6 - RBT) ⇒

⇒ ## Rotation: (CLRS 13.2)

- basic operation for changing tree structure
- Key idea: Rotation does not change BST properties



- change local pointer structure



- Precenve BST Property

  $\forall a \in \alpha, b \in \beta, c \in \gamma \quad a \le x \le b \le y \le c$

- pseudo code

Right - Rotation (T, y)

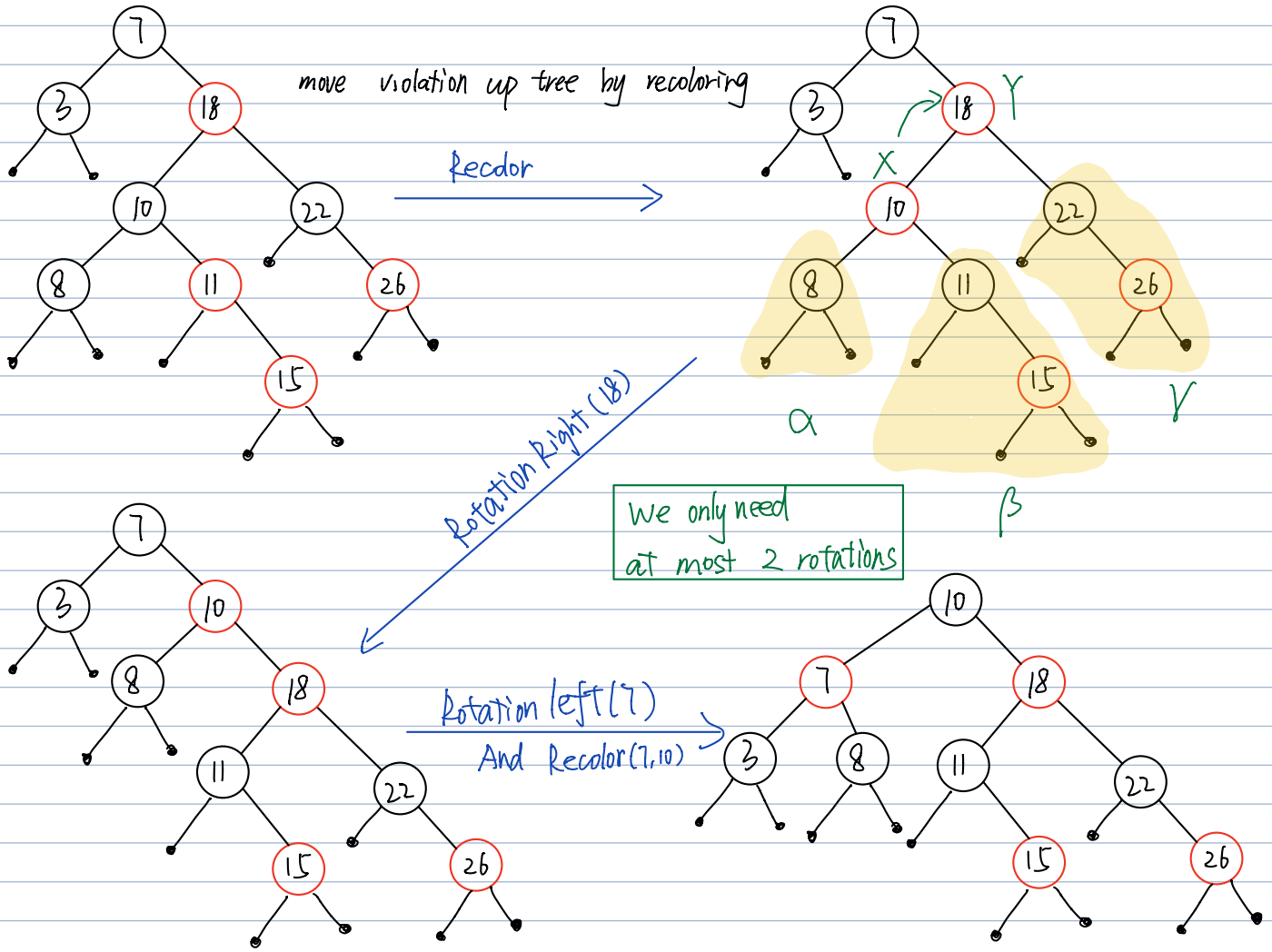
```
01  x = y.left          // set x
02  y.left = x.right
03  if x.right != nil
04          x.right.parent = y
05  x.parent = y.parent
06  if y.parent = nil
07          root[T] = x
08  else if y = y.parent.right
09          y.parent.right = x
10  else
11          y.parent.left = x
12  x.left = y
13  y.parent = x
```

LEFT-ROTATE(T, x)

```
1   y = x.right          // set y
2   x.right = y.left     // turn y's left subtree into x's right subtree
3   if y.left ≠ T.nil
4       y.left.p = x
5   y.p = x.p            // link x's parent to y
6   if x.p == T.nil
7       T.root = y
8   elseif x == x.p.left
9       x.p.left = y
10  else x.p.right = y
11  y.left = x           // put x on y's left
12  x.p = y
```

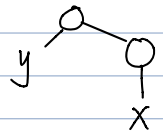
$T = O(1)$

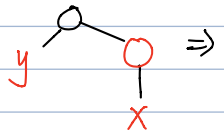
Example Insert (15)



- psedo code

```
01  BST-insert (T, x)
02  color x red
03  while (X ≠ T.root) and (x.parent.color = red):
04      if x.p == x.p.p.r      // x的parent是x的grandparent的左孩子
            y = x.p.p.l        // set y
            if y.color == red:
                x.color = black
                y.color = black
                x.p.p.color = red
                x = x.p.p
            else if x == x.p.left
                x = x.p
            then | right-rotate (T, x)
                 | x.p.color = black
                 | x.p.p.color = red
                 | left-rotate (T, x.p.p)
            else
                same as "then" clause, but swap "left" and "right"
```





Local Search

- what is local search?
  - a simple heuristic method for solving hard optimization problems
  - solving optimization problem:
    - find a solution minimizing / maximizing an objective function in solution space

Local solution method

- Start with an arbitrary solution S for a given problem
- Let $N(S)$ be set of solutions in "neighbour" of S obtained from local moves/changes
- If there is a "better" solution $S' \in N(S)$, switch to S'
- Else stop and output S

"Better": for minimization, S' is better than S if $val(S') < val(S)$
for maximization, S' is better than S if $val(S') > val(S)$

Example: Local search for 3-SAT

- Given: A 3-SAT instance (n variable, m clause, variables in each clause distinct)
- Let $x \in X \Rightarrow$ set of candidate solution to problem

- Starting Solution:
  Example: any random assignment $x \in X = \{0,1\}^n$
- For each $x \in X$ specify $y \in X$ are its neighbour
  Ex: x,y are neighbouring variable assignments $\Leftrightarrow$ differ in value of single variable
  Better solution in neighbourhood has fewer unsatisfied clauses

Key Questions For Local Search Algorithm

- how long will it run?
  - neighborhood of each solution small
  - need to find local optima in polynomial # of steps.
- how good is local optimal solution?
  - local optima is "nearly as good" as global optima

Local Search Outcomes

- Provably optimal solution

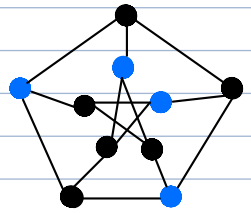
Ex: Network Flow (Ford-Fulkerson)
- start with zero flow: $f = 0$
- "local neighbourhood": set of all flows obtained by augmenting current flow along augmented path
- "Better": higher flow value
- provably optimal: $O(E^2 V)$ Time
  EDMONDS-KARP

local search terminates with local optimum that is a good approximation

- Bound ratio between optimal solutions and solutions that local search returns

- Example: Maximum Cut
  - input: undirected connected graph $G=(V,E)$
  - goal: a cut $(S, V-S)$ such that total # edges crossing cut is maximum
    Maximize $|E'|$ $E' = \{(u,v) \in E : u \in S, v \in V-S\}$

example



contast with: min-cut ⇒ karger: $\tilde{O}(V^2)$ Ford-Fu: $\tilde{O}(E^2V)$

$|V|=10$

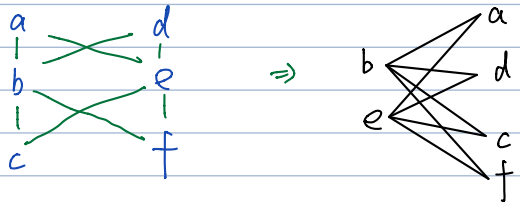
$|max\text{-}cut|=12$ Max-cut: NP-Hard

max-cut: $S=\{blue\}$ (reduction from 3-SAT, or, max-independent set)
$V-S=\{black\}$

(with weight ⇒ even harder) ⇒ KARP's 21 NP-complete problems.

⇒ computationnally tractable case: Bipartite Graphs

Ex:

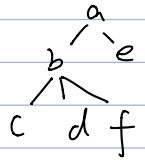


$|V|=6$ $S=\{b,e\}$

$|E|=8$ $V-S=\{a,d,c,f\}$ $|max\text{-}cut|=8$

Solve: Run BFS in $O(V+E)$ ⇒ run BFS on arbitrary vertex
odd layers: $V-S$
even layers: S

BFS: (e,a)

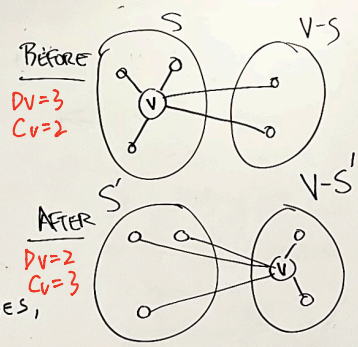


⇒ always find $|max\text{-}cut|$

LOCAL SEARCH ALGORITHM FOR MAX CUT

- START WITH ANY NONEMPTY SUBSET S OF V:
  THIS GIVES A CUT $(S, V-S)$
- WHAT IS A SMALL LOCAL CHANGE TO INCREASE # OF EDGES CROSSING CUT?
- REPEAT STEP UNTIL NO SUCH V EXISTS

→ WHILE THERE IS A VERTEX V SUCH THAT MOVING V TO OTHER SIDE OF CUT INCREASES # CROSSING EDGES, MOVE V TO OTHER SIDE



WHEN: WHEN V HAS MORE INCIDENT EDGES WITHIN S THAN CROSSING EDGES, i.e. $|\{(u,v) \in E : u \in S, v \in S\}| > |\{(u,v) \in E : u \in S, v \in V-S\}|$ (OR VICE VERSA), MOVE V

真论和动为 如梦.此

Notation: For a cut $(S, V-S)$ and vertex V. define
$C_v(S, V-S)$ = # edges incident on V that cross $(S, V-S)$
$D_v(S, V-S)$ = # edges incident on V that dont cross $(S, V-S)$

pseudo code:

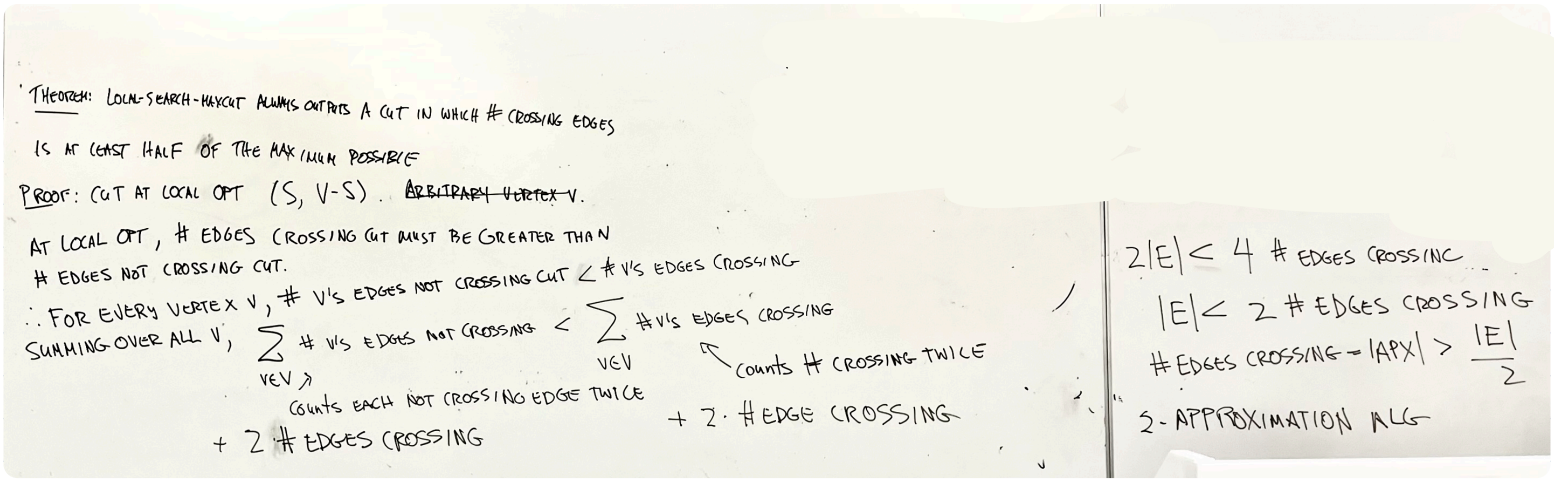
```
Local-Search-Maxcut
01  let (S, V-S) be arbitrary cut of G
02  while there is a vertex V with Dv(S, V-S) > Cv(S, V-S)
03        move v to other side of cut // increase # of crossing edges by Dv - Cv > 0
04  return find cut (S, V-S)
```

Running Time: THM: local-search-maxcut terminates in $O(n^2)$. n = # of vertices

Proof: each iteration ↑ # edges crossing cut at least by 1
since # edges crossing cut $\leq |E| = O(n^2)$
the ALGO terminates after $O(n^2)$

How good? $\frac{\#(APX)}{\#(OPT)} \geq \frac{1}{2}$.

⇒ proof:



THEOREM: LOCAL-SEARCH-MAXCUT ALWAYS OUTPUTS A CUT IN WHICH # CROSSING EDGES IS AT LEAST HALF OF THE MAXIMUM POSSIBLE

PROOF: CUT AT LOCAL OPT (S, V-S). ~~ARBITRARY VERTEX~~ V.

AT LOCAL OPT, # EDGES CROSSING CUT MUST BE GREATER THAN # EDGES NOT CROSSING CUT.

∴ FOR EVERY VERTEX V, # V's EDGES NOT CROSSING CUT < # V's EDGES CROSSING

SUMMING OVER ALL V, $\sum_{v \in V}$ # V's EDGES NOT CROSSING < $\sum_{v \in V}$ # V's EDGES CROSSING

counts EACH NOT CROSSING EDGE TWICE

counts # CROSSING TWICE

+ 2 # EDGES CROSSING

+ 2 · # EDGE CROSSING

$2|E| < 4$ # EDGES CROSSING

$|E| < 2$ # EDGES CROSSING

EDGES CROSSING = $|APX| > \frac{|E|}{2}$

2-APPROXIMATION ALG

Hash map m_1 : (key: v. value: Dv-Cv)
Hash map m_2 : (key: Dv-Cv value: Hashset(V))

```
random assignment:
 for all V:
       m1[V] = Dv - Cv
       if Dv-Cv > 0:
            m2[Dv-Cv].add(v)
while (m2 is not empty):
       key = any key in m2
       v = any element in m2[key]
       for u ∈ v.adj:
            if u,v in same part:
                 val = m1[u]
                 m1[u] += 1
                 if val > 0
                      m2[val].remove(u)
                      m2[val+1].add(u)
                 else if val == 0:
                      m2[1].add(u)
```

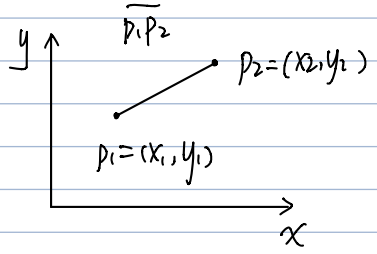
```
rand assignment   V ← hash set
heap ← V: Dv-Cv
while heap.max > 0
      v.C = heap.pop
      for v ∈ V.adj
            if u,v are in same side
```

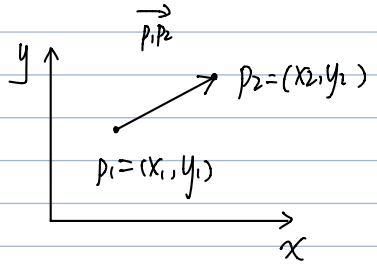
```
    else (u,v not in same part)
        val = m1[u]
        m1[u] -= 1
        if val > 1
            m2[val].remove(u)
            m2[val-1].remove(u)
        elseif (val == 1):
            m2[val].remove(u)
val2 = m1[v]
m1[v] = -val2
m2[val2].remove(v)
```

Week 7 Computational Geometry

- Algorithm and data structures for geometric problems

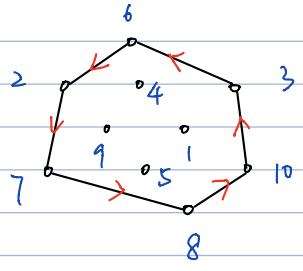
- input/ output / intermediate data: Geometric objects represented by real numbers
- work in 2 dimensions i.e. in plane
- Basic Definitions
  - points are (x,y) with $x,y \in \mathbb{R}$
  - line segment is a portion of a straight line between 2 points
    $p_1(x_1,y_1)$ $p_2(x_2,y_2)$





convex hull

problem: give a set P of n points in the plane
Goal: find the smallest convex polygon containing P ⇒ (given as array with 2 fields x-coordinate, Y- coordinate)
Example:



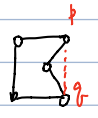
output : indices in some order
[7, 8, 10, 3, 6, 2]
6 vertices on boundary in CCW

Definition :

① convex hull of P : smallest convex polygon containing all points of P

② polygon : a circular sequence of points (vertices) and line segments (edges) between them

③ convex : X is convex : $p \in X$, $q \in X$ then $\overline{pq} \in X$ 反例



④ smallest : (inclusion minimal) : for every convex set X if $P \subseteq X$ then $conv(P) \subseteq X$
smallest : any convex proper subset of convex hull excludes at least one point of P
(凸多边形的任何凸正子集都漏掉P中的至少一个点。)

Representation : represent CH (convex hull)

As a circular doubly linked list of vertices in CCW order, if i-th point is a vertex of CH. next[i] is index of next vertex CCW and pred[i] index of next vertex CW, otherwise next[i] = pred[i] = 0

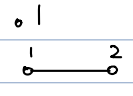
Q: How tell if a point is on one side or another of a line?

Convex Hull :

n=1 what is CH? point

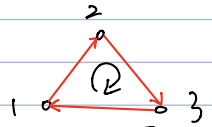
n=2 line segment

n=3 triangle

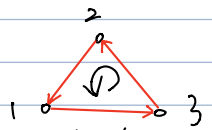


output : [1]

output : [1,2] or [2,1]



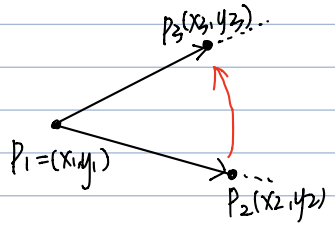
cant get [1,3,2]



cant get [1,2,3]

output: 6种可能

Given 3 points, what order are they in?
which line has bigger slope?



Goal: answer in O(1) time

- use only +, −, × and comparison
- avoid division and trig functions

$$\frac{y_2-y_1}{x_2-x_1} < \frac{y_3-y_1}{x_3-x_1} \Rightarrow (x_2-x_1)(y_3-y_1)-(x_3-x_1)(y_2-y_1) \begin{cases} >0 & \text{逆时针 CCW} \\ <0 & \text{顺时针 CW} \\ =0 & \text{collinear} \end{cases}$$

$$\Leftarrow$$

$$\begin{vmatrix} x_2-x_1 & y_2-y_1 \\ x_3-x_1 & y_3-y_1 \end{vmatrix} > 0 \quad \text{CCW}$$

or

$$\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} > 0 \quad \text{CCW}$$

orientation test

or

CCW test

orientation test is 2-Dim analogue to comparisons

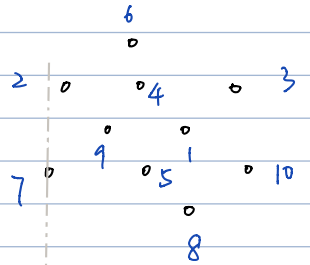
$a>b$? $\begin{vmatrix} 1 & a \\ 1 & b \end{vmatrix} < 0$?

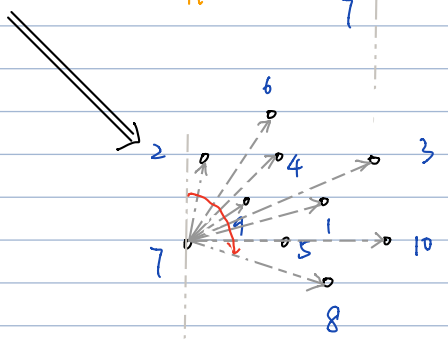
Jarvis March

① find left most point p_1 ⇒ all other points or other side p_1

② Repeat
find next hull vertex

③ until next point is p_1





repeat use CCW test until we find the "right" most point

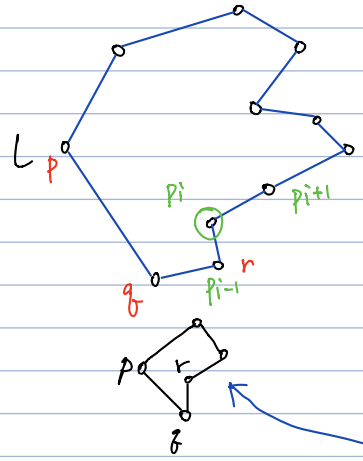
$O(n^2)$

```
01  l = 1
02  for i = 2 to n
03      if x[i] < x[l]
04          l = i
05  p = l
06  repeat
07      q = p+1    // make sure p≠q
08      for i = 2 to n // search for a point q s.t. ccw(p,i,q) is ccw for all points i
09          if ccw(p,i,q) // if i is more ccw than current q. update q.
10              q = i
11      next[p] = q : prev[q] = p
12  until p = l
```

找到一个q 让其他所有点在 pq 的逆时针方向

Graham's Scan

1. Find left most point $P_1 = L$ (by x-coordinate)
2. sort $P_2 \dots P_n$ around P_1 (using ccw test instead of comparision)
3. once points sorted, connect in CCW order starting and ending at $P_1 = L$



simple polygon formed by sorted vertices

↓ ?

convex polygon

Test if vertex convex or concave by ccw test:

P_i is concave $\Leftrightarrow$ (P_{i-1}, P_i, P_{i+1}) is CW

P_i is convex $\Leftrightarrow$ (P_{i-1}, P_i, P_{i+1}) is CCW

Three - penny - algorithm

Repeat: if p, q, r is CCW (q is convex)
move penny p forward to successor r
if p, q, r is CW (q is concave)
remove vertex q add edge $\overline{pr}$
move middle penny back to predecessor

Pseudo Code

```
01  P1 = left most point of P
02  [P2 ... Pn] = remaining points of P1 sorted in CCW order around P1
03  if n < 2
04      return CH is empty
05  else S = empty stack                                   O(n lg n)
06       push (P1, S)
07       push (P2, S)
08       push (P3, S)
09       for i := 4 to n
10            while CW (next-to-top(S), top(S), Pi)
11                 pop (S)
12            push (Pi, S)
13  return S
```

How many times iteration?

2 cases
1. How many iterations when move penney forward $n-2$
2. How many times move middle penny back ? How many times delete edge and add new ? ? $\Rightarrow$ $n-h$

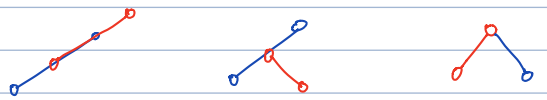
total : $2n-h-2$ h: size of (CH vertices)

Line segment intersection

Input: set of n line segments, each specified by X-and-y-coordinate
$\Rightarrow 4n$ real numbers

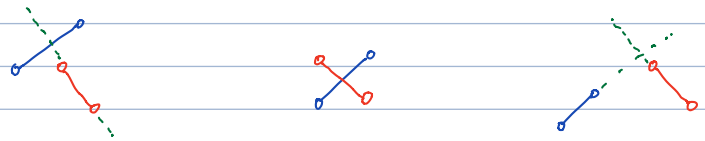
Goal: want to know if any 2 intersect

Assumptions: General position; i.e. No degenerate cases



这三种不允许

Two segments $n=2$ How to tell whether two line segments intersect?



Simple condition: 2 segments $\overline{ab}$ and $\overline{cd}$ intersect IFF the end points a and b are on opposite side of line $\overleftrightarrow{cd}$ and end points c and d are on opposite side of line $\overleftrightarrow{ab}$

Two segment intersect (a,b,c,d)

```
01  if CCW(a,b,c) = CCW(a,b,d)
02      return False
03  if CCW(a,c,d) = CCW(b,c,d)
04      return False
05  return True
```

Sweep-line Algorithm

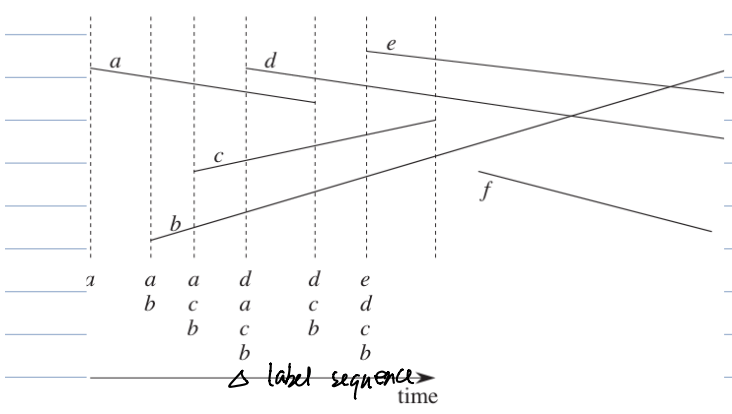
1-st: give line segment a unique label
next: image a vertical line sweeping across segment from left to right

at each position of sweep line, look at sorted sequence (by y-coordinate) of labels of segments that line hits

only times this sequence can change is when sweep line passes through endpoints or passes intersection points

ANY-SEGMENTS-INTERSECT(S)

```
1  T = ∅
2  sort the endpoints of the segments in S from left to right,
       breaking ties by putting left endpoints before right endpoints
       and breaking further ties by putting points with lower
       y-coordinates first
3  for each point p in the sorted list of endpoints
4      if p is the left endpoint of a segment s
5          INSERT(T, s)
6          if (ABOVE(T, s) exists and intersects s)
               or (BELOW(T, s) exists and intersects s)
7              return TRUE
8      if p is the right endpoint of a segment s
9          if both ABOVE(T, s) and BELOW(T, s) exist
               and ABOVE(T, s) intersects BELOW(T, s)
10             return TRUE
11         DELETE(T, s)
12 return FALSE
```



判断的伪代码

Any-intersections (S[1…n] // x,y coordinates)

```
sort the end points of S from Left to right
create an empty label sequence
for i = 1 to 2n
        l = label[i]
        if isleft[i]
                insert(l)
                if intersect(S[l], S[successor(l)])
                        return true
                if intersect(S[l], S[precessor(l)])
                        return true
        else (isright[i])
                if intersect(S[successor(l)], S[precessor(l)])
                        return true
                delete(label[i])
return False
```

Week8 Advanced Flow

standard flow network

Input: Directed Graph $G = (V, E)$
Capacity $c(e) > 0$ for each node $e \in E$ two vertices $s, t \in V$

Define: Feasible Flow
$f: E \to \mathbb{R}$ s.t. $0 \le f(e) \le c(e)$ $\forall e \in E$ (capacity)

$$\sum_{u \in V} f(u,v) = \sum_{w \in V} f(v,w) \quad \forall v \in V \quad v \neq s,t$$ conservation

Goal: Maximize total flow out of source vertex s.

Maximize : $|f| = \sum_w f(s,w) - \sum_u f(u,s)$

How to modify?

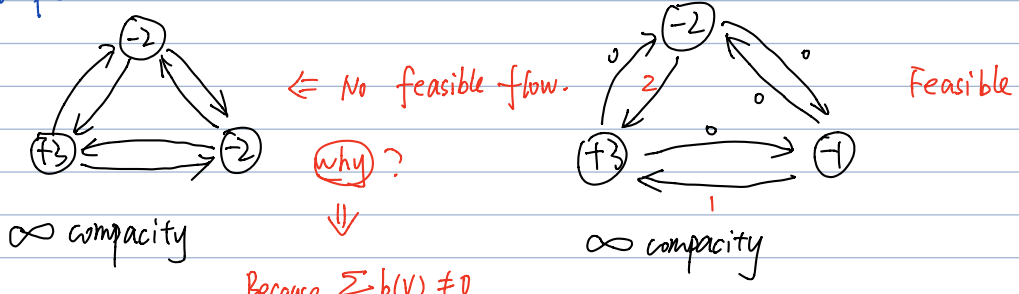
Balance $b(v)$ for every vertex $v \neq s,t$

Define: Feasible Flow
$f: E \to \mathbb{R}$ s.t. $0 \le f(e) \le c(e)$ $\forall e \in E$ (capacity)

$$\sum_{w \in V} f(v,w) - \sum_{u \in V} f(u,v) = b(v) \quad \forall v \in V$$ conservation

Goal: Find a feasible flow if one exists (从优化问题变为存在性问题)

Example:

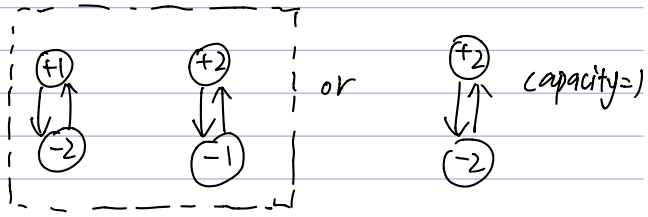


f is fesible $\Rightarrow$ $\sum_v b(v) = \sum_v \left(\sum_u f(u,v) - \sum_w f(v,w) \right)$

flow along all edges into v — flow away all edges out of v

$$= \sum_{u \to v} f(u,v) - \sum_{v \to w} f(v,w) = 0$$

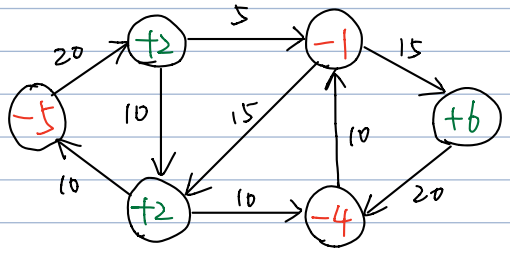
$\sum_v b(v) = 0$: necessary but not sufficient Example:



Intuition:
$b(v) > 0 \Rightarrow$ demand, compasition
$b(v) < 0 \Rightarrow$ supply, production

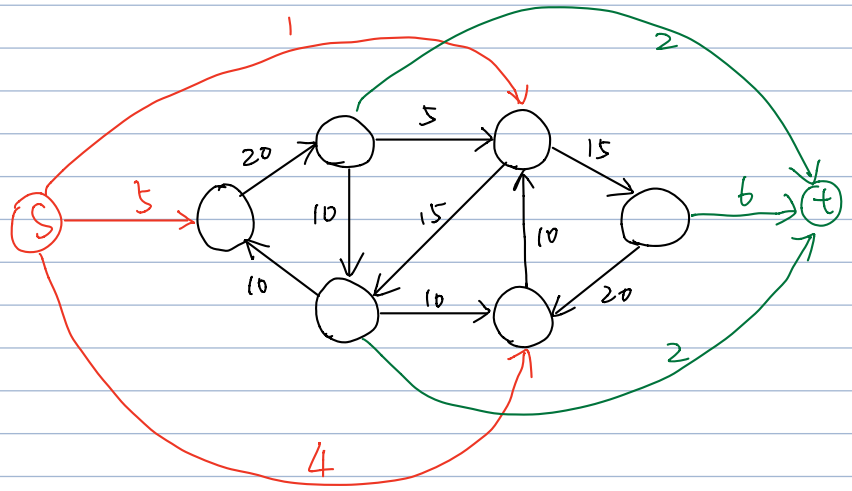
Solve by reducing to standard max flow

EX. G



Given : $G=(V,E)$, $c(e)$ for all $e \in E$ $b(v)$ for all $v \in V$

G'



construct $G'=(V',E')$

$V'=V \cup \{s,t\}$

$E'=E \cup \{(s,v): b(v)<0\}$
$\cup \{(v,t): b(v)>0\}$

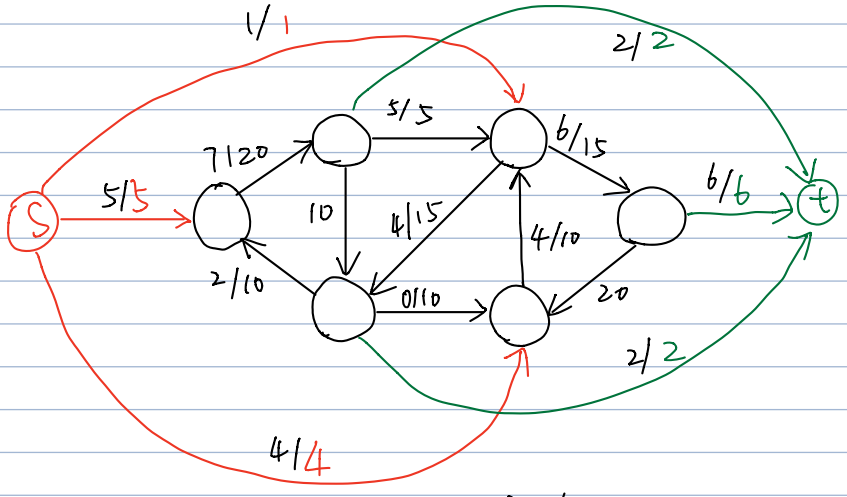
$c(e)$ for all $e \in E$.

$c(s,v)=-b(v)$ | 取絕對值
$c(v,t)=b(v)$

If there is a saturating flow f' in G' → feasible flow f in G.

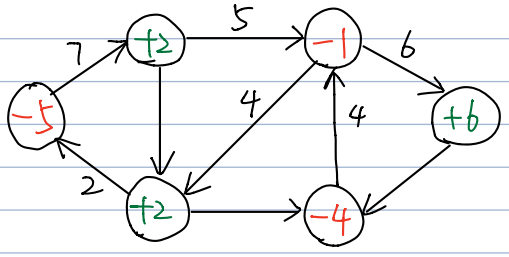
what is : saturating flow ? saturates all edges leaving s in G' (and entering t)

saturate : means : flow = capacity



we found a saturating flow

(dismiss the capacity)



f' is saturating $\Rightarrow$ $f'|_E = f$ Edmonds-Karp $O(E^2V)$

Week 8 Linear Programming (LP)

- Solve optimization problems
- Linear objective function
- Linear constraints

Given: n variables over $\mathbb{R}$
m linear equations / inequalities
Linear objective function

Goal: Assign real values to variables satisfy linear inequalities

Maximum / Minimize linear objective function

Example: Profit Maximization

2 produce A, B
How much of each produce to maximize profit?
. each unit of A: profit \$1
B: \$6

constraint: demand:
≤ 200 units of A per day
≤ 300 units of B per day

Total supply ≤ 400 units per day

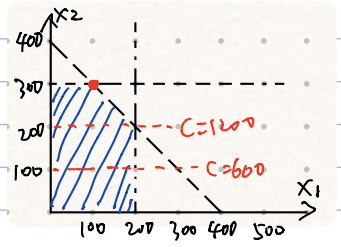
Associated LP: x_1 = # of units of A
x_2 = # of units of B

objective function: max $x_1 + 6x_2$

constraints: $x_1 \leq 200$
$x_2 \leq 300$
$x_1 + x_2 \leq 400$
$x_1, x_2 \geq 0$

Geometry of LP
five constraints = linear in equality half-space in 2-d plane (2 variable x_1, x_2)

Set of all feasible solutions of LP. = points (x_1, x_2) that satisfy all constraints.



feasible region

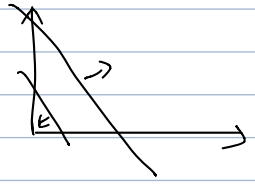
Goal: find max C where $x_1 + 6x_2 = C$ intersects feasible region
a point (x_1, x_2) in feasible region with max C is optimal solution

at point $(x_1, x_2) = (100, 300)$ profit = 1900 = optimum C
satisfy all constraints

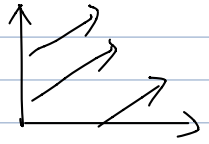
When is there no optimum solution?

① LP infeasible = feasible region empty

(constraints too tight) max $x_1 - x_2$
$2x_1 + x_2 \leq 1$
$x_1 + x_2 \geq 2$
$x_1, x_2 \geq 0$



② unbounded : max $x_1 + x_2$
$x_1, x_2 \geq 0$



Algorithm for solving LPs

- simplex algorithm (1947)
  - worst case running time is exponential
  - typically linear in size of problem
- polynomial time algorithm
  - ellipsoid method (1972), interior-point method (1967)

Simplex Algorithm: (= local search)

- start at a vertex of feasible polygon — in other case (0,0)
- search for an adjacent vertex (connected by an edge) of better objective value
- repeat until reach a vertex with no better neighbour
- return solutions as optimum (0,0) → (200,0) → (200,200) → (100,300)

LP in n dimensions

- constraints = half-space
- feasible region = intersection n+m half spaces
  = convex polyhedron in n dimensions

- vertices = corners of feasible regions
  = intersections of sides of feasible region
  = points satisfying n inequalities with equality satisfy remaining m inequalities
  $\binom{m+n}{n}$ vertices

neighbouring vertices share $n-1$ of their equality constraints.

How do we verify simplex "solution is optimal?"
- one way: check by checking all neighboring vertices
- alternative: LP duality

x_1 = # of units of A
x_2 = # of units of B

objective function : max $x_1 + 6x_2$

constraints : $x_1 \le 200$
$x_2 \le 300$
$x_1 + x_2 \le 400$
$x_1, x_2 \ge 0$

LP duality
Notice : 1st constraint + 6 · 2nd constraint
⇒ upper bound

$x_1 + 6x_2 \le 2000$

Can we find a tight upper bound sysmetically

⇒ LP duality : let $y = (y_1, y_2, y_3)$ with $y_1, y_2, y_3 \ge 0$

Multiplier	constraint
y_1	$x_1 \le 200$
y_2	$x_2 \le 300$
y_3	$x_1 + x_2 \le 400$

Then $\sum y_i$: constraint$_i$ = $(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \le 200y_1 + 300y_2 + 400y_3$

LHS : $x_1 + 6x_2$
obj function

RHS : upper bound on OPT solution
minimize to get tight bound

$\Rightarrow y_1 + y_3 \ge 1$

$y_2 + y_3 \ge 6$

Dual LP : min $200y_1 + 300y_2 + 400y_3$
constraint $y_1 + y_3 \ge 1$
$y_2 + y_3 \ge 6$
$y_1 + y_2 + y_3 \ge 0$

Dual LP : $(y_1, y_2, y_3) = (0, 5, 1) \Rightarrow 1900$

original LP :

objective function : max $x_1 + 6x_2$

constraints : $x_1 \le 200$
$x_2 \le 300$
$x_1 + x_2 \le 400$
$x_1, x_2 \ge 0$

Standard Form for LPs

variables : $x_1, x_2, \cdots, x_n$

OBJ function : $\max \sum_{j=1}^{n} c_j x_j$

constrains : $\sum_{j=1}^{n} a_{ij} x_j \le b_j$ for each $i = 1, \cdots, m$

$x_j \ge 0$ for each $j = 1 \cdots n$

LA $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ $c = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$

$m \times n$ constraint matrix $A \in \mathbb{R}^{m \times n}$

$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$ $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$

original LP	Dual LP
$\max c^T x$	$\min b^T y$
$Ax \le b$	$A^T y \ge c$
$x \ge 0$	$y \ge 0$

Ex. maximun flow as LP

capacity:

input: directed graph $G = (V, E)$, two specified vertices $s, t \in V$, every edge has $c(e) > 0$

output: flow f specified by $f(e)$ for every edge $e \in E$

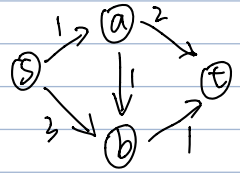
Maximizes : size $(f) = |f| = \sum_v f(s, v)$

constraints : for all $e \in E$ $\quad 0 \le f(e) \le c(e)$

for every $v \in V - \{s, t\}$: $\sum_{u:(u,v) \in E} f(u, v) = \sum_{z:(v,z) \in E} f(v, z)$

Example :

input



LP: define flow variables $f(i, j)$ on each edge $(i, j) \in E$

obj function : $\max f(s, a) + f(s, b)$

constraint : $0 \le f(s, a) \le 1$

$0 \le f(s, b) \le 3$ $\quad f(s, a) = f(a, b) + f(a, t)$

$0 \le f(a, b) \le 1$ $\quad f(s, b) + f(a, b) = f(b, t)$

variable : $f(e)$ for each edge

obj function : $\max \sum_{(s,v) \in E} f(s, v)$

constraints : for all $e \in E$, $0 \le f(e) \le c(e)$

for every $v \in V - \{s, t\}$:

$\sum_{u:(u,v) \in E} f(u, v) = \sum_{z:(v,z) \in E} f(v, z)$

$0 \le \int (a+t) \le 2$

$0 \le \int (b+t) \le 1$

Week 9 String Matching Problem (CLRS CH32)

Given 2 strings A text $T[1 \cdots n]$ and a pattern $P[1 \cdots m]$, is P a substring of T? (and if-so, where P)

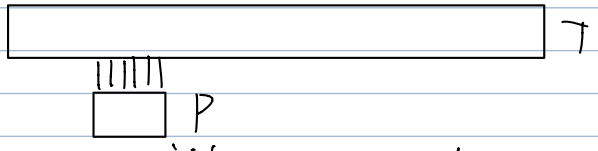
Ex: P: what type in Google
T: entire web

T is big, typically, and P is small. Treat both T and P as arrays.

$T[1 \cdots n]$ - Text
$P[1 \cdots m]$ - Pattern
$m \leq n$

Elements $\in$ Finite Alphabet Σ
elements: character

Obvious Way: Check each substring of T of length M
And see if matches P



Switch one over and so on

More formally, P occurs with shift s in T.
if $0 \leq s \leq n-m$ and $T[s+1, \ldots, s+m] = P[1 \cdots m]$

```
Simple-string-match (T[1...n], P[1...m]) // find small shifts or report no shift
01 for s=0 to n-m:
02     equal = true
03     i=1
04     while equal == True and i ≤ m
05         if T[s+i] ≠ P[i]
06             equal = false
07         else
08             i = i+1
09     if equal == true:
10         return s
11 return None
```

Running Time: $O((n-m) \cdot m) = O(mn)$

(n-m): # iteration
m: at most m compares

Worst case actually achieved: search for pattern: aaa···aab (m-1) in text aa····a (n) } very slow

Need at least $\Omega(m+n)$, could be $O(n^2)$, can do better? yes

- IDEA: have "sliding window" of size m
  characters T[s+1 ... s+m] is "visible"
  What to know: is it the same as P[1...m]?

IDEA: ① compute hash(P)
② for each length-m window of T,
compute hash(T[s+1 ... s+m]) and compute hash(P[1...m])
③ if =: check to see if strings really match
if ≠: move on to next S

compute hash values in O(1) times v.s. O(m) comparisons of characters

↓

How many iteration we need to compute hash? ⇒ n-m ⇒ #iterations stay the same

- Now we want the hash function such that we can go from one window to another in O(1) time

⇒ RABIN-KARP

① idea one: treat string as integers
Given: $|\Sigma| = d$ treat each string of characters in Σ as an integers in Radix-d notation.
Ex1. $\Sigma = \{0, 1, \dots, 9\}$ $d = 10$. string "274" has numberic value: $2 \cdot d^2 + 7 \cdot d^1 + 4 \cdot d^0 = 274$
Ex2. $\Sigma = \{a, b\}$ $d = 2$ mapping $a \to 0$ $b \to 1$. Then string "bab" $= 1 \cdot d^2 + 0 \cdot d^1 + 1 \cdot d^0 = 5$
Ex3. DNA: $\Sigma = \{A, C, G, T\}$, A 0, $d = 4$ ⇒ "TAT" = 51

C 1
G 2
T 3

RABIN-KARP ALGO

- need to compute hash values for P[1...m], T[s+1 ... s+m] for 1 ... n-m.
- Need an O(1) update hash value when slide window one position to right.

- P[1...m] ⇒ $\Theta(m)$ time (use Horner's Rule)

$$x = \sum_{i=1}^{m} a_i d^{m-i}$$

↓

$$p = P[m] + d(P[m-1] + d \cdot (P[m-2] \cdots + d(P(2) + d \cdot P(1))))$$

- Given substring T[1...m] let t_0 denote its decimal value, to computed in $\Theta(m)$
- How to do shift:
  ② second key idea:
  i-1 window: [i ... i+m-1]
  i window: [i+1 ... i+m]

$$t_i = (t_{i-1} - T[i] \cdot d^{m-1}) \cdot d + T[i+m]$$

Rabin-Karp-version 1 (T[1...n], P[1...m], $|\Sigma| = d$)

```
01  h = d^(m-1)
02  p = 0  // pattern value
03  t = 0  // substring value
04  for i = 1 to m:
05          p = 10·p + P[i]      } Horner's Rule
06          t = 10t + T[i]
07  for s = 0 to n-m:
08          if p == t:
```

```
09          return S
10      t = 10(t - h·T[s+1]) + T[s+m+1]
11  return None
```

Running Time: $O(m) + O(m) + O(n-m) = O(m+n) = O(n)$ $m \le n$

line 1 | line 4-6 | line 7-10

$O(n)$ worst case

Bounds # of arithmetic operations, since m may be large cannot assume $O(1)$

- **Rabin-Karp : Finger Printing**

One simple change to make efficient $d = |\Sigma|$
perform all arithmetic operations some prime q.

Choose q so that $d \cdot q$ fits into a standard interger variable

- $p \bmod q$ fingerprints of $P[1 \cdots m]$
- $t_s \bmod q$ fingerprints of $T[s+1, \ldots s+m]$
- $t_{s+1} \bmod q = (d(t_s - T[s+1] \cdot h) + T[s+1+m]) \bmod q$

⇓

New Problem : collision : $t_i = p$ $T[i+1 \cdots i+m] \neq P[1 \cdots m]$

① if $(p \bmod q) \neq (t_s \bmod q)$ $p \neq t_s$

② if $(p \bmod q) = (t_s \bmod q)$

⇒ p & t_s differ by $k \cdot q$, $k \in \mathbb{Z}$

⇒ if $p \neq t_s$ ⇒ False Match

e.g. T = 3 1 4 1 5 2 6 7 3 9 9 2

31415 mod 13 = 7 (△) 67399 mod 13 = 7 (△)

$q = 13$ $d = 10$

So we need brute force comparison $O(m)$

Rabin-Karp-version2 (T[1…n], P[1…m], $|\Sigma| = d$)

```
01  h = d^(m-1) mod q
02  p = 0  // pattern value
03  t = 0  // substring value
04  for i = 1 to m:
05          p = (10·p + P[i]) mod q
06          t = (10t + T[i]) mod q
07  for s = 0 to n-m:
08          if p == t:  → if P[1..m] = T[s+1 ... m]  // O(m)
09              return S
10          t = (10(t - h·T[s+1]) + T[s+m+1]) mod q
11  return None
```

Time : $O(n + F \cdot m)$

△ F : # False Matches

if mod q values are "random": $E(F) = \frac{n-m}{q} < \frac{n}{q}$

⇒ Time = $O(n + F \cdot m) \le O(n + \frac{n}{q} \cdot m)$ ⇒ want : $q > m$