30391355? int add $O(n)$ int mult $O(n^{\log_2 3})$ matrix mult $O(n^{\log_2 7})$

Two n-bit numbers: x, y.

$x = 2^{n/2}x_L + x_R$; $y = 2^{n/2}y_L + y_R$ $P_1 - P_2 - P_3$

$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$

$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R$.

$P_2 = x_L y_L$, $P_3 = x_R y_R$.

5432×6789 ⇒ 54×67 32×89 (54+32)×(67+89)

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \text{then } f(n) = O(g(n)) \\ c > 0 & \text{then } f(n) = \Theta(g(n)) \\ \infty & \text{then } f(n) = \Omega(g(n)) \end{cases}$

$T(n) = aT(n/b) + O(n^d)$

$d > \log_b a$ $T(n) = O(n^d)$

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

$d = \log_b a$ $T(n) = O(n^d \log_b n)$

$P(n,k) = \frac{n!}{(n-k)!}$ $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{1}{k!} \cdot n(n-1)\cdots(n-k+1) \cdot \text{poly}$ $\sum n^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$

12 $\log_2 n$ n^3 $2^{\log n}$ n n^3 2^n 2^n

Const < lo < sub linear < linear < poly < sub exp < exp < n! < n^n

①换底法 ②两边取 $\log_2$ ③放缩

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

depth = $\log n$

$\frac{n}{\log n} + \frac{n}{\log\frac{n}{2}} \cdot 2 + \ldots$

$\sum_{i=0}^{\log n} \frac{n}{\log n - i} = \frac{n}{\log n} + \frac{n}{\log n - 1} + \cdots + \frac{n}{\log n - (\log n - 1)}$

$= n \cdot \log\log n$

harmonic $\sum \frac{1}{k} = O(\log n)$

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

depth: $\log n$ ⇒ $n \log n$

$\frac{n}{3} + \frac{2n}{3} = n$

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

$n\log n + 2 \cdot \frac{n}{2}\log\frac{n}{2} + 4 \cdot \frac{n}{4}\log\frac{n}{4} \ldots = \sum_{i=0}^{\log n} n \log \frac{n}{2^i} = n \sum \log n - i = n(\log n \cdot \log n - \frac{\log n(\log n+1)}{2}) = O(n\log^2 n)$

④ $T(n) = 2T(n^{0.5}) + O(\log n)$

$m = \log n$ $n = 2^m$ $T(2^m) = 2T(2^{m/2}) + m$.

$S(m) = T(2^m)$ $S(m) = 2S(m/2) + m$. $S(m) = O(m \log m)$.

$T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log\log n)$.

⑤ $T(n) = T(n-1) + T(n-2)$

$T(n) \geq 2T(n-2) = \mathcal{R}(2^{\frac{n}{2}})$

$T(n) \leq 2T(n-1) = O(2^n)$

令 $T(n) = 2^{\Theta(n)} = \Theta(a^n)$

$a^n = a^{n-1} + a^{n-2}$ $a = \frac{1+\sqrt{5}}{2}$

$T(n) = 3T(n-2) + 5 = 3(3T(n-4) + 5) + 5$

layer $= \frac{n}{2}$ $= 3^{\frac{n}{2}} T(0) + 5(3^{\frac{n}{2}-1} + \cdots + 3^0)$

$\frac{a_1(1-q^n)}{1-q}$

$= O(3^{\frac{n}{2}})$

FFT

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

$O(n \log n) + O(n) + O(n\log n)$

$p(x) = 5 + x^2 + 2x^3$

1. **Coefficient** representation: {5, 0, 1, 2}
   a. Multiplication is O(n^2)
2. **Value** representation: ex. $\{p(0), p(\omega_n), p(\omega_n^2), \ldots p(\omega_n^2)\}$
   a. Multiplication is O(n)
   b. The product of two degree d polynomials is 2d
   c. A degree-2d polynomial is uniquely characterized by values at 2d + 1 distinct points.

$\omega_n = e^{\frac{2\pi i}{n}}$ ($n \geq 2d+1$ & n是2的幂次方) **nth roots** $1, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$

FFT

$$\begin{bmatrix} P(1) \\ P(\omega_n) \\ P(\omega_n^2) \\ \vdots \\ P(\omega_n^{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_{n-1} \end{bmatrix}$$

IFFT

$$M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$$

$$\frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \cdots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \cdots & \omega_n^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \cdots & \omega_n^{-(n-1)(n-1)} \end{bmatrix}$$

Conquer : Using pre-computed result from n/2th roots

Given $P(x) = p_0 + p_1x + p_2x^2 + \ldots + p_{n-1}x^{n-1}$ degree n

Let $E(x) = p_0 + p_2x^2 + p_4x^4 + \ldots$ so $E(x^2) = p_0 + p_2x + p_4x^2 + \ldots$ n/2

and $xO(x) = p_1 + p_3x^3 + p_5x^5 + \ldots$ so $O(x^2) = p_1 + p_3x + p_5x^2 + \ldots$ n/2

Then $P(x) = E(x^2) + xO(x^2)$ $E(x^2) == E((-x)^2)$

$P(-x) = E(x^2) - xO(x^2)$ $O(x^2) == O((-x)^2)$

```
def FFT(P):
    P - [p0, p1, ... pn-1]   coeff rep
    n = len(P)       n是2的幂次方
    if n == 1:
        return P
    ω = e^(2πi/n)
    Pe, Po = [p0, p2, ... pn-2], [p1, p3, ... pn-1]
    ye, yo = FFT(Pe), FFT(Po)
    y = [0] * n
    for j in range(n/2):
        y[j] = ye[j] + ω^j yo[j]
        y[j + n/2] = ye[j] - ω^j yo[j]
    return y
```

```
def IFFT(P):
    P - [P(ω^0), P(ω^1), ..., P(ω^(n-1))]   value rep
    n = len(P)
    if n == 1:
        return P
    ω = (1/n) * e^(-2πi/n)
    Pe, Po = P[::2], P[1::2]
    ye, yo = IFFT(Pe), IFFT(Po)
    y = [0] * n
    for j in range(n/2):
        y[j] = ye[j] + ω^j yo[j]
        y[j + n/2] = ye[j] - ω^j yo[j]
    return y
```

FFT(a, ω):

if $\omega = 1$ return a

$(s_0, s_1, \ldots, s_{n/2} - 1) = \text{FFT}((a_0, a_2, \ldots, a_{n-2}), \omega^2)$

$(s'_0, s'_1, \ldots, s'_{n/2} - 1) = \text{FFT}((a_1, a_3, \ldots, a_{n-1}), \omega^2)$

for $j = 0$ to $n/2 - 1$:

$r_j = s_j + \omega^j s'_j$

$r_{j+n/2} = s_j - \omega^j s'_j$

return $(r_0, r_1, \ldots, r_{n-1})$

$InverseFFT([a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]) = [0, 0, 2, 0, 0, 0, 0, 0]$.

$InverseFFT([a_0^3, a_1^3, a_2^3, a_3^3, a_4^3, a_5^3, a_6^3, a_7^3]) = [$ 800 $]$ $(2x^2)^3 = 8x^6$

P.Q. 2次FFT. 1次IFFT

FFT$(p_0, 1) = p_0$

3种application construct p.q. compute $pq = r$ via FFT.

use coeff/exponent of r to solve problem 指数

Convolution

$$r(x) = (p \cdot q)(x) = \left(\sum_{i=0}^{m} a_i x^i\right)\left(\sum_{j=0}^{n} b_j x^j\right) = \sum_{i=0}^{m}\sum_{j=0}^{n} a_i b_j x^i x^j = \sum_{i=0}^{m}\sum_{j=0}^{n} a_i b_{k-i} x^k = \sum_{k=0}^{m+n}\left(\sum_{i=0}^{k} a_i b_{k-i}\right) x^k$$

coeff of p * coeff of q

Construct coeff of p.q ⇒ $\sum_{i=0}^{k} a_i b_{k-i}$ (multiply then sum)

① Legoland visitors

Legoland is open for $2n$ days in the summer, and visitors arrive at the park for the first n days. On day i, a_i visitors arrive among visitors arriving on any given day, a p_t-fraction of visitors will leave after t-days (at t-th day)

More formally, we are given the following input:

1. Number of arrivals $\{a_1, a_2, \ldots a_n\}$.
2. $\{p_1, p_2, \ldots, p_n\}$ where p_t is the fraction of visitors that will spend t days.

find num of leaving visitor each day

leaving on day 1 2 3 4 d

0 a_1p_1 $a_1p_2 + a_2p_1$ $a_1p_3 + a_2p_2 + a_3p_1$ $\sum_{i=1}^{d-1} a_i p_{d-i}$ ($当 i > n, a_i = p_i = 0$)

$A(x) = \sum_{i=1} a_i x^i$ $P(x) = \sum_{j=1} p_j x^j$ $L(x) = (A \cdot P)(x) = \sum_{d=2}^{2n} c_d x^d$

O(n) O(nlogn)

c_d is num of leaving visitor on day d. $[0, c_2, c_3 \cdots c_{2n}]$

Cross Correlation / sliding dot product

$\sum_{i=0}^{k} a_i b_{k+i}$. k is length of a or b

Conv $\sum_{i=0}^{k} a_i b_{k-i}$ ⇒ reverse one of poly

$C = A \cdot B^R$. coeff of C is cross correlation of A,B

matching strings. 0→-1, 1→1

a0	a1	a2						Shift 0: $a_0b_0 + a_1b_1 + a_2b_2 = \sum_{i=0}^{m} a_i \cdot b_{i+0}$
	a0	a1	a2					Shift 1: $a_0b_1 + a_1b_2 + a_2b_3 = \sum_{i=0}^{m} a_i \cdot b_{i+1}$
		a0	a1	a2				Shift 2: $a_0b_2 + a_1b_3 + a_2b_4 = \sum_{i=0}^{m} a_i \cdot b_{i+2}$
b0	b1	b2	b3	b4	b5	b6	b7	

① string matching

Suppose we have a bitstring s of length $n+1$, and a pattern p (also a bitstring) of length $m+1 < n$. How do we efficiently find the (contiguous) substring of s which matches the pattern p at the largest number of positions?

Φ: 0→-1 1→1 largest dot product ⇒ largest similarity

$P(x) = \Phi(p_0) + \Phi(p_1)x^1 + \Phi(p_2)x^2 + \cdots + \Phi(p_m)x^m$ padding O(m+n)

$S(x) = \Phi(s_n) + \Phi(s_{n-1})x^1 + \Phi(s_{n-2})x^2 + \cdots + \Phi(s_0)x^n$

FFT ⇒ index k largest ⇒ return k-n is the starting point O(m+n)

O(nlogn)

② maximize dot (sub of A, B). m<n

We can use FFT. We reverse v, encode both v and a into a degree-n polynomial, and then multiply them using FFT in time $O(n \log n)$. Finally, we take the maximum coefficient of the middle $n - m + 1$ terms, which gives us our answer.

min-cut max-flow

Suppose $V = L \cup R$ is a minimum s-t cut in a directed graph $G = (V, E)$. Recall that in such a cut, s is in L and t is in R. In the following 4 questions, we will modify the capacity of an edge in the graph and consider how it affects the maximum flow from s to t. For each one, specify whether this modification will (1) necessarily increase the maximum flow, (2) necessarily decrease the maximum flow, (3) necessarily keep the maximum flow the same, or (4) it is not clear what effect it will have on the maximum flow. (Note that parts (c) and (d) below are not typos; they should indeed say "R to R".)

(a) We increase the capacity of an edge that goes from L to R. may have multiple min-cut

○ Increase ○ Decrease ○ Stay the same ✓ Not clear

(b) We decrease the capacity of an edge that goes from L to R.

○ Increase ✓ Decrease ○ Stay the same ○ Not clear

(c) We increase the capacity of an edge that goes from R to R.

○ Increase ○ Decrease ✓ Stay the same ○ Not clear

(d) We decrease the capacity of an edge that goes from R to R. may change min-cut

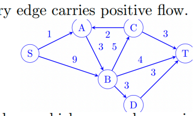
○ Increase ○ Decrease ○ Stay the same ✓ Not clear

all weight + constant. min-cut may change

× 正 int. unchange

(e) In any max flow, there is no directed cycle on which every edge carries positive flow.

F



(f) There exists a max flow such that there is no directed cycle on which every edge carries positive flow. T

3. There are connected undirected graphs G such that decreasing the capacity of any single edge in G does not change the value of the maximum flow.

○ True ○ False

Solution: False. Decrease those on the min-cut.

4. There are connected undirected graphs G such that increasing the capacity of any single edge in G does not change the value of the maximum flow.

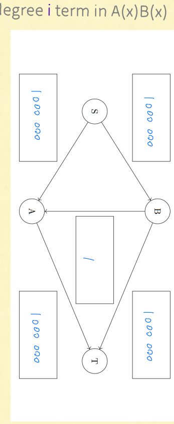
○ True ○ False

Solution: True, A-3-B-3.

5. There are directed graphs G such that decreasing the capacity of any single edge in G does not change the value of the maximum flow.

○ True ○ False

Solution: True, A-3->B-3->C, consider flow C->A.



Label the capacities of the edges of the above graph with integers between 1 and 1,000,000 inclusive so that

1. The max flow U of the graph G is equal to 2,000,000, AND
2. Ford-Fulkerson can take U iterations to compute the maximum flow.

You should assume that the Ford-Fulkerson algorithm sends the maximum amount of flow possible along the augmenting path it finds in each iteration.

Describe a sequence of augmenting paths that would make Ford-Fulkerson take U iterations.

2. Consider alternating between the augmenting paths $S \to B \to A \to T$ and then $S \to A \to B \to T$ a total of two million times. Ford-Fulkerson would send one unit of flow each iteration.

Now suppose that Ford-Fulkerson always selected its augmenting path to be a shortest s-t path in the residual graph. How many iterations would it take? Describe a sequence of augmenting paths it could use.

3. If we always select a shortest path, we would saturate $S \to A \to T$ followed by $S \to B \to T$ (or in the other order; it doesn't matter). Thus, Ford-Fulkerson would run for two iterations.

Quick select O(n)

Median: select ⌊n/2⌋+1 elt.

Select(k, S): $k = 7$ **S : 11,48,5,21,2,15,17,19,15**

Base Case: $k = 1$ and $|S| = 1$, return

Choose random pivot v $v = 15$

Form S_L containing all < v S_L : 11,5,2

Form S_v containing all = v S_v : 15,15

Form S_R containing all > v S_R : 48,21,17,19

If $k \leq |S_L|$, Select(k, S_L). 7 ≤ 3?

elseif $k \leq |S_L| + |S_v|$, return v. 7 ≤ 5?

else Select$(k - |S_L| - |S_v|, S_R)$ Select(2,[48,21,17,19])

Will eventually return 19, which is 7th element of list.

最坏1,最优n. 平均logn

binary search

```
def MS(A[1...n]):
    B = MS(A[1 ... n/2])
    C = MS(A[n/2 + 1 ...n])
    return merge(B, C)
```

Merge Sort O(nlogn)

① guess number

First, we guess $1, 2, 4, 8, \ldots$, the powers of 2, until we get to something that is greater than or equal to n. At this stage, we know n is less than our last guess, so we can transition to using binary search to guess n. We have used a total of around $2\log n$ guesses, which is sub-linear.

Quickselect

① |a|: median 51

pivot length k

70 69 51

30 39 21

47 22 4

51 0 0

② multi-query select. $[k_1, \cdots k_m]$ query $[a_1 \cdots a_n]$ select. O(n)

sort $[k_1 \cdots k_m]$ mlogm k^{th} largest

quick select find $k_{\frac{m}{2}}$ 在 a_i. partition list into L and R

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

O(n logm)

select $(k_{1\ldots\lfloor\frac{m}{2}\rfloor-1}, S_L)$ select (k, S_R)

k 更新 $k-\frac{m}{2}-1$

merge sort

① find inversion

[4 1 3 2]

procedure COUNT(A)

if length$[A] = 1$ **then**

return $A, 0$

$B, x \leftarrow$ COUNT(first half of A)

$C, y \leftarrow$ COUNT(rest of A)

$D \leftarrow$ empty list

$z \leftarrow 0$ count

while B is not empty or C is not empty **do**

merge

if B is empty **then**

Append C to D and remove elements from C

else if C is empty **then**

Append B to D and remove elements from B

else if $B[1] < C[1]$ **then**

Append $B[1]$ to D and remove $B[1]$ from B

else

Append $C[1]$ to D and remove $C[1]$ from C

$z \leftarrow z +$ length$[B]$

return $D, x + y + z$

Variation of k-sum

compute the number of ways to achieve some sum s

Construct poly by encoding the elements the exponents.

$r = ax^b \Rightarrow a$ is the number of ways to combine elements to achieve sum b.

① Triple sum

We are given an array $A[0, \ldots, n-1]$ with n integers in the range $[0, n-1]$ (not necessarily all distinct), and a non-negative integer s. We would like to know if there exist indices $0 \leq i, j, k \leq n-1$ (not necessarily distinct) such that $A[i] + A[j] + A[k] = s$

eg: 2-sum. $[1, 3, 5] \Rightarrow 2, 4, 6, 8, 10$

$p(x)^2 = (x^1 + x^3 + x^5)(x^1 + x^3 + x^5) = x^2 + 2x^4 + 3x^6 + 2x^8 + x^{10}$

3-sum. return yes or no. no need indices

$p(x) = x^{A[0]} + x^{A[1]} + \cdots + x^{A[n-1]}$. O(n)

FFT ⇒ $p(x)^3$ contains a sum of terms, where each term has the form

O(nlogn) $x^{A[i]} \cdot x^{A[j]} \cdot x^{A[k]} = x^{A[i]+A[j]+A[k]}$

check whether $p(x)^3$ contains x^s as a term.

② Ways to get cup

You want to buy a total of s cups. The store sells cups in packages of integer size c_1, c_2, c_3, c_4 all in the range $[1, s]$. you can only buy p of each package. compute the number of ways you can purchase exactly s cups

O(ps log(ps))

degree ≤ p·s

4-sum $A_1 = [0, c_1, 2c_1, \ldots, pc_1]$

O(p) $A_2 = [0, c_2, 2c_2, \ldots, pc_2]$

$A_3 = [0, c_3, 2c_3, \ldots, pc_3]$

$A_4 = [0, c_4, 2c_4, \ldots, pc_4]$

number of ways

$A_1[i] + A_2[j] + A_3[k] + A_4[l] = s$.

$f_1(x) = \sum_{i=0}^{p} x^{i \cdot c_1} = x^0 + x^{c_1} + x^{2c_1} + \cdots + x^{pc_1}$

$f_2(x) = \sum_{i=0}^{p} x^{i \cdot c_2}$ $f_3(x) = \sum_{i=0}^{p} x^{i \cdot c_3}$ $f_4(x) = \sum_{i=0}^{p} x^{i \cdot c_4}$

3次FFT ⇒ x^s 的 coeff 就是 num of ways

③ Water bucket

Set A of buckets of water of sizes $a_1, a_2, \ldots a_k$

Set B of buckets of water of sizes $b_1, b_2, \ldots b_k$

Set C of empty buckets of sizes 1, 2, 3, …

For each bucket in C (of size i), find the *number of ways* (n_i) you can pour one bucket from A and one from B to fill it exactly.

Let $A(x) = x^{a_1} + x^{a_2} + \ldots + x^{a_k}$, $B(x) = x^{b_1} + x^{b_2} + \ldots + x^{b_k}$

Then $A(x)B(x) = x^{a_1+b_1} + x^{a_1+b_2} + \ldots$

So n_i = coefficient of degree i term in A(x)B(x)

divide & conquer 设计都是一分为二 induction proof

Divide into subproblems $S_1(n)$

Solve subproblem $aT(n/b)$

Combine results (conquer) $S_2(n)$

$T(n) = a\,T(n/b) + \Theta(S_1(n) + S_2(n))$

④ root of poly

Design an algorithm to find the coefficients of a degree n polynomial with roots $(r_1, r_2, \ldots, r_n)$.

Let $m = n // 2$

Find coefficients of $a(x) = (x - r_1) \ldots (x - r_m)$ and $b(x) = (x - r_{m+1}) \ldots (x - r_n)$ recursively

Calculate the product $p(x) = a(x)b(x)$ using the FFT

$T(n) = 2T(n/2) + O(n \log n)$, so $T(n) = O(n\log^2 n)$

① Peak finding. just find one peak

$A[i] \geq A[i-1]$. $A[i] \geq A[i+1]$. (boundary just one)

$A = [4, 8, 5, 9, 10, 13, 4, 5, 6, 0]$ return one of $\{1, 5, 8\}$

Take the middle element of A, compare to neighbors

if ≥ both neighbor, Return its index

if ≤ left. recurse on left half not including middle element.

else. right

4	8	5	6	9	10	13	4	5	6	0
						13	4	5	6	0
									6	0

$T(n) = T(\frac{n}{2}) + O(1)$ ⇒ O(logn)

② max $A[i] + A[j]^2$, $i < j$

elements are non-negative

$A = [4, 8, 5, 9, 10, 13, 4, 5, 6, 0]$

return = 10 + 13^2 = 179

split into two halves A_1 A_2. optimal ① $i^*, j^* \in A_1$ ② $i^*, j^* \in A_2$

③ $i^* \in A_1$ and $j^* \in A_2$

FindMax$(A[0..n-1])$:

1. If $n \leq 1$, return $-\infty$.
2. Let $k := \lfloor n/2 \rfloor$. 下取整
3. Set $x :=$ FindMax$(A[0 \ldots k-1])$.
4. Set $y :=$ FindMax$(A[k \ldots n])$.
5. Set $z := \max(A[0], \ldots, A[k-1]) + (\max(A[k], \ldots, A[n]))^2$.
6. Return $\max(x, y, z)$.

$T(n) = 2T(\frac{n}{2}) + O(n)$ ⇒ O(nlogn)

⑤ undominated point

A point (x, y) is dominated by another point (x', y') if x < x' and y < y'.

$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, find all the undominated points.

(1) 暴力 $O(n^2)$

for point i:

for point j != i:

check if j dominates i

$(x_j > x_i$ && $y_j > y_i)$

(2) Find median of x-values M_x; divide S into S1, S2

U1 = Find_Undominated(S1)

U2 = Find_Undominated(S2)

Find_Undominated(S)

= U2 + {(x, y) in U1: y > max y-value in S2}

$T(n) = T(\frac{n}{2}) + O(n)$ median O(nlogn)

⑥ row 找到 min 所在位置出现 m×n O(n logm)

1	3	4	6	7	8
7	3	2	5	6	4
7	9	6	3	10	0

MONOTONE MATRIX MINS (A):

if A is empty:

return []

i = len(A)/2

j = RowMIN(A[i])

mins L = MONOTONE MATRIX MINS (A[:i, :j])

mins R = MONOTONE MATRIX MINS (A[i+1:, j+1:])

return mins L + [j] + mins R

③ birthday party $T(n) = 2T(\frac{n}{2}) + O(n\log n)$ FFT ⇒ $O(n\log^2 n)$

$\prod_{i=1}^{n/2}((1 - p_i) + p_i x) + \prod_{i=n/2+1}^{n}((1 - p_i) + p_i x)$ recursively

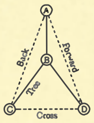
Graph: matrix 对角线为0。 adj list: 0:… 1:… l_i is length of one list

	Matrix	Adj. List
Edge (u,v)?	$O(1)$	$O(d)$
Neighbors of u	$O(\|V\|)$	$O(d)$
Space	$O(\|V\|^2)$	$O(\|E\|)$

Tree/forward edge (u,v): $int(v) \subset int(u)$, $inv(v) = [pre(v), post(v)]$
Forward (A,F): [10,11] in [0,13] or [0,[10,11],13]
Back edge (u,v): $int(u) \subset int(v)$. (C,B): [3,4] in [1,8] or [1,[3,4],8]
Cross edge (u,v): $int(v)$ < $int(u)$. (F,D): [2,5] before [10,11]

Types of Edges

- **Tree Edge**: part of the DFS forest DFS树边
- **Forward Edge**: leads to a non-child descendent in the tree
- **Back Edge**: leads to an ancestor in the tree. DFS不经过的边
- **Cross Edge**: leads to a node that's neither descendant nor ancestor

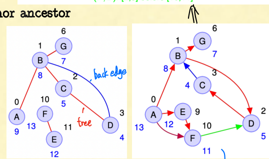


An edge $(u,v) \in E$ is a:

Tree/Forward Edge if
$pre(u) < pre(v) < post(v) < post(u)$ [[ ]]

Back Edge if
$pre(v) < pre(u) < post(u) < post(v)$ [[ ]]

Cross Edge if
$pre(v) < post(v) < pre(u) < post(u)$ [ ] [ ]



stack A A A A A A A A A / F F F /
有向图是否有循环
⇒ 找back edge. $O(V+E) + O(E)$

DAG. 有向无环
Topological sort (无环有向图)
⇒ 按post序号降序排列 A E F B G D C
at least one source/sink

SCC: $\forall u,v \in$ SCC. ∃ path $u \to v \to u$
every directed graph is a DAG of SCC
find SCC in DAG: $O(V+E)$
run DFS on G^R. 按post num 降序
reverse
run DFS on G. start node 按上述顺序
scc num +1

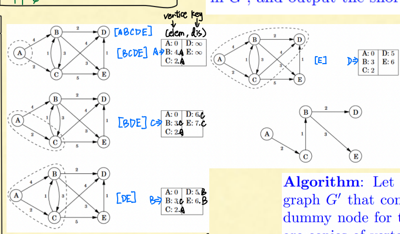
	Pseudocode	Explanation

DFS	`def explore(G,v):`
	`visited(v) = true`
	`previsit(v)`
	`for each edge(v,u) in E:`
	`if not visited(u):`
	`explore(u)`
	`postvisit(v)`
	
	`def dfs(G):` clock = 0, 在first. cc num = 0
	`for all v in V:`
	`if not visited(v):`
	`explore(v)` cc num +=1
		- Runtime: $O(\|V\|+\|E\|)$
		- Uses a stack
		previsit(v):
		Set pre[v] := clock.
		clock := clock+1
		postvisit(v):
		Set post[v] := clock.
		clock := clock+1
		topo_sort_list.append(u)
		reverse topo_sort_list

BFS	`def bfs(G, s):`
	`for all u in V:`
	`dist(u) = infinity`
	`dist(s) = 0`
	`Q = [s]`
	`while Q is not empty:`
	`v = Q.eject()`
	`for each edge(v,u) in E:`
	`if dist(u) == infinity:`
	`Q.inject(u)`
	`dist(u) = dist(v) + 1`
		- Runtime: $O(\|V\|+\|E\|)$
		- Uses a queue
		- find shortest path in unweighted graph
		u / Q: S [S]; S [ACD]; A [CDB]; C [DB]; D [BEF]; B [EF]; E [F]; F [ ]
		dist: S; dist1: ACD; dist2: B; dist2: EF

Dijkstra's Pseudocode

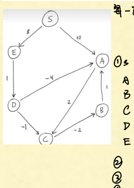
```
def dijkstra(G,l,s):          (|V|+|E|)log|V|
  for all u in V:
    dist(u) = infinity
    prev(u) = nil              no negative edges
  dist(s) = 0                  priority queue
  H = makequeue(V)
  while H is not empty:
    u = deletemin(H)
    for all edges (u,v) in E:
      if dist(v) > dist(u) + l(u,v)
        dist(v) = dist(u) + l(u,v)
        prev(v) = u
        decreasekey(H,v)
```



Bellman-Ford Pseudocode

```
def bellman-ford(G,l,s):      O(|V||E|)
  for all u in V :
    dist(u) = infinity         negative edge
    prev(u) = nil              update |V|-1 times
  dist(s) = 0
  repeat |V| - 1 times:
    for all e in E:            detect neg cycle
      update(e)                ⇒ |V| times. value change

def update(e = (u, v)):
  d(v) = min(d(v), d(u)+l(u,v))
```



每一轮更新 S→E 顺序

	S	A	B	C	D	E
	0	∞	∞	∞	∞	∞
S	0	10				8
A	0	10		12		8
B	不变					
C	0	10	10	12		8
D	不变					
E	0	10	10	12	9	8
②	0	5	10	8	9	8
③						
④						
⑤						

DAG SSSP. Single Source Shortest Path.

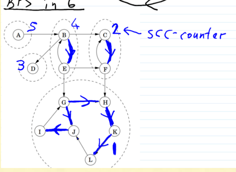
```
def dag-shortest path(G,l,s):
  run DFS on DAG. reverse order vertice by post
  ∀ u in the order   topo order    O(|V|+|E|)
    ∀ (u,v) ∈ E : update (u,v)     allow neg edges
```

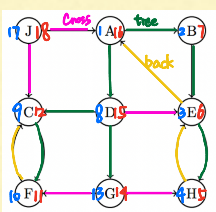
SCC

Ordered by post in G^T

G I J L K H C F D B E A

BFS in G





Huffman Coding priority queue

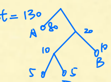
Goal: Given characters/frequencies $\{(i, f_i)\}$

Main Idea: Find the two symbols with the smallest frequencies, say i and j, and make them children of a new node (meta-node), which then has frequency $f_i + f_j$, and repeat until one node remaining.

$f_1 = 80$ $f_2 = 10$ $f_3 = 5$ $f_4 = 5$

higher node must has larger f than lower

Cost = 130



$f_1, \ldots f_n$, possible longest code → n-1

Horn-SAT

$F = (\bar{a} \vee b) \wedge \bar{c} \wedge d$ — clause: C_1, C_2, C_3 — SAT, CNF

- Boolean variables and any number of the two types of clauses:
- Implications:
  - $(a \wedge b \wedge \ldots) \Rightarrow c$
  - AND of any number of positive literals
  - Implies a single positive literal
- Negative Clauses:
  - (!a V !b V !c)
  - OR of any number of negative literals
- Must satisfy all implications and clauses

(a.b.c. d.e) → (T.F.F.T.T) ⇒ (T.F.F.T.F) X
(T.T.F.T.T) ✓

Algorithm:
1. Set all variables to be false
2. If there is an implication unsatisfied, set the right side variable to true
   Repeat until all implications are satisfied
3. If all the pure negatives are satisfied we did it! Return the assignments
4. If not, we did not do it :(return "not satisfiable"

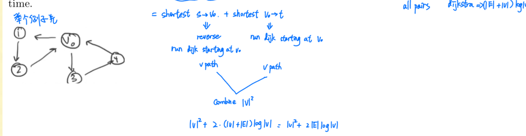
generate minimum True to satisfy SAT

Dijkstra

① shortest path through v_0 — reverse path. start from center

You are given a strongly connected directed graph $G = (V, E)$ with positive edge weights, and there is a special node $v_0 \in V$. Give an efficient algorithm that computes, for all node pairs s, t, the length of the shortest path from s to t that passes through v_0. Your algorithm should take $O(|V|^2 + |E| \log |V|)$ time.

all pairs. Dijkstra $O((|V|+|E|)\log|V|)$



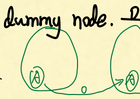
⑦ dummy node → n个起点. w = 0
run on reverse graph

② Booking flight

We want to book a cheap flight route to travel from city s to city t. There are n cities with airports including s and t. Airlines offer m flights, where the ith flight goes non-stop from city u_i to v_i and costs c_i dollars. We wish to find the cheapest route from s to t, but if there are multiple cheapest routes, we wish to find one with the fewest flights

If dist(v) > dist(u) + c(u, v)
dist(v) = dist(u) + c(u, v), l(v) = l(u) + 1
If dist(v) = dist(u) + c(u, v)
l(v) = min(l(v), l(u) + 1)

⑤ n个区选上选k个. dummy node. → k个点 $\binom{n}{k}$种



③ A类点 → B类 → 七大*

Main idea: Create $k + 1$ copies of G, called $G_0, G_1, \ldots G_k$, to form G'. Let the copy of v in G_i be v_i. For every v in S_i, we add an edge from v_{i-1} to v_i with weight 0. We run Dijkstra's starting from h_0 in G', and output the shortest path length to any vertex in G_k.

④ from 1 to n. n node. m flight. k for free. min cost
create k+1 copies of graph. $k(m+n)\log(kn)$
for each $(u_i, v_i) \in E$. $l \in \{1,2 \cdots k\}$. $= k(m+n)\log(n)$
add weight 0 edge from l-th copy of u_i to (l+1)th copy of v_i

⑥ k选走. 跳过t个点

Algorithm: Let graph G be the same graph as in part a). For this part we will construct a new graph G' that contains t copies of graph G (but dummy node is not copied since we only need one dummy node for the entire graph G'). Specifically, we start with the set of vertices $v_1, \ldots, v_t$ which are copies of vertex v at levels 1 through t, and a set of edges (u_l, v_l) corresponding to (u, v) in the original graph for each level $l \in \{1, \ldots, t\}$. Then, for every edge (u, v) in the original graph, we will add a directed edge going from u_l to v_{l+1} and from v_l to u_{l+1} with weight of 0. We also add a directed edge going from v_l to v_{l+1} with weight 0. These edges connect each level of our graph with the next level. Finally we run Dijkstra on G' and the delivery times we get for vertices at level k will be the new delivery time for each household.

Bellman-Ford Guaranteed / possible Correct

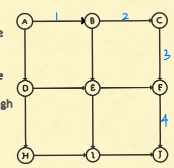
① Suppose we run the Bellman-Ford algorithm on the following graph to compute shortest paths from the vertex A. Let T denote the number of individual "update" calls after which the value of dist[J] reaches its correct value (which may occur prior to the algorithm's termination). T depends on the actual weights on the edges and the order in which the edges are updated. Assume there's a fixed ordering of the edges. That is, we always go through the edges in the same order every time we run for all e in E.

- What's the smallest possible value for T? 4
- What's the largest possible value for T? 45

path from A to J always length 4
assume A→B→C→F→J

8 edge 4 3 2 1
8 edge 4 3 2 1
8 edge 4 3 2 1
8 edge 4 3 2 1
8 edge 4 3 2 1

4×12−3 = 45



If every shortest path from s to any vertex uses at most k edges, then updating all edges k times is sufficient to compute shortest path distances from s. True

② 利率 $-\log r_{ij}$ + Bell

Minimum Spanning Trees (MST)

Tree: $|E| = |V| - 1$

Goal: Given a weighted undirected graph $G = (V, E)$, find the lightest weight tree that connects all vertices V.

Cut Property: The lightest edge across a cut is in some MST.

Cut Property 2: Suppose edges X are part of some MST. Let $(S, V \setminus S)$ be any cut for which edges in X do not cross the cut, and let e be the lightest edge across the cut. Then edges $X \cup \{e\}$ are part of some MST.

Kruskal's Algorithm $O(|E|\log(|V|))$
1. choose edge in order of weights (sort)
2. skip an edge if it create circle
disjoint set. 不相交集合

```
for all u ∈ V :
    makeset(u)
X = {}
Sort the edges E by weight
for {u,v} ∈ E, increasing order
    if find(u) ≠ find(v):
        add edge {u,v} to X
        union(u,v)
```

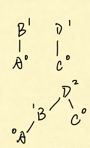
Prim's Algorithm $O(|E|\log(|V|))$
On each iteration, pick the lightest edge between a vertex in the current subtree S and a vertex outside S. repeat $|V|-1$ time

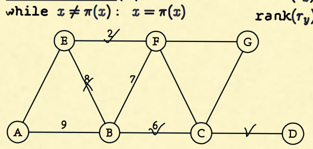
Union find — π. parent; rank. height under x
小树 → 大树
2个等rank 树 → rank+1
n element $O(\log n)$

```
procedure makeset(x)
π(x) = x
rank(x) = 0

function find(x)
while x ≠ π(x): x = π(x)

procedure union(x, y)
r_x = find(x)
r_y = find(y)
if r_x = r_y: return
if rank(r_x) > rank(r_y):
    π(r_y) = r_x
else:
    π(r_x) = r_y
    if rank(r_x) = rank(r_y):
        rank(r_y) = rank(r_y) + 1
```





Edge (E, F) must be part of every MST because it's the smallest edge across the cut $\{A, E\}$ (or the cut $\{A, E, B\}$).
Edge (B, C) must be part of every MST because it's the smallest edge across the cut $\{B\}$.
Edge (C, D) must be part of every MST because it's the only edge across the cut $\{D\}$, and hence must be the smallest edge.
(E, B) (as it is the largest edge in a cycle).

SCC

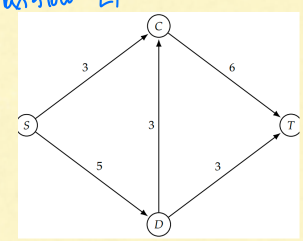
1. The vertex with the smallest *pre* value is
   - (A) necessarily part of a source SCC.
   - (B) necessarily part of a sink SCC.
   - (C) none of the above. ✓

2. The vertex with the largest *post* value is
   - (A) necessarily part of a source SCC. ✓
   - (B) necessarily part of a sink SCC.
   - (C) none of the above.

3. If the graph G is strongly connected then
   - (A) there is necessarily an edge from $u \to v$.
   - (B) there is necessarily an edge from $v \to u$.
   - (C) vertex u is the same as vertex v. ✓
   - (D) none of the above.

4. The vertex with the largest *pre* value
   - (A) is necessarily in a sink SCC.
   - (B) is necessarily not in a source SCC.
   - (C) none of the above. ✓

max flow LP



$$\max \quad x_{sc} + x_{sd}$$
$$\text{s.t.} \quad x_{sc} + x_{dc} - x_{ct} = 0$$
$$x_{dc} + x_{dt} - x_{sd} = 0$$
$$x_{sc} \le 3, x_{sd} \le 5, x_{dc} \le 3, x_{ct} \le 6, x_{dt} \le 3, x \ge 0$$

Network Flow / Bipartite Perfect Matching

Premise: Given some network (Graph G, edges with capacities, nodes s, t) find the maximum flow from source s to sink t. Subject to:

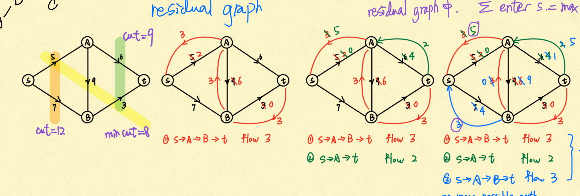
- 0 <= f_e (flow from u to v in e=(u,v)) <= c_e (capacity of e)
- For nodes != s,t: flow in = flow out

** If all capacities are integers, then there is a max flow st. the flow on each edge will be an integer.

Ford-Fulkerson algorithm — BFS to find paths $O(V \cdot |E|^2)$
augmenting path: not saturated yet (greedy)

1. Find a path from s to t and route max flow through this path
2. Update capacities by updating the residual graph
3. Repeat 1 and 2 until algorithm halts (no path from s to t is found)

residual graph — residual graph中. Σ enter s = max flow



Max flow ≤ / = min cut (dual)

An (s,t)-cut partitions V into two subsets S, T, where $s \in S$, $t \in T$. **Capacity of** (s,t)**-cut** C: $\text{val}(C) = \sum_{e \in \{(u,v) | u \in S, v \in T\}} c_e$.

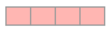
Bipartite perfect matching. ∃ perfect matching ⟺ max flow ≥ n from s to t

Dynamic Programming ⇒ DAG. node is subproblem, edge is dependency

High Level Approach: Define subproblems s.t. the solution to a big problem can be easily derived from the solutions to its subproblems. Solve all subproblems from small to large, using results from previous subproblems to solve the current subproblem. Both recursion with memoization (top-down) and iteration (bottom-up) approaches exist.

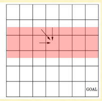
(1) subproblem
(2) recursion relation
(3) dependency/calculation order

time complexity: O(# subproblem × time for one subproblem)
space complexity: $L(j) = 1 + \max_{i<j, a_i<a_j} L(i)$



L(i) = need to check all previous subproblems

E(i, j) = only checks subproblems from 2 most recent rows



Longest Increasing Subsequence	$L(i)$ = length of longest increasing subsequence ending at i	$L(j) = 1 + \max_{i<j, a_i<a_j} L(i)$
Edit Distance	$E(i,j)$ = edit distance between strings $x[1...i]$ and $y[1...j]$	$E(i,j) = \min \begin{cases} 1 + E(i-1, j), \\ 1 + E(i, j-1), \\ diff(i,j) + E(i-1, j-1), \end{cases}$ diff(i, j) = 0 if x[i] = y[j] else 1
Knapsack w/ rep	$K(w)$ = maximum possible value with knapsack of capacity w	$K(w) = \max_{i: w_i \le w} \{K(w - w_i) + v_i\}$
Knapsack w/o rep	$K(w)$ = maximum possible value with knapsack of capacity w and items $1, \ldots, j$	$K(w, j) = \max \begin{cases} K(w - w_j, j-1) + v_j, \\ K(w, j-1) \end{cases}$
Matrix Multiplication	$C(i,j)$ = minimum cost of multiplying $A_i, \ldots, A_j$	$C(i,j) = \min_{i \le k < j} \{C(i,k) + C(k+1,j) + m_{i-1} * m_k * m_j\}$

① longest path (DAG. no weight) O(V+E)

(1) subproblem L(u)
(2) recursion relation $L(v) = \max_{(u,v) \in E} L(u) + 1$
(3) dependency/calculation order topological search
(4) input L=0 if no incoming edge

② longest increasing sequence O(n²)

1 5 7 3 6 9 10
1 7 9 10 change into DAG
1 5 7 9 10 (i,j) ∈ E iff i<j & $a_i < a_j$
1 3 6 9 10 check if $a_i < a_j$ for $\binom{n}{2}$

③ edit distance O(mn). parallel ⇒ O(n+m)

min cost. keep/insert/delete/substitute
(1) subproblem edit distance between prefix
(2) recursion relation

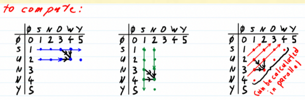
E[SUNNY, SNOWY] = min { E[SUNN, SNOW] keep; E[SUNNY, SNOW] + 1 insert; E[SUNN, SNOWY] + 1 delete (delete original y of sunny) }
SNNY, SNOY, SNOWY cost=3

$E[i,j] = \min \begin{cases} E[i, j-1] + 1 & \text{Insert} \\ E[i-1, j] + 1 & \text{Delete} \\ E[i-1, j-1] + Diff(x_i, y_j) & \text{keep/replace} \end{cases}$

(3) dependency/calculation order

Dependencies: E[i,j] depends on E[i, j-1], E[i-1, j], E[i-1, j-1]
For i=1…n
For j=1…m
or 相反

Order to compute:



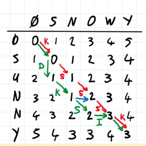
Run Time	O(nm)	O(nm)	O(n+m) in parallel
Space	O(m)	O(n)	O(n+m)

(4) input
E[i:0] = i
E[0:j] = j
output
E[n,m]

modify
E[i:0] = 3i
E[0:j] = 2j
(delete 3. insert 2)

找到所有 edit path. O(mn). cost 只作不上. 因此右下角只可 keep

S[i,j] ∈ {↖, ↖, ↓, →}



	a	p	p	l	e
	0	1	2	3	4	5
0						
P x_1 1		1	1			
a x_2 2		1				
l x_3 3					2	
e x_4 4						2

④ knapsack exponential in log w

O(nW)

Example:

Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

W=10 weight limit

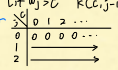
Without replac. Items 1, 3 → $36
With replac. Items 1, 4, 4 → $38

(1) with replacement

(1) subproblem K(C). max value with weight limit C
(2) recursion relation $K(C) = \max_{1 \le i \le n, w_i \le C} \{V_i + K(C - w_i)\}$
(3) dependency/calculation order C=1…W space O(W+n)
(4) input K(0)=0. input V. W [1…n]
output K(W)

(2) without replacement

(1) subproblem K(C,j) max value with weight limit C & subset {1…j}
(2) recursion relation $K(C,j) = \begin{cases} \text{if } w_j \le C & \max\{V_j + K(C - w_j, j-1), K(C, j-1)\} \\ \text{if } w_j > C & K(C, j-1) \end{cases}$
(3) dependency/calculation order space O(nW)



O(n+W) vector v&w 2行
(4) input K(C,0)=0 for all C

⑤ shortest path (DAG. weighted. single source) O(V.E)

$dist(v) = \min_u \{dist(u) + w_{uv}\}$ is not a subproblem
[O(V.(E+V))] as worst case of Bellman-Ford.

(1) subproblem dist(v, k) = length of shortest path from s to v. use ≤ k edges (# edge)
(2) recursion relation $dist(v,k) = \min \begin{cases} dist(v, k-1) \\ \min_{(u,v) \in E} \{dist(u, k-1) + l(u,v)\} \end{cases}$
(3) dependency/calculation order for k=1…n-1. for all v
(4) input dist(v,0) = 0
output dist(v, n-1)

```
For v∈V : dist(v,0) = ∞
dist(v,0) = 0
For k=1, …, n-1
 For all v∈V
   dist(v,k) = dist(v,k-1)
   For all (u,v)∈E
     If dist(v,k) > dist(u,k-1) + l(u,v)
        dist(v,k) = dist(u,k-1) + l(u,v)
∀v∈V Output dist(v,n-1)
```

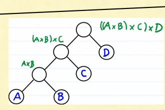
Floyd-Warshall

⑥ all pairs shortest path (not single source) O(n³)

(1) subproblem dist(i,j;k): vertex i to vertex j use intermediate vertex {1,2…k}
(2) recursion relation $dist(i,j;k) = \min \begin{cases} dist(i,j;k-1) & \text{not use vertex } k \\ dist(i,k;k-1) + dist(k,j;k-1) & \text{use vertex } k \end{cases}$
(3) dependency/calculation order For k=1…n, For i=1…n, For j=1…n
(4) input dist(i,j;0) = l(i,j). dist(i,i;0) = ∞
output dist(i,j;n)

⑨ matrix mult O(n³)

m×k · k×n ⇒ time m×k×n
binary tree. can't change order



(1) subproblem C(i,j): optimal runtime for $A_i \times A_{i+1} \cdots \times A_j$
(2) recursion relation $C(i,j) = \min_{i \le k \le j} \{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m_j\}$

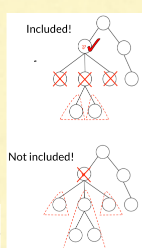
⑦ TSP O(n²·2ⁿ) space O(n·2ⁿ)

n cities. find path { start & end at 1. visit all city once. min length }
(1) subproblem city subset S = {1…k}
C(S). TSP on S
C(S,j). length of shortest path. 1 to j∈S & j≠1 visit all city in S once
(2) recursion relation C(S) = min{C(S,j) + d_{j1}}
$C(S,j) = \min_{i \in S, i \ne j} \{C(S \setminus \{j\}, i) + d_{ij}\}$

```
C({1}, 1) = 0
for s = 2 to n:
  for all subsets S ⊆ {1, 2, ..., n} of size s and containing 1:
    C(S, 1) = ∞
    for all j ∈ S, j ≠ 1:
      C(S, j) = min{C(S − {j}, i) + d_ij : i ∈ S, i ≠ j}
return min_j C({1, ..., n}, j) + d_j1
```

⑧ independent set O(V)

set of vertex with no edge connect them
max size
(1) subproblem I(v). problem at root v
(2) recursion relation ① v is included
I(v) = 1 + Σ I(u)   u: grandson of v
② v is not included
I(v) = Σ I(u)   u: son of v



$$I[v] = \max\left\{1 + \sum_{u:\text{grandchild of } v} I[u], \sum_{u:\text{ child of } v} I[u]\right\}$$

(3) dependency/calculation order leaf to root (topo search)

```
for i = 1 to n:  C(i, i) = 0
for s = 1 to n − 1:
  for i = 1 to n − s:
    j = i + s
    C(i, j) = min{C(i, k) + C(k + 1, j) + m_{i−1} · m_k · m_j : i ≤ k < j}
return C(1, n)
```

① i=1…n-1, j=2…n   ② i=1…n-2, j=3…n

LP & duality

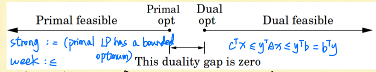
Constraint Transformations:

Changing Objective	Inequality to Equality
$\max \mathbf{c}^T\mathbf{x} = \min -\mathbf{c}^T\mathbf{x}$ $\min \mathbf{c}^T\mathbf{x} = \max -\mathbf{c}^T\mathbf{x}$	$ax \le b \rightarrow ax + s = b,$ $s \ge 0$
Equality to Inequality	**Unrestricted Variable**
$ax = b \rightarrow ax \le b,$ $ax \ge b$	$x \in \mathbb{R} \rightarrow x = x_+ - x_-$ $x_+, x_- \ge 0$

$|x_1 + x_2| \le b_2$. $x_1, x_2 \in \mathbb{R}$
$(x_1^+ - x_1^-) + (x_2^+ - x_2^-) \le b_2$
$(x_1^+ - x_1^-) + (x_2^+ - x_2^-) \ge -b_2$

Primal: Canonical
$\max \mathbf{c}^T\mathbf{x}$ n variable
$\mathbf{A}\mathbf{x} \le \mathbf{b}$ m+n constraint
$\mathbf{x} \ge 0$

Dual: y is magic number
$\min \mathbf{y}^T\mathbf{b}$
$\mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T$
$\mathbf{y} \ge 0$



strong: = (primal LP has a bounded optimum)
week: ≤ $c^Tx \le y^TAx \le y^Tb = b^Ty$
This duality gap is zero

vertex: intersection of ≥ n constraint. ≤ $\binom{m}{n}$
convex polyhedron. bounded. non-empty feasible region ⇒ optimum at vertex
brute-force. $\binom{m}{n} \cdot n^3$ (gaussian elimination)
simplex. start at a vertex. find neighbor with highest value. move to it.
O(n³. mn) per step
2 optimal ⇒ whole edge optimal

LP 概念

1. **Feasible:** 1 or more optimum assignments. Bounded or unbounded feasible region
2. **Infeasible:** No assignments even satisfy the constraints. No feasible region
3. **Unbounded:** No single best assignment. Unbounded feasible region

unbounded optimal ⇒ unbounded feasible region
⇍

$$\max x + y$$
$$ax + by \le 1$$
$$x, y \ge 0$$

(a) **Is infeasible.** LP is never infeasible as the (0,0) satisfies for any choice of a and b
(b) **Is unbounded.** a≤0 or b≤0 . If a ≤ 0,then we can increase x arbitrarily, (same thing for y if b≤0)
(c) **Has a unique optimal solution.**
a≠b, a>0, b>0. The have to be positive, otherwise the LP is unbounded. If a>b, the optimal is (0,1/b). If a<b, the optimal is (1/a,0)

zero-sum game pay-off is for row player

Example:

	B1	B2
A1	a	c
A2	b	d

A's strategy is $[x_1, x_2]$ ≥ε $x_1 + x_2 = 1$
B's strategy is $[y_1, y_2]$ max k. k ≤ ax₁+bx₂. k ≤ cx₁+dx₂. $x_1, x_2 \ge 0$
row first
A picks $[x_1, x_2]$ to maximize the value $\min\{ax_1 + bx_2, cx_1 + dx_2\}$
B picks $[y_1, y_2]$ to minimize the value $\max\{ay_1 + cy_2, by_1 + dy_2\}$
col first
Min-Max Theorem: $\max_x \min_y \sum_{i,j} G_{ij} x_i y_j = \min_y \max_x \sum_{i,j} G_{ij} x_i y_j$
≤

row first. P_1, P_2 选 row1. row2
col 选 col1. row get $ap_1 + bp_2$.
col 选 col2. row get $cp_1 + dp_2$.
col 选 min{$ap_1 + bp_2$, $cp_1 + dp_2$}
row strategy ⇒ max min{$ap_1 + bp_2$, $cp_1 + dp_2$}

col first. q_1, q_2 选 col1. col2
row 选 row1. col get $aq_1 + cq_2$
row 选 row2. col get $bq_1 + dq_2$
row 选 max{$aq_1 + cq_2$, $bq_1 + dq_2$}
row strategy ⇒ min max{$aq_1 + cq_2$, $bq_1 + dq_2$}

expected value $ap_1q_1 + cp_1q_2 + bp_2q_1 + dp_2q_2$
The optimal value of zero-sum game is unique, but this value may be achieved with multiple strategy
optimal strategy is sometimes deterministic

dual

a. If the primal is unbounded, dual is infeasible
b. If primal is infeasible, dual is unbounded/infeasible

	max $3x_1 + 6x_2 - 7x_3$	Min $100y_1 + 20y_2 + 32y_3$
		Constraints:
y_1	$x_1 + x_2 + x_3 \le 100$	$y_1 + y_3 \ge 3$,
y_2	$6x_2 - 7x_3 \le 20$	$y_1 + 6y_2 + y_3 \ge 6$,
y_3	$x_1 + x_2 \le 32$	$y_1 - 7y_2 \ge -7$,
不管≤	$x_1, x_2, x_3 \ge 0$	$y_1, y_2, y_3 \ge 0$

$(y_1+y_3)x_1 + (y_1+6y_2+y_3)x_2 + (y_1-7y_2)x_3$
<=
$100y_1 + 20y_2 + 32y_3$

Maximize $-8x - 7y - 2z$
Subject to $+5y + 2z \le 8$,
$-5x - 2z \le 7$,
$-2x + 2y \le 2$,
$x, y, z \ge 0$.

Minimize $8a + 7b + 2c$
Subject to $-5b - 2c \ge -8$,
$5a + 2c \ge -7$,
$2a - 2b \ge -2$,
$a, b, c \ge 0$.

optimal ⇒ primal = dual
feasible ⇒ primal = v. dual = −v

(**2 points**) Suppose (x, y, z) is a feasible solution to the primal linear program Prime from above with objective value $v \in \mathbb{R}$. Explain why $a = x, b = y, c = z$ is also a feasible solution to the dual linear program Dual, and give its dual objective value (in terms of v).

Solution: The inequalities in the dual LP are identical to the inequalities in the primal LP, so the feasible sets are the same. However, the objective function is negated. Hence, if x, y, z has value v, then $a = x, b = y, c = z$ has value $-v$

P/NP

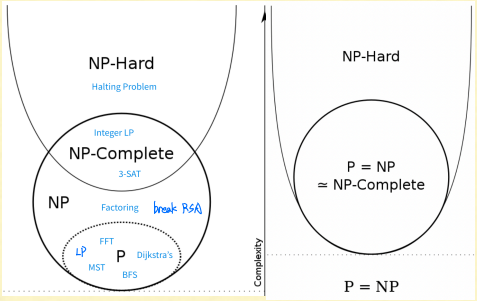
P: if this problem can be **solved** in polynomial time (DCL)
NP: if a solution of this problem can be **verified** in polynomial time
NP-Hard: if all problems in NP reduce to this problem
NP-Complete: if this problem is in NP and NP-Hard

P ⊆ NP. verify P: calculate sol in poly time. check they're the same
not P & not NP:

the halting problem: given input program, judge whether it has infinite loop

the counting problem: given a graph, tell the number of the 3-coloring

game winning strategy: can white win chess no matter how black move?



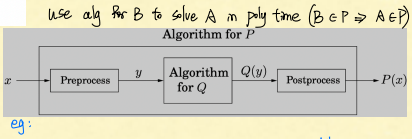
unweighted graph: Rudrata cycle → NP
weighted graph: min-TSP (min-cost) → not NP — optimal set
budget-TSP (cost ≤ B) → NP — budget set

NP-complete list
3SAT: Given a set of clauses, each containing between 1 and 3 literals, for example $(\bar{x} \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(\bar{x} \vee \bar{y})$, find a satisfying truth assignment.
Independent Set: Given a graph and a number g, find a set of g pairwise non-adjacent vertices. $|V| \geq g$
Vertex Cover: Given a graph, find a subset of vertice. touch every edge
min |S|
Integer Linear Programming: Given a system of linear inequalities, find a feasible integer solution ≥ B (& bound B)
Set Cover: Given a set of elements E and and several subsets of it $S_1, \ldots, S_m$ with budget b, select b of these subsets s.t. their union is E.

Rudrata/hamiltonian cycle
3-Coloring

reduction: A ≤p B

A reduce to B in poly time. A → B
(B is harder than A) B is at least as hard as A
use alg for B to solve A in poly time (B ∈ P ⇒ A ∈ P)



eg:
Rudrata cycle ≤p half cycle: find a cycle with $\frac{|V|}{2}$ vertice in unweighted G

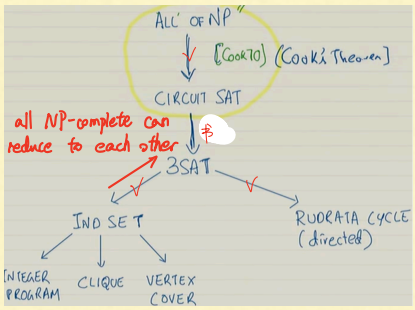
reduction algo: given graph G(V,E). for Rudrata cycle
add |V| vertices with no edge.
run half cycle

proof for A ≤p B:
① construct reduction algo F
② ∃ sol to A ⇒ ∃ sol to B = F(A)
③ ⇐

eg: independent set ≤p integer programming

① F: G = (V,E) ⟶ max $\sum x_i$
$0 \leq x_i \leq 1$
$\forall$ edge $(i,j) \in E$
$x_i + x_j \leq 1$
$(B = k)$

vertice $i \in$ ind set ⇒ $x_i = 1$

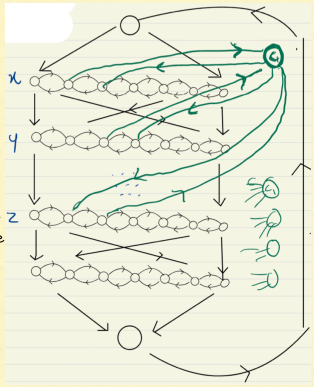


show NP-complete

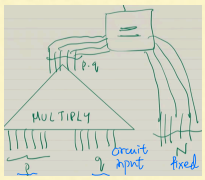
① prove A ∈ NP
② take one known NP-complete B.
show B ≤p A

eg: show ind set is NP-complete
① ind set ∈ NP
② 3SAT ≤p ind set
(1) reduction algo: $(x \wedge y \wedge z)$ ⟶

For all variable a. add edge between a & $\bar{a}$
(2) ∃ sol to 3SAT. ⇒ ∃ sol to ind set
given a sol to 3 SAT. for all $(x \wedge y \wedge z) = 1$
at least one of $x, y, z = 1$
add exactly one of x, y, z to ind set
(3) ∃ sol to ind set ⇒ ∃ sol to 3SAT.
for all variable a. $a = \begin{cases} 1 & \text{if } a \in \text{ind set} \\ 0 & \text{if } \bar{a} \in \text{ind set} \\ \text{arbitrary} & \text{otherwise} \end{cases}$



eg: show Rudrata cycle is NP-complete
① ∈ NP
② 3SAT ≤p Rudrata cycle
(1) reduction algo:
$x_i = 1$ left to right
$x = 0$ right to left
$C = (x \vee y \vee \bar{z})$



eg: all of NP (factoring) ≤p Circuit SAT
n bit number N — Circuit with and/or/not gate
Find $p, q > 1$. $p \cdot q = N$ — n input. assign to get output = 1
all NP has verify algo. can run on circuit.

eg: triclique NP-complete

1. **TriClique:** Given a graph $G = (V, E)$, a subset $S \subset V$ of vertices is called a **clique**, if every pair of vertices in S are connected by an edge, i.e., for every $u, v \in S$ we have $(u, v) \in E$.

Here is the **TriClique** problem.

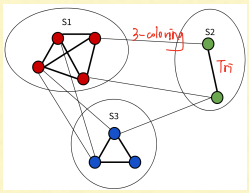
Input: A graph $G = (V, E)$

Solution: A partition of the vertices into three disjoint sets $S_1 \cup S_2 \cup S_3 = V$ such that, each of the sets S_1, S_2, S_3 is a *clique*.

Proof. It is clear that the TriClique problem is in NP. Now we will use a polynomial time reduction to show that the problem is indeed NP-complete. 3-coloring
Given an instance $G = (V, E)$ of $3 - Coloring$, define an instance $G' = (V', E')$ of TriClique as follows:
Set $V' = V$. For each pair (u, v) define,

$$(u, v) \in E' \leftrightarrow (u, v) \notin E$$

(In other words, edge (u, v) exists in E' if and only if it does not exist in E.



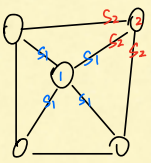
Set ordering

Input: A family of subsets $S_1, \ldots, S_n$ of $\{1, \ldots, m\}$.

Solution: An ordering of the sets so that consecutive sets in the ordering have intersection exactly one. Formally, an ordering of the sets is given by a permutation $\pi : [n] \to [n]$, so that for each $i \in \{1, \ldots, n-1\}$, $|S_{\pi(i)} \cap S_{\pi(i+1)}| = 1$.

Proof. It is clear that the Set-Ordering problem is in NP. Now we will use a polynomial time reduction to show that the problem is indeed NP-complete. Rudrata cycle
Given an instance $G = (V, E)$ of Rudrata Path, define an instance of Set Ordering as follows: Let the universe be the set of edges E and we will have each S_i be a subset of this univers. For each vertex $i \in V$, define

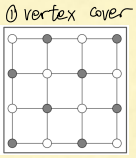
$$S_i = \{ \text{edges incident at vertex } i \}$$



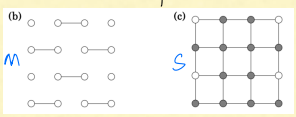
approximation algo

def. For optimal min problem P. α-approx: approx algo(I) ≤ α · OPT(I)
max ≥ OPT(I)/α

① vertex cover. maximal matching: keep adding edges to M (2-approx)
until add one more cause vertex overlap
output S: all vertex of the edges



V



proof: a) S is vertex cover (2) |S| ≤ 2|V|
Let S be a set that contains both endpoints of each edge in a maximal matching M. Then S must be a vertex cover—if it isn't, that is, if it doesn't touch some edge e ∈ E, then M could not possibly be maximal since we could still add e to it. But our cover S has 2|M| vertices....we know that any vertex cover must have size at least |M|.

|S| = 2|M|, |V| ≥ |M| ⇒ |S| ≤ 2|V|

② LP for vertex Cover
for each vertex i. min $\sum x_i$
(x_i can be fraction) $0 \leq x_i \leq 1$
$x_i + x_j \geq 1 \quad \forall (i,j) \in E$

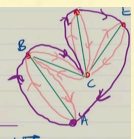
rounding algo: output $S = \{ i \mid x_i \geq \frac{1}{2} \}$
proof:
(1) S is vertex cover.
$x_i + x_j \geq 1 \quad \forall (i,j) \in E$
at least one of $x_i, x_j \geq \frac{1}{2}$
at least one of $i, j \in S$

(2) ① $|S| \leq 2$ (cost of LP = $\sum x_i$)
$\forall i \in S$: cost 1 — cost $x_i^* \geq \frac{1}{2}$
② cost of LP ≤ OPT
OPT is also sol to LP
but LP can use fraction

③ metric TSP. approx-2
all edge exist
$d(i,j) + d(j,k) \geq d(i,k)$

|S| ≤ 2|OPT)

Approx ALG:
STEP 1: Find a MST T
COST(T) ≤ COST(OPTIMAL TSP TOUR)



STEP 2: Find a Depth First Traversal of T
A → B → C → D → C → E → C → B → A
COST(DFS TRAVERSAL) = 2 · COST(Tree)

STEP 3: Skip ALL repeated vertices
A → B → C → D → C → E → C → B → A
COST(OUTPUT TOUR) ≤ COST(DFS TRAVERSAL)

3-tour TSP

Recall that a TSP tour is a cycle that visits all the vertices of a graph exactly once.

Analogously, a three-tour consists of three **disjoint** cycles C_1, C_2, C_3 such that each vertex appears in exactly one of the cycles exactly once. The cost of a 3-tour is the sum of the lengths of the edges on the three cycles. (Here a single vertex is considered a cycle of length 1, so one or more of the cycles C_1, C_2, C_3 can consist of a single vertex.)

Suppose you are given a complete graph on n vertices with distances $dist(\cdot,\cdot)$ between pairs of vertices. Assume that the distances satisfy the triangle inequality in that for each triple of vertices i, j, k,

$$dist(i,j) + dist(j,k) \geq dist(i,k)$$

Describe a 2-approximation algorithm for the minimum cost three-tour problem. (Give a succinct but clear description of your algorithm in at most four sentences)
Solution: Compute the Minimum Spanning Tree (MST) and delete the two heaviest edges to recover a forest with three trees T_1, T_2, T_3. The rest of the algorithm is analogous to the TSP approximation algorithm, but on three trees T_1, T_2, T_3. In words, perform a DFS on each of the three trees T_1, T_2, T_3 to obtain a traversal $\Gamma_1, \Gamma_2, \Gamma_3$. Shortcut the traversals to not have any vertex appearing more than once. This yields a Three-Tour solution.

3-approx. triangle removal

In the triangle removal problem, the input is a graph $G = (V, E)$. The goal is to remove the smallest number of edges from E, so as to delete all triangles in the graph.

Formally, here is the definition of triangle-freeness.

Definition: A graph is *triangle-free* if there are NO three vertices $i, j, k \in V$ such that (i, j), (j, k) and (k, i) are all edges.

Here is the formal definition of triangle removal problem:

Input: An unweighted undirected graph $G = (V, E)$

Solution: Find the smallest set of edges E' such that deleting E' makes the graph become *triangle-free*

We will now develop an approximation algorithm for the problem. First, write a linear programming relaxation for the triangle-removal problem.
Solution: x_e for each edge $e \in E$. The intended meaning is that x_e is the amount of edge e that is deleted.
Minimize $\sum_{e \in E} x_e$
$0 \leq x_e \leq 1$ for all $e \in E$. $x_a + x_b + x_c \geq 1$ for all triangles a, b, c.
Solution: If $x_e \geq 1/3$, round it to 1. Otherwise round it to 0.
E' is the set of edges e with $x_e = 1$.
Solution: For any triangle (a, b, c) in G, since $x_a + x_b + x_c \geq 1$, at least one of the edges $e \in \{a, b, c\}$ has $x_e \geq 1/3$, so we delete at least one of the edges in the triangle. So there cannot be any triangle in E'.
3, because x_e is scaled up by a factor of at most 3 during rounding.

metric TSP

1. If the cost of the minimum travelling salesman tour is C, then the cost of the minimum spanning tree is at most C
2. If there is a spanning tree T (not necessarily the minimum spanning tree) of cost W, then the cost of the minimum travelling salesman tour is at most 2W

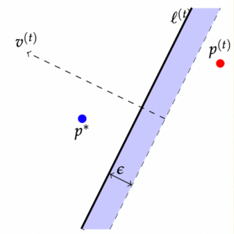
Gradient Descent

point pursuit game

Setup. A and B play a game, where there's some fixed point p^*. At each step of the game, **B chooses a point p**, then **A finds a line separating p and p* that is at least ϵ away from p. A loses when they can't find this line.**

Strategy. B chooses a point p based on the following rule:

$p_{t+1} = p_t + \epsilon\, v_t$ — move ϵ distance towards the separating line



Claim. In each iteration, the squared distance of B's point p to p* decreases by ϵ^2.

Theorem. If B plays by the strategy, then A fails to find a separating line within $\left\lceil \frac{\|p^{(0)}-p^*\|^2}{\epsilon^2} \right\rceil$ iterations.

high dim

Definitions. $x, y \in \mathbb{R}^d$

- **Inner Product.** $\langle x, y\rangle = \sum_{i=1}^{d} x_i \cdot y_i$.
- **Euclidean Length.** $\|x\|^2 = \sum_{i=1}^{d} x_i^2 = \langle x, x\rangle = \sqrt{x_1^2 + x_2^2 + \ldots + x_d^2}$
- **Hyperplane.** Set of points x that satisfies $\langle v, x\rangle - \theta = 0$ (normal vector; p^*: ≤ 0, $p^{(t)}$: $> \epsilon$)
  - v is a normal vector to the hyperplane

solve LP with exp constraint

max $3x + 4y$ → $3x + 4y \geq A$. normalize $\frac{3}{5}x + \frac{4}{5}y \geq B$. binary search B

Feasibility Problem. Find a point that satisfies a set of constraints. For an LP, these constraints are linear, i.e. take on the form $\langle a_m, x\rangle \geqslant b_m$

- Constraints can be normalized so $\|a_i\| = 1$

Algorithm 1: Solving LPs via Gradient Descent

$x^0 \leftarrow 1$
for $t = 0$ to T **do**
- (Find violated constraint) Find a constraint $\langle a_i, x\rangle \geqslant b_i$ violated by the $x^{(t)}$ with an error at least ϵ, i.e., $\langle a_i, x^{(t)}\rangle \leqslant b_i - \epsilon$ ($\langle w, \vec{x}\rangle$)

  If there is no violated constraint then **return** $x^{(t)}$.
- Set $x^{(t+1)} \leftarrow x^{(t)} + \eta \cdot a_i$ ($\eta \cdot w$)

end for
return *"No feasible solution within distance $\epsilon\sqrt{T}$ of the starting point $x^{(0)} = 0$"*

Applying Point Pursuit to LPs. Same setup. **A proposes a solution point** $x_{(t)}$, then **B looks for a violated constraint** $\langle a_i, x^{(t)}\rangle \leqslant b_i - \epsilon$.

- Constraint can be returned as a hyperplane separating x and x*: $\ell(x) = \langle a_i, x\rangle - b_i$

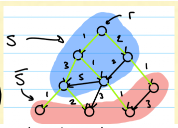
Arbrosence (min-cost)

find directed tree. away from r. reach all vertice.

$x_e = \begin{cases} 1 & e \in \text{Arb} \\ 0 & \text{otherwise} \end{cases}$

$\sum_e w_e x_e \leq B$

for all $S \cup \bar{S} = V$. $r \in S$: $\sum_{e \text{ leaving } S \text{ to } \bar{S}} x_e \geq 1$ $(2^{n-1} - 1)$



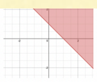
So, if we calculate the min cut between r and every other vertex, and the min cut is < 1 => this constraint is violated for the min cut!

Convex Optimization (LP with infinite constraint)

Halfspace. Set of points on one side of a hyperplane.

$$H = \left\{x \in \mathbb{R}^d \middle| \langle w, x\rangle - \theta \geqslant 0\right\}$$

Example of a halfspace defined by $x_1 + x_2 - 1 \geq 0$.

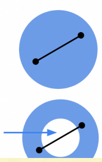


Convex Set. An intersection of a (possibly infinite) family of halfspaces.

- Equivalently, a set S is convex if for any two points x, y ∈ S, the line segment joining x, y is also contained in S.

Convex. For any two points, all points on their connecting line segment are within the set.

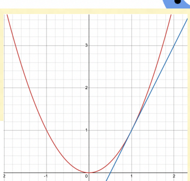
Not convex. There are points on a joining line segment that are not in the set.



S is convex set in $\mathbb{R}^d$. p is pt not in S
⇒ ∃ half space H s.t. $P \notin H$. $S \subseteq H$

Convex Function. A convex function f obeys:

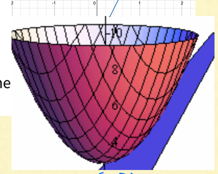
f((x1 + x2)/2) <= (f(x1) + f(x2))/2 (for any x, y)



Convex Function. A convex function f obeys:

$$f(x^*) \geq f(x) + \frac{df}{dx}(x) \cdot (x^* - x),$$

In words: Pick any points on the line, and draw a tangent. The tangent should always stay below the function (or touch it).



$f(x^*) \geq f(x) + \nabla f(x) \cdot (x^* - x)$

$f(x) = x^2 + y^2$. $\nabla f = (2x, 2y)$. $p = (5, 7)$ $(\nabla f)_p = (10, 14)$

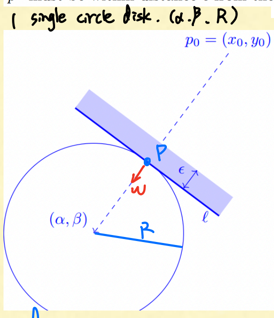
$x^2 + y^2 \geq 10$ 圆3外. not convex

Input: n circular disks on the two-dimensional plane specified by $\{(x_i, y_i, R_i) | i = 1, \ldots, n\}$ where (x_i, y_i) is the center and R_i is the radius of the i^{th} disk.

Output: A point $p^* = (x^*, y^*)$ that is inside all the disks (approximately). Formally, the point p^* must be within distance ϵ from the interior of every disk.

1 single circle disk. (α, β, R)

$\sqrt{(\alpha - x_1)^2 + (\beta - x_2)^2}$



$P = (\alpha, \beta) - R \cdot w$

line: $w_1 x_1 + w_2 x_2 - (\langle w, p\rangle) = 0$

$x^{t+1} = x^t + \eta \cdot w$

separating hyperplane oracle: 代入 $\begin{bmatrix} P_0 \\ (\alpha, \beta) \end{bmatrix}$ <0, >0

n disk → start at center of smallest circle (ridus = r)
$D \leq r$
$O(n \cdot r^2/\epsilon^2)$

Zero-sum game

1. If the column player is restricted to the three pure strategies $\{(1,0,0), (0,1,0), (0,0,1)\}$, row player's expected payoff doesn't change.

   ○ True ○ False

   True

2. If the row player is restricted to the three pure strategies $\{(1,0,0), (0,1,0), (0,0,1)\}$, column player's payoff doesn't change.

   ○ True ○ False

   False

3. Suppose we modify the game to include an additional row (so the resulting matrix is a 4×3 payoff matrix). Then the value of the game can possibly both increase or decrease depending on the entries of the new row.

   ○ True ○ False

   False. The row player has more moves in the new game, but the column player has the same set of moves. Therefore, the value of the row player cannot decrease.

Quick select

1 3 4 4 5 6 8 9 12. find median

slowest. 1 3 4 12 9 8 6 5

Graph SCC at least n edge (only a cycle)

3. An n-vertex directed acyclic graph can have as many as $\binom{n}{2}$ directed edges, while an n-vertex undirected graph with no cycles can have as many as $n - 1$ edges.

DP. edit distance

The dynamic programming based algorithm for edit distance between two strings $x[1, \ldots, m]$ and $y[1, \ldots, n]$ takes $O(mn)$ time, because the algorithm uses $O(mn)$ many subproblems computing each of which takes $O(1)$ time.

FFT

Let $\omega = e^{2\pi i/16}$ be the 16^{th} root of unity. Suppose the Fourier transform of a vector $(p_0, p_1, \ldots, p_{15})$ is $(1, \omega, \omega^2, \ldots, \omega^{15})$.

What is the value of $\sum_{i=0}^{15} p_i \cdot 2^i$? 2

$P(2)$ $P(1) = 1$ $P(\omega) = \omega$. $P(\omega^2) = \omega^2$ ... ⇒ $P(x) = x$

non-MST edge

Devise a $O(|V|)$-time algorithm for the following problem:

Input: Graph $G = (V, E)$ with distinct edge weights w_{ij} for each edge $(i, j) \in E$.

Solution: Find some edge $(i, j) \in E$ such that (i, j) is NOT part of any MST.

1. The idea is to find the heaviest edge in some cycle of the graph.

   Run DFS on the graph until we encounter a back-edge $u \to v$, discovering a cycle. We will find the heaviest edge on this cycle and return it. To find the heaviest edge, check all edges along the path from v to u in the tree. This can be done efficiently by popping the call stack for Explore to retrace the path up the tree.

2. Each edge encountered during DFS in an undirected graph is either a tree edge or a back-edge. Since there are only $|V| - 1$ tree edges, DFS will encounter a back-edge in $O(|V|)$ time.

Graph 变形

Consider a directed graph $G = (V, E)$ with positive weights $c(i, j)$ on its edges.

Given a path $P = v_0 \to v_1 \ldots \to v_k$, the *discounted cost* of the path is the sum of lengths of all but the the longest edge. Formally,

$$cost(P) = \left(\sum_{i=0}^{k-1} c(v_i, v_{i+1})\right) - \max_i c(v_i, v_{i+1})$$

(It is the sum of cost of all the edges - maximum cost edge)

Devise an algorithm to compute the smallest discounted cost path from a vertex s to vertex t. Give a succinct and precise description of your algorithm. Proof of correctness and run-time analysis are not necessary. The runtime of your algorithm should be $O((|V| + |E|) \log |V|)$.

Solution: Minimizing the discounted cost of a path is the same as minimizing the cost of a path wherein one edge can be skipped at zero cost.

Since we are looking for paths with exactly one "skipped edge", we can create a layered graph G' as follows:

For each vertex $v \in V$, there would be two copies v_0, v_1. For each edge $u \to v$, add edges $u_0 \to v_0$, $u_0 \to v_1$ and $u_1 \to v_1$.

Weights of the edges are

$$c(u_0 \to v_0) = c(u_1 \to v_1) = c(u \to v)$$

and

$$c(u_0, v_1) = 0$$

Essentially G' consists of two separate copies of G, with zero-weight "skip" edges from one copy to other. We construct the graph G' and use Djkistra's algorithm to find the shortest path from s_0 to t_1.

DP. all pair shortest path (worse runtime)

Given an input graph $G = (V, E)$ with positive edge weights w_{ij} for each edge (i, j). Consider the following sub-problem:

$D[i, j, t]$ = *length of the shortest path from i to j with at most 2^t hops*

(Here *"hop"* refers to the number of edges on the path).

$$D[i, j, t] = \min_{k \in V} \{D[i, k, t-1] + D[k, j, t-1]\}$$

For $i, j \in V$

$D[i, j] = \begin{cases} w_{ij} & \text{if } i \to j \in E \\ \infty & \text{otherwise} \end{cases}$

For $t = 1$ to $\lceil \log n \rceil$
For i in V
For j in V
For k in V

$$D[i, j, t] = \min_{k \in V} \{D[i, k, t-1] + D[k, j, t-1]\}$$

return $D[i, j, \lceil \log n \rceil]$ for all i, j.

What is the runtime of your algorithm? **Solution:** $\Theta(n^3 \log n)$ where $n = |V|$

Devise an algorithm to compute a spanning tree of minimum total length such that if two cities are connected by a bikeable path in the graph, then they have a bikeable path in the tree.

Here is the formal description of the problem.

Formal description:

Input:

1. An undirected graph $G = (V, E)$ where for each vertex $v \in V$, we are given its altitude $h(v)$.
2. Length $c(u, v)$ for each edge $(u, v) \in E$.

Assume that all edge lengths $\{c(u, v)\}$ and altitudes $h(v)$ are positive.

Definition (bikeable path): *For a pair of vertices u,v, a path P from u to v is said to be a bikeable path if the path P never passes through a vertex of height greater than maximum of $h(u)$ and $h(v)$.*

Output: A spanning tree T of minimum total length that satisfies the following:
If any pair of cities u, v are connected via some bikeable path in the graph G, then they are connected by a bikeable path in the tree T.

Solution: Suppose T is the optimal tree that we seek. Consider the vertex v with the highest altitude. Suppose we delete the vertex v from the tree T, we get a set of disconnected trees $T_1, \ldots, T_k$. Let V_i denote the set of vertices in the tree T_i.

Observe the following:

- T_i is the optimal tree for the set of vertices V_i.
- There are no bikeable paths from V_i to V_j for $i \neq j$ in the graph G (since there are no paths in tree T).
- Suppose w_i is the vertex of highest altitude in the tree T_i, then every pair of vertices in V_i are connected by paths of altitude $< h(w_i)$. So V_i is nothing but a connected component of the graph G, once we delete vertices of altitude $> h(w_i)$.

Therefore, we can now write down the subproblems.

1. For a given altitude h, let $G_{\leq h}$ denote the subgraph consisting of vertices of altitude $\leq h$. The subproblems are as follows:
   For each vertex v, we will have a subproblem for each connected component of the graph $G_{\leq h(v)}$. Formally, for each vertex v, and a connected component C of $G_{\leq h(v)}$ define:
   $$T[C] = \text{ cost of the optimal tree for the connected component } C \text{ of the graph } G_{\leq h(v)}$$
2. Consider a connected component C of $G_{\leq h(v)}$.
   Let W_C be the set of vertices of highest altitude within C (there may be many vertices of the same highest altitude in C.) Let $C_1, \ldots, C_k$ denote the connected components of $C \setminus W_C$. Then,
   $$T[C] = \sum_{i=1}^{k} T[C_i] + \text{ cost of minimum spanning tree connecting } W_C \text{ with each other and } T[C_1], \ldots, T[C_k]$$

In other words, we compute $T[C]$ by the following version of Kruskal's algorithm:

(a) Start with trees $T[C_1], \ldots T[C_k]$.

(b) For each edge e within C in increasing order of length: add e to the tree unless it creates a cycle.

GD

Fair Allocation

There are N dollars to be allocated among n employees. Based on work experience, employee i is entitled at least ℓ_i dollars.

An allocation is defined to be fair if no subset of $n/10$ employees or fewer receives more than half the dollars.

Find a fair allocation if possible; otherwise, report that none exists.
Write a linear program for the above problem (with possibly exponentially many constraints).

Solution: We can define a linear program with the following constraints. Note that since we are solving a feasibility problem, no objective function is needed.

Let x_i be the amount of money that employee i is allocated.

$$x_i \geq \ell_i \qquad \forall i$$
$$\sum_{x_i \in S} x_i \leq \frac{N}{2} \qquad \forall \text{ subsets } S \text{ of most } n/10 \text{ employees}$$

The first set of constraints guarantee that each employee gets at least l_i dollars, and the second constraint guarantees that the allocation is fair.
Describe a polynomial-time algorithm implementing the separation oracle[1] for the linear program from part (a).

Solution: Given some solution x, we can verify the first set of constraints in $O(n)$ time by just checking each individual constraint.

To determine if any of the fairness constraints are broken, simply check whether the $n/10$ highest paid employees are allocated to more than $\frac{N}{2}$ dollars. If they are, then we have found one such broken constraint. Otherwise, no fairness constraint is broken. This can be done in $O(n \log n)$ by sorting the employees in allocations and taking the $n/10$ highest paid. It can also be done in $O(n)$ by find the $n/10$-th highest paid employee using quickselect then iterating the employees to find the $n/10$ highest paid.

Setting Tolls

The city council has decided to impose tolls on some its most popular roads to discourage people from driving, aiming to reduce traffic and emissions. In particular, the goal is to discourage all driving from location s to location t in the city. The formal description of the problem is as follows.

Input: An undirected unweighted graph $G = (V, E)$, a pair of nodes s and t, and a positive real number B.

Output: A set of edge weights $w : E \to \mathbb{R}^+$ such that:

1. For every $s - t$ path P in the graph G, the total weight on the path is at least 1
2. The total weight of all edges in the graph is at most B

(weights w_e are the "tolls" on the edges)

Describe a polynomial-time algorithm implementing the separation oracle for the linear program.

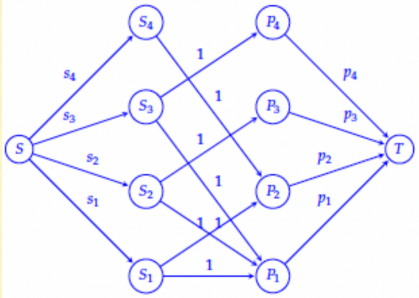
Solution: The first set of constraints can be verified in $O(E \log V)$ time. Given a set of edge weights, w, run Dijkstra's algorithm from s to t. If the total weight of this path is less than 1, then we found a violated constraint. Otherwise, none of the constraints in the first set are violated.

The second constraint can be verified in $O(E)$ time by just summing the edge weights.

(4 points) A compatibility graph between 4 students and 4 pets is shown below. Here are the constraints on the meetings:

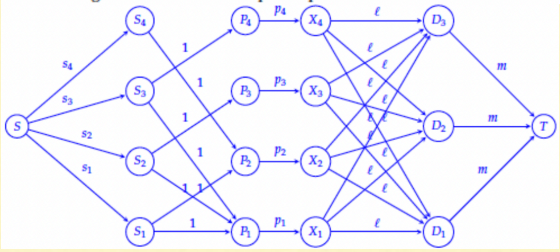
- The i^{th} student S_i has indicated a preference of meeting exactly s_i pets.
- The j^{th} pet P_j can meet at most p_j students.
- All meetings must be between compatible pairs.

Solution:

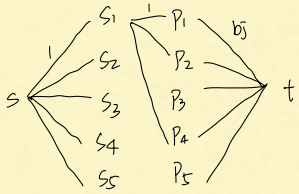


This works very similarly to matching, although it constrains each of the students to be matched with at most s_i pets, and similarly with the pets. There exists a matching that satisfies the constraints iff all the edges leaving s are saturated by the max flow (so each student meets with exactly the number of pets desired).

- The meetings need to be scheduled over 3 days.
- The animal shelter cannot host more than m meetings in total on the same day.
- No pets wants to meet more than ℓ students on the same day, but the students don't mind meeting any number of pets on the same day.
- The i^{th} student S_i has indicated a preference of meeting exactly s_i pets.
- The j^{th} pet P_j can meet at most p_j students.
- All meetings must be between compatible pairs.



- Each student s_i has a list of preferred professors $L_i \subseteq \{p_1, \ldots, p_m\}$. They want to meet with one professor from this list.
- Each professor p_j has an integer bound $b_j \in \{1, 2, 3, \ldots, \}$. They want at most b_j students in their advising session.

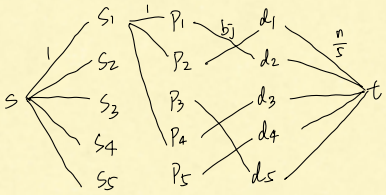


(i) Describe the nodes of the graph G.
Solution: There should be one node per student s_i, one node per professor p_j, one source node s, and one sink node t.

(ii) Describe the edges of the graph G (draw a picture if needed).
Solution: Each for each student s_i and the professors $p_j \in L_i$, add an edge (s_i, p_j) with capacity 1. Add an edge from the source s to each student s_i with capacity 1. Add an edge from each professor p_j to the sink t with capacity b_j.

5 department

This means that the number of students advised by professors in each department d_i should be exactly equal to $n/5$ (assume that n is divisible by 5).



Write a linear program for the above problem (with possibly exponentially many constraints).

Solution: We can define a linear program with the following constraints. Note that since we are solving a feasibility problem, no objective function is needed.

$$\sum_{e \in p} w_e \geq 1 \qquad \forall p \text{ that is an } s - t \text{ path}$$
$$\sum_{e \in E} w_e \leq B$$

The first set of constraints guarantee that the total weight of all paths are at least 1. The second constraint guarantees that the total weight of all edges is at most B.