COSC-311 Final Notes

Algo Examples

- Divide + conquer : Mergesort
- Greedy: Interval scheduling
- Dynamic: weighted interval scheduling

Runtimes (Big)

- $f(n) = O(n^2) \rightarrow f(n) \leq cn^2$ normal Big O
  - $= \Theta(n^2) \rightarrow cn^2 \leq f(n) \leq cn^2$
  - $= \Omega(n^2) \rightarrow f(n) \geq cn^2$ larger than
  - $= o(n^2) \rightarrow f(n) < cn^2$ strictly less than

Master Theorem

$T(n) = aT(\frac{n}{b}) + f(n)$ for $n > 0$, constant a and b, asymptotically positive function $f(n)$.
$\hookrightarrow cn^d$

Simplified

Case 1: $d < \log_b a$, then $O(n^{\lg_b a})$

2: $d = \log_b a \rightarrow O(n^d \lg n)$

3: $d > \log_b a \rightarrow O(n^d)$

Cormen

1: $f(n) = O(n^{\log_b a - \varepsilon})$ for $\varepsilon > 0 \rightarrow \Theta(n^{\log_b a})$

2: $f(n) = \Theta(n^{\log_b a}) = T(n) \rightarrow \Theta(n^{\log_b a} \lg n)$

3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$ and $af(\frac{n}{b}) \leq cf(n)$ for $c < 1 \rightarrow O(f(n))$.

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

$\log_b a = \log_2 4 = 2$ $\rightarrow$ $d < \log_b a$, so Case 1: $T(n) = O(n^2)$

$d = 1$

Ex. $T(n) = 3T(\frac{n}{4}) + n \lg n$

$\log_b a = \log_4 3 \leq 1$

$\rightarrow n\lg n \nleq n^{\log_b a - \varepsilon}$ NOT Case 1

$\rightarrow n\lg n \neq n^{\log_b a}$ NOT Case 2

$\rightarrow n\lg n \geq n^{\log_b a + \varepsilon}$ Yes, so check regularity condition : $3(\frac{n}{4}) \lg(\frac{n}{4}) \leq cn\lg n$

$\frac{3n}{4} \lg(n) - \lg(4) \leq cn \lg n$

$\frac{3}{4} \lg(n) - \lg(4) \leq c \lg n$ $c \leq \frac{3}{4}$

Yes, so $\Theta(n\lg n)$

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

$\log_3 3 = 1$ $\rightarrow$ $d = \log_b a$, so case 2: $O(n \lg n)$

$d = 1$

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

$\log_2 2 = 1$

$\rightarrow \frac{n}{\lg n} \leq n^{1-\varepsilon} \rightarrow \frac{1}{\lg n} \leq n^{-\varepsilon} \rightarrow \frac{1}{\lg n} \leq \cancel{\frac{1}{n^\varepsilon}}^{0}$ This cannot be true, because $\lg n \geq 0$. So, no solution

Ex. $T(n) = 6T(\frac{n}{3}) + n^2 \lg n$

$\log_3 6 < 2$ placeholder number

$\rightarrow n^2 \lg n \notin c n^{1.9-\varepsilon}$, so Case 1, 2 don't apply

$\rightarrow n^2 \lg n \geq cn^{1.9+\varepsilon}$

$\lg n \geq cn^{-0.1+\varepsilon}$

$\lg(\lg n) \geq (-0.1+\varepsilon)c \cdot \lg(n)$ let $\varepsilon < 0.1$

Regularity condition: $6\left[(\frac{n}{3})^2 \lg(\frac{n}{3})\right] \leq cn^2 \lg n$

$6\left[\frac{n^2}{9} \lg(\frac{n}{3})\right] \leq cn^2 \lg n$

$\frac{2n^2}{3} \lg(\frac{n}{3}) \leq cn^2 \lg n$ let $c > \frac{2}{3}$. Then, $T(n) = O(n^2 \lg n)$

Ex. Solve via substitution: $T(n) = 2T(\frac{n}{2}) + 1$, solution $T(n) \leq cn - d$, to find

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

$\leq 2(\frac{cn}{2} - d) + 1$

$\leq cn - d + 1$

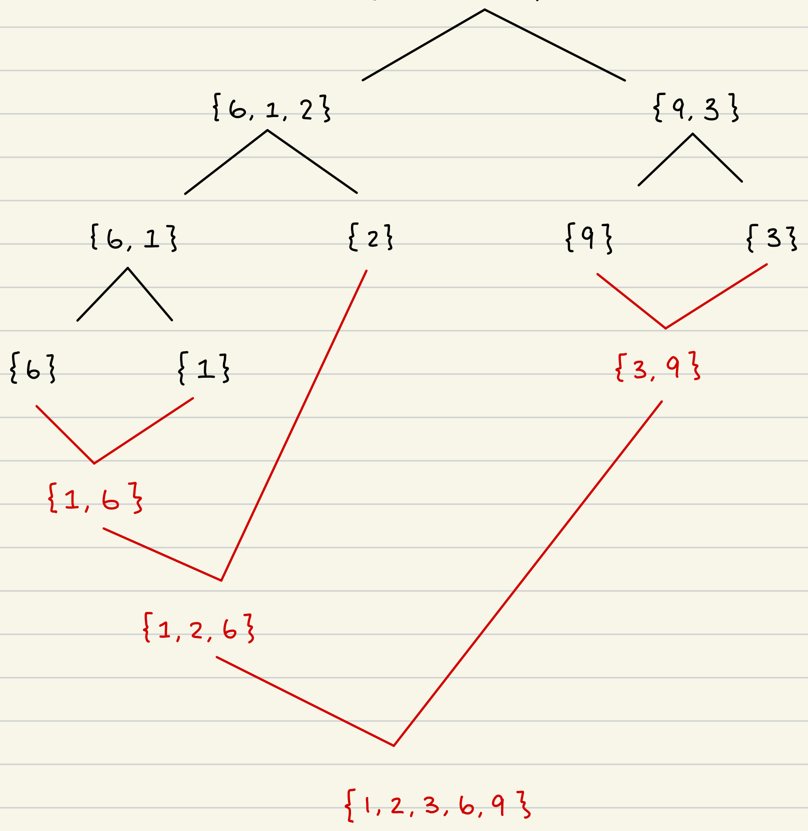
$d > 1$

$T(n) \leq cn - d + 1 - d \leq cn - d \rightarrow$ Thus, $T(n) \leq cn - d$. Thus, $T(n) = O(n)$.

Divide + conquer

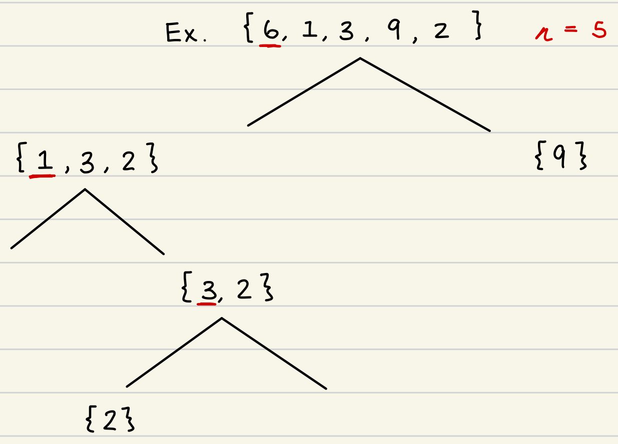
- Merge Sort
  - Divide array into halves, sort when you merge back up.

Ex. $\{6, 1, 2, 9, 3\}$ $n = 5$



Runtime: n cmps, $\lg(n)$ times, so $O(n \lg n)$

- Quicksort

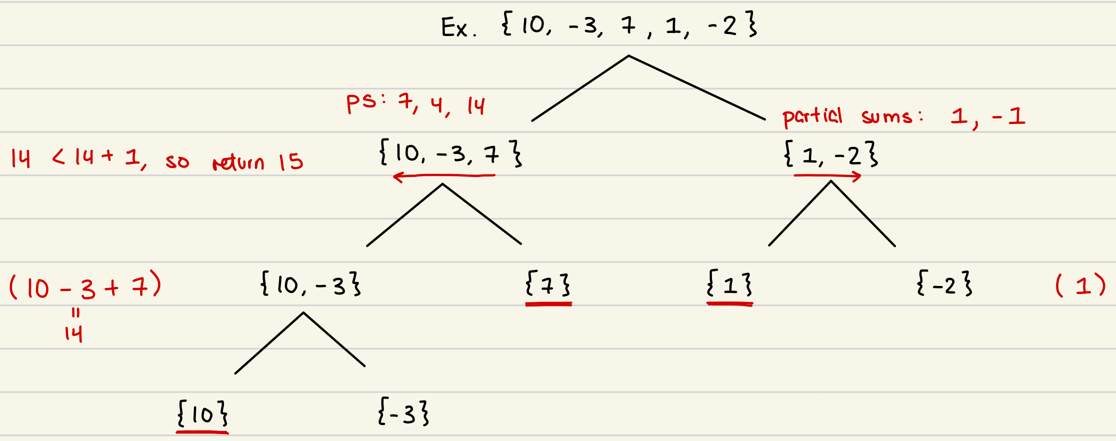


merge from left to right : {1, 2, 3, 6, 9}

Runtime : best case → $O(n \lg n)$, like merge sort

worst case → $O(n^2)$, if you pick bad partition elements to make (n-1) and 1 "halves"

- Longest Sum Subsequence
  - max (MSS(left half), MSS (right half), MSS (span of right + left halves))

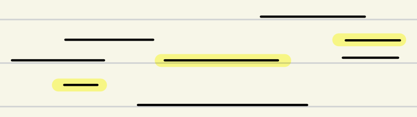


Runtime : $O(n \lg n) \rightarrow T(n) = 2T(\frac{n}{2}) + O(n)$

GREEDY

- Not guaranteed optimal solution
- Pick best-seeming choice

- Interval Scheduling
  - Choose interval that ends earliest, remove all overlapping intervals. Repeat



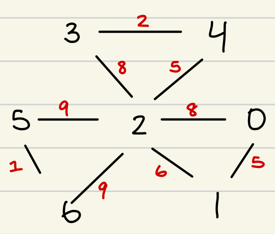
implementation : min Heap by ending times

Runtime ?

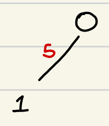
- With these ↓ , make sure you don't create cycles!
- Notice : 6 nodes, 6 steps

- Dijkstra's
  - Find shortest path from root. Builds MST from root by adding vertex w/ shortest path from root

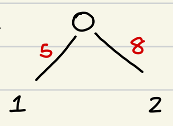
Ex.



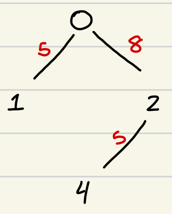
S1 :



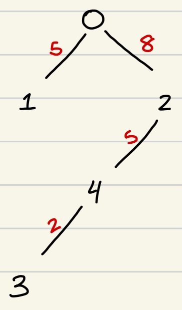
S2 :



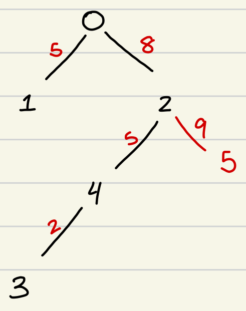
S3 :



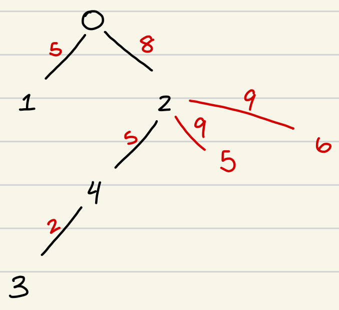
S4 :



S5 :

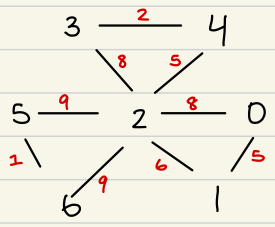


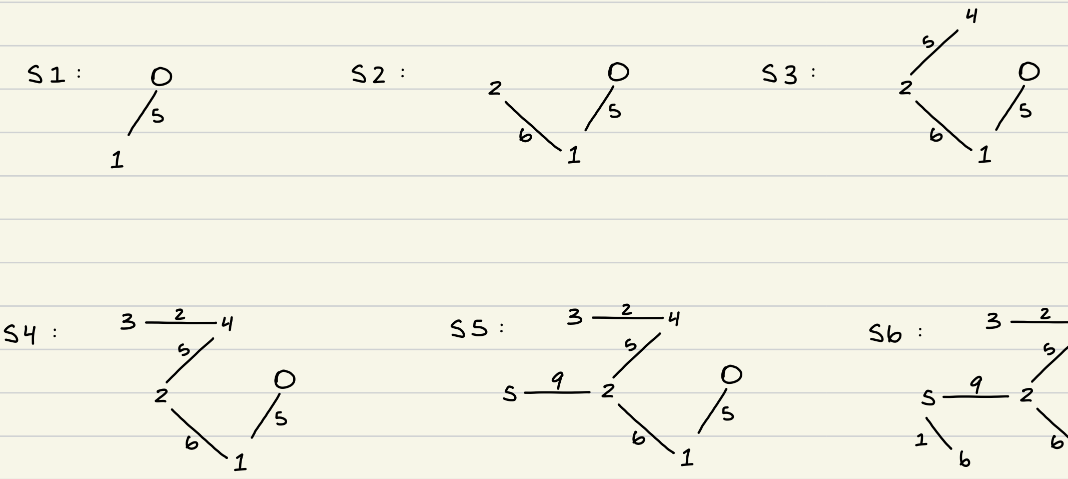
S6 :



- Prim's (like Dijkstra's) → no unique solution
    - Start from one vertex, adding edges w/ lowest connection to existing tree

Ex.



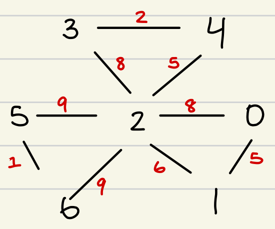


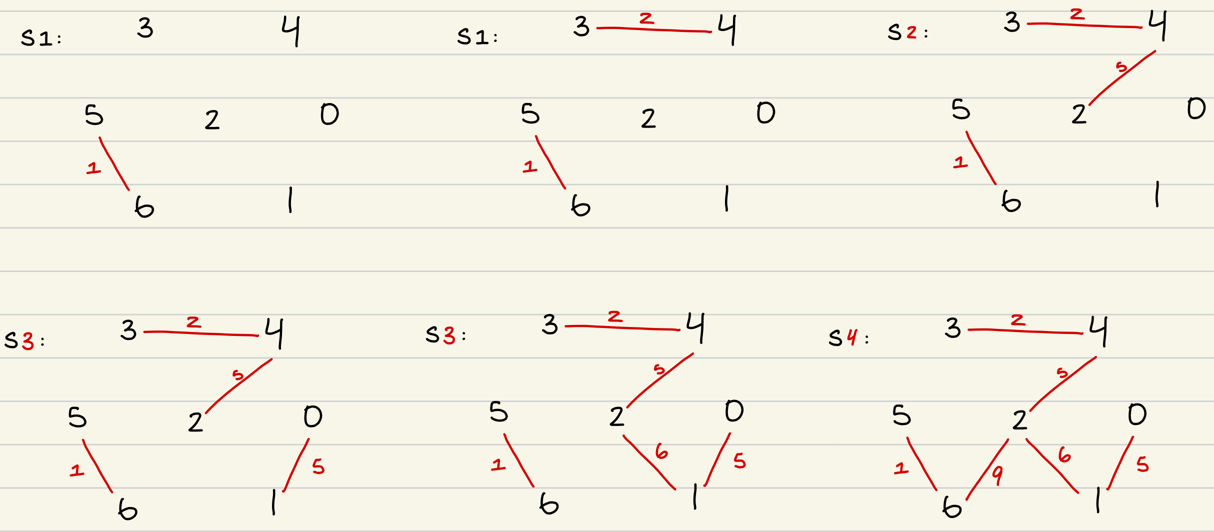
- Runtime: O(V lgV + E lg V) → adjacency list

  O(V^2 + E) → array (deleting min connection + updating edges)

- Kruskal's
    - Start w/ your vertices (remove edges). Add smallest edge

Ex.





Runtime: O(E lg E + E v)

↳ checking we don't have a cycle

↓ removing from heap E times

DYNAMIC

STEPS : 1) Find base case(s).

2) Make recursion. → make sure you have overlapping subproblems

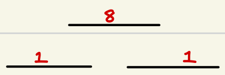
3) Describe size and shape of table → normally, n arguments In recursion ⟹ $n \times n$ table

4) Find out how to fill in table → start from $i = 0$. Where do you go now, so subsequent fillings only have to reference the array?

5. Find runtime → (length of input)(time to fill In 1 element)

Recursions often either consider or Don'T consider the n^{th} element.

- Weighted Interval Scheduling
    - diff from interval scheduling, because you can't just pick the earliest ending interval. You have to calculate the weight of the diff combos of intervals.
    - OPT (n intervals) = OPT ($0 \to n-1$) + ^{n}weight   maximize over n, only consider nonconflicting intervals
      OPT($0 \to n-1$)

Ex. 

→ first sort by finish time (heap)

Array : [0, 0, 0, 0] → (3 intervals + base case of 0 intervals)

- $n = 1$: [0, 1, 0, 0}
- $n = 2$: [0, 1, 8, 0}

max [A[0] + 8 , A[0], A[1]]

- $n = 3$: { 0, 1, 8, 8 }

max [A[0] + 1, A[1] + 1, A[0], A[1], A[2]]

- Runtime : $O(n \cdot 1)$

- OPT Binary Search Tree
    - Find lowest cost MST
    - $OPT(i, j \text{ nodes}) = OPT(i, r-1) + OPT(r+1, j) + [Prob(i) + \dots + Prob(j)]$ minimize over $i \le r \le j$
        - Base case : $OPT(i, i) = Prob(i)$
          $OPT(i, i-1) = 0$
    - Ex. $k_1 < k_2 < k_3 < k_4 < k_5$
      0.25 0.2 0.1 0.15 0.3

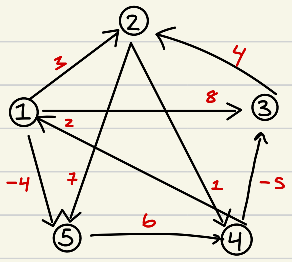
★ fill in diagonally, upper half

↓i j→	0	1	2	3	4	5
0	0					
1	0	.25	★ 0.65			
2		0	.2			
3			0	.1		
4				0	.15	
5					0	.3

★ OPT(1, 2) : r = 1 | A[1, 0] + A[2, 2] + 0.45 = 0.65
r = 2 | A[1, 1] + A[3, 2] + 0.45 = 0.7

Runtime : $O(n^2 \cdot n) = O(n^3)$

- Floyd-Warshall
  - Find shortest dist btwn every pair of vertices in a weighted, directed graph
  - $OPT(i, j, K) = OPT(i, j, K-1)$ OR
    $OPT(i, K, K-1) + OPT(K, j, K-1)$



↱ go through vertex 1

Step 0	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

Step 1	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$OPT(4, 2, 1) = \overset{\infty}{OPT(4, 2, 0)}$ or
$\underset{2}{OPT(4, 1, 0)} + \underset{3}{OPT(1, 2, 0)} = 5$

$OPT(4, 5, 1) = \overset{\infty}{OPT(4, 5, 0)}$ OR
$\underset{2}{OPT(4, 1, 0)} + \underset{-4}{OPT(1, 5, 0)} = -2$

STRATEGY

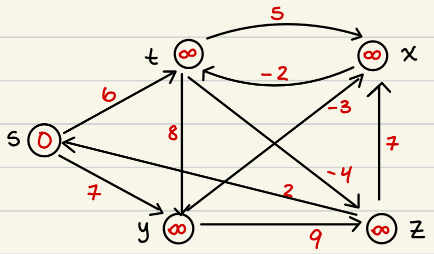
- Create a $n \times n$ table for Step 0. Set the values; $A[i, i] = 0$, $A[i, j] = \propto$ if there is an edge connecting i and j, or ∞ if ∄!

- Create another table for Step 1. Use the recursive formula to fill in. Since $K-1 = 0$, index into the first table, using the 2 other arguments in OPT().

- Repeat until you reach step 6 (going through $6 - 1 = 5$th vertex).

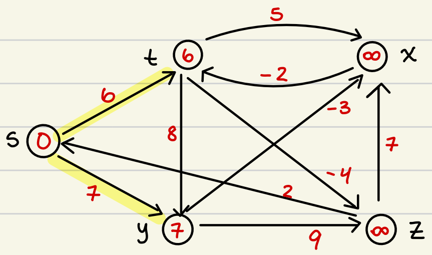
- Bellman – Ford
    - Find shortest path from a single vertex to all other vertices → works w/ negative edges, unlike Dijkstra
    - OPT(n edges) = OPT(n-1) OR
      OPT(n-1) followed by an edge

Ex.

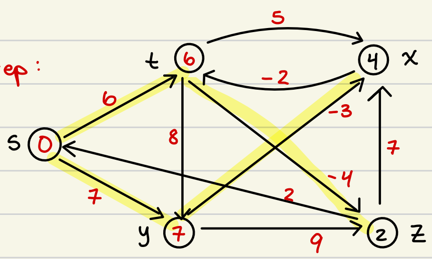


	S	T	X	Y	Z
0 steps	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	-2

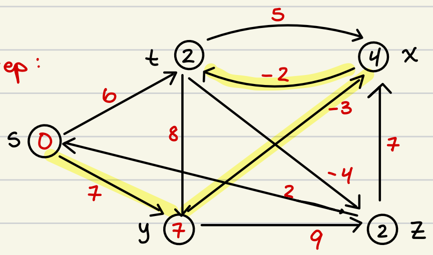
1 Step :



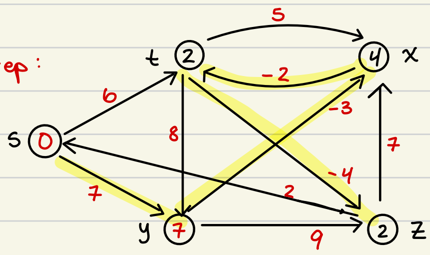
2 Step :



3 Step :



4 Step :



STRATEGY

- Create a 2x2 array, with the # of steps going down and the names of the vertices going across.
- Initialize 1st column to 0.
- Fill in the array row by row.

- Knapsack
  - Given weight capacity and target value, try to reach target.
  - $OPT(n \text{ items}, w' \text{ cap}) = \left.\begin{matrix} OPT(n-1, w' - n_w) + n_{\text{weight}} \\ OPT(n-1, w') \end{matrix}\right]$ OR ☆ either add the nth item or not

Ex. Capacity : 5 Target : 10

Items' Weight :	2	3	1	1	6
Value :	8	5	3	3	10

- A =

n↓ w→	0	1	2	3	4	5
0						$\rightarrow$ 0
1		0	8	8	8	8
2		0	8	8	8	13
3		3	8	11	11	13
4		3	8	11	14	14
5	$\downarrow$ 0	3	8	11	14	14☆

☆ fill in one row at a time

$\rightarrow$ max(A[4, 5-6) + 10, A[4, 5])

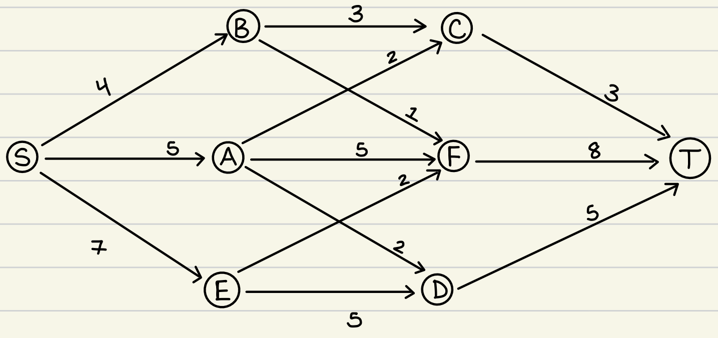
Runtime : O(nW), not polynomial

- BC: no items, no capacity

Max Flow

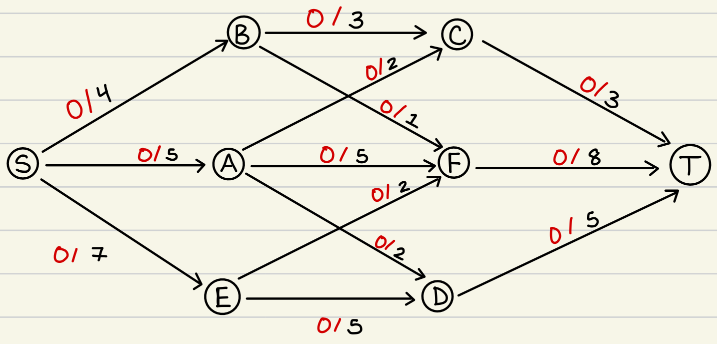
- Ford-Fulkerson
    - Find the max flow from source to sink (weighted, directed graph)
    - S1 : Initialize flow graph (all flows set to 0) and residual graph (set to given graph).
    - Pick a path. Increase flow through the vertices by the smallest capacity. In the residual graph, update the cap. and create backward edges w/ the current flow.
        - You can follow forward and backward edges. When you follow backward edges, you are correcting a previous, inefficient path by pushing those units BACK. On the flow graph, you decrease the flow. On the residual graph, the "backward edge" resembles a forward edge.

<u>Given graph</u> :

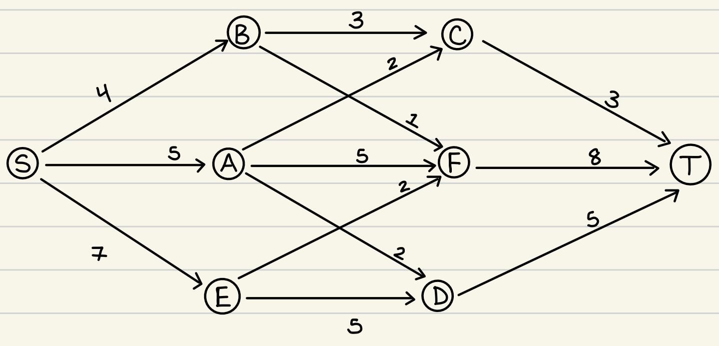


<u>Step 0 : Initialization</u>

- Flow graph (initial flow is 0) :



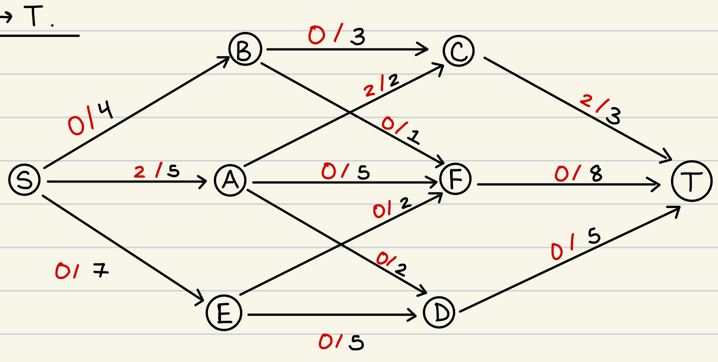
- Residual graph (set to given graph) :



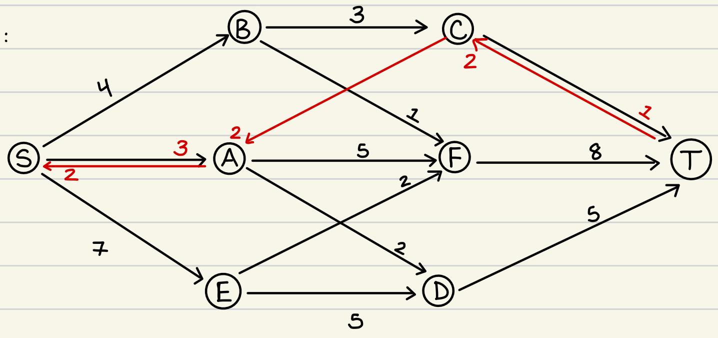
→

Step 1 : Consider the path S → A → C → T.

- Flow graph (min. capacity = 2):

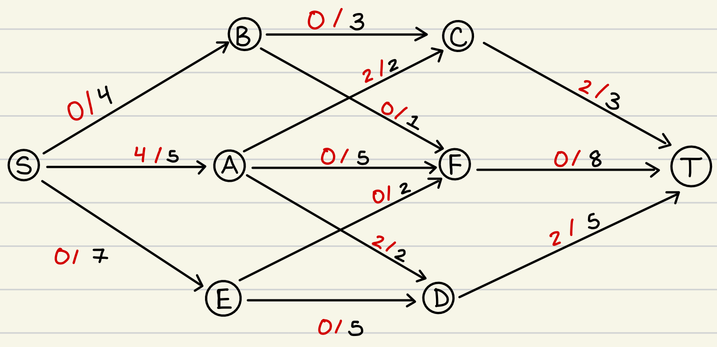


- Residual graph (cap. -2, add backward edges):

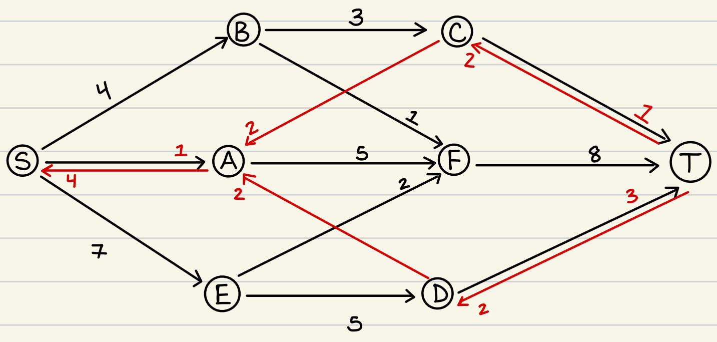


Step 2 : Consider the path S → A → D → T.

- Flow graph (min cap : 2):

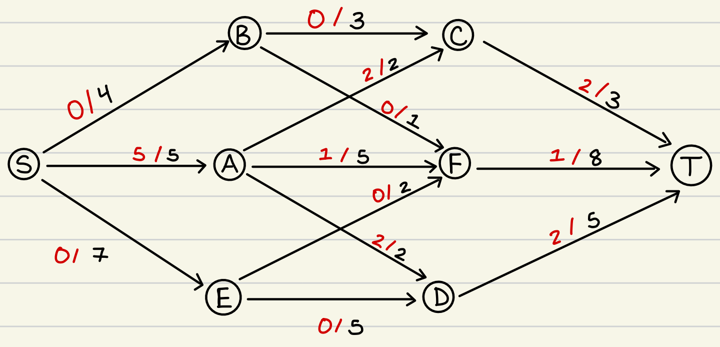


- Residual graph (capacity -2):

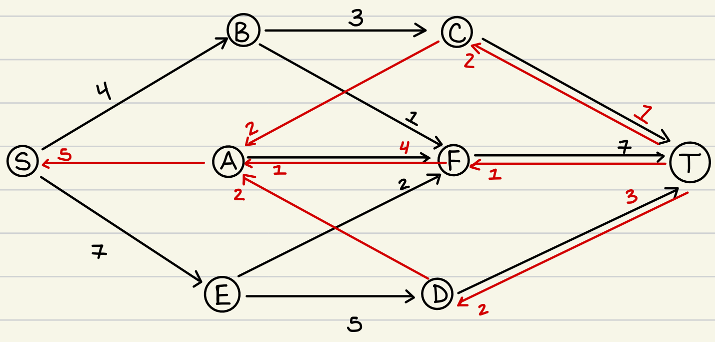


Step 3: Consider the path S → A → F → T.

- Flow graph (min cap : 1):

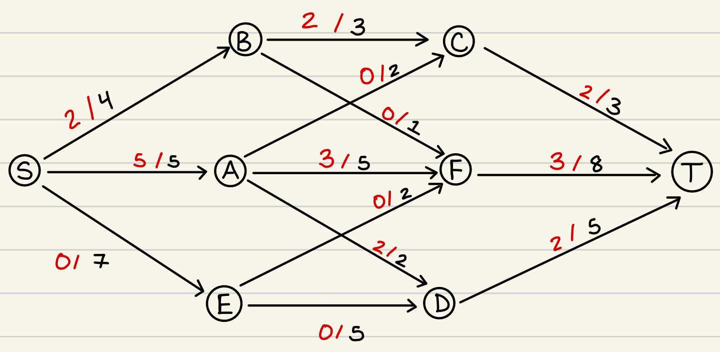


- Residual graph (capacity − 1):

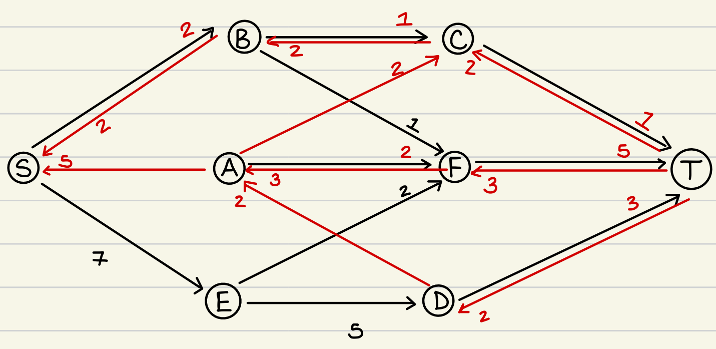


Step 4: Consider the path S → B → C $\overset{\leftarrow}{\rightarrow}$ A → F → T

- Flow graph (min cap : 2):

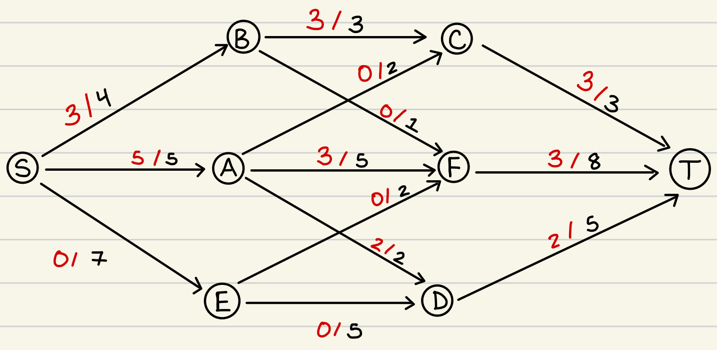


- Residual graph (capacity − 2):

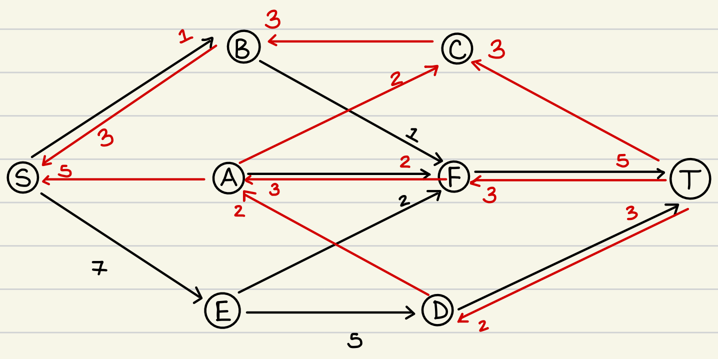


Step 5 : Consider the path S → B → C → T.

Flow graph (min cap : 1):

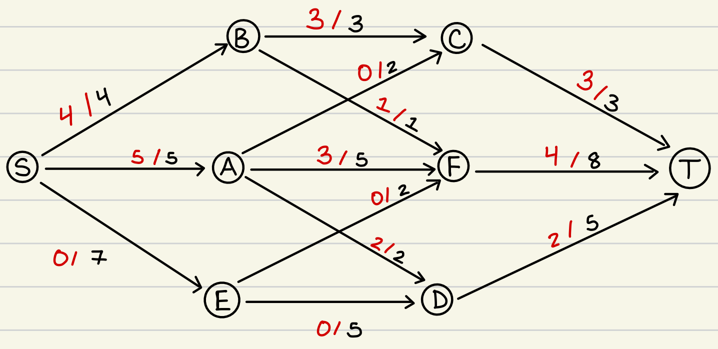


· Residual graph (capacity − 1):

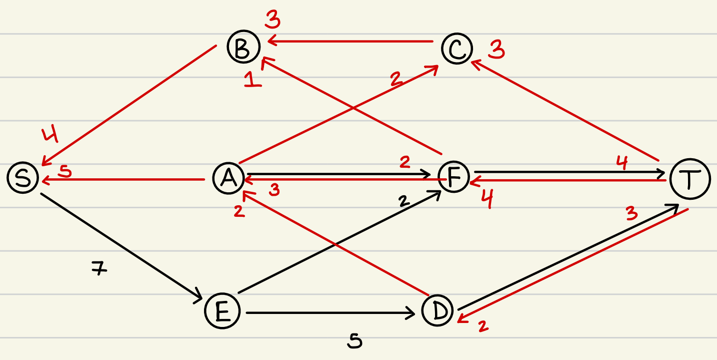


Step 6 : Consider the path S → B → F → T.

Flow graph (min cap : 1):

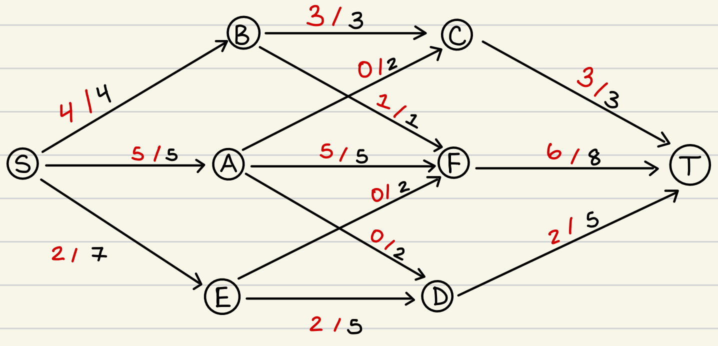


· Residual graph (capacity − 1):

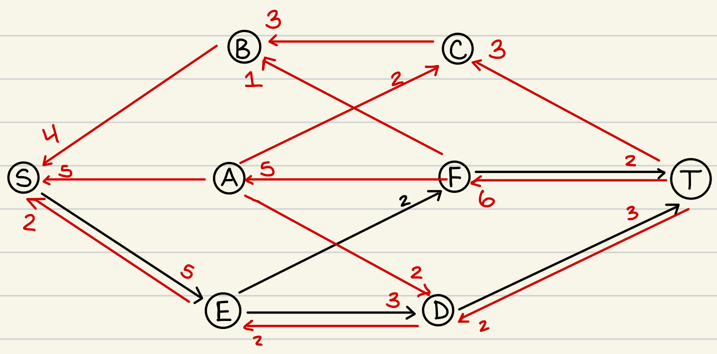


Step 7: Consider the path S → E → $\overleftarrow{D \to}$ A → F → T

Flow graph (min cap : 2):

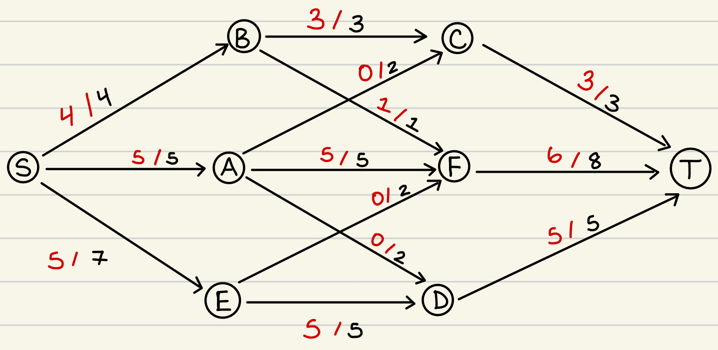


· Residual graph (capacity − 2):

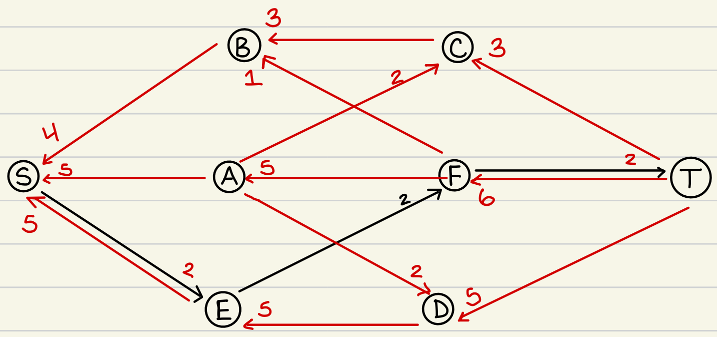


Step 8 : Consider the path S → E → D → T

Flow graph (min cap : 3):

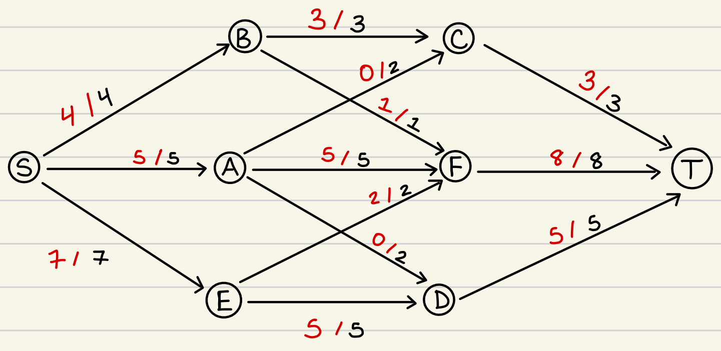


· Residual graph (capacity − 3):

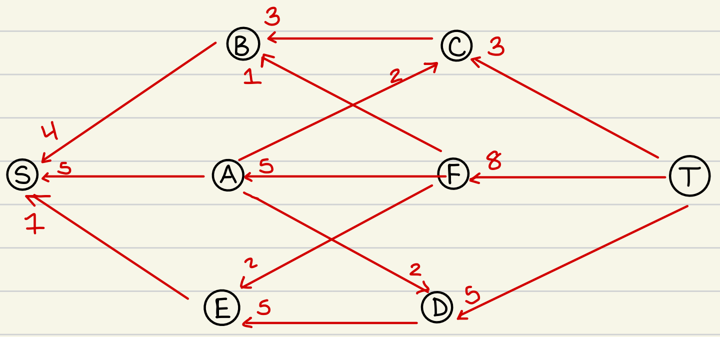


Step 9 : Consider the path $S \to E \to F \to T$

Flow graph (min cap : 2) :



· Residual graph (capacity − 2):



3 units flow from C to T, 8 units flow from F to T, and 5 units flow from D to T. So, 3 + 8 + 5 = 16 units reach the sink.

NP-Complete

· What does it mean to be in P?
  · The problem is solvable in polynomial time.
  · Decision prob $L \in P$ iff $\exists$ a polynomial algo A s.t. if X = Yes instance of L, A(X) = Yes. Same w/ No.

· In NP?
  · The answer can be verified in polynomial time. $\exists$ an algorithm $A(x, y)$, where X is an instance of a decision problem and y is the alleged solution. If X = Yes, $A(x,y)$ = yes.

· NP-Complete?
  · The problem L is in NP, and $\forall Y \in NP$, $Y \overset{\text{reduces to}}{\leq} L$ ($Y \to L$).
  · To show that L is NPC, show $L \in NP$ and that a known NPC problem (ex. 3-SAT) reduces to L.

· If we can reduce X to Y, that means Y is at least as hard as X.
  · if we have a Yes instance of X, f(X) is Yes

Ex. Show Triple 3-SAT $\in$ NP-Complete.
↳ Like 3-SAT, but needs at least 3 satisfying assignments

1) Show T3S $\in$ NP: Define an algorithm $A(x, y)$ that takes in an instance of Triple 3-SAT (x) and a certificate of 3 assignments (Y). Plug the 3 assignments into X. If all 3 return true, $A(x,y)$ = Yes. Else, $A(x,y)$ = No. If $\exists$ a Y s.t. X = Yes, then X is a Yes instance of T3S. If $\nexists$!, then X is a No instance.

2) Show a Known NPC problem reduces to Triple 3-SAT.

We know that 3-SAT is NP-Complete, so we WTS 3-SAT → Triple 3-SAT.

Union the 3-SAT expression with 1 clause of 3 distinct variables (not already present). If the original expression can be satisfied, then this new one can be too; simply set every new variable to F and change one var to T at a time to get 3 satisfying assignments.

If the original expression can't be satisfied, the new one is a No instance of Triple 3-SAT. Thus, a Yes instance of 3SAT → Yes of T3S, and a No instance → No Instance of T3S. Thus, 3Sat → Triple 3-SAT. Thus, Triple 3-SAT is NP-Complete.

Textbook

- GREEDY

Ex. The productivity of each programming pair = speed of slowest programmer. Assume an even # of coders. Maximize productivity.

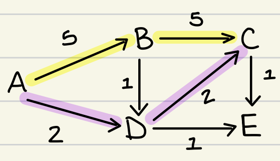
$P_1 = 2$

$P_2 = 6$ $\longrightarrow$ merge Sort the coders in decreasing order of productivity – $O(n \lg n)$.

$P_3 = 3$ Remove 2 coders at a time, and pair them.

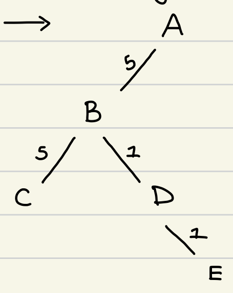
$P_4 = 4$

Ex. Each vertex represents a router, each edge a wire btwn routers. Each edge has a weight. Find the path w/ max weight from any source to any destination.

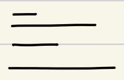


$\rightarrow$ Like Dijkstra, except you're picking the edge w/ the largest weight from source to vertex. Build the MST, and return the path you went. The max weight is the min. weight of the edges in the returned path.

$\longrightarrow$



Ex. Given n rods of lengths L_1, L_2, etc, minimize the cost of connecting all rods into one. The cost of connecting 2 rods = sum of lengths.



Insert the lengths into a min Heap. Remove 2, add them, insert the sum into the min Heap, and add that sum to a running total. Once the heap only has one item, remove it and add it to the running total. Return it.

- Divide + Conquer

Ex. 2 sorted arrays of size n. Find median after merging 2 arrays.

$[1, 10]$ $\rightarrow$ $[1, 10, 2, 6]$ $\longrightarrow$ merge Sort $\rightarrow$ take $\frac{n}{2}$ th element of array

$[2, 6]$

Ex. Unsorted arr of ints from 0 to $2^k - 1$, missing int M. Find M.

```
f(A[], i, j) = {  int med = $\frac{i+j}{2}$
                  if med ∉ A, return med.
                      else, put all elements < med on the left, and all ≥ med on the right.
                  if size(left) < med + 1, return f(left, i, med).
                      else return f(right, med + 1, j)
```

1 6 | 5 2

Ex. Search in a sorted singly linked list of size n.

f(list, target t) : Find the median element by going down the list $\frac{n}{2}$ times.
If med < t , then return f(list starting from this node, t)
else, f (first node, t)

Ex. Sort a singly linked list.

Just like merge sort

Ex. Given a sorted array of numbers that has been rotated a certain number of times, find X.

Find pivot point (where array "loops" around).
Sort numbers, like in quick sort.
Recur on the appropriate half (if X < pivot, recur on left half).

Ex. $n \times n$ array w/ ints. Each row, column is sorted in increasing order. Find X.

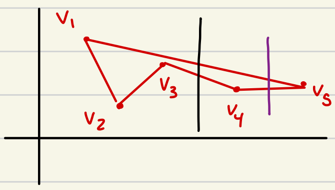
Base Case : if n = 1, return A[0] === x
if n = 2, do 4 cmps to see if x is any of the 4 elements

	1	2	3
1	1	7	8
2	2	9	10
3	4	13	15

Divide the array into 4 quadrants. Look at the bottom right element in the quadrants (this is the largest element). If the element < X, then throw away the quadrant. If element == x, then return X Else, recur on the quadrant

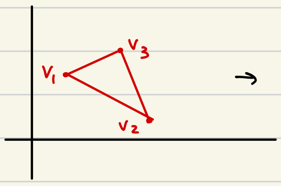
find 10

Ex. A convex polygon is represented as an array V w/ n vertices. V[1] = vertex w/ least x- coord , and V[1], V[2], etc are ordered counter clockwise. Find vertex w/ largest x- coord.



*Divide V in half. If A[n] < B[0], the x- coord is increasing ; recur on B. Else, recur on A.

★ If V.length = 1, return V[0].



→ V= [1,|5,| 2]

DYNAMIC

Ex. Return the length of the longest palindromic subsequence.

BC : s.length = 1, return 1

s.length = 2 → if chars match, return 2. else, return 1.

$OPT(s_1 \to s_n) = OPT(s_1 \to s_{n-1})$

$OPT(s_2 \to s_{n-1}) + 2$ if $s_1 = s_n$

- S = ABA

A = [0, 1, 1, 3]

↳ 2 + A[1]

Ex. Determine if str can be segmented into a sequence of dictionary words.

BC : if str.length = 1, return isWord(str)

if str.length = 0, return T

$OPT(s_1 \to s_n) = OPT(s_1 \to s_{n-i})$ && isWord($s_{n-i+1} \to s_n$) $1 \le i < n$

| isWord(str) → if T, return T

- str = MYDOG

A =	0	1	2	3	4	5
	T	F	T	F	T	T

☆ Ex. n balloons, indexed from 0 to $n-1$. Burst balloon i, get $A[l] \cdot A[i] \cdot A[r]$ coins. Find max coins you can collect.

BC ⇒ $i = j = 0$, no balloons to burst. Return 0.

$A[n+2][n+2] \to A[i][j] = \max[A[i][k] + A[k][j] + cost_i \cdot cost_j \cdot cost_k]$ $i+1 \le k \le j-1$

Ex. Rope w/ n units. Cut the rope into smaller pieces P_j, so that the product of lengths of the new smaller ropes is maximized. How to cut? (You must cut at least once)

BC : $n = 1$, return 1

$n = 2$, ↗

$1 \le i < n$, maximize over i

$OPT(n) = OPT(n-i)\, i$ OR you'll have to cut again after this

$(n-i)\, i$ this cut produces the max

	0	1	2	3	4	5
OPT(5) → A =	[0,	1,	1,	2,	4,	6]

OPT(3) = A[2] 1 = 1
A[1] 2 = 2
(2)(1) = 2

OPT(4) = A[3] 1 = 2
A[2] 2 = 2
A[1] 3 = 3
→ (2)(2) = 4

OPT(5) = A[4] 1 = 4
A[3] 2 = 4
A[2] 3 = 3
A[1] 4 = 4 → (2)(3) = 6

Ex. $n > 0$ jobs, each w/ t_i units of time, P_i income, S_i jobs you can't complete due to working on this job. Max income in T units of time.

	t_i	P_i	S_i	
Job 1:	1	5	1	
Job 2:	2	8	1	
Job 3:	3	9	2	

$T = 3$

BC : $T = 0$, return 0
$n = 1$ and $t_1 < T$: return P_1

$OPT(n, T) = OPT(n-1, T - t_n) + P_n$ do nth job
$OPT(n-1, T)$ don't do nth job

$A = [0, 5, 13, 13]$

Ex. String S of length n, T of m. Compute longest common subsequence.

BC : n or $m = 1$. if the one element is in the other string, return it.

$OPT(S_1 \to S_n, T_1 \to T_m) = OPT(S_1 \to S_{n-1}, T_1 \to T_{m-1}) + 1$ if $S_n = T_m$
$\max[OPT(S_1 \to S_{n-1}, T_1 \to T_m), OPT(S_1 \to S_n, T_1 \to T_{n-1})]$

	A	E	B	C
A	1	1	1	1
B	1			
C				
D				

Ex. 2 rooms to rent out, n customers, ith cust $\to$ 1 room for $d[i]$ days, for \$ bid[i]. Find max profit over D days.

BC : $d_1 = d_2 = 0$, return 0
$n = 0$, return 0

$$OPT(d_1, d_2, n) = \max \begin{cases} OPT(d_1 - d[n], d_2, n-1) + bid[n] \\ OPT(d_1, d_2 - d[n], n-1) + bid[n] \\ OPT(d_1, d_2, n-1) \end{cases}$$

days remaining for room 1

give room 1 to nth

give room 2

turn away nth customer

$D = 3$
$n = 1 \to 3$ days, \$30
$n = 2 \to 2$ days, \$20
$n = 1 \to 1$ day, \$15

$d_1 \times d_2 \times n$ array. Fill in, increasing d_1 first, then d_2.
$\downarrow$
ex. $A[1, 0, 1]$
$\vdots$
$A[1, 0, n]$
$A[2, 0, 1]$
etc

Ex. $C_1, \dots, C_n$ classes happening. Each C_i has start time s_i, finish time f_i, s.t. $s_i < f_i$, score v_i. Max that score.

→ it's like weighted interval scheduling

Sort by earliest finish time. BC: $n = 0$, return 0

$\text{OPT}(C_n \text{ classes}) = \text{OPT}(C_0 \rightarrow C_{n-1}) + v_n$ → take nonoverlapping classes

$\text{OPT}(C_{n-1})$ → discard n^{th} class

$$A = \begin{array}{c|c|c|c} 0 & 1 & 2 & 3 \\ 0 & 10 & 10 & 10 \end{array}$$

C_1 : 9am - 12pm, 10

C_2 : 10am - 11am, 7

C_3 : 11am - 12pm, 2

Ex. n trading posts numbered $n, n-1, \dots, 1$. At any post, you can go downstream. Find min cost from each i to each j.

$$\begin{tikzcd} j & 1 & & 2 \arrow[ll, "1"'] & \\ & & 3 \arrow[ul, "5"] \arrow[ur, "1"'] & i & \end{tikzcd}$$

BC: $i = j \rightarrow$ return 0

$i < j \rightarrow$ return 0

$\text{OPT}(i, j) = \text{OPT}(i, r-1) + \text{OPT}(r+1, j) + \text{cost}_{i \rightarrow r}$ $\quad j \leq r < i$

$i \downarrow$ $\overset{j}{\rightarrow}$	1	2	3
1	0	0	0
2	1	0	0
3		1	0

$\text{OPT}(j) = \text{OPT}(k) + \text{cost}[k, j]$ $\quad 1 \leq k < j$, minimize over k

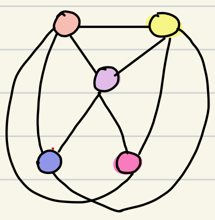
because we are only going downstream

3-SAT, 3-Color, Independent Set, Vertex Cover ∈ NPC.

NP-Complete

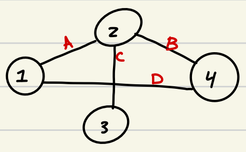
Ex. 3-Color → 5-Color
↓
given undirected graph X,

add 2 new nodes of distinct colors that haven't appeared before. Connect each og node to these nodes.



Thus, if X = Yes in 3-color, X = Yes in 5-Color

· Vertex Cover : a set of vertices s.t. each edge in the graph is incident to at least 1 vertex



Vertex cover : {1, 2}

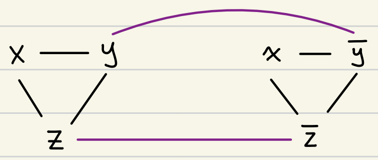
· 3-SAT → Independent Set → Vertex Cover
↳ set of unconnected vertices, size ≥ K
↓

given an expression w/ 3-literal clauses, OR's inside, AND's outside.

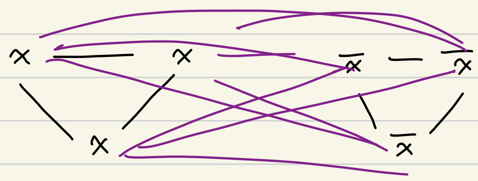
$(x \cup y \cup z) \cap (x \cup \bar{y} \cup \bar{x})$

· K = # clauses

· For each clause, make the literals into nodes and connect them like a triangle. Connect complements ACROSS clauses.



$(x \cup x \cup x) \cap (\bar{x} \cup \bar{x} \cup \bar{x})$



Ex. n ingredients. nxn array D → penalty of combining i and j. decision prob A

Independent Set → A
↓

given graph, size K
↓
nodes = ingredients
edges = penalty (no edge if D[i][j] = 0 , yes edge if D[i][j] = 1)
at least K

give D :

	1	2	...
1	0	1	
2	0	0	

Set p = 0.
IS(G, K) = Yes iff A(K, D, 0) = Yes. We can choose K ingreds w/ p=0 iff ∃ independent set w/ size K

Ex. n people, L pairs of people. Can these n ppl be seated around a circular table s.t. each person has friends on both sides? Prove NPC.

1. $D \in NP$.

Define an Algo $A(x, y)$ s.t. X = instance of decision prob and Y = certificate, configuration of the table.

Create an nxn array of people, arr[n][n] = T if friends, F if not. At each seat, find arr[n][i] and arr[n][j]. if one returns false, then break − $A(x, y)$ = No.

2. $\rightarrow D$

Ex. Given undirected, weighted graph $G = (V, E)$ w/ positive edge costs, a subset R of vertices, and constant C. Prob A: is there a tree in G that spans all vertices in R w/ a total edge cost $\leq C$.

$$\begin{tikzcd}
\text{Vertex Cover} \arrow[r] \arrow[d] & A(G, R, C) \arrow[d] \arrow[r, phantom, "\hookrightarrow =0"] & {} \\
\text{given } G, \text{ size } K & \text{a set that satisfies Vertex cover} &
\end{tikzcd}$$