Dynamic Programming

A problem-solving approach, in which we precompute and store simpler, similar subproblems, in order to build up the solution to a complex problem.

Dynamic programming is used if there is

- **Optimal substructure**

  Problem can be solved using the solution of subproblems.
- **Overlapping subproblems**

  Larger problems are divided into smaller problems.

  And there may be duplicate sub problems.

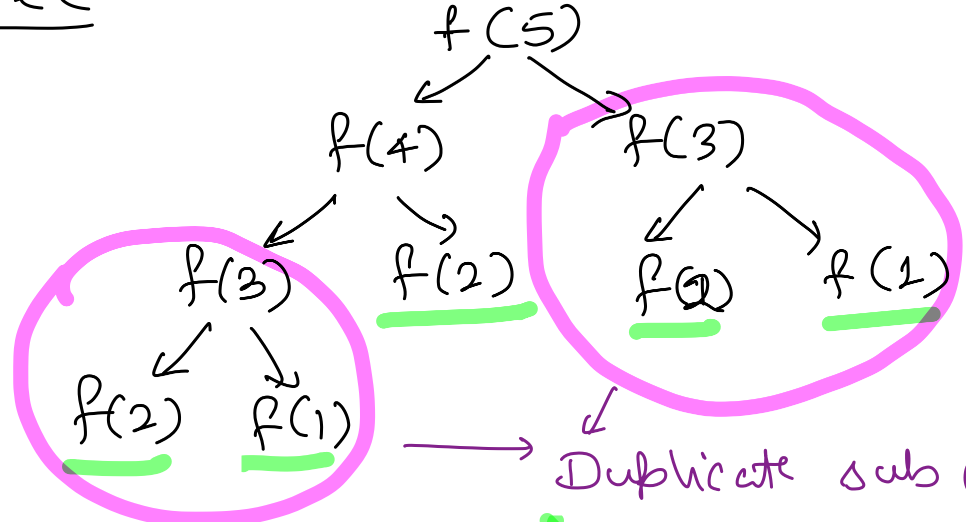
  Memoization - Each unique problem is solved once and its solution is remembered.

Find n^{th} term of fibonacci series.

$$f(n) = f(n-1) + f(n-2)$$

$$f(n) = \begin{cases} 1, & \text{if } n=0 \text{ or } n=1 \\ f(n-1) + f(n-2), & \text{otherwise.} \end{cases}$$

Recursion Tree



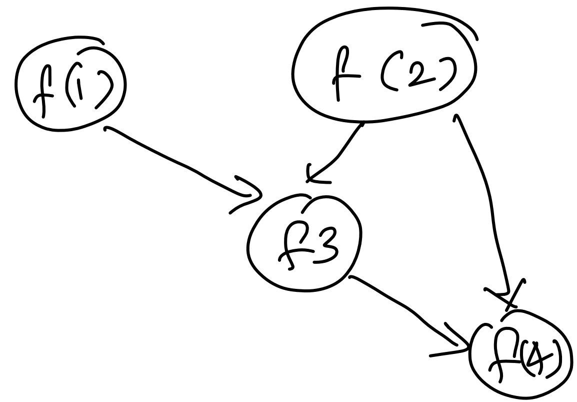
Duplicate sub problems.

Top Down

Find solution of larger problem by breaking it into smaller sub problems.

Bottom up

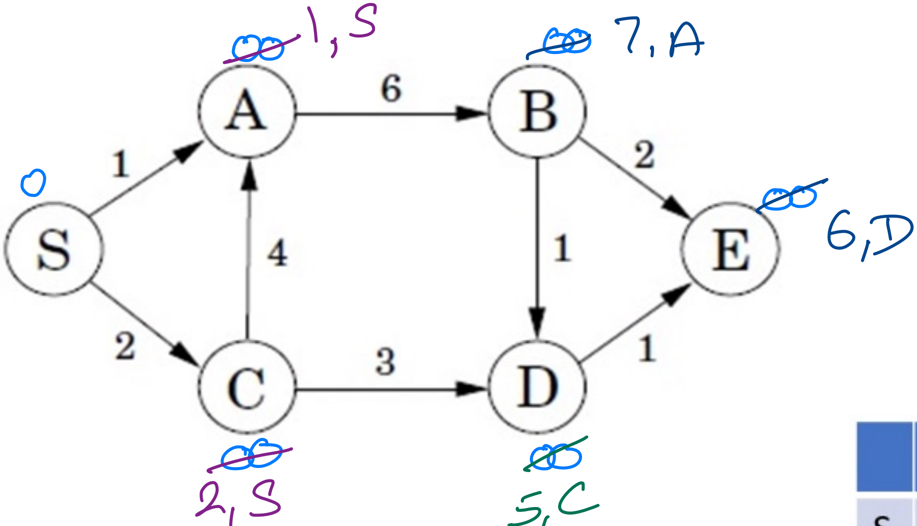
Build the solution from bottom (base case) an work towards root (larger problem).



Dijkstra's Shortest Path (startVertex, endVertex) → Single Source Shortest Path

← Greedy algorithm.

- Set currentDistance for all vertices as infinity.
- Set currentDistance for startVertex as 0.
- Create a list of all vertices in graph, vertexList.
- while vertexList is not empty do
  - Get a vertex, u, from vertexList such that u has smallest distance
  // Update the distance of each vertex v, adjacent to u.
  - distanceToVviaU = currentDistance[u] + weight of edge (u, v)
  - if (currentDistance[v] > distanceToVvisU) then
    - Set currentDistance[v] to distanceToVviaU
    - Set predecessor of v to u.
- Stop



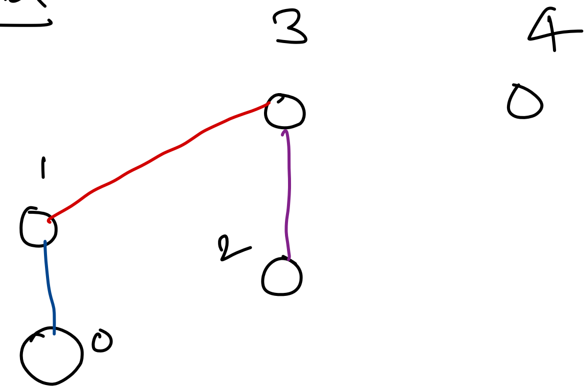
Dist $S \rightarrow E = 6$

Path $E \leftarrow D \leftarrow C \leftarrow S$

~~S~~ ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~

	Init	S	A	C	D	E	B
S	0						
A	∞	1,S					
B	∞	∞	7,A	7,A	7,A	7,A	
C	∞	2,S	2,S				
D	∞	∞	∞	5,C			
E	∞	∞	∞	∞	6,D		

Union Find



p q

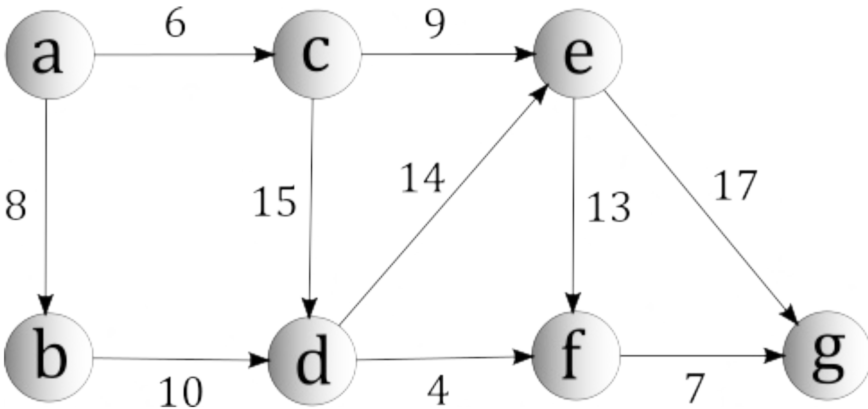
union (0, 1)

union (2, 3)

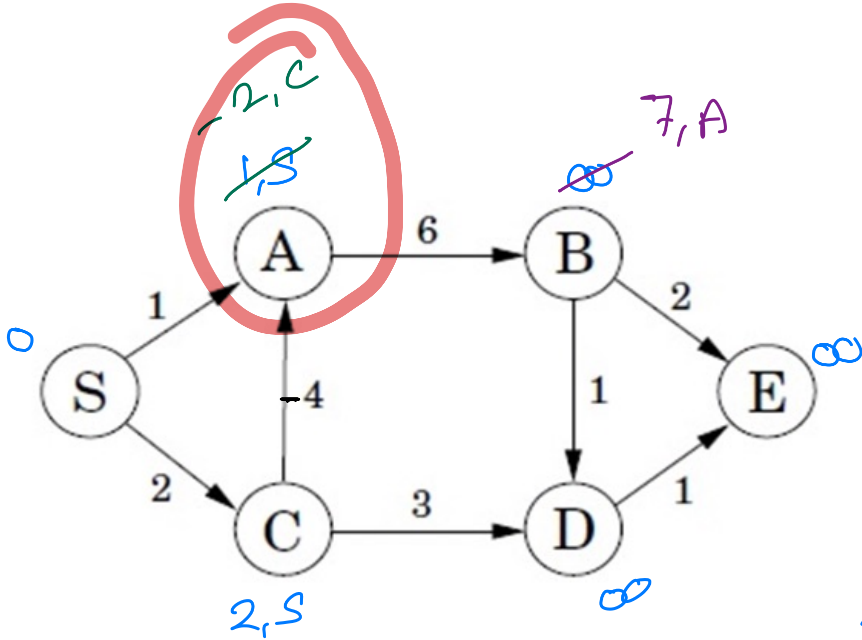
union (1, 3)

0	1	2	3	4
~~1~~ ~~0~~ 3	~~1~~ 3	3 ~~2~~	3	4

ids



	Init	a	c	b	e	d	f	g
a	0							
b	∞	8,a	8,a					
c	∞	6,a						
d	∞	∞	21,c	18,b	18,b			
e	∞	∞	15,c	15,c				
f	∞	∞	∞	∞	28,e	22,d		
g	∞	∞	∞	∞	32,e	32,e	29,f	

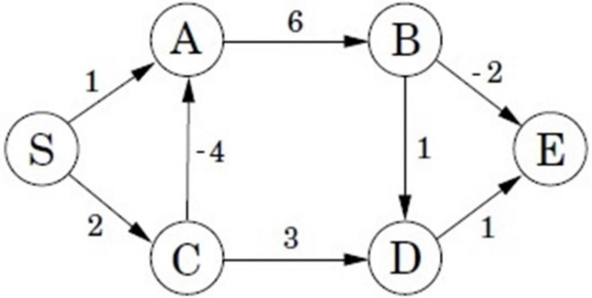


~~S~~ ~~A~~ B ~~C~~ D E

Bellman-Ford Shortest Path Algorithm

Like Dijkstra, it is also a single-source shortest Path algorithm.
It allows edges with negative weight in the graph.

Bellman-Ford Shortest Path Algorithm
- Set the current distance for all vertices as infinity.
- For startVertex, set the current distance to 0.
- Create a list of all edges in the graph.
- For |V| - 1 times do
  - distanceToVviaU = currentDistance[u] + weight of edge (u, v)
  - if (currentDistance[v] > distanceToVviaU) then
    - Set currentDistance[v] to distanceToVviaU
    - Set predecessor of v to u
- Done.

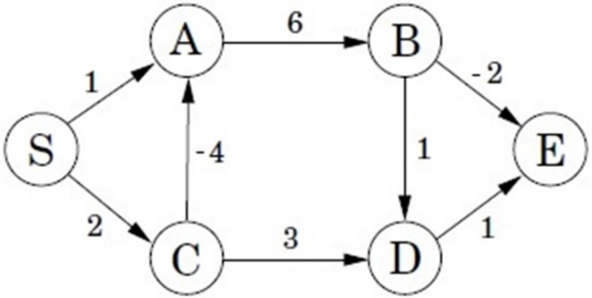


	Init	1st Iter	2nd Iter	3rd Iter	. . .	5th Iter
S	0					
A	∞	~~1, S [SA]~~ -2, C [CA]				
B	∞	7, A [A B]	4, A [AB]			
C	∞	2, S [S C]				
D	∞	5, C [CD]				
E	∞	5, B [BE]	2, B [BE]			

currentDistance

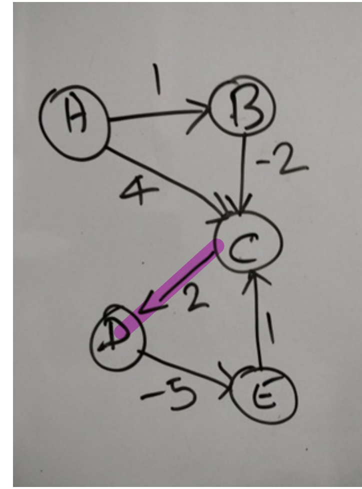
edges

u	v	w
S	A	1
S	C	2
A	B	6
C	A	-4
C	D	3
B	E	-2
B	D	1
D	E	1



Current Distance

	Init	1	2	3	4	5
S	0	0	0	0	0	0
A	∞	-2,C	-2,C	-2,C	-2,C	-2,C
B	∞	7,A	4,A	4,A	4,A	4,A
C	∞	2,S	2,S	2,S	2,S	2,S
D	∞	5,C	5,C	5,C	5,C	5,C
E	∞	5,B	2,B	2,B	2,B	2,B

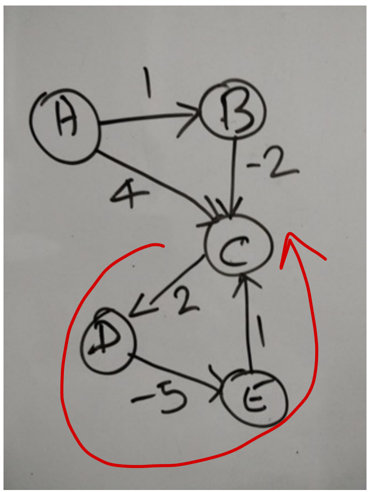


	Init	1	2	3	4
A	0	0	0	0	0
B	∞	1,A	1,A	1,A	1,A
C	∞	-3,E	-5,E	-7,E	-9,E
D	∞	1,C	-1,C	-3,C	-5,C
E	∞	-4,D	-6,D	-8,D	-10,D

Negative weight cycle

$\Downarrow$

Sum of weight of edges in cycle is negative.



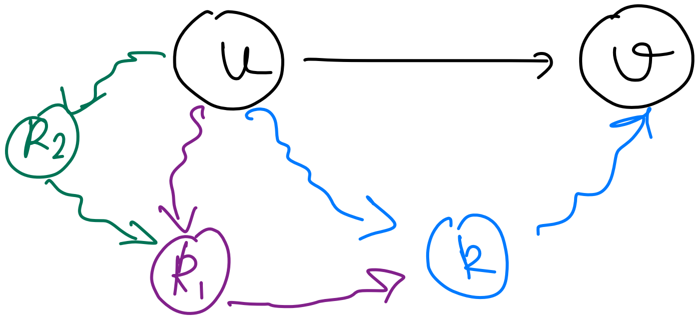
$2 - 5 + 1 = -2$

$\Uparrow$

Graph with negative weight cycle do not have shortest path.

Execute loop in Bellman Ford algorithm for one extra time.

If current Distance for any vertex is further reduced then graph has negative weight cycle.


u
v
R_2
R_1
R

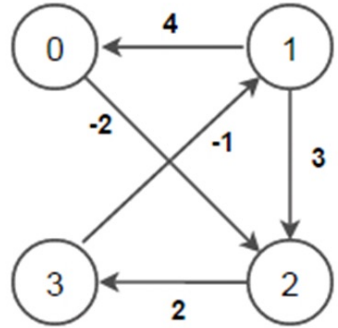
Floyd-Warshall - All-pairs shortest path

Find shortest paths in a weighted graph with positive and negative edge weights (but no negative weight cycles).

Break every possible path between two vertices (u, v) by inserting another vertex k.

Compare the sum of distance of two new paths formed (uk, kv) with the original path (uv) and record shorter path.

Floyd-Warshall - All-pairs shortest path
- Start with adjacency matrix, d,representing the graph and initialize path and
  set non-adjacent vertex distanceto infinity
- For each vertex k from 0 to |V| - 1
  - For each vertex u from 0 to |V| - 1
    - For each vertex v from 0 to |V| - 1
      - if (d[u][v] > (d[u][k] + d[k][v])) then
        - d[u][v]= d[u][k] + d[k][v]
        - path[u][v]= path[k][v]
- Stop



Init	0	1	2	3
0	0	∞	-2,0	∞
1	4,1	0	3,1	∞
2	∞	∞	0	2,2
3	∞	-1,3	∞	0

K = 0	0	1	2	3
0	0	∞	-2,0	∞
1	4,1	0	2,0	∞
2	∞	∞	0	2,2
3	∞	-1,3	∞	0

K = 1	0	1	2	3
0	0	∞	-2,0	∞
1	4,1	0	2,0	∞
2	∞	∞	0	2,2
3	3,1	-1,3	1,0	0

K = 2	0	1	2	3
0	0	∞	-2,0	0,2
1	4,1	0	2,0	4,2
2	∞	∞	0	2,2
3	3,1	-1,3	1,0	0

K = 3	0	1	2	3
0	0	-1,3	-2,0	0,2
1	4,1	0	2,0	4,2
2	5,1	1,3	0	2,2
3	3,1	-1,3	1,0	0