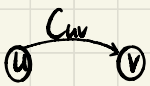
- SSSP [Single Source Shortest Path]

Given graph $G=(V, E)$, list E → Distance, and a starting vertex s
find the shortest path distance from s to all other $u \in V$

Have negative weighted edge	DAG: Y	DAG: N
Y	SP on DAG $O(n+m)$	Bellman Ford $O(mn)$
N		Dijkstra's algo $O(m \log n)$ $O(n^2)$

Pre-defined helper function
input : $G=(V, E)$, starting vertex s
output : dist[u] : shortest distance from s to u

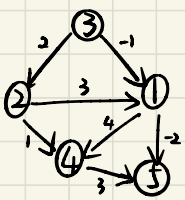
```
relax(u, v){
    if(dist[u] + Cuv < dist[v]){
        dist[v] = dist[u] + Cuv
    }
}
```



1. Shortest Path on DAG

input : $G=(V, E)$, s

dist[s] = 0 ; dist[rest vertices] = Int.MAX
Topological Sort ⇒ $O(n+m)$
for(u in topological ordering){ ⇒ $O(n+m)$
    for(v in G.neighbor(u) {
        relax(u, v)
    }
}
Overall : $O(n+m)$



tp sort : [3, 2, 1, 4, 5] Start = 3

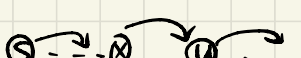
	1	2	3	4	5
dist	~~∞~~ -1	~~∞~~ 2	0	~~∞~~ 3	~~∞~~ -3

u=3
- 3 → 1 : dist[3] + (-1) < dist[1]
- 3 → 2 : dist[2] + (2) < dist[2]

u=2
- 2 → 1 : 2+3 > -1
- 2 → 4 : 2+1 < ∞

u=1
- 1 → 4 : (-1)+4 = 3
- 1 → 5 : (-1)+(-2) < ∞

u=4 ...

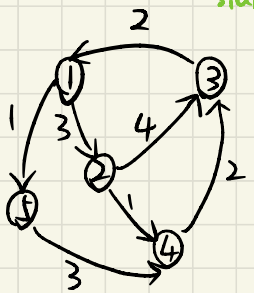


Only vertices before u can have impact on dist[u]

$dist[u] = \min\{ dist[x] + C_{xu} \}$ $(s \le x < u)$

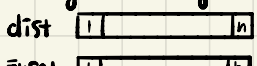
2. Dijkstra's Algorithm

```
dist[];
N={}
while(|N|!=n){ // Not all vertices have been examined
    ① pick vertex u∈V\N with smallest dist[] value
    ② N=N∪{u} // Add u to set N
    ③ for(v in G.neighbor(u))
        relax(u,v)
}
```



How to choose the minimum dist[]

1. Using an array to store then iterate to



pick the smallest u

① pick u O(n)
② N=N∪{u} O(1)
③ relax its neighbor O(du)

} O(n)

	1	2	3	4	5
dist	~~0~~	0	~~4~~ 3	∞	∞

N={2}

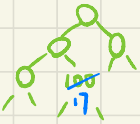
u=2 ②→③ 0+4<∞
②→④ 0+1<∞ N={2,4}
u=4 ④→③ 1+2<4 N={2,4,3}
u=3 ③→① 3+2<∞ N={2,4,3,1}

2. Using MinHeap

① pick u O(1)
② Delete u from heap O(logn)
③ relax u's neighbors O(du logn)

d1logn + d2logn + ··· + dnlogn = mlogn

Heap property is compromised
You need to swap 17 with its parent
until the heap property is restored



3. Bellman-Ford

dist[s] = 0

```
for(i = 1, ..., n-1){
    for(u = 1, ..., n){
        for(v in G.neighbor(u)){
            relax(u, v);
        }
    }
}
```

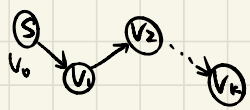
n-1 times

go over all m edges.

Do one more relax to check negative cycle

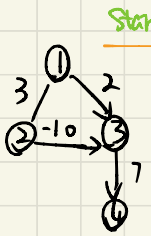
```
for(u = 1 ... n){
    for(v in G.neighbor(u))
        relax(u, v)
}
```

if dist[] changes, there is a negative cycle.



Simple shortest path: $k \leq |V|-1$

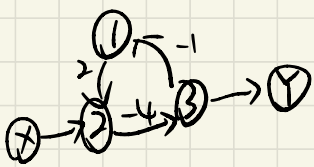
if $k > |V|-1$, means there is a cycle



[2, 3, 4, 1] Going vertex in this order

i = 1 ~~②~~ ~~③~~ ~~④~~ ①

i = k, if #edges ≤ k



In this graph, after n-1 iteration, if you run one more relax, the dist[] will change, used for detect negatively weighted cycle.

Proof of correctness:

This path p is a shortest path with min #edges

Grow the frontier one node at every iteration

After 1 pass on E, we have dist[v_1] because we will relax edge (v_0, v_1)

After 2 passes on E, we have dist[v_2] because we will relax edge (v_1, v_2)

⋮

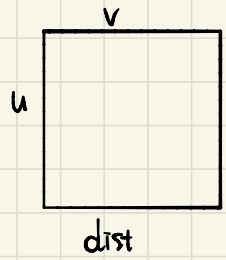
After k passes, we have dist[v_k] because we have relaxed all reachable vertices

Update relax() function to record the short path

```
relax(){
    if(dist[u] + Cuv < dist[v]){
        dist[v] = dist[u] + Cuv;
        parent[v] = u;
    }
}
```

ASAP (All pair shortest path)

dist[u,v] = shortest distance from u to v



Implementing SSSAP n times

DAG? (n iteration)

Y	N	
SPonDAG $O(n^2+nm)$	Dijkstra $O(m\log n)$	N
$n^2 \sim n^3$	BF $O(n^2m)$	Y
	$n^3 \sim n^4$	

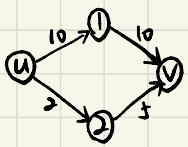
$n^2\log n \sim n^3$ Negative weight?

m: # of edges $\Omega(n) \le m \le O(n^2)$

$f^3(u,v) = \min\{f(u,v),\ f(u,2)+f(2,v)\}$

Floyd-Warshall Algorithm

$f^k(u,v)$ = shortest distance from u to v with only vertices $\in \{1,\cdots,k\}$ on shortest path

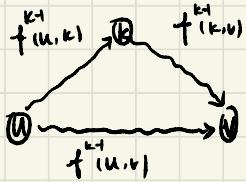


$f^1(u,v) = 20$, $V=\{1\}$, $f^0(u,v) = Inf$

$f^2(u,v) = 7$ $V=\{1,2\}$

Base Case

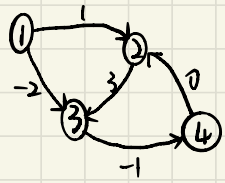
$$f^0(u,v) = \begin{cases} C_{uv}, & (u,v)\in E \\ \infty, & \text{otherwise} \end{cases} \Rightarrow f^n(u,v) = \text{true shortest distance}$$



$$f^k(u,v) = \min\{\underbrace{f^{k-1}(u,v)}_{\text{Not using}},\ \underbrace{f^{k-1}(u,k)+f^{k-1}(k,v)}_{\text{using intermediate}}\}$$

Pseudocode for Floyd-Warshall

```
FW(G){
    dist[,] = adj matrix ; // If two vertices are connected, dist[u,v] = weight
    for(k=1,···,n){                // o/w dist[u,v] : Inf
        for(u=1,···,n){
            for(v=1,···,n){
                dist[u,v] = min{ dist[u,v] , dist[u,k] + dist[k,v] };
            }
        }
    }
```



In this algo, better to use Adj Matrix

	1	2	3	4
1	0	1	-2	/
2	/	0	3	/
3	/	/	0	-1
4	/	0	/	0

k=1 // Not updating any

	1	2	3	4
1	/	1	-2	/
2	/	/	3	/
3	/	/	/	-1
4	/	0	/	/

k=2

	1	2	3	4
1	0	1	-2	/
2	/	0	3	/
3	/	/	0	-1
4	/	0	3	0

k=3

	1	2	3	4
1	0	1	-2	-3
2	/	0	3	2
3	/	/	0	-1
4	/	0	3	0

Check diagonal value after each iteration, if one of value is <0, then there is negatively weighted cycle.