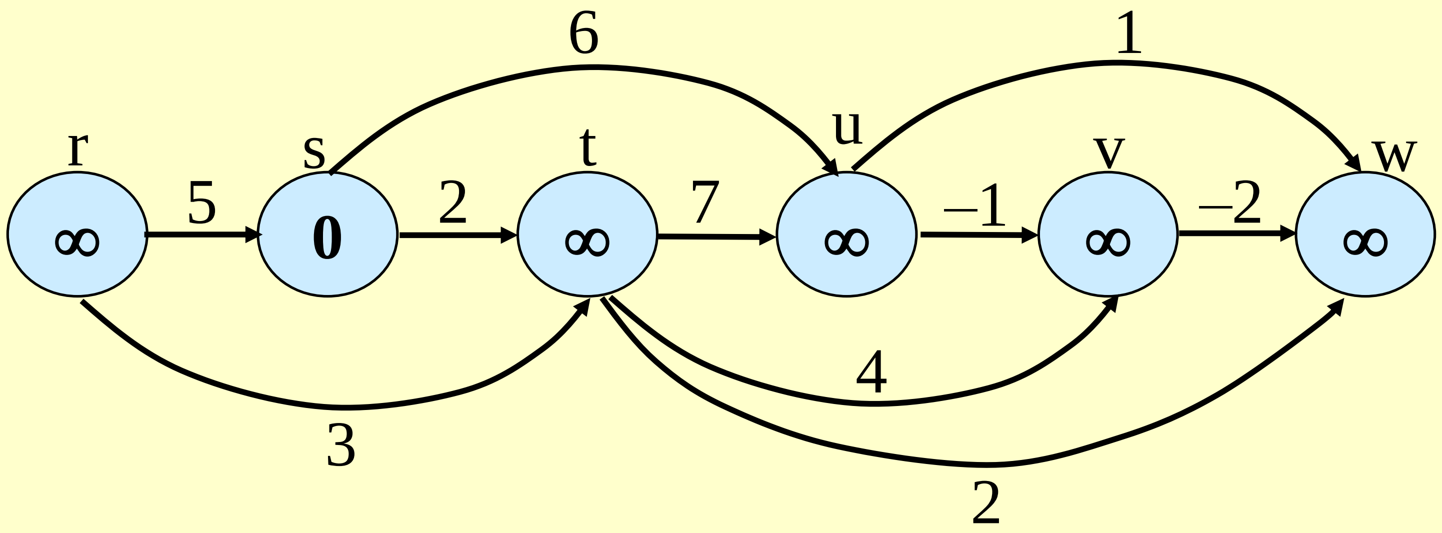
Shortest Paths in DAGs (like BFS)

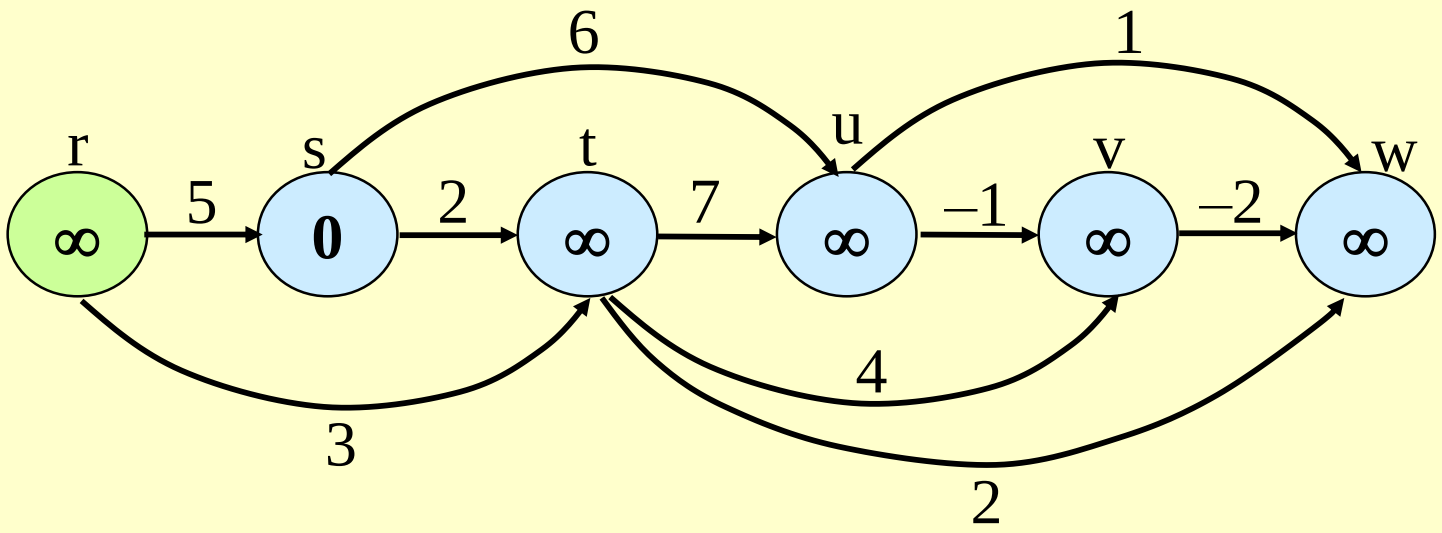
```
Topologically sort vertices in G;
Initialize(G, s);
for each u in V[G] (in order) do
for each v in Adj[u] do
      Relax(u, v, w)
od
od
```

Relax edges out of parents before you expand the child node

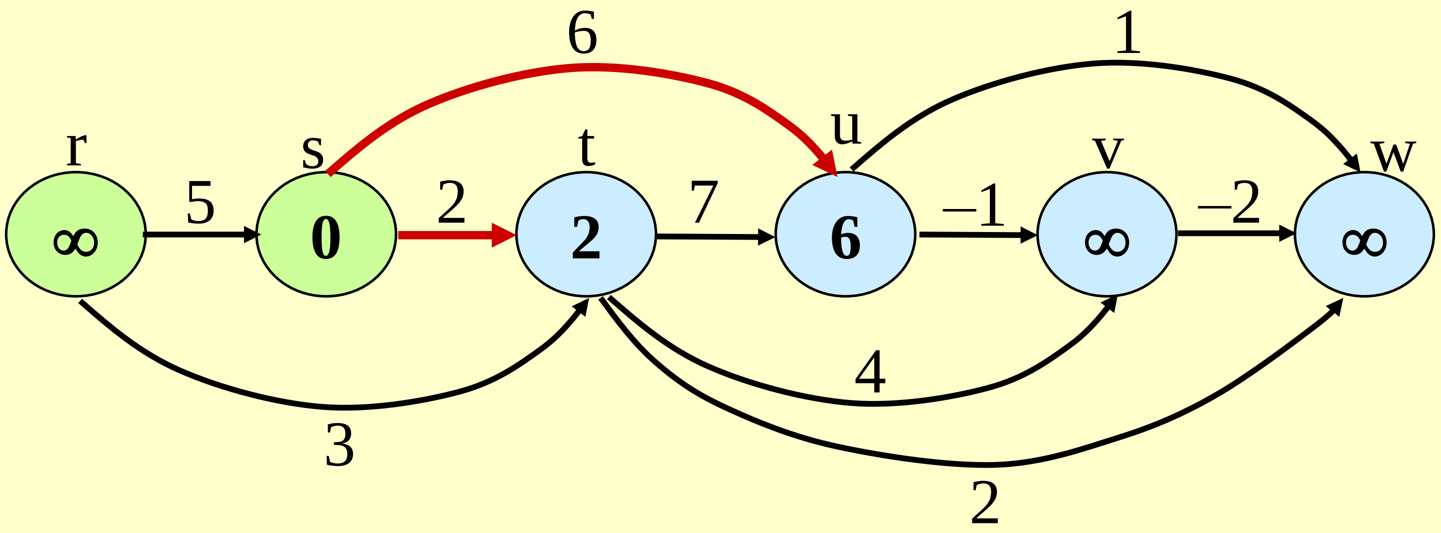
Example



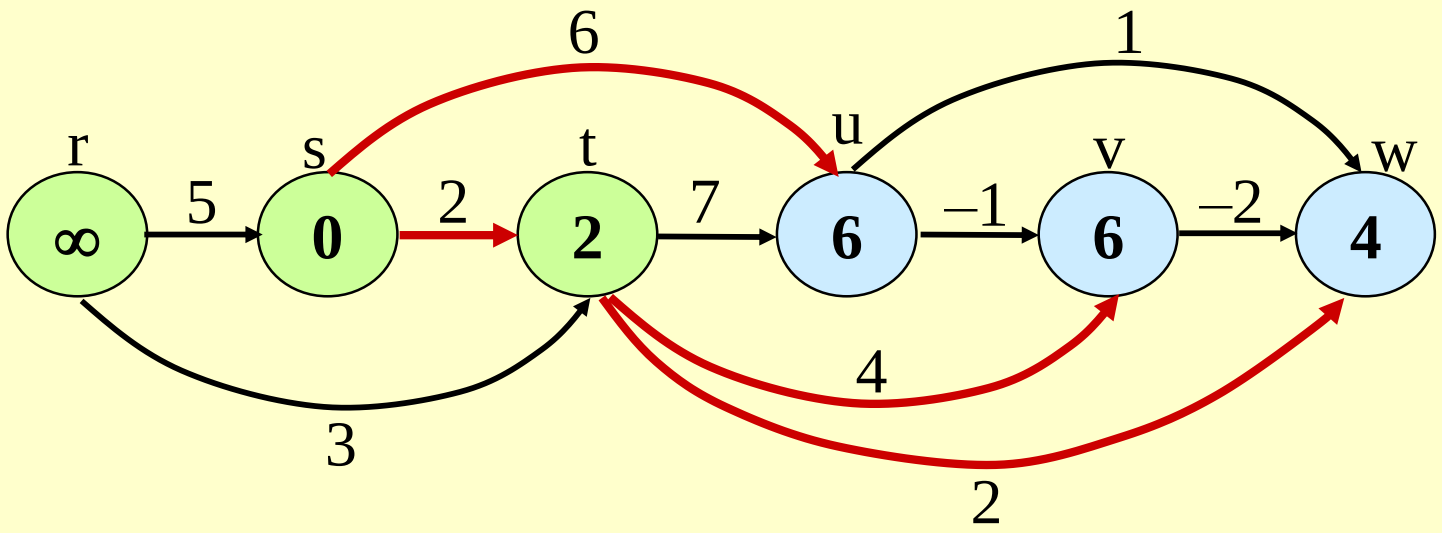
Example



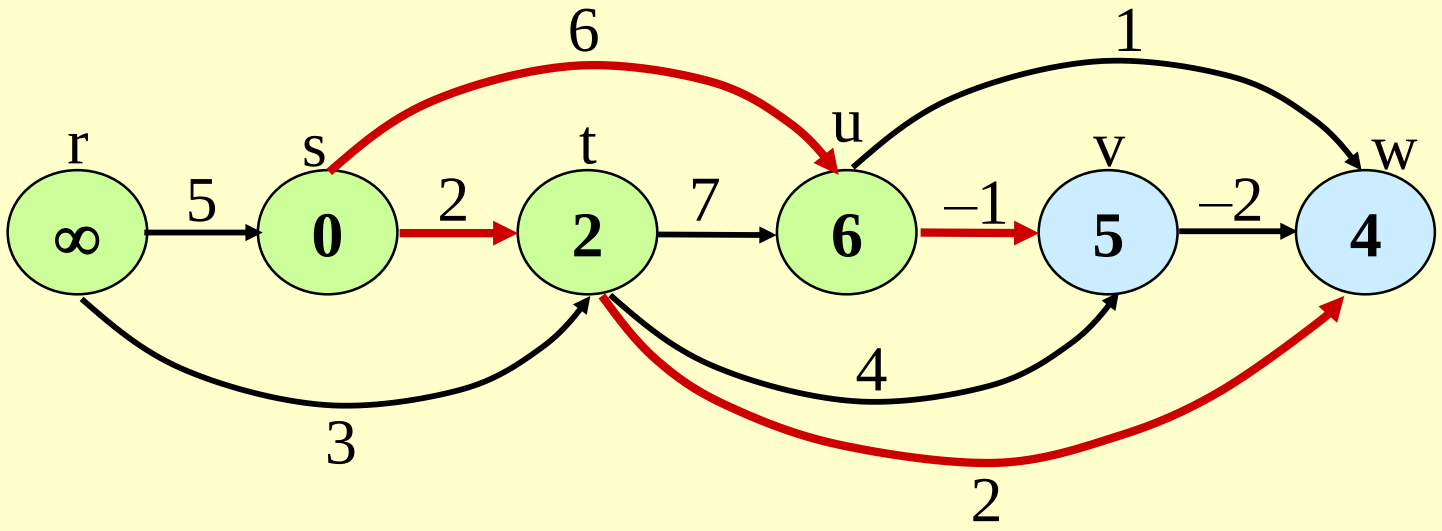
Example



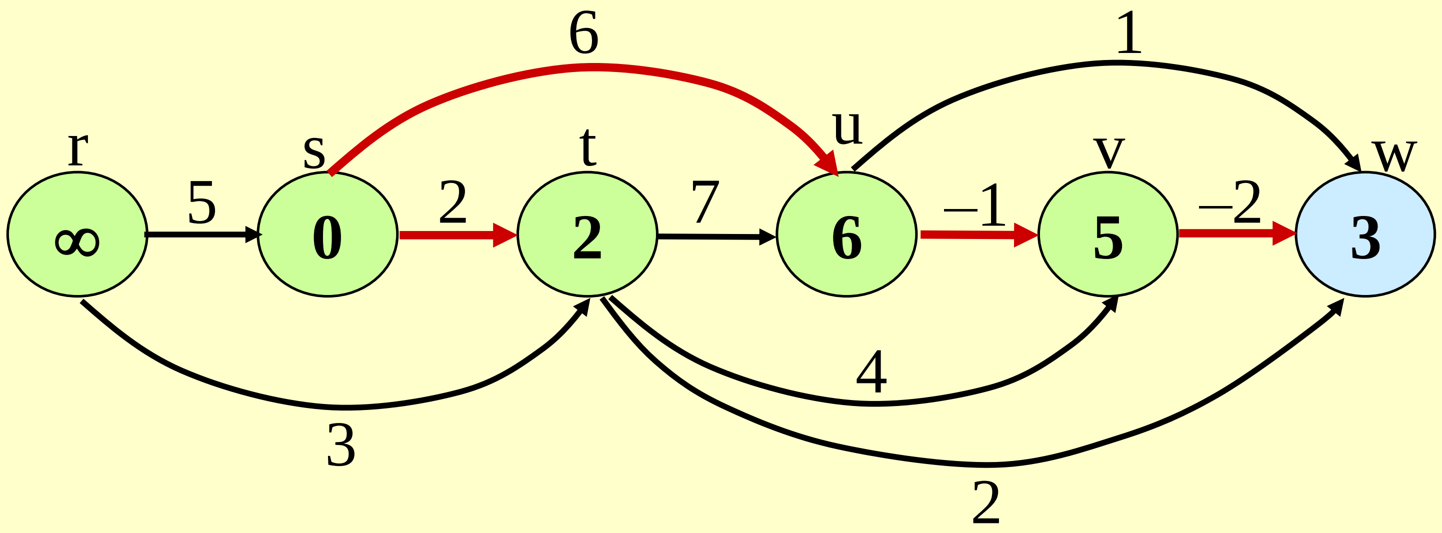
Example



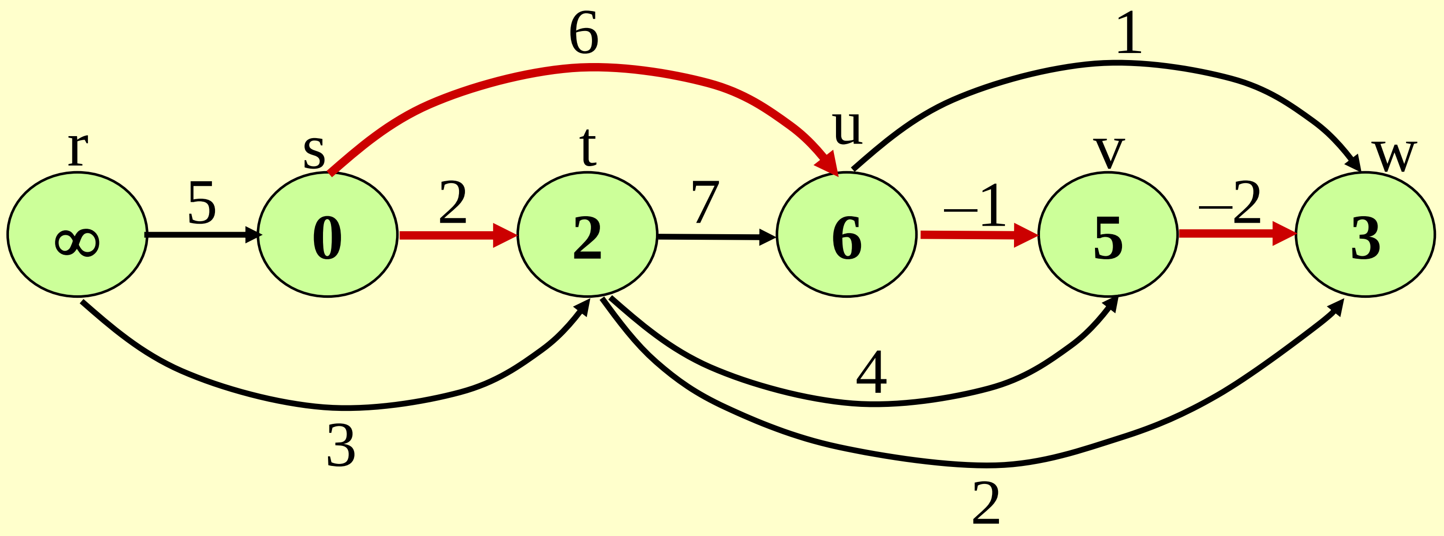
Example



Example



Example



Dijkstra's Algorithm

Assumes **no negative-weight edges**.

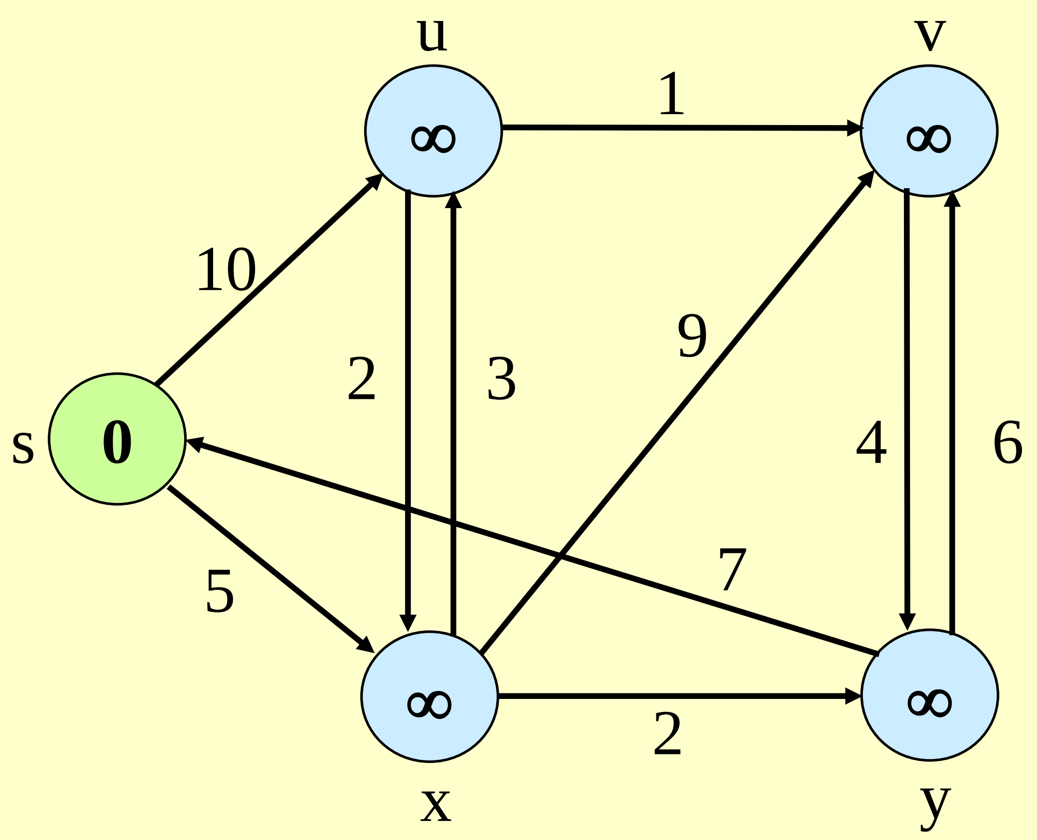
Maintains a set S of vertices whose SP from s has been determined.

Repeatedly selects u in V–S with minimum SP estimate (greedy choice).

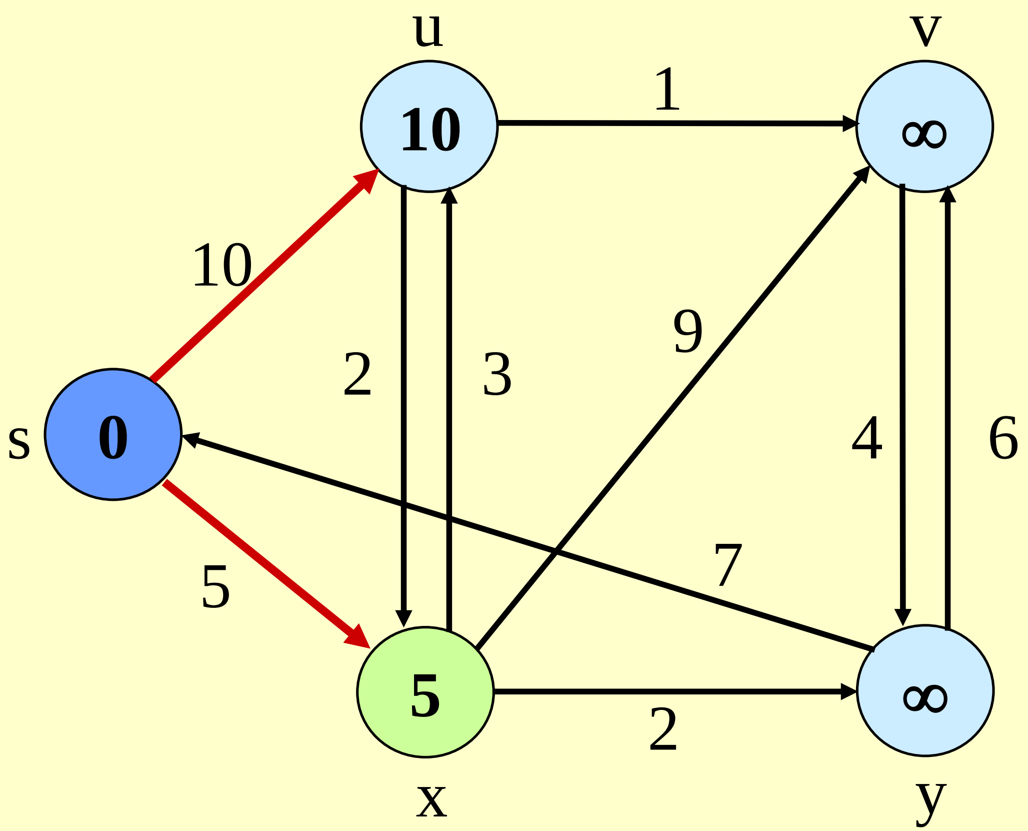
Store V–S in priority queue Q.

```
Initialize(G, s);
S := ∅;
Q := V[G];
while Q ≠ ∅ do
u := Extract-Min(Q);
S := S ∪ {u};
for each v ∈ Adj[u] do
    Relax(u, v, w)
od
od
```

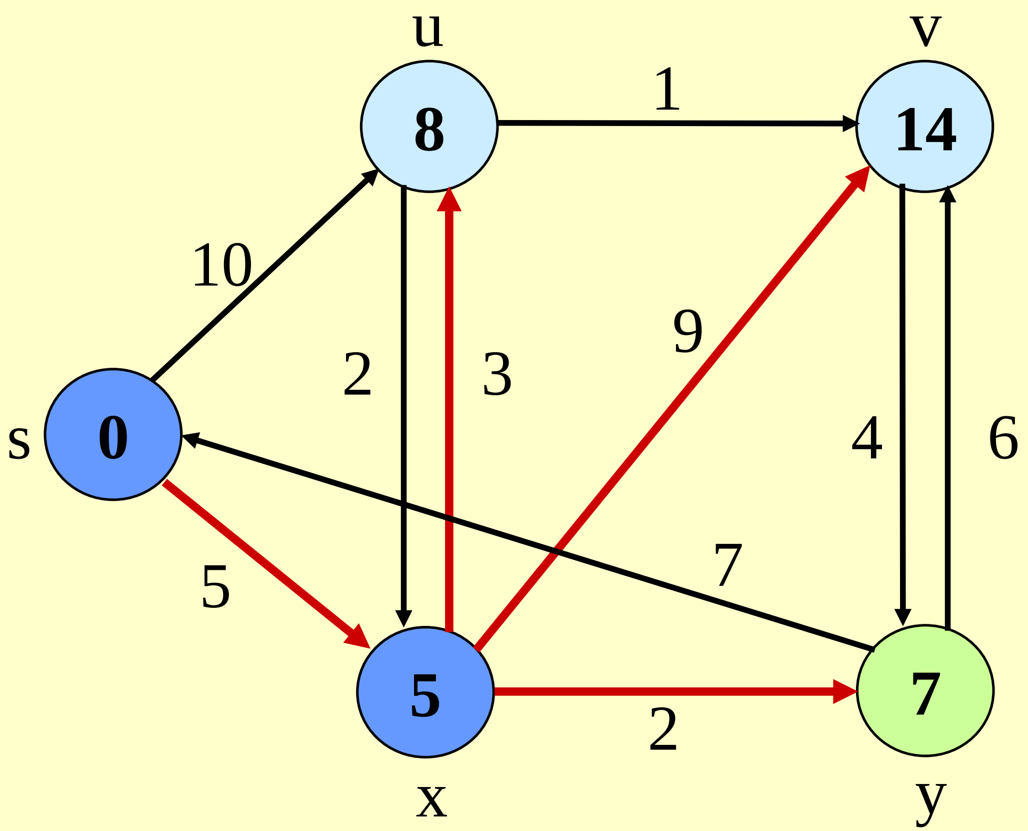
Example



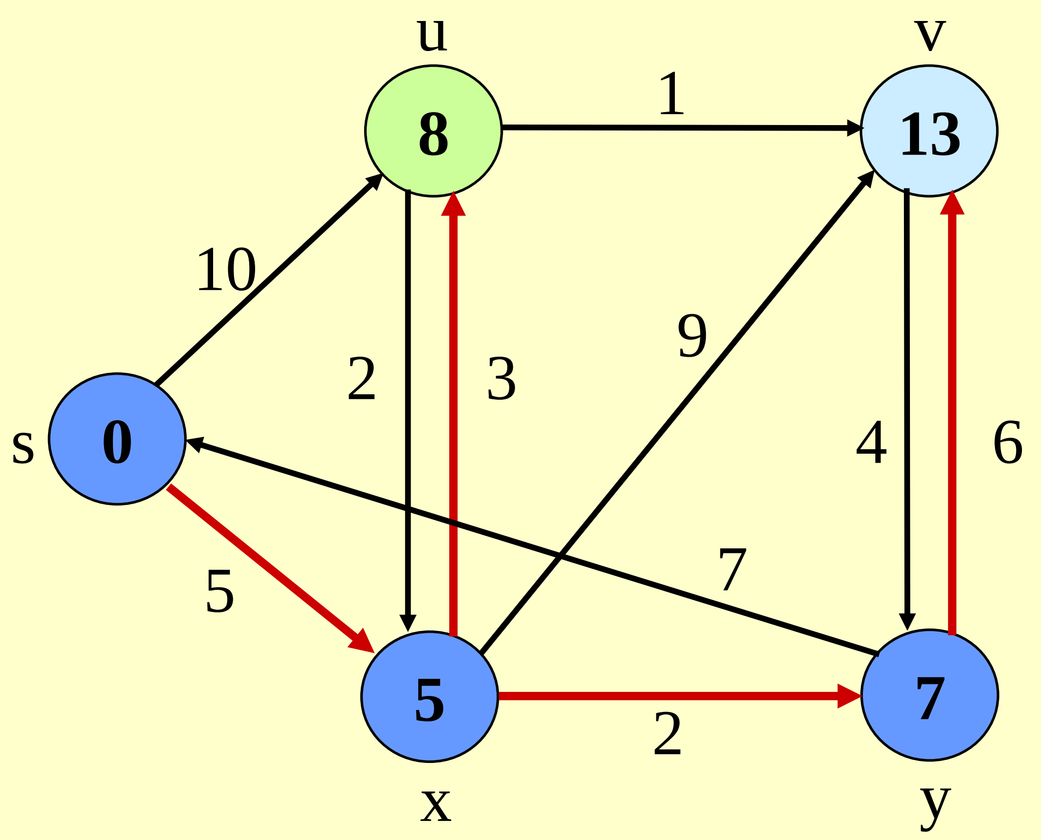
Example



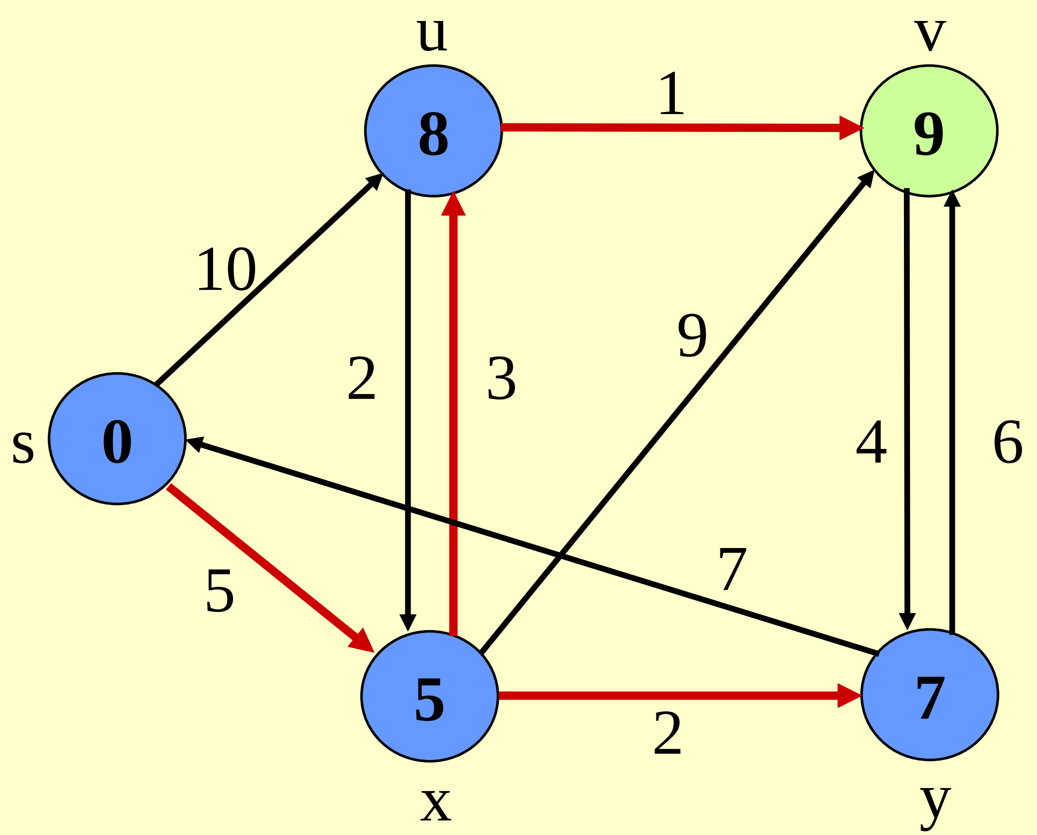
Example



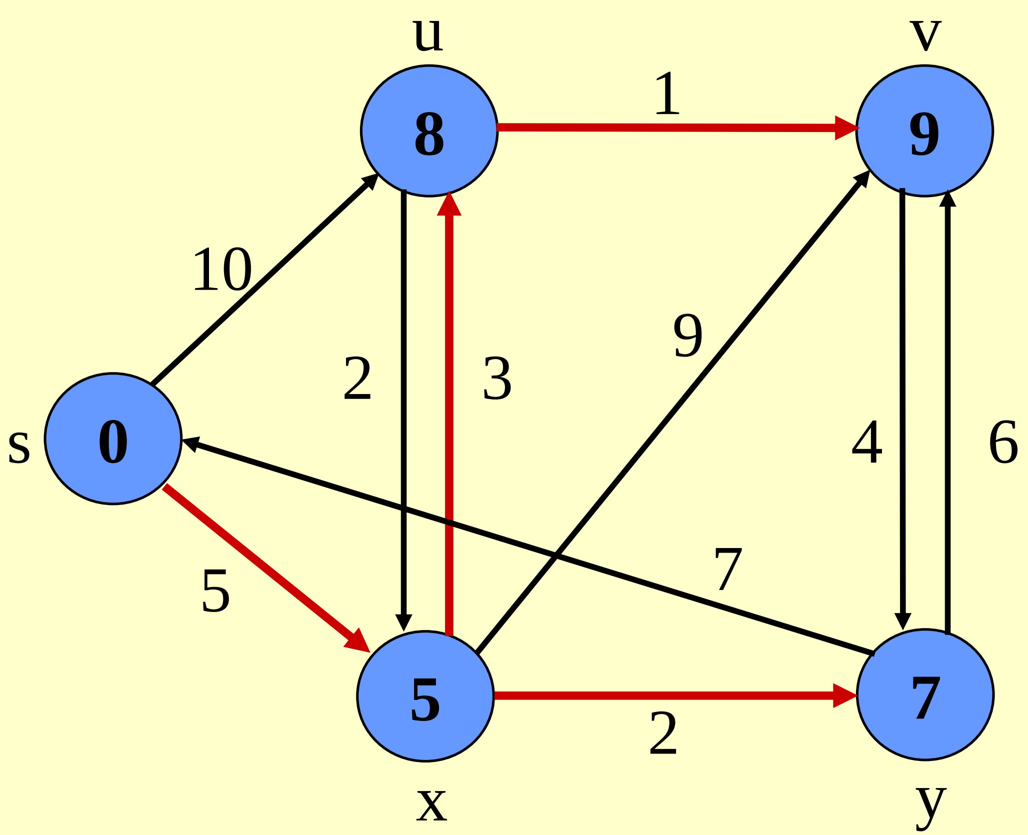
Example



Example



Example



Correctness

Theorem 24.6: Upon termination, d[u] = δ(s, u) for all u in V (assuming non-negative weights).

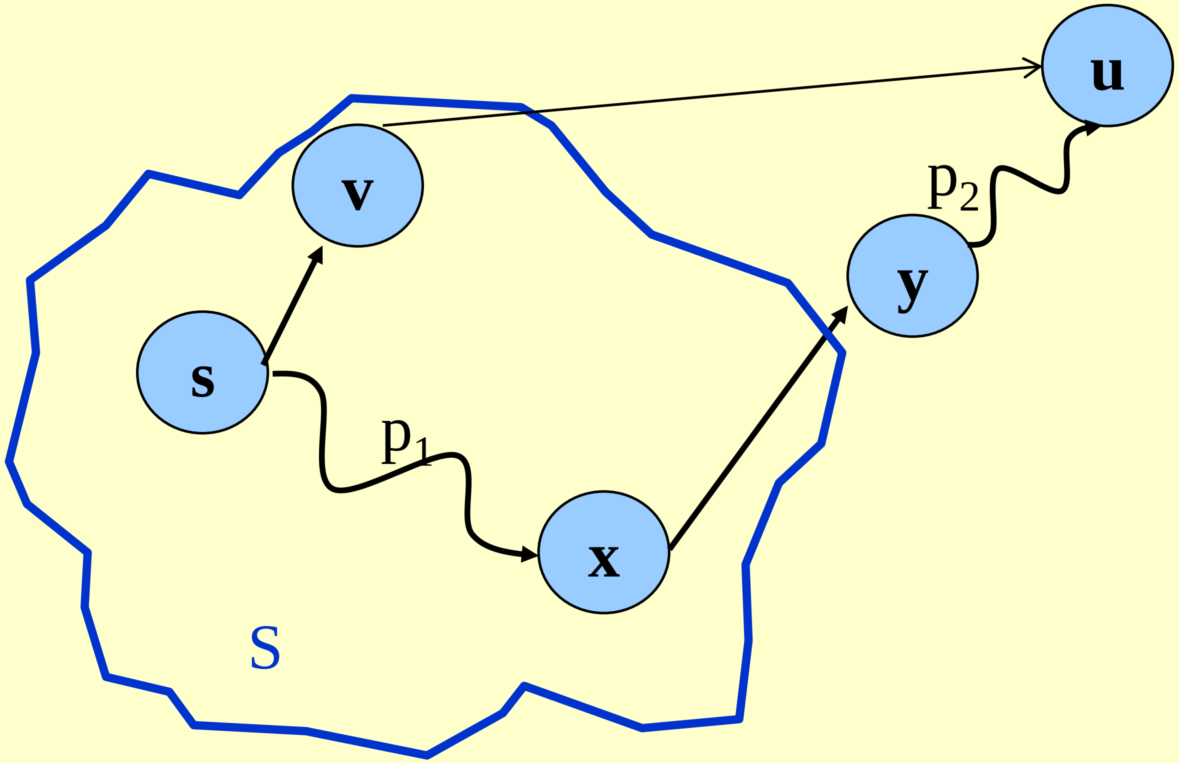
Proof:

By induction we would like to assume that till now all u in S have d[u]= δ(s, u) and we need to prove that a new node x entering S has d[x]=δ(s, x)

For this we need to prove two things
1. There can not be shorter path involving some y outside S such that
 δ(s, x)= δ(s, u)+ d[u,y] +d[y,x]
That is, the shortest path δ(s, x) will contain only nodes in S (as in Dijkstr
2. Prove that δ(s, u) will not require any recomputation of distance among nodes in S since if δ(s, u) changes once some y enters S last then I should be checking also d[s→y→u→x] and not just d[s->u→.x] for all u

Suppose shortest path contains y not in S

There exists a path from s to u, for otherwise d[u] = δ(s, u) = ∞
Suppose d[u] has the minimum value on it from some vertex v in S
but the actual shortest path from s to u looks like as below:



Proof (Continued)

Claim: d[y] = δ(s, y) when u is inserted into S.

We had d[x] = δ(s, x) when x was inserted into S.

Edge (x, y) was relaxed at that time.

By Lemma 24.14, this implies the claim.

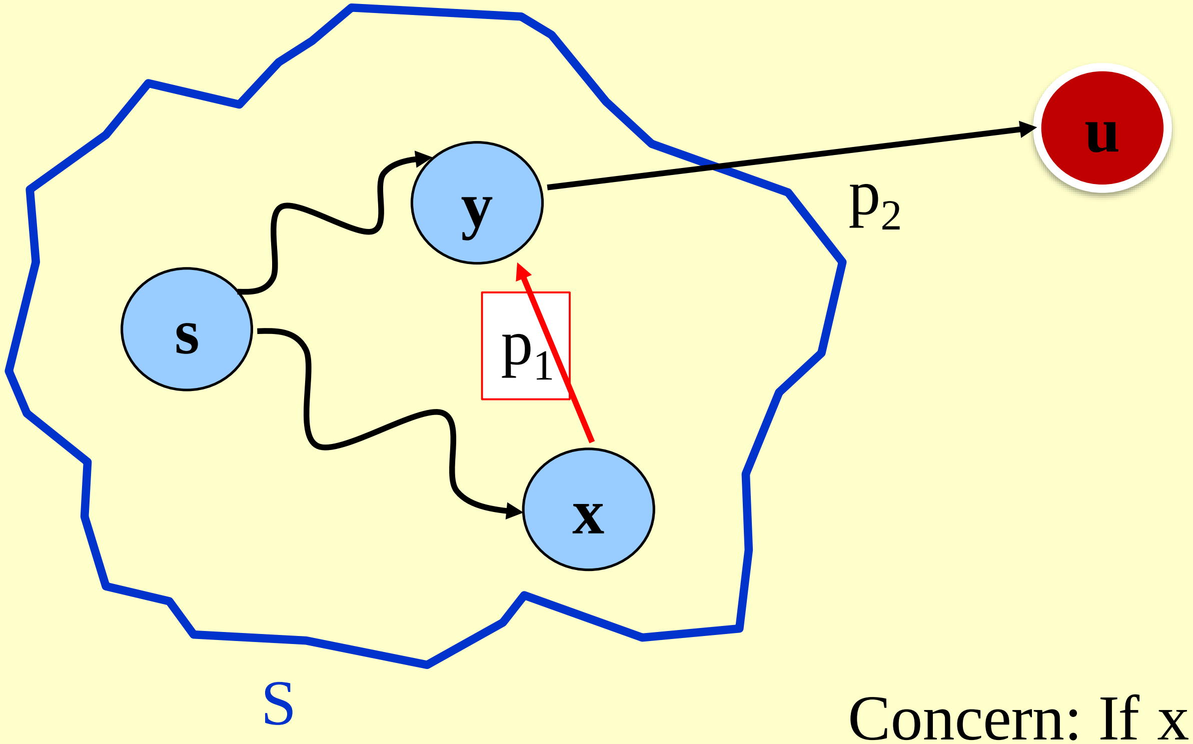
Now, we have: d[y] = δ(s, y) , by Claim.
≤ δ(s, u) , nonnegative edge weights.
≤ d[u] , by Lemma 24.11.

Because u was added to S before y, d[u] ≤ d[y].

Thus, d[y] = δ(s, y) = δ(s, u) = d[u].

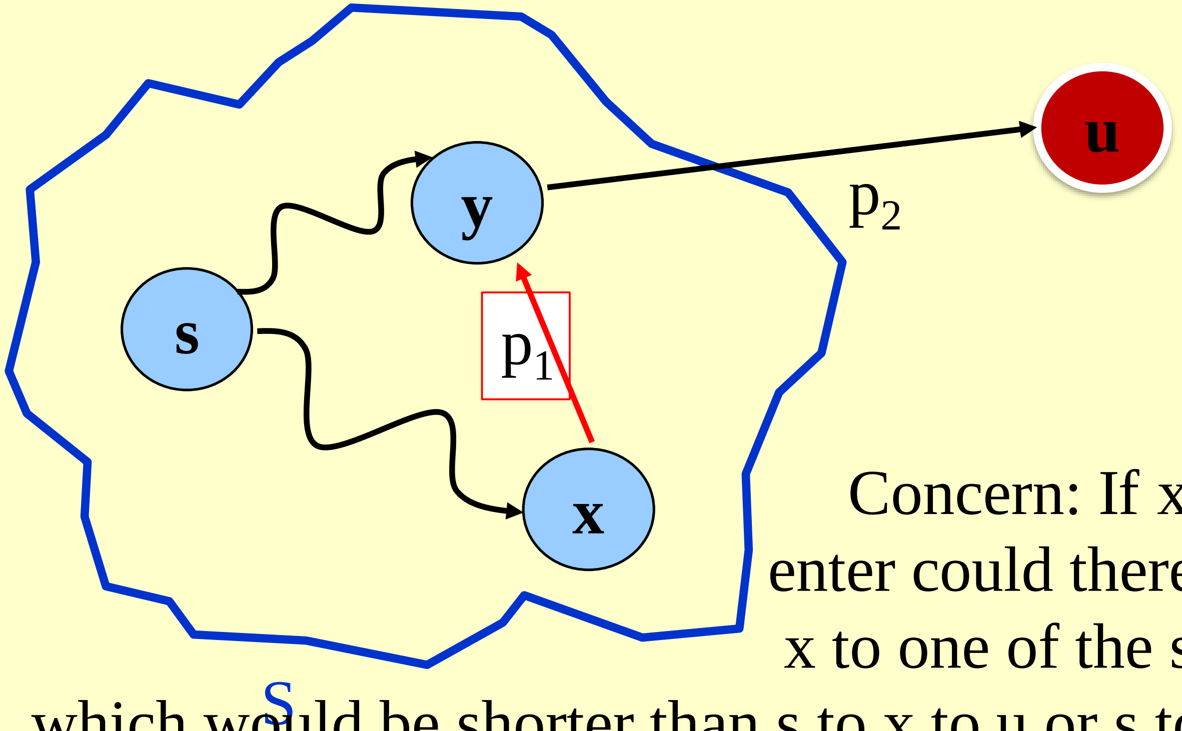
Contradiction.

Suppose shortest path from within S would change based
On last node inserted which is x



Concern: If x was the last node to
enter could there be a shorter path through
x to one of the seen nodes (say y) to u which
would be shorter than s to x to u or s to y to u ?

Suppose shortest path from within S would change based On last node inserted which is x



Concern: If x was the last node to enter could there be a shorter path through x to one of the seen nodes (say y) to u which would be shorter than s to x to u or s to y to u ? Not possible. By induction we have assumed that till now all nodes in S are at shortest distance and we want to prove that the new node that enters is at shortest distance. So since all nodes so far are at shortest distance and y entered before x so d[y] without x is the shortest distance and not s->x->y.

Complexity

Dijkstra Operations are :
Relax (total E edges that will be relaxed)
Update: A relaxation might lead to an update of node distance
PickMin: Each iteration pick node which is min distance away.

	Relax	Update	PickMin
Array	E	1 X E	V x V
Heap	E	log V X E	log V X V

Cost of update in Array is O(1) and in Heap is log(V) since you might need to delete and reheap so that the new value falls in the right place.
The number of updates is the number of relaxations = O(E)

Cost of PickMin in array is O(V) in array and O(log V) in heap
The number of PickMin operations is O(V)

Complexity

Running time is

$O(V^2)$ using linear array

$O((V + E) \lg V)$ using binary heap.

$O(V \lg V + E)$ using Fibonacci heap.

1. What happens when you have some negative edges in Dijkstra?

2. Can I modify Dijkstras algorithm to work for negative weights?

3. Suppose a graph G=(V,E) has small values of int
eger edge weights in the range 1 to W where W<<|V|.
Given this information we would like to compute the
shortest path in the graph from source to all vertices in
the graph. The complexity of Dijkstra discussed in class
using heaps is O((E+V)logV). Discuss the data structure
that can now be used to solve Dijkstras algorithm in O(E+WV)
which is O(E) given that W is a small constant.

What happens when you add negative edges in Dijkstra?

$$\begin{tikzcd}
S \arrow[r, "10"] \arrow[d, "5"] & A \arrow[r, "1"] & D \arrow[d, "-20"] \\
B \arrow[rr, "1"'] & & C
\end{tikzcd}$$

Distance from S to C gets marks as 6 from S->B->C

While shortest distance from S to C is -9 from S->A->D->C

Can I modify Dijkstra to work on negative weights?

No.

You can not also try to add a large positive constant to convert all negative weights to positive weights .