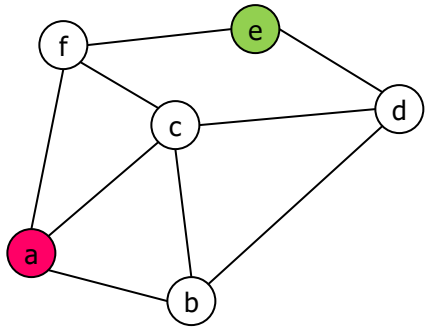
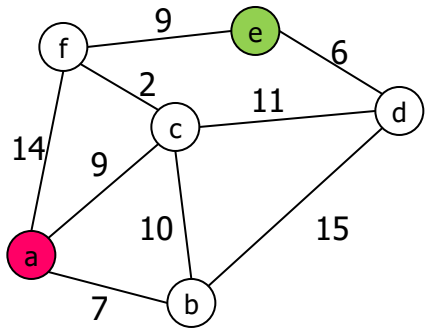
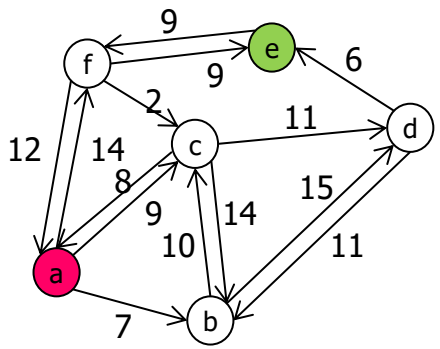
Shortest Paths

Alexander Lam

Shortest Paths



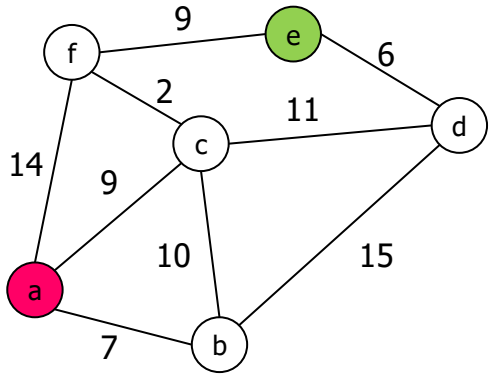




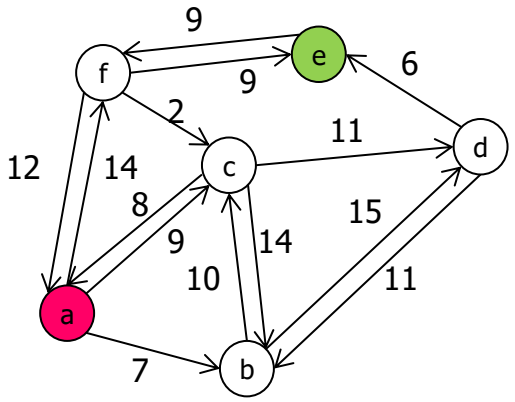
- Finding the shortest path from one node to another.
  - The starting node is often called source node in graph theory.
  - The target node is often called destination node.
  - The graphs may be weighted or unweighted
  - The graphs may be directed or undirected.
- There are other applications on a graph, e.g. breadth first search, depth first search, topological sort.

Shortest Path

- Can you find the shortest path from a to e?
  - Algorithm 1: Layman approach to find all possible paths first.
    - a → f → e — 23
    - a → c → f → e — 20
    - a → c → d → e — 26
    - a → b → c → d → e — 34
    - a → b → d → e — 28
    - a → c → b → d → e — 40
    - Any more?
    - a → b → c → f → e — 28
    - a → b → d → c → f → e — 44
    - Yet more?
    - a → f → c → d → e — 33
    - a → f → c → b → d → e — 47
  - Very slow for larger graphs!



Shortest Path



- Can you find the shortest path from a to e?
  - This is a directed graph.
  - There are fewer possible paths than previous one.
    - a → f → e
    - a → c → d → e
    - a → b → c → d → e
    - a → b → d → e
    - a → c → b → d → e
    - a → f → c → d → e
    - a → f → c → b → d → e
  - Still too difficult for a larger graph!
  - What is the maximum number of possible paths from a to e in this small graph with 6 nodes?

Technically Infinity, but if we don't allow cycles then 65.

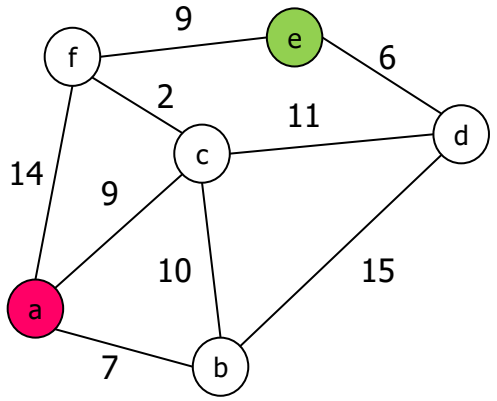
Improvements?

- There are so many possible paths.
- Algorithm 2: search for paths starting with fewer steps.
  - a → f → e
  - a → b → d → e
  - a → c → d → e
  - a → c → f → e
  - Stop here?
  - a → b → c → d → e
  - a → b → c → f → e
  - a → c → b → d → e
  - a → f → c → d → e
  - Stop here?
  - a → b → d → c → f → e
  - a → f → c → b → d → e

- 23
- 28
- 26
- 20

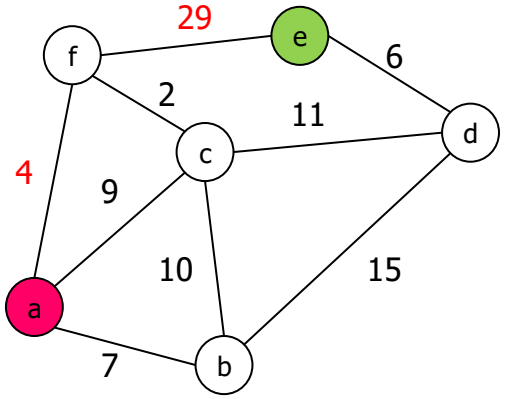
- 34
- 28
- 40
- 33

- 44
- 47



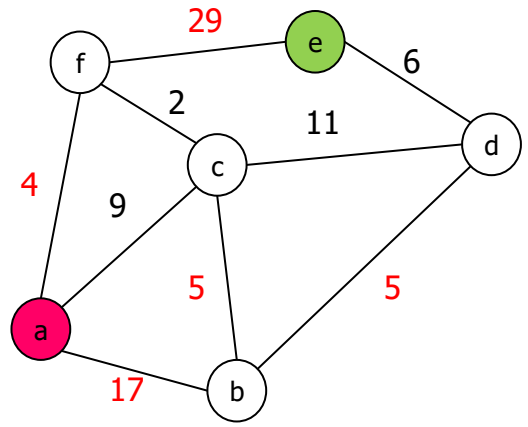
Improvements?

- Consider this graph.
- Search for paths starting with fewer steps.
  - $a \to f \to e$ — 33
  - $a \to b \to d \to e$ — 28
  - $a \to c \to d \to e$ — 26
  - $a \to c \to f \to e$ — 40
  - Stop here?
  - $a \to b \to c \to d \to e$ — 34
  - $a \to b \to c \to f \to e$ — 48
  - $a \to c \to b \to d \to e$ — 40
  - $a \to f \to c \to d \to e$ — 23
  - Stop here?
  - $a \to b \to d \to c \to f \to e$ — 64
  - $a \to f \to c \to b \to d \to e$ — 37

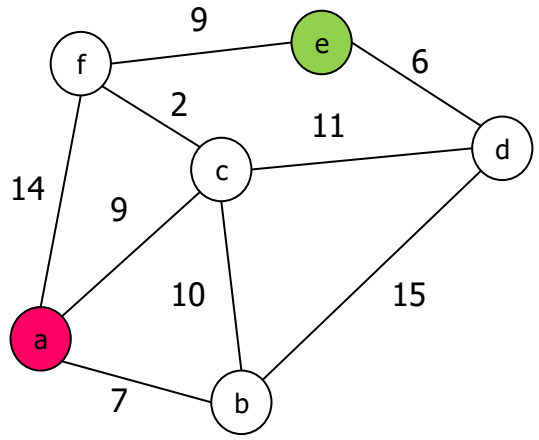


Improvements?

- Consider this graph.
- Search for paths starting with fewer steps.
  - $a \to f \to e$ — 33
  - $a \to b \to d \to e$ — 28
  - $a \to c \to d \to e$ — 26
  - $a \to c \to f \to e$ — 40
  - Stop here?
  - $a \to b \to c \to d \to e$ — 39
  - $a \to b \to c \to f \to e$ — 53
  - $a \to c \to b \to d \to e$ — 25
  - $a \to f \to c \to d \to e$ — 23
  - Stop here?
  - $a \to b \to d \to c \to f \to e$ — 64
  - $a \to f \to c \to b \to d \to e$ — 22



Improvements?

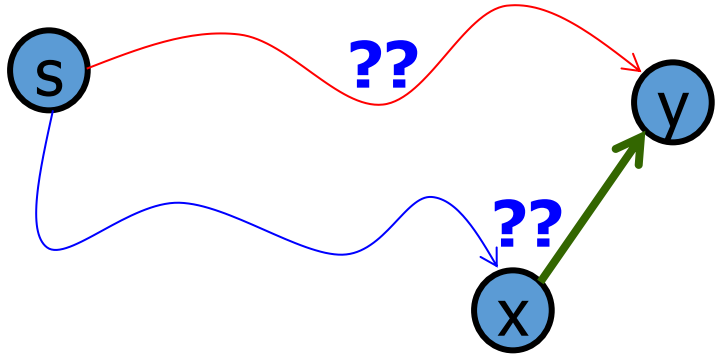


- We may miss the correct answer if we don't try all possible paths.
  - We cannot afford to try all possible paths in larger graph.
  - We need a more clever and systematic approach.
- Possible approach:
  - Look at edges and use them to improve on existing paths.
  - An edge is said to lead to improvement, if passing through it would lead to a better path.
  - Example:
    - Going from a to c directly, the distance is 9.
    - Going from a to f directly, the distance is 14.
    - Going from a to f via c, the distance improves to 11 $<$ 14.
  - We will try to implement this idea.

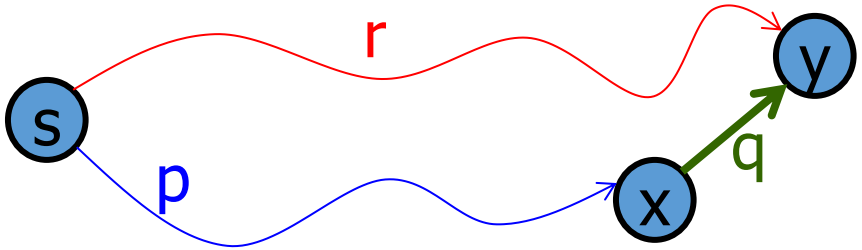
Exploring Edges

- Input:
  - A graph (N, E) and distance information D(x,y) on edges
  - A source node s and a destination node d
- Output:
  - Shortest distance from s to d
- Algorithm 3:

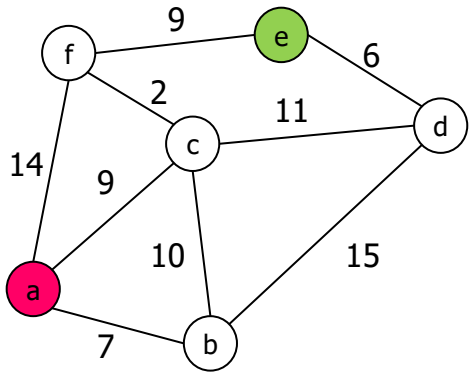
```
initialize best[n] = infinity for all n except s, best[s] = 0
for each edge (x,y) in E do
    if best[x]+D(x,y) < best[y] then
        # it is better to go to y via x
        best[y] = best[x]+D(x,y)
    # for undirected graph, need to check also
    # best[y]+D(y,x) < best[x] to improve best[x]
```



Exploring Edges

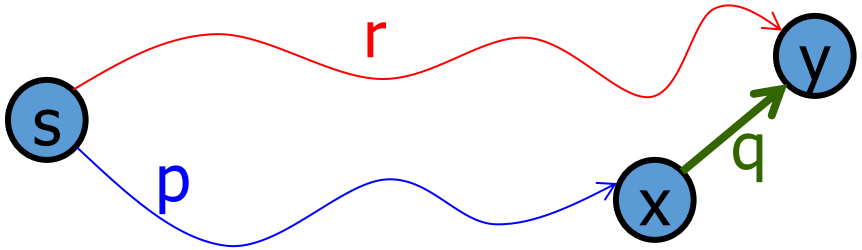


p+q<r?

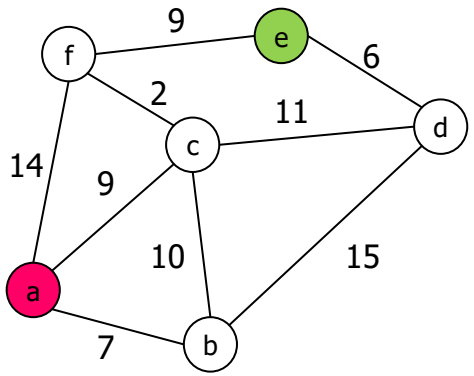


- Initialize best distances.
  - best[a] = 0, best[b] = best[c] = best[d] = best[e] = best[f] = ∞.
- Pick the edges, let us assume alphabetical order of those 9 edges.
  - E = {ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9}.
  - With ab, improve best[b]: best[a] = 0, best[b] = 7, best[c] = ∞, best[d] = ∞, best[e] = ∞, best[f] = ∞.
  - With ac, improve best[c]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = ∞, best[e] = ∞, best[f] = ∞.
  - With af, improve best[f]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = ∞, best[e] = ∞, best[f] = 14.
  - With bc, no improvement to best[b] and best[c].

Exploring Edges

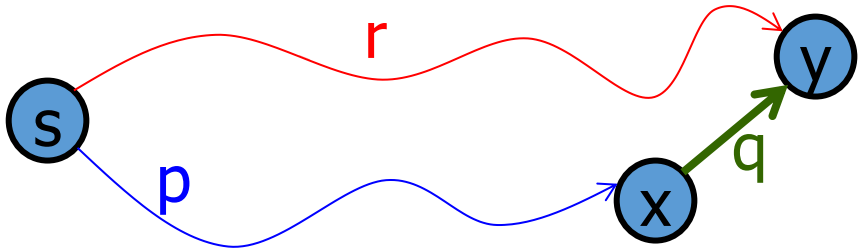


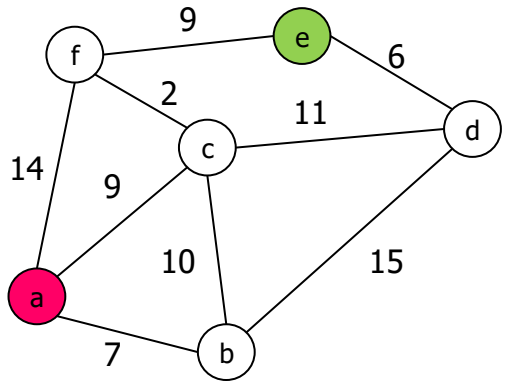
p+q<r?



- Pick the edges, let us assume alphabetical order of those 9 edges.
  - E = {ab=7, ac=9, af=14, bc=10, bd=15, cd=11, cf=2, de=6, ef=9}.
  - From last slide, we have best[a] = 0, best[b] = 7, best[c] = 9, best[d] = ∞, best[e] = ∞, best[f] = 14.
  - With bd, improve best[d]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 22, best[e] = ∞, best[f] = 14, no improvement to best[b].
  - With cd, improve best[d]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = ∞, best[f] = 14, no improvement to best[c].
  - With cf, improve best[f]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = ∞, best[f] = 11, no improvement to best[c].
  - With de, improve best[e]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = 26, best[f] = 11, no improvement to best[d].
  - With ef, improve best[e]: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = 20, best[f] = 11, no improvement to best[f].

Exploring Edges





- Is it sufficient to explore with just one round?
  - We assumed alphabetical ordering of the edges to be processed in this example.

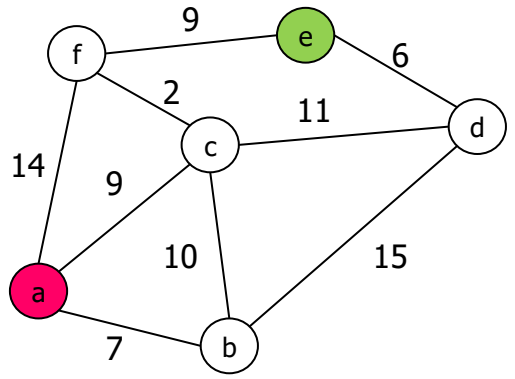
- Repeat our algorithm with a different edge ordering.
  - Initialize best distances.
    - best[a] = 0, best[b] = best[c] = best[d] = best[e] = best[f] = ∞.
  - Pick the 9 edges in this order.
    - E = {ef=9, de=6, cf=2, cd=11, bd=15, bc=10, af=14, ac=9, ab=7}.
  - What will be your result?
    - best[a] = 0, best[b] = 7, best[c] = 9, best[d] = ∞, best[e] = ∞, best[f] = 14.

Exploring Edges

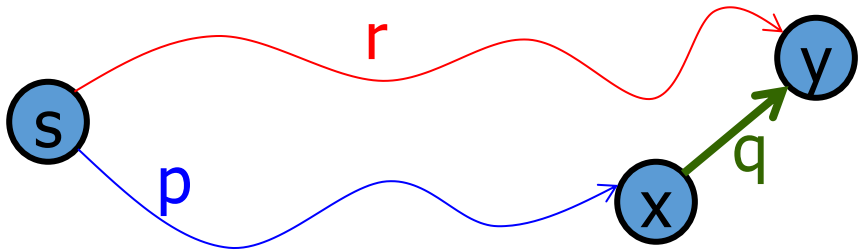
- It is not sufficient to just execute one round.
- Algorithm 4:

```
initialize best[n] to infinity for all n except s, best[s] = 0
repeat
   for each edge (x,y) in E do
      if best[x]+D(x,y) < best[y] then
                  best[y] = best[x]+D(x,y)
      # for undirected graph, need to check also
      # best[y]+D(y,x) < best[x] to improve best[x]
until we are satisfied with the answer
```

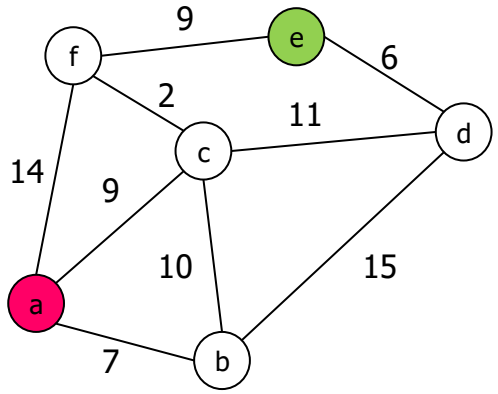
- How many rounds would be needed?



Exploring Edges

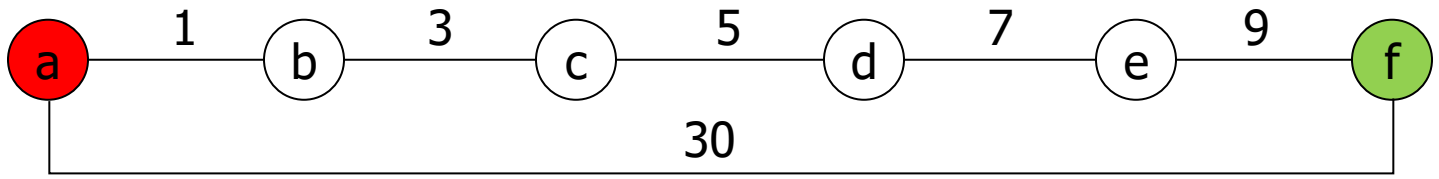


p+q<r?



- Taking the edges in this order.
  - E = {ef=9, de=6, cf=2, cd=11, bd=15, bc=10, af=14, ac=9, ab=7}.
  - Round 1: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = ∞, best[e] = ∞, best[f] = 14.
  - Round 2: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = 23, best[f] = 11.
  - Round 3: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = 20, best[f] = 11.
  - Round 4: best[a] = 0, best[b] = 7, best[c] = 9, best[d] = 20, best[e] = 20, best[f] = 11. (no change observed)
  - Do we need to go for round 5?
- Conclusion: algorithm 4 works, but could be slow.

Exploring Edges



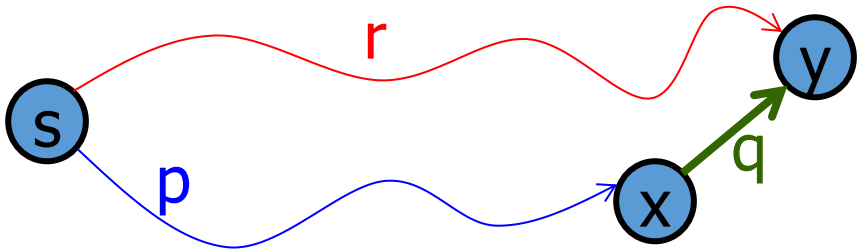
- In a worst case, you need 5 rounds from a to f!
  - E = {af=30, ef=9, de=7, cd=5, bc=3, ab=1}.
  - Round 1: best[a] = 0, best[b] = 1, best[c] = 51, best[d] = 46, best[e] = 39, best[f] = 30.
  - Round 2: best[a] = 0, best[b] = 1, best[c] = 4, best[d] = 46, best[e] = 39, best[f] = 30.
  - Round 3: best[a] = 0, best[b] = 1, best[c] = 4, best[d] = 9, best[e] = 39, best[f] = 30.
  - Round 4: best[a] = 0, best[b] = 1, best[c] = 4, best[d] = 9, best[e] = 16, best[f] = 30.
  - Round 5: best[a] = 0, best[b] = 1, best[c] = 4, best[d] = 9, best[e] = 16, best[f] = 25.
  - Round 6: best[a] = 0, best[b] = 1, best[c] = 4, best[d] = 9, best[e] = 16, best[f] = 25 (proven no change).

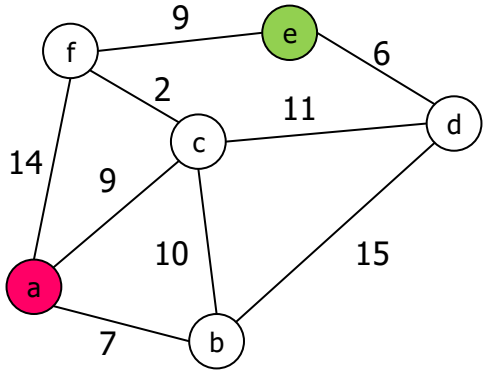
Exploring Nodes

- Algorithm 4 is systematic. It works but is a bit slow.
  - We need a more clever approach.
    - We do not want to try all edges repeatedly.
  - Observations:
    - If you find a proven good path, there is no need to look again.
    - From the starting point, simply look around and explore the neighborhood (like searching in a ring).
    - Continue exploration until the destination is reached.
    - During exploration, remember the best path so far to any intermediate location.
    - The hard part is to know whether we have found a proven good path.
  - Let us work out from an example.

Exploring Nodes

Best so far:
a to b at distance of 7
a to c at distance of 9
a to f at distance of 14

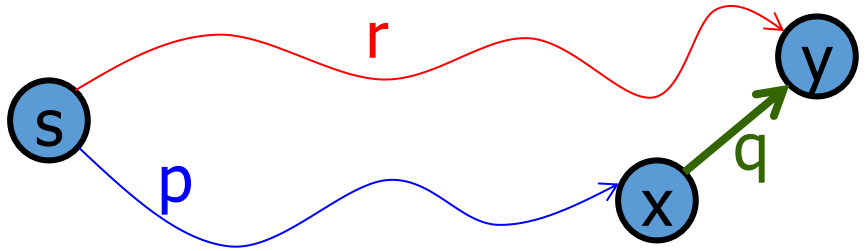


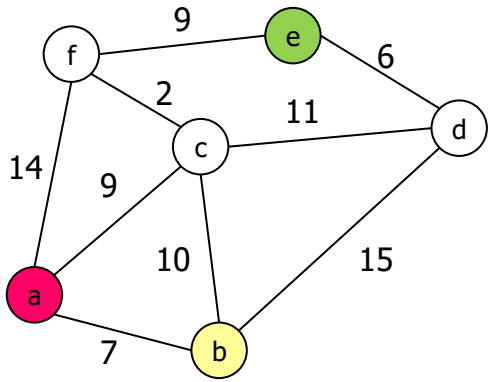


- Can you find the shortest path from a to e?
  - Start from a, where can we go?
    - a to b at distance of 7
    - a to c at distance of 9
    - a to f at distance of 14
    - We say that this step explores node a.
  - Can we go to b, c, f with shorter distance?
    - Obviously, from a to b, that is already shortest.
    - It is unclear whether it is shorter to go from a to c via b, or a to f via b/c.
  - Therefore, we are done with b, but not c and f.

Exploring Nodes

Best so far:
a to b at distance of 7
a to c at distance of 9
a to f at distance of 14
a to d at distance of 22 (via b)

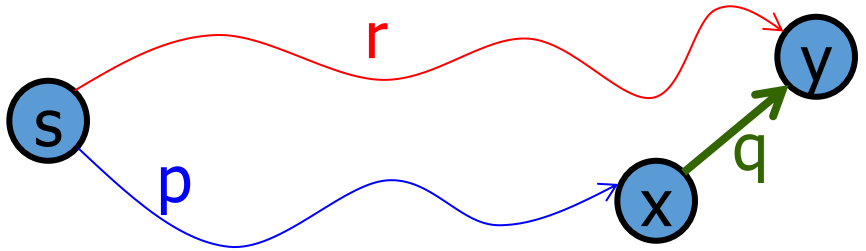


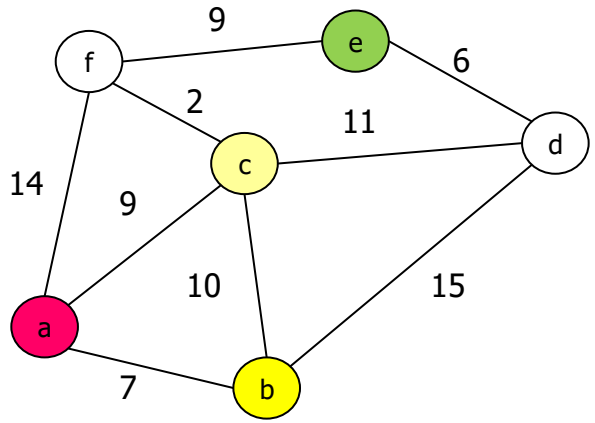


- Finding the shortest path from a to e.
  - Node b is the current best from a, so we try b with a short-sighted (or greedy) mindset.
  - Start from a and via b, where can we go?
    - a to b to c at distance of 17 $>$ 9 (i.e. a to c directly)
    - a to b to d at distance of 22 (new place)
    - We say that we explored node b.
  - Can we go to c, d, f with shorter distance?
    - Obviously, from a to c, that is already shortest.
    - It is unclear whether it is shorter to go from a to d or a to f indirectly.
  - Therefore, we are done with c, but not d and f.

Exploring Edges

Best so far:
a to b at distance of 7
a to c at distance of 9
a to f at distance of 11 (via c)
a to d at distance of 20 (via c)

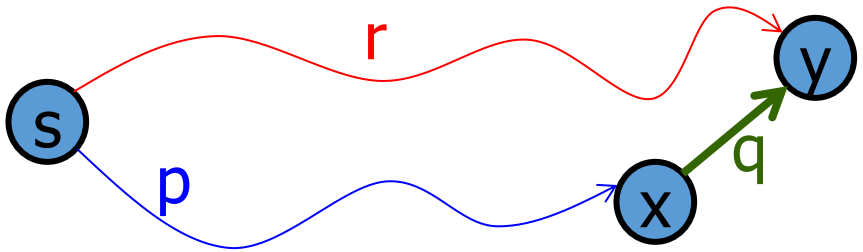


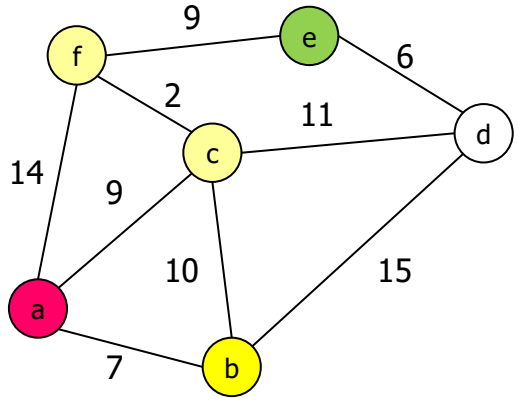


- Finding the shortest path from a to e.
  - Start from a and via b, c, where can we go?
    - a to c to f at distance of 11 $<$ 14 (i.e. a to f directly)
    - a to c to d at distance of 20 $<$ 22 (i.e. a to b to d)
    - We explored node c.
  - Can we go to d, f with shorter distance?
    - Obviously, from a to f (via c), that is already shortest.
    - It is unclear whether it is shorter to go from a to d indirectly (maybe via f).
  - Therefore, we are done with f, but not d.

Exploring Nodes

Best so far:
a to b at distance of 7
a to c at distance of 9
a to f at distance of 11 (via c)
a to d at distance of 20 (via c)
a to e at distance of 20 (via c and f)

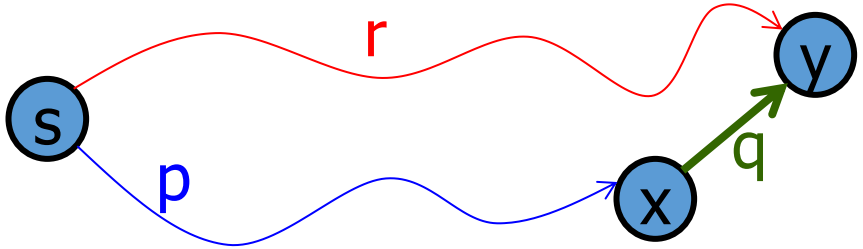


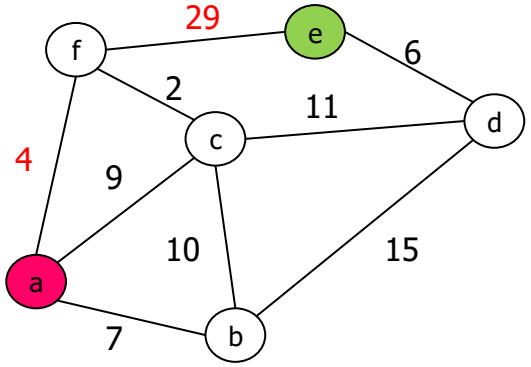


- Finding the shortest path from a to e.
  - Start from a and via b, c, f, explore f.
    - a to c to f to e at distance of 20 (new place)
  - Can we go to d, e with shorter distance?
    - Obviously, from a to d (via c), that is shortest (20).
    - Similarly, from a to e (via c and f) is shortest (20).
  - Therefore, we are done with both d and e.
    - a to c to f to e is shortest, with a distance of 20.
  - We are to explore either d or e next.
  - Let us explore e.
    - Once we explore destination e, we could stop and return the answer 20.

Exploring Nodes

Best so far:
a to f at distance of 4
a to b at distance of 7
a to c at distance of 9

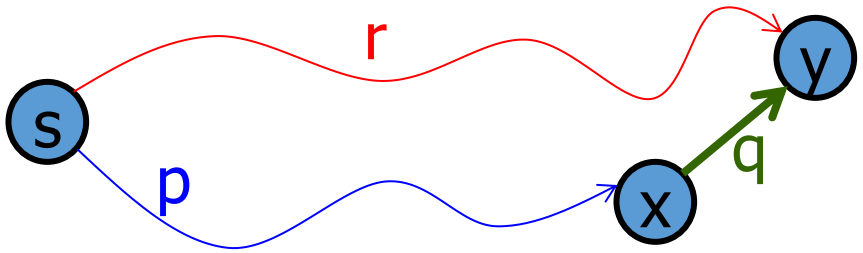


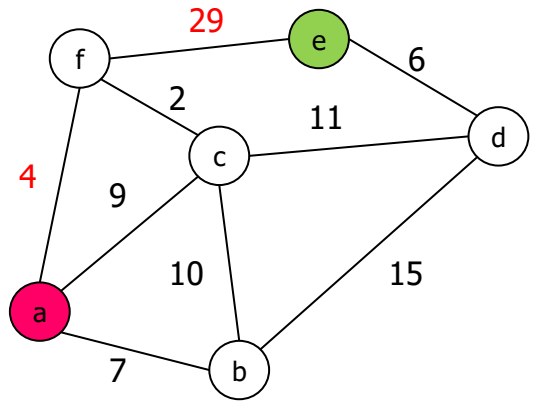


- There is a temptation that when e is reached (not explored), we can stop.
  - Start from a, where can we go?
    - a to b at distance of 7
    - a to c at distance of 9
    - a to f at distance of 4
  - Can we go to b, c, f with shorter distance?
    - Obviously, from a to f, that is already shortest.
    - It is unclear whether it is shorter to go from a to b via c/f, or a to c via b/f.
  - Therefore, we are done with f, but not b and c.

Exploring Nodes

Best so far:
a to f at distance of 4
a to c at distance of 6 (via f)
a to b at distance of 7
a to e at distance of 33 (via f)

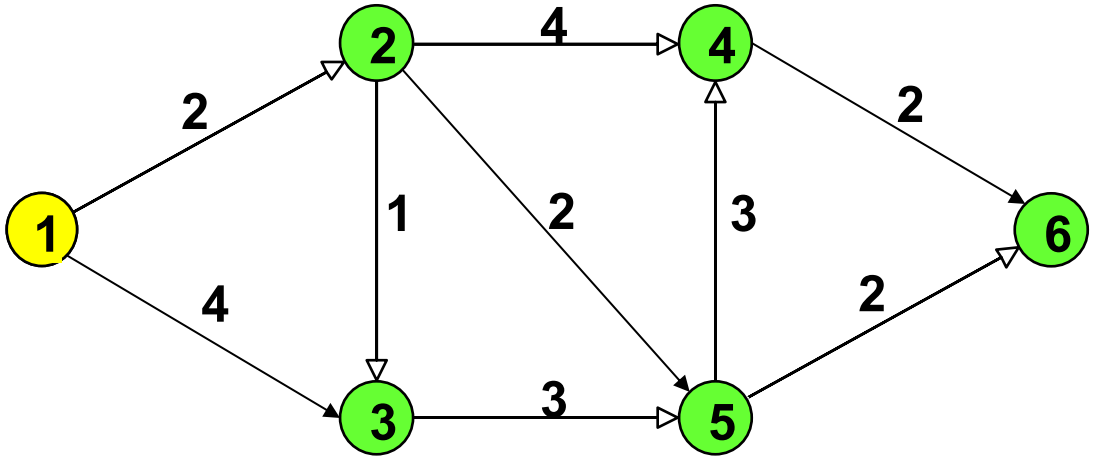




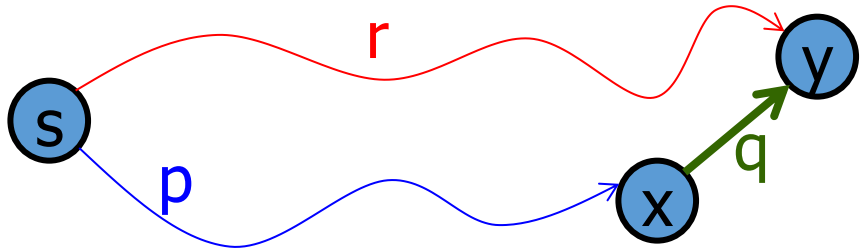
- Finding the shortest path from a to e.
  - Start from a and via f, we explore f.
    - a to f to c at distance of $6 < 9$ (i.e. better than a to c)
    - a to f to e at distance of 33 (new place)
  - We reach destination e, and are tempted to stop.
    - The shortest path is a $\rightarrow$ f $\rightarrow$ e with distance 33!
    - Obviously, if you look more carefully, a $\rightarrow$ f $\rightarrow$ c $\rightarrow$ d $\rightarrow$ e is a better path with distance 23.
  - Back to our idea, we are only done with f, but not b and c and the newly found e.
    - Next node to explore should be c.
    - Be patient until we are ready to explore e.

Example 2

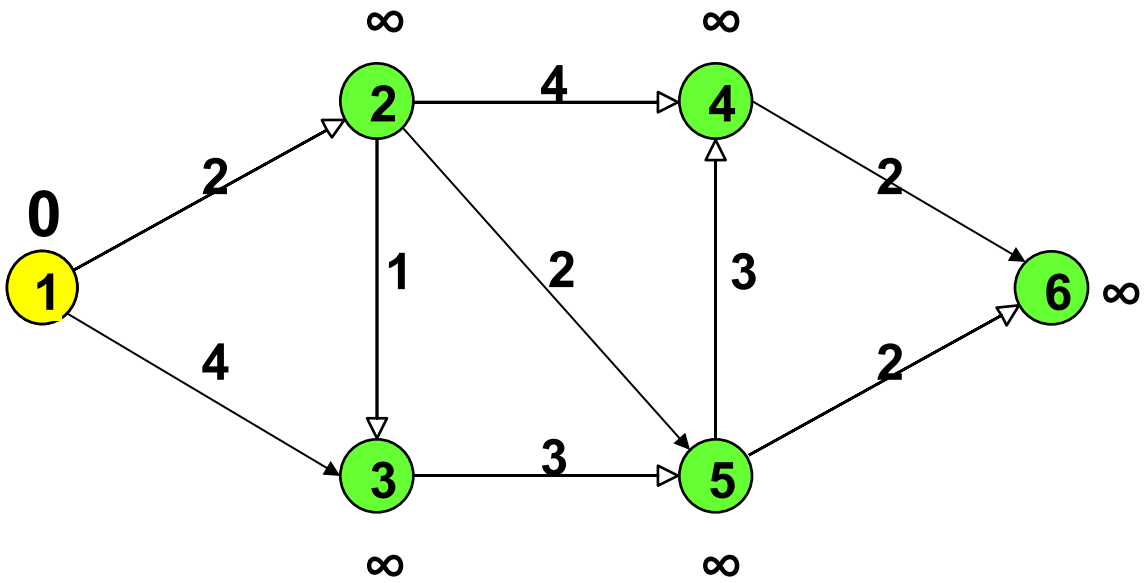
- Another example going from 1 to 6.
  - How many possible paths?
    - 1 2 4 6
    - 1 3 5 6
    - 1 2 5 6
    - 1 2 3 5 6
    - 1 3 5 4 6
    - 1 2 5 4 6
    - 1 2 3 5 4 6



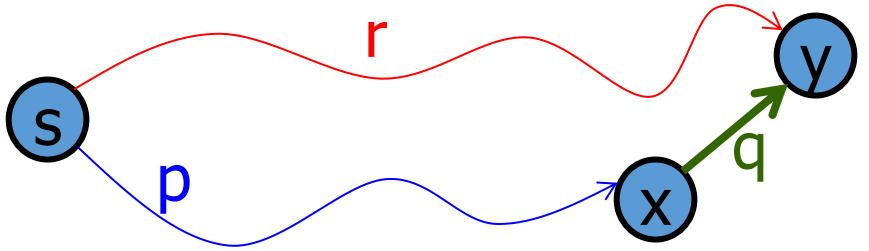
Example 2



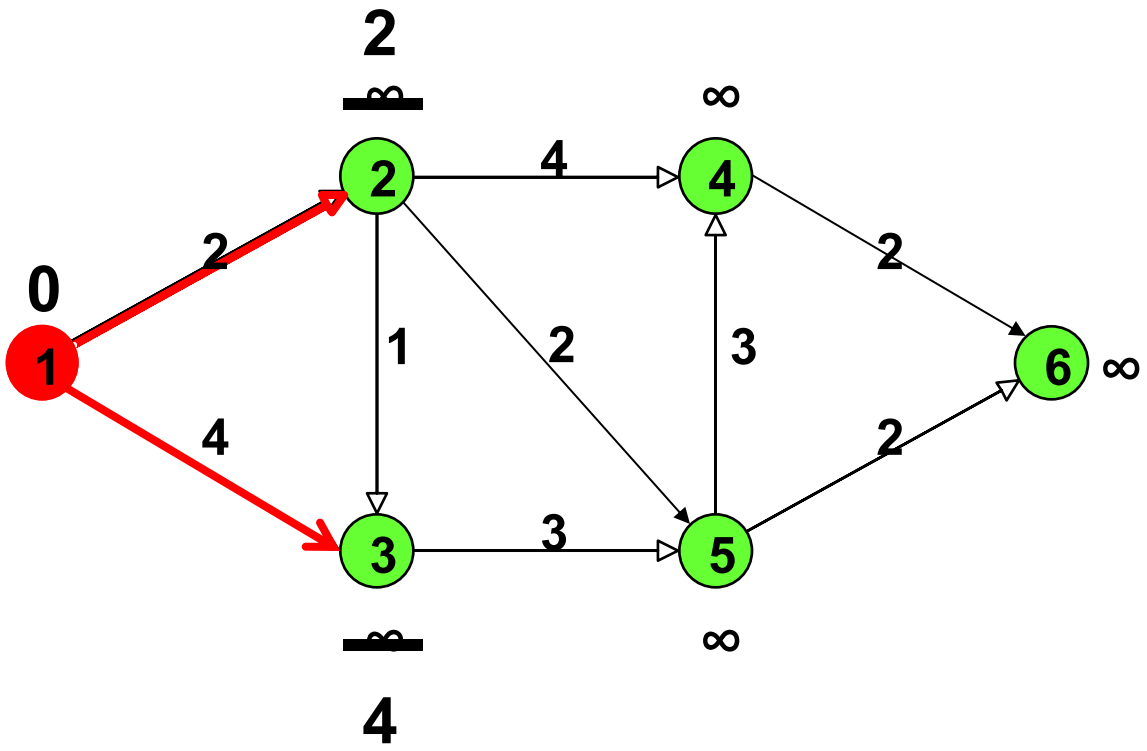
- Best distance so far are stored in nodes.
  - 0 for source and infinity for others respectively.



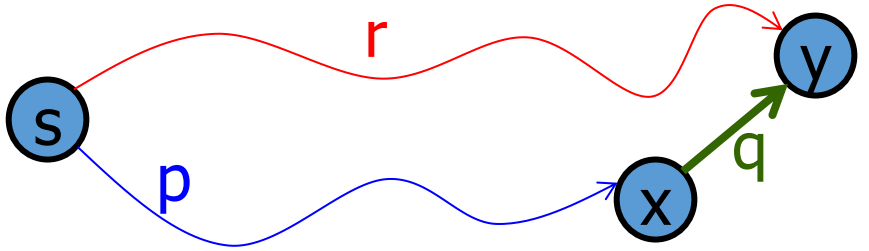
Example 2



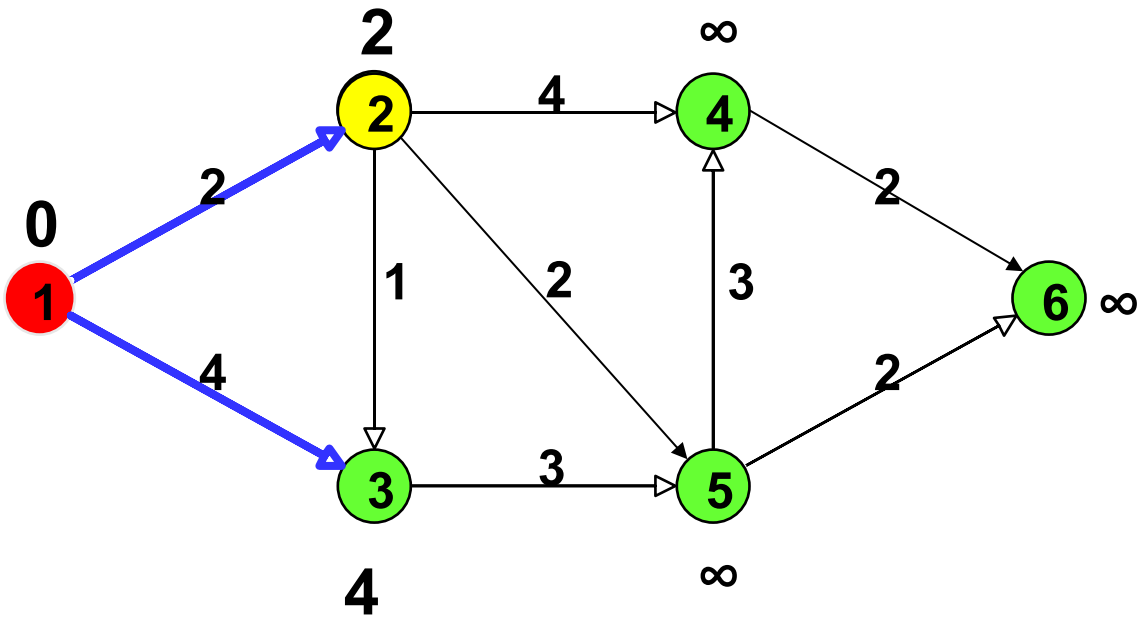
- Update distance of 2 and 3 coming from 1.



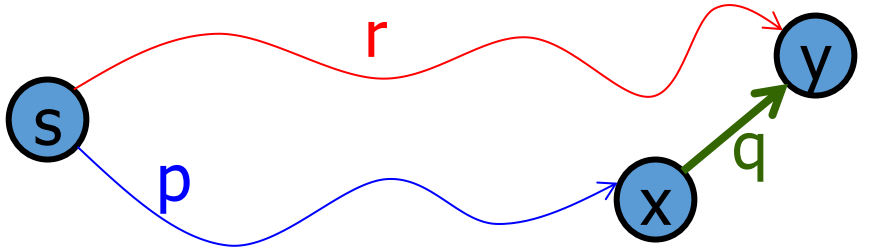
Example 2



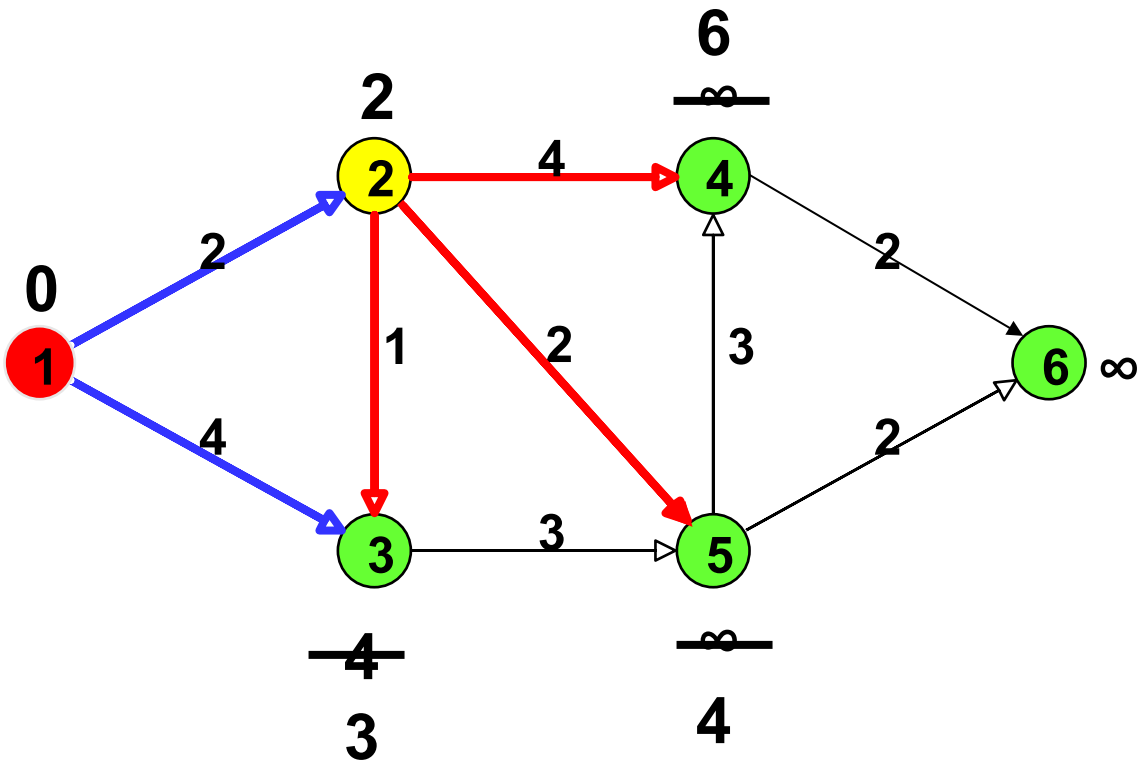
- Select next node: 2



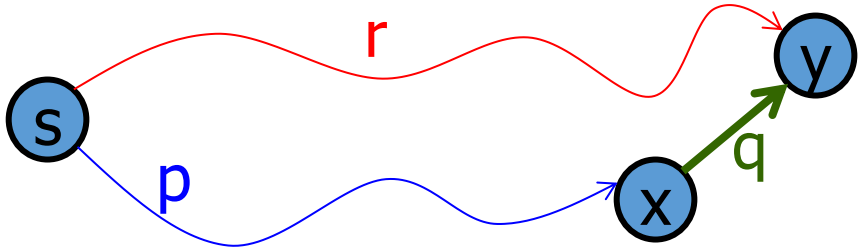
Example 2



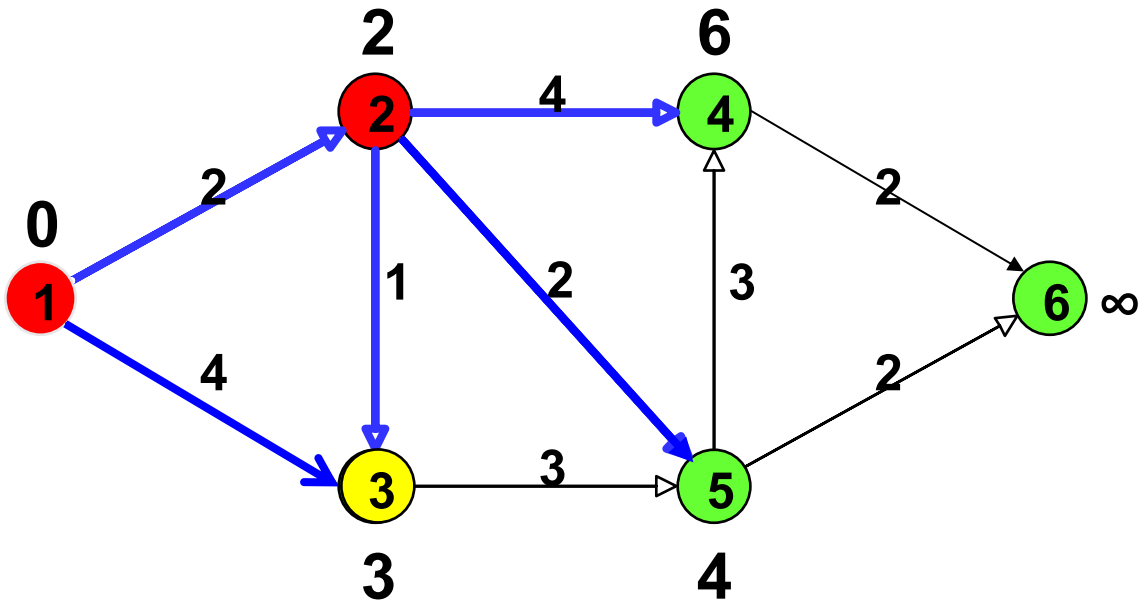
- Update distance of 3, 4 and 5 coming from 2.



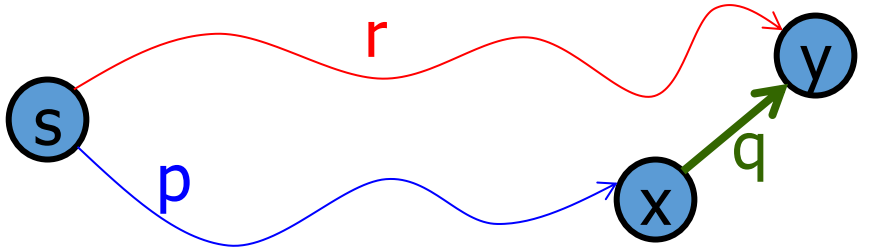
Example 2



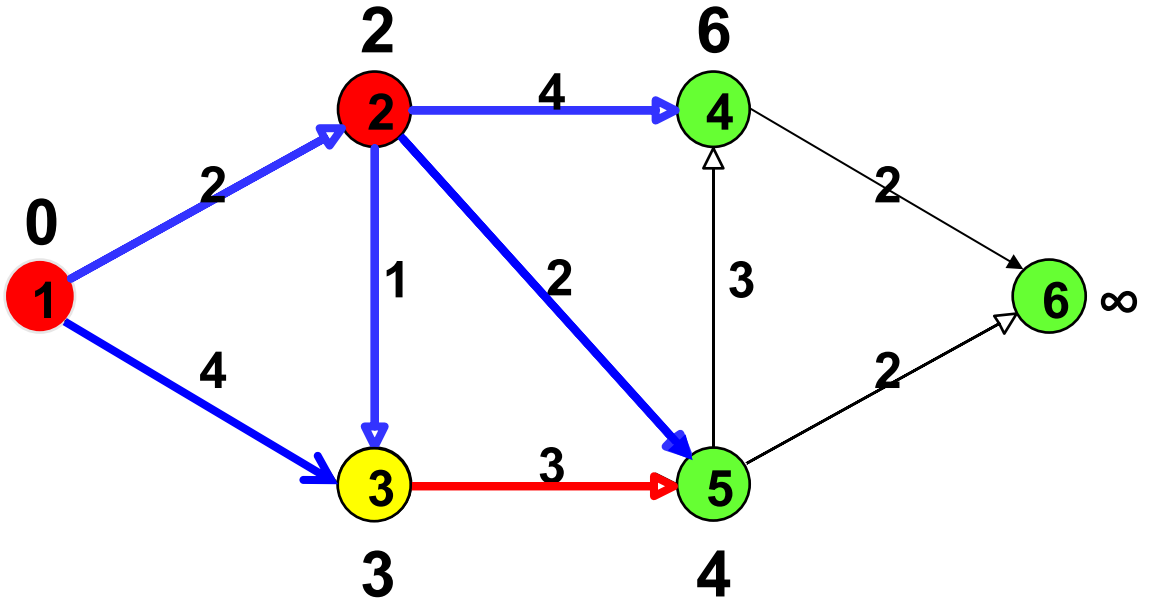
- Select next node: 3



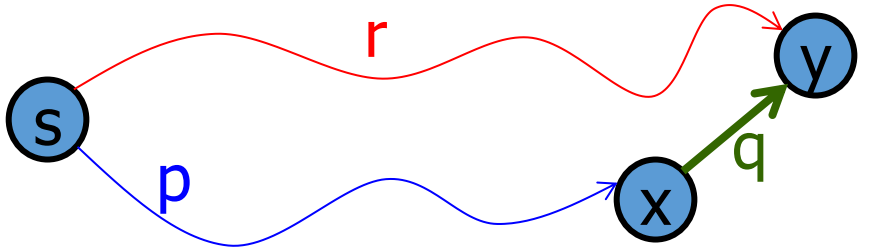
Example 2



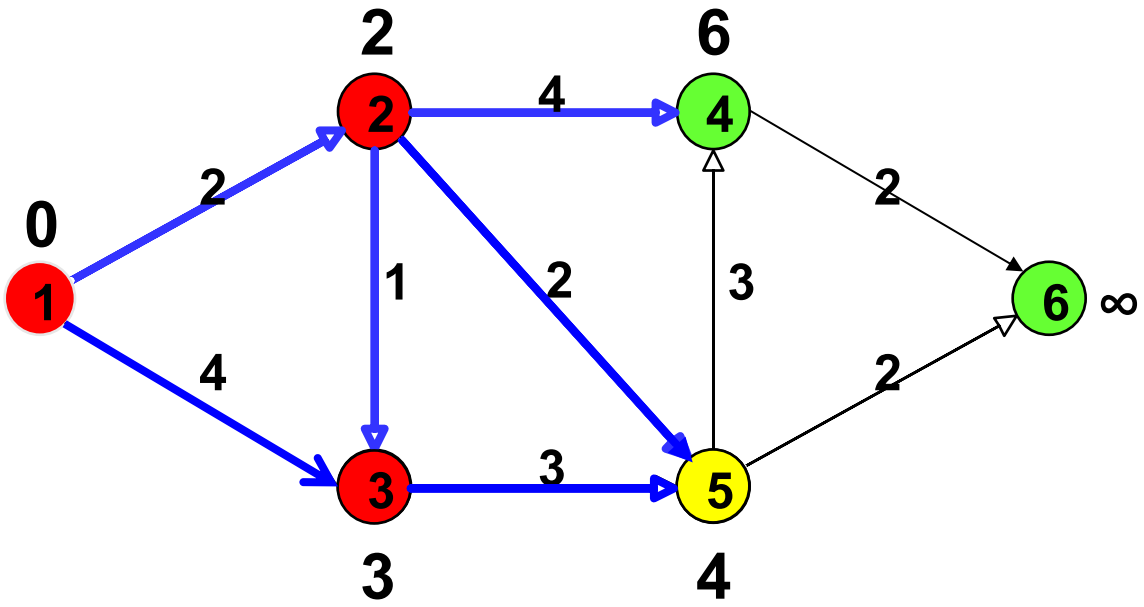
- Update distance of 5 coming from 3.
  - Except it isn't updated



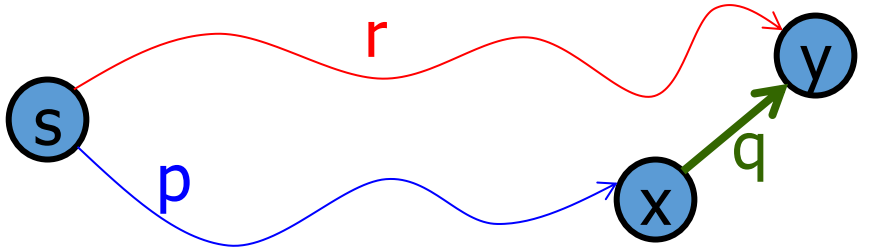
Example 2



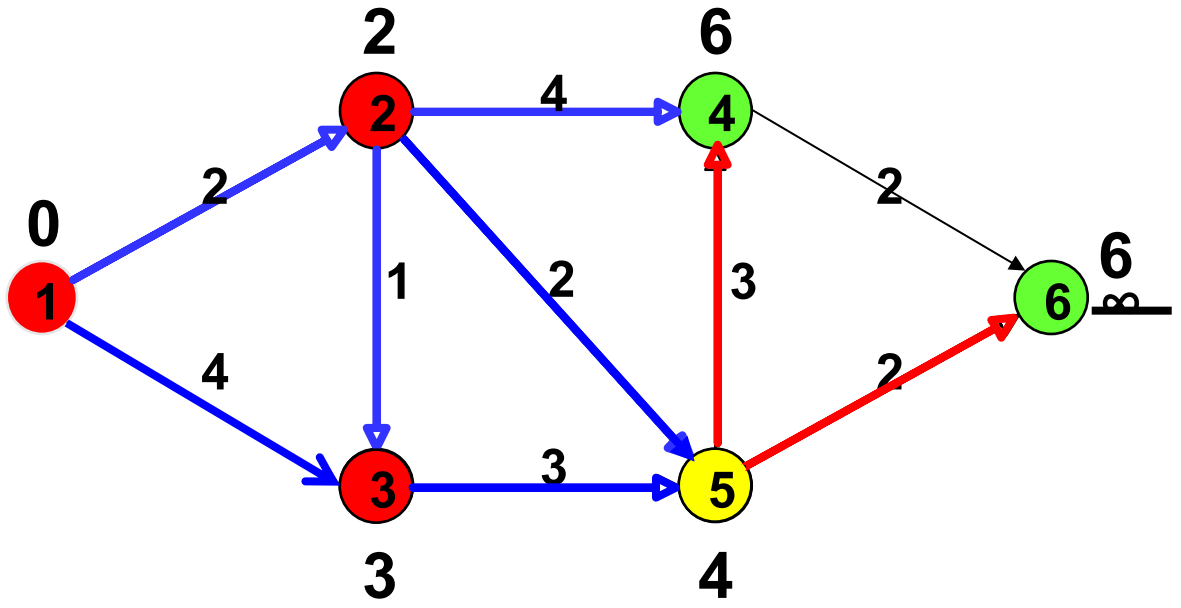
- Select next node: 5



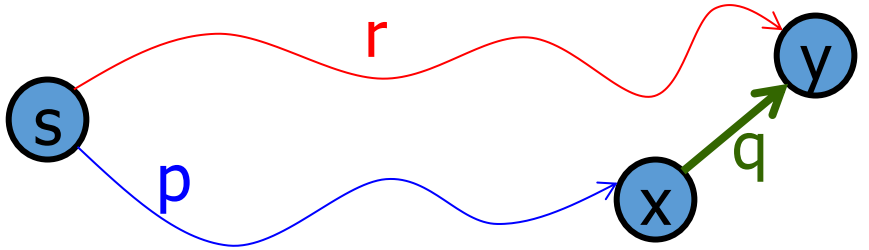
Example 2



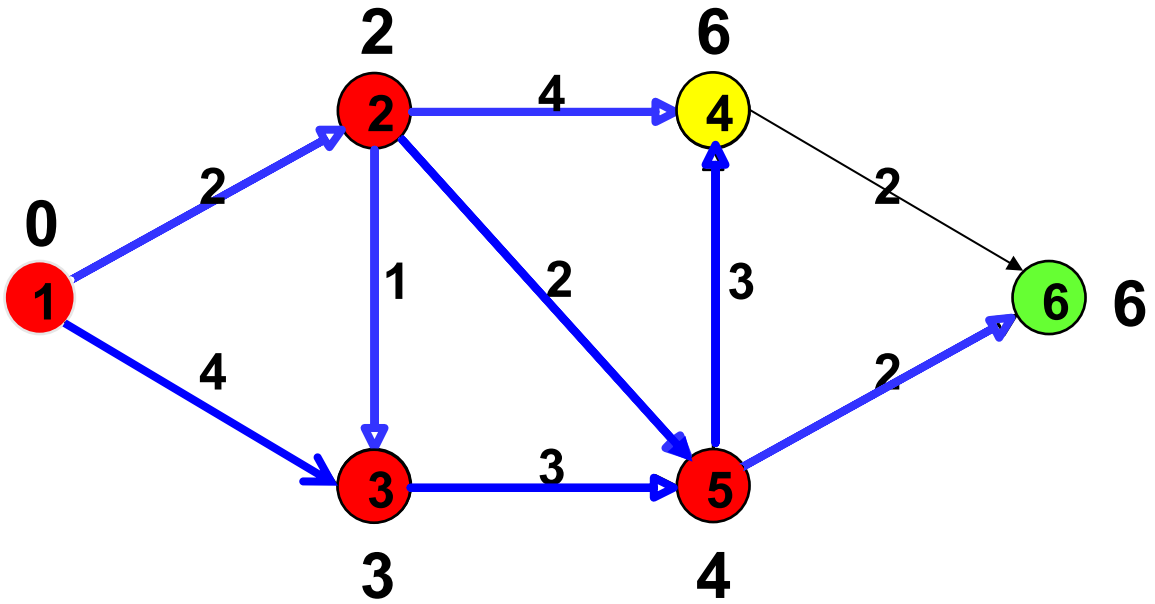
- Update distance of 4 and 6 coming from 5.
  - Except node 4 doesn't change.



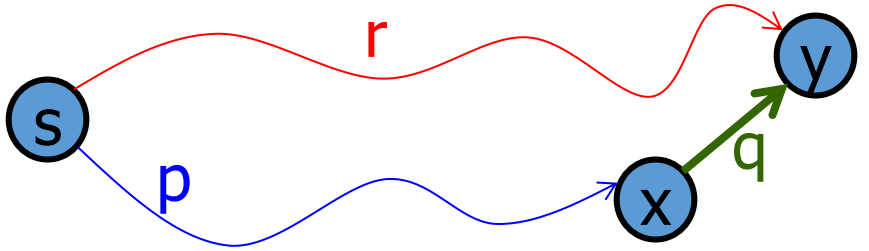
Example 2



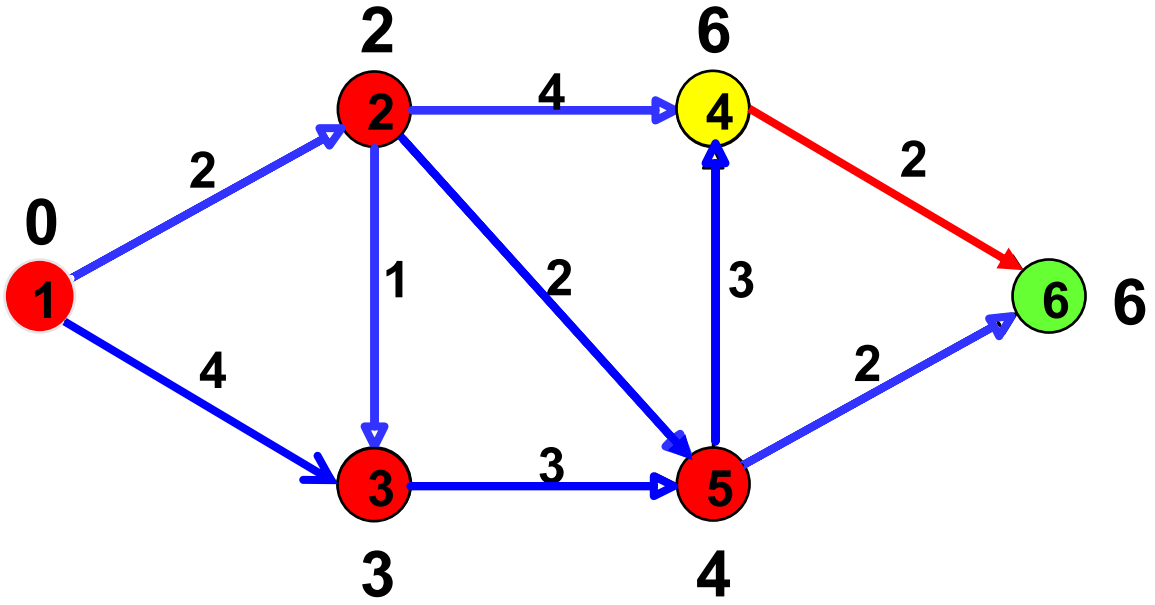
- Select next node: 4



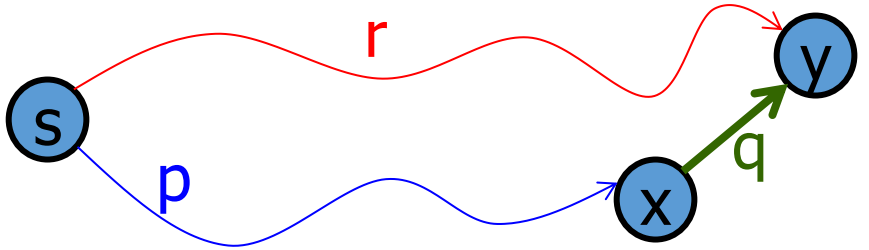
Example 2



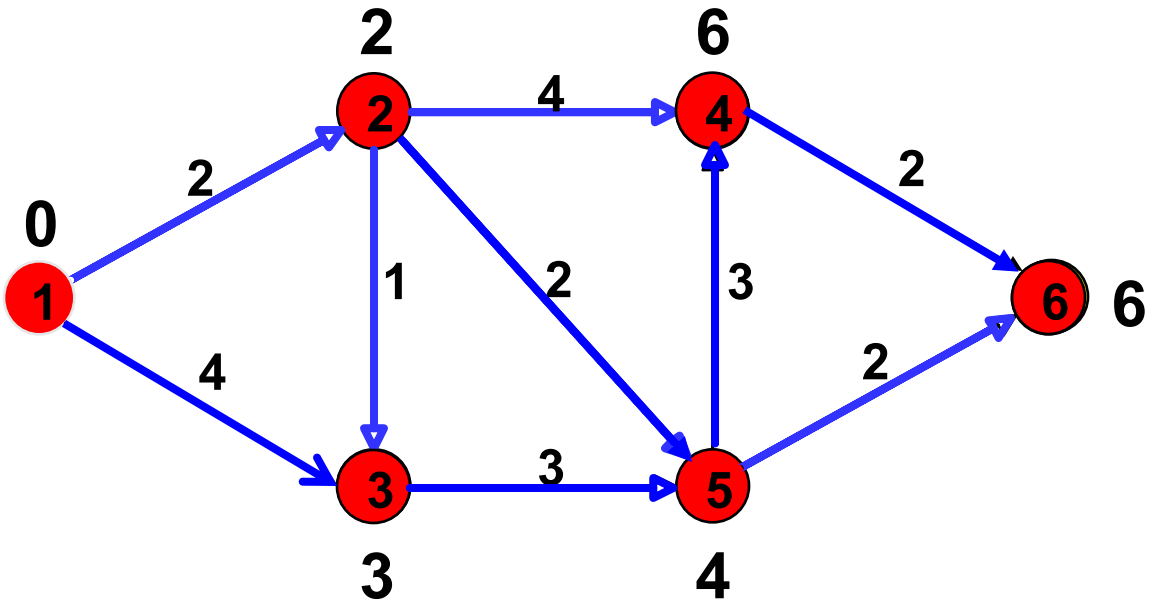
- Update distance of 6 coming from 4.
  - Except it's not updated



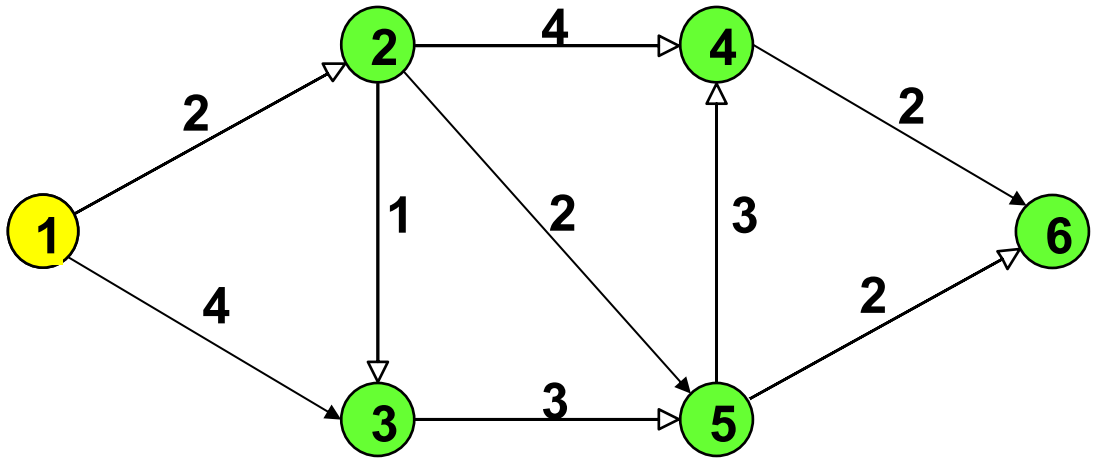
Example 2



- Select next node: 6
  - We reach the destination: done

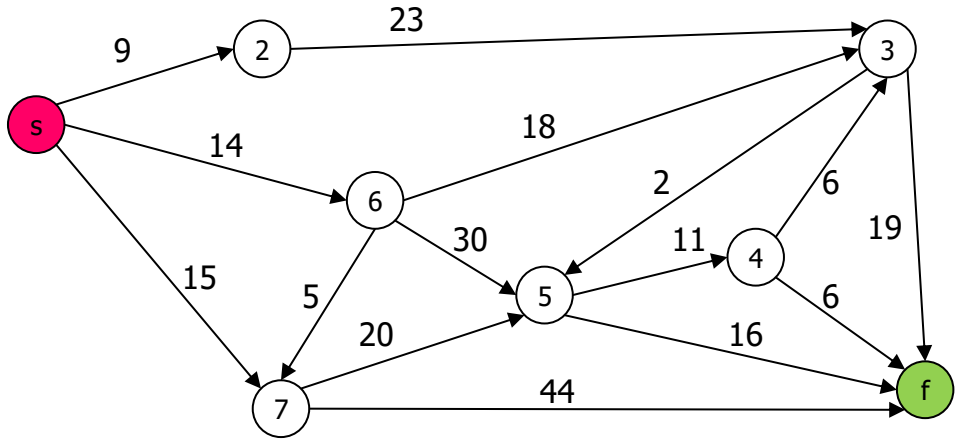


Example 2



Step	Select	Known	To do	Best 1	Best 2	Best 3	Best 4	Best 5	Best 6
0	-	-	1	0	∞	∞	∞	∞	∞
1	1	123	23	0	2	4	∞	∞	∞
2	2	12345	345	0	2	3	6	4	∞
3	3	12345	45	0	2	3	6	4	∞
4	5	123456	46	0	2	3	6	4	6
5	4	123456	6	0	2	3	6	4	6
6	6	123456	-	0	2	3	6	4	6

Example 3



Step	Select	Known	To do	s	2	3	4	5	6	7	f
0	-	-	s	0	∞	∞	∞	∞	∞	∞	∞
1	s	s267	267	0	9	∞	∞	∞	14	15	∞
2	2	s2367	367	0	9	32	∞	∞	14	15	∞
3	6	s23567	357	0	9	32	∞	44	14	15	∞
4	7	s23567f	35f	0	9	32	∞	35	14	15	59
5	3	s23567f	5f	0	9	32	∞	34	14	15	51
6	5	s234567f	4f	0	9	32	45	34	14	15	50
7	4	s234567f	f	0	9	32	45	34	14	15	50
8	f	s234567f	-	0	9	32	45	34	14	15	50

Algorithm 5

- Input: A graph G=(N, E), edge distances D(x,y), a source node s and a destination node d
- Output: Shortest distance from s to d
- Algorithm from previous working step experience:

```
initialize best[n] to infinity for all n except s, best[s] = 0
todo = s, known = {}, done = {}
repeat
    for all neighbors b of todo do
        if best[todo]+D(todo,b) < best[b] then best[b] = best[todo]+D(todo,b)
        add b to known
    add todo to done # we are done with todo
    set todo to be the node in (known – done) such that best[todo] is minimized
until known = N (or d is in done)
```

Algorithm 6

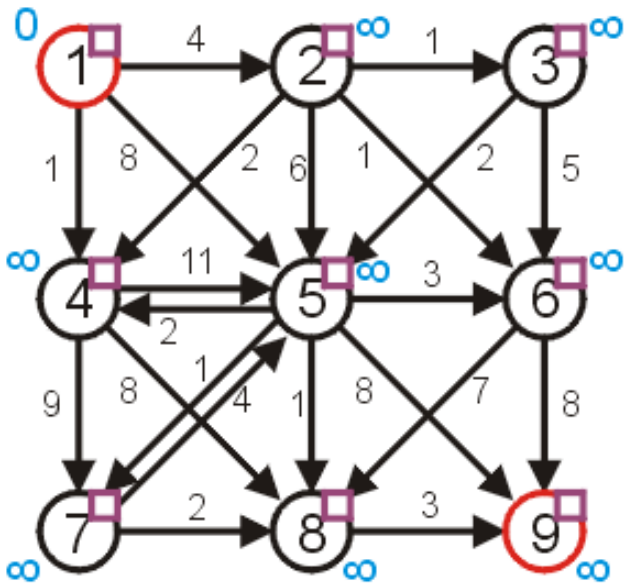
- A slightly rewritten algorithm:

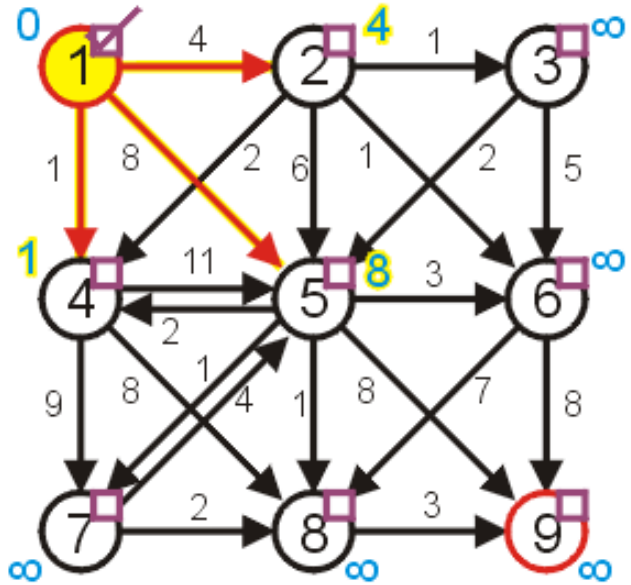
```
done = {}
best[s] = 0, best[n] = infinity for n != s
while |done| != N do
   choose t in (N – done) with minimum best[t] # unknown not considered
   for each b in the neighborhood of t do
      if best[t]+D(t,b) < best[b] then best[b] = best[t]+D(t,b)
   add t to done
```

- Note:
  - Step 1: Source node s is in the set (N – done) and is the first t selected (best is 0).
    - Other nodes in (N – done) will not be selected with best being ∞.
  - It can compute all shortest paths coming from s.
  - If destination d is in done, you may jump out of the while loop earlier.
    - Side benefit is that all nodes on the shortest path to d will also have their shortest paths computed.

Example 4

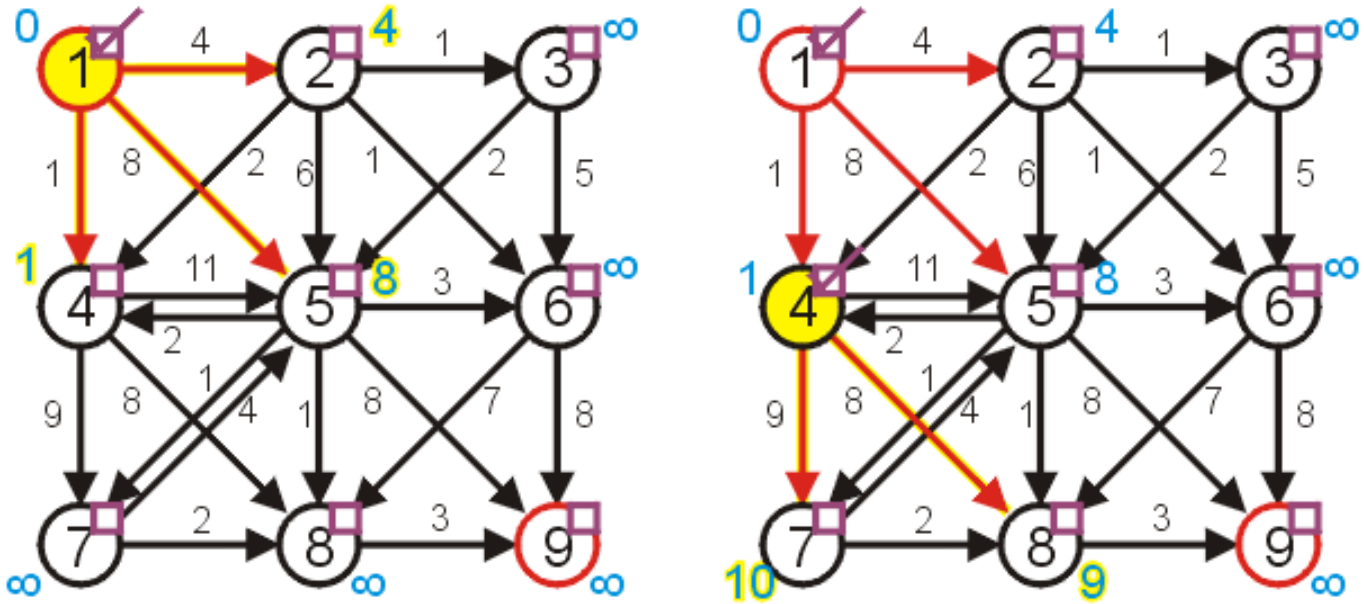
- Another example: go from 1 to 9.
  - There are 47 possible paths!
  - Start from 1, update best for 2, 4, 5.





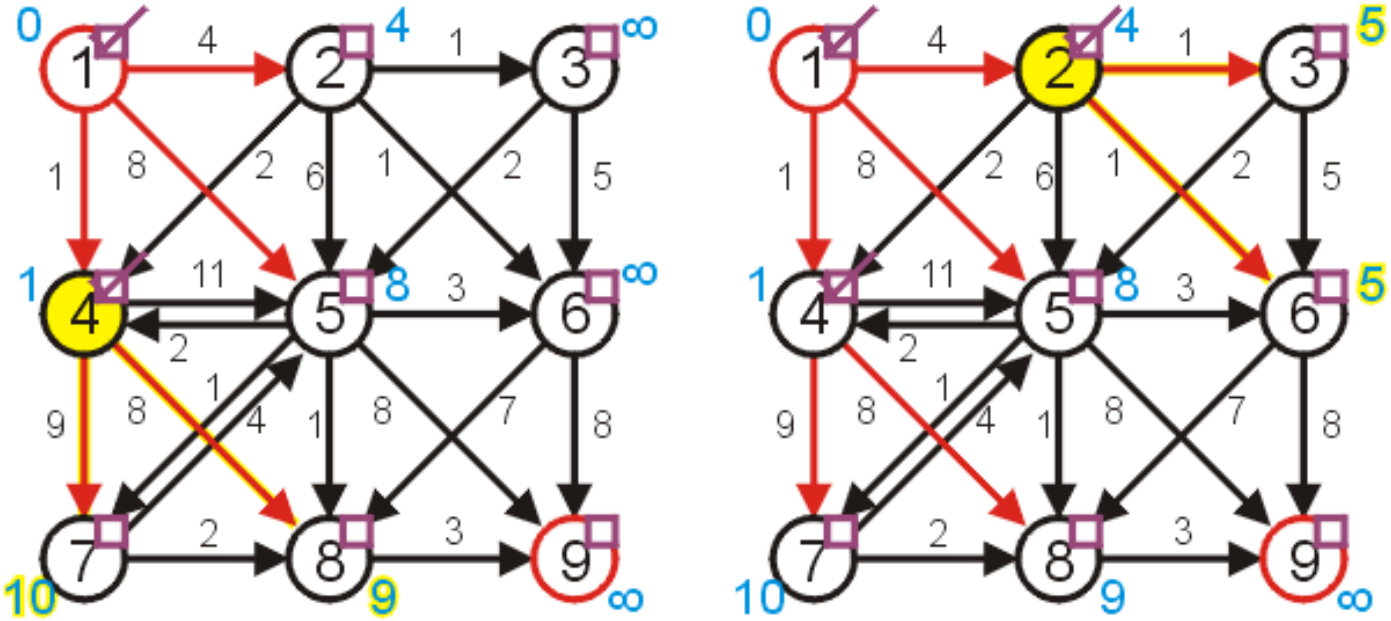
Example 4

- Select 4.
- Update best for 5, 7, 8.



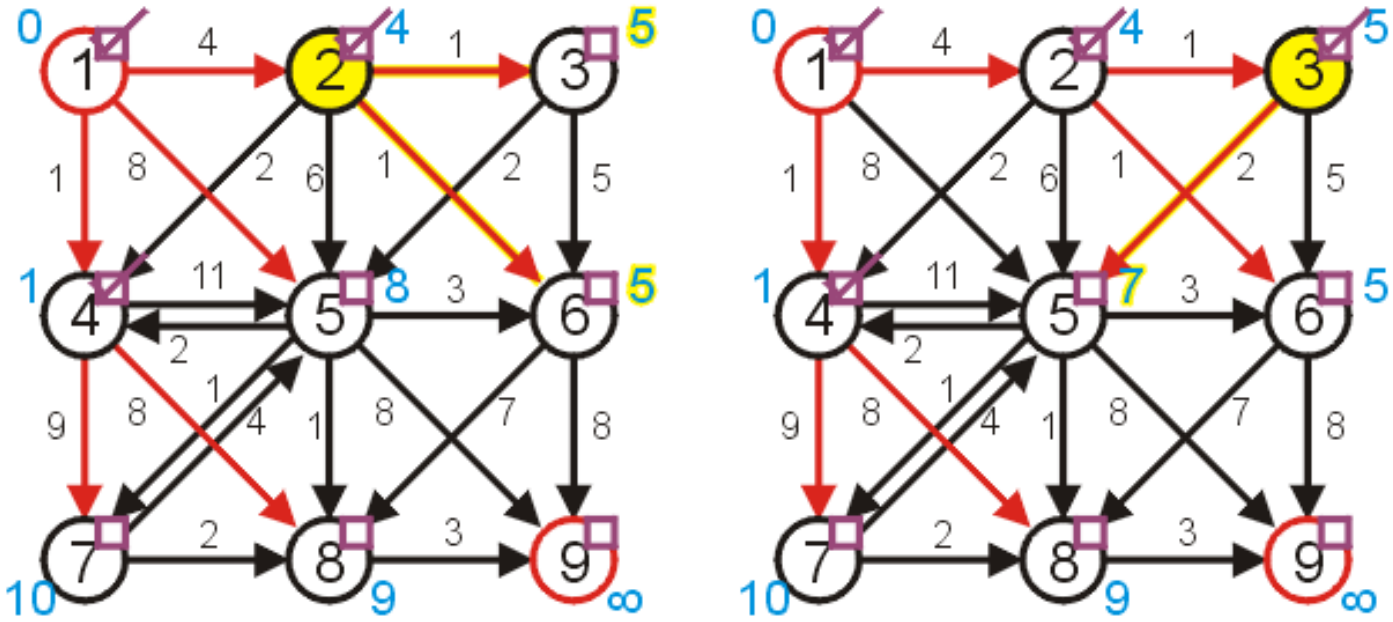
Example 4

- Select 2.
- Update best for 3, 4, 5, 6.



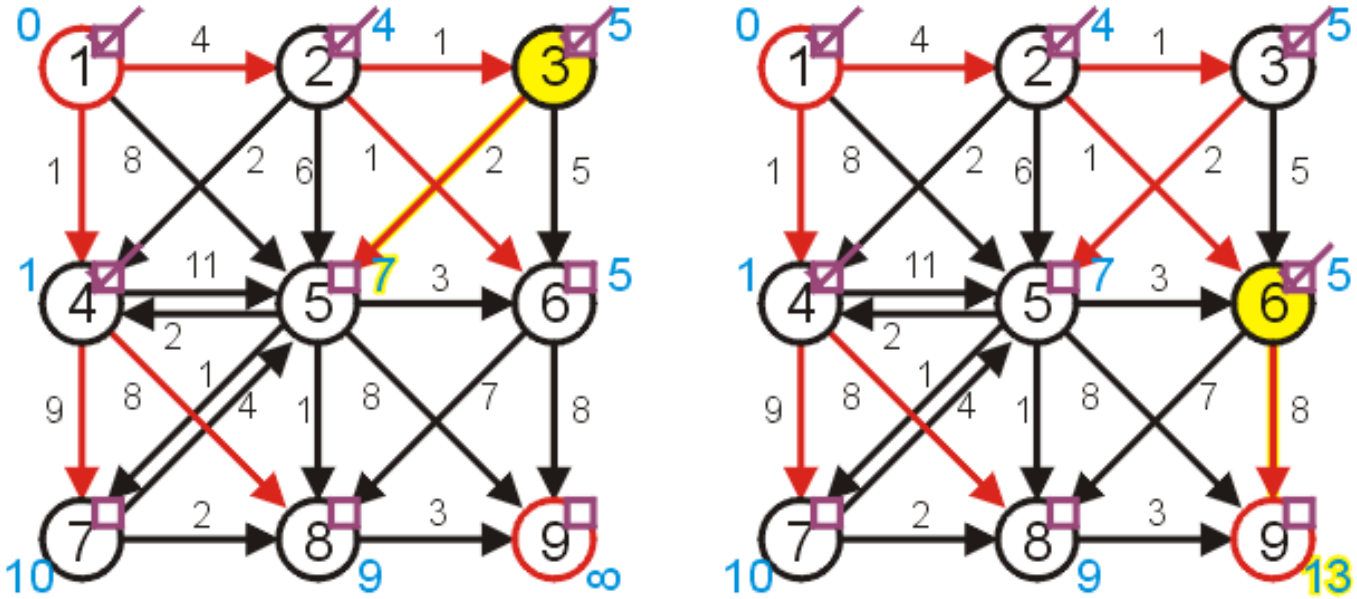
Example 4

- Select 3 (may also select 6).
- Update best for 5, 6.



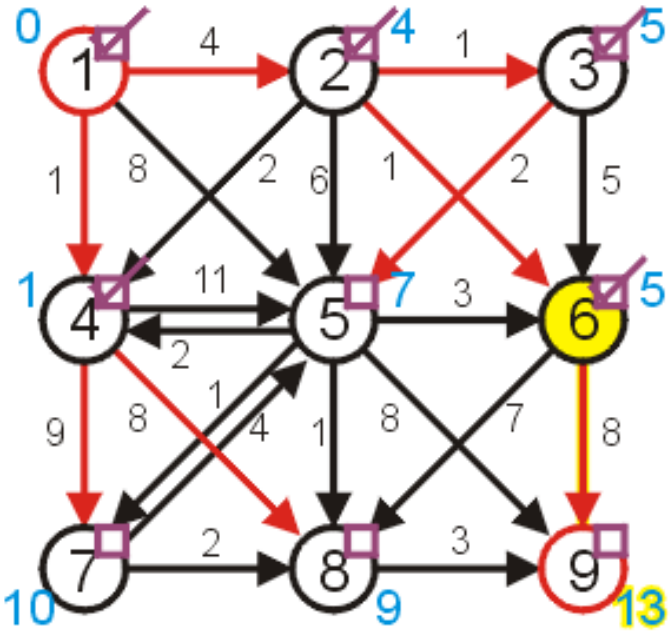
Example 4

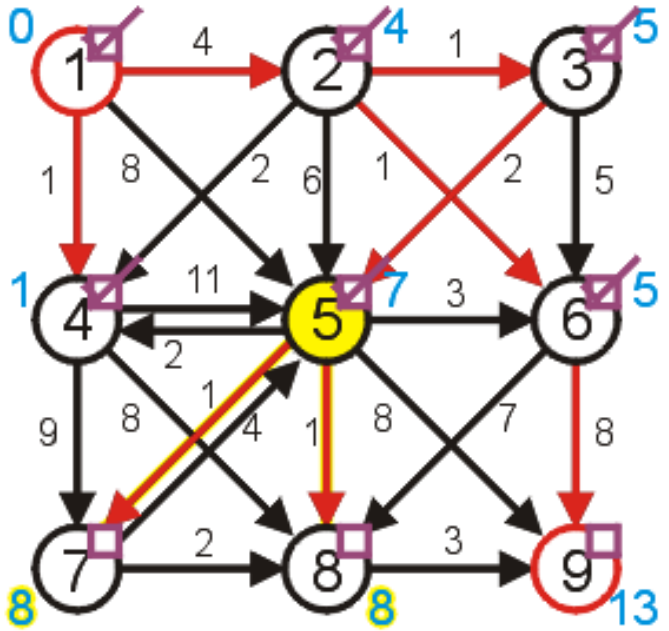
- Select 6.
- Update best for 8, 9.



Example 4

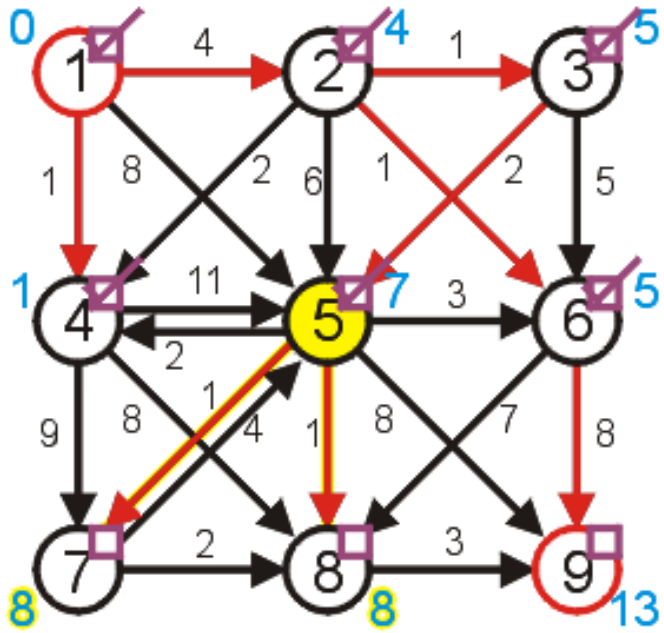
- Select 5.
- Update best for 4, 6, 7, 8, 9.

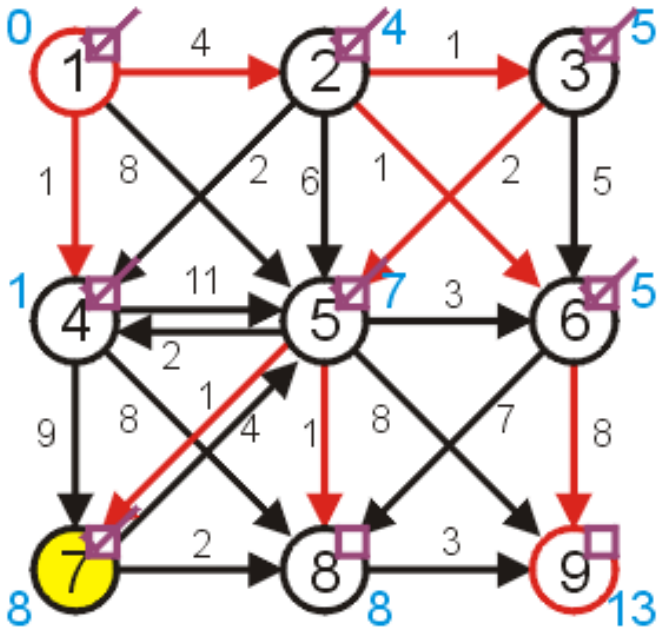




Example 4

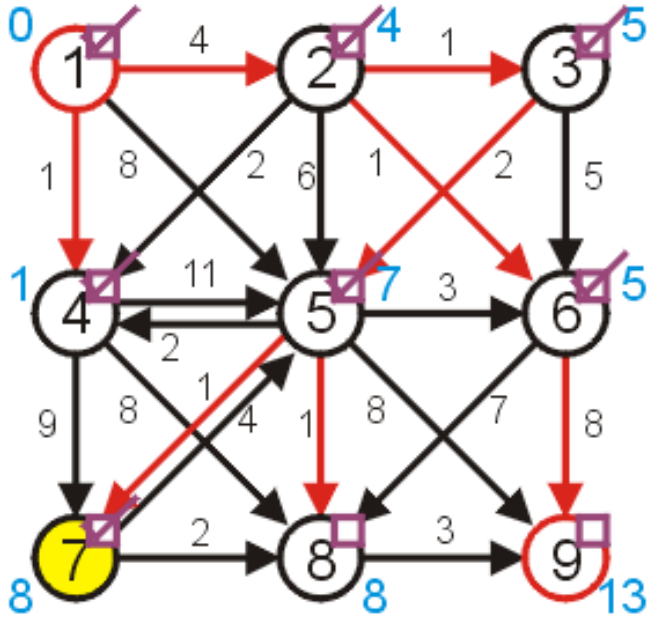
- Select 7 (or 8).
- Update best for 8.

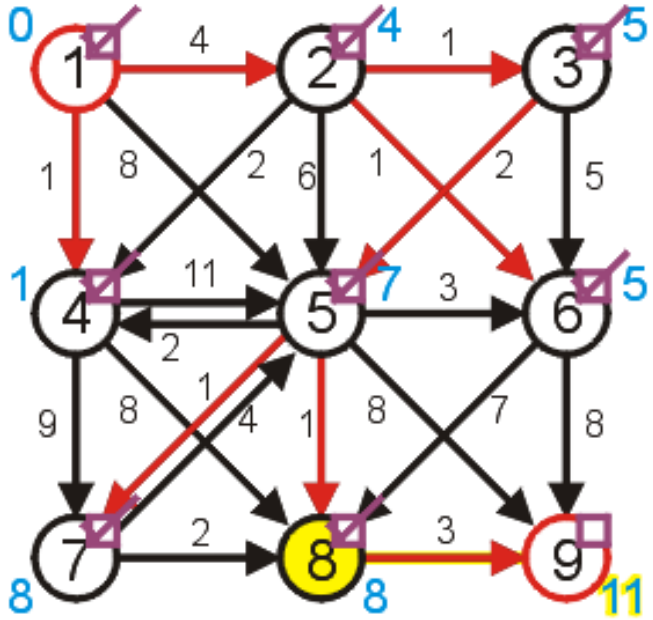




Example 4

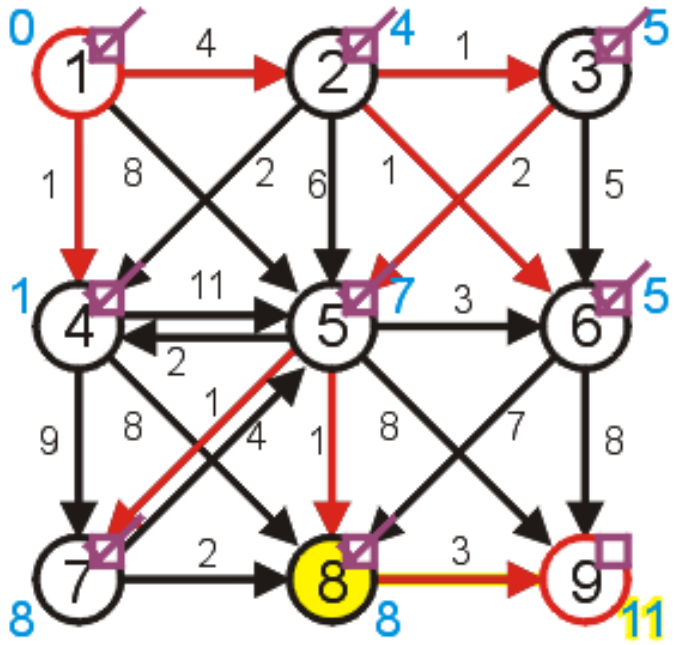
- Select 8.
- Update best for 9.

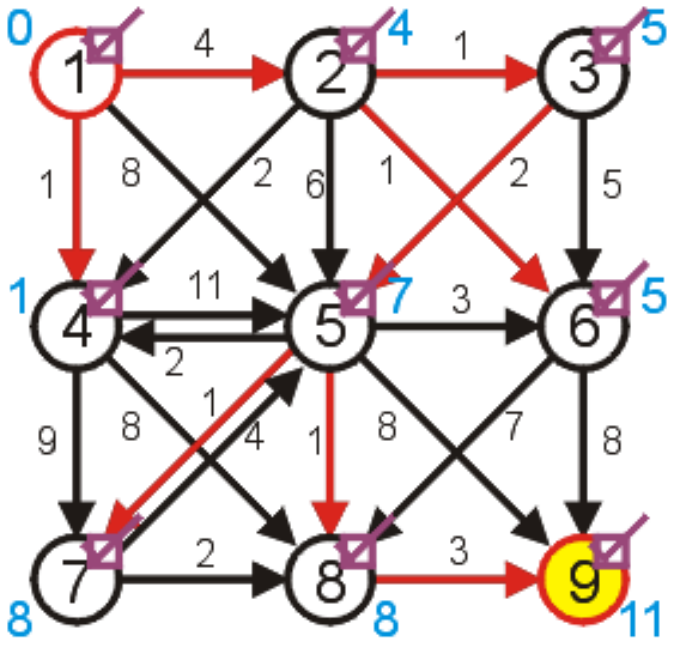




Example 4

- Select 9.
- Done





The Final Step

- What is missing so far?
  - We only know the shortest distance from s to d.
  - How can we get back the shortest path leading from s to d?
- We need to remember where we come from at each node when a new minimum is found.
  - We do not need to remember the full path here.
  - Remembering the full path is too tedious.
  - Change to Algorithm 4:

```
if best[x]+D(x,y) < best[y] then
  best[y] = best[x]+D(x,y)
  from[y] = x
```

The Final Step

- We remember where we come from.
  - Change to Algorithm 5:

```
if best[todo]+D(todo,b) < best[b] then
    best[b] = best[todo]+D(todo,b)
    from[b] = todo
```

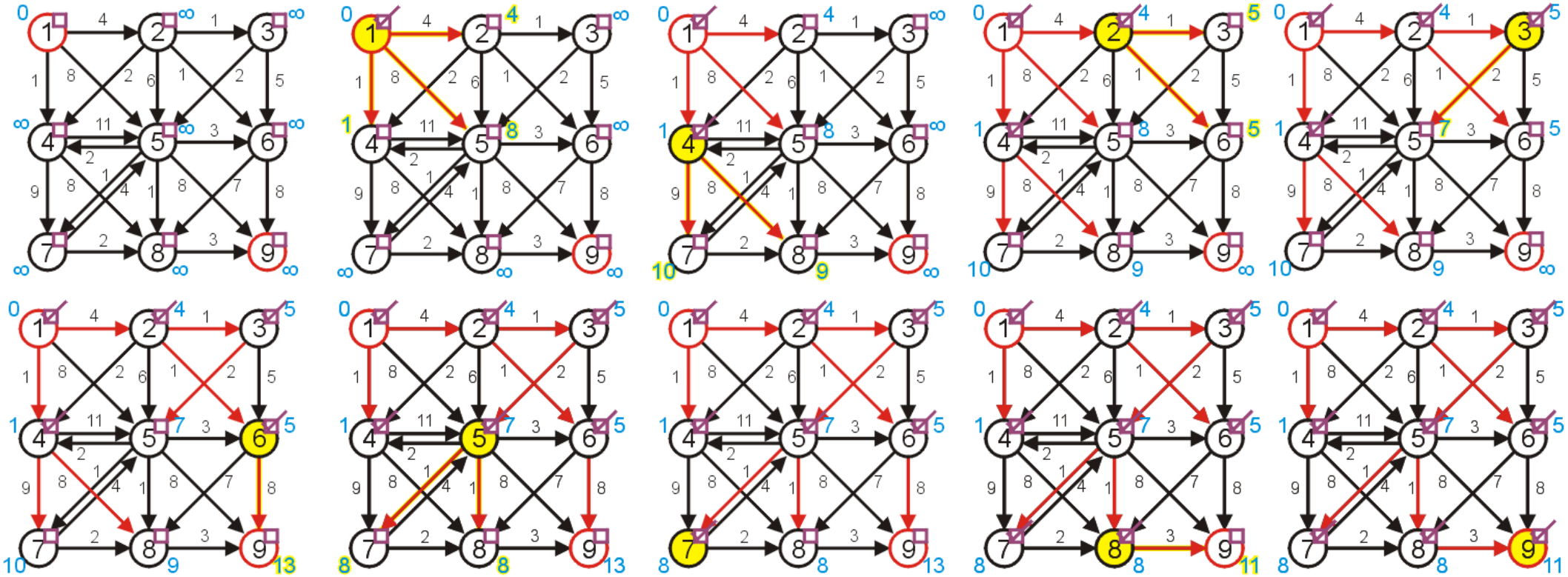
  - Change to Algorithm 6:

```
if best[t]+D(t,b) < best[b] then
    best[b] = best[t]+D(t,b)
    from[b] = t
```

- Upon algorithm completion, trace back starting at from[d] until we reach s.
  - The shortest path is the reverse of this traced path.

The Final Step

- Find the reverse path via red edges (representing best).
  - Tracing 9 8 5 3 2 1 $\Rightarrow$ shortest path is 1 2 3 5 8 9.



The Final Step

- Path tracing allows you to get information of where you come from.
  - You just need to remember where you come from at each step, without worrying about the full path.
- This same path tracing technique in Algorithm 5/6 by saving the information in the from[] list can be also applied to Algorithm 4.
  - Algorithms 1 and 2 already have the path ready, so no need to implement this.
- This type of partial information saving technique could also be applied in some other problem domains.

Problem Solving Lessons

- Lessons learned from this shortest path problem.
  - Come up with a first workable perhaps inefficient solution.
    - We have Algorithm 1: find all possible paths and compute the distance and it clearly is correct.
  - Improve on the workable solution.
    - Algorithm 2: try to reduce the paths to study, but unfortunately fail.
  - Take an alternative view.
    - Start from using no edge, explore how using edges can lead to better path.
    - We have Algorithm 3: try each edge on improving the paths.
  - Separate the issues.
    - Algorithm 3 does not give the path.
    - Be patient and solve finding the distance problem first.

Problem Solving Lessons

- Lessons learned from this shortest path problem.
  - Verify the solution.
    - Algorithm 3 works sometimes, and fails sometimes.
  - Fix the solution.
    - Repeat steps in Algorithm 3 until we are comfortable with the answer to produce Algorithm 4.
    - We actually repeat until all best distances do not change between two rounds.
      - We call this the convergence of the algorithm.
    - It has been proven that Algorithm 4 will converge after at most m-1 rounds for a graph with m nodes.
    - Repeating until convergence is a common technique.
      - Solving an equation.
      - Finding the minimum point.
      - Finding key information in building a compiler.

Problem Solving Lessons

- Lessons learned from this shortest path problem.
  - Take yet another alternative view.
    - Edges in Algorithm 4 need to be studied repeatedly, perhaps we should start from nodes.
    - Explore nodes carefully by trying the most promising first.
    - Look at edges coming out of the exploring node.
    - Each edge with direction is only considered once.
    - Each edge without direction is only considered twice (actually, an undirected edge means a pair of directed edge).
    - This results in Algorithm 5: already a good algorithm.
  - Verify and tidy up the solution.
    - Algorithm 5 can still be tidied up to make the solution cleaner.
  - Complete the solution.
    - We need to find the actual path (put aside earlier).

Graph Summary

- Representing a graph conceptually.
- Representing a graph in Python.
- Trying to solve the problem of finding the shortest path.
  - Algorithm 1 is very slow.
  - Algorithm 2 is not quite correct.
  - Algorithm 3 is not quite correct.
  - Algorithm 4 is correct, but could need more rounds.
  - Algorithm 5 works well.
  - Algorithm 6 is the standard.
  - Establish the final mechanism to return the shortest path.
- There are still many related graph problems to be solved.
- Question: Is an adjacency matrix or an adjacency list better for the shortest path problem?

General Problem Solving

- Practice alot


BUT WAIT
THERE'S MORE!