# CSE 421

## Alg Design by Induction, Dynamic Programming

Shayan Oveis Gharan

# Q/A

- How to practice more?
  - Try more exercises: there are lots of exercise in the book
  - See https://train.usaco.org/usacogate

- How to think, how to write?
  - Many cases it is better to spend more time on thinking than writing.
  - Try to write concise proofs for HW problems.
  - Make sure you use all assumptions of the problem.

# Sample Soln of Problem 2 Midterm

In HW2-P3 we designed an algorithm to find the shortest path in a graph with weights {1,2,3} where we break edge of weight $w_e$ into a path of length $w_e$. Since all edge weights have the positive integer weights, we can run the same algorithm to construct a modified graph G'. Solve problem on G' by DFS.

**Runtime**: Since sum of edge weights is at most 4m G' has O(m) edges and O(m+n) vertices so the algorithm runs in O(m+n).

**Correctness**: Similar to HW there is a bijection between all paths from s to a vertex v in G, G', where we substitute each edge e with a path of length $w_e$. Therefore, the shortest path from s to v in G,G' are the same (for all v). The algorithm works since BFS finds the shortest path.

# Sample Soln of Problem 3 Midterm

Run the algorithm form P4 of Sample midterm except whenever comparing A[I] with I compare A[I]/2 with I and go to left if A[I]/2 > I and right if A[I]/2 < I.

Runtime: Similar to sample midterm we have the recursion T(n)=T(n/2)+O(1), So, T(n)=O(log n).

Proof of correctness: Construct an array B where B[i]=A[i]/2 (note that this is just for sake of analysis). Since A has distinct and sorted elements, array B elements are distinct and sorted. Furthermore, since elements of A are even, elements of B are integers. Our modifiied algorithm above is essentially running the algorithm from sample midterm on B. Since B is sorted and has distinct integers by the same proof the algorithm succeeds.

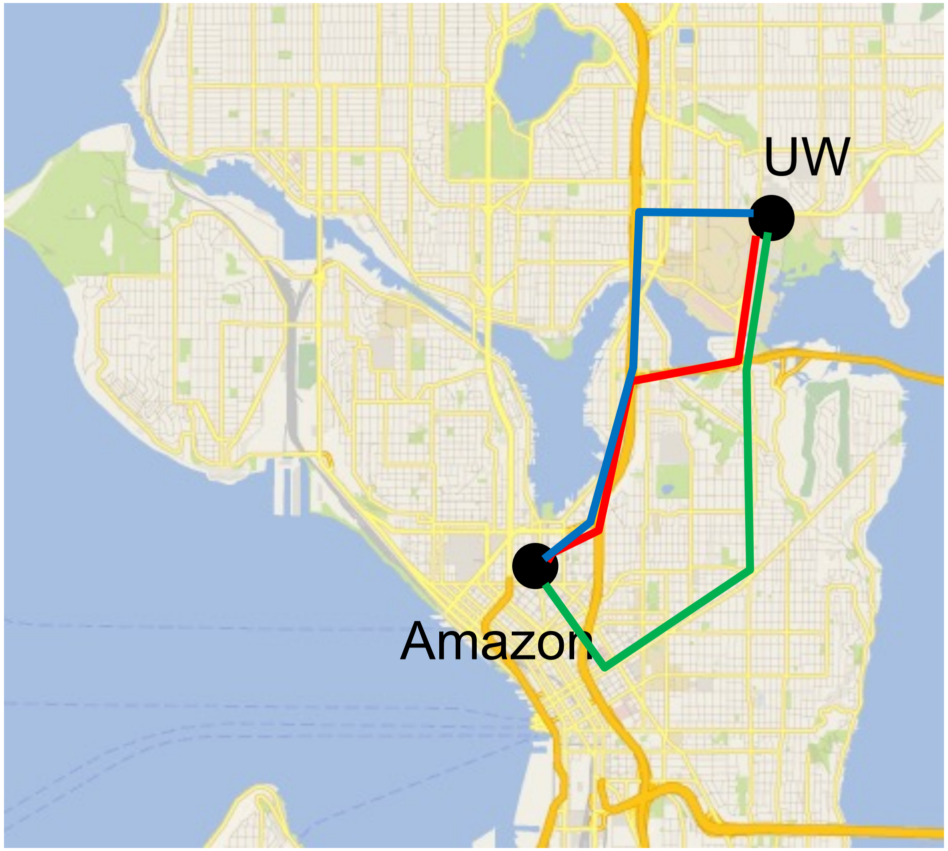
# Approximation Alg Summary

- To design approximation Alg, always find a way to lower bound OPT
- The best known approximation Alg for vertex cover is the greedy.
  - It has been open for 50 years to obtain a polynomial time algorithm with approximation ratio better than 2
- The best known approximation Alg for set cover is the greedy.
  - It is NP-Complete to obtain better than ln n approximation ratio for set cover.

# Single Source Shortest Path

Given an (un)directed graph G=(V,E) with non-negative edge weights $c_e \geq 0$ and a start vertex s

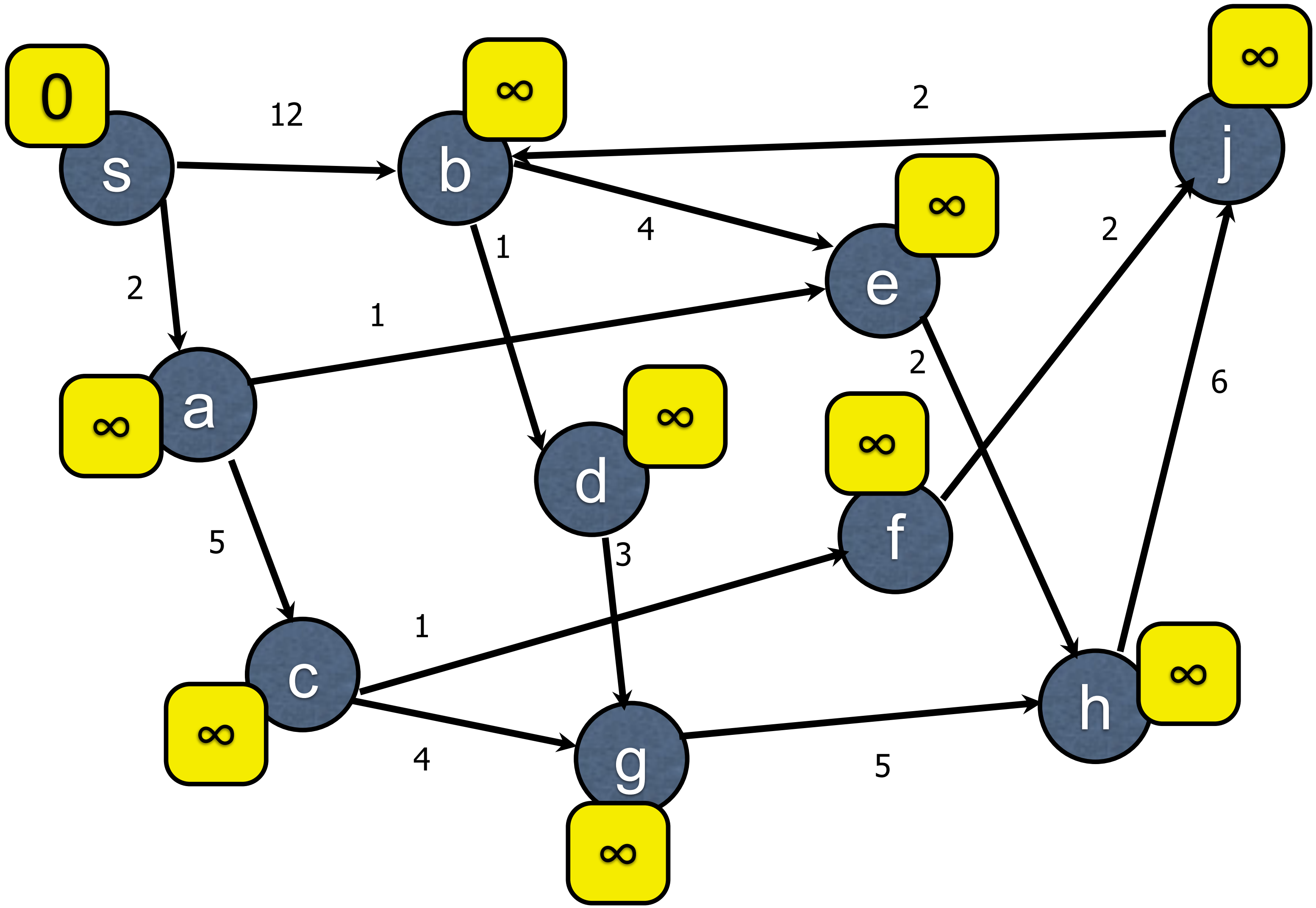
Find length of shortest paths from s to each vertex in G



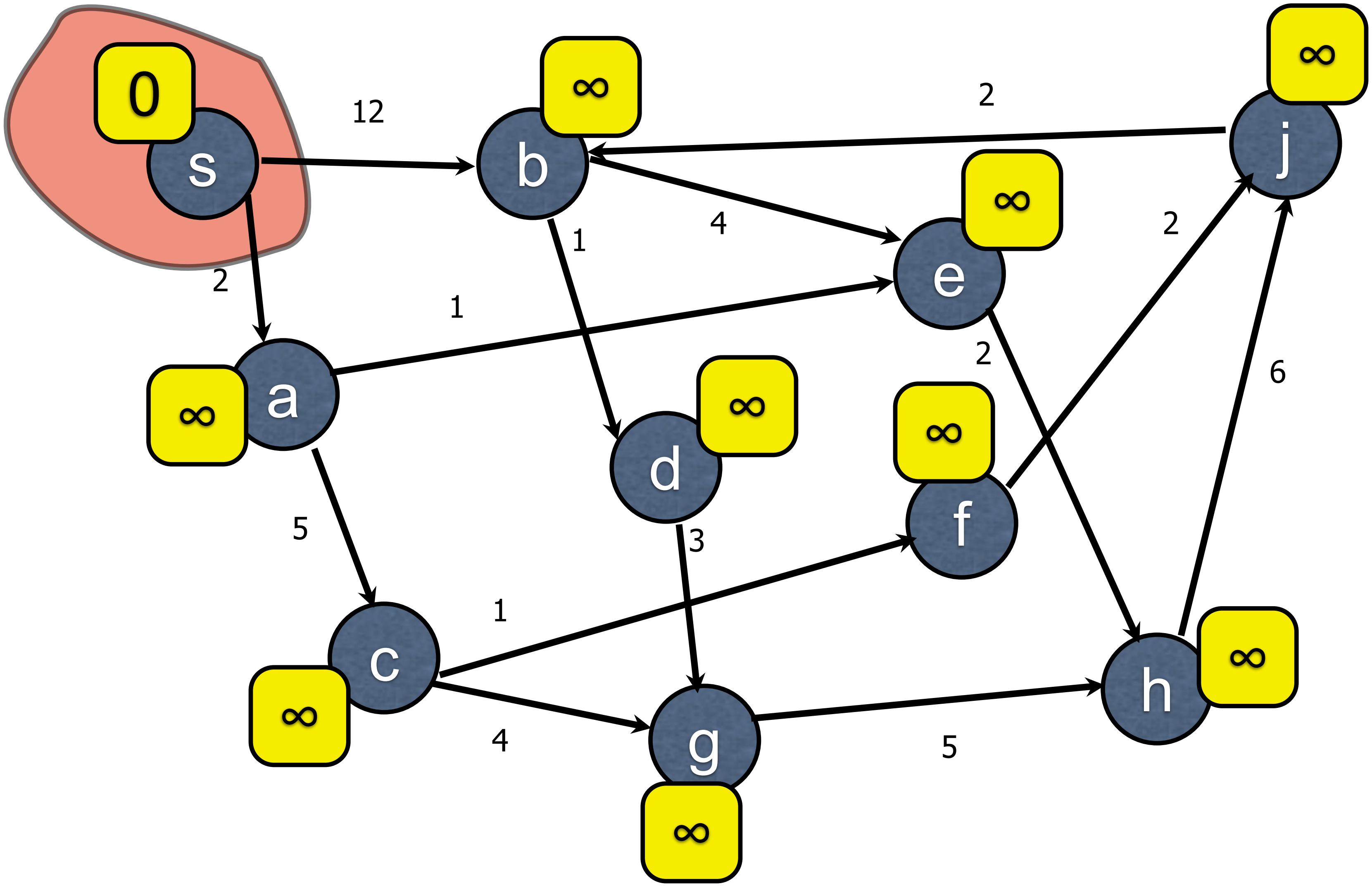
# Dijkstra(s)

- Set all vertices v undiscovered, $d(v) = \infty$
  Set $d(s) = 0$, mark s discovered.
  **while** there is edge from discovered vertex to undiscovered vertex,
- let (u,v) be such edge minimizing
  $$d(u) + c_{u,v}$$
- set $d(v) = d(u) + c_{u,v}$, mark $v$ discovered

# Dijkstra’s Algorithm



# Dijkstra's Algorithm

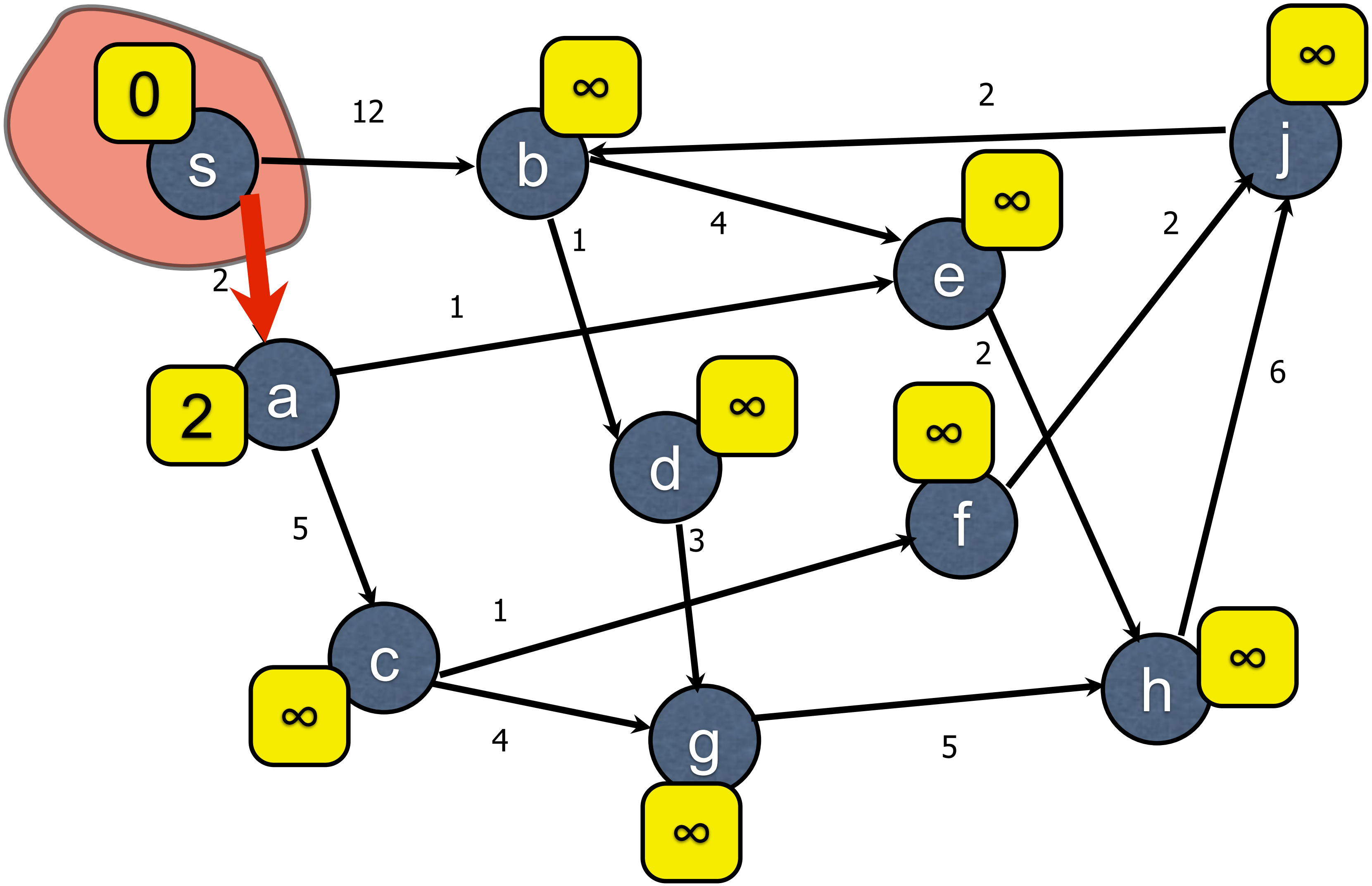



while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

set $d(v) = d(u) + l_{u,v}$, mark v discovered

# Dijkstra's Algorithm

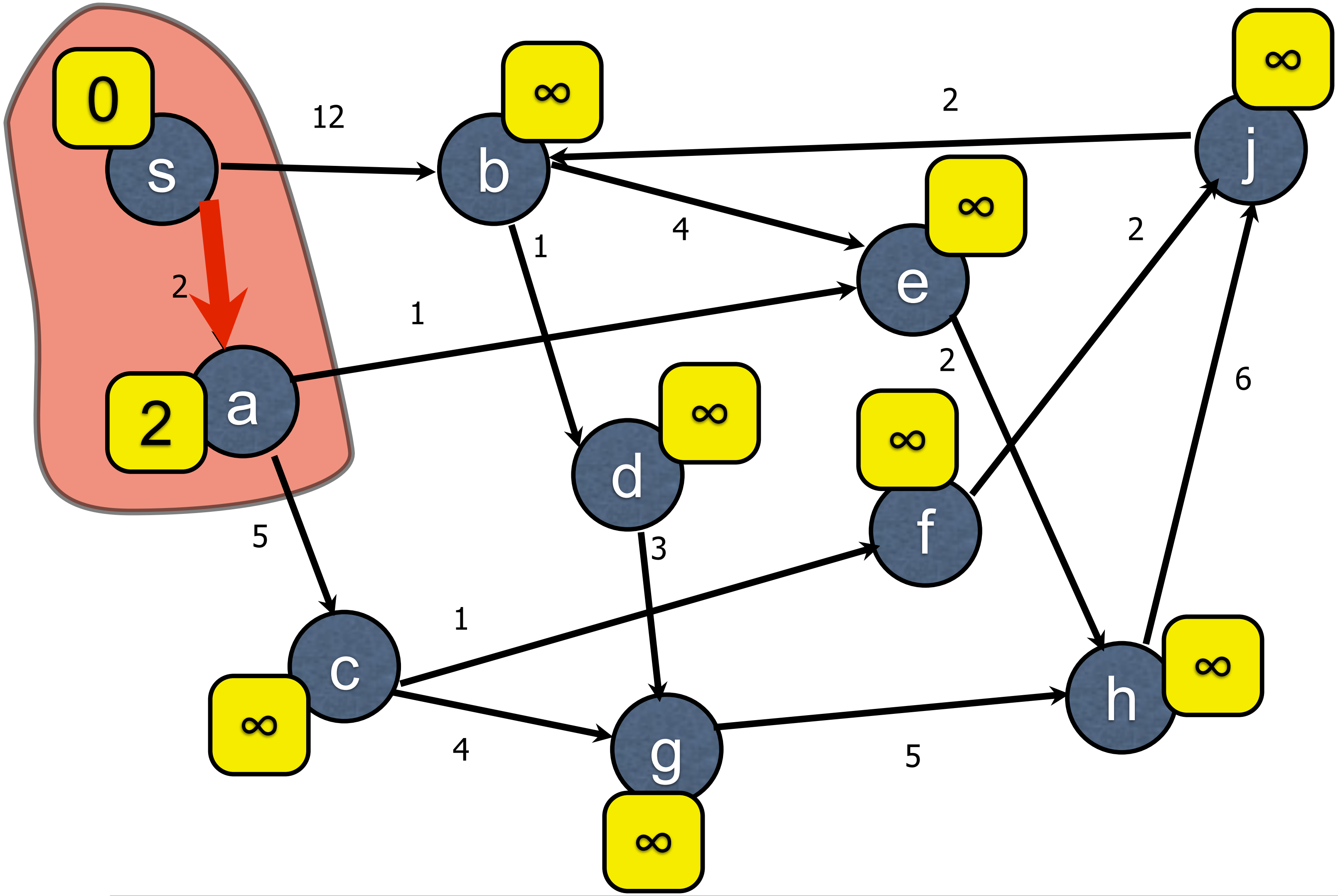



while there is edge from discovered vertex to undiscovered vertex,

let (u,v) be such edge minimizing $d(u)+l_{u,v}$

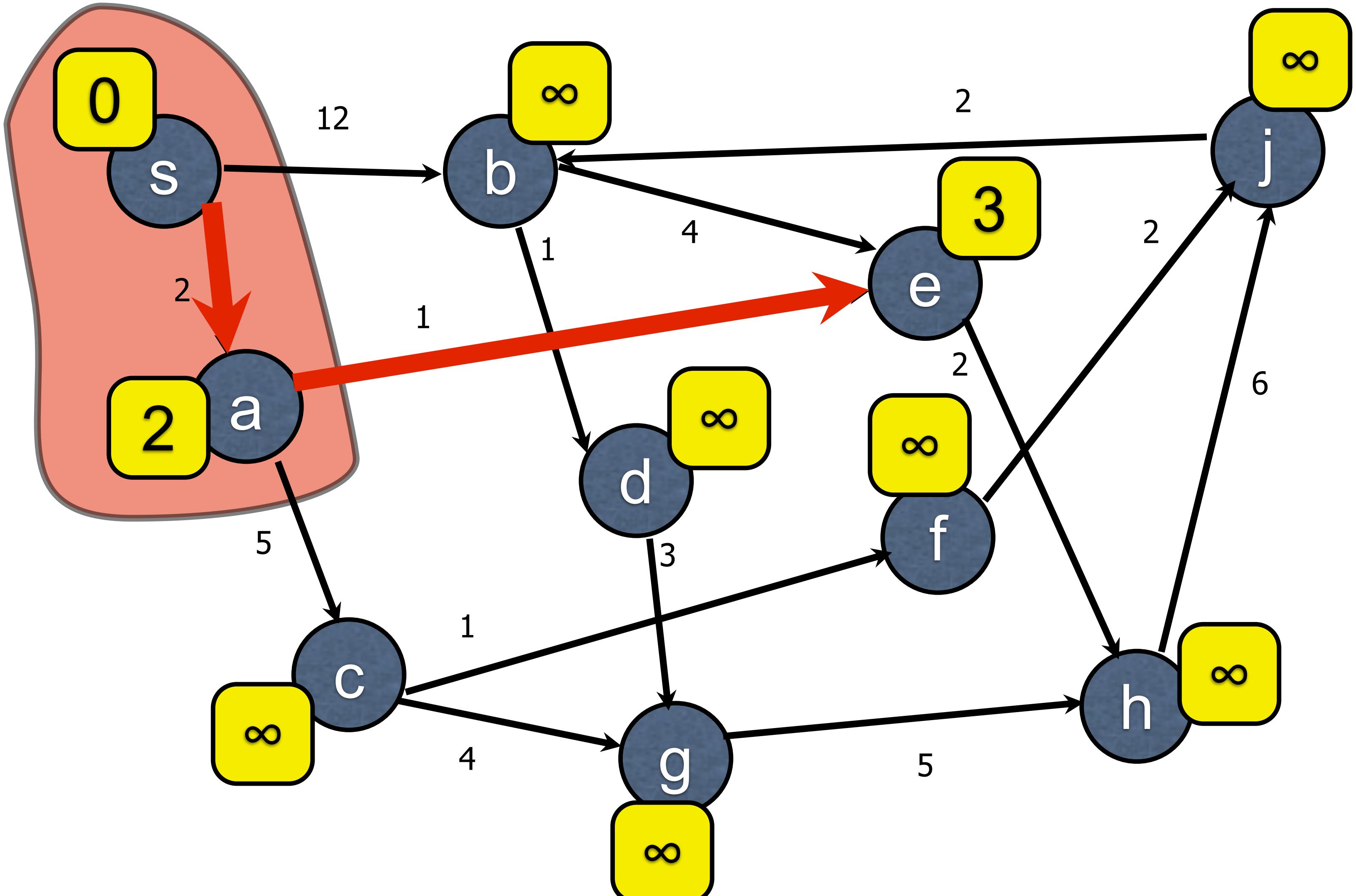
set $d(v) = d(u) + l_{u,v}$, mark v discovered

# Dijkstra's Algorithm




while there is edge from discovered vertex to undiscovered vertex,
    let $(u, v)$ be such edge minimizing $d(u) + l_{u,v}$
    set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm

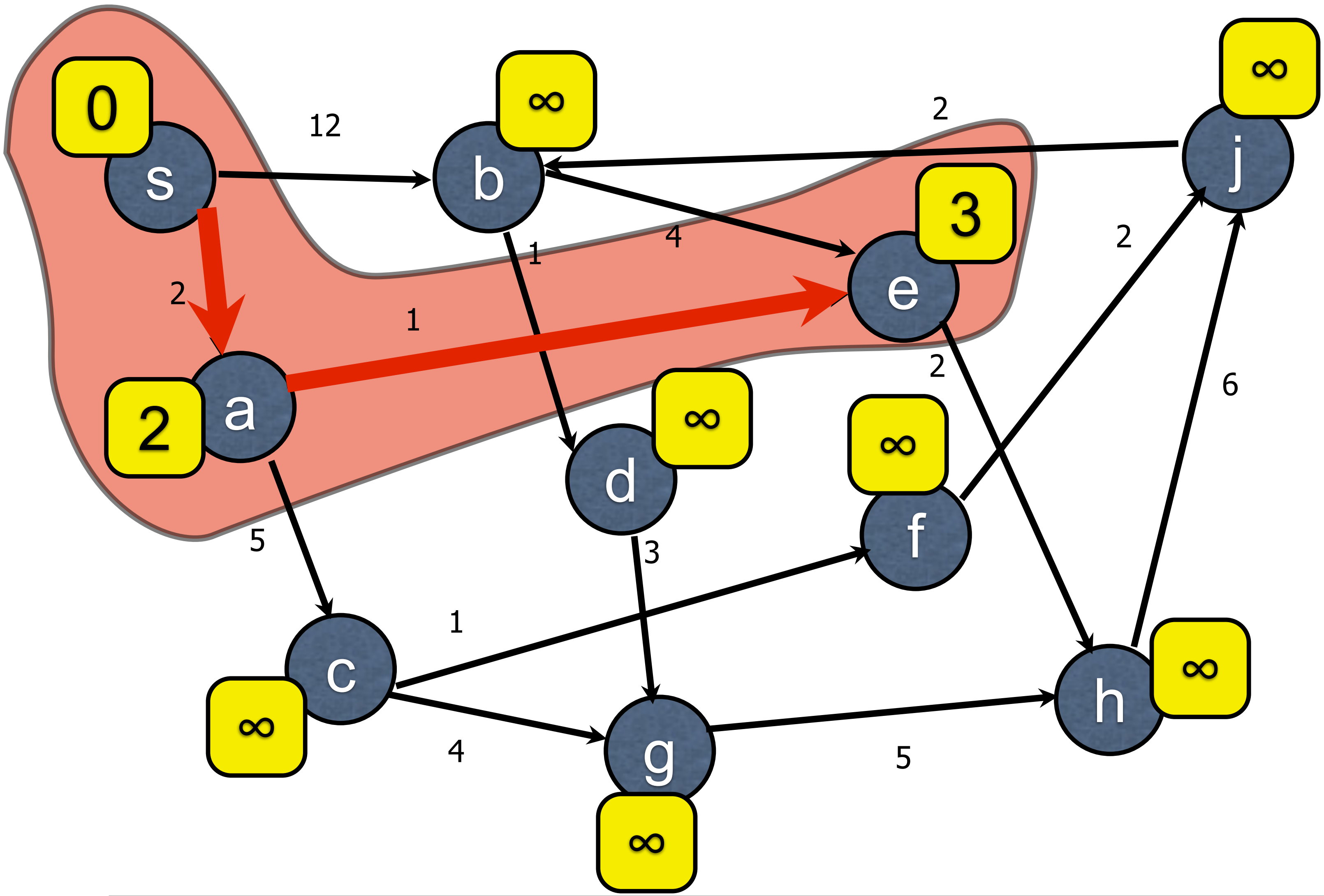



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm

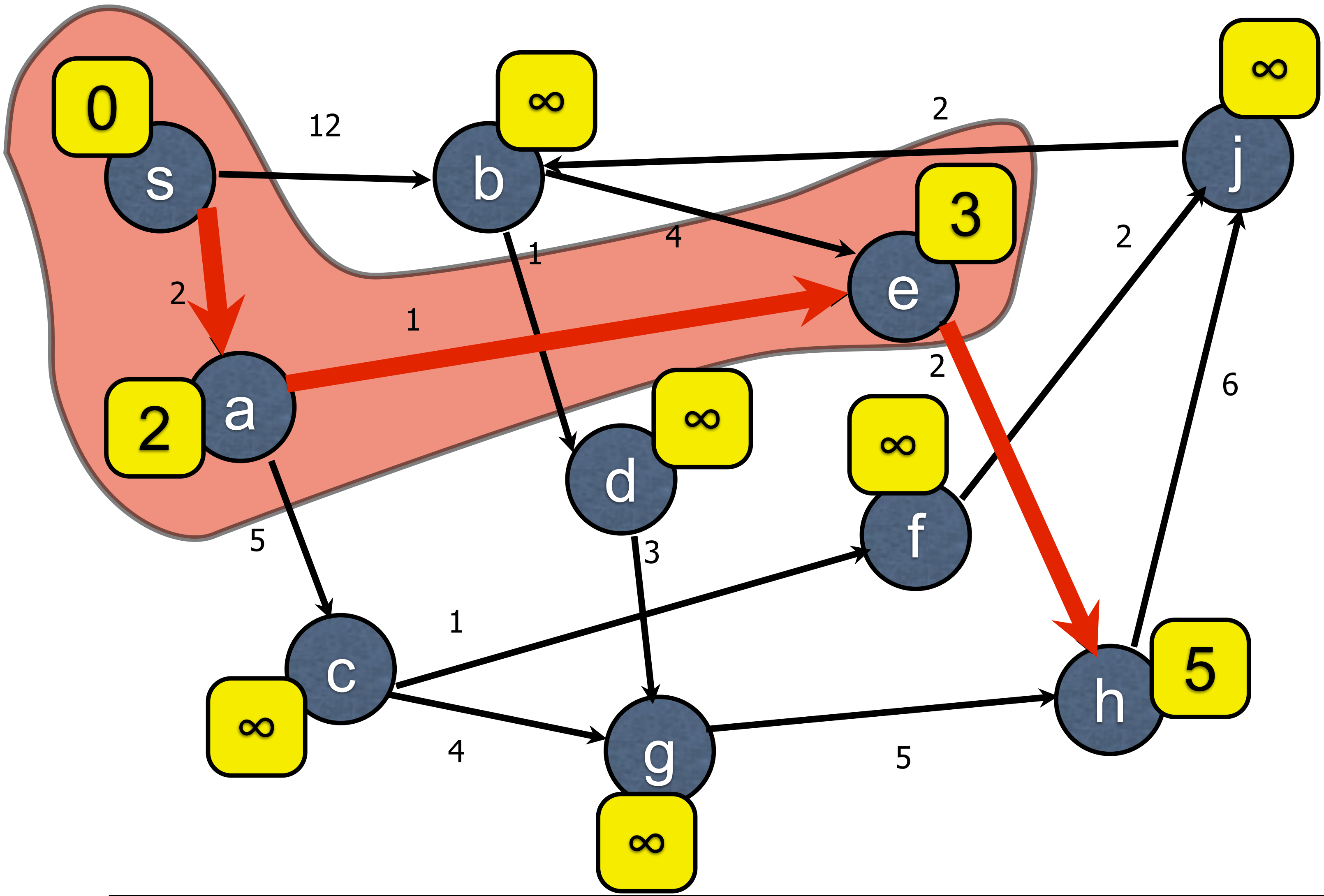



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

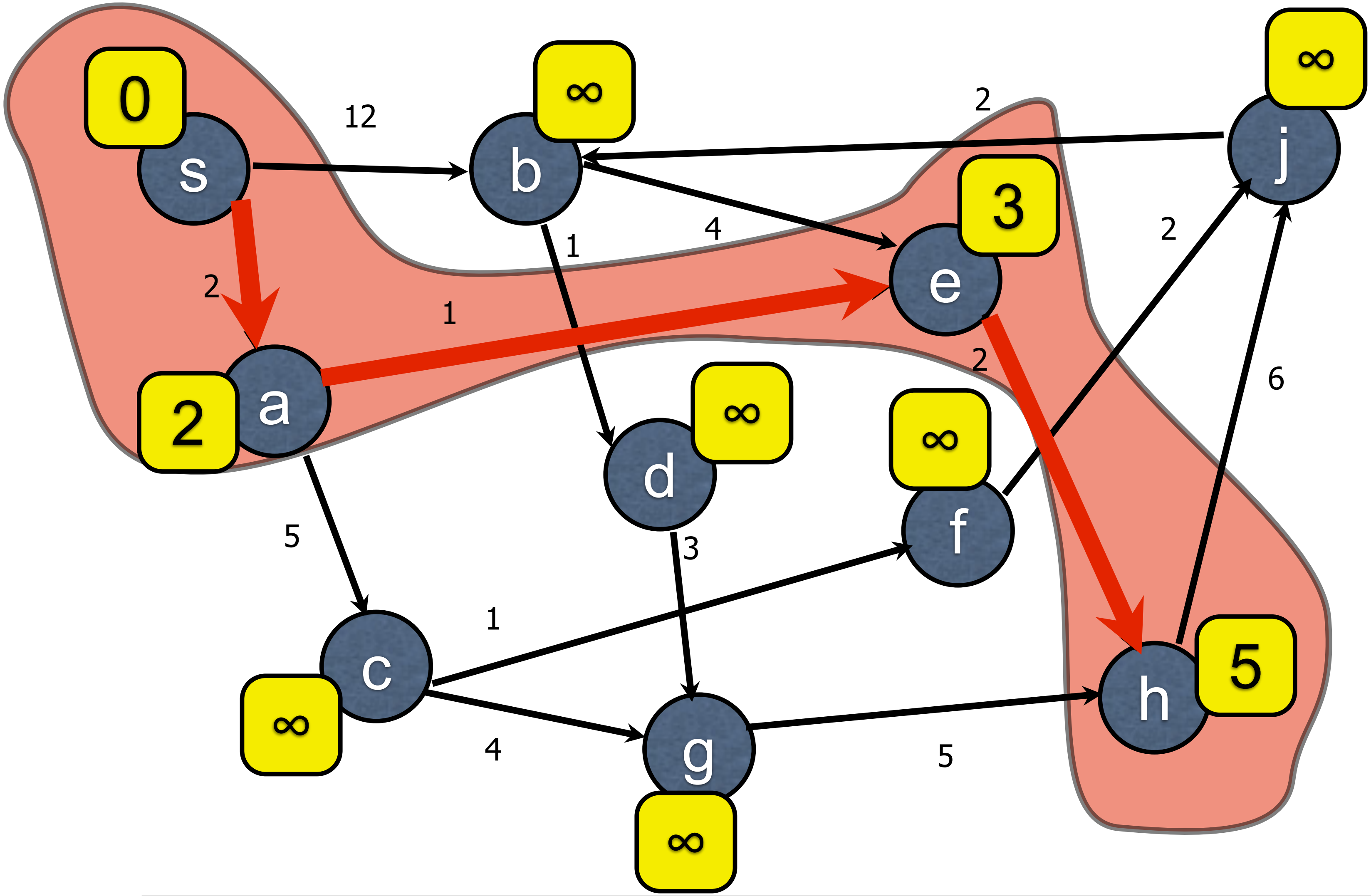
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm




while there is edge from discovered vertex to undiscovered vertex,
  let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
  set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm

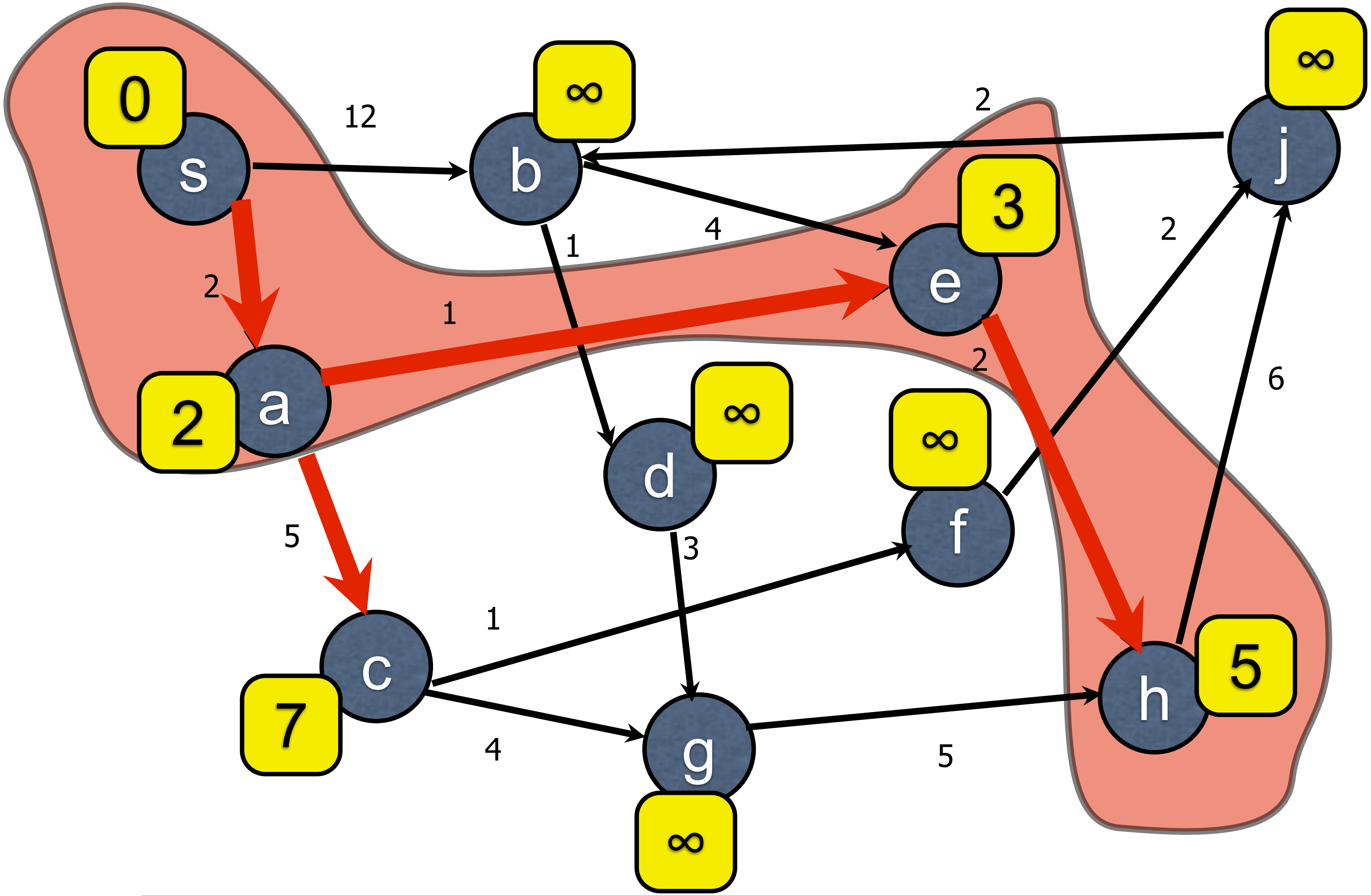



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm

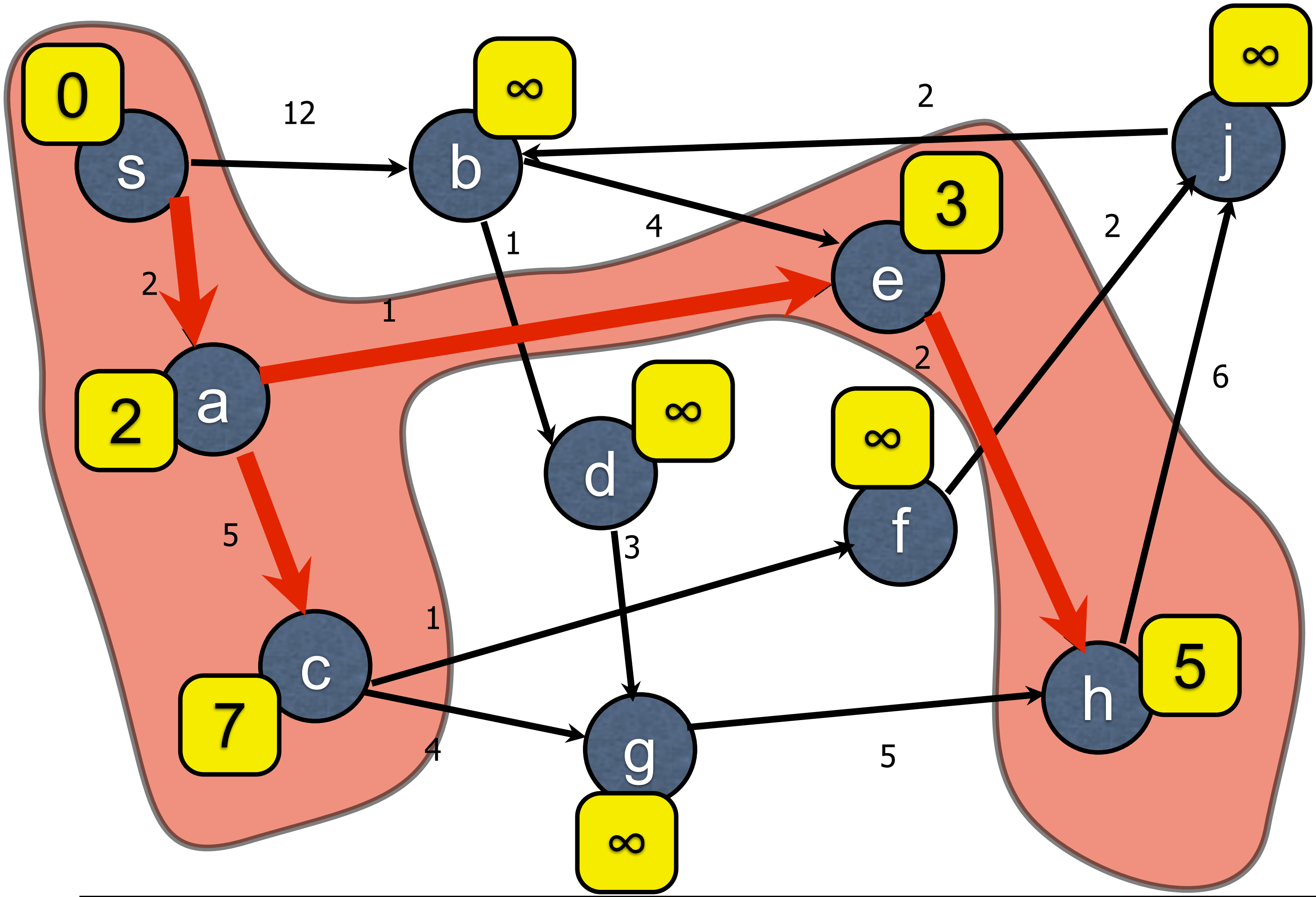



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

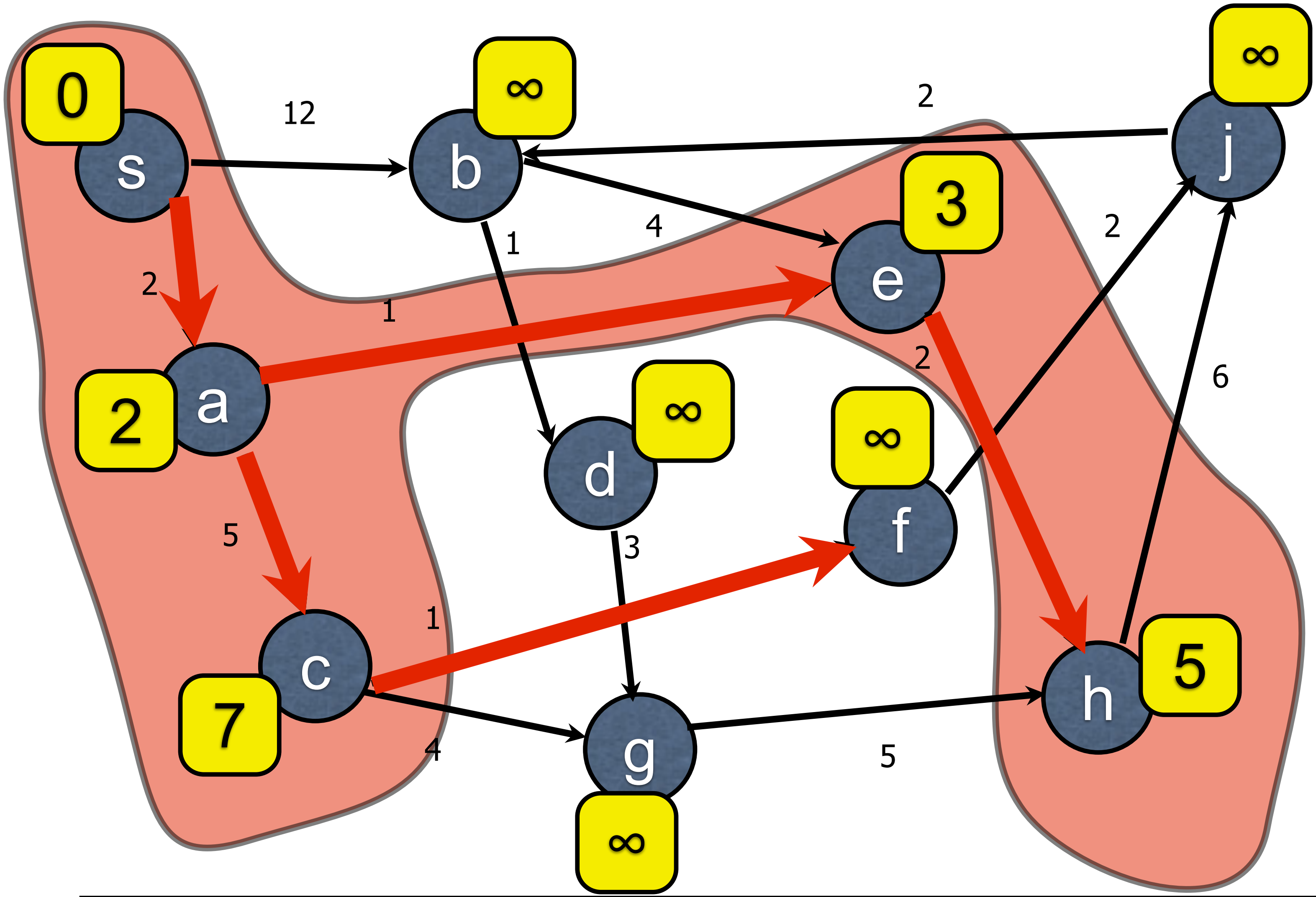
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



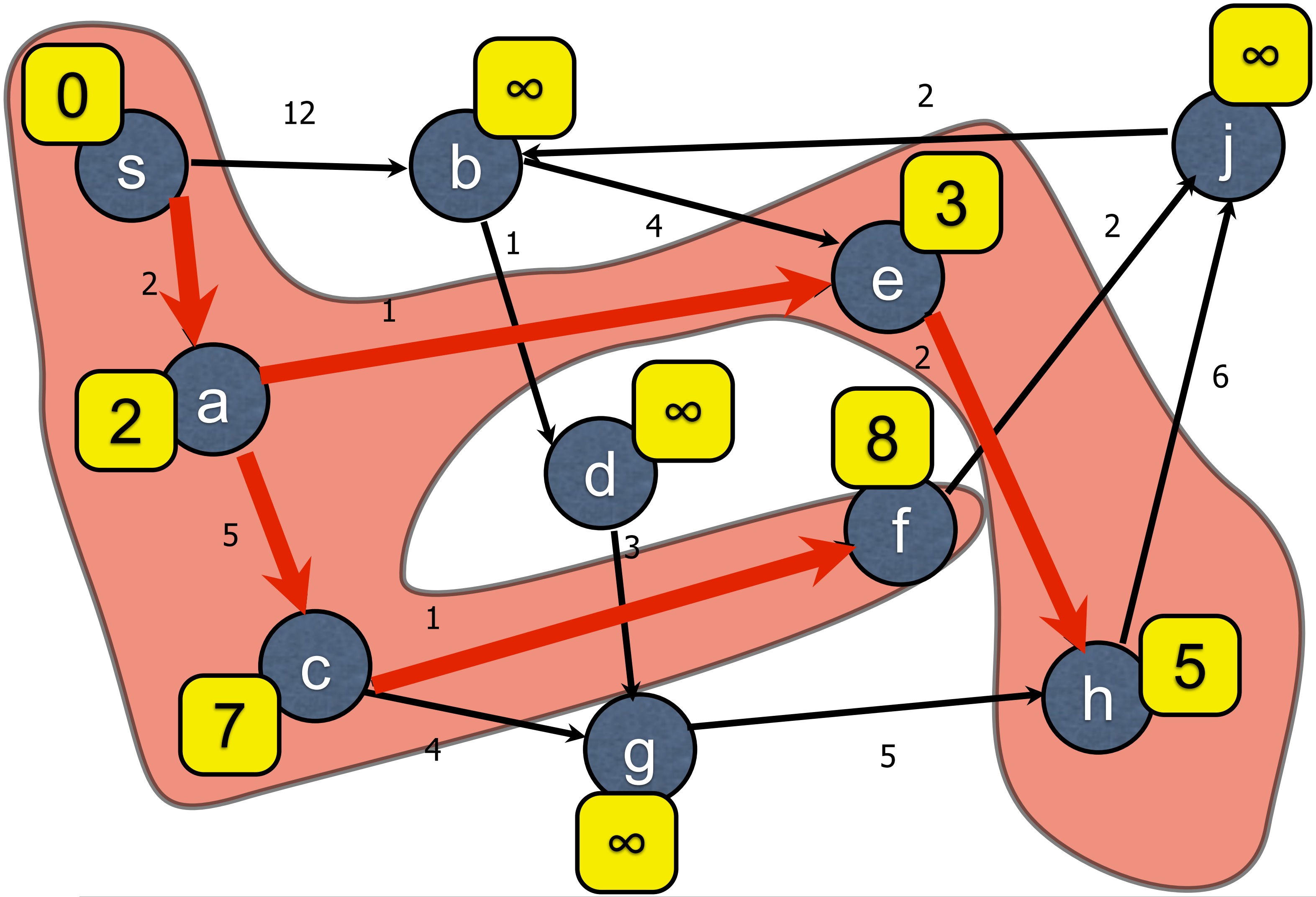

while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



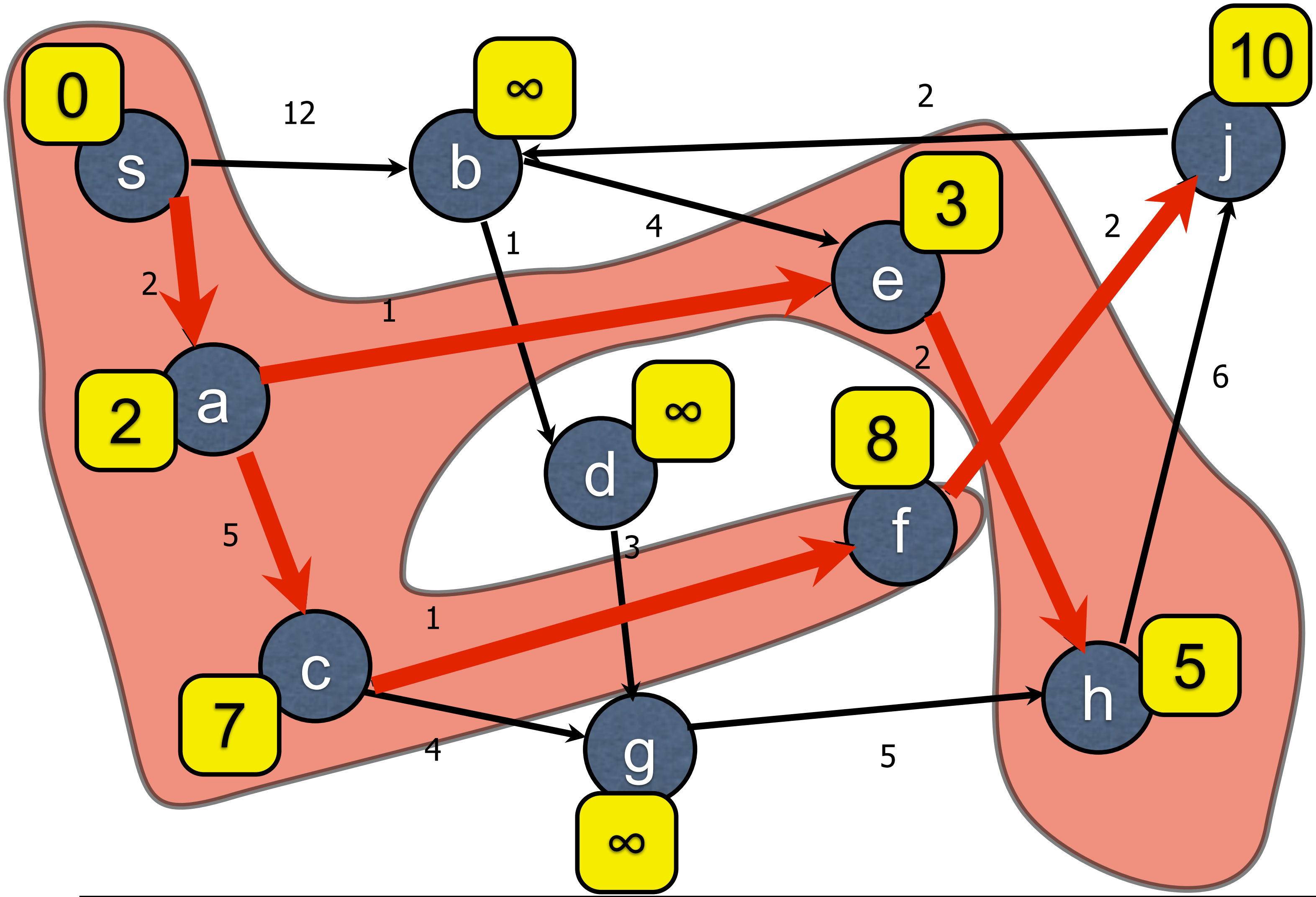

while there is edge from discovered vertex to undiscovered vertex,
 let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
 set $d(v) = d(u) + c_{u,v}$, mark v discovered

Dijkstra's Algorithm




while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + l_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra’s Algorithm

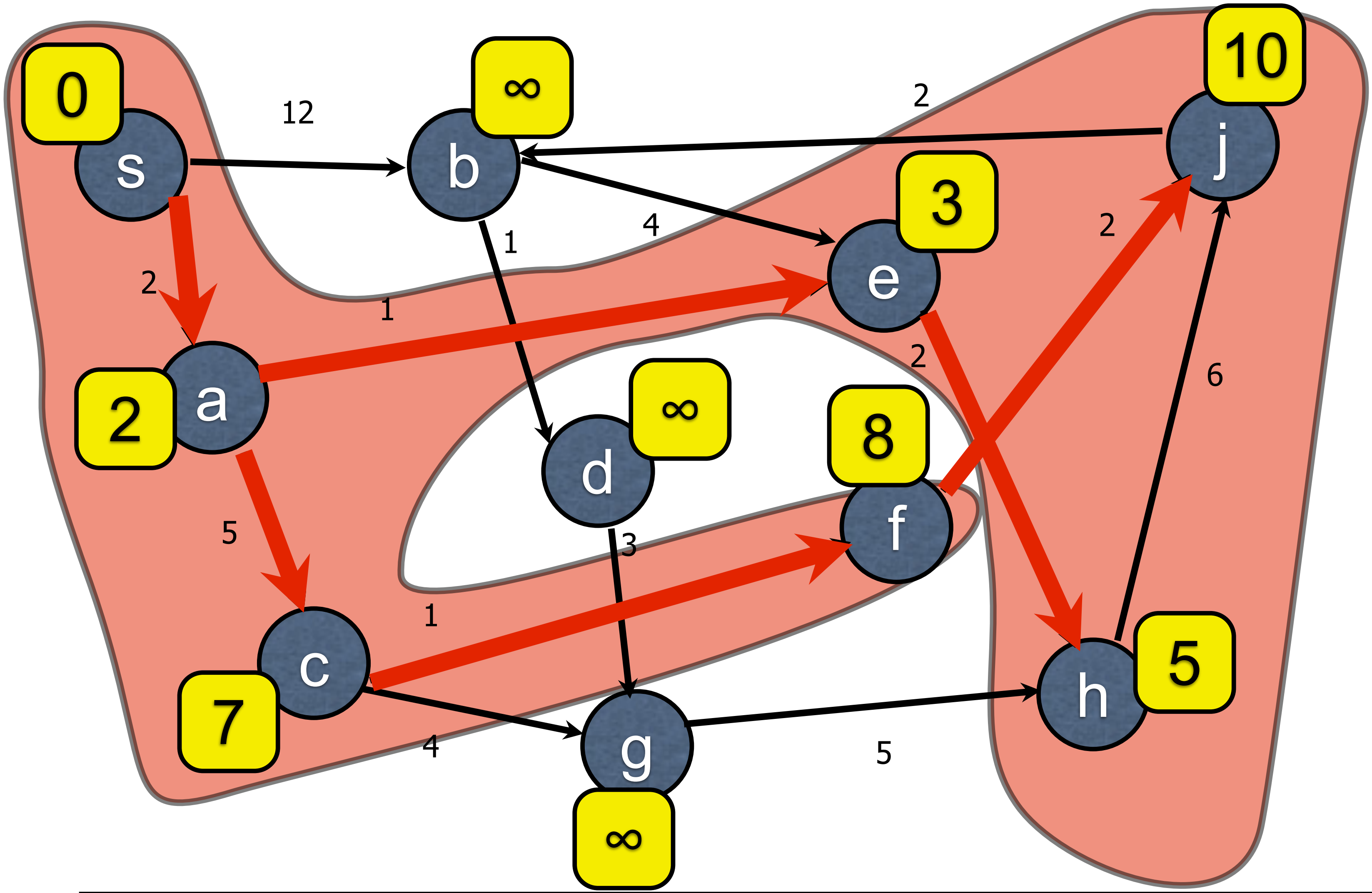



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm

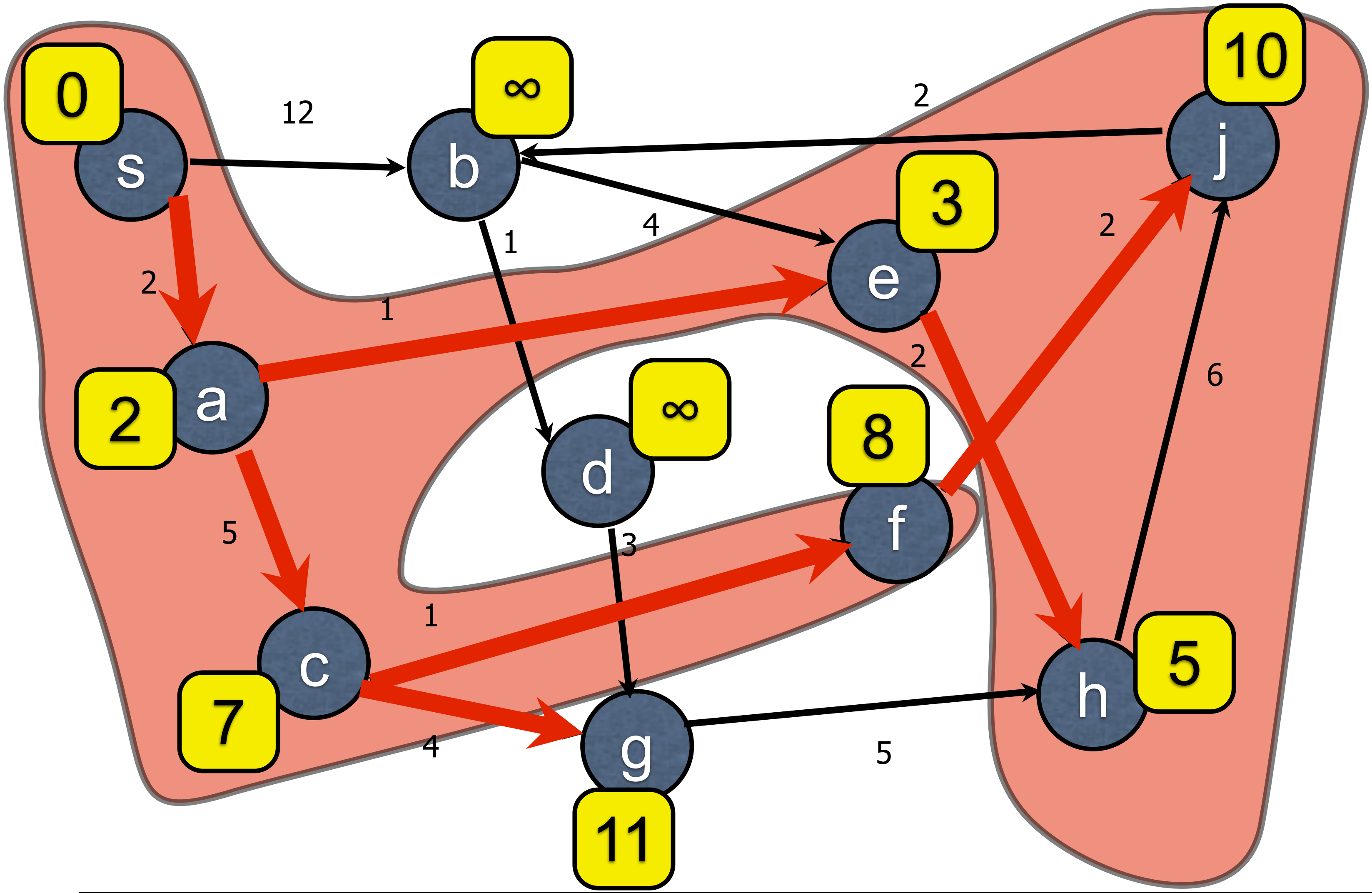



while there is edge from discovered vertex to undiscovered vertex,

let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$

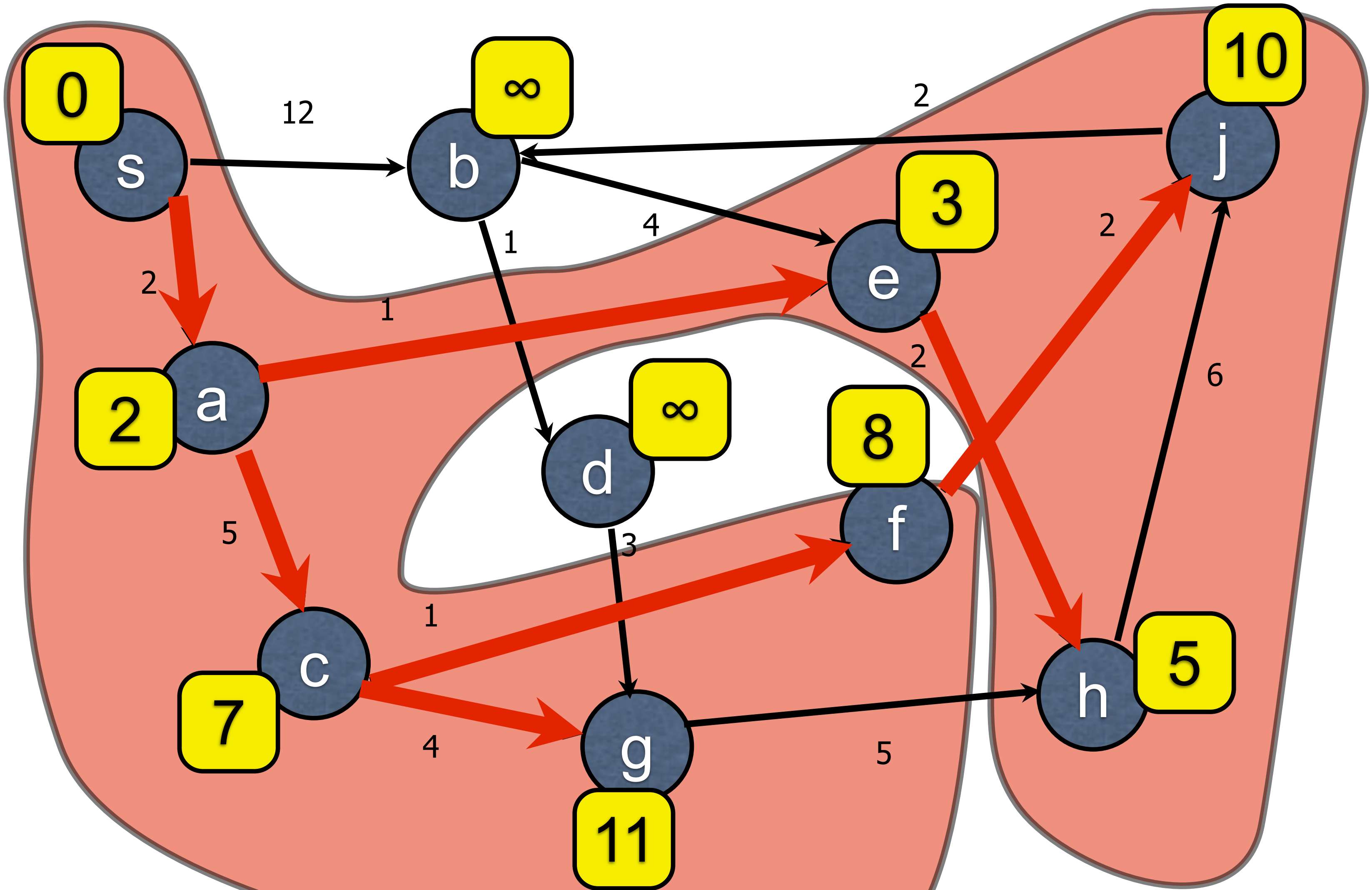
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



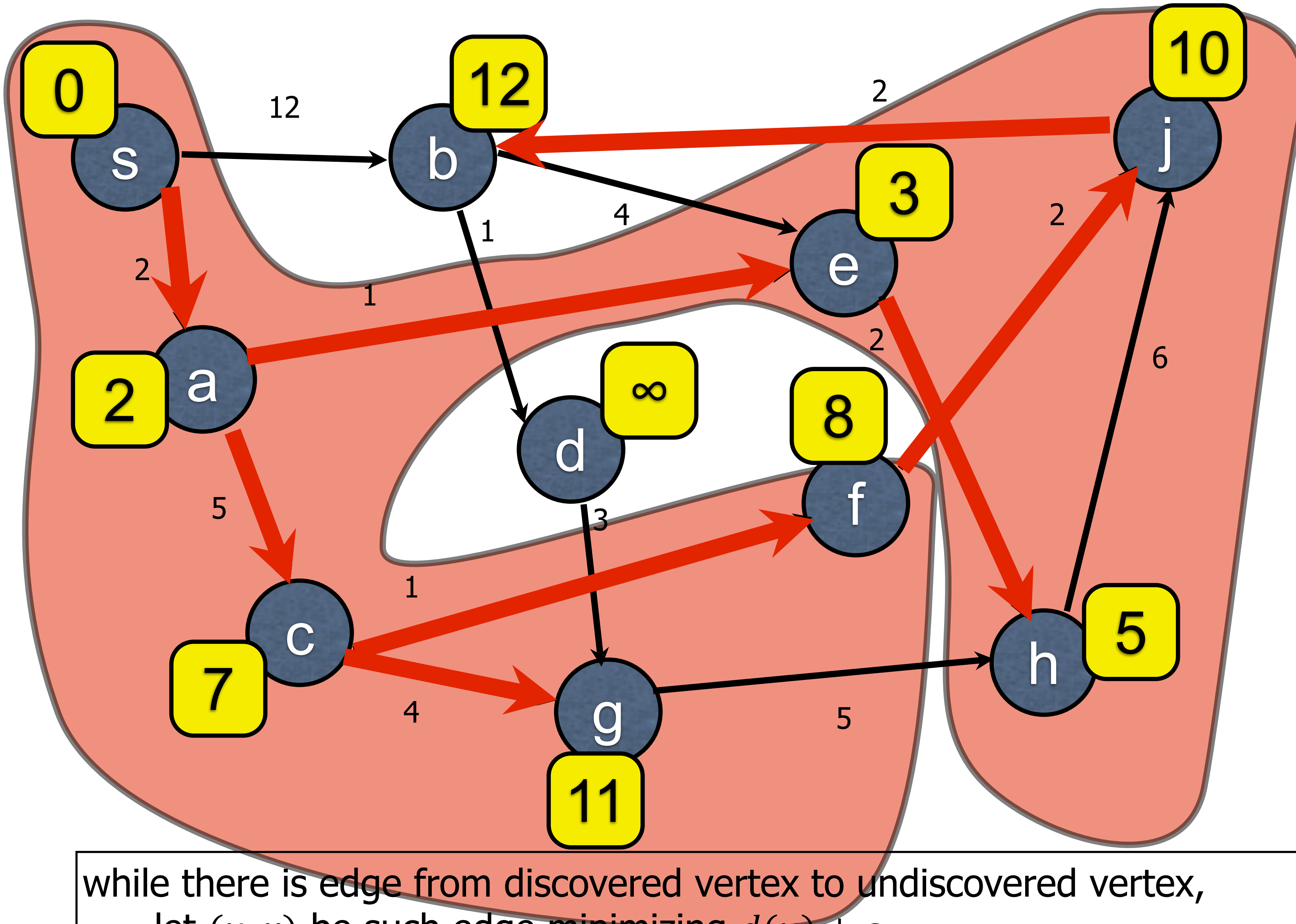

while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



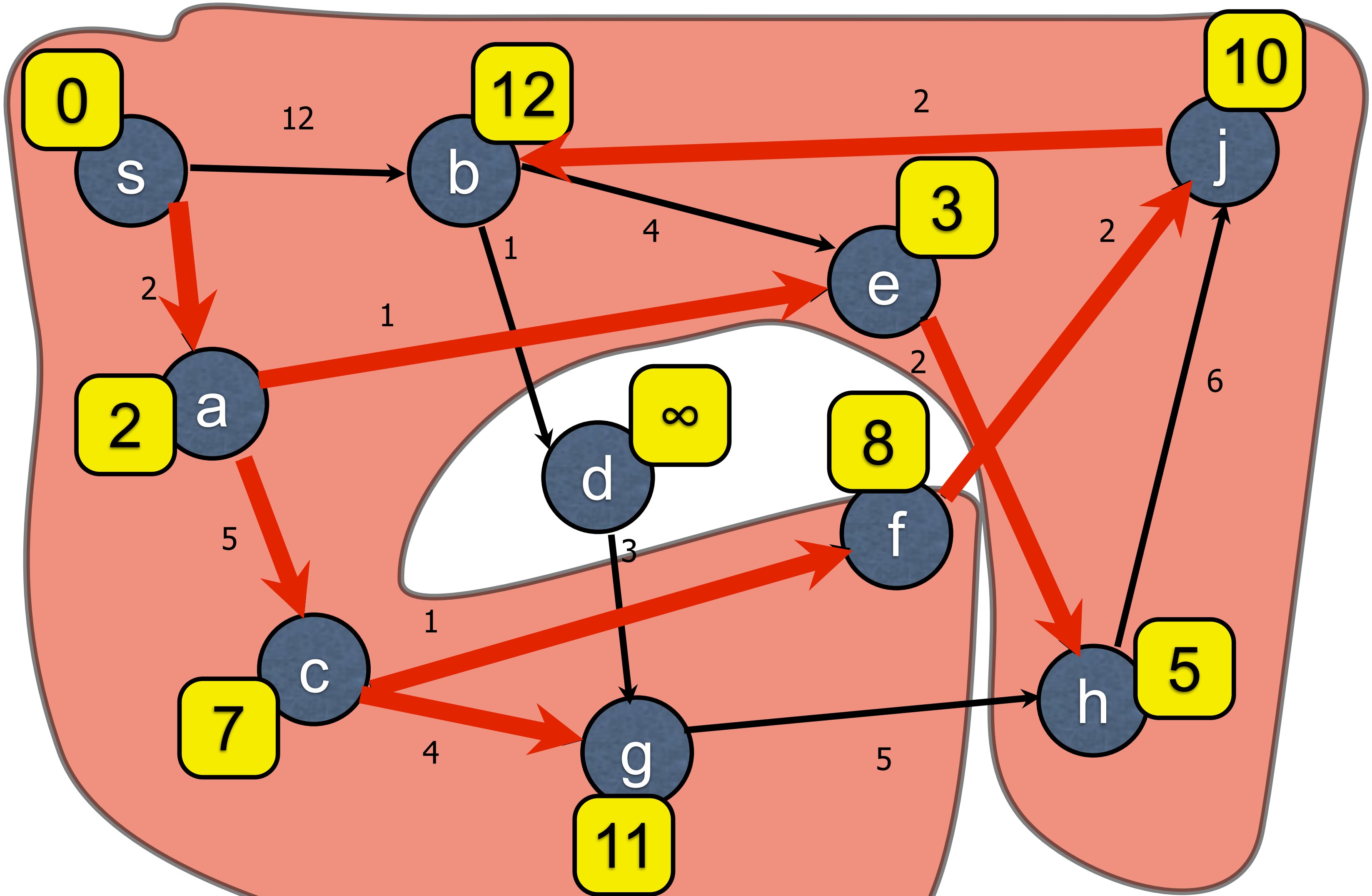

while there is edge from discovered vertex to undiscovered vertex,
 let $(u,v)$ be such edge minimizing $d(u) + c_{u,v}$
 set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



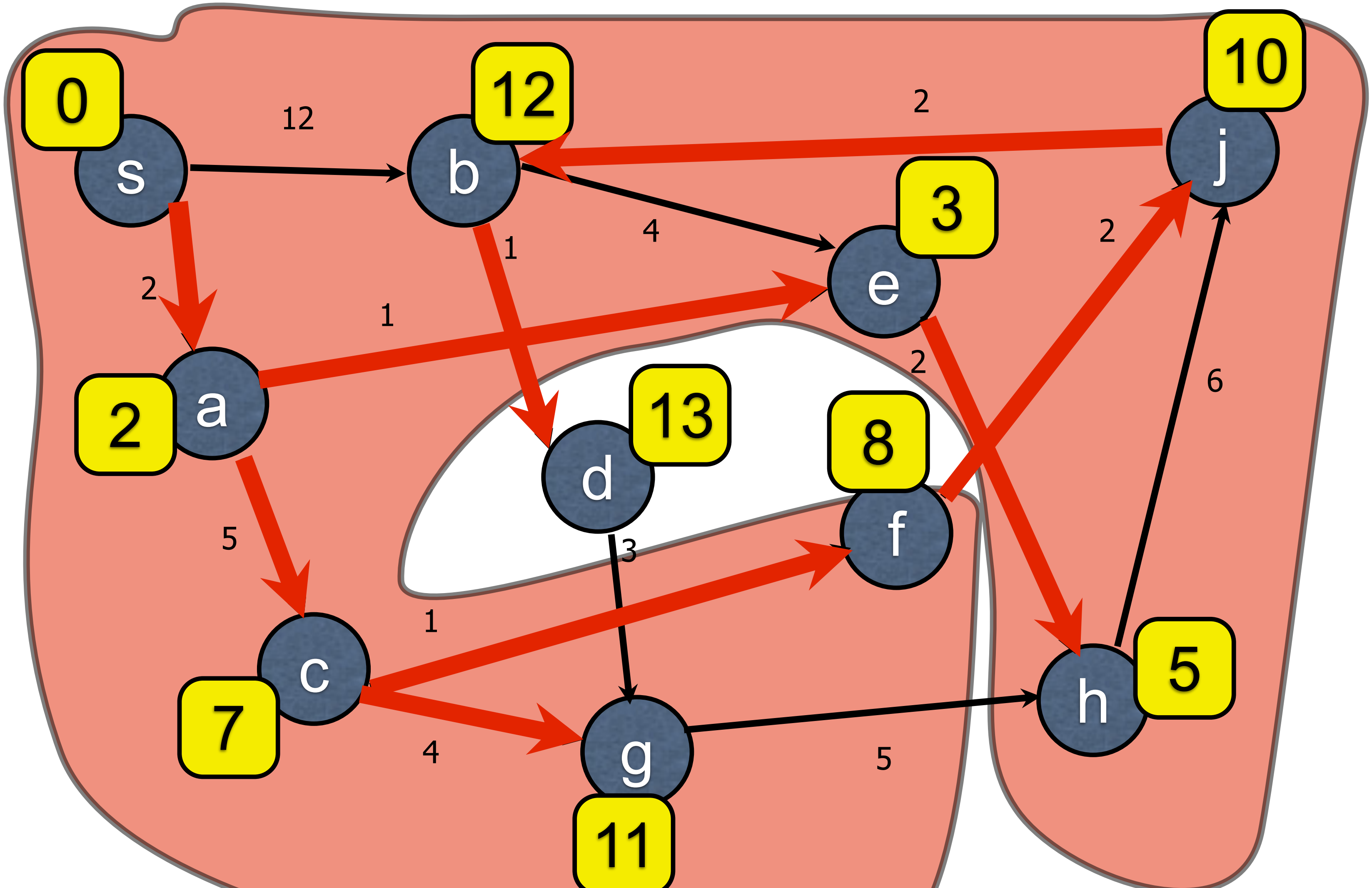

while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm



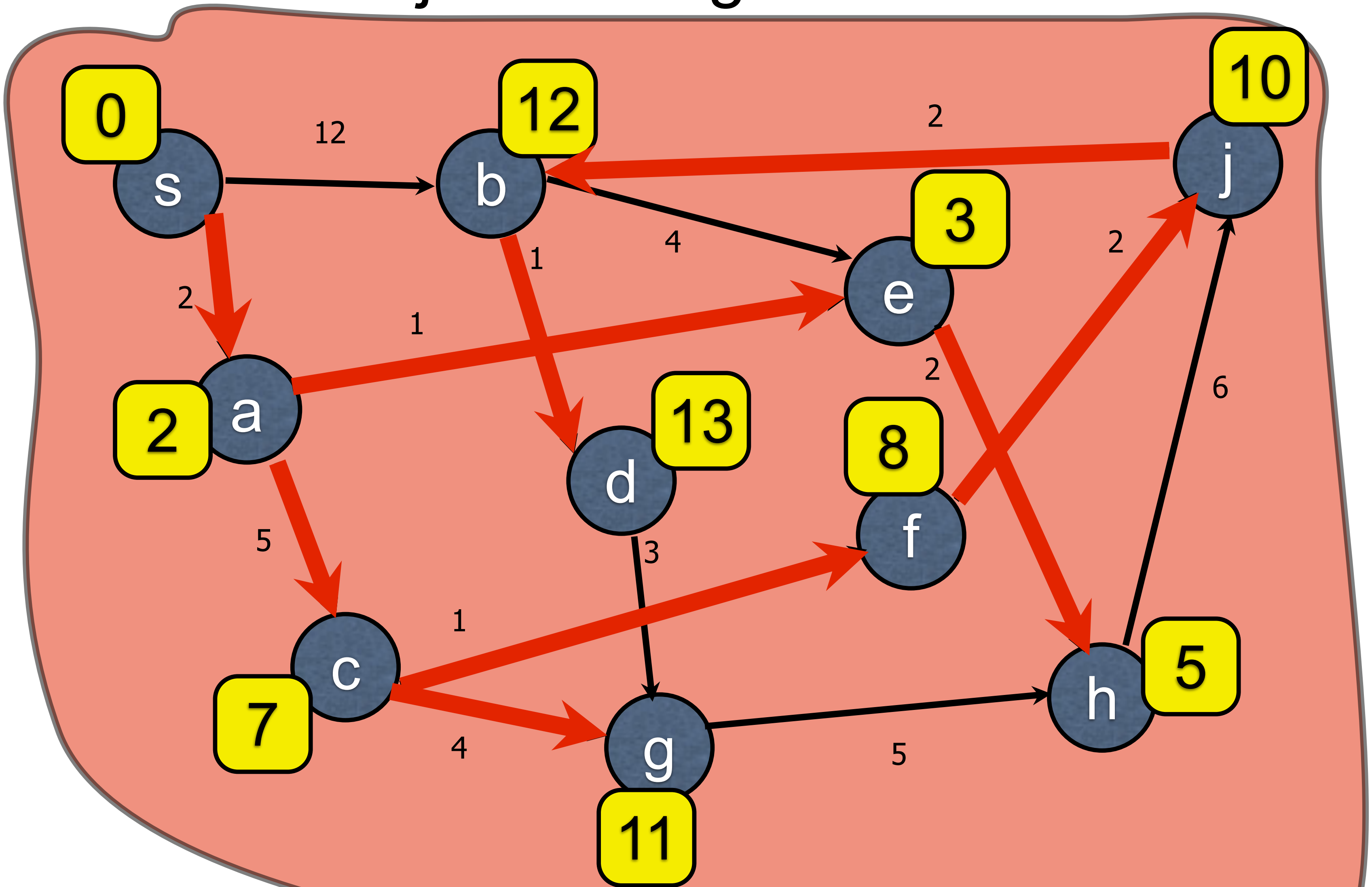

while there is edge from discovered vertex to undiscovered vertex,
    let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
    set $d(v) = \ d(u) + c_{u,v}$, mark v discovered

# Dijkstra’s Algorithm




while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Dijkstra's Algorithm




while there is edge from discovered vertex to undiscovered vertex,
let $(u, v)$ be such edge minimizing $d(u) + c_{u,v}$
set $d(v) = d(u) + c_{u,v}$, mark v discovered

# Disjkstra's Algorithm: Correctness

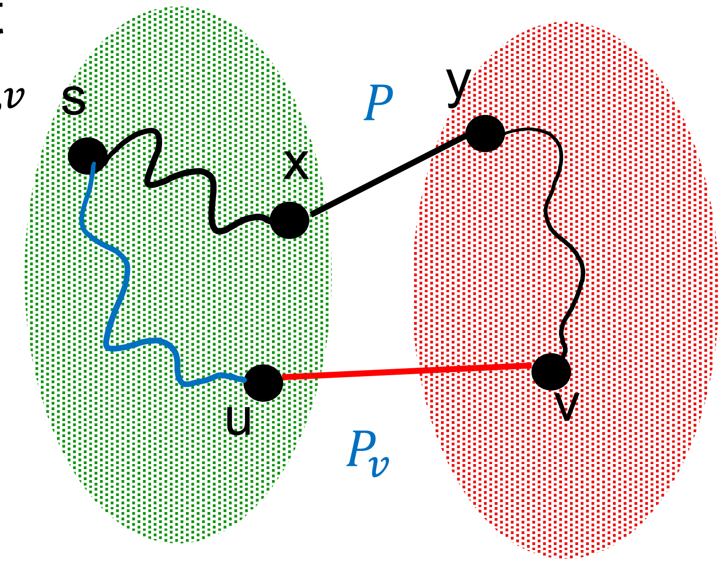
Let S be the set of discovered vertices, P(k)=``If $|S| = k$, then for all discovered vertices $v \in S$, $d(v)$ is the shortest path from s to v.

Base Case: This is always true when $S = \{s\}$.

IH: P(k) holds

IS: Say $v$ is the k+1-st vertex that we discover using edge (u,v) and we set

$$d(v) = d(u) + c_{u,v}$$




Call the path to v, $P_v$. If $P_v$ is not the Shortest path, there is a shorter path $P$

Consider the first time that P leaves S (say with edge (x,y)).

S -> x has weight (at least) $d(x)$

So, $c(P) \geq d(x) + c_{x,y} \geq d(u) + c_{u,v} = d(v) = c(P_v)$.

A contradiction.

# Remarks on Dijkstra’s Algorithm

- Algorithm also produces a tree of shortest paths to s following Parent links
- Algorithm works on directed graph (with nonnegative weights)

- The algorithm fails with negative edge weights.
  - e.g., some airline tickets

Why does it fail?

- Dijkstra’s algorithm is similar to BFS:
  - Subtitute every edge with $c_e = k$ with a path of length k, then run BFS.

# Implementing Dijkstra’s Algorithm

Priority Queue: Elements each with an associated key Operations

- Insert
- Find-min
  - Return the element with the smallest key
- Delete-min
  - Return the element with the smallest key and delete it from the data structure
- Decrease-key
  - Decrease the key value of some element

Implementations

Arrays:

- O(n) time find/delete-min,
- O(1) time insert/decrease key

Binary Heaps:

- O(log n) time insert/decrease-key/delete-min,
- O(1) time find-min

# Dijkstra's Algorithm

Runs in O((n+m)log n).

```
Dijkstra(G, c, s) {
   foreach (v ∈ V) d[v] ← ∞ //This is the key of node v
   d[s] ← 0
   foreach (v ∈ V) insert v onto a priority queue Q
   Initialize set of explored nodes S ← {s}

   while (Q is not empty) {
      u ← delete min element from Q
      S ← S ∪ { u }
      foreach (edge e = (u, v) incident to u)
         if ((v ∉ S) and (d[u]+c_e < d[v]))
            d[v] ← d[u] + c_e
            Decrease key of v to d[v].
            Parent(v) ← u
}
```

$O(n)$ of delete min, each in O(log n)

$O(m)$ of decrease key, each runs in $O(\log n)$

# Algorithm Design by Induction

# Maximum Consecutive Subsequence

Problem: Given a sequence $x_1, \dots, x_n$ of integers (not necessarily positive),

Goal: Find a subsequence of consecutive elements s.t., the sum of its numbers is maximum.

1 -3 [ 7 -2 -3 8 ] -10 1 -7

Applications: Figuring out the highest interest rate period in stock market

# Brute Force Approach

Try all consecutive subsequences of the input sequence.

There are $\binom{n}{2} = \Theta(n^2)$ such sequences.

We can compute the sum of numbers in each such sequence in $O(n)$ steps.

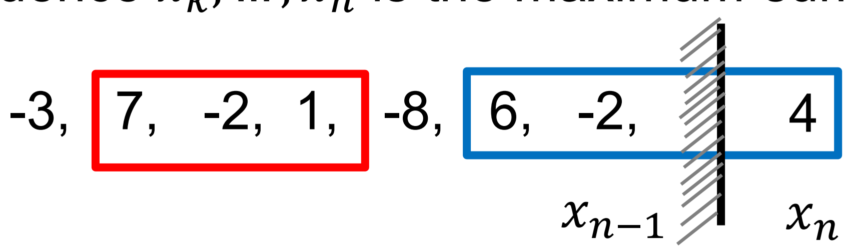
So, the ALG runs in $O(n^3)$.

With a clever loop we can do this in $O(n^2)$.
But, can we solve in linear time?

# First Attempt (Induction)

Suppose we can find the maximum-sum subsequence of $x_1, \dots, x_{n-1}$. Say it is $x_i, \dots, x_j$

- If $x_n < 0$ then it does not belong to the largest subsequence. So, we can output $x_i, \dots, x_j$
- Suppose $x_n > 0$.
  - If $j = n - 1$ then $x_i, \dots, x_n$ is the maximum-sum subsequence.
  - If $j < n - 1$ there are two possibilities
    1) $x_i, \dots, x_j$ is still the maximum-sum subsequence
    2) A sequence $x_k, \dots, x_n$ is the maximum-sum subseqence



# Second Attempt (Strengthing Ind Hyp)

Stronger Ind Hypothesis: Given $x_1, \ldots, x_{n-1}$ we can compute the maximum-sum subsequence, and the maximum-sum suffix subsequence.

Can be empty

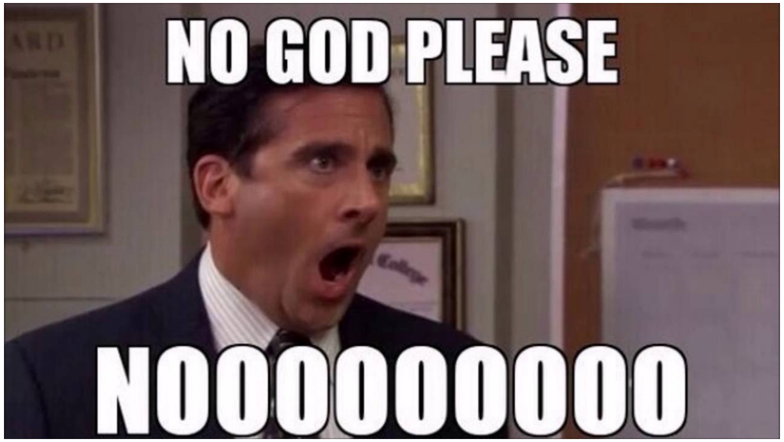
-3, [7, -2, 1,] -8, [6, -2]

$x_i$ $x_j$ $x_k$ $x_{n-1}$

Say $\boldsymbol{x_i, \ldots, x_j}$ is the maximum-sum and $x_k, \ldots, x_{n-1}$ is the maximum-sum suffix subsequences.

- If $x_k + \cdots + x_{n-1} + x_n > x_i + \cdots + x_j$ then $x_k, \ldots, x_n$ will be the new maximum-sum subsequence

# Are we done?



# Updating Max Suffix Subsequence

-3, 7, -2, 1, -8, 6, -2, | 4

$x_n$

Say $x_k, \dots, x_{n-1}$ is the maximum-sum suffix subsequences of $x_1, \dots, x_{n-1}$.

- If $x_k + \cdots + x_n \geq 0$ then,
  $x_k, \dots, x_n$ is the new maximum-sum suffix subsequence

- Otherwise,
  The new maximum-sum suffix is the empty string.

# Maximum Sum Subsequence ALG

```
Initialize S=0 (Sum of numbers in Maximum Subseq)
Initialize U=0 (Sum of numbers in Maximum Suffix)
for  (i=1 to n) {
    if (x[i] + U > S)
        S = x[i] + U

    if (x[i] + U > 0)
        U = x[i] + U
    else
        U = 0
}
Output S.
```

-3 7 -2 1 -8 6 -2 4

# Pf of Correct: Maximum Sum Subseq

Ind Hypo: Suppose

- $x_i, \dots, x_j$ is the max-sum-subseq of $x_1, \dots, x_{n-1}$
- $x_k, \dots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \dots, x_{n-1}$

Ind Step: Suppose $x_a, \dots, x_b$ is the max-sum-subseq of $x_1, \dots, x_n$

Case 1 ($b < n$): $x_a, \dots, x_b$ is also the max-sum-subseq of $x_1, \dots, x_{n-1}$
So, $a = i, b = j$ and the algorithm correctly outputs OPT

Case 2 ($b = n$): We must have $x_a, \dots, x_{b-1}$ is the max-suff-sum of $x_1, \dots, x_{n-1}$.
If not, then

$$x_k + \cdots x_{n-1} > x_a + \cdots + x_{n-1}$$

So, $x_k + \cdots + x_n > x_a + \cdots + x_b$ which is a contradiction.
Therefore, $a = k$ and the algorithm correctly outputs OPT

Special Cases (You don't need to mention if follows from above):

- The max-suffix-sum is empty string
- There are multiple maximum sum subsequences.

# Pf of Correct: Max-Sum Suff Subseq

Ind Hypo: Suppose

- $x_i, \dots, x_j$ is the max-sum-subseq of $x_1, \dots, x_{n-1}$
- $x_k, \dots, x_{n-1}$ is the max-suffix-sum-sub of $x_1, \dots, x_{n-1}$

Ind Step: Suppose $x_a, \dots, x_n$ is the max-suffix-sum-subseq of $x_1, \dots, x_n$
Note that we may also have an empty sequence

Case 1 (OPT is empty): Then, we must have $x_k + \cdots + x_n < 0$. So the algorithm correctly finds max-suffix-sum subsequence.

Case 2 ($x_a, \dots, x_n$ is nonempty): We must have $x_a + \cdots + x_n \geq 0$.
Also, $x_a, \dots, x_{n-1}$ must be the max-suffix-sum of $x_1, \dots, x_{n-1}$. If not,

$$x_a + \cdots + x_{n-1} < x_k + \cdots + x_{n-1}$$

which implies $x_a + \cdots + x_n < x_k + \cdots + x_n$ which is a contradiction.

Therefore, $a = k$. So, the algorithm correctly finds max-suffix-sum subseqence.

# Summary

- Try to reduce an instance of size n to smaller instances
  - Never solve a problem twice
- Before designing an algorithm study properties of optimum solution
- If ordinary induction fails, you may need to strengthen the induction hypothesis