Approximation Algorithms

Why Approximation Algorithms

- Problems that we cannot find an optimal solution in a polynomial time
- Need to find a near-optimal solution:
  - Heuristic
  - Approximation algorithms:
    - This gives us a guarantee approximation ratio

How to use it

- Your advisers/bosses give you a computationally hard problem. Here are two scenarios:
  - No knowledge about approximation:
    - Spend a few months looking for an optimal solution
    - Come to their office and confess that you cannot do it
    - Get fired
  - Knowledge about approximation:

How to use it(cont)

- Knowledge about approximation
  - Show your boss that this is a NP-complete (NP-hard) problem
  - There does not exist any polynomial time algorithm to find an exact solution
  - Propose a good algorithm (either heuristic or approximation) to find a near-optimal solution
  - Better yet, prove the approximation ratio

Approximation algorithm

- Most dynamic programming and greedy algorithms that we have seen had 3 properties:
  - Deterministic
  - Always Correct
  - Worst case bounded by a polynomial function of Input Size
  - $O(n)$, $O(nlog_n)$, $O(n^2)$, $O(n^3)$
- $O(2^n)$, $O(n!)$

Relax it

- [An Example]

Definition of Approximation Algorithms

- **Definition**: An *α-approximation algorithm* is a polynomial-time algorithm which always produces a solution of value within α times the value of an optimal solution.

  That is, for any instance of the problem

  $$Z_{algo} / Z_{opt} \leq \alpha \, , \quad (\textit{for a minimization problem})$$

  where Z_{algo} is the cost of the algorithm output,

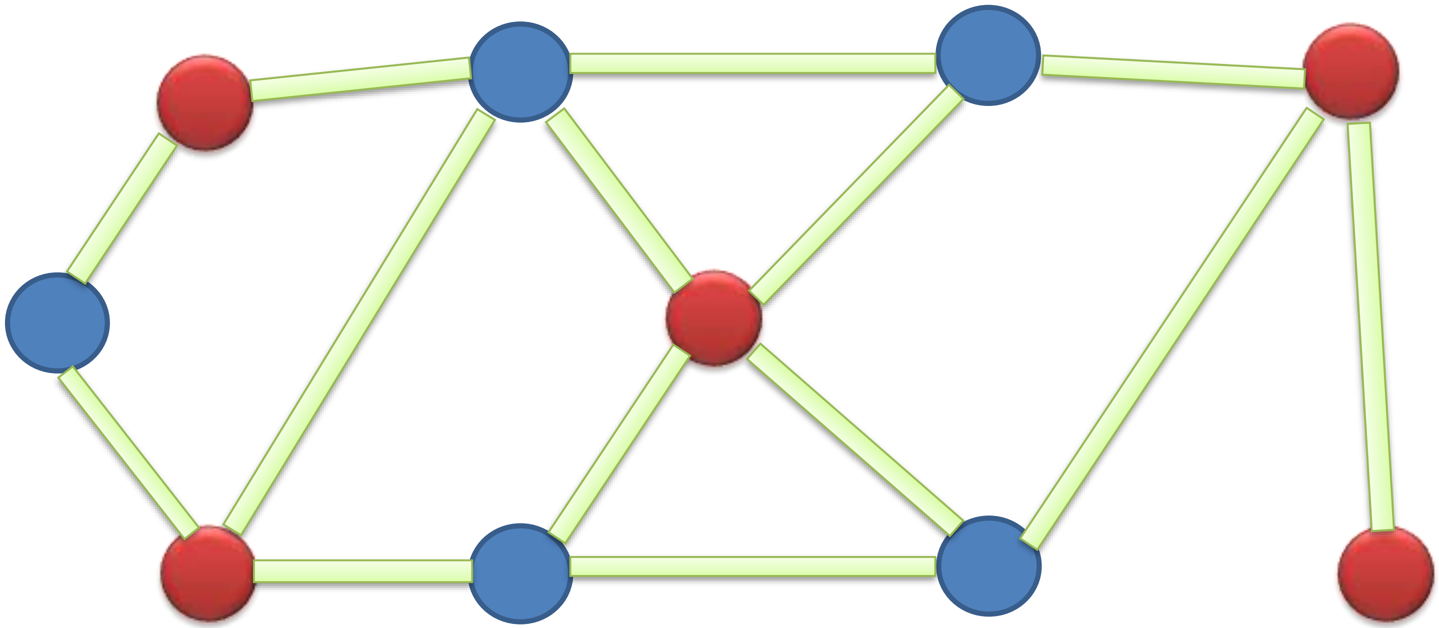
  Z_{opt} is the cost of an optimal solution.

- α is called the *approximation guarantee* (or *factor*) of the algorithm.

Some examples:

- Vertex cover problem.
- Traveling salesman problem.
- Center Selection Method
- Knapsack Problem

VERTEX COVER PROBLEM

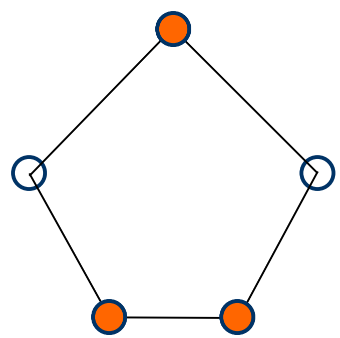


Vertex Cover Problem

- In the mathematical discipline of graph theory, “A *vertex cover* (sometimes node cover) of a graph is a subset of vertices which “*covers*” every edge.
- An edge is *covered* if one of its endpoint is chosen.
- In other words “A *vertex cover* for a graph G is a set of vertices incident to every edge in G.”
- The *vertex cover problem*: What is the minimum size vertex cover in G?

Vertex Cover Problem

Problem: Given graph G = (V, E), find *smallest* $V' \subseteq V$ *s. t.* if (u, v) $\in$ E, then u $\in$ V′ or v $\in$ V′ or both.



- Vertex cover problem is to find a vertex cover of minimum size in a given undirected graph.
- We call such vertex cover an optimal vertex cover.
- This problem is the optimization version of an NP-complete decision problem

Vertex Cover : Algorithm(1)

APPROX-VERTEX-COVER

1: $C \leftarrow \emptyset$;

2: $E' \leftarrow E$

3: **while** $E' \neq \emptyset$; **do**

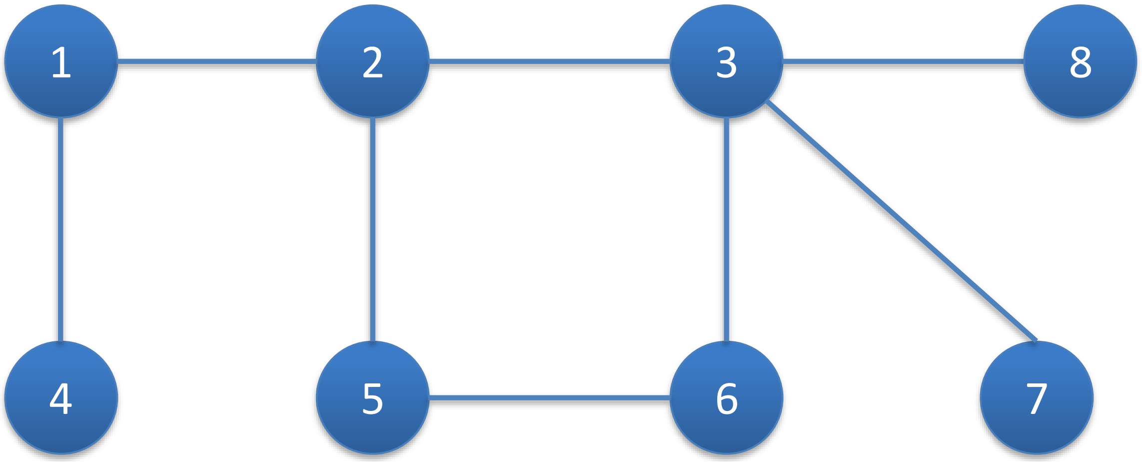
4: let (u, v) be an arbitrary edge of E'

5: $C \leftarrow C \cup \{(u, v)\}$

6: remove from E' all edges incident on either u or v

7: **end while**

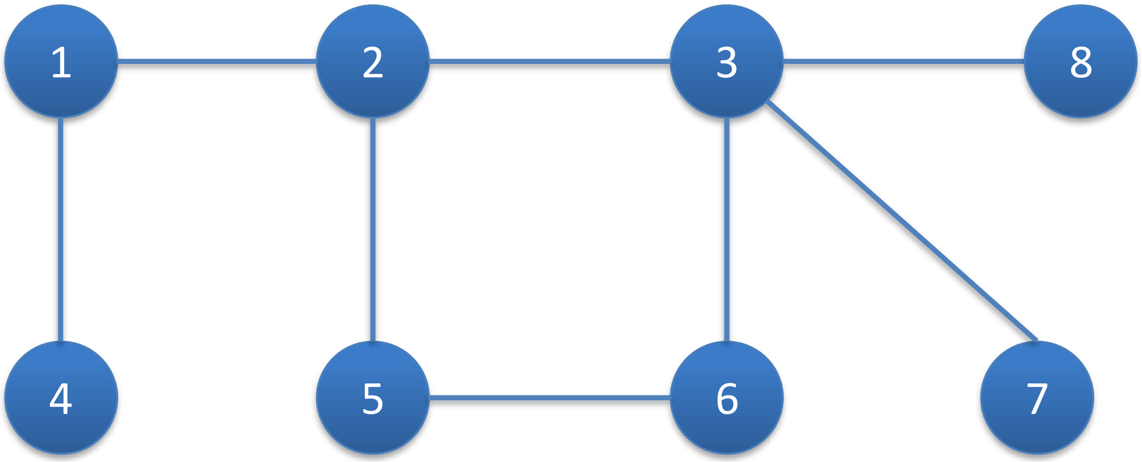
Algorithm(1): Example



Initially C = Ø

E′ = {(1,2) (2,3) (1,4) (2,5) (3,6) (5,6) (3,7) (3,8)}

Algorithm(1): Example

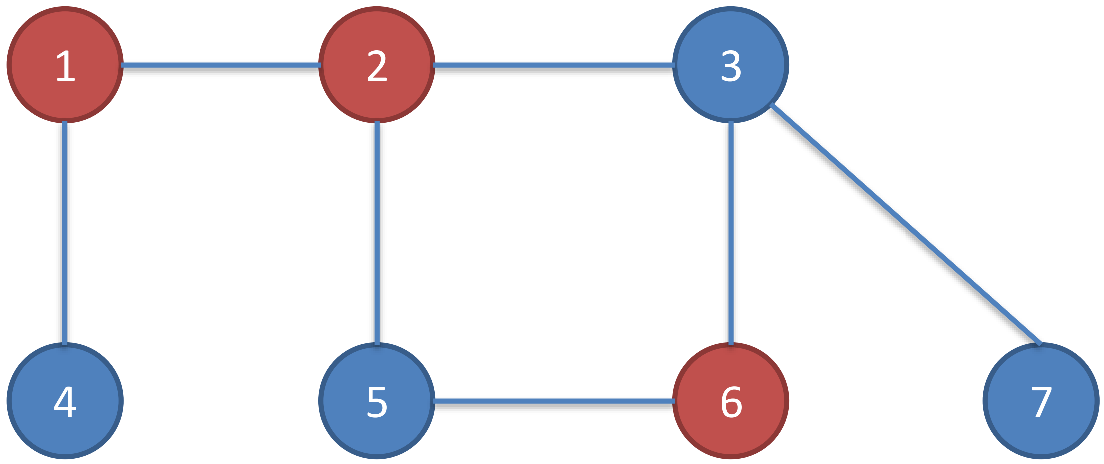


C =

1	2	3	6

E′ = ~~{(3,6) (5,6) (3,7) (3,8)}~~

Algorithm(1): Example (Cont…)

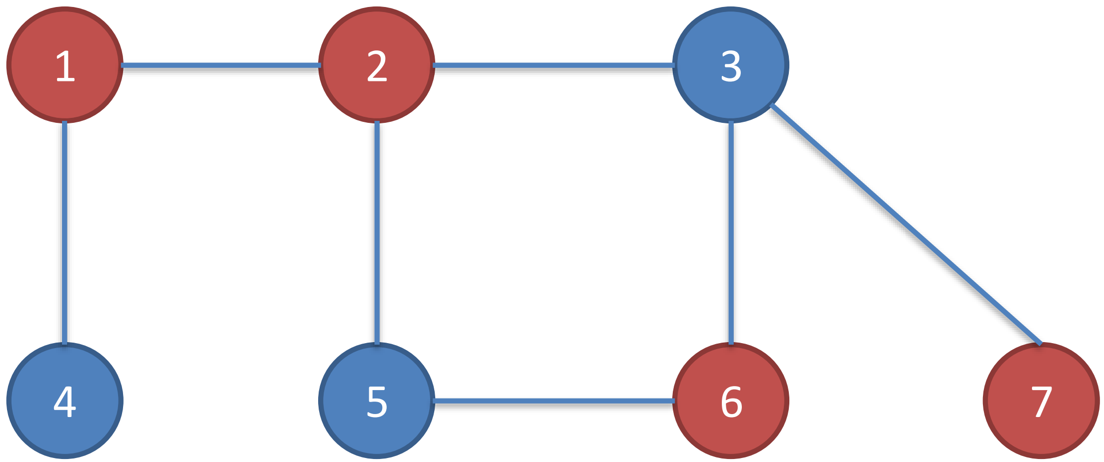


Are the red vertices a vertex-cover?

No………. why?

Edge (3, 7) is not covered by it.

Algorithm(1): Example (Cont…)



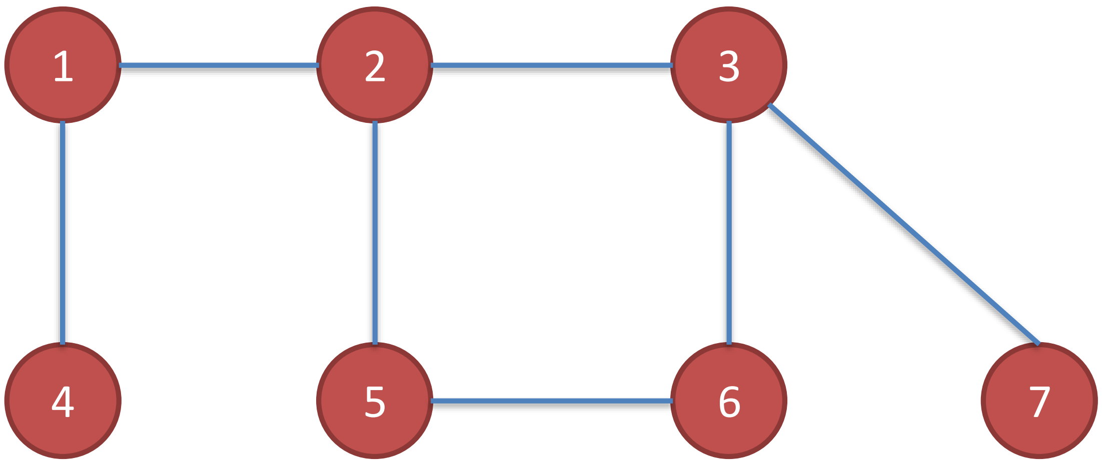
Are the red vertices a vertex-cover?

Yes

What is the size?

Size = 4

Algorithm(1): Example (Cont...)



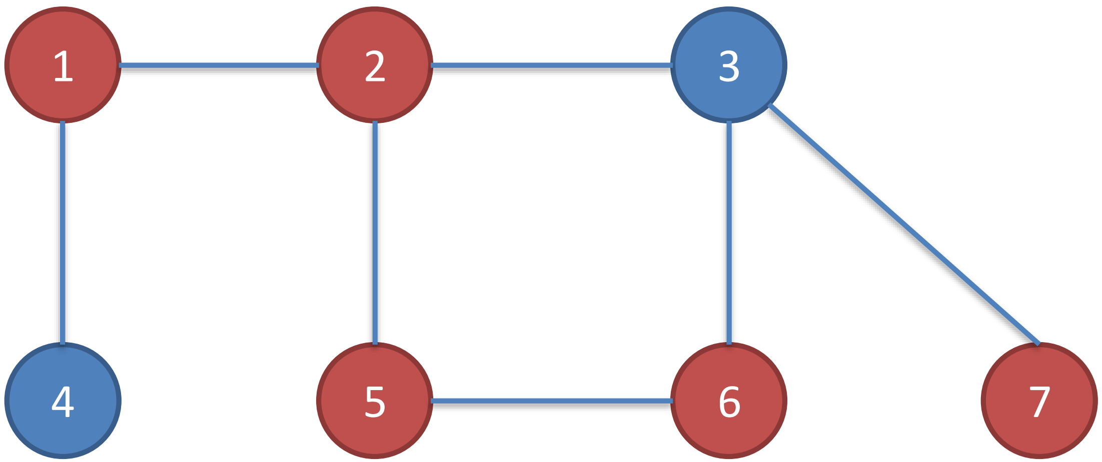
Are the red vertices a vertex-cover?

Of course……..

What is the size?

Size = 7

Algorithm(1): Example (Cont…)



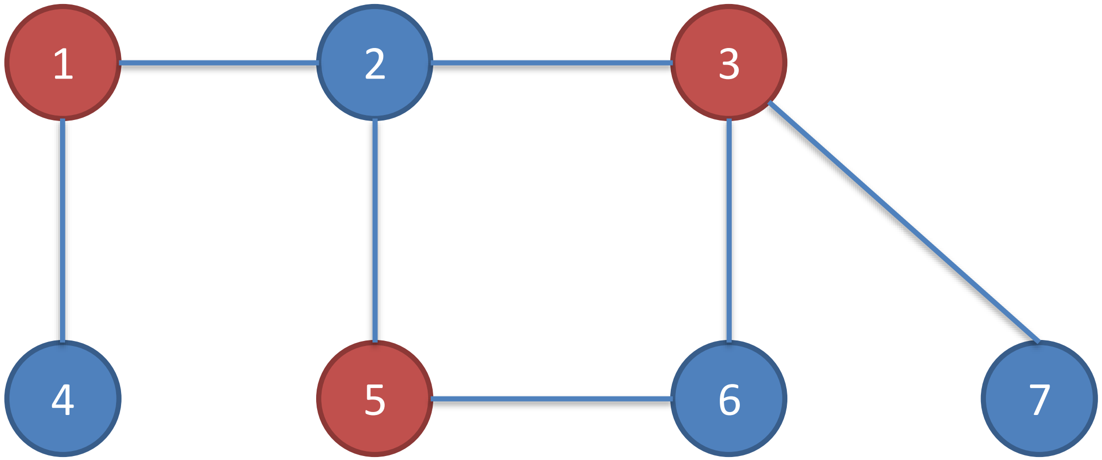
Are the red vertices a vertex-cover?

Yes

What is the size?

Size = 5

Algorithm(1): Example (Cont…)



Are the red vertices a vertex-cover?

Yes

What is the size?

Size = 3

Conclusion

- A set of **vertices** such that each edge of the graph is incident to at least one **vertex** of the set, is called the vertex cover.
- Approximation algorithm always produce optimal solution.

An approximation algorithm for TSP

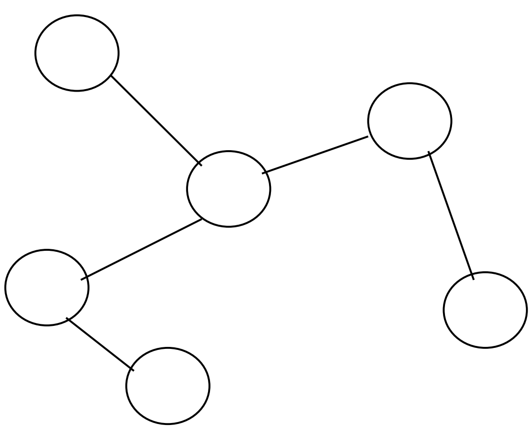
Given an instance for TSP problem,

1. Find a minimum spanning tree (MST) for that instance.

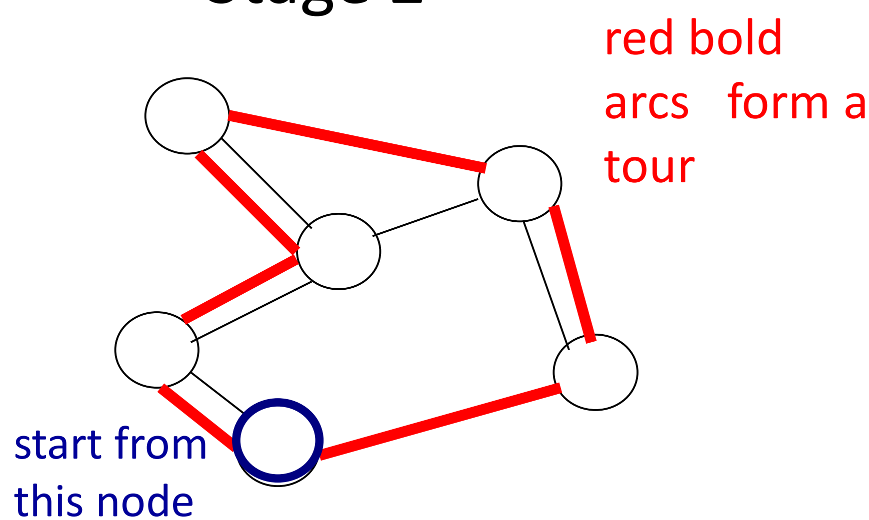
2. To get a tour, start from any node and traverse the arcs of MST by taking shortcuts when necessary.

Example:

Stage 1



Stage 2



Approximation guarantee for the algorithm

- In many situations, it is reasonable to assume that triangle inequality holds for the cost function $c: E \rightarrow R$ defined on the arcs of network $G=(V,E)$:

$$c_{uw} \leq c_{uv} + c_{vw} \quad \text{for any } u, v, w \in V$$