12.3.2

A recursive algorithm for Max Independent Set in a Graph

A Recursive Algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}$.
For a vertex u let $N(u)$ be its neighbors.

Observation

v_1: vertex in the graph.
One of the following two cases is true

Case 1 v_1 *is in <u>some</u> maximum independent set.*

Case 2 v_1 *is in <u>no</u> maximum independent set.*

We can try <u>both cases</u> to "reduce" the size of the problem

A Recursive Algorithm

Let $V = \{v_1, v_2, \ldots, v_n\}$.
For a vertex u let $N(u)$ be its neighbors.

Observation

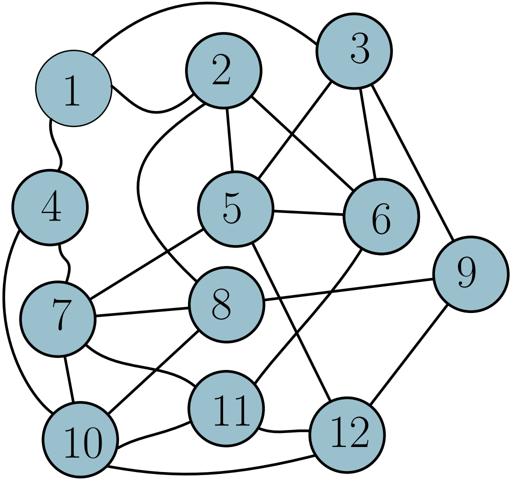
v_1: vertex in the graph.
One of the following two cases is true

- *Case 1* v_1 *is in* <u>*some*</u> *maximum independent set.*
- *Case 2* v_1 *is in* <u>*no*</u> *maximum independent set.*

We can try <u>*both cases*</u> *to "reduce" the size of the problem*

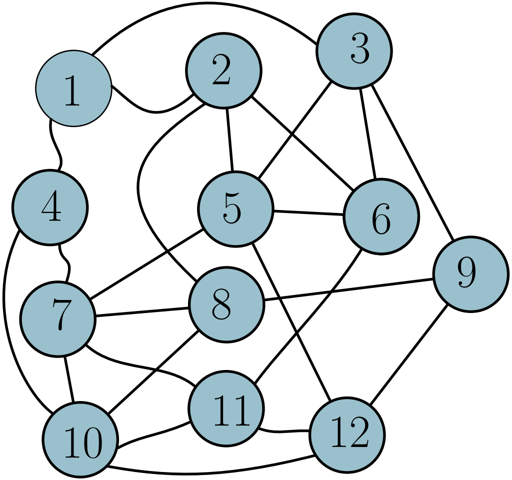
Removing a vertex (say 5)

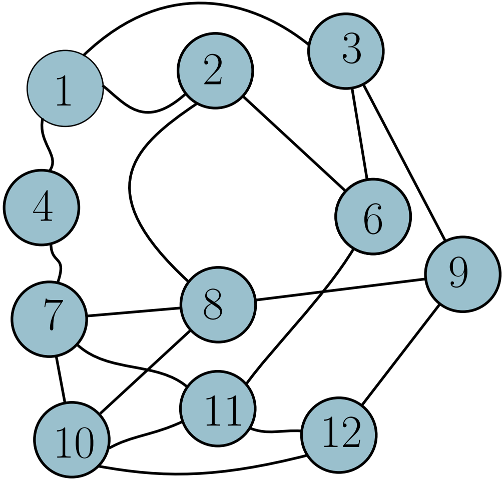
Because it is NOT in the independent set



Removing a vertex (say 5)

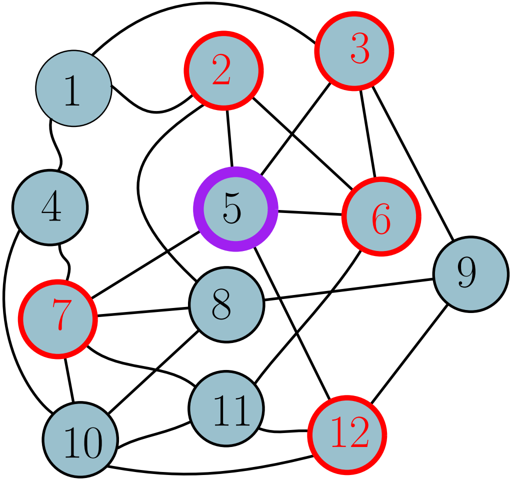
Because it is NOT in the independent set





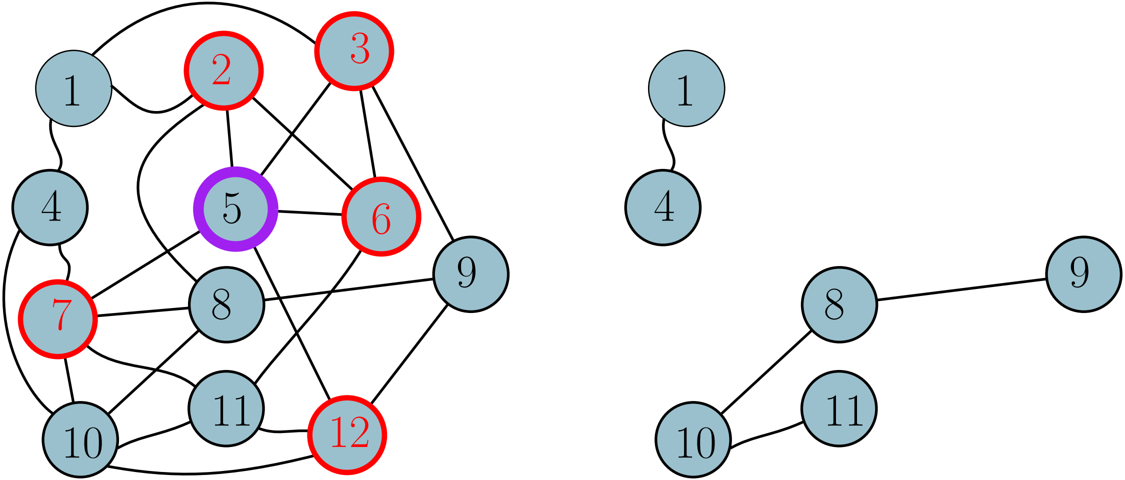
Removing a vertex (say 5) and its neighbors

Because it **is** in the independent set



Removing a vertex (say 5) and its neighbors

Because it **is** in the independent set



A Recursive Algorithm: The two possibilities

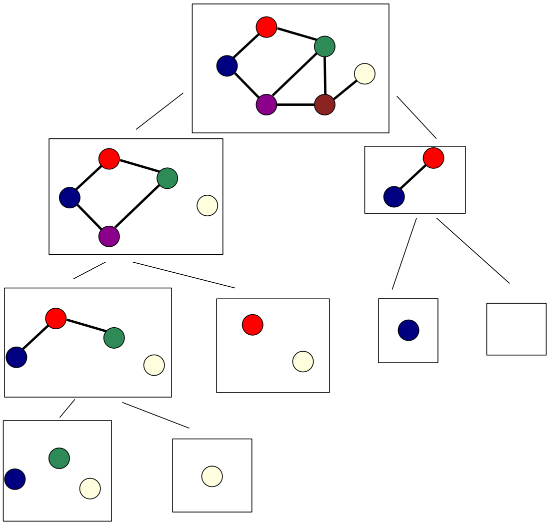
$G_1 = G - v_1$ obtained by removing v_1 and incident edges from G
$G_2 = G - v_1 - N(v_1)$ obtained by removing $N(v_1) \cup v_1$ from G

$$MIS(G) = \max\{MIS(G_1), MIS(G_2) + w(v_1)\}$$

A Recursive Algorithm

```
RecursiveMIS(G):
    if G is empty then Output 0
    a = RecursiveMIS(G − v_1)
    b = w(v_1) + RecursiveMIS(G − v_1 − N(v_n))
    Output max(a, b)
```

Example



Running time:

$$T(n) = T(n-1) + T\Big(n - 1 - deg(v_1)\Big) + O(1 + deg(v_1))$$

where $deg(v_1)$ is the degree of v_1. $T(0) = T(1) = 1$ is base case.

Worst case is when $deg(v_1) = 0$ when the recurrence becomes

$$T(n) = 2T(n-1) + O(1)$$

Solution to this is $T(n) = O(2^n)$.

Backtrack Search via Recursion

1. Recursive algorithm generates a tree of computation where each node is a smaller problem (subproblem)
2. Simple recursive algorithm computes/explores the whole tree blindly in some order.
3. Backtrack search is a way to explore the tree intelligently to prune the search space
    1. Some subproblems may be so simple that we can stop the recursive algorithm and solve it directly by some other method
    2. Memoization to avoid recomputing same problem
    3. Stop the recursion at a subproblem if it is clear that there is no need to explore further.
    4. Leads to a number of heuristics that are widely used in practice although the worst case running time may still be exponential.

THE END

...

(for now)