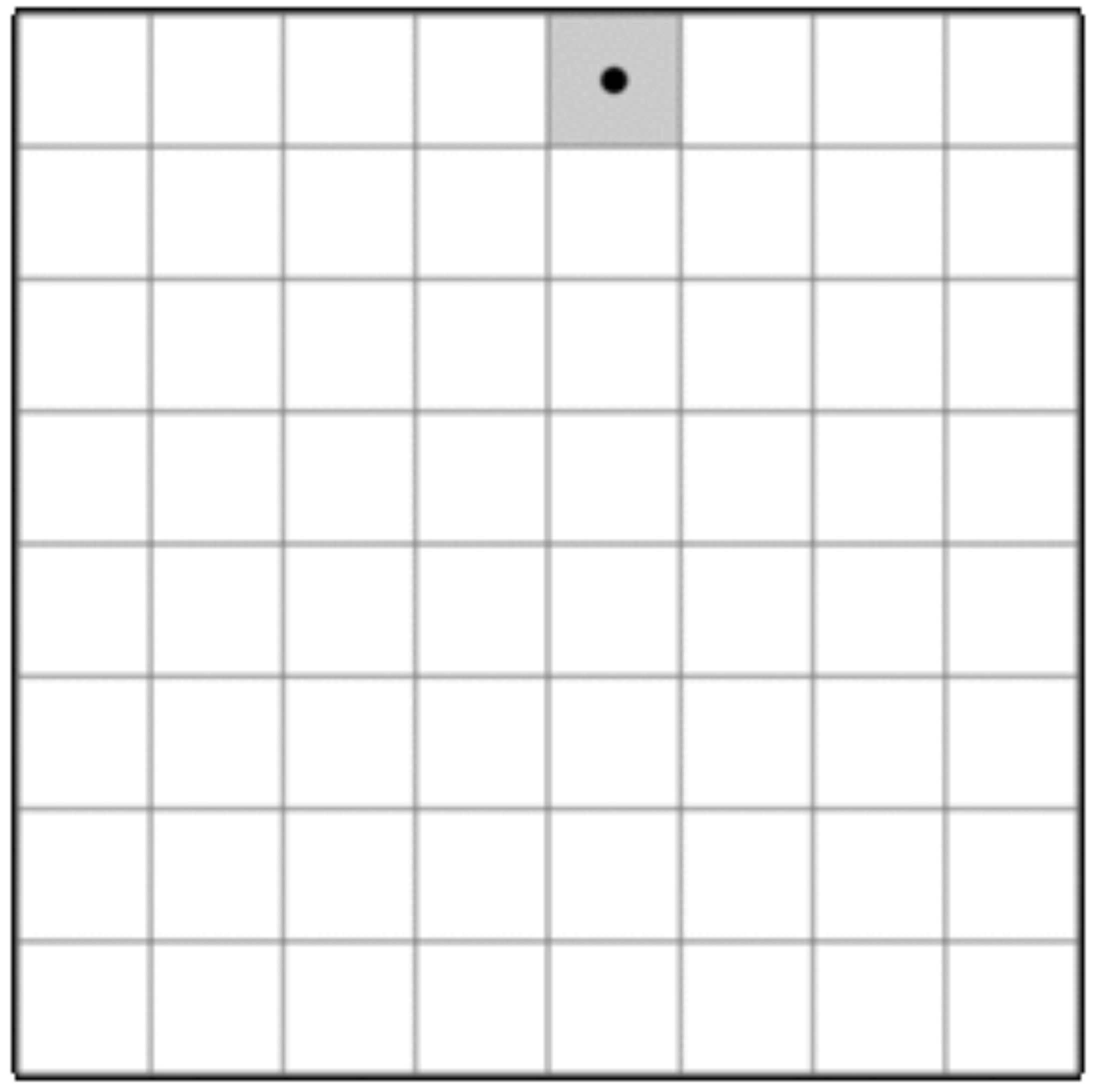
Backtracking:

Combining Recursion and Iteration

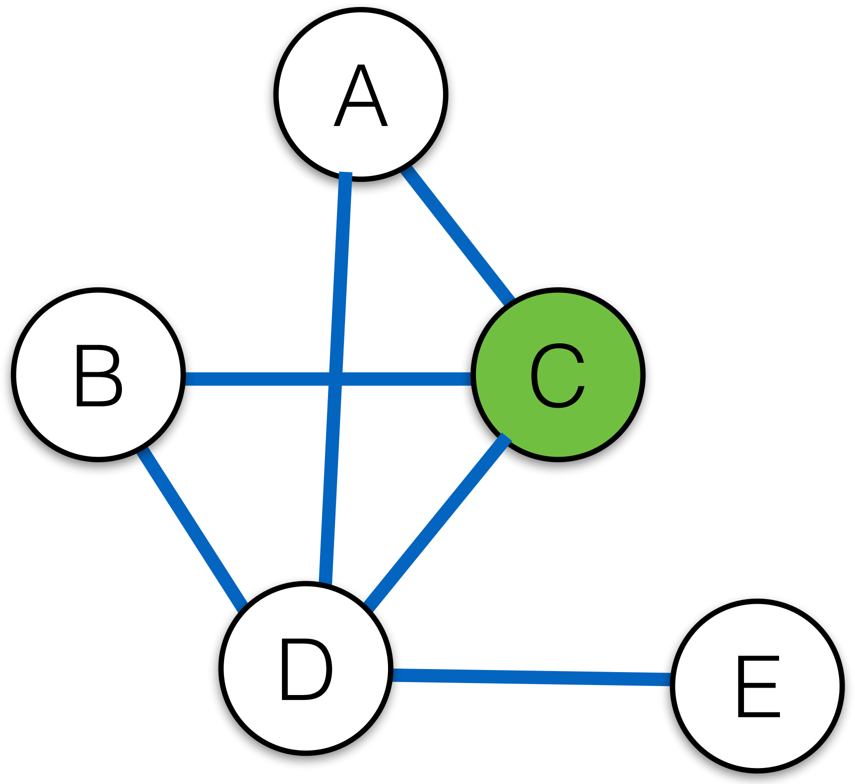
Hamiltonian paths

Definition: Given a graph $G = (V, E)$, a *Hamiltonian Path* from node v_0 is an enumeration $[v_0, v_1, \ldots, v_k]$ of V so every vertex $v \in V$ occurs *exactly once* in the list, such that for each $i < k$, $[v_i, v_{i+1}] \in E$.

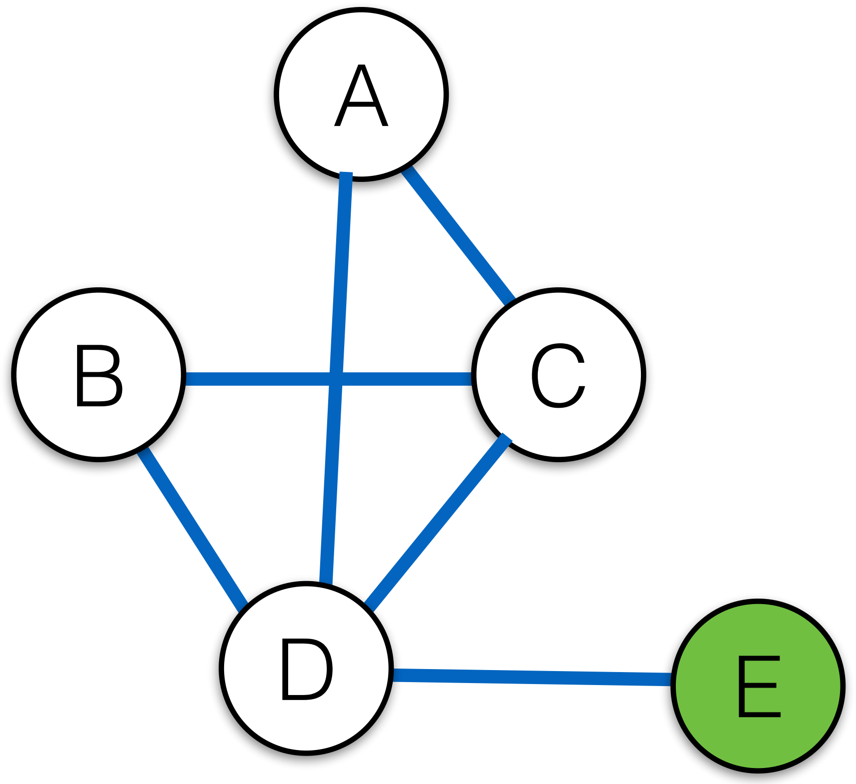
- Applications:
    - Visiting every pub only once during the night;
    - Covering a chess board with knight's moves.
- Surprisingly, finding Hamiltonian paths is a *very hard problem* in terms of computational complexity.



Is there a Hamiltonian path…

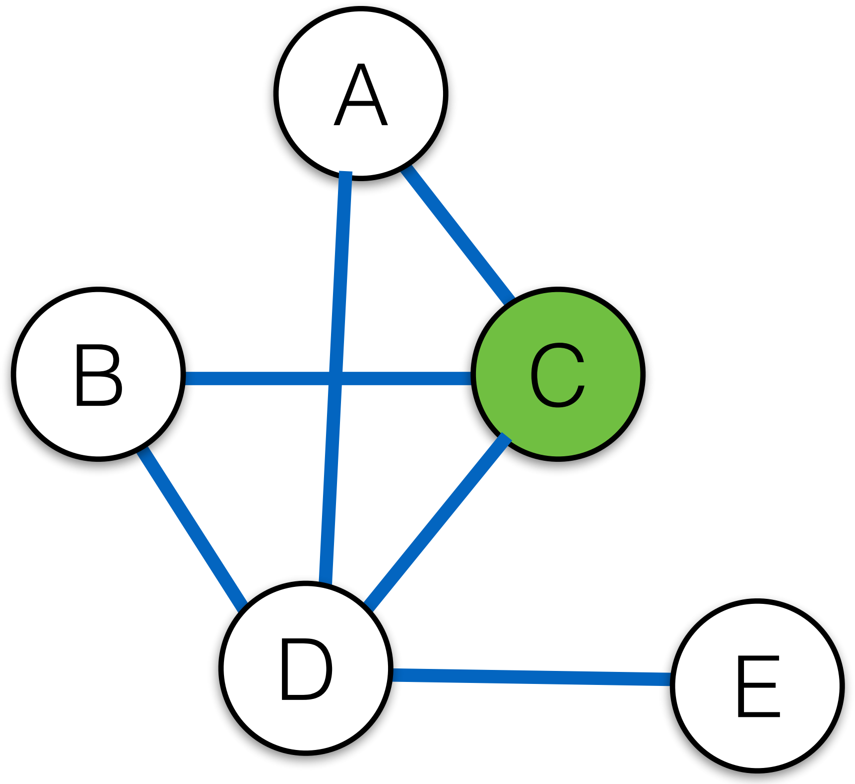


… from the node C?



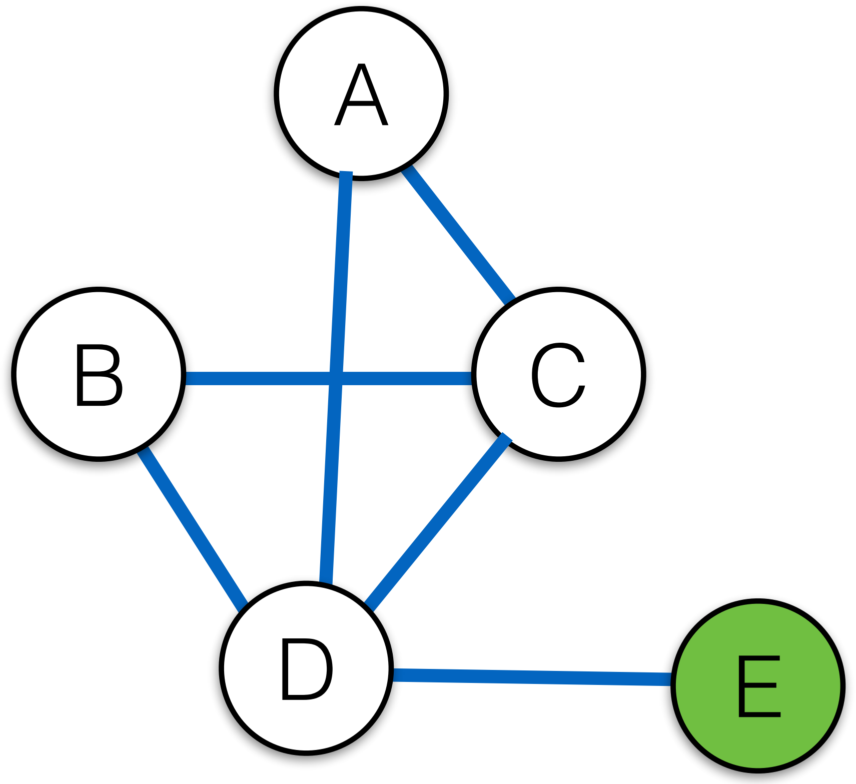
… from the node E?

Is there a Hamiltonian path…



… from the node C?

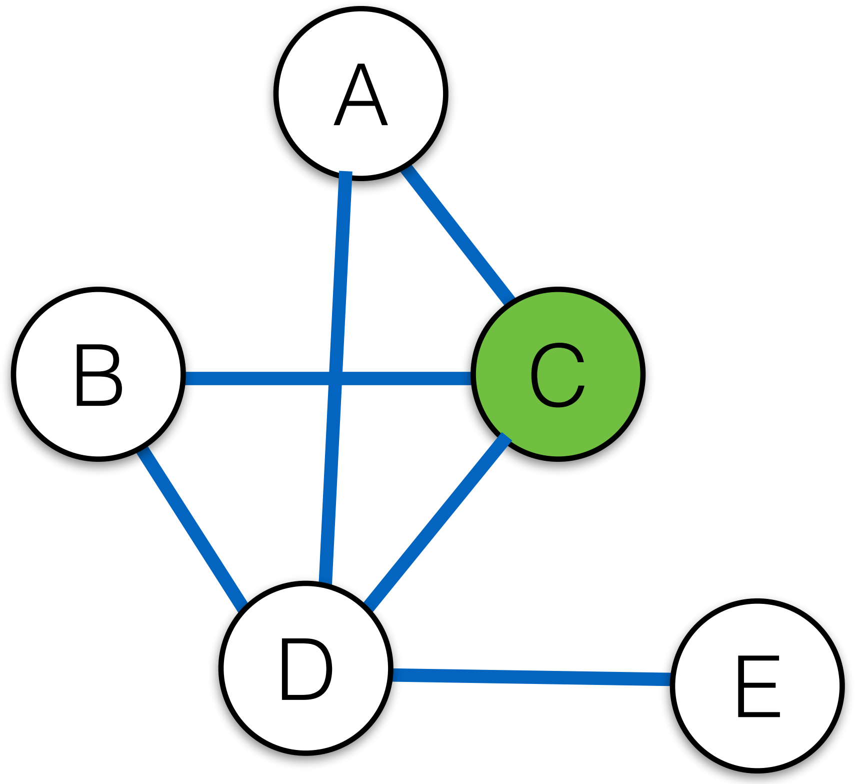
No



… from the node E?

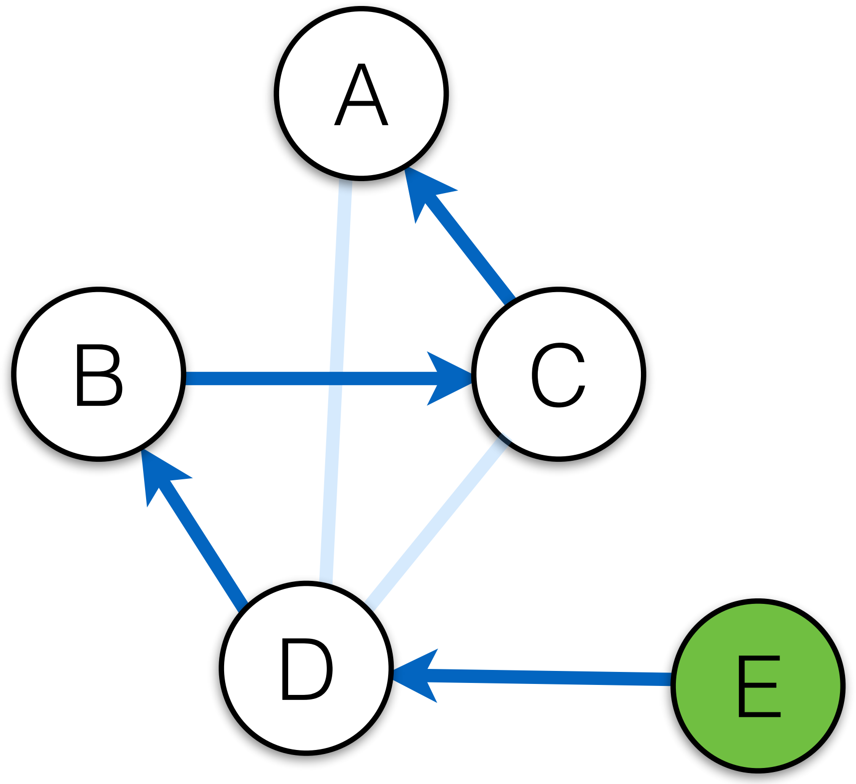
Yes

Is there a Hamiltonian path…



… from the node C?

No



… from the node E?

Yes

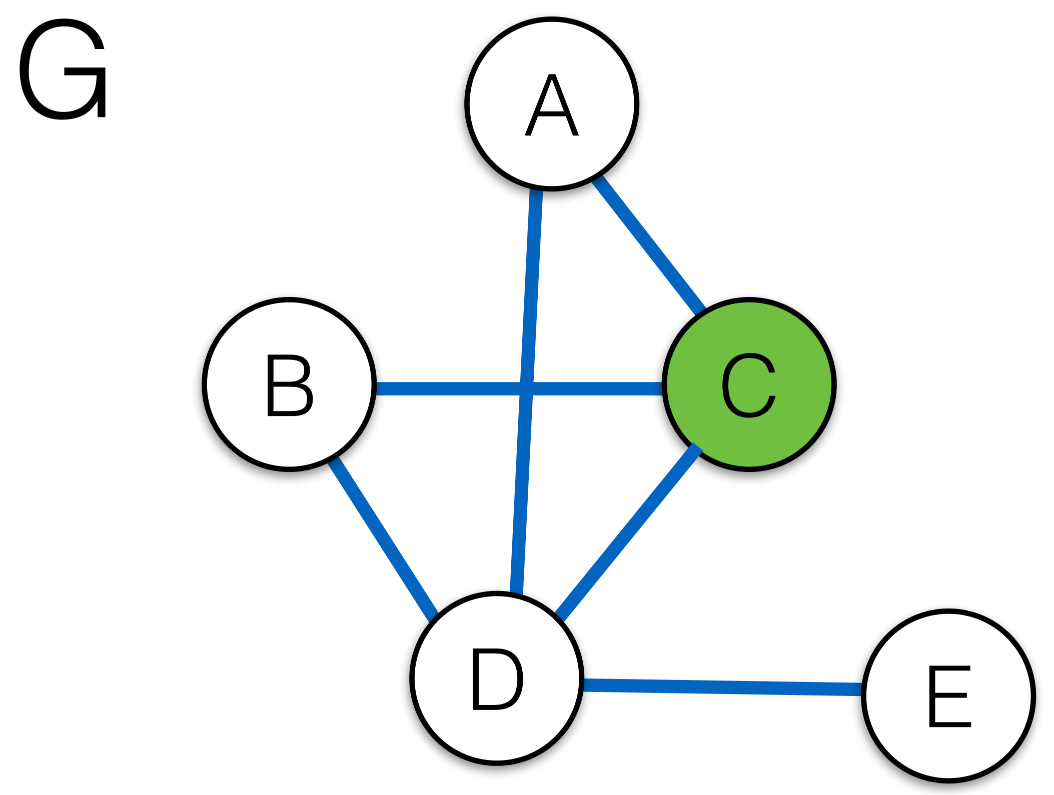
Hamiltonian path algorithm

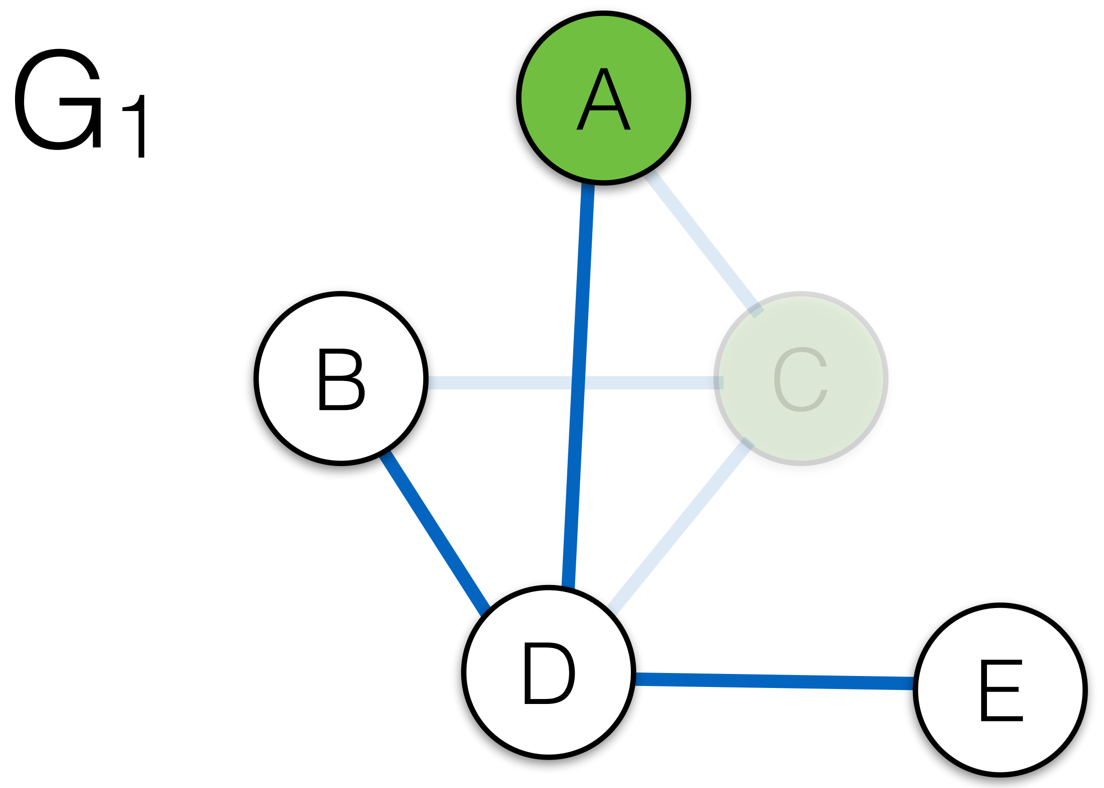
```
HPCheck(G, v) {
  if (|V| == 1) then {   // If it’s just one node, the problem is trivial
    return true;
  } else {
    $V_1$  := G.V \ {v};   // Remove the node v from the graph…
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);   // … as well as all edges that contain it.
    $G_1$  := ($V_1$, $E_1$);
    ans := false;
    foreach (w ∈ $V_1$) {   // Check recursively if we can build HP
      ans := ans ||          through some of v’s neighbours.
               [v, w] ∈ G.E && HPCheck($G_1$, w);
    }
    return ans;
  }
}
```

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    $V_1$  := G.V \ {v};
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);
    $G_1$  := ($V_1$, $E_1$);
    ans := false;
    foreach (w ∈ $V_1$) {
      ans := ans || [v, w] ∈ G.E && HPCheck($G_1$, w);
    }
    return ans;
  }
}
```

v = C





$G_1.V = \{A, B, D, E\}$

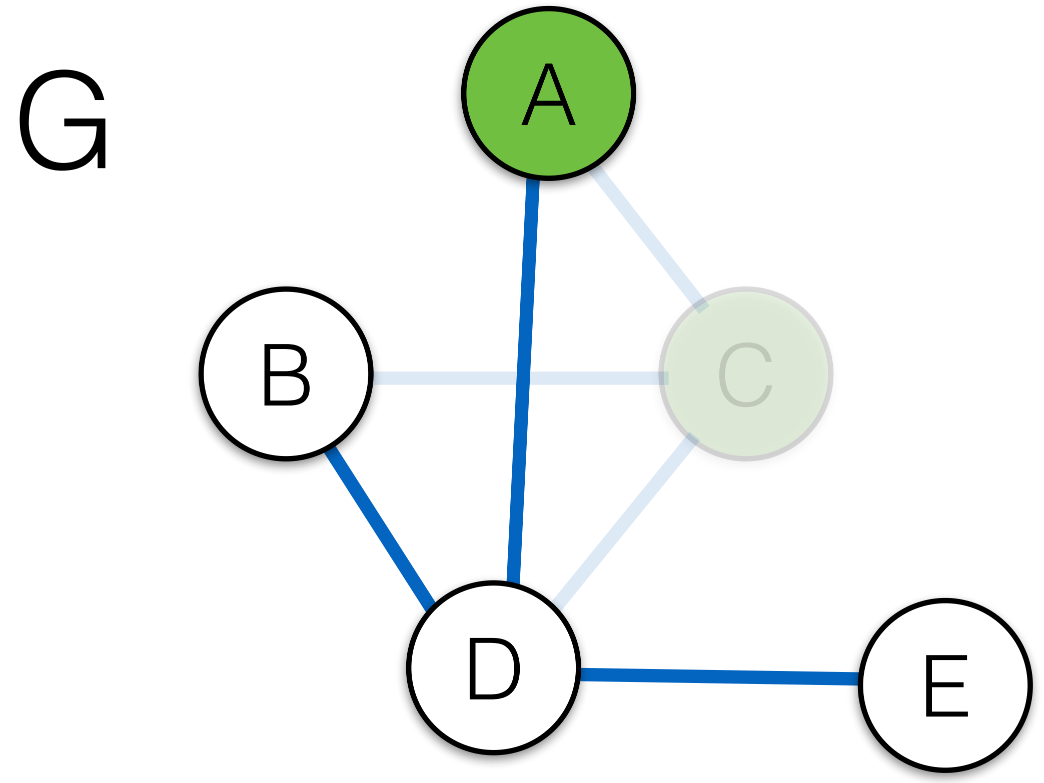
$G_1.E = \{[A, D], [B, D], [D, E]\}$

w = A

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    $V_1$  := G.V \ {v};
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);
    $G_1$  := ($V_1$, $E_1$);
    ans := false;
    foreach (w ∈ $V_1$) {
      ans := ans || [v, w] ∈ G.E && HPCheck($G_1$, w);
    }
    return ans;
  }
}
```

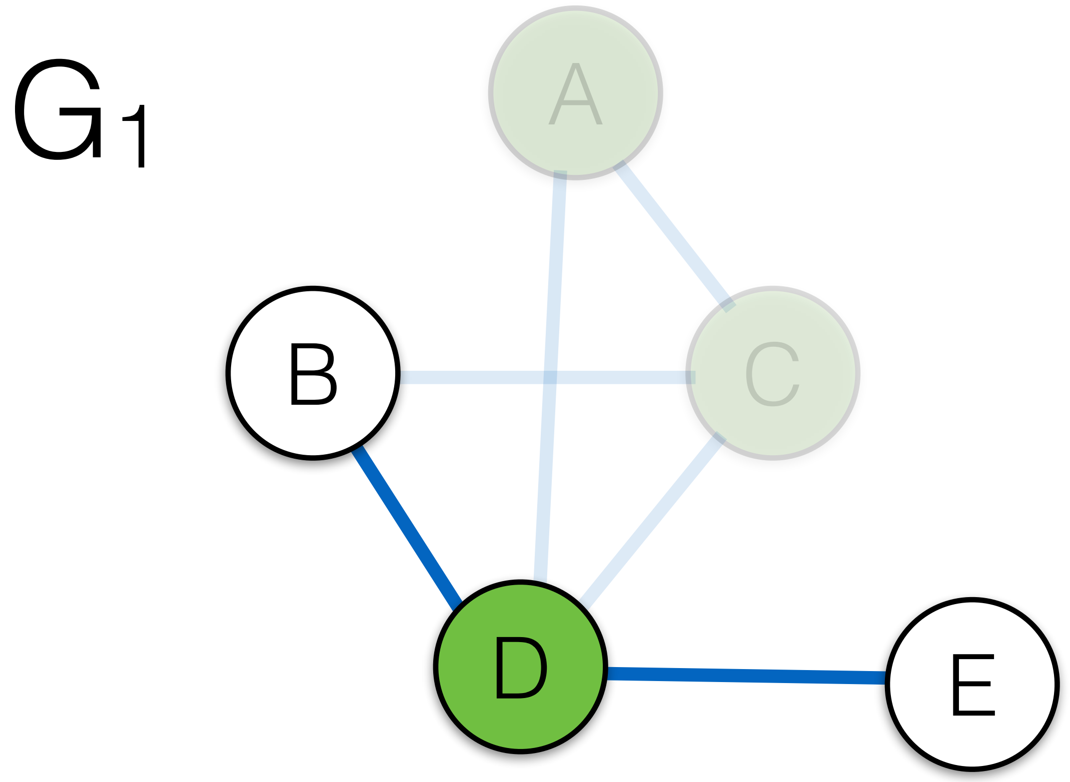
v = A



$G_1.V = \{B, D, E\}$

$G_1.E = \{[B, D], [D, E]\}$

w = D

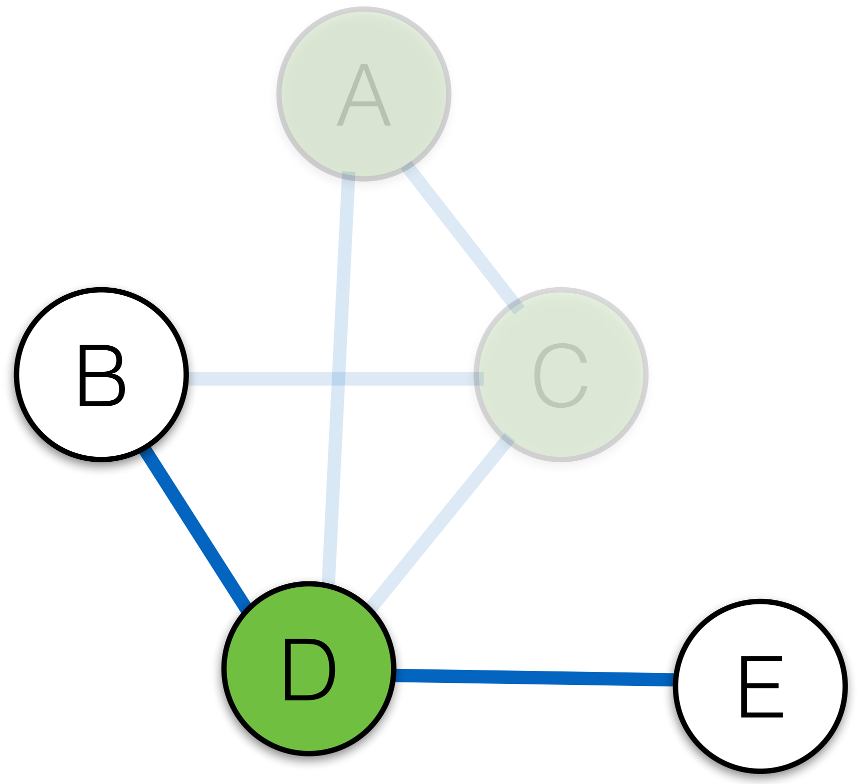


Example

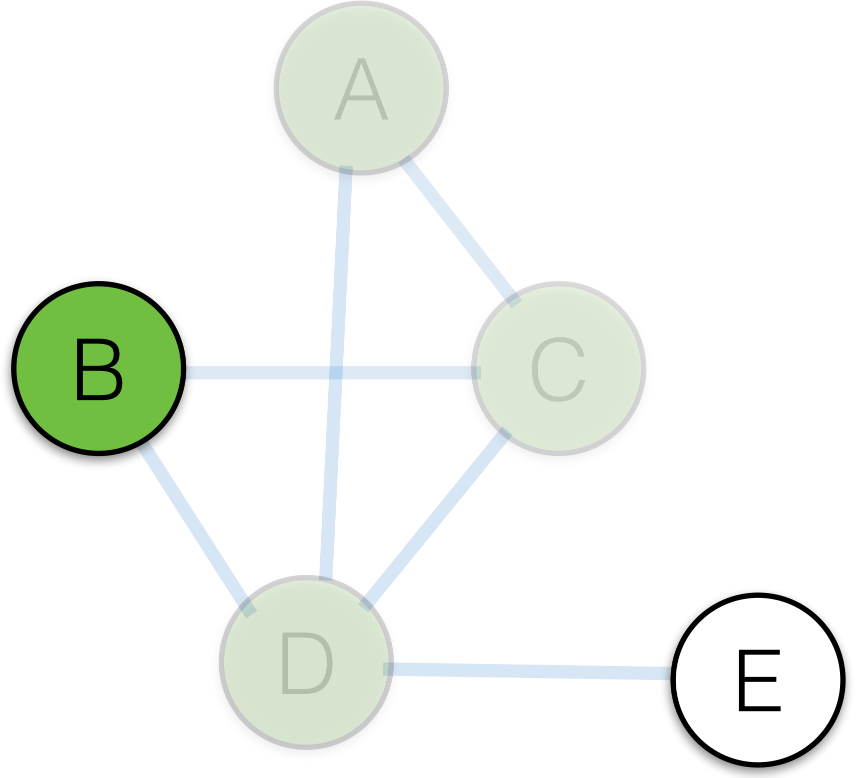
```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    V₁  := G.V \ {v};
    E₁  := G.E ∩ (V₁ × V₁);
    G₁  := (V₁, E₁);
    ans := false;
    foreach (w ∈ V₁) {
      ans := ans || [v, w] ∈ G.E && HPCheck(G₁, w);
    }
    return ans;
  }
}
```

v = D

G



G_1



$G_1.V = \{B, E\}$

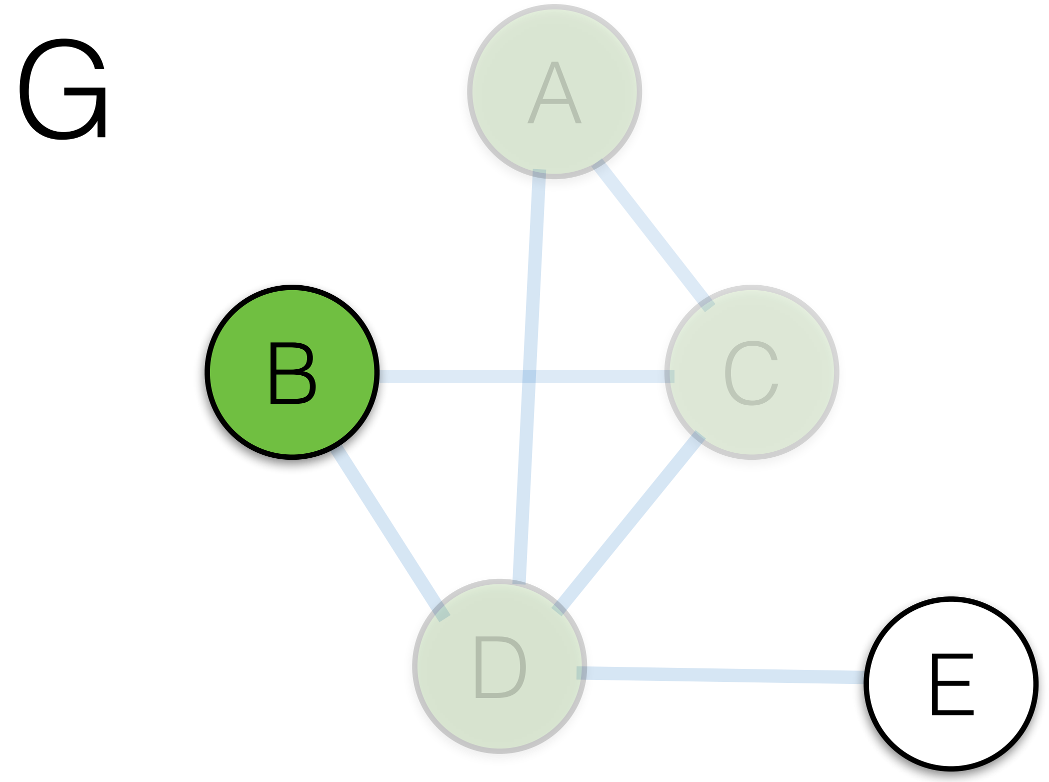
$G_1.E = \{\}$

w = B

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    V₁  := G.V \ {v};
    E₁  := G.E ∩ (V₁ × V₁);
    G₁  := (V₁, E₁);
    ans := false;
    foreach (w ∈ V₁) {
      ans := ans || [v, w] ∈ G.E && HPCheck(G₁, w);
    }
    return ans;
  }
}
```

v = B



$G_1.V = \{B, E\}$

$G_1.E = \{\}$

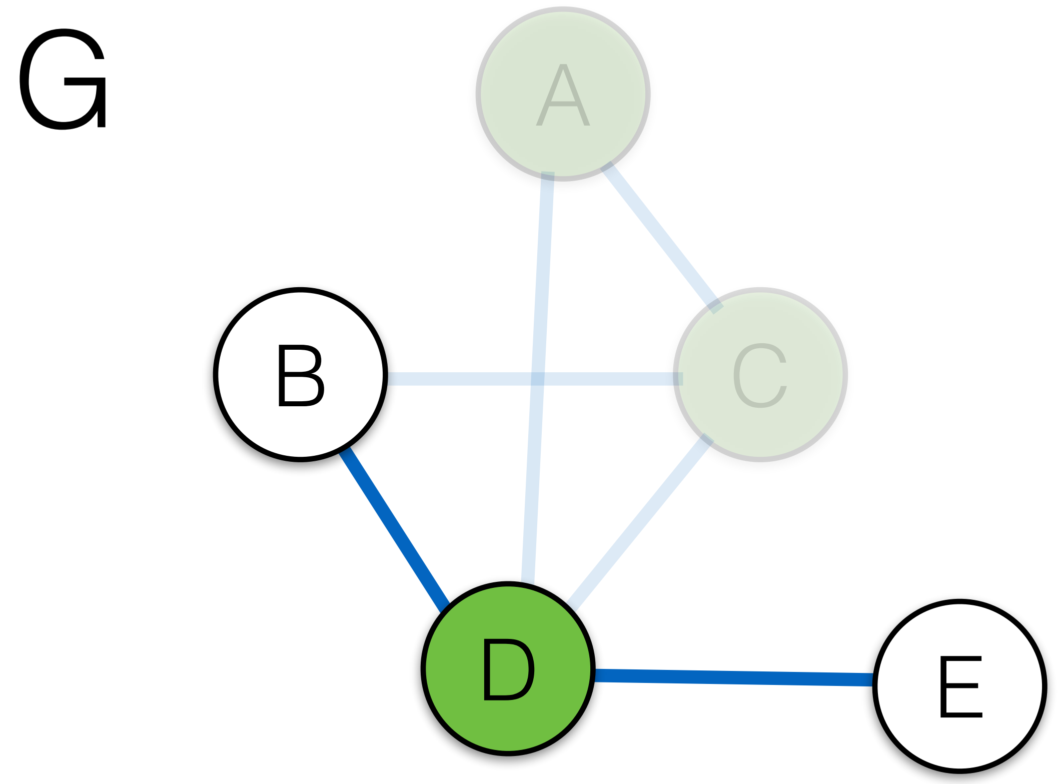
w = E ⇒ `ans = false`, since [B, E] ∉ G.E

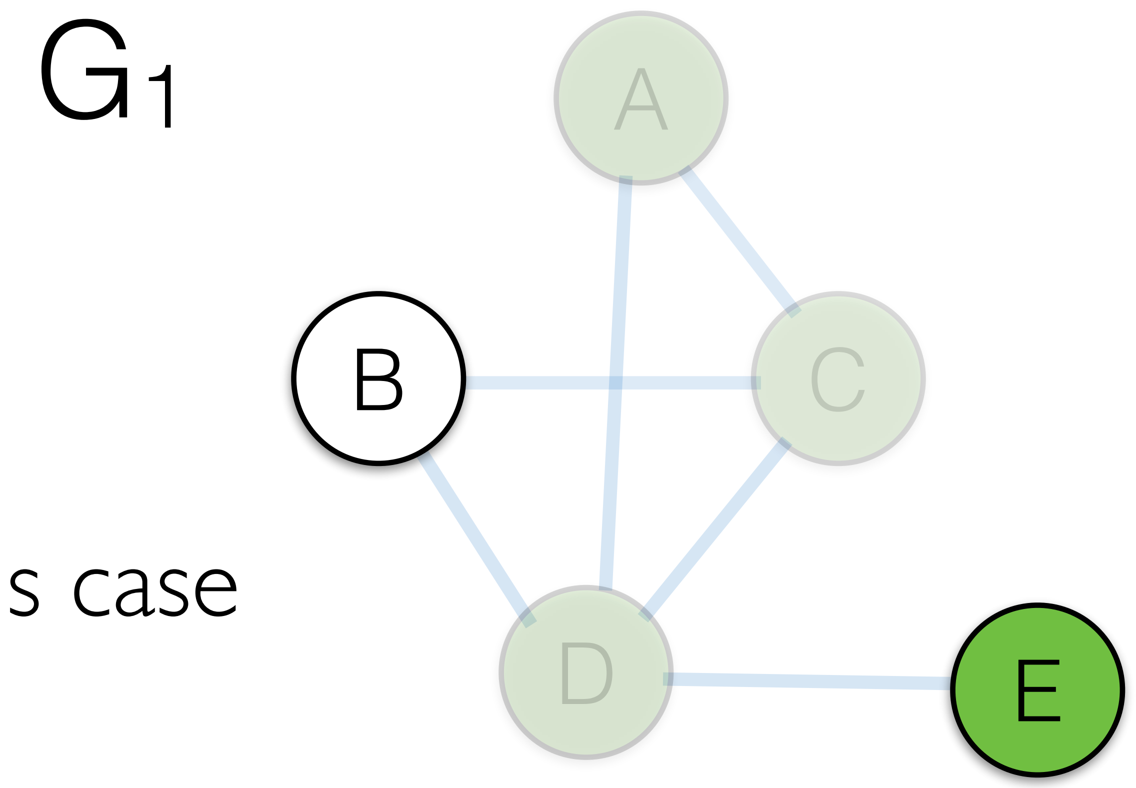
Backtrack to the previous level of recursion.

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    $V_1$  := G.V \ {v};
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);
    $G_1$  := ($V_1$, $E_1$);
    ans := false;
    foreach (w ∈ $V_1$) {
      ans := ans || [v, w] ∈ G.E && HPCheck($G_1$, w);
    }
    return ans;
  }
}
```

v = D





$G_1.V = \{B, E\}$

$G_1.E = \{\}$

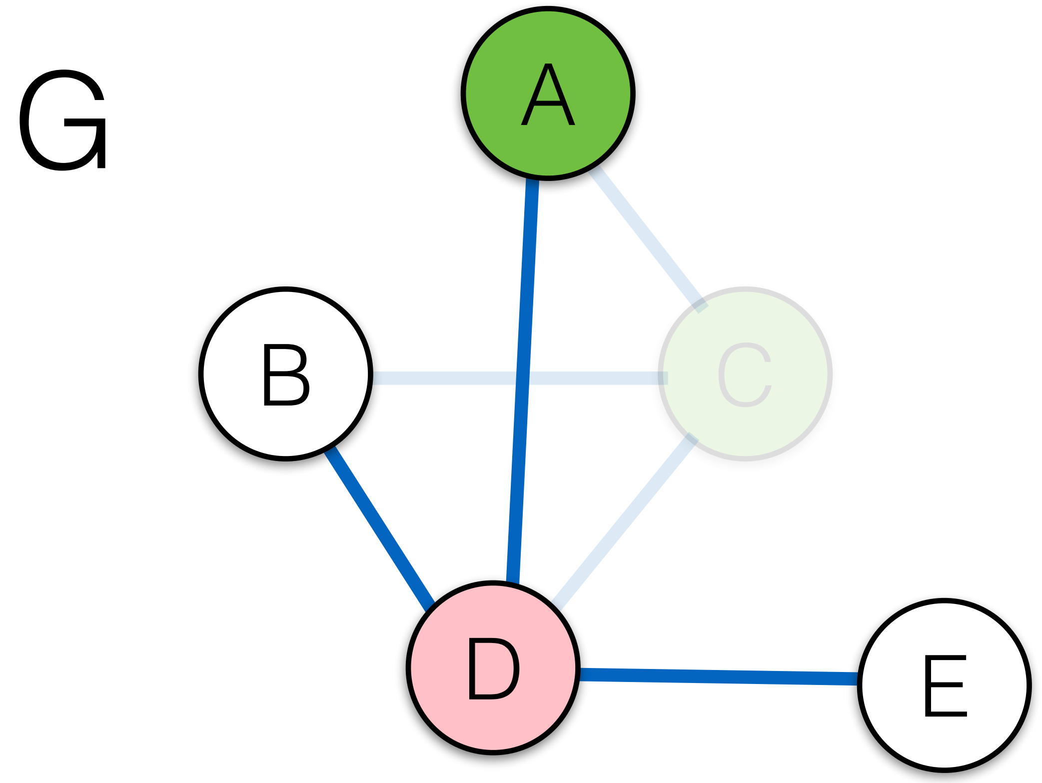
w = E ⇒ `ans = false`, similarly to the previous case

Backtrack even further…

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    V₁  := G.V \ {v};
    E₁  := G.E ∩ (V₁ × V₁);
    G₁  := (V₁, E₁);
    ans := false;
    foreach (w ∈ V₁) {
      ans := ans || [v, w] ∈ G.E && HPCheck(G₁, w);
    }
    return ans;
  }
}
```

v = A



$G_1.V = \{B, D, E\}$

$G_1.E = \{[B, D], [D, E]\}$

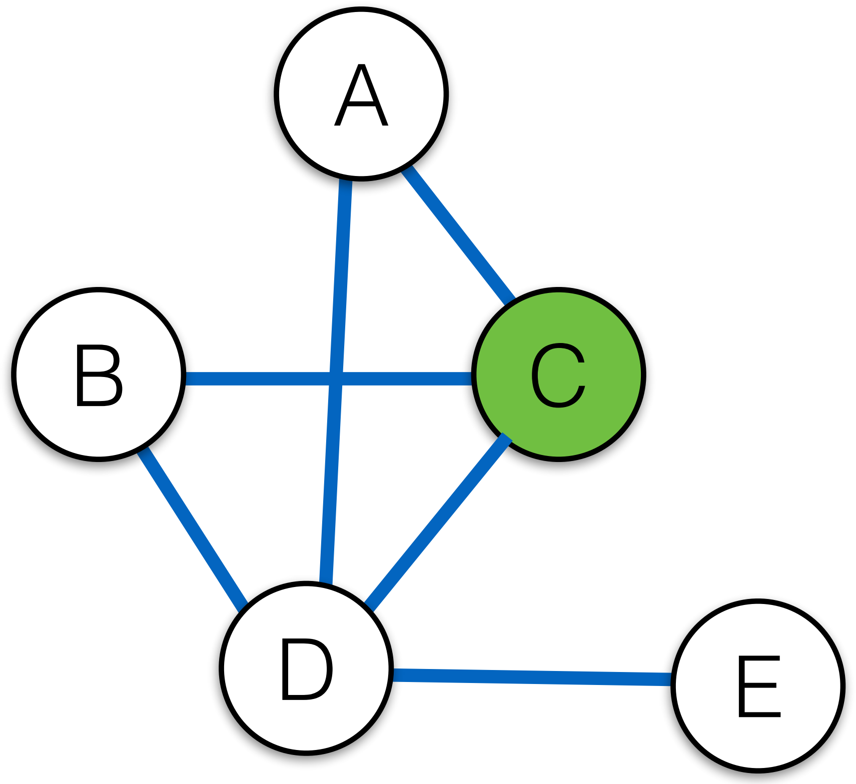
No more neighbours of A to explore ⇒ backtrack to the previous level…

Example

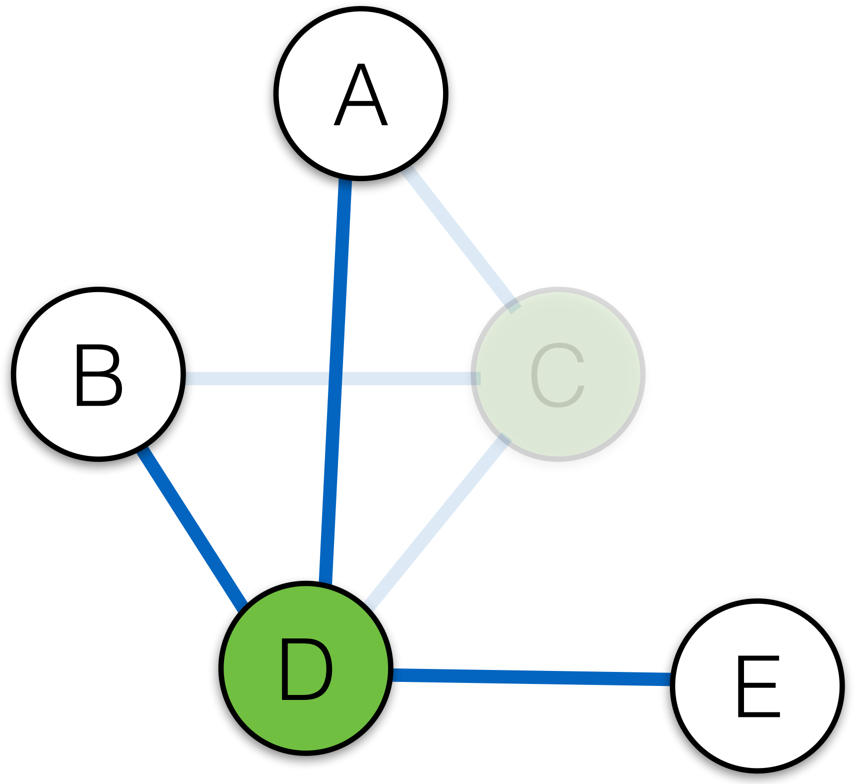
```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    $V_1$  := G.V \ {v};
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);
    $G_1$  := ($V_1$, $E_1$);
    ans := false;
    foreach (w ∈ $V_1$) {
      ans := ans || [v, w] ∈ G.E && HPCheck($G_1$, w);
    }
    return ans;
  }
}
```

v = C

G



G_1



G_1.V = {A, B, D, E}

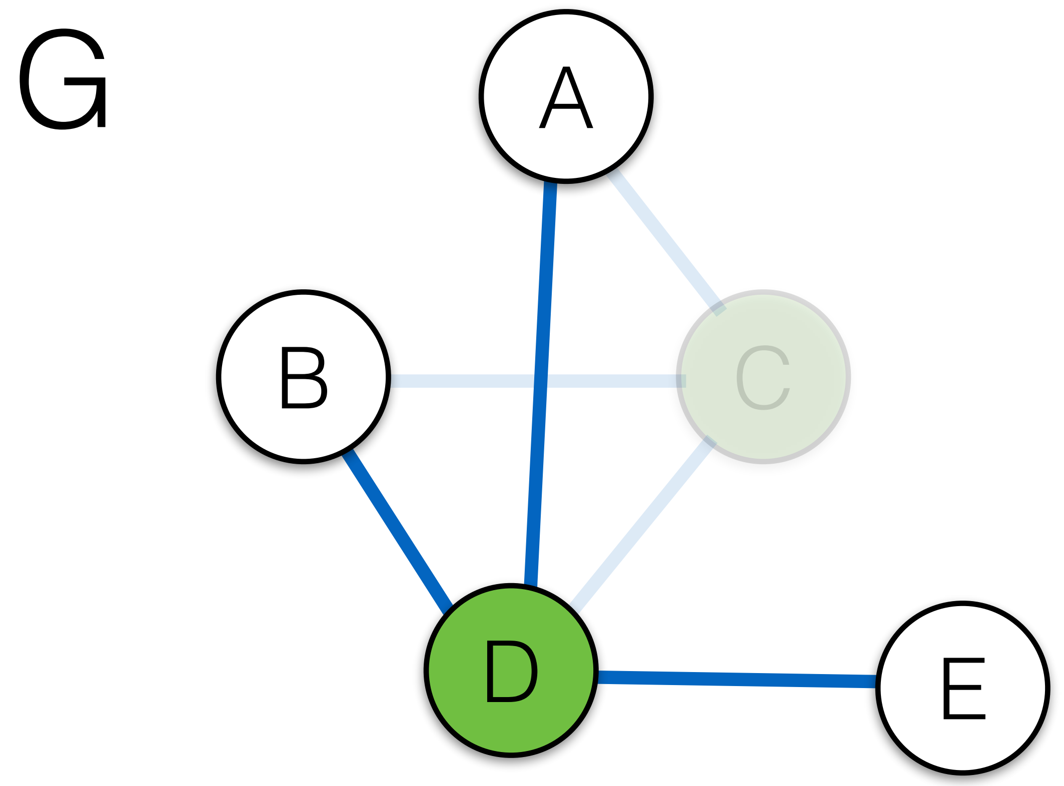
G_1.E = {[A, D], [B, D], [D, E]}

w = D

Example

```
HPCheck(G, v) {
  if (|V| == 1) then {
    return true;
  } else {
    V₁  := G.V \ {v};
    E₁  := G.E ∩ (V₁ × V₁);
    G₁  := (V₁, E₁);
    ans := false;
    foreach (w ∈ V₁) {
      ans := ans || [v, w] ∈ G.E && HPCheck(G₁, w);
    }
    return ans;
  }
}
```

v = D



All recursive calls from D are going to fail, hence the overall result is **`false`**.

The algorithms iterates through ***all*** possible subsets of V.

Hence, the complexity is likely to be very bad…

Hamiltonian paths complexity

In terms of *set operations* (element removals), assuming |V| = n.

```
HPCheck(G, v) {                              // h(n)
  if (|V| == 1) then {
    return true;                             // 1
  } else {
    $V_1$  := G.V \ {v};                     // 1
    $E_1$  := G.E ∩ ($V_1$ × $V_1$);         // 2·(n - 1)  — see next slide
    $G_1$  := ($V_1$, $E_1$);                // 0
    ans := false;                            // 0
    foreach (w ∈ $V_1$) {                    // (n - 1) times
      ans := ans ||
             [v, w] ∈ G.E && HPCheck($G_1$, w);    // h(n - 1)
    }
    return ans;
  }
}
```

“Filtering” the set of edges

$\boxed{G.E \cap (V_1 \times V_1)}$

$|V| = n$

$V_1 = V \setminus \{v\} \quad \Rightarrow |V_1| = n - 1$

```
$E_1$ := G.E;
foreach (w $\in$ $V_1$) {
  if ([v, w] $\in$ G.E) then $E_1$ := $E_1$ \ [v, w]; // n - 1 element removals
  if ([w, v] $\in$ G.E) then $E_1$ := $E_1$ \ [v, w]; // n - 1 element removals
}
```

Overall complexity: $2 \cdot (n - 1)$ removals.

Recurrence relation for Hamilton paths

$$\boxed{\begin{array}{l} h(n) = (n-1)\cdot h(n-1) + 2n - 1 \text{ if } n > 1 \\ h(1) = 1 \end{array}}$$

Change of function: $h(n) = (n-1)! \cdot g(n)$ since $b_i = (i-1)$ for all i

Substituting for $h(n)$: $(n-1)!\, g(n) = (n-1)(n-2)!\, g(n-1) + 2n - 1$

$$= (n-1)!\, g(n-1) + 2(n-1) + 1$$

$$g(n) = g(n-1) + \frac{2}{(n-2)!} + \frac{1}{(n-1)!}$$

Recurrence relation for Hamilton paths

h(n) = (n-1)·h(n-1) + 2n - 1 if n > 1
h(1) = 1

$$g(n) = g(n-1) + \frac{2}{(n-2)!} + \frac{1}{(n-1)!}$$

By method of differences: $g(n) = 2\sum_{i=1}^{n-2}\frac{1}{i!} + \sum_{i=1}^{n-1}\frac{1}{i!} \leq 3\sum_{i=1}^{n}\frac{1}{i!}$

From calculus: $\leq 3(e-1)$

$$\lim_{n\to\infty}\sum_{i=1}^{n}\frac{1}{i!} = e - 1$$

$$h(n) \leq 3(e-1)(n-1)!$$ ∈ O((n-1)!)

To take away

- The complexity of Hamiltonian path *checking* is $O((|V| - 1)!)$.
- It is a typical example of an algorithm that requires *backtracking:* an algorithmic implementation technique that combines recursion and iteration.
    - Iteration is used to enumerate current choices
    - Recursion "commits" to a particular choice and attempts to solve a "reduced" problem.
- Complexity of algorithms with backtracking is usually quite bad (but it's unavoidable.)