ITMO UNIVERSITY

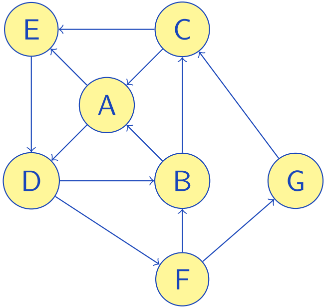
How to Win Coding Competitions: Secrets of Champions

Week 4: Algorithms on Graphs
Lecture 7: Hamiltonian paths and Hamiltonian tours

Maxim Buzdalov
Saint Petersburg 2016

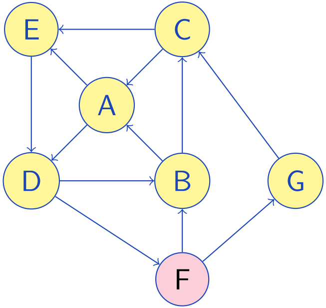
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



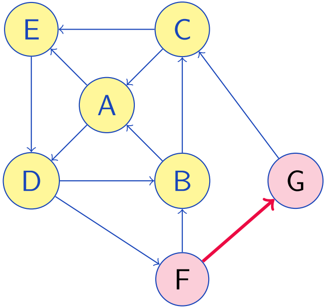
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



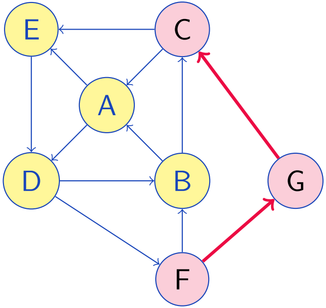
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



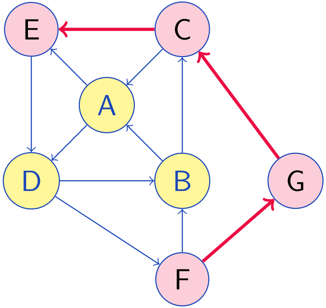
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



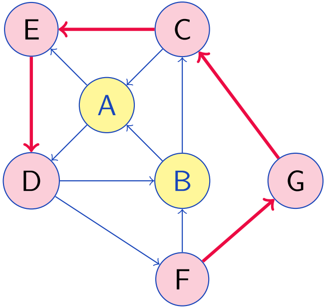
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



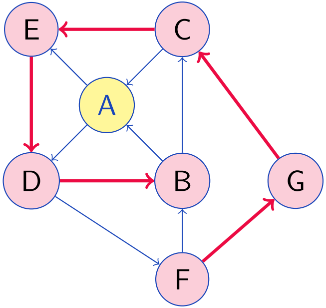
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



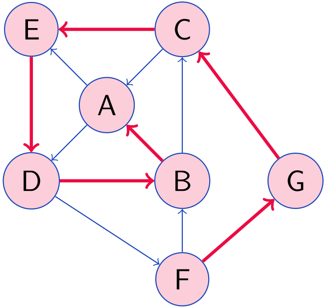
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



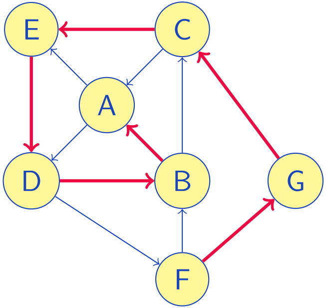
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



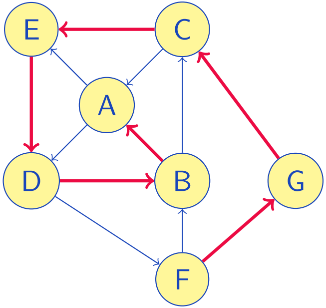
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



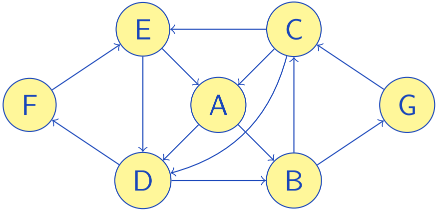
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



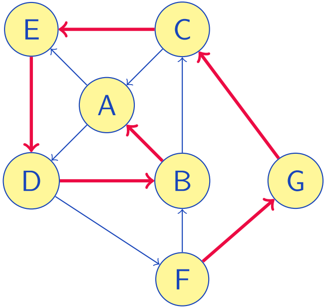
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



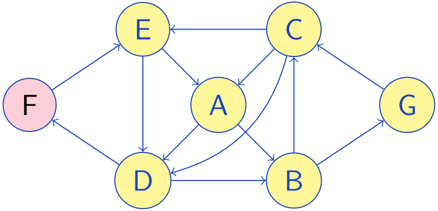
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



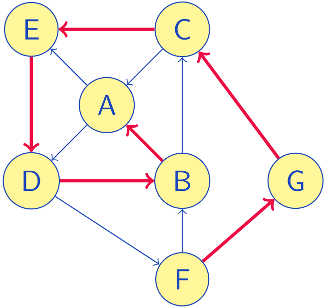
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



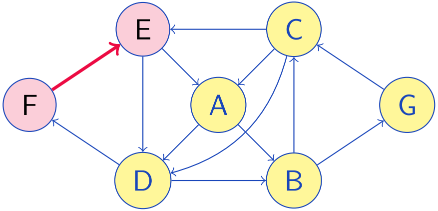
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



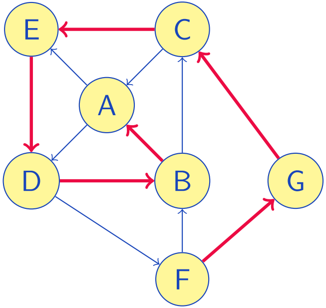
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



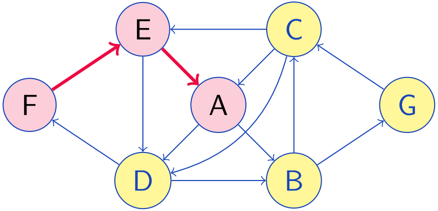
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



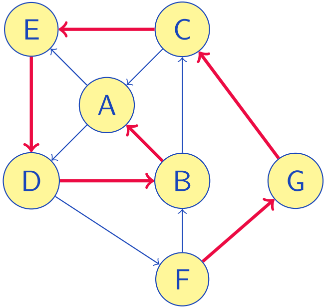
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



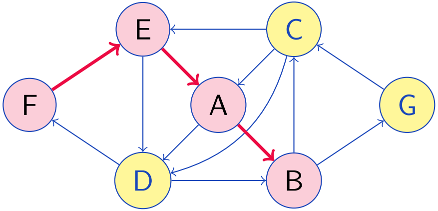
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



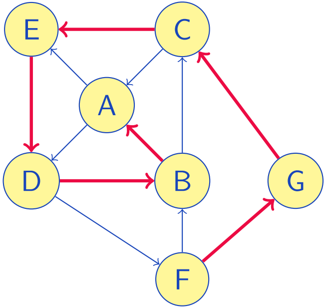
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



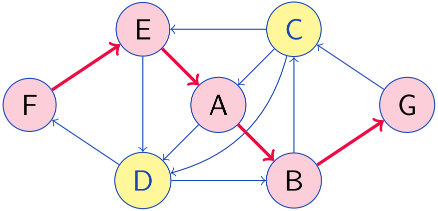
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



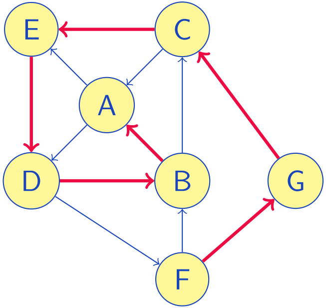
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



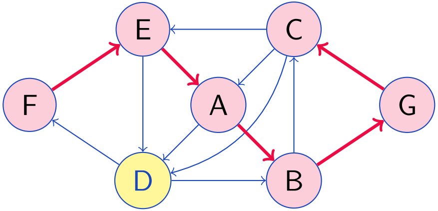
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



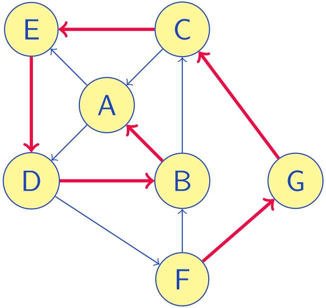
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



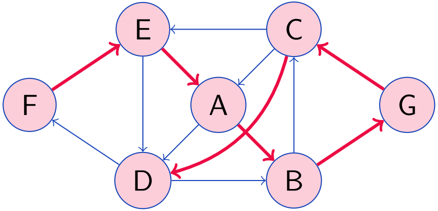
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



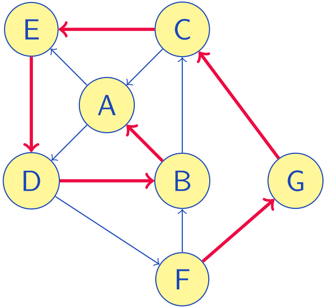
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



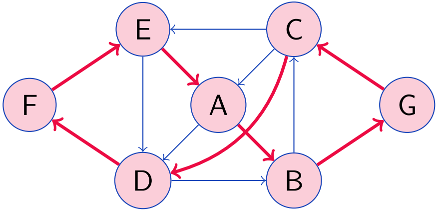
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



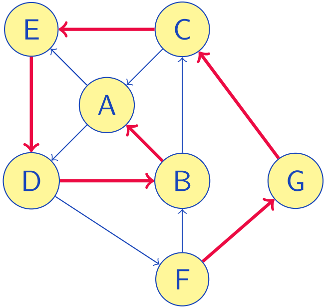
A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



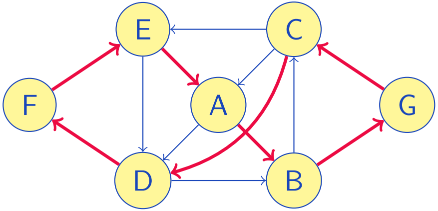
A Hamiltonian path is a path in a graph that contains each **vertex** of the graph exactly once

FGCEDBA



A Hamiltonian tour is a Hamiltonian path which starts and ends on the same vertex

FEABGCDF



Checking whether a Hamiltonian path/tour exists is NP-complete

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

- $O(N \cdot N!)$ where $N = |V|$

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

- $O(N \cdot N!)$ where $N = |V|$

More efficient solution: dynamic programming on vertex sets

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

- $O(N \cdot N!)$ where $N = |V|$

More efficient solution: dynamic programming on vertex sets

- $d[S][v]$: whether a path exists which:
  - Starts at vertex 1
  - Ends at vertex v
  - Visits exactly vertices from set S

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

- $O(N \cdot N!)$ where $N = |V|$

More efficient solution: dynamic programming on vertex sets

- $d[S][v]$: whether a path exists which:
  - Starts at vertex 1
  - Ends at vertex v
  - Visits exactly vertices from set S
- Vertex sets are stored as bitmasks
  - Numbers from 0 to $2^N - 1$
  - i-th bit is set if vertex number i is in the set

Checking whether a Hamiltonian path/tour exists is NP-complete

- No universal solution in polynomial time

Naïve solution: check all possible paths

- $O(N \cdot N!)$ where $N = |V|$

More efficient solution: dynamic programming on vertex sets

- $d[S][v]$: whether a path exists which:
  - Starts at vertex 1
  - Ends at vertex v
  - Visits exactly vertices from set S
- Vertex sets are stored as bitmasks
  - Numbers from 0 to $2^N - 1$
  - i-th bit is set if vertex number i is in the set
- We can solve Hamiltonian-related problems using the values of $d[S][v]$

procedure $\textsc{HamiltonianDP}(V, E)$
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow \textsc{true}$
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do**
        for $v \in S \setminus \{1\}$ **do**
            $d[S][v] \leftarrow \textsc{false}$
            $S' \leftarrow S \setminus \{v\}$
            for $u \in S'$ **do**
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if**
            end for
        end for
    end for
end procedure

procedure $\textsc{HamiltonianDP}(V, E)$
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow \textsc{true}$ ▷ A path consisting of vertex 1 exists
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do**
        for $v \in S \setminus \{1\}$ **do**
            $d[S][v] \leftarrow \textsc{false}$
            $S' \leftarrow S \setminus \{v\}$
            for $u \in S'$ **do**
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if**
            end for
        end for
    end for
end procedure

procedure $\textsc{HamiltonianDP}(V, E)$
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow \textsc{true}$ ▷ A path consisting of vertex 1 exists
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do** ▷ Check all sets
        for $v \in S \setminus \{1\}$ **do**
            $d[S][v] \leftarrow \textsc{false}$
            $S' \leftarrow S \setminus \{v\}$
            for $u \in S'$ **do**
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if**
            end for
        end for
    end for
end procedure

procedure HAMILTONIANDP(V, E)
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow$ TRUE ▷ A path consisting of vertex 1 exists
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do** ▷ Check all sets
        for $v \in S \setminus \{1\}$ **do** ▷ Check all possible endpoints
            $d[S][v] \leftarrow$ FALSE
            $S' \leftarrow S \setminus \{v\}$
            for $u \in S'$ **do**
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if**
            end for
        end for
    end for
end procedure

```
procedure HamiltonianDP(V, E)
    d[S][v]: if a path exists which starts at 1, ends at v and visits vertices from S
    d[{1}][1] ← TRUE                                  ▷ A path consisting of vertex 1 exists
    for S ∈ 2^V in non-decreasing order of |S| where |S| ≥ 2 do        ▷ Check all sets
        for v ∈ S \ {1} do                            ▷ Check all possible endpoints
            d[S][v] ← FALSE                           ▷ Initially no path
            S' ← S \ {v}
            for u ∈ S' do
                if (u, v) ∈ E then d[S][v] ← d[S][v] or d[S'][u] end if
            end for
        end for
    end for
end procedure
```

procedure $\text{HAMILTONIANDP}(V, E)$
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow \text{TRUE}$ ▷ A path consisting of vertex 1 exists
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do** ▷ Check all sets
        for $v \in S \setminus \{1\}$ **do** ▷ Check all possible endpoints
            $d[S][v] \leftarrow \text{FALSE}$ ▷ Initially no path
            $S' \leftarrow S \setminus \{v\}$ ▷ The previous vertex set: ready as $|S'| < |S|$
            for $u \in S'$ **do**
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if**
            end for
        end for
    end for
end procedure

```
procedure HamiltonianDP(V, E)
    d[S][v]: if a path exists which starts at 1, ends at v and visits vertices from S
    d[{1}][1] ← TRUE                                   ▷ A path consisting of vertex 1 exists
    for S ∈ 2^V in non-decreasing order of |S| where |S| ≥ 2 do      ▷ Check all sets
        for v ∈ S \ {1} do                              ▷ Check all possible endpoints
            d[S][v] ← FALSE                             ▷ Initially no path
            S' ← S \ {v}                ▷ The previous vertex set: ready as |S'| < |S|
            for u ∈ S' do                               ▷ Check all possible previous vertices
                if (u, v) ∈ E then d[S][v] ← d[S][v] or d[S'][u] end if
            end for
        end for
    end for
end procedure
```

procedure HamiltonianDP(V, E)
    $d[S][v]$: if a path exists which starts at 1, ends at v and visits vertices from S
    $d[\{1\}][1] \leftarrow$ TRUE ▷ A path consisting of vertex 1 exists
    for $S \in 2^V$ in non-decreasing order of $|S|$ where $|S| \geq 2$ **do** ▷ Check all sets
        for $v \in S \setminus \{1\}$ **do** ▷ Check all possible endpoints
            $d[S][v] \leftarrow$ FALSE ▷ Initially no path
            $S' \leftarrow S \setminus \{v\}$ ▷ The previous vertex set: ready as $|S'| < |S|$
            for $u \in S'$ **do** ▷ Check all possible previous vertices
                if $(u, v) \in E$ **then** $d[S][v] \leftarrow d[S][v]$ **or** $d[S'][u]$ **end if** ▷ Update
            end for
        end for
    end for
end procedure

```java
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];           // save one bit, reduce memory 2x times
    d[0][0] = 1;                                               // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {              // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {                // mask contains v
                int prev = mask ^ (1 << (v - 1));              // previous mask
                boolean curr = d[prev][0] && graph[v][0];      // consider 0 separately
                for (int u = 1; u < n; ++u) {                  // check previous vertices
                    if (graph[v][u]) {                         // if graph has the (v,u) edge ...
                        if ((prev & (1 << (u - 1))) != 0) {    // ... and if u is in the mask ...
                            curr |= d[prev][u];                // update the current value
                        }
                    }
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

```java
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];          // save one bit, reduce memory 2x times
    d[0][0] = 1;                                               // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {             // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {               // mask contains v
                int prev = mask ^ (1 << (v - 1));              // previous mask
                boolean curr = d[prev][0] && graph[v][0];      // consider 0 separately
                for (int u = 1; u < n; ++u) {                  // check previous vertices
                    if (graph[v][u]) {                         // if graph has the (v,u) edge ...
                        if ((prev & (1 << (u - 1))) != 0) {    // ... and if u is in the mask ...
                            curr |= d[prev][u];                // update the current value
                        }
                    }
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

```java
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];        // save one bit, reduce memory 2x times
    d[0][0] = 1;                                            // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {           // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {             // mask contains v
                int prev = mask ^ (1 << (v - 1));           // previous mask
                boolean curr = d[prev][0] && graph[v][0];   // consider 0 separately
                for (int u = 1; u < n; ++u) {               // check previous vertices
                    curr |= d[prev][u] && graph[v][u];      // if graph has the (v,u) edge, update
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

An efficient implementation

```
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];        // save one bit, reduce memory 2x times
    d[0][0] = 1;                                            // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {           // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {             // mask contains v
                int prev = mask ^ (1 << (v - 1));           // previous mask
                boolean curr = d[prev][0] && graph[v][0];   // consider 0 separately
                for (int u = 1; u < n; ++u) {               // check previous vertices
                    curr |= d[prev][u] && graph[v][u];      // if graph has the (v,u) edge, update
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

```
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];          // save one bit, reduce memory 2x times
    d[0][0] = 1;                                              // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {             // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {               // mask contains v
                int prev = mask ^ (1 << (v - 1));             // previous mask
                boolean curr = false;                         // consider 0 separately
                for (int u = 0; u < n; ++u) {                 // check previous vertices
                    curr |= d[prev][u] && graph[v][u];        // if graph has the (v,u) edge, update
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

```java
boolean[][] hamiltonianDP(boolean[][] graph) {
    int n = graph.length;
    boolean[][] d = new boolean[(1 << (n - 1))][n];     // save one bit, reduce memory 2x times
    d[0][0] = 1;                                          // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {         // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {           // mask contains v
                int prev = mask ^ (1 << (v - 1));         // previous mask
                boolean curr = false;                     // consider 0 separately
                for (int u = 0; u < n; ++u) {             // check previous vertices
                    curr |= d[prev][u] && graph[v][u];    // if graph has the (v,u) edge, update
                }
                d[mask][v] = curr;
            }
        }
    }
    return d;
}
```

An efficient implementation

```java
int[] hamiltonianDP(int[] graph) {                 // running time looks more like `O(2^n * n)`
    int n = graph.length;                          //          and memory more like `O(2^n)`
    int[] d = new int[(1 << (n - 1))];
    d[0] = 1;                                      // count vertices from 0
    for (int mask = 1; mask < d.length; ++mask) {  // locally ordered by size
        for (int v = 1; v < n; ++v) {
            if ((mask & (1 << (v - 1))) != 0) {    // mask contains v
                int prev = mask ^ (1 << (v - 1));  // previous mask
                if ((d[prev] & graph[v]) != 0) {   // if u exists with path 1-u and with edge (u,v)
                    d[mask] |= 1 << v;             // saying the path 1-v also exists
                }
            }
        }
    }
    return d;
}
```

- Does a Hamiltonian tour exist?

- Does a Hamiltonian tour exist?
  - Evaluate $d[S][v]$ for all S and v
  - If, for some $v \neq 1$, $d[V][v] = \text{TRUE}$ and $(v, 1) \in E$, then the Hamiltonian tour exists
  - Otherwise it does not exist
  - Running time: $O(2^{|V|} \cdot |V|) + O(|V|)$

- Does a Hamiltonian tour exist?
  - Evaluate $d[S][v]$ for all S and v
  - If, for some $v \neq 1$, $d[V][v] = \text{TRUE}$ and $(v, 1) \in E$, then the Hamiltonian tour exists
  - Otherwise it does not exist
  - Running time: $O(2^{|V|} \cdot |V|) + O(|V|)$
- Does a Hamiltonian path between a and b exist?

- Does a Hamiltonian tour exist?
  - Evaluate $d[S][v]$ for all S and v
  - If, for some $v \neq 1$, $d[V][v] = \text{TRUE}$ and $(v, 1) \in E$, then the Hamiltonian tour exists
  - Otherwise it does not exist
  - Running time: $O(2^{|V|} \cdot |V|) + O(|V|)$
- Does a Hamiltonian path between a and b exist?
  - Evaluate $d[S][v]$ for all S and v
  - If there exists $S' \subseteq 2^{V \setminus \{1,a,b\}}$, such that:
    - $d[S' \cup \{1, a\}][a] = \text{TRUE}$
    - $d[V \setminus S' \setminus \{a\}][b] = \text{TRUE}$

    then a Hamiltonian path between a and b exists, otherwise not
  - Simple special cases if $a = 1$ or $b = 1$
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$

- Does any Hamiltonian path exist?

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour
  - Values of $d[S][v]$ provide enough information to restore a path in $O(|V|^2)$

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour
  - Values of $d[S][v]$ provide enough information to restore a path in $O(|V|^2)$
- Count Hamiltonian paths/tours

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$then a Hamiltonian path exists between a and b
    - $\ldots$and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour
  - Values of $d[S][v]$ provide enough information to restore a path in $O(|V|^2)$
- Count Hamiltonian paths/tours
  - $d[S][v]$ stores the number of paths from 1 to v using vertices from S

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour
  - Values of $d[S][v]$ provide enough information to restore a path in $O(|V|^2)$
- Count Hamiltonian paths/tours
  - $d[S][v]$ stores the number of paths from 1 to v using vertices from S
- Shortest Hamiltonian path/tour (Traveling Salesperson Problem)

- Does any Hamiltonian path exist?
  - Evaluate $d[S][v]$ for all S and v
  - Check all $S' \subseteq V \setminus \{1\}$:
    - If exists $a \in S'$ such that $d[S' \cup \{1\}][a] = \text{TRUE}\ldots$
    - $\ldots$ and $b \notin S'$ such that $d[V \setminus S'][b] = \text{TRUE}\ldots$
    - $\ldots$ then a Hamiltonian path exists between a and b
    - $\ldots$ and this can be done in $O(1)$ per single S' using bit arithmetic!
  - Running time: $O(2^{|V|} \cdot |V|) + O(2^{|V|})$
- Restore a Hamiltonian path/tour
  - Values of $d[S][v]$ provide enough information to restore a path in $O(|V|^2)$
- Count Hamiltonian paths/tours
  - $d[S][v]$ stores the number of paths from 1 to v using vertices from S
- Shortest Hamiltonian path/tour (Traveling Salesperson Problem)
  - $d[S][v]$ stores the shortest length of a path from 1 to v using vertices from S

Special case: Every tournament has a Hamiltonian path.

Special case: Every tournament has a Hamiltonian path. Proof:

- Start building this path from an arbitary vertex, say, v_1
- Assume a path $v_1 \ldots v_k$ is built. Add a new vertex v:
  - If there is an edge $(v, v_1) \in E$, prepend v
  - Otherwise, if there is an edge $(v_k, v) \in E$, append v
  - Otherwise: find i such that $(v_i, v) \in E$ and $(v, v_{i+1}) \in E$ – it will exist because the graph is a tournament – then insert v between v_i and v_{i+1}