# Algorithmic Strategies 2024/25

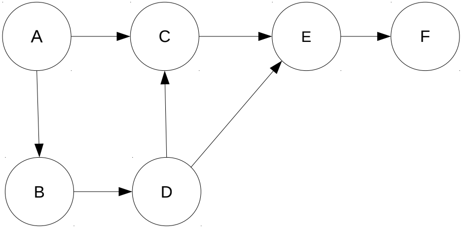
# Week 4 – Dynamic Programming

· U C ·

Universidade de Coimbra

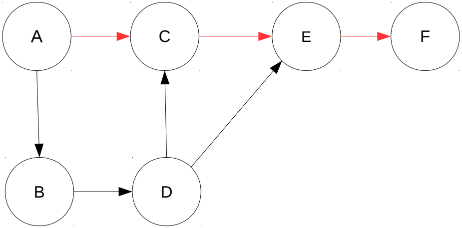
# Shortest path in an acyclic directed graph

- Given an acyclic directed graph, find the shortest path between two vertices.



# Shortest path in an acyclic directed graph

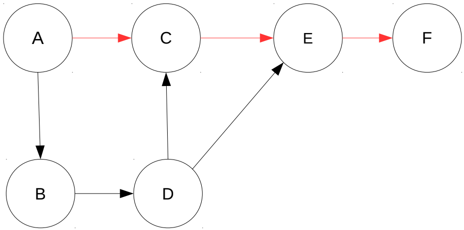
- Given an acyclic directed graph, find the shortest path between two vertices.




The shortest path from A to F is (A,C,E,F)

# Shortest path in an acyclic directed graph

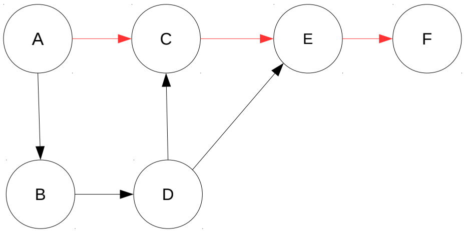
- Subproblem: Find the shortest path from source to a node $v$ in the graph.




It has optimal substructure – the shortest path from source to target contains the shortest path for small subproblems.

# Shortest path in an acyclic directed graph

- Subproblem: Find the shortest path from source to a node $v$ in the graph.




Example: The shortest path from A to F is (A,C,E,F) $\implies$
The shortest path from A to E is (A,C,E).

# Shortest path in an acyclic directed graph

Recursive algorithm to compute the value of the shortest path from node $s$ to a target node $t$. The call should be $Path(t, 0)$

---

**Function** $Path(v)$

  **if** $v = s$ **then**

    **return** 0

  $\ell = \infty$

  **for each** $(i, v) \in A$ **do**

    $\ell = \min\{\ell, Path(i) + 1\}$   {node $i$ connects to node $v$}

  **return** $\ell$

---

At each recursion, it computes the shortest path from node $s$ to a given node $v$ in the graph.

# Shortest path in an acyclic directed graph

Top-down DP to compute the value of the shortest path from node $s$ to a target node $t$. The call should be $Path(t, 0)$

---

**Function** $Path(v)$

  **if** $DP[v]$ is cached **then**
    **return** $DP[v]$
  **if** $v = s$ **then**
    **return** $0$
  $DP[v] = \infty$
  **for each** $(i, v) \in A$ **do**
    $DP[v] = \min\{DP[v], Path(i) + 1\}$   {node $i$ connects to node $v$}
  **return** $DP[v]$

---

At each recursion, it computes the shortest path from node $s$ to a given node $v$ in the graph.

# Shortest path in an acyclic directed graph

Bottom-up DP to compute the value of the shortest path from $s$ to a target node $t$. The nodes are visited according to a topological ordering (more about this in some weeks). Assume that $s$ is node 1 and $t$ is node $n$.

---

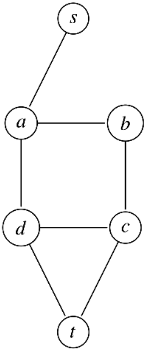
**Function** *Path*()

  $DP[1] = 0$
  **for** $v = 2$ **to** $n$ **do**
    $DP[v] = \infty$
  **for** $v = 2$ **to** $n$ **do**
    **for each** $(i, v) \in A$ **do**
      $DP[v] = \min\{DP[v], DP[i] + 1)\}$    {node $i$ connects to node $v$}
  **return** $DP[n]$

---

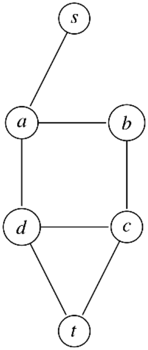
# Longest simple path in an undirected graph

- Given an undirected graph, find the longest simple path between two vertices.



# Longest simple path in an undirected graph

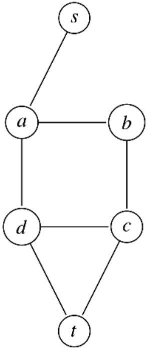
- Given an undirected graph, find the longest simple path between two vertices.




The longest simplest path from $s$ to $t$ is $(s, a, b, c, d, t)$

# Longest simple path in an undirected graph

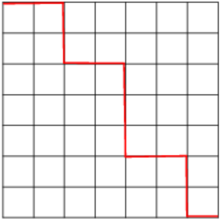
- Subproblem: Find the longest simple path from source to a node $v$ in the graph.




It has no optimal substructure – the longest simple path from $s$ to $t$ does not contain the longest simple path from $s$ to $d$, which is $(s, a, b, c, t, d)$.

# Number of monotonic paths

- Given a grid of size $n \times m$, count the number of monotonic paths.
- Monotonic means that you can only turn right or down



# Number of monotonic paths

## Recursive solution

```
Function count(x, y)
   if x = 1 or y = 1 then                 {base case}
      return 1
   C1 = count(x − 1, y)                    {recursion}
   C2 = count(x, y − 1)
   return C1 + C2
```

# Number of monotonic paths

## Top-down dynamic programming

```
Function count(x, y)
  if T[x, y] is cached then                    {if already visited}
    return T[x, y]
  if x = 1 or y = 1 then                       {base case}
    return 1
  C1 = count(x − 1, y)                         {recursion}
  C2 = count(x, y − 1)
  T[x, y] = C1 + C2                            {memoizing}
  return T[x, y]
```

# Number of monotonic paths

## Bottom-up dynamic programming

---

**Function** *count*$(n, m)$

  **for** $i = 1$ **to** $n$ **do** {1st base case}

    $T[i, 1] = 1$

  **for** $j = 1$ **to** $m$ **do** {2nd base case}

    $T[1, j] = 1$

  **for** $i = 2$ **to** $n$ **do**

    **for** $j = 2$ **to** $m$ **do**

      $T[i, j] = T[i-1, j] + T[i, j-1]$

  **return** $T[n, m]$

---

- Bottom-up approach in $O(mn)$ time.