Algorithms, Design & Analysis

Lecture 20: **Shortest Path**

Mihab & Maira

Information Technology University

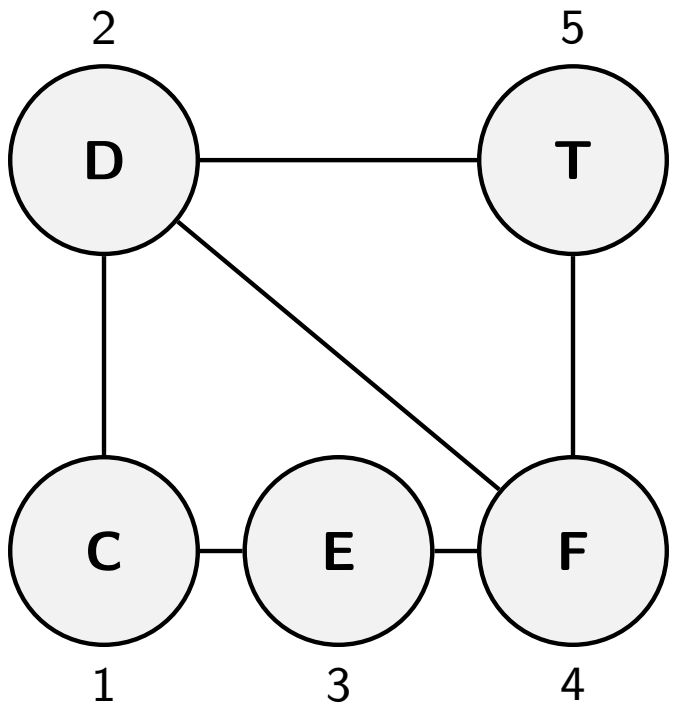
June 19, 2025

About Your Fellows

- Hi there! We are **Mihab Khan** and **Maira Fatima**.
- We are Associate Students at ITU.

Graph of Nodes C, D, E, F, T

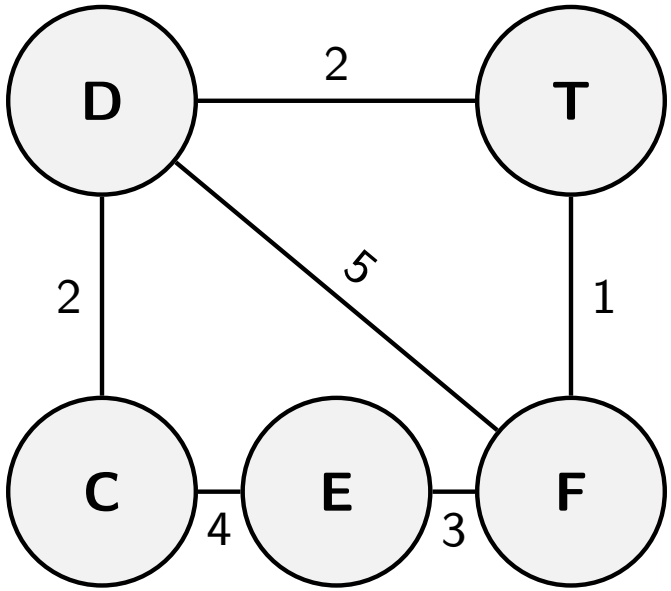
Note: We can write the table in any order, but maintaining the same order makes interpretation easier.



Shortest Path

Find the distance from node i to j, but there will be restricted subset of vertices that can be visited while finding the shortest path.

When all vertices are allowed in the graph, the shortest path is found.



This graph could have been negative, and it wouldn't have affected anything (process/solution).

$d(i, j, 0)$

	C	D	E	F	T
C	0				
D	2				
E	4				
F	∞				
T	∞				

Diagonal is on zero, so you don't have to go anywhere to visit the vertex where it is standing on (i.e. distance from D to D is 0 , C to C is 0 , E to E is 0 etc.)

$d(i, j, 0)$

	C	D	E	F	T
C	0	2			
D	2	0			
E	4	∞			
F	∞	5			
T	∞	2			

$d(i, j, 0)$

	C	D	E	F	T
C	0	2	4		
D	2	0	∞		
E	4	∞	0		
F	∞	5	3		
T	∞	2	∞		

$d(i, j, 0)$

	C	D	E	F	T
C	0	2	4	∞	
D	2	0	∞	5	
E	4	∞	0	3	
F	∞	5	3	0	
T	∞	2	∞	1	

$d(i, j, 0)$

	C	D	E	F	T
C	0	2	4	∞	∞
D	2	0	∞	5	2
E	4	∞	0	3	∞
F	∞	5	3	0	1
T	∞	2	∞	1	0

d(i, j, 1)

$d(i, j, 1)$: **Allowed to visit vertex 1 {C}**

	C	D	E	F	T
C	0	2	4	∞	∞
D	2	0	6	5	2
E	4	6	0	3	∞
F	∞	5	3	0	1
T	∞	2	∞	1	0

$d(D, E, \{C\})$
$= d(D, C, \{\}) = 2$
$+ d(C, E, \{\}) = 4$
$= 6$

$d(D, F, 1) = \min(5,$
$d(D, C, 0) = 2 +$
$d(C, F, 0) = \infty)$
$= \min(5, \infty)$
$= 5$

- C $\rightarrow$ C $= 0$ (same node)
- D $\rightarrow$ C $= 2$
- E $\rightarrow$ C $= 4$
- F and T have no direct path to C, so it's ∞

Only direct paths or paths going through C are considered.

$d(i, j, 2)$

$d(i, j, 1)$: **Allowed to visit 2 vertices {C,D}**

	C	D	E	F	T
C	0	2	4	7	4
D	2	0	6	5	2
E	4	6	0	3	8
F	7	5	3	0	1
T	4	2	8	1	0

Assadullah volunteered to solve the matrix

d(i, j, 3)

$d(i, j, 3)$: **Allowed to visit 3 vertices {C,D,E}**

	C	D	E	F	T
C	0	2	4	7	4
D	2	0	6	5	2
E	4	6	0	3	8
F	7	5	3	0	1
T	4	2	8	1	0

Saifullah volunteered to solve the matrix

Asharib proposed the question:
"If there are no ∞ values, does that mean we have found all shortest paths?"

Professor replied:
"Just because there is no ∞, it doesn't mean that every path is the shortest."

Hence, we need to perform all steps till the end.

d(i, j, 4)

To make it interesting , professor erased the graph from board.

$d(i, j, 4)$: **Allowed to visit 4 vertices {C,D,E,F}**

	C	D	E	F	T
C	0	2	4	7	4
D	2	0	6	5	2
E	4	6	0	3	4
F	7	5	3	0	1
T	4	2	4	1	0

Wasif volunteered to solve the matrix

He mentioned that rest of the matrix will remain same just the distance b/w E & T will be updated.

$d(i, j, 5)$

$d(i, j, 5)$: **Allowed to all vertices {C,D,E,F,T}**

	C	D	E	F	T
C	0	2	4	7	4
D	2	0	6	5	2
E	4	6	0	3	4
F	7	5	3	0	1
T	4	2	4	1	0

Matrix will not change because all shortest paths are discovered

Floyd-Warshall

Algorithm:

$$d_{ij} = \begin{cases} w_{ij} & \text{if } w_{ij} \neq 0 \quad \textit{(if there is a edge, edge weight is the shortest path)} \\ \infty & \text{otherwise} \quad \textit{(if no edge, weight is } \infty\textit{)} \end{cases}$$

for $k = 1$ to n :
  for $i = 1$ to n;
  for $j = 1$ to n;
$d(i,j) = \min(d(i,j),\ d(i,k) + d(k,j))$
ret d

(This algo is giving the distance between vertices not the shortest path.)

H.W Q:How do we use Floyd-Warshall to find negative cycles in graph?
H.W Q:Rewrite this algo to find shortest path?

Dynamic Programming

Dynamic Programming is used to solve optimization problems.

- *Design paradigm*
- *Used in special circumstances*

Back in the 1950s, a person introduced a method with a technical and complex name. To make it sound more appealing, especially for funding purposes in the US, he renamed it "Dynamic Programming". The new name helped it gain popularity. Interestingly, the name has nothing to do with the actual method.There is neither anything "dynamic" nor any "programming" involved in the technique, yet the term "DP" stuck, and that is what everyone started calling it.

Dynamic Programming

Requirement:

1) The problem should have recursive aspect.

(You should be able to write problem as recurrence)

2)Over-Lapping Subproblem

Fibonacci Recursive

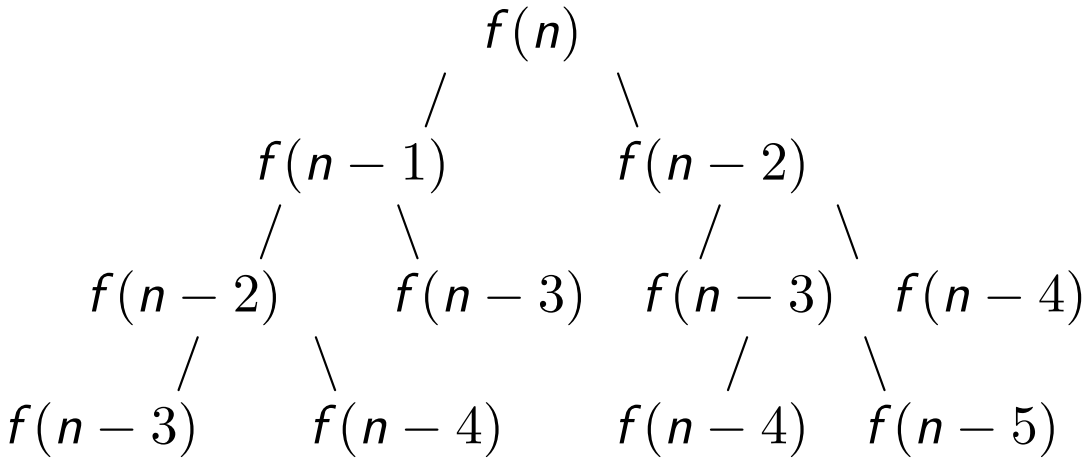
(Without Dynamic Programming, you'd typically write a recursive function, which can be inefficient. Instead, using DP is a better approach.)

Recursive Function Example:

```
fib(n)
{
   if (n <= 2)
     return n - 1;
   else
     return fib(n - 1) + fib(n - 2);
}
```

(This algorithm computes the distance between vertices, not the shortest path.)

Recursive Tree for `f(n)`:



Overlapping Subproblems

Dynamic programming exploits this overlap. Instead of re-computing the same subproblems, we solve each one once, store the result, and reuse it when needed.

Memoization

memo - means small letter

DP says that write this thing on memo and use that memo whenever you need.

```
fib(n)
{
   memo = []
     memo[0] = 0
     memo[1] = 1
   for(i = 2; i < n; i++) {
     memo[i] = memo[i-1] + memo[i-2]
   }
   return memo[n-1]
}
```

Time Complexity is O(n).

Difference in recursive and dynamic programming is time complexity

Hamza pointed out it increases space complexity

Professor replied:
Dynamic programming has a memory impact, storing all solutions. In this case, we maintain a stack of at least size n. We make n calls for $f(n)$, we store $f(n-2)$, $f(n-3)$, etc., so the entire stack is linear. However, with a clever approach, you can achieve the same result using a constant amount of memory. In general, dynamic programming requires a table that typically requires a linear amount of memory.

Real-World Context: DNA and Disease Cure Discovery

- In the fight against emerging diseases, researchers often need rapid cures for newly discovered pathogens.
- These pathogens have DNA/RNA sequences that can be compared with those of known viruses or bacteria.

Why Compare DNA Sequences?

- DNA carries biological instructions.
- Similar DNA $\rightarrow$ similar biological behavior and possible treatment response.

Edit Distance and Drug Development

Edit Distance as a Measure of Similarity

- Counts insertions, deletions, substitutions needed to convert one DNA sequence into another.
- Small edit distance $\Rightarrow$ high similarity $\Rightarrow$ possibly similar treatment.

Using Known Treatments:

- Genetically close pathogens (low edit distance) suggest reuse or slight modification of existing drugs.

Optimizing Drug Design:

- Bioinformatics tools use edit distance to align sequences and highlight mutations.
- Predict impact of mutations and suggest changes to drugs.

Why Efficient Algorithms Matter

- DNA sequences can be millions of base pairs long.
- Brute-force comparison is too slow.
- Dynamic programming-based edit distance algorithms make large-scale comparisons feasible.

"Imagine you're a computational biologist working on a virus like COVID-25. You're given its genome. Your task: Find which known virus it is genetically closest to. How do you measure that?"

$\Rightarrow$ Leads directly into the Edit Distance Problem

Example

Example:
String 1: AAAACTGG
String 2: GCATG
We are allowed to:

- insert letter (cost $= 1$)
- delete letter (cost $= 1$)
- Exchange/Swap letter (cost $= 1$)
- Match (cost $= 0$)

This problem becomes very easy with dynamic programming

It is not a string matching problem
because none of the strings are ever gonna be exactly same

Example: Solution Steps

1. **Match: "A" (String 1) with "A" (String 2)**
   No change needed.
2. **Delete: "A" (String 1)**
   String 1 becomes "AACTGG".
3. **Delete: "A" (String 1)**
   String 1 becomes "ACTGG".
4. **Match: "C" (String 1) with "C" (String 2)**
   No change needed.
5. **Exchange: Swap "T" and "G" (String 1)**
   String 1 becomes "ACGTTG".
6. **Match: "G" (String 1) with "G" (String 2)**
   No change needed.
7. **Delete: "T" (String 1)**
   String 1 becomes "ACGTG".
8. **Delete: "G" (String 1)**
   String 1 becomes "ACGT".
9. **Insert: "T" (String 2)**
   String 1 becomes "GCATG".

Example: Final Result

Final Result:

- **Operations**:
  - 3 Matches
  - 4 Deletions
  - 1 Insertion
  - 1 Exchange
- **Total Operations: 9**
- **Total Cost**: $0 \times 3 + 1 \times 4 + 1 \times 1 + 1 \times 1 = 6$
- **Distance**:

$$d(\text{String 1}, \text{String 2}) = 4\,(\text{deletions}) + 1\,(\text{insertion}) + 1\,(\text{exchange}) = 6$$

So, the distance $d(\text{String 1}, \text{String 2}) = 6$.