# Dynamic Programming

# Contents


- Introduction
- Calculation the binomial coefficient
- Shortest paths problem
- Complexity time of Floyd algorithm

# Introduction

- Divide-and-Conquer is a top-down method.
  - Divide the problem / instance into smaller and smaller sub-problem / sub-instances.
  - Combine the sub-solutions thus obtained the solution of the original problems.
  - It commonly uses with recursion.
  - But, it requires extra memory to all the intermediate arguments and return values on the system's internal stack.
  - And,

# Introduction

- Divide-and-Conquer is a top-down method.
  - Using recursive method to solve the problem, there would have some overlapping sub-instances to be solved.
  - These overlapping sub-instances / duplication will cause the algorithm inefficient.
- The dynamic programming is used to solve the above problem, avoid calculating the same thing twice.

# Introduction

- Dynamic programming on the other hand is bottom-up technique.
  - Start with the smallest, and hence the simplest, sub-problems / sub-instances.
  - By combining their solutions, obtain the answers to sub-problems / sub-instances of increasing size.
  - Until finally arrive at the solution of the original instance.
  - Usually keeps a table of known results which fills up as sub-instances are solved.

# Calculation the binomial coefficient

- Consider the problem of calculating the binomial coefficient:
  - $_{n}C_{r} = 1$ *if $r = 0$ or $r = n$*
  - $_{n}C_{r} = {}_{n-1}C_{r-1} + {}_{n-1}C_{r}$ *if $0 < r < n$*
  - $_{n}C_{r} = 0$ *otherwise*

# Calculation the binomial coefficient

- Use Recursion method to solve the binomial coefficient problem:

```
int C(int n, int r)
{
   if (r == 0 || n == r)
     return 1;
   else
     return C(n-1, r-1) + C(n-1,r);
}
```

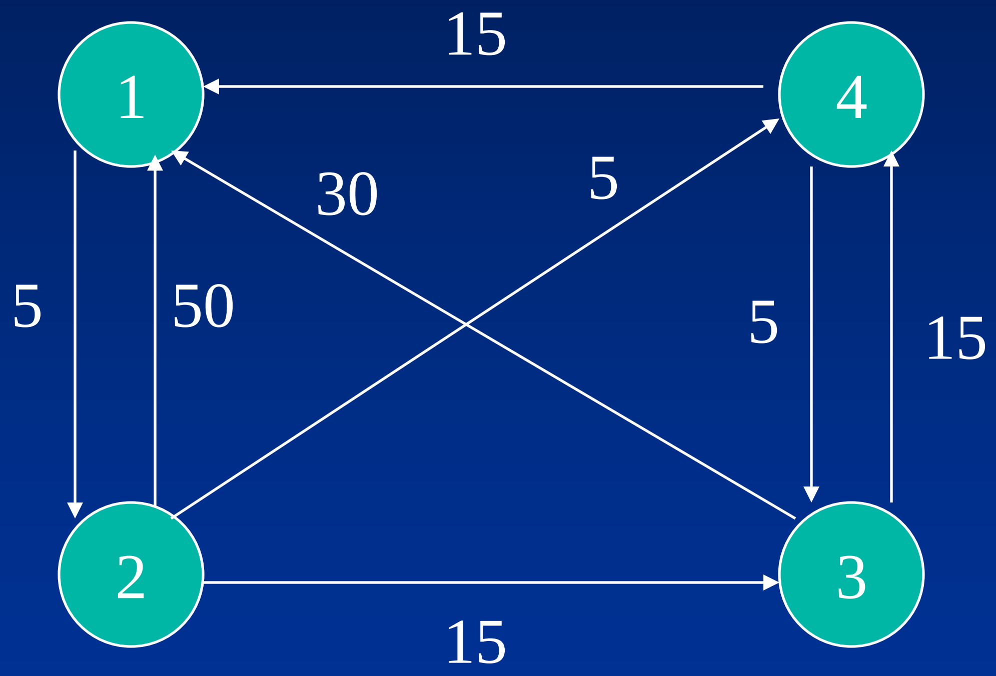
# Calculation the binomial coefficient

- *Pascal's triangle*.

| | 0 | 1 | 2 | 3 | 4 | … | *r*-*1* | *r* |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | | | | | | | |
| 1 | 1 | 1 | | | | | | |
| 2 | 1 | 2 | 1 | | | | | |
| 3 | 1 | 3 | 3 | 1 | | | | |
| 4 | 1 | 4 | 6 | 4 | 1 | | | |
| . | | | | | | | | |
| . | | | | | | | | |
| . | | | | | | | | |
| n-1 | | | | | | | $C(n-1,r-1)$ | $C(n-1,r)$ |
| | | | | | | | | + |
| n | | | | | | | | $C(n,r)$ |

# Shortest paths problem

- Calculate the length of the shortest path between *each pair of vertices*.



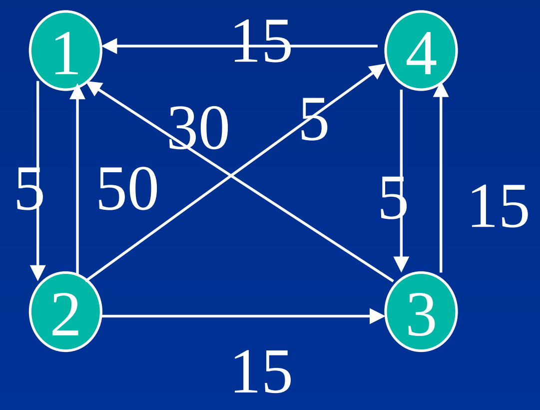
# Floyd's algorithm

```
public void Floyd (int[ ] arr, int n)
{
    for( int k =1; k < n; k++)
        for (int i = 1; i < n; i++)
            for (int j = 1; j < n; j++)
                arr[i,j] = min(arr[i,j], arr[i,k] + arr[k,j]);
}
```

# Shortest paths problem

$D_0 = L =$

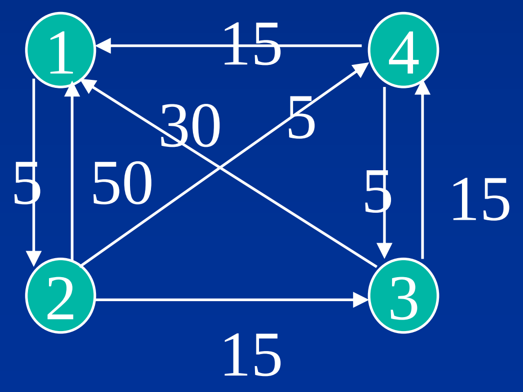
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | Inf | Inf |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | Inf | 0 | 15 |
| 4 | 15 | Inf | 5 | 0 |



# Shortest paths problem

$D_1$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | Inf | Inf |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

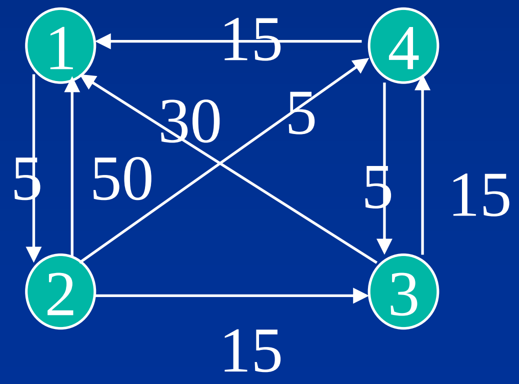



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | Inf | Inf |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | Inf | 0 | 15 |
| 4 | 15 | Inf | 5 | 0 |

# Shortest paths problem

$D_2$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 20 | 10 |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

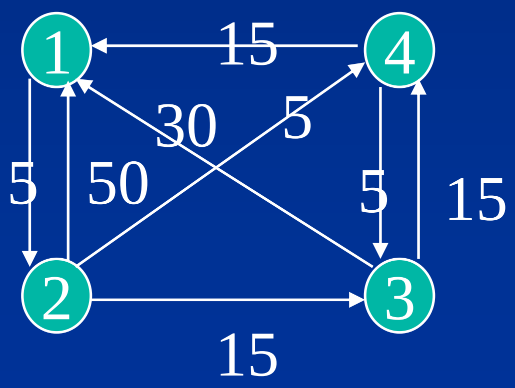



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | Inf | Inf |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

# Shortest paths problem

$D_3$ =

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 20 | 10 |
| 2 | 45 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

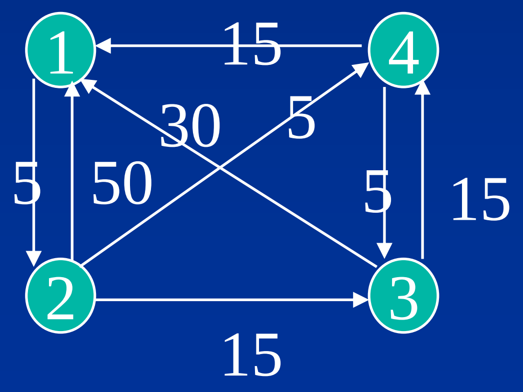



| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 20 | 10 |
| 2 | 50 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

# Shortest paths problem

$D_4 =$

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 15 | 10 |
| 2 | 20 | 0 | 10 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 5 | 20 | 10 |
| 2 | 45 | 0 | 15 | 5 |
| 3 | 30 | 35 | 0 | 15 |
| 4 | 15 | 20 | 5 | 0 |

# Complexity time

- The complexity time is $O(n^3)$
- The inner loop needs $O(n)$ times.
- The middle loop also needs n times, each time would do inner loop, so, if include the inner loop, the actual running times for middle loop is n*n, $O(n^{2)}$.
- So, for the outer loop, the complexity time is $n^* n^2$, equals to $O(n^3)$.