# Lecture 5: Algorithms for Multidimensional Arrays

# Multidimensional Arrays:

## Representing Matrices and other Multidimensional Objects:

```
T = [[11, 12, 5, 2], [15, 6,10,19], [10, 8, 12, 5], [12,15,8,6]]

print(T[0])

print(T[1][2])
>>[11, 12, 5, 2]
>>10
```

# Matrix Operations

```
def matAdd(A,B,C):
    n=len(A)
    m=len(A[0])
    assert len(B)==n and len(B[0])==m
    assert len(C)==n and len(C[0])==m
   for i in range(n)
        for j range(m):
            C[i][j]=A[i][j]+B[i][j]
```

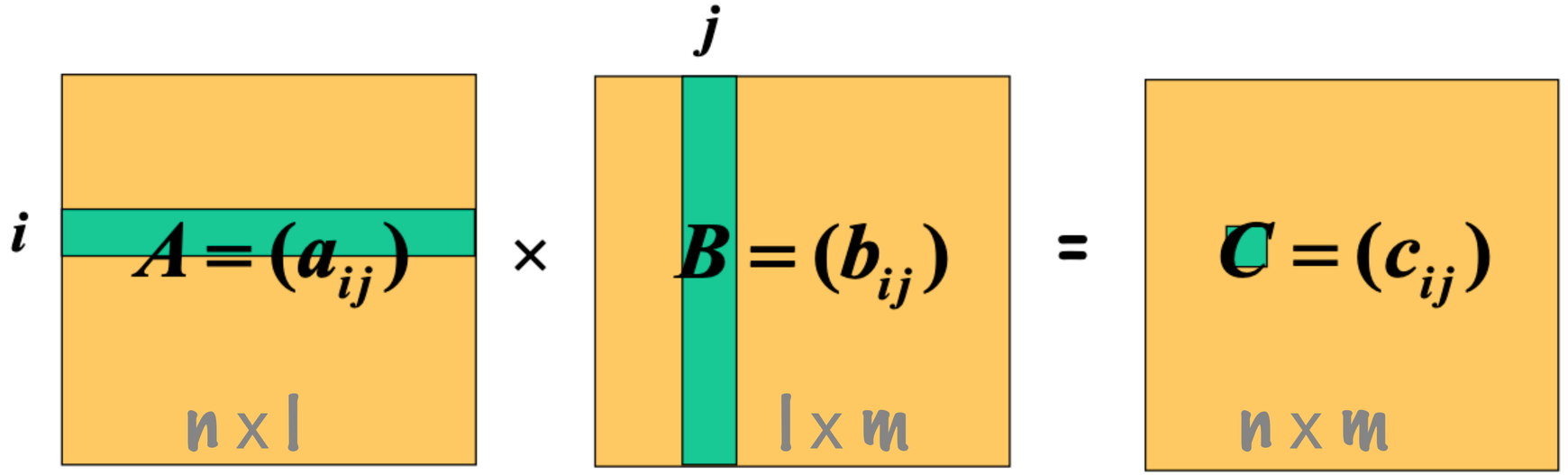
Time Complexity O(nm)

# Matrix Operations

Time Complexity O(nm)

```
#Version which creates a new array C
def matAdd(A,B):
    n=len(A)
    m=len(A[0])
    assert len(B)==n and len(B[0])==m
    C=[[A[i][j]+B[i][j] for j in range(m)] for i in range(n)]
    return C
```

# Matrix Multiplication

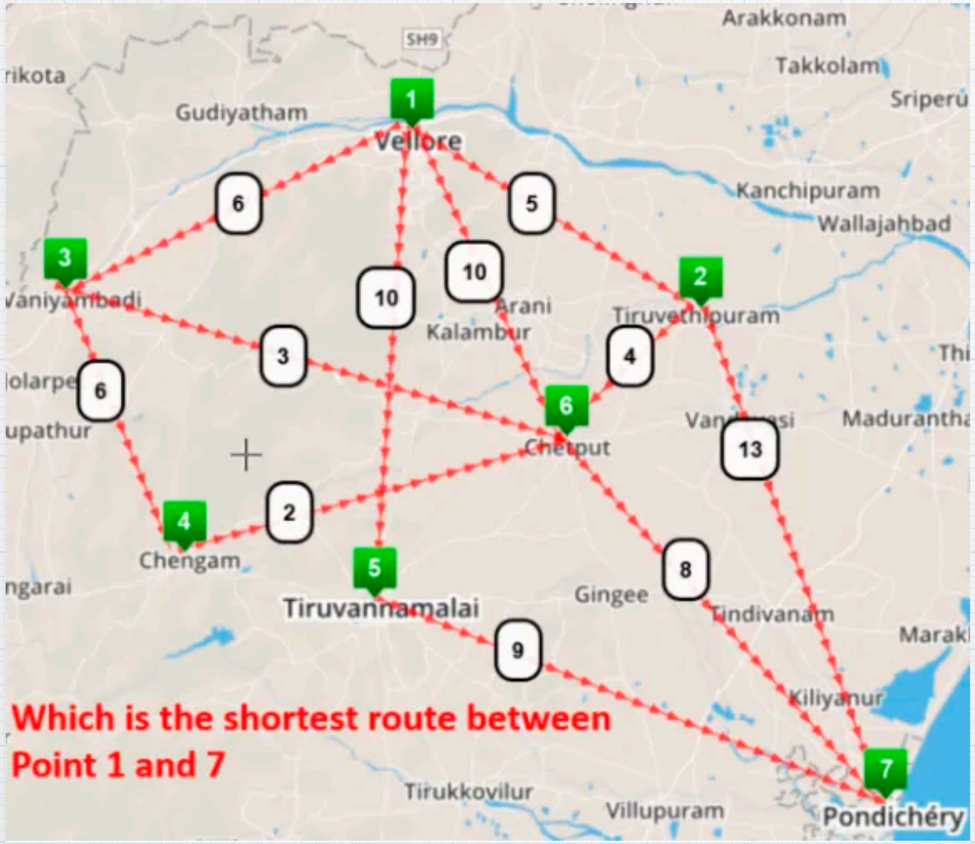



$$c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}$$

```
def matmult(A,B,C):
        n=len(A)
        l=len(A[0])
        m=len(B[0])
        assert len(B)==l and len(C)==n  and len(C[n-1])==m
        for i in range(n)
            for j range(m):
                C[i][]j]=0
                for k in range(l):
                        C[i][j]+= A[i][k]*B[k][j]
```

Time Complexity O(nlm)

# Representing real world Problems




| Adj | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | 5 | 6 | | 10 | 10 | |
| 2 | 5 | | | | | 4 | 13 |
| 3 | 6 | | | 6 | | 3 | |
| 4 | | | 6 | | | 2 | |
| 5 | 10 | | | | | | 9 |
| 6 | 10 | 4 | 3 | 2 | | | 8 |
| 7 | | 13 | | | 9 | 8 | |

Adjacency Matrix (weighted)

# Adj Matrix representation for Graphs

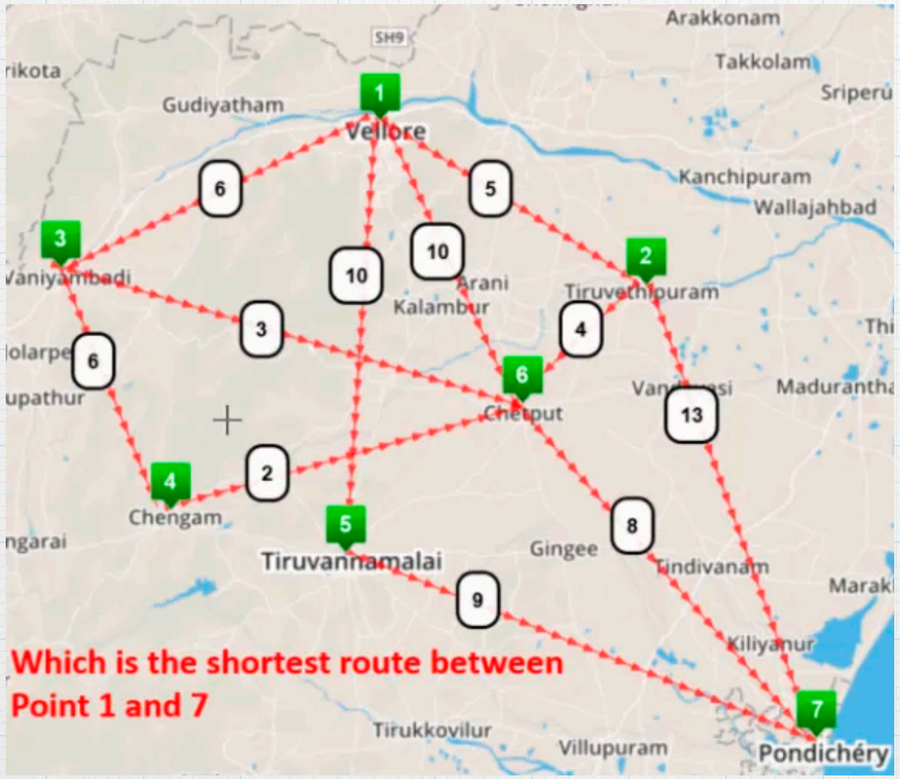
## Adjacency Matrix (Un-weighted)

| Adj | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

## Adjacency Matrix (weighted) with A[i,i]=0 & A[i,j]=∞ if no edge between i & j.

| Adj | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 5 | 6 | ∞ | 10 | 10 | ∞ |
| 2 | 5 | 0 | ∞ | ∞ | ∞ | 4 | 13 |
| 3 | 6 | ∞ | 0 | 6 | ∞ | 3 | ∞ |
| 4 | ∞ | ∞ | 6 | 0 | ∞ | 2 | ∞ |
| 5 | 10 | ∞ | ∞ | ∞ | 0 | ∞ | 9 |
| 6 | 10 | 4 | 3 | 2 | ∞ | 0 | 8 |
| 7 | ∞ | 13 | ∞ | ∞ | 9 | 8 | 0 |

# Solution




Possible Paths:

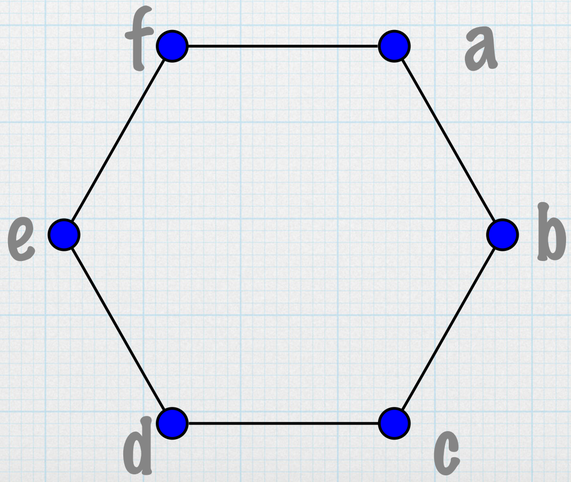
| | | |
|---|---|---|
| 1-2-7 | 5+13=18 | |
| 1-6-7 | 10+8=18 | |
| 1-5-7 | 10+9=19 | |
| 1-2-6-7 | 5+4+8=17 | Shortest |
| 1-3-6-7 | 6+3+8=17 | alt shortest |

# Solving For Shortest Path

- Idea use recursion — Can we represent $SP_k(i,j)$ in terms of $SP_{k-1}(i,j)$?
- What to use for k? we want to start with something trivial for base case and end with $SP_n(i,j)$ giving us Shortest path from i to j for each i,j?
- Such an approach is called Dynamic Programming
  - Represent Optimizing an instance as finding optimal among a number of smaller sub-problems
  - principle of optimal — we need to show optimal from the sub-problems leads to optimal overall

# Dynamic Programming: Principle of Optimality

- **Definition: A problem is said to satisfy the Principle of Optimality if the subsolutions of an optimal solution of the problem are themsleves optimal solutions for their subproblems.**
- **Examples:**
  - **The shortest path problem satisfies the Principle of Optimality.**
  - **This is because if a,x1,x2,...,xn,b is a shortest path from node a to node b in a graph, then the portion of xi to xj on that path is a shortest path from xi to xj.**
  - **The longest path problem, on the other hand, does not satisfy the Principle of Optimality. Take for example the undirected graph G of nodes a, b, c, d, and e, and f edges (a,b) (b,c) (c,d) (d,e) (e,f) and (f,a). That is, G is a ring. The longest (noncyclic) path from a to e is a,b,c,d,e. The sub-path from b to c on that path is simply the edge b,c. But that is not the longest path from b to c. Thus, the subpath on a longest path is not necessarily a longest path.**



# Dynamic Programming (cont'd)

**Dynamic programming design involves 4 major steps:**

1. **Develop a mathematical notation that can express any solution and subsolution for the problem at hand.**

2. **Prove that the Principle of Optimality holds.**

3. **Develop a recurrence relation that relates a solution to its subsolutions, using the math notation of step 1. Indicate what the initial values are for that recurrence relation, and which term signifies the final solution.**

4. **Write an algorithm to compute the recurrence relation.**

# Floyd Warshall Shortest path algorithm

$SP_k(i,j)$ => $i—x_1—x_2—x_3—...—j$, $x_1,x_2,x_3... \in \{1,2,..k\}$

* Define $SP_k(i,j)$ as shortest path from i to j which may use only intermediate vertices {,1,2,..k}
* $SP_0(i,j)$ means no intermediate vertices so cost of edge from i,j if it exists.
* $SP_{k+1}(i,j)$= either a path that does not use k+1 or a path of the type

  $i—x_1—...—k+1—...—j$ => $SP_k(i,k+1)+SP_k(k+1,j)$

  $SP_{k+1}(i,j)$= min { $SP_k(i,j)$, $SP_k(i,k+1)+SP_k(k+1,j)$ }
* Termination: $SP_n(i,j)=SP(i,j)$

Note; above vertex no's are 1,2..k whereas the corresponding array indices are 0,1,..k-1

```
def shortestpath(A,S):
    n=len(A)
    for i in range(n):    #Initialize S=SP0(i,j)
        for j range(n):
            if i==j:
                S[i][j]=0
            else:
                S[i][j]=A[i][j]
    for k in range(n):
        for i in range(n)
            for j range(n):
                S[i][j]=min(S[i][j], S[i][k]+S[k][j])
```

* Complexity of Floyd Warshall $O(n^3)$
  * Limitations — what if edge costs are negative?

* Other Dynamic Programming Problems
  * Recall Longest Common Subsequence
  * String matching eg Compare two DNA sequences with errors — minimum edit distance!

# Python Input & Output

- Console Input  x=input([prompt])

```
n=int(input("Please enter a number"))
if n % 2 == 0:
    print("Even")
else:
    print("odd")
```

# Processing String input: str.split(sep)

split(x) : converts string into a list by splitting string on occurrences of x

Split the string, using comma, followed by a space, as a separator:

```
txt = input()
>>>hello, my name is Peter, I am 26 years old
x = txt.split(", ")
print(x)
['hello', 'my name is Peter', 'I am 26 years old']
str = input()
>>> 23, 12,19,18,  16   ,9
l=str.split(',')
L=[int(x)  for x in l]
>>> L
[23, 12, 19, 18, 16, 9]
```

- Python output:

  print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

```
print ('IIT Delhi \n is best for Col100.')
   \n :is used to add a new blank line while printing a statement.
```

```
import time
for i in reversed(range(4)):
  if i > 0:
      print(i, end='>>>',flush=True)
      time.sleep(1)
  else:
      print('Start')
```

print(“Hello",20,sep=':')
>Hello:20

## Output formatting

```
>>> x = 5; y = 10
>>> print('The value of x is {} and y is {}'.format(x,y))
The value of x is 5 and y is 10

print('I love {0} and {1}'.format('bread','butter'))
print('I love {1} and {0}'.format('bread','butter'))
I love bread and butter
I love butter and bread

>>> print('Hello {name}, {say}'.format(say ='Goodmorning',name= 'John'))
Hello John, Goodmorning
```

Inside the placeholders you can add a formatting type to format the result:

```
:f fixed point format
txt = "The price is {:.2f} dollars."
print(txt.format(45))
The price is 45.00 dollars.
:^ Center aligns the result (within the available space)
:+ Use a plus sign to indicate if the result is positive or negative
:% Percentage format
```

see https://www.w3schools.com/python/ref_string_format.asp for more

# File Input & Output

```
f = open(“demofile.txt”,"at")  – open a text file for writing at end
```

**Note:** Make sure the file exists, or else you will get an error.

| Character | Meaning |
|---|---|
| `'r'` | open for reading (default) |
| `'w'` | open for writing, truncating the file first |
| `'x'` | open for exclusive creation, failing if the file already exists |
| `'a'` | open for writing, appending to the end of the file if it exists |
| `'b'` | binary mode |
| `'t'` | text mode (default) |
| `'+'` | open for updating (reading and writing) |

```
f.write("Now the file has more content!")
f.close()
```

see https://docs.python.org/3/library/functions.html#open for more details

# Reading Input

```
with open('workfile') as f:
      read_data = f.read(size)
```

`str=f.readline()` reads a single line from the file;

list=f.readlines()  reads all lines in the file creating a list of lines

list[line1,line2,…..]

Another way of reading a file line-by-line: by iterating over the file object in a for  loop

```
for line in f:
    print(line)
```