Tusk 1

cdef vs def

Cython is a programming language that allows you to write Python code that can be compiled to C for speed and efficiency. Cython has three ways of defining functions: def, cdef and cpdef.

- **def** functions are regular Python functions that can be called from both Python and Cython code. They have the same syntax and semantics as Python functions, but they are compiled to C by Cython for faster execution. def functions can take any type of argument, including Python objects, but they always return a Python object.

- **cdef** functions are C functions that can only be called from Cython or C code. They have a more restricted syntax and semantics than Python functions, but they allow you to specify the types of the arguments and the return value for better performance and compatibility with C libraries. cdef functions can take any type of argument, including pointers and structs, but they cannot take *args or **kwargs. cdef functions can also return any type of value, including void.

- **cpdef** functions are a hybrid of def and cdef functions. They cause Cython to generate both a cdef function (for fast calls from Cython or C) and a def function (for calls from Python). The def function internally calls the cdef function with the same arguments and returns its result. cpdef functions have the same syntax and semantics as def functions, but they can also specify the types of the arguments and the return value like cdef functions. cpdef functions can take any type of argument that def or cdef functions can take, but they cannot have *args or **kwargs.

The main difference between these three types of functions is in where they can be called from and how fast they are. def functions are the most flexible and compatible, but they are also the slowest. cdef functions are the most efficient and powerful, but they are also the most limited and incompatible. cpdef functions are a compromise between the two, offering both speed and flexibility.

To declare a string variable in Cython, you can use either str or bytes types, depending on whether you want a Unicode string or a byte string. For example:

```
cdef str s = "Hello"

cdef bytes b = b"World"
```

An example of using cpdef. cpdef is a way of defining a function that can be called from both Python and Cython code, without losing performance or flexibility. Here is a simple example of a cpdef function that computes the factorial of a number:

```
cpdef int factorial(int n):
    cdef int i, result
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

This function can be called from Python code like this:

```
>>> from factorial import factorial
>>> factorial(5)
120
```

Or from Cython code like this:

```
cimport factorial
```

cdef int x = factorial.factorial(5)

print(x)

In both cases, the function will run at C speed and avoid any Python overhead.

.....

Tusk 2

BFS and DFS

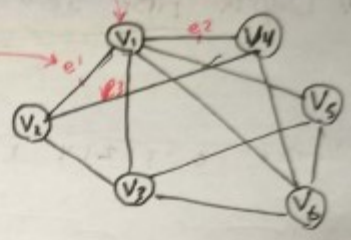
"Data Structure Graphs"

Note = vertex = Vertices

$G = (V, E)$

Graph, vertices, Edges

Edges



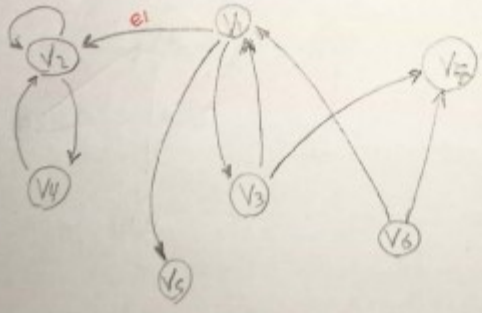
$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$

$E = \{e_1, e_2, e_3, \ldots\ldots, e_n\}$

$e_1 = (v_1, v_2)$ $\quad e_2 = (v_1, v_4)$

Graphs Types

Directed



where

$e_1 = (v_1, v_2) \neq (v_2, v_1)$

ex

undirected

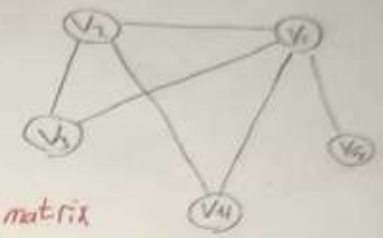
where

$e_1 = (v_1, v_2) = (v_2, v_1)$

Graphs Representation

1) Undirected

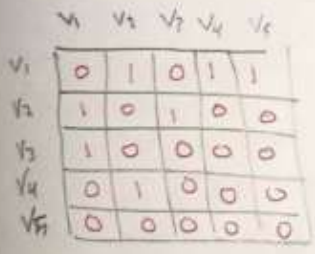
using 2 dimentional Array

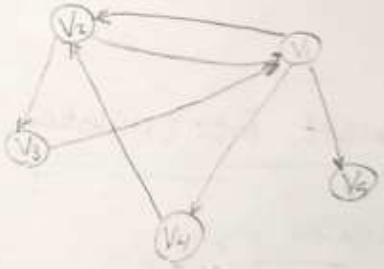


* adjacency matrix
* Symetric
* $G_{1,2} = G_{2,1}$

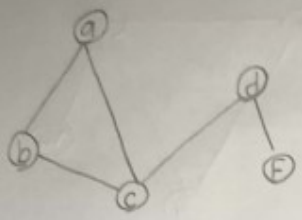
$(V_a, V_b) = (V_b, V_a)$

2) Directed



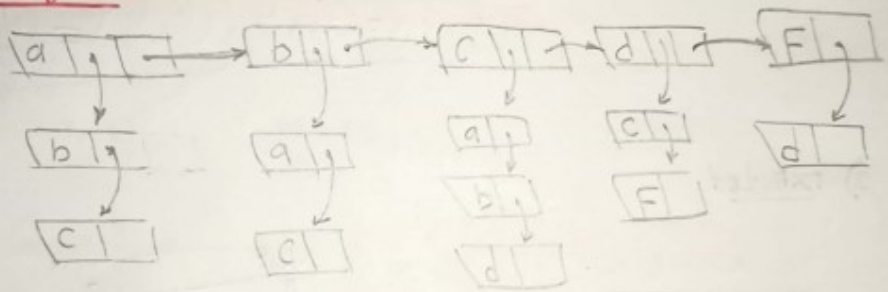


* anti Symetric
* $G_{1,4} \neq G_{4,1}$
* $(V_a, V_b) \neq (V_b, V_a)$
* adjacency matrix



a b c
b a c
c a b d
d c F
f d

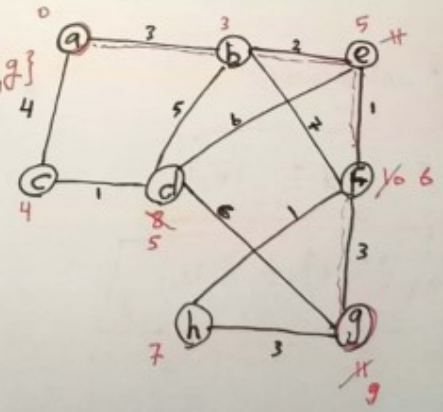
* adjacency list



Shortest Path (Dijkstra Algorithm)

weighted Graph

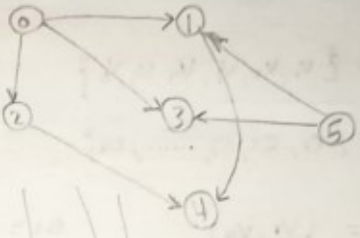
Visited Verteces = {a, b, c, d, e, F, h, g}



Graph Traversal ← Visiting each vertex only one time.

1) Depth First search (DFS)

Visited | 0 | 3 | 2 | 4 | 1 | 5 |

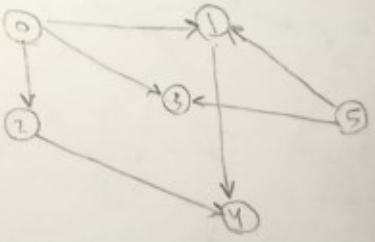


Stack

∴ DFS 0, 3, 2, 4, 1, 5

2) Breadth first search (BFS)

visited | 0 | 1 | 2 | 3 | 4 | 5 |



Queue

| 0 | 1 | 2 | 3 | 4 | 5 |

∴ BFS 0, 1, 2, 3, 4, 5