Sunbeam Institute of Information Technology
Pune and Karad

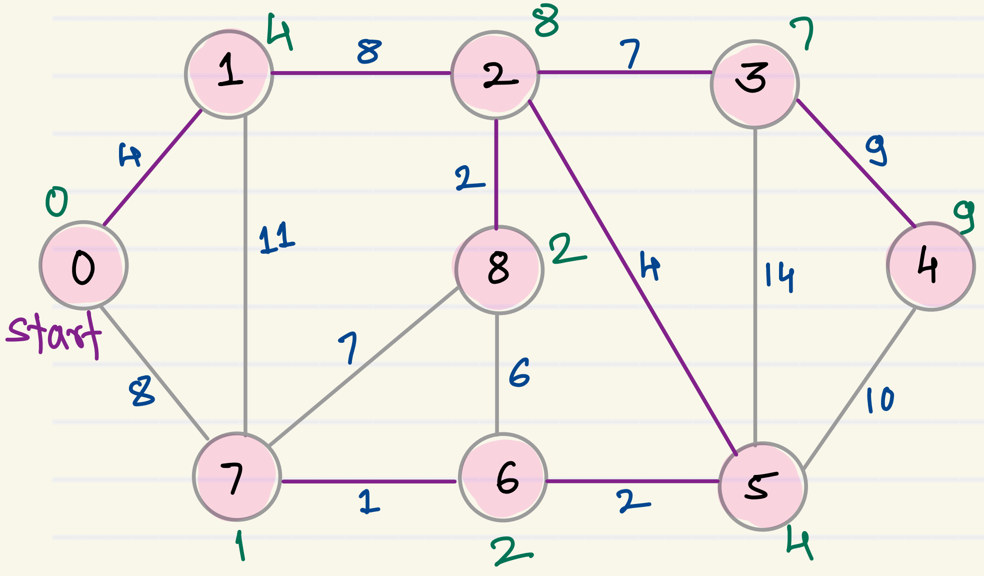
Data structures and Algorithms

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Prim's MST

- **Initialisations**
  - Create a set mst to keep track of vertices included in MST.
  - Also keep track of parent of each vertex. Initialise parent of each vertex to -1.
  - Assign a key to all vertices in the input graph. Key of all vertices should be initialised to INF. The start vertex key should be 0.

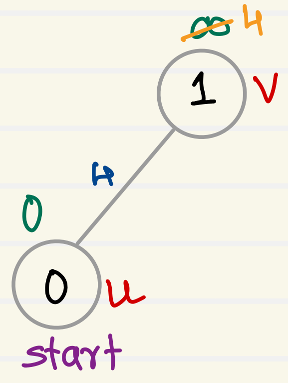


	Key	Parent
0	0	-1
1	4	0
2	8	1
3	7	2
4	9	3
5	4	2
6	2	5
7	1	6
8	2	2

wt = 37

- **Algorithm**

  While mst doesn't include all vertices
  1. Pick a vertex u which is not there in mst and has minimum key.
  2. Include vertex u in mst.
  3. Update key and parent of all adjacent vertices of u which are not there in mst
     i. For each adjacent vertex v, if weight of edge u-v is less than the current key of v, then update the key as weight of u-v.
     ii. Record u as parent of v.

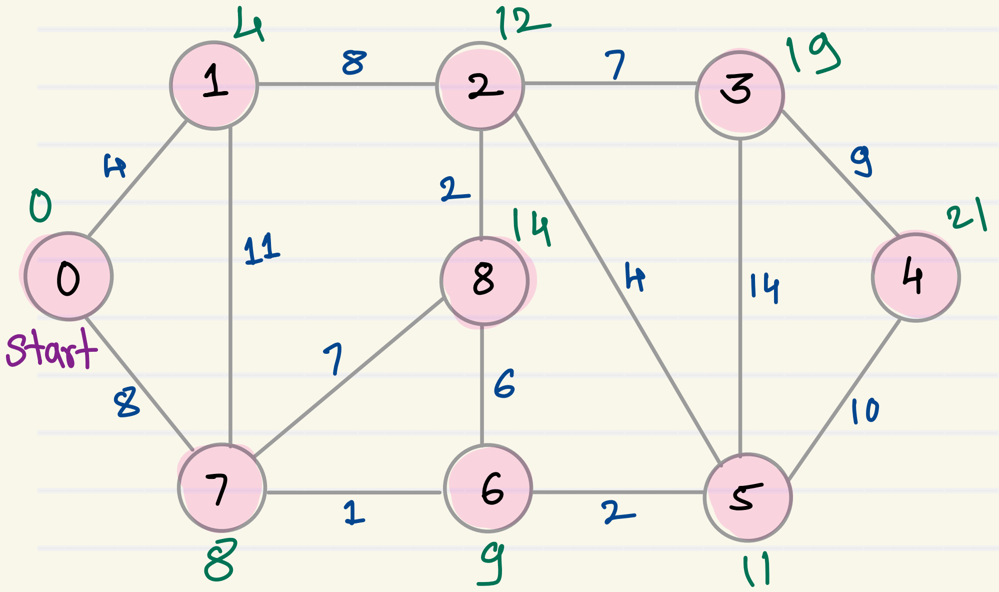



```
if( adjMat[u][v] < key[v] ){
    key[v] = adjMat[u][v];
    parent[v] = u;
}
```

Dijkstra's Algorithm

single source shortest path/dist algo

- **Initialisations**
  - Create a set spt to keep track of vertices included in shortest path tree.
  - Track distance of all vertices in the input graph. Distance of all vertices should be initialised to INF. The start vertex distance should be 0.

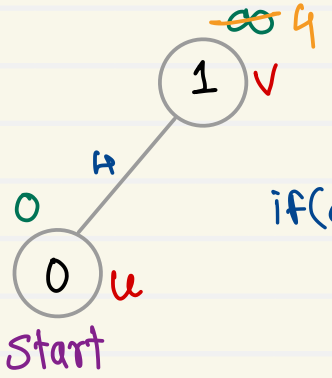


	Dist
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14

- **Algorithm**

  While spt doesn't include all vertices
  1. Pick a vertex u which is not there in spt and has minimum distance.
  2. Include vertex u in spt.
  3. Update distances of all adjacent vertices of u which are not there in spt

     For each adjacent vertex v, if distance of u + weight of edge u-v is less than the current distance of v, then update it's distance as distance of u + weight of edge u-v.




```
if(dist[u]+ adjMat[u][v] < dist[v])
    dist[v] = dist[u]+ adjMat[u][v];
```

Greedy Algorithm Design Technique

- A greedy algorithm is any algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.
- We can make choice that seems best at the moment and then solve the sub problems that arise later.
- The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the sub problem.
- It iteratively makes one greedy choice after another, reducing each given problem into a smaller one.
- A greedy algorithm never reconsiders it's choices.
- A greedy strategy may not always produce an optimal solution.

- Greedy algorithm decides minimum number of coins to give while making change.
- Available coins : 50, 20, 10, 5, 2, 1

amount = 36

36 - 20 = 16 20

16 - 10 = 6 20 10

6 - 5 = 1 20 10 5

1 - 1 = 0 20 10 5 1

Recursion

- Function calling itself is called as recursive function
- For each function call, stack frame is created on the stack. (FAR)
- Thus it needs more space as well as more time for execution.
- However recursive functions are easy to program.
- Typical divide and conquer problems are solved using recursion.
- For recursive functions two things are must
  1. Recursive call (explain process in terms of itself
  2. Terminating or base condition (where to stop)

Recursive problems :

1. Simple (loop alterative)
   e.g. factorial, power, factors
2. Divide & conquer
   e.g. binary search, tree algo, quick sort, merge sort.
3. Overlapping problem
   e.g. fibonacci

$$T_n = T_{n-1} + T_{n-2}$$

$$T_1 = T_2 = 1$$

Recursion - Fibonacci series (n^{th} term)

- **Recursive formula**

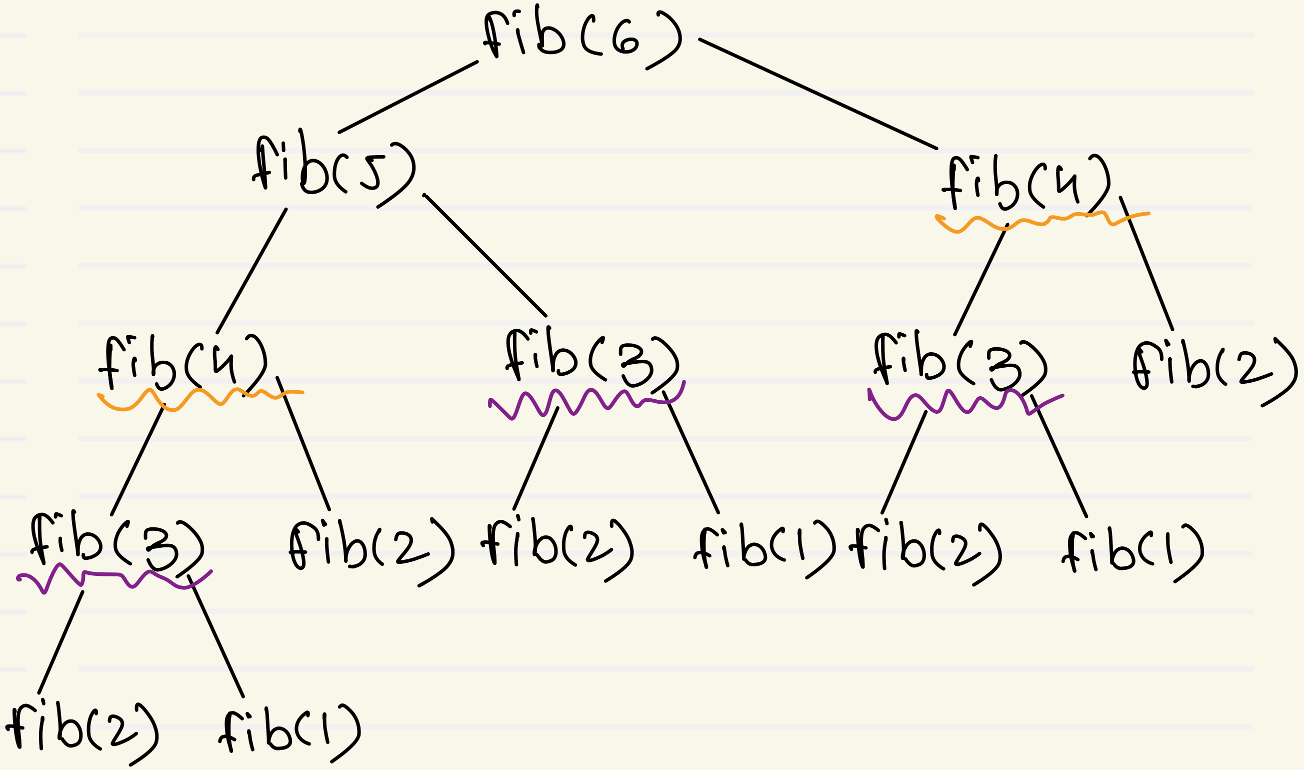
  $T_n = T_{n-1} + T_{n-2}$

- **Terminating condition**

  $T_1 = T_2 = 1$

- **Overlapping sub problem**

```
int fib(int n) {
    if(n==1 || n==2)
        return 1;
    return fib(n-1) + fib(n-2);
}
```



$T(n) = O(2^n)$

Memoization

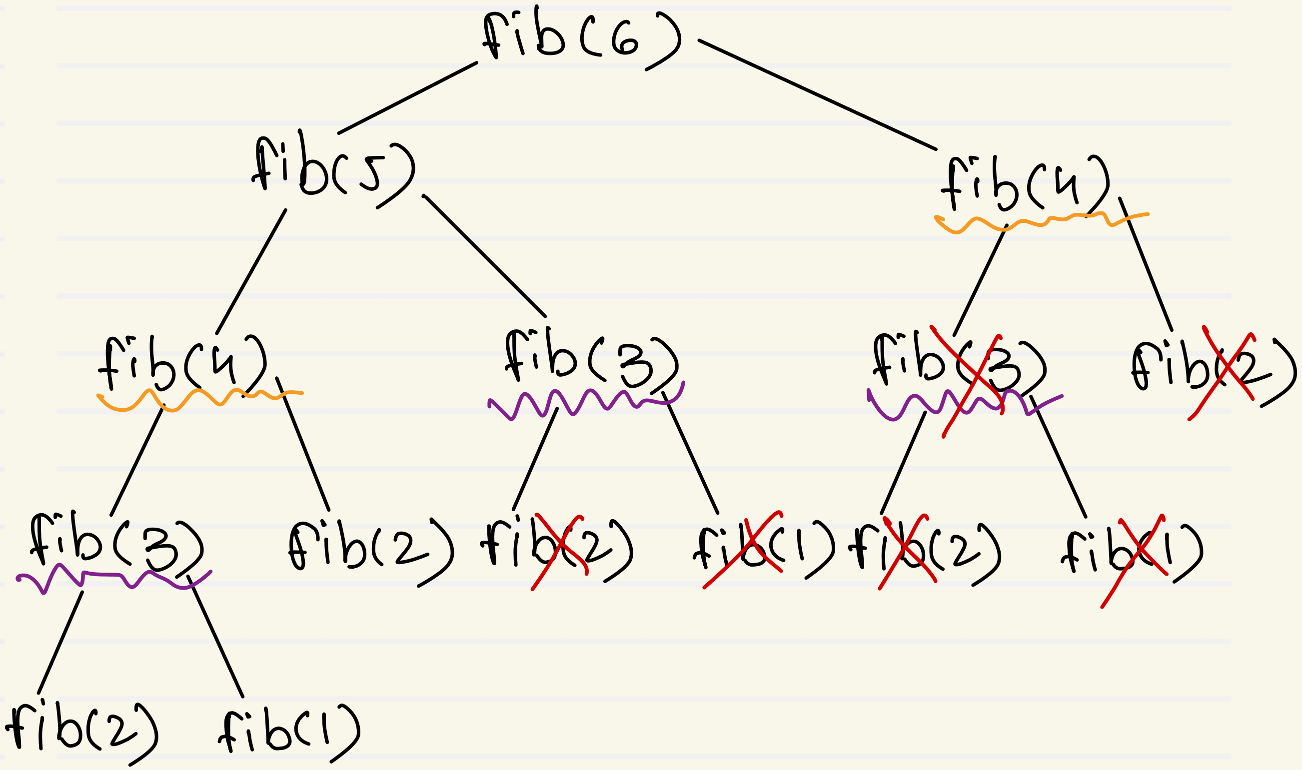
- It's based on the Latin word memorandum meaning "to be remembered"
- Memoization is a technique used in computing to speed up programs.
- This is accomplished by memorizing the calculation results of processed input such as the results of function calls.
- If the same input or a function call with the same parameters is used, the previously stored results can be used again and unnecessary calculation are avoided.
- Need to rewrite recursive algorithms. Using simple arrays or map/dictonary.
- Memoization uses top down approach.

Memoization - Fibonacci series

```
int terms[] = new int[n+1];

int fib(int n) {
    if( terms[n] != 0)
        return terms[n];

    if(n==1 || n==2) {
        terms[n] = 1;
        return 1;
    }
    terms[n] = fib(n-1) + fib(n-2);
    return terms[n];
}
```



Dynamic programming

- Dynamic programming is another optimization over recursion.

- Typical DP problem give choices (to select from) and ask for optimal result (maximum or minimum).

- Technically it can be used for the problems having two properties
  1. Overlapping sub problems
     - To solve problem, we need to solve it’s sub problems multiple times.
  2. Optimal sub structure
     - Optimal solution of problem can be obtained using optimal solutions of its sub problems.

- DP solution is bottom up approach.

- DP uses 1D or 2D array to save state.

- DP algorithms solve the sub problem in a iteration and improves upon it in subsequent iterations.

Dynamic programming - Fibonacci series

```
int dpFib(int n) {
    int terms[] = new int[n+1];
    terms[1] = terms[2] = 1;
    for(int i=3; i<=n; i++)
        terms[i] = terms[i-1] + term[i-2];
    return terms[i];
}
```

$n = 6$

						i
1	1	2	3	5	8	13
0	1	2	3	4	5	6