**Chapter 3: GREEDY ALGORITHM**

**Topic – 1: Overview**

* General characteristics of greedy algorithms
* Activity selection problem
* Elements of greedy strategy
* Minimum spanning trees
* Kruskal’s algorithm
* Prim’s algorithm
* Shortest paths – Dijkstra’s algorithm
* The knapsack problem

**Topic – 2: Characteristics Of Greedy Algorithms**

**Most Straightforward Design Technique**

* Takes **n** number of inputs & output is **subset** of the given group of inputs.
* Objective is to find the **most optimal solution**.

**Determines Feasible Solutions**

* Chooses the **most optimal solution** at each step, which we hope will lead to **globally** most optimal solution.

**Greedy v/s Dynamic Programming**

|  |  |
| --- | --- |
| **Greedy Algorithm** | **Dynamic Programming** |
| **Makes best local choice at a given step.** | **Used for searching all possibilities.** |
| **If evidence for reaching globally most optimal point is less, it might fail.** | **Will guaranteed work as it sees all possibilities.** |
| **Doesn’t store any computing data.** | **Stores computing data to not recompute again.** |
| **Choice is made before solving sub-problem.** | **Sub-problems are solved first.** |
| **Top-down approach.** | **Bottom-up approach.** |
| **Fast** | **Slow** |
| **Straightforward** | **Complex** |

**General Greedy Algorithm**

* **Step 1:** We start moving from the **initial node** (value).
* **Step 2:** We **discover** many possible choices (nodes) we have at each level to move.
* **Step 3:** We choose the **most optimal choice** (node), or the choice which will lead us to our desired solution most optimally.
* **Step 4:** Then we move to the **chosen node**.

**Example**

**Given: We live in a country with currency having coins of $1, $0.25, $0.10, $0.01.**

**Problem: Find a way to pay $2.89 with minimum number of coins.**

**Solution: $2.89 = ($1\*2) + ($0.25\*3) + ($0.10\*1) + ($0.01\*4)**

**Note!**

**🡪 We were finding the most optimal solution in the example above.**

**Topic – 3: Activity Selection Problem**

**Introduction**

* This problem refers to problem where we have to **organize** activities/tasks at such a time that the **shared resources** **can’t** be used **concurrently**.

**Example**

* Let’s say that various activities have to be done inside a **common hall**, such that one activity’s time **doesn’t** clash with any other activity’s time.
* Meaning no two activities are performed at the **same time**, though the hall shared is common among them.

**Recursive Activity Selector**

* So, if we are given data about various activities & find out that some of the **timings clash**, then we pick up only the **compatible activities** & **discard** the others.
* Our goal is to let **most activities** **occur** in the hall.
* Sorting these data can help us achieve this goal.

**Elements Of Greedy Strategy**

* Works on **recursion** when making choice.
* We have to prove that the **optimal choice** we are making at any level is the ***greedy choice***.
* **Greedy choice:** The **safe** choice which leads to desired solution.
* Show that one of the all of sub-problems created when making greedy choice is **empty**.

**Greedy Choice Property**

* Actually, we **can’t** know if a choice will lead us to solution or **not**.
* But the points discussed above if satisfied, then we may likely develop a **greedy algorithm**.

**Topic – 4: Minimum Spanning Tree**

**Introduction**

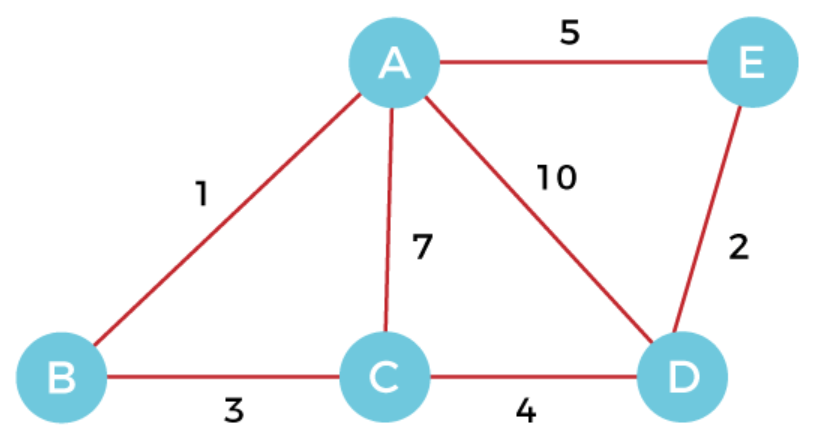
* **Max edges** possible among a given number of vertices is **n(n-1)/2**.
* **Spanning tree:** A tree with **(n-1)** edges where **n** is **number of vertices**.
* There can be **weight/cost** on the edges.

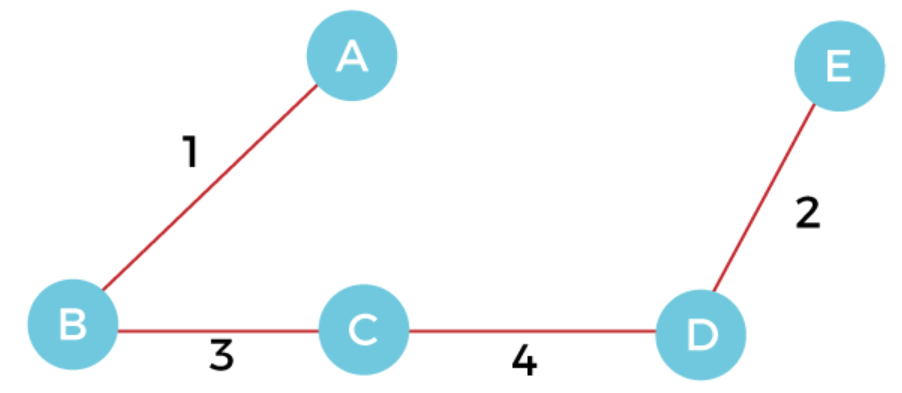
**Greedy Minimum Spanning Tree**

* The edge to be chosen **next** is as per some **optimal criteria**.
* It is usually the edge which had **minimum weight** from the available options.

**Kruskal’s Algorithm**

* **Lowest to highest** weighted edges are chosen one-by-one, **without** creating a cycle.
* We repeat it until **all** the vertices are done.





**Prim’s Algorithm**

* **Step 1:** A **random** **vertex** is chosen.
* **Step 2:** Then **most optimal** **edge** is chosen from that vertex.
* **Step 3:** Then we chose the **most optimal edge** from the vertices we have included so far.
* **Step 4:** We repeat these **without** forming any cycle.

**Dijkstra’s Algorithm**

* **Step 1:** We choose a **vertex/node** as per our choice.
* **Step 2:** Then we find **shortest path** from that node to other nodes & track them in tables.
* **Step 3:** When we find a shorter path to already visited node, we **update** the distance in table.

***\*This all works on basic recursion model\****

**Topic – 5: Knapsack Problem**

* Remember the ***bag & weight problem***.
* Each block has a **value** & **weight**.
* We have to gain **max value** without reaching the **max weight**.

**Topic – 6: Huffman Coding**

**Introduction**

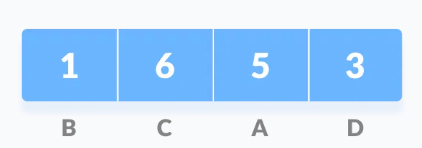
* It’s a **data compression technique** without losing any details.
* Data are treated as **strings** when being compressed.

**Example**

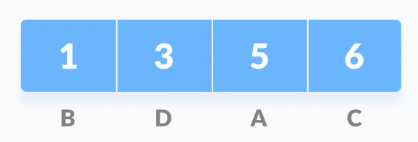
**Step 1: Write the string.**

****

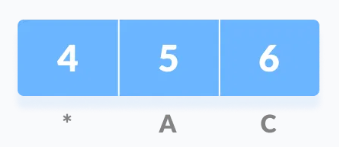
**Step 2: Make an array with frequencies of each character’s appearance.**

****

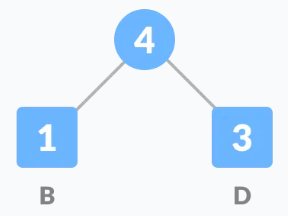
**Step 3: Put them in ascending order.**

****

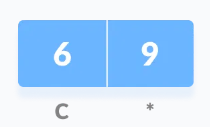
**Step 4: Merge the first two nodes, adding their frequencies. Arrange again if required.**

****

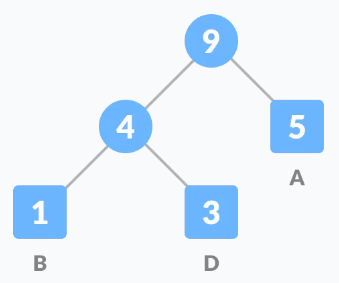
**Step 5: Make a tree with merged value as parent node, smaller one of the two merged nodes being at its left & larger one at its right.**

****

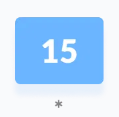
**Step 5 (repeat): Again, merge the next two nodes & rearrange if required.**

****

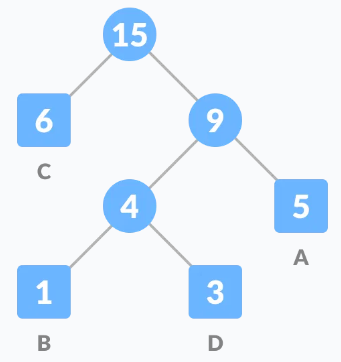
**Step 6 (repeat): Modify the tree with same rules.**

****

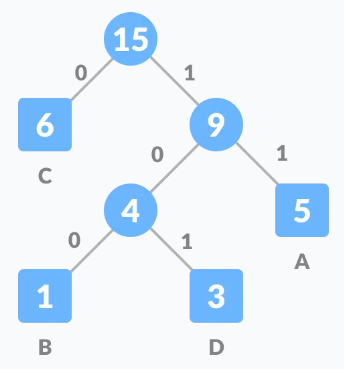
**Step 5 (repeat-2): Merge & rearrange if required.**

****

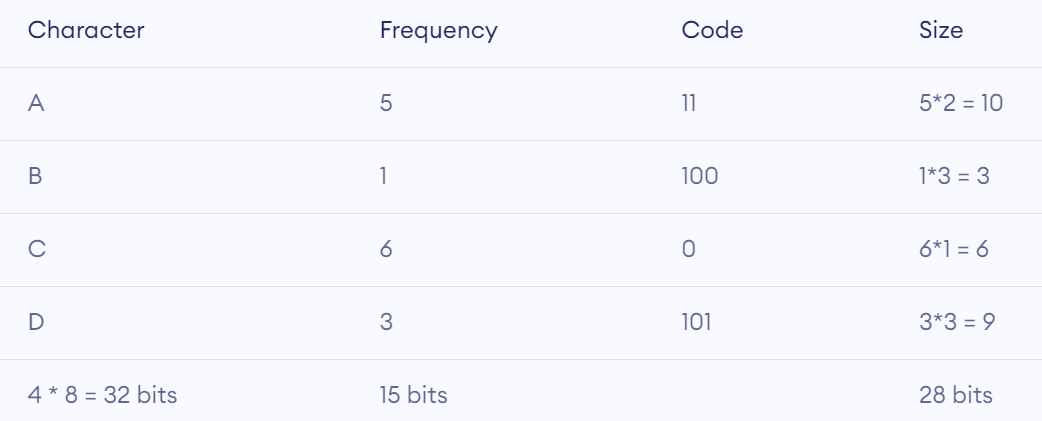
**Step 6 (repeat-2): Modify the tree, again.**

****

**Step 7: Assign all left edges as 0 & right edges as 1.**

****

**Step 8: Set up the calculation table.**



**T(n) = O(nlog(n))**

**Applications**

* File compression formats.
* Text & fax transmissions.