**Department of Computer Engineering**



**Cairo University**

**Faculty of Engineering**

**Advanced Database Project**

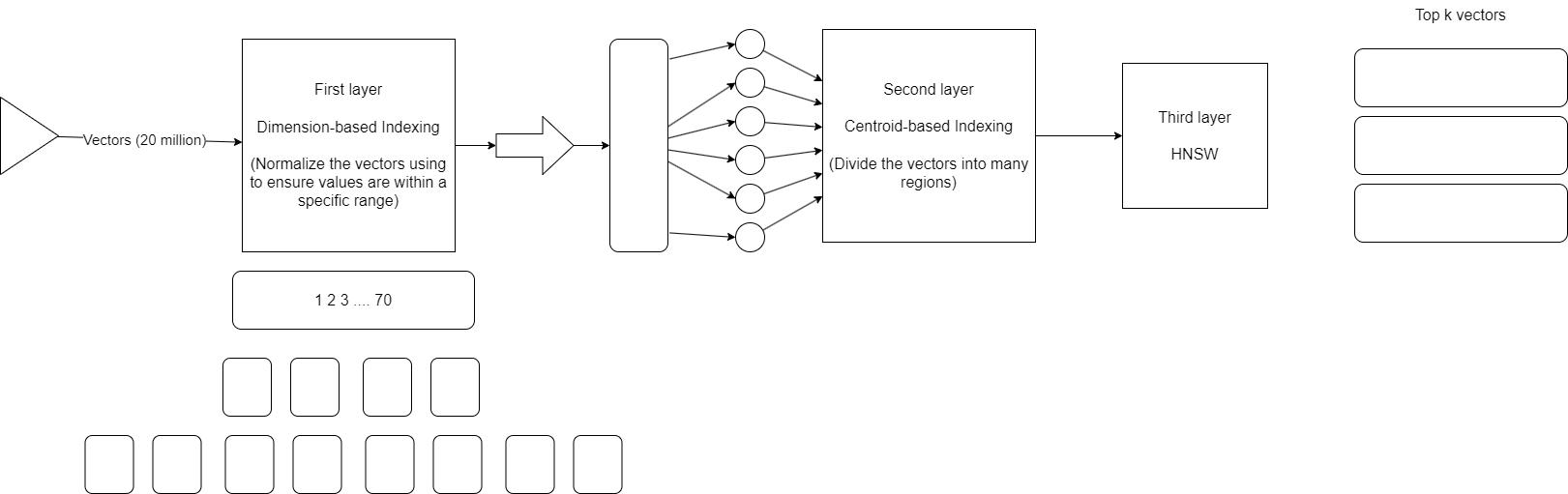
Semantic Search Engine

***Team 10***

|  |  |
| --- | --- |
| Name | ID |
| Eslam Ashraf | 9202256 |
| Beshoy Morad Atya | 9202405 |
| Abdelrahman Hamdy | 9202833 |
| Abdelrahman Noaman | 9202851 |

Date: 11/15/2023 Presented to: Eng. Abdelrahman Kaseb

**Design Image:**



**Layer 1:**

**Multi-dimensional Space Partitioning Algorithm**

Input: A vector V that consists of 70 Dimensions

Output: Name of the file that will store that vector

Description:

* Our goal from this layer is to categorize each vector entering it into one of the set of spaces. These spaces are represented by some comparisons so that we narrow the variety of vectors in each space as much as possible. Once we’ve placed each vector in a specific criteria, it’ll be easier to pick one of them and work on the vectors contained inside which will be approximately ~10k vectors for example rather than searching through the whole dataset ~20m
* The way vectors are partitioned into different spaces is by comparing each element with a normalized number between 0 and 1 (in this case it’s 0.5) and repeating it with all combinations of the various comparisons that can hold for all spaces.

Example, let’s say we have a vector containing these elements

V1 = [0.6, 0.7, 0.75, 0.3]

V2 = [0.8, 0.54, 0.77, 0.6]  
First space: [(>0.5), (>0.5), (>0.5), (>0.5)]: V2

Second space: [(>0.5), (>0.5), (>0.5), (<0.5)]: V1

And so on…

* Data structure: A variable size Multi-Dimensional Array

**Layer 2:**

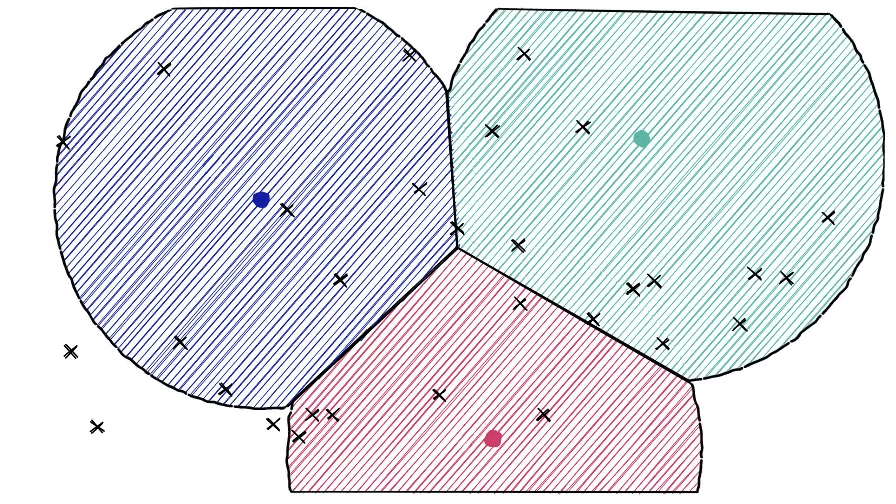
**Centroid-based partitioning**

Input: Single space file from the previous layer containing ~10k vectors (for example)

Output: Index file that have only the centroids of that space ~30 centroid (for example)

Description:

* From each input file, we’ll have to form centroids to ease the search process. These centroids help us divide the amount of vectors into regions of certain similarity so that we land on a region from its centroid. This narrows the number of vectors we’ll search from even more.
* Centroid-based partitioning: We can use any algorithm that divide the space into centers like (IVF or IVF-PQ).
* The Inverted File Index (IVF) index consists of search scope reduction through clustering. It works on the concept of Voronoi diagrams as shown in the picture. Our goal is to place few additional points (The ones colored) which will become our ‘cluster’ centroids. We then extend an equal radius out from each of our centroids so that the circumferences of each cell circle collide with one another, creating our cell edges. Surely we’ll need to handle the popular case of having a point placed beside an edge and figure out how to gather points around it even if they aren’t in its particular cluster.
* Every data point will be contained within a cell and be assigned to that respective centroid. The query vector must land within one of our cells so that we restrict our search scope to that single cell.
* Data structure: K-Dimensional Tree or Hierarchical Tree like the B+ Tree



**Layer 3:**

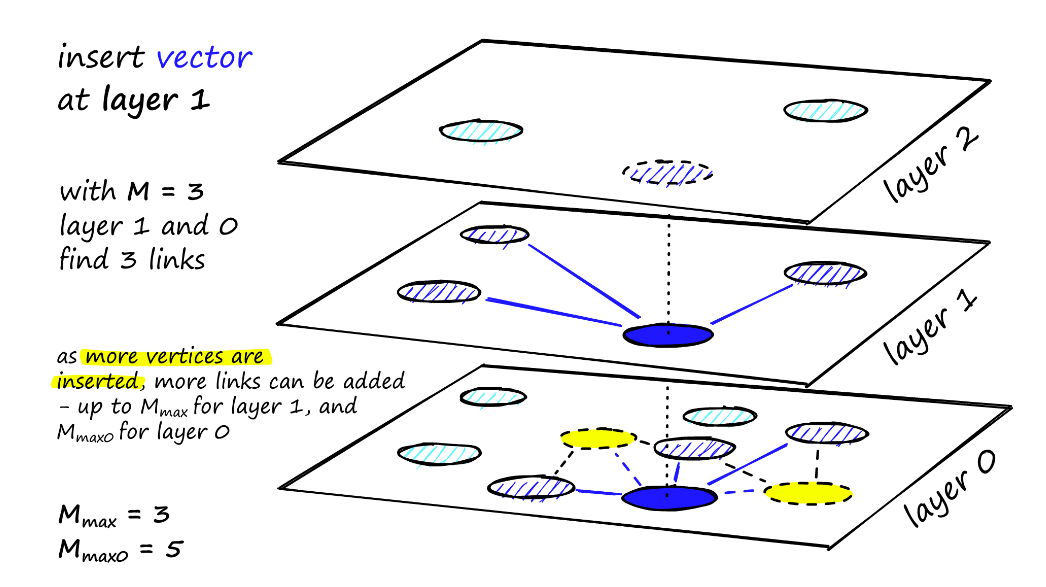
**Hierarchical Navigable Small World (HNSW) Index**

Input: Minimum number of vectors which will contain the nearest neighbors

Output: The top K similar vectors to our query

Description:

* This is the final step for a reason. HNSW is not the best index in terms of memory utilization so we made sure to use it with the minimum amount of vectors to locate our query. It’s known for the super-fast search speeds in high-dimensional spaces by traversing through layers of linked nodes (forming a graph) using the NSW greedy approach and the optimization of the Probability Skip List to approximate the nearest neighbors.
* HNSW creates a multi-layered graph where each layer represents a subset of the data and we can quickly traverse these layers with simple comparisons and distance calculations (Vectors in each layer are sorted according to the distance function).
* During graph construction, vectors are iteratively inserted one-by-one starting from the top layer.



* During querying, start at the root node and navigate the graph to find the k most similar vectors.
* Data structure: A Navigable Graph

