Université d'Ottawa
Faculté de génie

École de science
d'informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical
Engineering
and Computer Science

Nearest neighbor algorithms
Programming assignment
Part 2 (10%)
Approximate Nearest Neighbour
CSI2110 Algorithms and Data Structures

Fall 2023

This is an individual assignment. You can obviously discuss with your colleagues but do not show your code, do not share your code, do not publish your code, do not copy code. Doing so may lead to accusation of fraud.

Late assignment policy: ***1min-24hs late are accepted with 30% off; no assignments accepted after 24hs late.***

Problem description

In the first part of this project, we wrote an algorithm to find the K nearest neighbors of query points. We used a strategy to accelerate the search for these K nearest neighbors, but we also realized that this search could be very time consuming if the dataset to explore is very large. However, in several applications it is not necessary to obtain the exact nearest neighbors, an approximate solution could suffice. For example, in an image search or a music recommendation app, it is more important to provide quick suggestions rather than finding the absolute most similar data.

Consequently, in the second part of this project we will study an Approximate Nearest Neighbor (ANN) search algorithm. In this case, we want to find vectors in the dataset that are among the nearest neighbors without necessarily being the closest one. The concern here is to obtain these vectors as quickly as possible, which means that we will not scan over the entire dataset during the search.

The trick is to insert the vectors of the dataset in a data structure that will facilitate the search. Remember that we have a large number of vectors (the dataset) in a high-dimensional space. We also mentioned, in the description of the part 1 of this project that K-D trees is an example of such data structure, however the search in K-D trees is not efficient when the search space is of high dimension since its complexity is O(DlogN). An alternative could be to use a method called Locally Sensitive Hashing (LSH), an extension of the regular hashing seen in class. In this project, we will consider an algorithm that is using what is considered to be the current state-of-the-art strategy to perform ANN search: Graph Traversal.

Graph-based ANN search

As the name suggest, Graph-based ANN (GANN) proceeds by building a graph from the vectors in the dataset in which connected nodes are vectors relatively close to each other. When a query point is received, GANN will traverse the graph until it finds a good candidate nearest neighbor. This approach is composed of two steps:

1. Offline step: building the graph. This is a pre-processing step that is executed only once using all the vectors in the dataset. The execution time to complete this step is not too important since this one is executed once before users start to query the database.
2. Online step: graph traversal. This is the step that finds an approximate nearest neighbors from a query point. It is crucial to execute this task very efficiently to make sure the approximate nearest neighbour is obtained quickly.

Let's start by describing the online search as this is the main step. We therefore assume that a graph has been constructed from the vectors of the dataset.

1. <u>ANN search from graph traversal</u>

We have a connected graph in which each vertex is a vector in the dataset. Since our vectors are associated to vertices, when we mention the distance between two vertices, we refer to the distance between the two corresponding vectors of the dataset.

Suppose we have a query vector Q, not in the database, and we want to find the K nearest neighbors. We have a graph in which each vertex V corresponds to a vector in the dataset. Define an empty array A of capacity S in which the vertices will be sorted by their distance to Q. The GANN search proceeds as follows:

1. Randomly select a vertex W in the graph.
2. Compute the distance between W and Q.
3. Insert W into the sorted array A.
4. Get the unchecked vertex C in A that has the smallest distance to Q
5. Mark vertex C as checked
6. For each vertex V adjacent to C
   a. If the distance between V and Q has not been computed then
      i. Compute the distance between V and Q
      ii. Insert V into A
7. Repeat 4. until there is no more unchecked vertices in A
8. The first K vertices in A are the approximate K nearest neighbors.

Notes:

- See Appendix A for an illustration of the algorithm.
- The array A contains the S nearest vectors found so far.
- The element at index 0 is the nearest vector.

- If at step 6.a.ii the vertex V is farther than the last element in the array A, then it is not inserted. Otherwise the vertex V is inserted at its position. The last element is therefore removed to maintain the capacity at S.
- To simplify the test 6.a, it could be a good idea to associate an integer key to each vertex and then assign it the ID number of the current query point when the distance is computed (the distance being the floating point key). This way you can simply check if the current vertex has its integer key equals to the query point ID, if yes, that means that this distance has been already computed.
- When a vertex is removed from A, it could be a good idea to uncheck it such that it will be ready for a new search with a new query vector.
- Even if, in theory, one might set S=K, it is a good idea to set the capacity of the array at a value S>K, this way a larger portion of the graph will be explored.

2. Building a graph for ANN search

As we have explained in the previous section, a graph is used to navigate inside the dataset in order to find the nearest neighbors of a query vector. The hope is that by traversing the graph from a random vertex to iteratively closer vertices, the nearest vertex will eventually be reached. The speed at which the nearest neighbors can be identified completely depend on the structure of the graph and its connectivity. Graphs with high connectivity can generally converge in fewer iterations, but each iteration requires more computations because of the large number of adjacent vertices. For instance, a fully-connected graph would correspond to an exhaustive search. In addition, highly-connected graph are often more costly to build and occupy more memory. On the other hand, a sparse graph will require more traversal steps. In addition, the search may get stuck in local minima or worst, the graph might be not connected (one way to cope with disconnected graph is to repeat the search few times, starting from different random vertices). You will therefore find, in the scientific literature, several algorithms to build better graphs for ANN search. In our case, we will choose the simplest approach, the k-NN graph, which does not provide the optimal performances, but still can produce very efficient search results.

The definition of a k-NN graph is very simple: a vertex Vi is connected to a vertex Vj if Vj is one of the k nearest neighbor of Vj.

All you need is therefore the list of k nearest neighbors for all vectors in the dataset. This is very expensive since there are many vectors in the dataset but as we explain, this is an offline step done only once.

From the list of k nearest neighbors you can create the graph using adjacency lists. We give you here a very simple implementation that uses `java.util.Map`

```
class UndirectedGraph<T> {
    private Map<T, LinkedList<T>> adjacencyList;

    // Constructs an empty graph
    public UndirectedGraph() {
        adjacencyList = new HashMap<>();
    }

    // adds a vertex to the graph with an empty adjacency list
    public void addVertex(T vertex) {
        if (!adjacencyList.containsKey(vertex)) {
            adjacencyList.put(vertex, new LinkedList<>());
        }
    }

    // adds an edge between vertex1 and vertex 2.
    // the two adjacency lists are updated
    public void addEdge(T vertex1, T vertex2) {
        addVertex(vertex1);
        addVertex(vertex2);

        adjacencyList.get(vertex1).add(vertex2);
        adjacencyList.get(vertex2).add(vertex1);
    }

    // gets the list of all adjacency vertices of a vertex
    public List<T> getNeighbors(T vertex) {
        return adjacencyList.
           getOrDefault(vertex, new LinkedList<>());
    }
}
```

Your task

Your mission for Programming Assignment P1 is to test the speed and the accuracy of the graph-based ANN search. In this project we will only test the 1NN but, as explained, the extension to the kNN is straightforward.

The data

We give you the same four files than in part 1. Two files containing point datasets and two files containing query points for your 1NN searches. One of the main drawbacks of graph-based algorithms is that they use a large amount of memory, therefore, in this programming assignment, you will mainly work with the small files

You have the file `PointSet.java`, containing an utility class that reads a point set from these files.

The programming task

- We ask you to create the class `GraphA1NN` that finds the 1NN of a query point. You have to define this class with the following methods:
  - An instance of this class is constructed by providing either
    - a reference to a `PointSet` object;
    - or the name of a `fvecs` file;
  - A setter method `setS` that sets the capacity of the array used to set the value of S, the capacity of array A;
  - A method `constructKNNGraph(int K)` that constructs a k-NN graph from the `PointSet`. To simplify your work, the k-NNs do not have to be searched, they are provided in a file called `knn.txt` (hard-coded);
  - A method `find1NN(LabelledPoint pt)` that finds the nearest neighbor of a query point and return it as an instance of `LabelledPoint`.

You can also create other methods or classes.

How to measure the speed of the search?

You simply measure the speed of the call to the `find1NN` method and you average this timing over all the tested query points.

How to measure the accuracy of the result?

An ANN search is considered successful if it is among the 10 nearest neighbors of the query vector. A file is provided to you containing the 10 nearest neighbors of all the query vectors used for testing.

Optionally, you can also compute the average position of the approximate nearest neighbor when the search is successful (i.e. position between 1 and 10).

The experiments

Once you have defined and tested your `GraphA1NN` class, you now have to perform the following experiments:

- Using the dataset of 10,000 points, we ask you to measure the execution for finding nearest neighbors of 100 query points;
  - Use the files provided matching this description.
  - You perform this experiment for k= 10, 25, 100 and for S=25, 100
  - You can also test other values
- Using the dataset of 1,000,000 points, we ask you to measure the execution for finding nearest neighbors of 100 query points;
  - Use the files provided matching this description.
  - You perform this experiment for k= 25 and for S=100
  - You can also test other values
- Report your speed and accuracy results:
  - on a graph showing k-value on the x-axis and execution time in ms on the y-axis. Create two such graphs, one for each value of S.
  - on a graph showing k-value on the x-axis and accuracy in % on the y-axis. Create two such graphs, one for each value of S.
- Comment on the results obtained.

Submission instructions

For this first programming assignment, you have to submit the following items:

- **All your Java classes in a zip file**, in particular
  - the `GraphA1NN` class;
  - to run your program, the following arguments should be provided to the main program (via: `public static void main(String[] args)`)
    - value of k
    - value of s
    - dataset filename
    - query point filename

    for example, the command line for running this class would look like:
    ```
    java GraphA1NN 25 10 sift_base.fvecs siftsmall_query.fvecs
    ```

Marking scheme

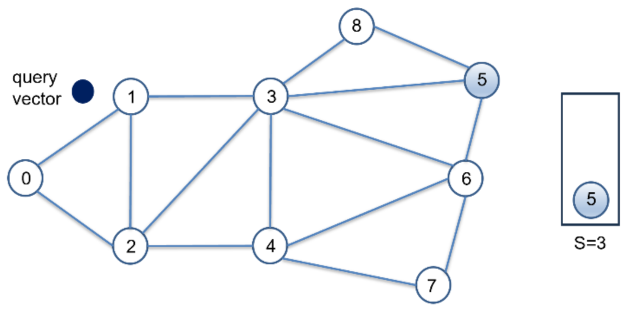
`GraphA1NN` class	15%
Other classes	10%
k-NN Graph construction	10%
ANN search method	15%
Program efficiency (time and memory)	10%
Quality of documentation (report, comments and headers)	10%
Computational time and accuracy experiments with the 10,000 vector dataset	10%
Computational time and accuracy experiments with the 1,000,000 vector dataset	5%
Discussion and comments on results	15%

- *All your files must include a header that includes your name and student number. All the files must be submitted in a single zip file.*
- *Also add a readme file that includes instructions on how to run your program. Make sure to include all the files required to run your program.*
- *Do not include the .class files*
- *Limit the size of your .zip file*

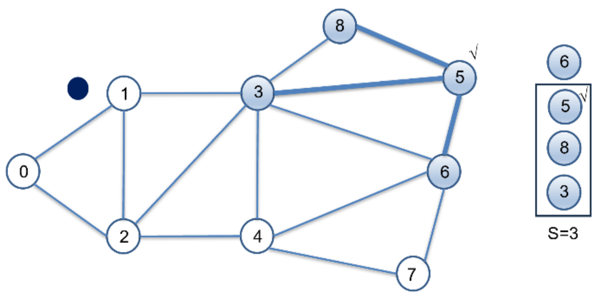
Appendix A: Graph-based ANN search algorithm

A blue vertex is one for which the distance to the query point has been computed.
Vertex with a √ are vertex marked as checked.

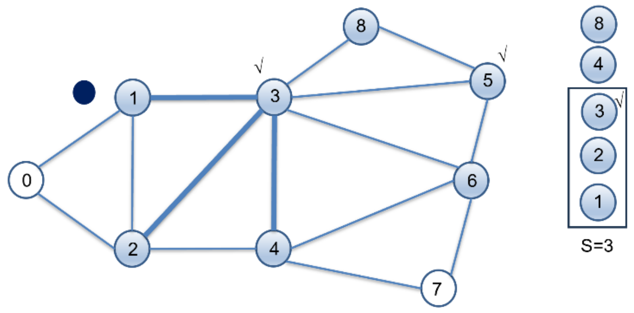
Step 1:



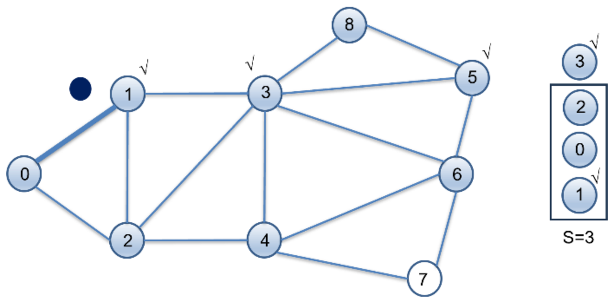
Step 2:



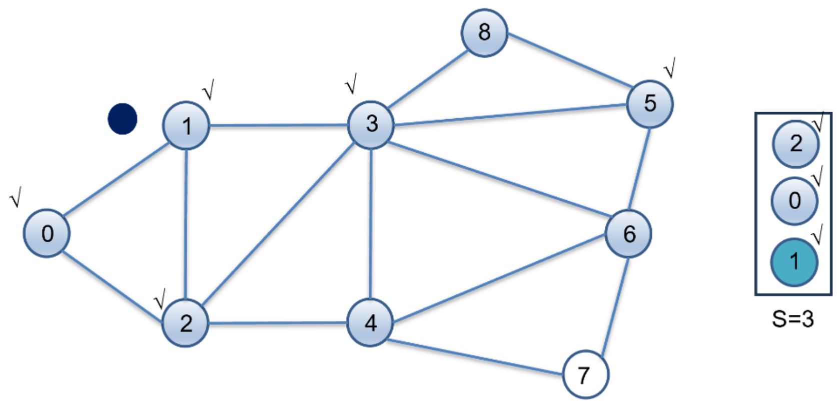
Step 3:



Step 4:



Step 5:


0
1
2
3
4
5
6
7
8
2
0
1
S=3