CS2710 - Programming and Data Structures Lab

Lab 12 (graded)

Oct 23, 2024

Instructions

- For each question, first think and write up the pseudocode of your solution in the answer sheet provided, then code it up, test it and upload tested code to moodle. **Remember to submit the answer sheet containing the pseudocode** to your TA, as it will also be graded, besides the uploaded code.
- You are expected to solve ALL the problems in the lab, using the local computer, C++ language, and g++ compiler. (Bonus problems can fetch 5% capped bonus, and Optional problems are just for fun and won't be graded.)
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm (so that TAs can subsequently grade your submissions for graded lab assignments using the private test cases).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Session 12 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB12_Q1.cpp** and **CS23B000_LAB12_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB12**
  - Zip this directory into a file named **CS23B000_LAB12.zip**
  - Submit only this single .zip file to moodle.
- If you need assistance, ask your TA, not your classmate.
- Internet access and mobile devices are prohibited in the lab.
- Check course Moodle for the public test cases and the evaluation script, which you can use to test your programs.
- **Hash table, as implemented in std::unordered_set or std::unordered_map, is the primary data structure you would use** for solving the questions in this assignment (i.e., no need to implement your own hash table, and you can use the C++ implementations). You can use the standard string or vector data structures provided by C++ for book-keeping purposes. Please clarify with the TAs if you have any questions about which data structures to use or avoid.

1. [Hash Table Warmup: Finding Sum-consistent Triplets] Given an array A of distinct integers, find all triplets of distinct indices (i, j, k) such that $A[i] + A[j] = A[k]$. Note that $i < j$ and $i \neq j \neq k$.
Expected Time Complexity: $O(n^2)$ average-case running time, where n is the number of elements in the input array.
Expected DS: While there may be many different ways/logic to solve this problem, we would like you to use hash table to solve this question.

Input Format:

- The first line contains an integer n, representing the number of elements in the array.
- The second line contains n distinct integers, separated by space.

Output Format:

- For each valid triplet, output the pairs in the format: (i, j, k), where $i < j$.
- If multiple triplets are found, list them in ascending order, first sorting by index i, then by j to break ties in i.
- If no such triplets exist, output: `No triplets found`.

Constraints:

- Number of elements, $n \leq 1000$.
- $-10^4 \leq$ Value of elements $\leq 10^4$.

Examples:

Input1:
7
3 4 7 11 2 9 8

Output1:
(0, 1, 2)
(0, 6, 3)
(1, 2, 3)
(2, 4, 5)
(4, 5, 3)

Explanation: At these respective index triplets, we've:
3 + 4 = 7
3 + 8 = 11
4 + 7 = 11, etc.

Input2:
6
65 30 7 90 9 8

Output2:
No triplets found

2. [MARKOV-CHAIN-BASED READABLE NONSENSE GENERATOR - A RUDIMENTARY “CHATGPT”?]
You are tasked with implementing a “Markov Chain” based word generator that uses **bigrams** (pairs of words) to **predict the next word** in a sequence. The goal is to to generate readable but nonsensical text based on an input reference text.

The generator works by *recording all possible third words* that follow each bigram (pair of consecutive words) in the input text in a data structure DS. Once this DS is built, the generator will output a random text as follows:

```
srand(1234); //set random seed (requires: #include <cstdlib>)
bigram (w1,w2) = first bigram in the input text
output w1, w2
repeat for n times {
  predict the next word w3 for the current bigram
    (by picking at random one of the possible third words for the current bigram in the DS)
    //implement using: thirdWordsForCurBigram[rand() % thirdWordsForCurBigram.size()]
    //break loop if (thirdWordsForCurBigram.size()==0)
  output w3
  update current bigram: (w1, w2) to (w2, w3)
}
```
//Note: For simplicity (and to generate readable text), assume that the words are single-space-separated (so words can contain punctuation marks and capitalized letters). Also assume that thirdWordsForCurBigram is a `std::vector<string>` containing all possible third words following the current bigram in the input text in the same order they appear in the input text (and allowing duplicates).

Input Format:

- First line contains a string of text that will be used as the reference for generating text.
- Second line contains an integer `n`, the maximum number of words to generate in the output text.

Output Format:

- Text generated by the above pseudocode.
- Closely follow the random seed, instructions, etc., in the above pseudocode to generate random text that will pass the test cases.

Constraints:

- $\texttt{n} \leq 1000$.
- Number of words $\leq 10^5$.

Example:
Input:
Thou shalt not kill. Thou shalt not commit adultery. Thou shalt not steal. Thou shalt not bear false witness against thy neighbour. Thou shalt not covet thy neighbours house. Thou shalt not covet thy neighbours wife. Thou shalt not covet his manservant. Thou shalt not covet his maidservant. Thou shalt not his ox. Thou shalt not covet his ass.
43

Output:
Thou shalt not covet thy neighbours house. Thou shalt not steal. Thou shalt not covet his manservant. Thou shalt not covet thy neighbours house. Thou shalt not kill. Thou shalt not covet his manservant. Thou shalt not covet thy neighbours wife. Thou shalt

Explanation:

- The process starts with the first bigram in the input text, **”Thou shalt”**, as the current bigram. The possible next word after this bigram is chosen at random from the third words vector: [”not”, ”not”, ”not”, ”not”, ”not”, ”not”]. So the first three words generated are: **”Thou shalt not”**.
- The updated current bigram is ”shalt not”. The possible next words after ”shalt not” are [”kill.”, ”commit”, ”steal”, ”bear”, ”covet”, ”covet”]. The word selected at random could be ”bear”, so the next word generated is ”bear”, making the sentence: **”Thou shalt not bear”**.
- This process continues until `n` words are generated or we run out of no third words for the current bigram.