CS-3364 Design and Analysis of Algorithms

Project II

Problem description

Polynizer has hired the excellent students of CS-3364 to create an algorithm that allows to find the optimal way to play the chords that the application infers, on the guitar. Specifically, we want to minimize the amount of movement made by the left hand. The algorithm takes as input the list of chords of a song in standard notation (e.g., C, Am, Dm, G7, and C) and a file specifying different ways to play each chord, and should print on the screen how to play each chord in order to minimize the total movement of the hand over the song.

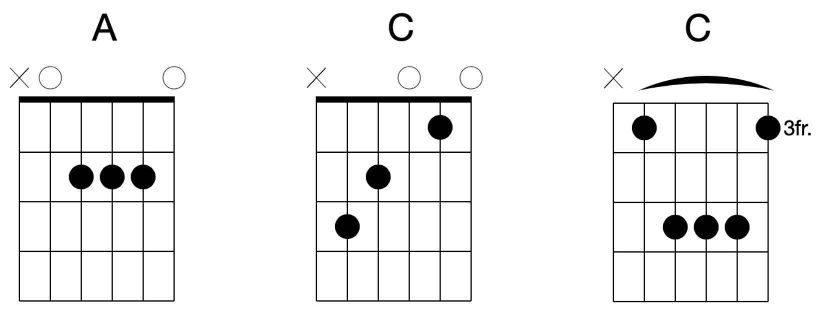
Guitar chords specification

The chord specification file consists of multiple lines. Each line begins with the name of a chord, and is followed by the fret numbers, separated by commas, on which the fingers of the left hand should press each string.[1] To form the position of some chords, some frets are left *free*; in such case we write zeros. It is also common that some strings are not plucked (with the right hand); in that case we write nothing. Figure 1 shows how to play the chords A major (A) and C major (C). In the former, the first string (rightmost string) and the fifth string are played "open", that is, without pressing any fret, while the second, third, and fourth strings are played on the second fret; the sixth string is left off. This is represented as a sequence of numbers, starting with the sixth string and ending with the first string, and preceded by the chord name: "`A,,0,2,2,2,0`" (there is nothing between the first two commas because the sixth string is not used in this chord).

Shifting diagrams

Some chord positions use higher order frets (5th, 6th, 7th, etc.). In such case, we move the diagram up and indicate what fret of the guitar corresponds to the first fret of the diagram (the third fret, in the third example in figure 1). To facilitate diagramming, after the

[1] Entries are separated with commas to facilitate handling of the chord definition file with spreadsheets (e.g. Excel, Numbers and Calc), using the CSV format (comma separated values).



(a) A chord (b) C chord (c) C chord (alternative form)

Figure 1: Examples of how to play the A and C chords (with the left hand) on guitar. The vertical lines represent the strings (the rightmost string is called *first* string and the leftmost *sixth* string). The upper horizontal line (thick one) represents the *nut* of the guitar and the lower ones the frets. (The upper fret is called *first* fret, and the following, *second*, *third*, etc.). The filled circles indicate where the strings should be pressed. Strings with a non-filled circle (at the top of the diagram) indicate that they should not be pressed. Strings with a '$\times$' symbol should not be plucked (with the right hand, which is in charge of setting the strings in motion to produce sound). A string is said to be *played on the first fret* if it is pressed just before the first fret, *on the second fret* if pressed just before the second fret, etc. When a chord requires that only higher order frets be used (5th, 6th, 7th, etc.), it is customary to move the diagram as far up as possible and indicate to what fret of the guitar the first fret of the diagram corresponds to. For example, (c) shows a different way of playing the C chord. The label "3fr." indicates that the first fret of the diagram corresponds to the third fret of the guitar. This is done to reduce the size of the diagram. (Thanks to this most chords can be diagrammed using only four frets.)

chord name we insert a number that specifies what fret of the guitar corresponds to the first fret of the diagram. However, if the first fret of the *diagram* does indeed correspond to the first fret of the guitar, we omit that number. For example, the representation of the chords in Figure 1 would be “`A,,,0,2,2,2,0`” for A, “`C,,,3,2,0,1,0`” for C in its first form, and “`C,3,,1,3,3,3,1`” for C in its second form.

Although most guitars have way more than 12 frets (some having up to 24), they are used mostly to play melodies, no chords. **For this project we will consider only positions whose lowest fret does not exceed the seventh**. For example, you should ignore positions like “`Ab,8,,4,3,1,2,1`” (starts on the 8th fret) or “`B5/F#,9,,1,1,3,4,`” (starts on the 9th fret).

Compacting the different ways of playing a chord

To simplify the search of the different forms of a chord, they are compacted into a single line and the chord name is specified once at the beginning of the line. In other words, each line starts with the name of the chord followed by groups of seven numbers: the offset, which can be empty, and the frets where each of the six string should be played, which can also be empty if the string is not to be played. For example, the two aforementioned variations of C major are represented as follows: “`C,,,3,2,0,1,0,`**`3,,1,3,3,3,1`**”. (We write in bold the numbers of the second form to facilitate appreciating where the first one ends and the second begins).

Sections with no chords

When a section of a piece is meant to be played without chords, the symbol N.C. (*no chord*) is used. To simplify the algorithm, the N.C. symbol will be ignored, and the preceding and next chords will be considered following each other (i.e. the displacement between them counts for the total).

Measurement of displacement

To measure the amount of left hand displacement, use the *centroids* of the positions. We define the centroid of a position as the arithmetic mean of the frets of that position. For example, the centroids of the chord positions shown in Figure 1 are $(0+2+2+2+0)/5 = 1.2$, $(3+2+0+1+0)/5 = 1.2$ and $(3+5+5+5+3)/5 = 4.2$, respectively. Thus, the total displacement of the hand, which is what we want to minimize, would be the sum of the **square of the displacements** of the centroids throughout the song. For example, if the song had only three chords, C, A and C (a very short piece, indeed), and the C with centroid 1.2 was played first, then the A with centroid 1.2 and finally the C with centroid at 4.2, the total displacement would be $(1.2-1.2)^2 + (1.2-4.2)^2 = 9$. (The units are fret^2, in case you were wondering.)

Forming groups

This project can be done in groups of two or individually.

Project deliverables

The project has two deliverables. The first is a preliminary report that includes the formulation and results obtained with the brute-force method (part I). The second is a final report that includes brute-force, as well as dynamic programming and greedy algorithms (part II). Reports should have the sections of a standard scientific article: title, abstract, introduction, methodology, results, conclusions, and bibliography, but the names may vary as needed. The report must follow the IEEE *Transactions* journal format. Format specifications and templates for LaTeX and Word are available at `https://www.ieee.org/publications_standards/publications/authors/author_templates.html`. LyX also provides a template with this format (go to File ▷ New from Template and select `IEEEtran-Journal.lyx`). The algorithms can be programmed in any language. You must also submit your code with the reports.

Part I.

Brute-force search (50 pts.)

1. Find a vector representation of the solution to the problem. Specify the meaning of each entry. [2 pts.]

2. Determine the solution space S, clearly defining the set(s) to which each of the entries belong. [2 pts.]

3. Determine the size of the space. [2 pts.] (Valid only if the spacing is correct.)

4. If possible, bound the space using a constraint of the type $S' = \{\sigma \in S : \text{ restriction on } \sigma\}$. [5 pts.] (Valid only if the space is correct.)

5. Write an algorithm that explores all candidate solutions in S or S' and returns an optimal solution. [30 pts.]

   Remark. The points obtained in this section are

$$\frac{30}{k}\sum_{i=1}^{k}\frac{D_i'}{D_i},$$

where k is the number of test cases (provided later), D'_i is the total displacement (in squared units) corresponding to the guitar positions recommended by your algorithm, and D_i is the minimum total displacement (in squared units) the piece can be played with (known by the instructor, but kept secret).

6. Determine an asymptotic bound for the execution time of the algorithm. [2 pts.] (Valid only if the algorithm is correct.)
7. Run the algorithm with each of the given test cases. If the run time exceeds 10 minutes for a certain song, you may abort it. Report in a chart the execution time for each run and plot the times as a function of the chord sequence size. If several test cases have the same duration, use their average time. [5 pts.] (Valid only if the algorithm is correct.)
8. Indicate whether the execution times grew as expected with number of chords and explain why. [2 pts.] (Valid only if the algorithm is correct).

Part II.

Dynamic programming (50 pts.)

1. Determine an oracle, describe its meaning [1.5 pts.], the meaning of its arguments [1.5 pts.], the objective value [2 pts.], and the base [2 pts.] and recursive [10 pts.] steps that permit to build the oracle.
2. Based on your answer to the previous item, write an algorithm that solves the problem [20 pts.].

   Remark. The score obtained in this section will be

   $$\frac{20}{K}\sum_{i=1}^{K}\frac{D'_i}{D_i},$$

   where k is the number of test cases (provided later), D'_i is the total displacement corresponding to the guitar positions recommended by your algorithm, and D_i is the minimum total displacement the piece can be played with.
3. Find an asymptotic bound for the execution time of the algorithm. [3 pts.]
4. Run the algorithm using the provided test cases. Indicate if the obtained solutions produce the same total displacement as the solutions produced by brute force search [3 pts]. Plot the run times as a function of chord sequence size, representing the

times on a logarithmic scale. Add to the plot the curve corresponding to the times produced by the exhaustive search algorithm [4 pts]. Compare the curves and report if the relationship between times was as expected [3 pts.]. (Valid only if the algorithm is correct.)

Greedy algorithms (5–35 pts.)

1. Determine whether it is possible to solve the problem using a greedy algorithm whose execution time is asymptotically better than that of dynamic programming. Explain your answer. [5 pts.]

2. If you answered yes to the previous question:

   a) Write a greedy algorithm that solves the problem. [10 pts.]

   Remark. The score in this section will be

   $$\frac{10}{K}\sum_{i=1}^{k}\frac{D_i'}{D_i},$$

   where k is the number of test cases (provided later), D_i' is the total displacement corresponding to the guitar positions recommended by your algorithm, and D_i is the minimum total displacement the piece can be played with.

   b) Prove your solution is correct. *Suggestion*: Use the *transformation method* of any of the other methods recommended in the course's textbook. [7 pts.] (Valid only if the algorithm is correct.)

   c) Find an asymptotic bound for the execution time of the algorithm. [3 pts] (Valid only if the algorithm is correct.)

   d) Run the algorithm using the provided test cases. Indicate whether the solutions obtained produce the same total displacement as the previous algorithms [3 pts]. Plot the run times as a function of the size of the chord sequences, representing the times on a logarithmic scale. Add to the plot the curve corresponding to the times produced by your brute force search and dynamic programming algorithms [4 pts.] Compare the curves and report whether the relationship between the times was as expected [3 pts.] (Valid only if the algorithm is correct).