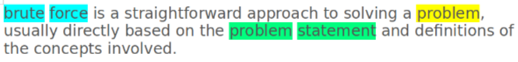
Question 2: Colourful Study Notes

Tala is studying very hard for COMP20007. They read the corresponding book chapters every week to consolidate their understanding of the subject material.

To help with their studying. Tala likes to use highlighters of different colours. Here is an example, taken from the chapter on Brute Force methods:



brute force is a straightforward approach to solving a problem,
usually directly based on the problem statement and definitions of
the concepts involved.

In this example, Tala used **blue** to highlight the words "brute force", **yellow** to highlight the word "problem" and **green** to highlight "problem statement".

After doing this manually for years, Tala had the idea to write a program that automatically highlights terms. To do this, they used a program to read their past notes and check, for each word, how many times it was highlighted. This led to a collection of tables, one per word, with scores relative to how frequent that word was highlighted and which colour. For example, for the word "problem", the table looks like this:

colour	score
blue	0
yellow	10
green	8
no colour	5

In this example, **yellow** has a larger score because it is the most frequent colour the word "problem" appears in Tala's notes. Note there is also a row for **no colour**, meaning the word is not highlighted. After using the program to read their notes, Tala ends up with a set of tables like the one above, one per word.

Part A (code)

For the first automatic highlighter program, you should implement a C program that reads as input:
- A **sentence**. This is just a sequence of words split by whitespace, with no punctuation marks.
- A **set of word tables.** The set will only contain tables for words that appear in the sentence. You can assume there are only 4 colours and they are represented as **integers** from 0 to 3: 0 is "no colour", 1 is "green", 2 is "yellow" and 3 is "blue".

Then, it generates as the output:
- A **sequence of colours**, one per word. This should be a sequence of integer values, where each integer represent a colour, as above.

The output sequence of colours should follow an **optimisation criterion:** it is the sequence with the **highest total score**. For Part A, the total score is the sum of individual scores per word. Formally speaking, assume a sentence has *n* words, with each word *w* numbered from *1* to *n*. Assume *F(n)* gives the maximum score for a sentence, which we define as

$$F(n) = \max_{C=c_1 \ldots c_n} \sum_{i=1}^{n} WC(w_i, c_i) = \sum_{i=1}^{n} \max_{c} WC(w_i, c)$$

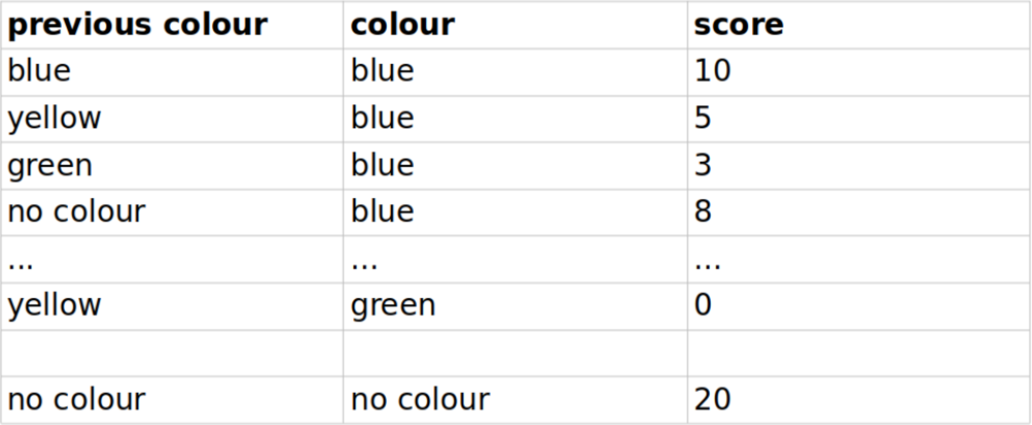
where *WC(w,c)* corresponds to the score for colour *c* in word table *w*. The first term states that our goal is to the maximise the score given a **sequence** of words/colours The second term then simplifies this to maximise the score for each **individual** word/colour.

The equation above only gives the maximum score: your code should generate the sequence of colours that gives this maximum score.

Part B (code)

Following testing of the previous program, Tala is a bit frustrated because it would always pick the same colour for each term. But as show in the example above with the word "problem", sometimes the same word can be highlighted with different colours, depending on the sentence.

To solve this problem, Tala introduces an extra **colour transition** table. This table gives scores for colours given the **previous colour**. Here is an example:



previous colour	colour	score
blue	blue	10
yellow	blue	5
green	blue	3
no colour	blue	8
...	...	...
yellow	green	0
no colour	no colour	20

In this example table, if the previous word is highlighted in **blue** and the current word is also **blue** (as in the "brute force" example above), this **transition** gives a score of 10.

For the second automatic highlighter, you should enhance the code in Part A. The input is now:

- A **sentence**. As in Part A.
- A **set of word tables.** As in Part A.
- A **colour transition table**. Each entry in this table has two integers, corresponding to **previous colour** and **colour**, and a score. Integers represents colours, as in Part A.

The output is the same as Part A: a sequence of integers that gives the optimal highlighting. However, the criterion of highest total score now needs to sum the colour transition scores as well. Your program should follow a **greedy** approach: for every word from left to right, select the colour based on the maximum **sum** of two scores: the one from the word table and the one for the transition table.

Part C (written solution)

Tala is pleased with the result but notices the colouring could sometimes still be better. They realise the greedy algorithm in Part B is not actually generating the optimal sequence. To see this, we can write the formal equation for the maximiser with the colour transition table:

$$F(n) \; = \max_{C=c_1 \ldots c_n} WC(w_1, c_1) + \sum_{i=2}^{n} \left(WC(w_i, c_i) + CT(c_{i-1}, c_i) \right)$$

CT(c1,c2) corresponds to the score in the colour transition where *c1* is the previous colour. The equation is similar to the one in Part A, but for the second to the last word, we take the sum of *WC* and *CT*. The main challenge here is the *CT* term inside the sum, which requires the colour for the previous word. This is fine if we are iterating over entire sequences of colours. However, the greedy algorithm picks one colour at a time from left to right, potentially missing the optimal sequence because it contains a colour that was optimal for an individual word.

Tala then decides to create a better algorithm. Their first idea is to use a brute force approach: generate all possible sequences of colours, calculate their scores and select the one with max score.

Assume you have *n* words and a total of *C* colours. State the complexity of the approach above in all cases.

Part D (written solution)

Tala quickly realises the brute force approach is too slow. They believe it is possible to use a Dynamic Programming solution instead. Here is Tala's reasoning:

- For the first word, there is no transition, so we can pick the best colour according to the word table.
- For the second word and afterwards, there are two options:
- Option 1: the best sequence includes the best colour for the previous word. In this case, we just need to pick the colour that maximises the sum of *WC(w,c)* and *CT(c1,c),* where *c1* is the colour of the previous word.
- Option 2: the best sequence does not include the best colour for the previous word. In this case, we need to check the sum of *WC(w,c)* and *CT(c1,c)* for **all other colours** that the previous word **could** have been highlighted with. Then we choose the colour that maximises this sum.

The greedy algorithm only stored the score for the best colour at each word. The above reasoning shows that we can get the best sequence if we store the scores for **all colours** at each word, that is, if we also take into account the total number of colours.

With this in mind, write a recurrence relation for the score *F.*

Part E (code)

Write a C program that implements the dynamic programming approach in Part D to get the best **score** for a sequence of colours. Inputs are the same as Part B, Output should be a single number containing the best score. For this part, you can assume you only have 4 colours, as in Parts A and B.

Part F (code)

Modify the C program from Part E to get the best sequence of colours. Inputs and Outputs are the same as Part B. For this part, you can assume you only have 4 colours, as in Parts A and B.

Part G (written solution)

Assume you have *n* words and a total of *C* colours. State the complexity of the dynamic programming approach described in Part D in all cases.