FIT1008 – Intro to Computer Science
Workshop Week 10

Semester 1, 2018

Objectives of this practical session

- To practice the implementation of Dynamic Programming solutions.

Task 1

Consider the edit distance problem discussed in the lectures.

1. Based on the Edit Distance algorithm given in the lectures create a function that finds the sequence of transformations required to transform one string into another.
2. Rewrite the Edit Distance algorithm given in the lectures to find the minimum cost of transforming one string into another when you have different costs for substitutions, deletions and insertions. These costs should be given as parameters of your algorithm.
3. Implement a purely recursive solution for the edit distance problem, and compare the running time to the Dynamic Programming solution studied in the lectures. Do this by using the technique used in Task 2 of the week 6 workshop, i..e, using the `timeit` module and generating random instances of the problem.

Task 2

1. Based on the DP Knapsack algorithm discussed in the lectures, write a function that finds not only the optimal value of the knapsack, but also the items associated with the solution.
2. Implement a modification of the algorithm discussed in the lectures to solve the knapsack problem, now assuming that you are allowed to take as many items as you can of each type as long as the knapsack capacity is not exceeded.

(You should try to reach this point before your lab session)

Task 3

1. Implement a brute force algorithm to solve the Knapsack problem discussed in the lectures, and compare the performance to the Dynamic Programming solution studied in the lectures. Do this by using the technique used in Task 2 of the week 6 workshop, i..e, using the `timeit` module and generating random instances of the problem.

2. Implement a greedy algorithm to solve the Knapsack problem discussed in the lectures. Using random instance of the problem try to answer the following question: How likely is it that the greedy algorithm will produce the optimal solution for different problem sizes.

Task 4

This is the code review task.
Many typesetting systems have the functionality to *justify* text. This prac sheet was created with LaTeX, which by default "justifies" each line deciding where to break a sequence of words [1]
Design an algorithmic strategy to determine line break locations. Every arrangement of lines will add some extra spaces in each line to make the text nice and tidy. You do no need to add the spaces, it is sufficient to determine the line breaks. For simplicity we will ignore hyphenation and assume a line break can only occur after a complete word.
Consider the text: "you can use dynamic programming to justify text and I learned that in FIT1008".
Here's one possible arrangement (let's call it example 1):

```
you can use
dynamic
programming
to justify text
and I learned
that in FIT1008
```

Here's another possible arrangement (let's call it example 2):

```
you can
use dynamic
programming
to justify text
and I learned
that in FIT1008
```

[1] **Note:** Please Read the problem statement *carefully* a few times. Take notes. Before asking any questions make a list of all questions you have and try to answer them. When asking a question state what you tried, or how you attempted to solve it. Study the problem.

The idea is to obtain lines that are balanced. This is often achieved by minimising a cost function. Suppose you are given a list of n words, $L = (l_0, l_1, \ldots, l_{n-1})$ as well as a line width M.
A common cost function is $C(L) = \sum_{l=0^{n}-1} c_j$, with $c_j = x^3$. Here c_j is the cost of line j, when x is the unused space in the line. A line running from word i to word j, has $x(i,j) = M - j - i - \sum_{k=i^{j}} len(l_k)$

- Given as input a list of words $W = [w_0, w_1, \ldots, w_{n-1}]$, and a width M , determine what constitutes a solution to the "justification" problem.
- How can you use dynamic programming to find the optimal cost? Implement your DP solution.
- How can you use dynamic programming to find the optimal arrangement? Implement your solution.
- **Optional:** Implement a `print_tidy(text, M)` function that nicely prints out the text with width M.
- **Optional:** Why does the cost function $c_j = x^3$ work well? What about just using $c_j = x$?
- **Optional:** Add a justify functionality to the editor from the previous prac.

s