**Change Problem:**

**Reading**

Change problem: Section "An Introduction to Dynamic Programming: The Change Problem" of [CP]

**Visualizations**

[Making change](http://www.cs.usfca.edu/~galles/visualization/DPChange.html) by David Galles

**References**

[CP] Phillip Compeau, Pavel Pevzner. Bioinformatics Algorithms: An Active Learning Approach. Active Learning Publishers. 2014.

## String Comparison:

## Reading

Edit distance: Section 6.3 of [DPV08]

## Visualizations

[Edit distance calculator](http://www.let.rug.nl/kleiweg/lev/) by Peter Kleiweg

[Longest common subsequence](http://www.cs.usfca.edu/~galles/visualization/DPLCS.html) by David Galles (note the longest common subsequence problem is a special case of the edit distance problem where we allow insertions and deletions only)

## Advanced Reading

Chapter 5 "How Do We Compare Biological Sequences" of [CP]

[Advanced dynamic programming lecture notes](http://jeffe.cs.illinois.edu/teaching/algorithms/notes/06-sparsedynprog.pdf) by Jeff Erickson

Both sources explain, in particular, Hirschber's algorithm that allows to compute an optimal alignment (but not just its score!) of two strings of length nnn and mmm in quadratic time O(nm)O(nm)O(nm) and a linear space O(m+n)O(m+n)O(m+n) only.

## References

[DPV] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. Algorithms (1st Edition). McGraw-Hill Higher Education. 2008.

[CP] Phillip Compeau, Pavel Pevzner. Bioinformatics Algorithms: An Active Learning Approach. Active Learning Publishers. 2014.

## Knapsack:

## Reading

Knapsack: Section 6.4 of [DPV08]

## References

[DPV] Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani. Algorithms (1st Edition). McGraw-Hill Higher Education. 2008.

## Polynomial vs Pseudopolynomial

Many of you are surprised to learn that the running time O(nW) for the knapsack algorithm is called pseudo polynomial, but not just polynomial. The catch is that the input size is proportional to , rather than W.

To further illustrate this, consider the following two scenarios:

1. The input consists of m objects (say, integers).
2. The input is an integer m.

They look similar, but there is a dramatic difference. Assume that we have an algorithm that loops for m iterations. Then, in the first case it is a polynomial time algorithm (in fact, even linear time), whereas in the second case it is an exponential time algorithm. This is because we always measure the running time in terms of the input size. In the first case the input size is proportional to m, but in the second case it is proportional to . Indeed, a file containing just a number “100000” occupies about 7 bytes on your disc while a file containing a sequence of 100000 zeroes (separated by spaces) occupies about 200000 bytes (or 200 KB). Hence, in the first case the running time of the algorithm is O(size), whereas in the second case the running time is .

Let’s also consider the same issue from a slightly different angle. Assume that we have a file containing a single integer 74145970345617824751. If we treat it as a sequence of m=20 digits, then an algorithm working in time O(m) will be extremely fast in practice. If, on the other hand, we treat it as an integer m=74145970345617824751, then an algorithm making m iterations will work for

years, assuming that the underlying machine performs operations per second.

Further reading: [a question at stackoverflow](https://stackoverflow.com/questions/4538581/why-is-the-knapsack-problem-pseudo-polynomial#answer-4538668).