COMP1730/COMP6730
Programming for Scientists

Control, part 3: Dynamic programming

Outline

* **Dynamic programming**
* (DNA) sequence alignment

Algorithm design paradigms

* When looking at the algorithms designed to efficiently solve many optimisation and search problems, certain general approaches can be identified.
* These are called algorithm design paradigms. In this lecture we will cover one common approach: **Dynamic programming**
* (see `https://en.wikipedia.org/wiki/Algorithmic_paradigm`)
* We will see its application in the problem of (DNA) sequence alignment

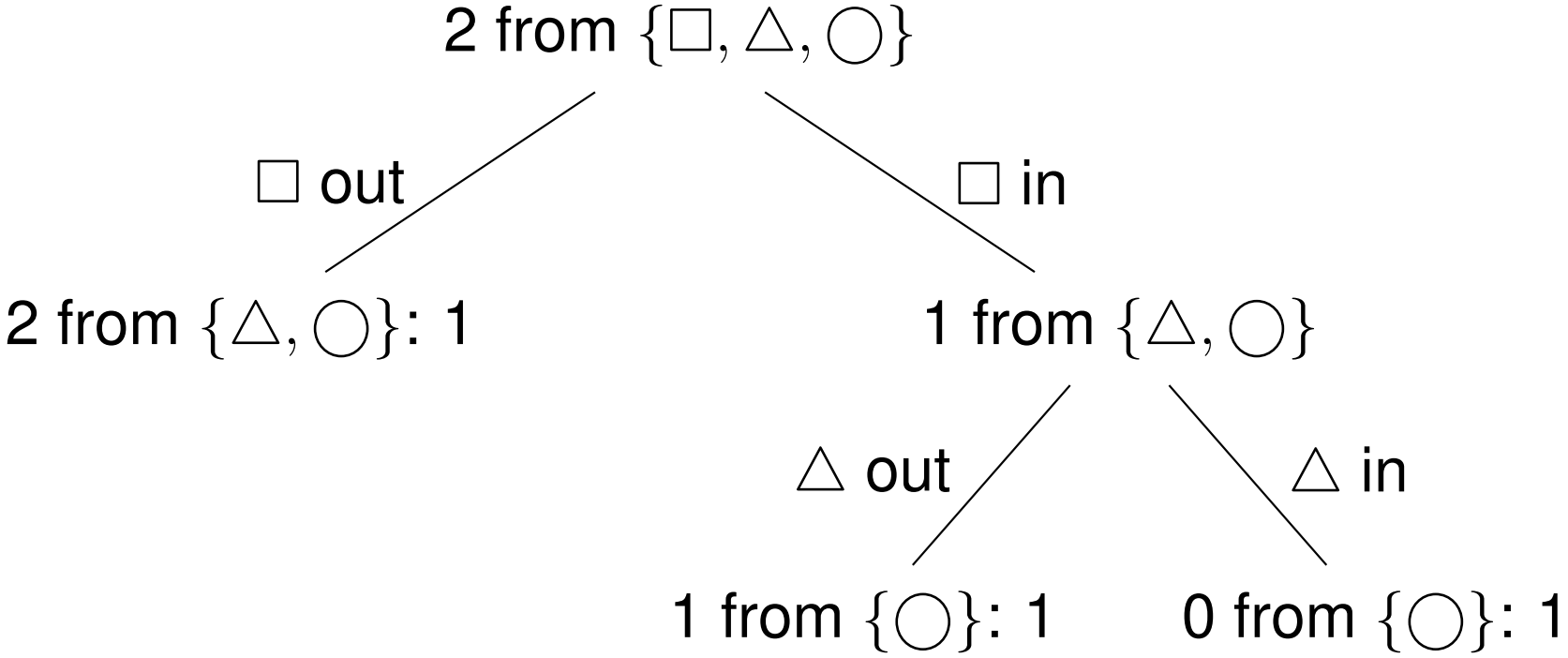
Dynamic programming

Dynamic programming is an approach that can often (not always) be used to efficiently solve certain optimisation problems where:

* the problem can be broken down into sub-problems, such that-
* if the subproblems could be solved, then the partial solutions can be efficiently combined into a full solution .

Example: Counting selections

* Compute the number of ways to choose k elements from a set of n, $C(n, k)$, aka the binomial coefficient.



* Simple recursive formulation:

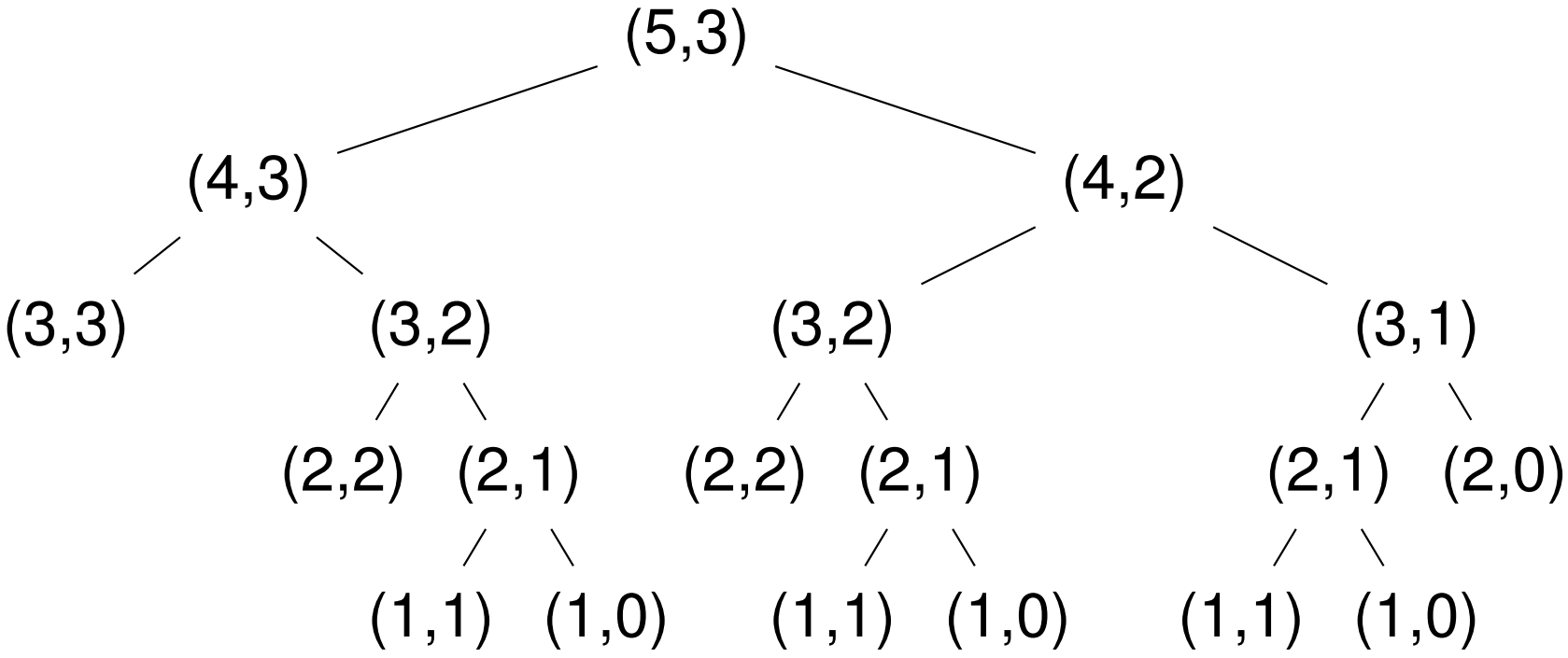
$$C(n,k) = C(n-1,k) + C(n-1,k-1)$$
$$C(n,0) = 1$$
$$C(n,n) = 1$$

* Simple recursive implementation:

```
def choices(n, k):
    if k == n or k == 0:
        return 1
    else:
        return choices(n - 1, k) + choices(n - 1, k - 1)
```

* This brute-force solution is $O(2^n)$. How to implement this efficiently?

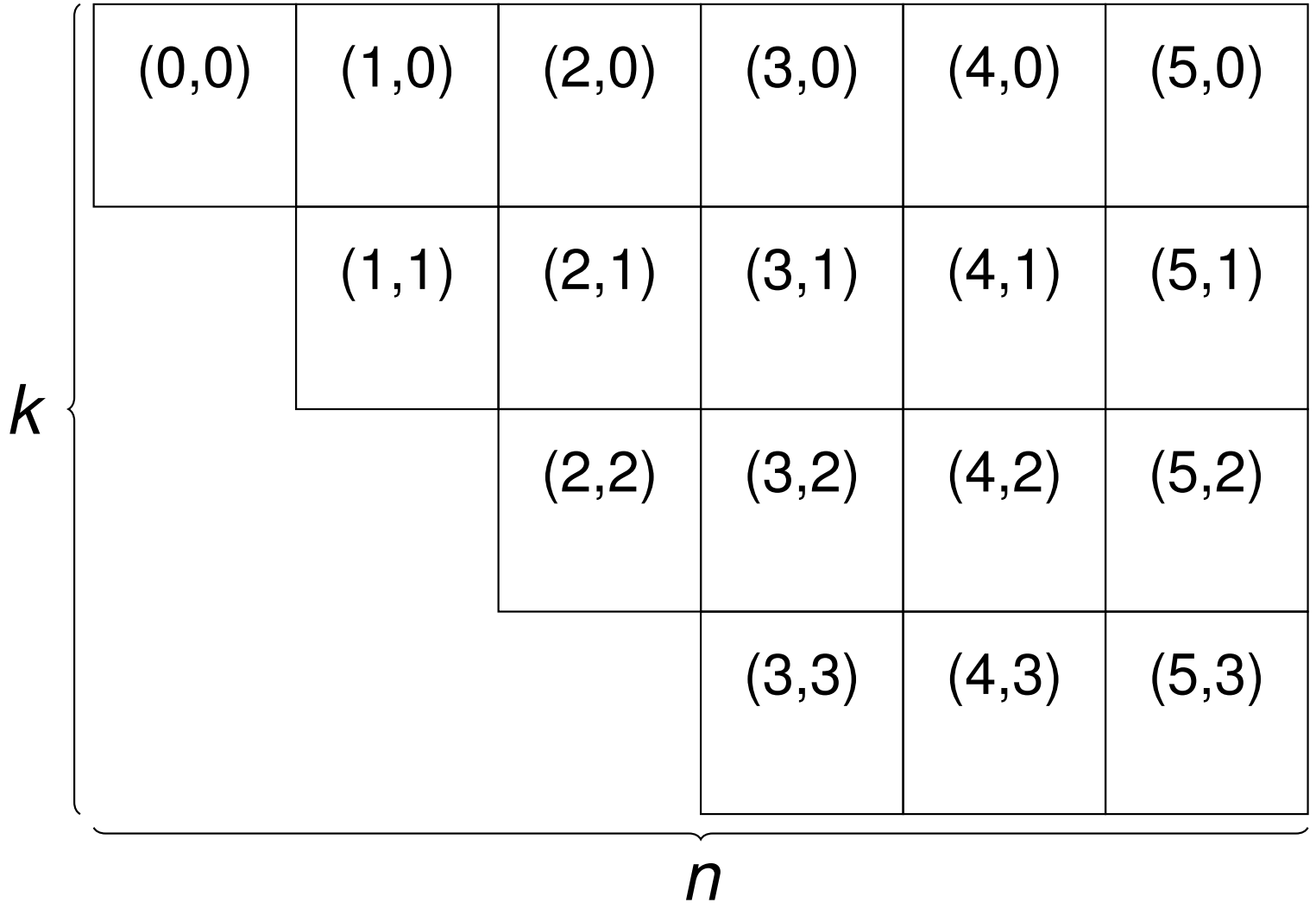
* Recursive calls by `choices(5, 3)`:



* Note repeated work.

* The idea of **dynamic programming** is to store answers to (recursively defined) subproblems, to avoid computing them repeatedly.
  - Trade memory for computation time.
  - solve the base cases first;
  - then, repeatedly, solve larger problems using subproblems which already solved;
  - until the whole problem is solved.
* Need a way to index subproblems- we will use a 2D table.

* Array of subproblems:



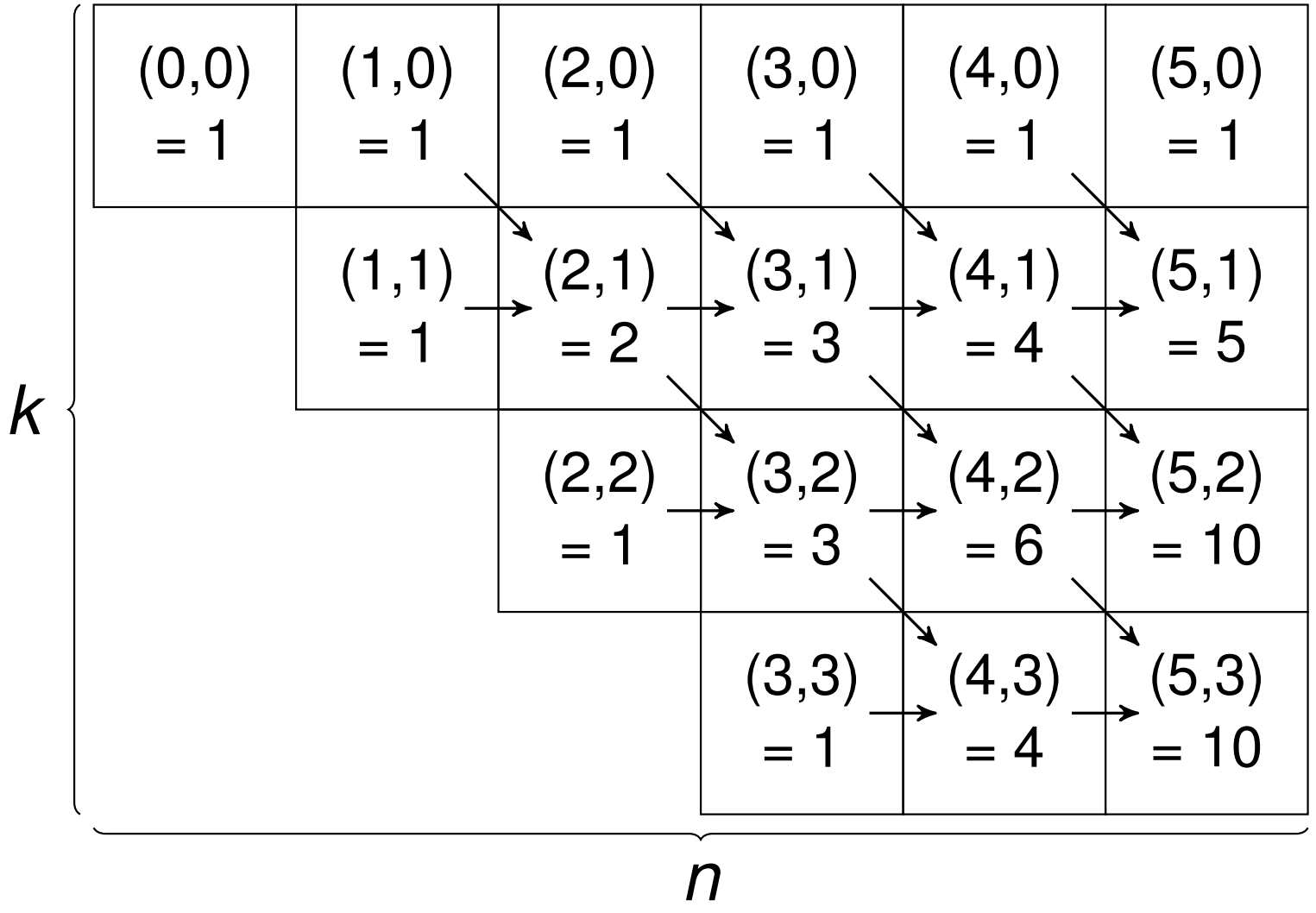
* With base cases solved:

(0,0) = 1	(1,0) = 1	(2,0) = 1	(3,0) = 1	(4,0) = 1	(5,0) = 1
	(1,1) = 1	(2,1)	(3,1)	(4,1)	(5,1)
		(2,2) = 1	(3,2)	(4,2)	(5,2)
			(3,3) = 1	(4,3)	(5,3)

k

n

* Complete:


(0,0) = 1
(1,0) = 1
(2,0) = 1
(3,0) = 1
(4,0) = 1
(5,0) = 1
(1,1) = 1
(2,1) = 2
(3,1) = 3
(4,1) = 4
(5,1) = 5
(2,2) = 1
(3,2) = 3
(4,2) = 6
(5,2) = 10
(3,3) = 1
(4,3) = 4
(5,3) = 10
k
n

Computational complexity

* The dynamic programming solution has time complexity O($n \times k$)
* (Note that the table has $n \times k$ entries and we need to scan half of it to complete it)
* (Note: this table was first published by Plaise Pascal in 1665).

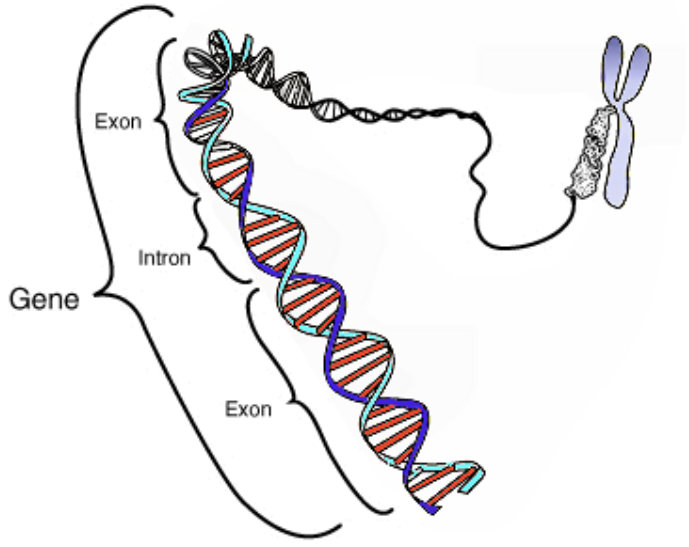
Outline

* Dynamic programming
* **(DNA) pairwise sequence alignment**

BRCA 1 gene

CTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAAGCGCGGGAATTACAGATAAATTAAAACTGCGACTGCGCGGCGTGAGCTCGC
TGAGACTTCCTGGACGGGGGACAGGCTGTGGGGTTTCTCAGATAACTGGGCCCCTGCGCTCAGGAGGCCTTCACCCTCTGCTCTGGTTC
ATTGGAACAGAAAGAAATGGATTTATCTGCTCTTCGCGTTGAAGAAGTACAAAATGTCATTAATGCTATGCAGAAAATCTTAGAGTGTC
CCATCTGTCTGGAGTTGATCAAGGAACCTGTCTCCACAAAGTGTGACCACATATTTTGCAAATTTTGCATGCTGAAACTTCTCAACCAG
AAGAAAGGGCCTTCACAGTGTCCTTTATGTAAGAATGATATAACCAAAAGGAGCCTACAAGAAAGTACGAGATTTAGTCAACTTGTTGA
AGAGCTATTGAAAATCATTTGTGCTTTTCAGCTTGACACAGGTTTGGAGTATGCAAACAGCTATAATTTTGCAAAAAAGGAAAATAACT
CTCCTGAACATCTAAAAGATGAAGTTTCTATCATCCAAAGTATGGGCTACAGAAACCGTGCCAAAAGACTTCTACAGAGTGAACCCGAA
AATCCTTCCTTGGAAACCAGTCTCAGTGTCCAACTCTCTAACCTTGGAACTGTGAGAACTCTGAGGACAAAGCAGCGGATACAACCTCA
AAAGACGTCTGTCTACATTGAATTGGGATCTGATTCTTCTGAAGATACCGTTAATAAGGCAACTTATTGCAGTGTGGGAGATCAAGAAT
TGTTACAAATCACCCCTCAAGGAACCAGGGATGAAATCAGTTTGGATTCTGCAAAAAAGGCTGCTTGTGAATTTTCTGAGACGGATGTA
ACAAATACTGAACATCATCAACCCAGTAATAATGATTTGAACACCACTGAGAAGCGTGCAGCTGAGAGGCATCCAGAAAAGTATCAGGG
TGAAGCAGCATCTGGGTGTGAGAGTGAAACAAGCGTCTCTGAAGACTGCTCAGGGCTATCCTCTCAGAGTGACATTTTAACCACTCAGC
AGAGGGATACCATGCAACATAACCTGATAAAGCTCCAGCAGGAAATGGCTGAACTAGAAGCTGTGTTAGAACAGCATGGGAGCCAGCCT
TCTAACAGCTACCCTTCCATCATAAGTGACTCTTCTGCCCTTGAGGACCTGCGAAATCCAGAACAAAGCACATCAGAAAAAGCAGTATT
AACTTCACAGAAAAGTAGTGAATACCCTATAAGCCAGAATCCAGAAGGCCTTTCTGCTGACAAGTTTGAGGTGTCTGCAGATAGTTCTA
CCAGTAAAAATAAAGAACCAGGAGTGGAAAGGTCATCCCCTTCTAAATGCCCATCATTAGATGATAGGTGGTACATGCACAGTTGCTCT
GGGAGTCTTCAGAATAGAAACTACCCATCTCAAGAGGAGCTCATTAAGGTTGTTGATGTGGAGGAGCAACAGCTGGAAGAGTCTGGGCC
ACACGATTTGACGGAAACATCTTACTTGCCAAGGCAAGATCTAGAGGGAACCCCTTACCTGGAATCTGGAATCAGCCTCTTCTCTGATG
ACCCTGAATCTGATCCTTCTGAAGACAGAGCCCCAGAGTCAGCTCGTGTTGGCAACATACCATCTTCAACCTCTGCATTGAAAGTTCCC
CAATTGAAAGTTGCAGAATCTGCCCAGAGTCCAGCTGCTGCTCATACTACTGATACTGCTGGGTATAATGCAATGGAAGAAAGTGTGAG
CAGGGAGAAGCCAGAATTGACAGCTTCAACAGAAAGGGTCAACAAAAGAATGTCCATGGTGGTGTCTGGCCTGACCCCAGAAGAATTTA
TGCTCGTGTACAAGTTTGCCAGAAAACACCACATCACTTTAACTAATCTAATTACTGAAGAGACTACTCATGTTGTTATGAAAACAGAT
GCTGAGTTTGTGTGTGAACGGACACTGAAATATTTTCTAGGAATTGCGGGAGGAAAATGGGTAGTTAGCTATTTCTGGGTGACCCAGTC
TATTAAAGAAAGAAAAATGCTGAATGAG

Biological sequence data



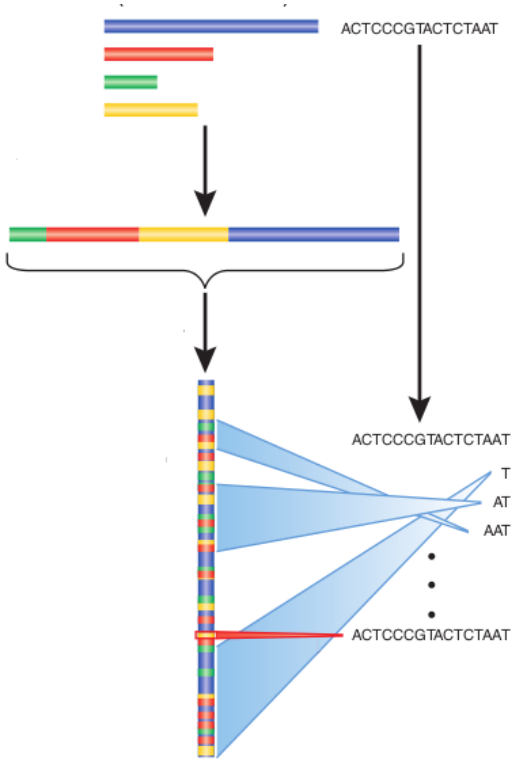
* DNA and RNA.
* Protein amino acid sequence.
* Arrangement of genes in chromosome / genome.
* Human DNA is $\sim$3 billion bp.
* BRCA 1 & 2 genes are $\sim$80kb (incl. exons).
* DNA sequencer reads are 100–2k bases.
* Across evolutionary time, both mutations and small insertions and deletions (indels) can occur.

* Alignment

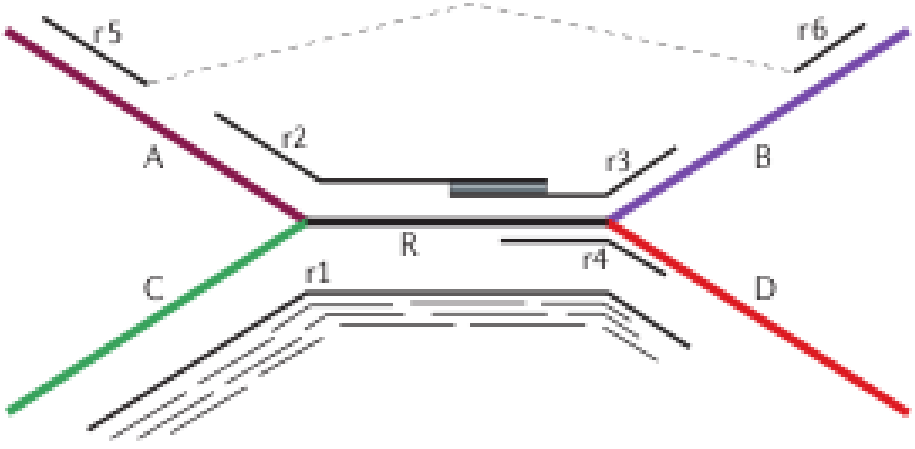
```
GAATTCAG  GAATTCAG
|  |  ||  |  |  ||  |  |
GGA-TC-G  GCAT-C-G

GAATTC-A  GAATTC-A
|  |  ||  |  |  ||  |  |
GGA-TCGA  GCAT-CGA
```

* Mapping



* Assembly



Edit distance

* We need a measure of sequence similarity we can optimise. Typically, we will assign different scores to matches, mismatches, and indels at each position.
* In this example we will use a simple edit distance: Minimum (weighted) number of “edit operations” needed to transform one sequence into the other.
* Levenshtein (string edit) distance:
  - insert a character (gap in other string);
  - delete a character (gap in this string);
  - substitute a character

* distance(GAATTCA, GGATCGA) = 3.
* Edits:

		G A A T T C A
(subst. 1 G)	$\Rightarrow$	G G A T T C A
(del 4)	$\Rightarrow$	G G A T C A
(ins 5 G)	$\Rightarrow$	G G A T C G A

* Alignment:

G	**A**	A	T	T	C	_	A
G	**G**	A	T	_	C	G	A
	+1			+1		+1	

Recursive formulation

$$\text{dist}(s, '\,') = \text{len}(s) * w_{\text{gap}}$$
$$\text{dist}('\,', t) = \text{len}(t) * w_{\text{gap}}$$

$$\text{dist}(s + x, t + y) =$$
$$\min \begin{cases} \text{dist}(s, t) + \begin{cases} 0 & \text{if} x = y \\ w_{\text{sub}} & \text{otherwise} \end{cases} \\ \text{dist}(s + x, t) + w_{\text{gap}} \\ \text{dist}(s, t + y) + w_{\text{gap}} \end{cases}$$

* In example, $w_{\text{sub}} = w_{\text{gap}} = 1$.

```
def align(s, t, w_gap = 1, w_sub = 1):
    """
        Align two sequences s and t with gap cost
        w_gap and substitution cost w_gap
        Returns the edit distance between 2 sequences
    """
    if len(s) == 0:
        return len(t) * w_gap
    elif len(t) == 0:
        return len(s) * w_gap
    else:
        if s[-1] == t[-1]:
            d1 = align(s[:-1], t[:-1])
        else:
            d1 = align(s[:-1], t[:-1]) + w_sub
        d2 = align(s, t[:-1]) + w_gap
        d3 = align(s[:-1], t) + w_gap
        return min(d1, d2, d3)
```

Dynamic programming formulation

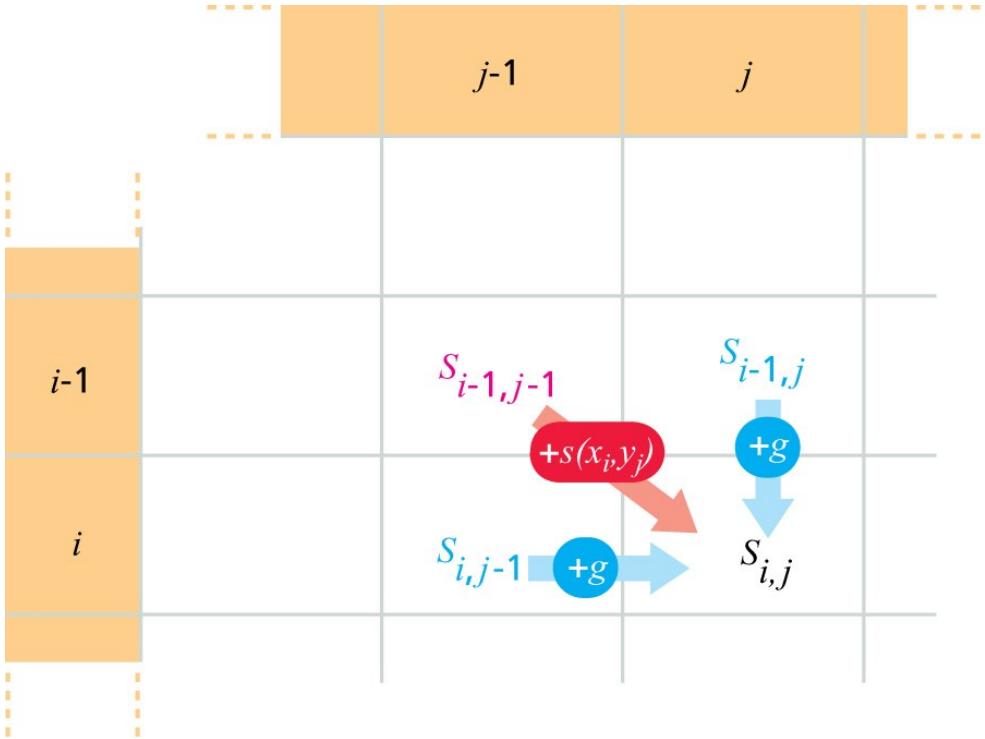
* We will use a table with bases of one sequence indexing rows, and bases of other sequence indexing columns.
* Each table position `[i, j]` will then be updated to store the minimum edit distance between the subsequences from 0 to i and 0 to j.
* The minimum edit distance between the full sequences can then be read from the bottom right position of the table.

Dynamic programming: Pairwise sequence alignment

A. GLOBAL alignment

Match score = +1
Mismatch score = 0
Gap penalty = -1

$$\text{Score}(i,j) = \max \begin{cases} \text{(i-1,j-1) + match/mismatch = diagonal move} \\ \text{(i-1,j) - gap penalty = horizontal move} \\ \text{(i,j-1) - gap penalty = vertical move} \end{cases}$$

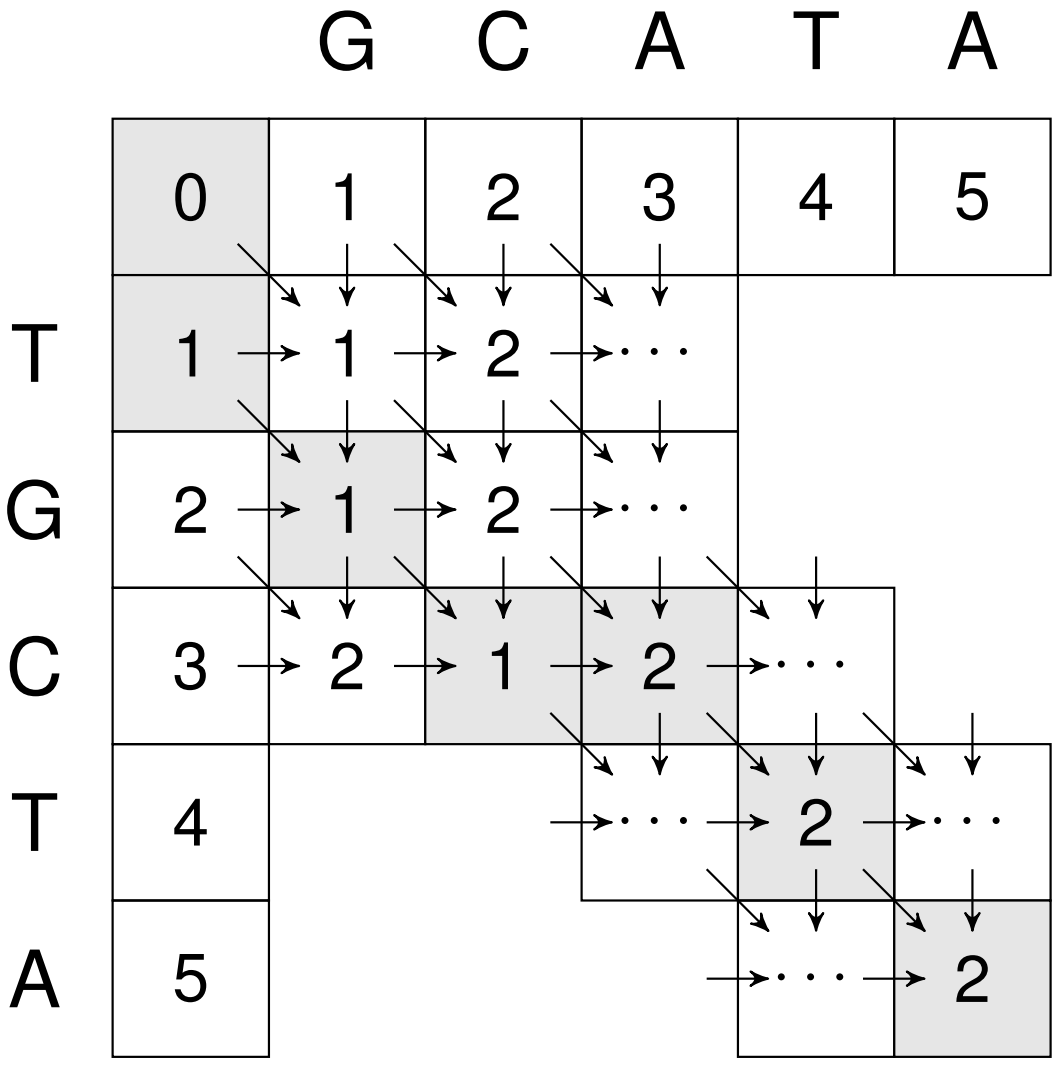


Dynamic programming formulation

* Note that in this example we only return the edit distance of the optimal alignment of the full sequences, we do not return the actual optimal alignment.
* This can be efficiently calculated by a back-trace from the bottom right position, but we do not discuss that in this simple example.

Dynamic programming formulation

* How to index subproblems?
  - Each call aligns two sequence prefixes.
  - (i, j): `align(s[:i], t[:j])`.
* Base cases?
  - One sequence is empty ($i = 0$ or $j = 0$).
* Update: min of $(i - 1, j - 1)$ (plus subst. weight if `s[i-1] != t[j-1]`), $(i - 1, j)$ plus gap weight, and $(i, j - 1)$ plus gap weight.


G C A T A
T G C T A
0 1 2 3 4 5
1 1 2 · · ·
2 1 2 · · ·
3 2 1 2 · · ·
4 · · · 2 · · ·
5 · · · 2

Time complexity

$O(n^2)$ as n by n table needs to be scanned once.
Naive approach would be exponential time.

Summary

Dynamic programming is one common algorithm design paradigm that has many applications in science and engineering. It was first applied to the key problem of pairwise sequence alignment by Needleman-Wunsch and Smith-Waterman. In computational biology it is used for pairwise sequence alignment, RNA secondary structure determination and other applications

Summary

In general, it can be applied for optimisation problems where:

* The problem can be broken into subproblems, such that
* the subproblems can be combined efficiently to give the solution to the full problem