COMP1730/COMP6730
Programming for Scientists

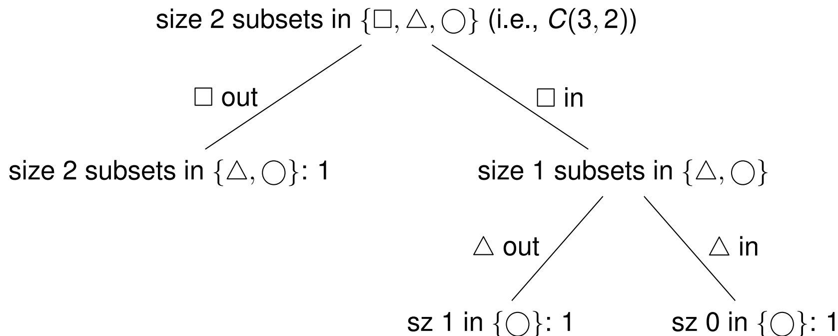
Dynamic programming

Outline

- **Dynamic programming toy example**: counting subsets
- Real-world example: (DNA) sequence alignment

Example: Counting subsets of size k

- Compute number of different subsets with k elements (i.e., of size k) in a set with n elements ($n \geq k \geq 0$)
- Denoted as $C(n, k)$ (example with $n = 3$, $k = 2$)



Recursive formulation

* Simple recursive formulation:

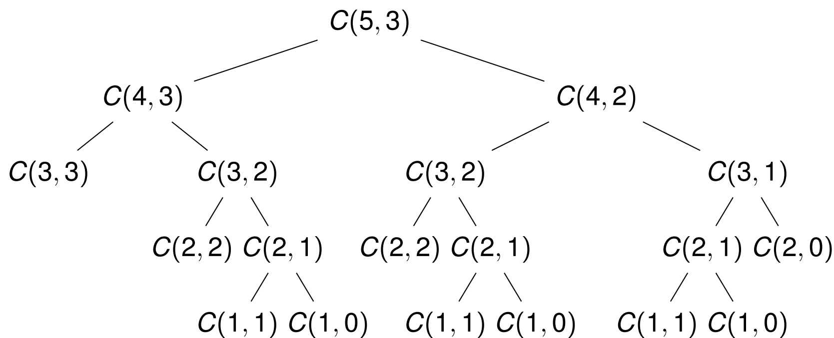
$$C(n,k) = C(n-1,k) + C(n-1,k-1)$$
$$C(n,0) = 1$$
$$C(n,n) = 1$$

* Simple recursive implementation:

```
def C(n, k):
    if k == n or k == 0:
        return 1  # base cases
    else:
        return C(n-1, k) + C(n-1, k-1)
```

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

Call tree for $C(5, 3)$



Note repeated work

Dynamic programming (basic idea)

* The idea of **dynamic programming** is to store answers to (recursively defined) subproblems, to avoid computing them repeatedly
* Trade memory for computation time: at the price of extra memory we (significantly) reduce number of operations
* By computing subproblem solutions “from the bottom up”, we can also transform a recursive algorithm into an iterative one:
  - solve the base cases first;
  - then, repeatedly, solve problems whose subproblems are already solved;
  - repeat until the whole problem is solved
* Need a way to index stored solutions to subproblems

2D array with subproblem solutions

$C(0,0)$	$C(1,0)$	$C(2,0)$	$C(3,0)$	$C(4,0)$	$C(5,0)$
	$C(1,1)$	$C(2,1)$	$C(3,1)$	$C(4,1)$	$C(5,1)$
		$C(2,2)$	$C(3,2)$	$C(4,2)$	$C(5,2)$
			$C(3,3)$	$C(4,3)$	$C(5,3)$

$k+1$ (rows)

$n+1$ (columns)

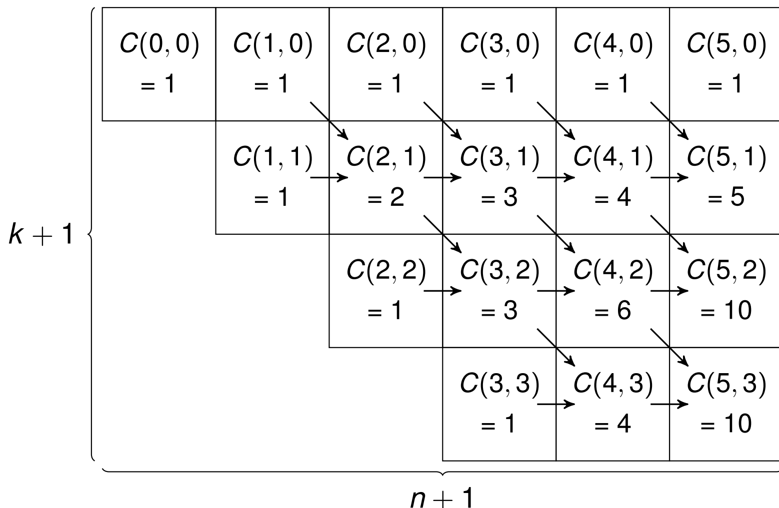
With base cases solved

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

Rows: $k+1$

Columns: $n+1$

Complete remaining subproblems



Computational complexity analysis

* 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

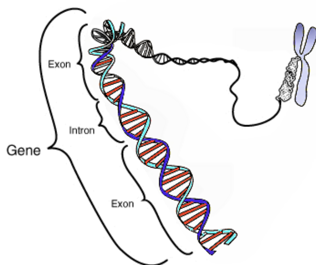
* Dynamic programming toy example: counting subsets
* **Real-world example**: (DNA) sequence alignment

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



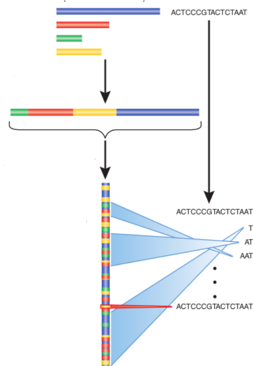
* Human DNA is ∼3 billion (i.e., 3×10^9) base pairs
* BRCA 1 & 2 genes are ∼80kb (incl. exons)
* Harmful mutations change as few as 2 bases
* DNA sequencer reads are 100–2k bases

* Alignment

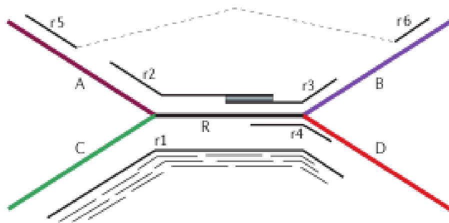
```
GAATTCAG   GAATTCAG
|  | || |  |  || |  |
GGA-TC-G   GCAT-C-G

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

* Mapping



* Assembly



Edit distance

* Minimum (weighted) number of “edit operations” needed to transform one sequence (source) into the other (target)
* Levenshtein (string edit) distance. Edit operations:
  - **Insert** a character (gap in source string)
  - **Delete** a character (gap in target string)
  - **Substitute** a character
* Minimum edit distance equals to the “score” of best sequence alignment

Levenshtein edit distance (example)

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

$$
\begin{array}{rccccccccc}
 & & G & A & A & T & T & C & A \\
\text{(subst. 1 G)} & \Rightarrow & G & G & A & T & T & C & A \\
\text{(del 4)} & \Rightarrow & G & G & A & T & C & A & \\
\text{(ins 5 G)} & \Rightarrow & G & G & A & T & C & G & A
\end{array}
$$

* Alignment (score$=$ 3):

$$
\begin{array}{cccccccc}
G & \mathbf{A} & A & T & T & C & __ & A \\
G & \mathbf{G} & A & T & __ & C & G & A \\
 & +1 & & & +1 & & +1 &
\end{array}
$$

Recursive formulation (definition)

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

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

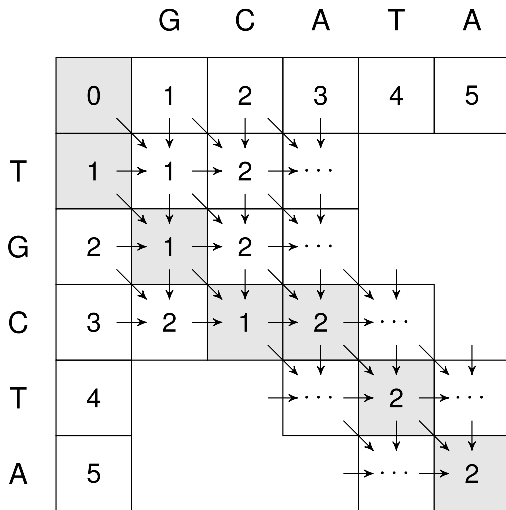
Recursive formulation (implementation)

```
def edit_distance(s, t, w_gap = 1, w_sub = 1):
    """
    Returns the edit distance between 2 sequences
    s and t with gap cost w_gap and substitution
    cost w_sub
    """
    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 = edit_distance(s[:-1], t[:-1])
        else:
            d1 = edit_distance(s[:-1], t[:-1]) + w_sub
        d2 = edit_distance(s, t[:-1]) + w_gap
        d3 = edit_distance(s[:-1], t) + w_gap
        return min(d1, d2, d3)
```

Dynamic programming formulation (sketch)

- How to index stored solutions to subproblems?
  - 2D array of shape `(len(s)+1, len(t)+1)`
  - (i, j): `edit_distance(s[:i], t[:j])`
- Base cases?
  - One of the two sequences is empty ($i = 0$ or $j = 0$)
- Update: (i, j) is equal to minimum of:
  - $(i - 1, j - 1)$ (plus subst. weight if `s[i-1] != t[j-1]`)
  - $(i - 1, j)$ plus gap weight
  - $(i, j - 1)$ plus gap weight

Dynamic programming formulation (dynamics)



Time complexity analysis of DP solution

- $O(n^2)$ as n by n table needs to be scanned once. Naive recursive implementation is exponential time.

Take-home messages

* Dynamic programming is an algorithmic paradigm that can be used to solve optimization problems in polynomial time for which brute-force approaches (e.g., recursion) may result in exponential time complexity
* It comes at a price: increased memory consumption
* Applicable to many different problems, but not always
* The optimization problem has to have (or should be recasted to have) the property that the optimal solution can be expressed as a combination of optimal solutions to **overlapping** subproblems