

COMP1730/COMP6730

Programming for Scientists

Control, part 3: Dynamic programming



Outline

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



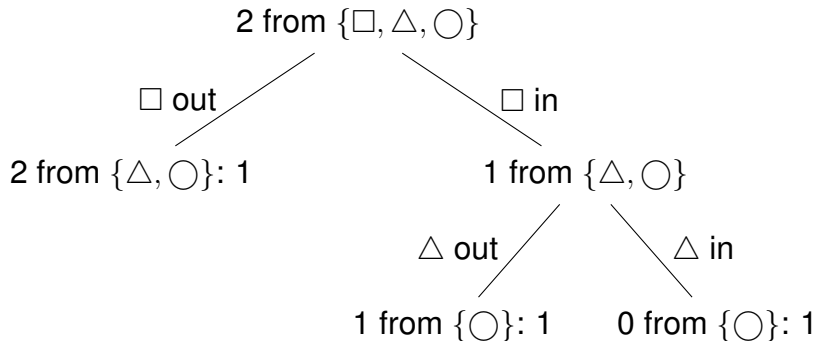
Dynamic programming

Recursion or iteration?

- * Examples of problems that could be solved both with recursion and with iteration:
 - Counting boxes in a stack.
 - Solving an equation (the interval-halving algorithm).
- * Examples of problems that we have only seen recursive solutions for:
 - Counting selections (“ n choose k ”).
 - The subset sum problem.

Example: Counting selections

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



- ★ Simple recursive formulation:

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

$$C(n, 0) = 1$$

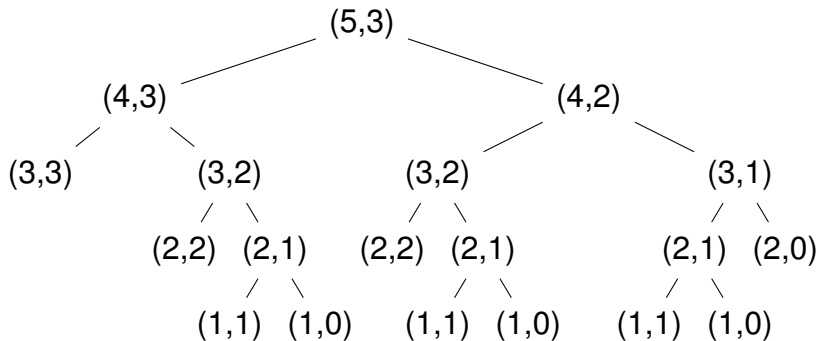
$$C(n, n) = 1$$

- ★ 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)
```

- ★ How to implement with iteration?

* 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.
- ★ 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;
 - until the whole problem is solved.
- ★ Need a way to index subproblems.

* Array of subproblems:

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

n

* With base cases solved:

k	$(0,0)$ = 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	$(4,3)$	$(5,3)$			
	n					

★ Complete:

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



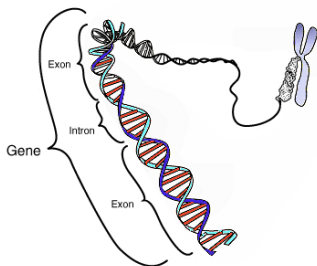
(DNA) sequence alignment

BRCA 1 gene (BReast CAncer)

CTTAGCGGTAGCCCTTGGTTTTCCGTGGCAACGGAAAAGCGCGGAATTACAGATAAAATAAACTGCGACTGCGCGGCGTGAGCTCGC
TGAGACTTCCTGGACGGGGACAGGCTGTGGGGTTTTCTAGATAACTGGGCCCTGCGCTCAGGAGGCCTCACCCTCTGCTCTGGTTC
ATTGGAAACGAAAAGAAATGGATTATCTGTCTCTCGCGTTGAAGAAGTACAAAATGTCATTAATGCTATGCGAAAAATCTTAGAGTGT
CCATCTGTCTGGAGTTGATCAAGGAACCTGTCTCCACAAAGTGTGACCACATATTTTGCAAATTTGCGATGCTGAAACTTCTCAACCAG
AAGAAAGGGCCTTCACAGTGTCTTTATGTAAAGATGATATAACCAAAAGGAGCCTACAAGAAAGTACGAGATTTAGTCAACTTGTGGA
AGAGCTATTGAAAATCATTGTGCTTTTCAGCTTGACACAGGTTTGAGATATGCAAACAGCTATAATTTTGCAAAAAAGGAAAAATACT
CTCCTGAACATCTAAAAGATGAAGTTTCTATCATCCAAAGTATGGGCTACAGAAACCGTGCCAAAAGACTTCTACAGAGTGAACCCGAA
AATCCTTCCTTGAAAACAGTCTCAGTGTCCAACCTCTTAACCTTGGAACGTGTGAGAAGTCTGAGGACAAAGCAGCGGATACAACCTCA
AAAGACGTCTGTCTACATTGAATTGGGATCTGATTCTTCTGAAGATACCGTTAATAAGGCAACTTATTGCAGTGTGGGAGATCAAGAAT
TGTTACAAATCACCCCTCAAGGAACCGGGATGAAATCAGTTTGGATTCTGCAAAAAAGGCTGCTGTGGAATTTTCTGAGACGGATGTA
ACAAATACTGAACATCATCAACCAGTAATAATGATTGAACACCCTGAGAAGCGTGCAGCTGAGAGGCATCCAGAAAAGTATCAGGG
TGAAGCAGCATCTGGGTGTGAGAGTGAACAAGCGTCTCTGAAGACTGCTCAGGGCTATCCTCTCAGAGTGCATTTAACCACTCAGC
AGAGGATACCATGCAACATAACCTGATAAAGCTCCAGCAGGAAATGGCTGAAC TAGAAGCTGTGTTAGAACAGCATGGGAGCCAGCCT
CTTAACAGTACCCTTCCATCATAAGTACTCTTCTGCCCTTGAGGACCTGCGAAATCCAGAACAAGCACATCAGAAAAAGCAGATT
AACTTCACAGAAAAGTAGTGAATACCCTATAAGCCAGAATCCAGAAGGCCCTTTCTGCTGACAAGTTTGGAGTGTCTGCAGATAGTTCTA
CCAGTAAAAATAAAGAACCAGGAGTGGAAAAGTTCATCCCTTCTAAATGCCATCATTAGATGATAGGTGGTACATGCACAGTTGCTCT
GGGAGTCTTCAGAATAGAAACTACCCATCTCAAGAGGAGCTCATTAAAGTTGTGTGATGTGGAGGAGCAACAGCTGGAAGAGTCTGGGCC
ACACGATTTGACGGAAACATCTTACTTGCCAAGGCAAGATCTAGAGGGAAACCCCTTACCTGGAATCTGGAATCAGCCTTCTCTGTAG
ACCCTGAATCTGATCCTTCTGAAGACAGAGCCAGAGTCAAGTGTGTTGGCAACATACCATCTCAACCTCTGCATTGAAAGTTCCC
CAATTGAAAGTTGCAGAACTGCCCCAGAGTCCAGCTGCTGCTCATACTACTGATACTGCTGGGTATAATGCAATGGAAGAAAAGTGTGAG
CAGGGAGAAGCCAGAATTGACAGCTTCAACAGAAAGGGTCAACAAAAGAAATGTCCATGGTGGTGTCTGGCCTGACCCAGAAGAATTTA
TGCTCGTGTACAAGTTTGCCAGAAAACACCACATCCTTTAACTAATCTAATTACTGAAGAGACTACTCATGTTGTTATGAAAACAGAT
GCTGAGTTTGTGTGTAACGGCACTGAAATATTTCTAGGAATTGCGGGAGGAAAATGGGTAGTTAGCTATTTCTGGGTGACCCAGTC
TATTAAGAAAAGAAAAATGCTGAATGAG

Biological sequence data

- * DNA and RNA.
- * Protein amino acid sequence.
- * Arrangement of genes in chromosome / genome.
- * Human DNA is ~ 3 billion bp.
- * BRCA 1 & 2 genes are ~ 80 kb (incl. exons).
- * Harmful mutations change as few as 2 bases.
- * DNA sequencer reads are 100–2k bases.

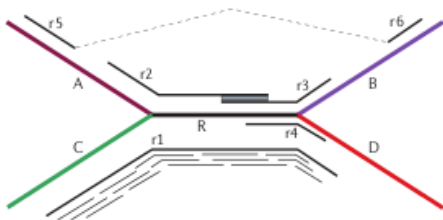


* Alignment

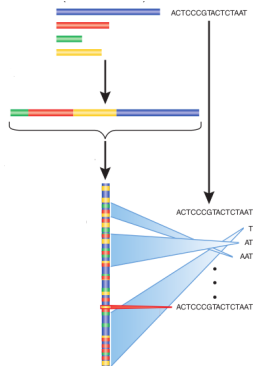
GAATTCAG	GAATTCAG
GGA-TC-G	GCAT-C-G

GAATTC-A	GAATTC-A
GGA-TCGA	GCAT-CGA

* Assembly



* Mapping





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.
- * Minimum edit equals best sequence alignment.

* distance(GAATTCA, GGATCGA) = 3.

* Edits:

		G	A	A	T	T	C	A
(subst. 1 G)	⇒	G	G	A	T	T	C	A
(del 4)	⇒	G	G	A	T	C	A	
(ins 5 G)	⇒	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

$$\text{dist}(s, ' ') = \text{len}(s) * w_{\text{gap}}$$

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

$$\text{dist}(s + x, t + y) =$$

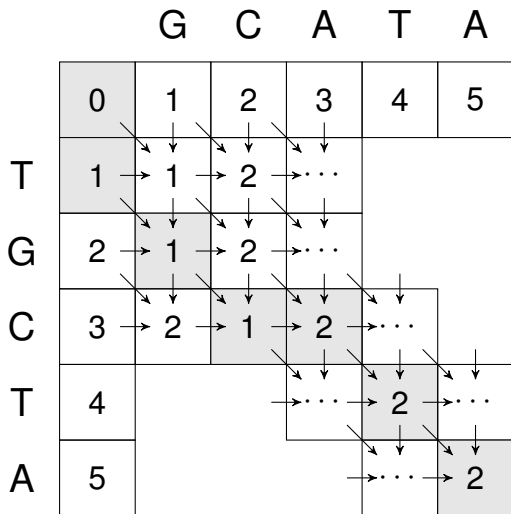
$$\min \begin{cases} \text{dist}(s, t) + \begin{cases} 0 & \text{if } x = y \\ w_{\text{sub}} & \text{otherwise} \end{cases} \\ \text{dist}(s + x, t) + w_{\text{gap}} \\ \text{dist}(s, t + y) + w_{\text{gap}} \end{cases}$$

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

```
def align(s, t):
    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)
```

Iterative 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 $s[i] \neq t[j]$), $(i - 1, j)$ plus gap weight, and $(i, j - 1)$ plus gap weight.



Summary

- * Recursion, iteration and dynamic programming are all useful algorithm design ideas.
- * There is no single “best” idea.
- * It is not always easy to know which is the right one to apply to a given problem.