Topic 7: Algorithms

Algorithms

An algorithm is a ”recipe” for solving a problem.

There are simple algorithms like the procedure for adding two numbers and long division that everyone learns in grade school

In computer science we are usually interested in problems that can ”grow” in size.

One simple problem is Search: we have a list of things and we want to see whether (yes or no) a certain thing is in the list.

Search Algorithm

If we use a Python type list, linear search means we have to look at each element in the list

```
key = 5
things = [ 3, -1, 85, 5, 23 ]
found = False
for t in things :
    if (t == key) :
        print( ”found ”, key, ” in list” )
        found = True
if (found == False) :
    print( ”Did not find ”, key, ” in list” )
```

Search Algorithm

Using the loop we just saw and assuming that there are N things in the list, how many times do we have to check if `t == key`?

What if the `key` is not in the list?

Linear search has a ”time complexity” of _____

Search Algorithm

Can we do better than a time complexity of N for search?

→ Binary search!

It's faster. Worst-case time complexity is **$\log_2(N)$** but the list has to be set up just right – the list has to be in **sorted** order.

Sort Algorithms

Sorting: arrange things in a list so that they are in some order

Example: Sort these 5 numbers → 6, 0, 10, -1, 7 in increasing order.

We can do this in our heads but a computer needs to be told exactly how to do it → an algorithm.

There are many algorithms for sorting.

Sort Algorithms

Sorting: arrange things in a list so that they are in some order

Usually implemented using nested loops:

```
for (i in list):
    for (j in list):
   ...
```

If the size of the list is N, the two loops do N^2 amount of work → Time complexity is **N^2**

Sort Algorithms

Can we do better than a time complexity of N^2

Yes! Quicksort usually needs N log(N) time

MergeSort needs N log(N) worst case

Sequence Alignment with gaps

```
GCCAT

+-o++
G_AAT
```

A + means a match
A o means a mismatch
A – is a gap _

What we want is the best - an ”optimal” - alignment

Can we go through all the possible alignments and pick the best one?

If each sequence has N bases, and gaps are not allowed, alignment takes cN^2 time steps (c is a constant) $\rightarrow$ time complexity of **N^2**

Optimal Alignment with gaps

Optimal alignment: If both sequences are N bases long, and gaps **are** allowed, the best techniques have a ”time complexity” of $\mathbf{N^{4N}}$.

This means that as N becomes large, the time needed to find the best alignment becomes very large very quickly!

The optimal sequence alignment problem is one of those problems that are hard to solve – so hard that we ”abandon” our quest.

"Acceptable" Alignment with gaps

The optimal sequence alignment problem is hard: a ”brute-force” method will take too long for large sequence lengths.

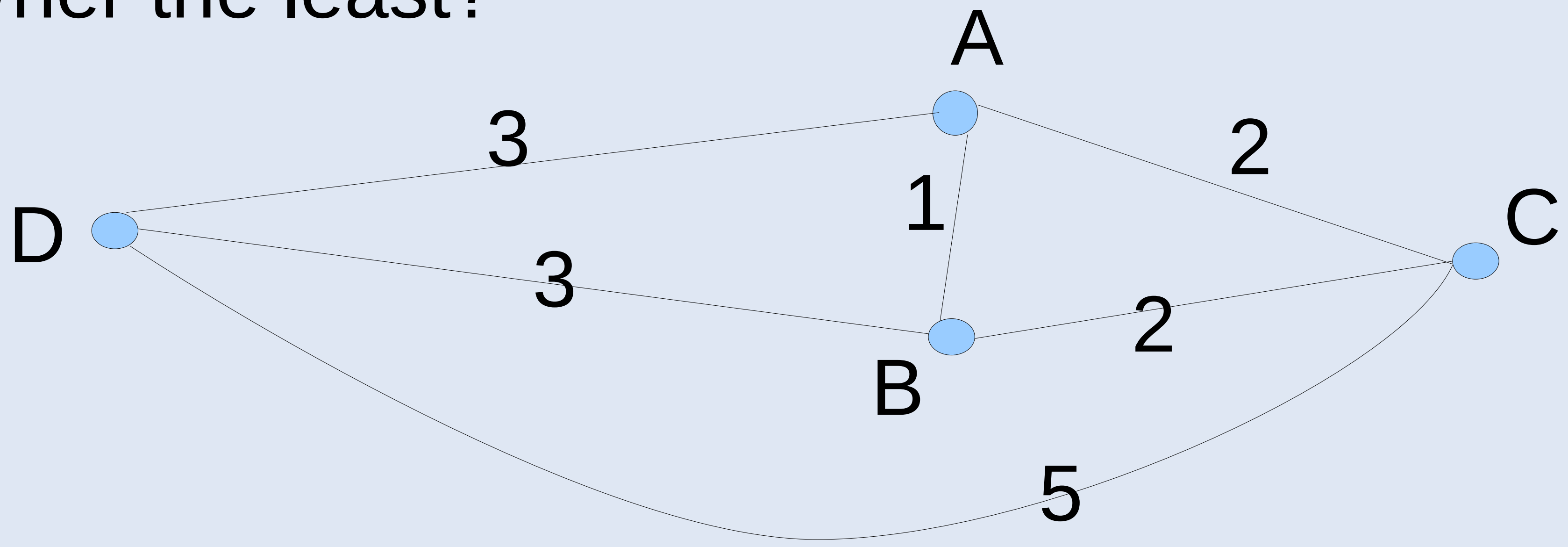
Instead, we use methods that can give us reasonable alignments but not necessarily the ”optimal” one.

There are many such ”heuristics” in computer science.

Traveling Salesman

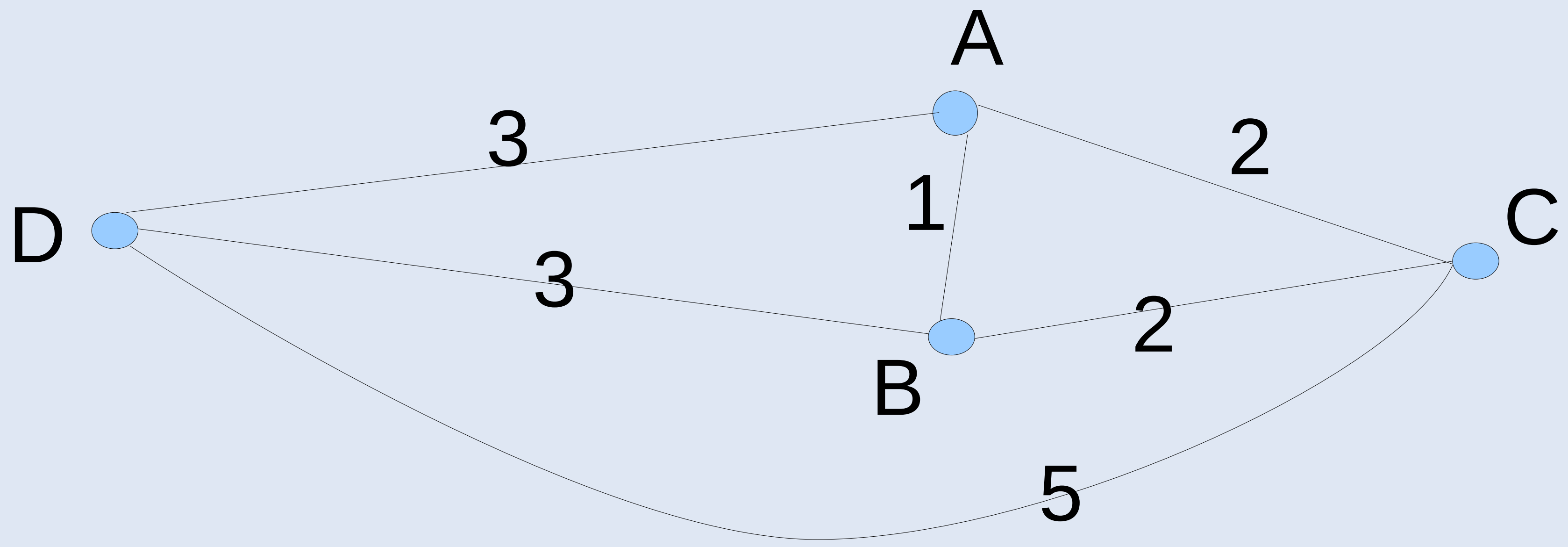
Another problem that is hard to solve is the "Traveling Salesman" problem:

A salesperson wants to visit a number of cities, returning to the starting city having been to each of the other cities only once; what route will cost him/her the least?



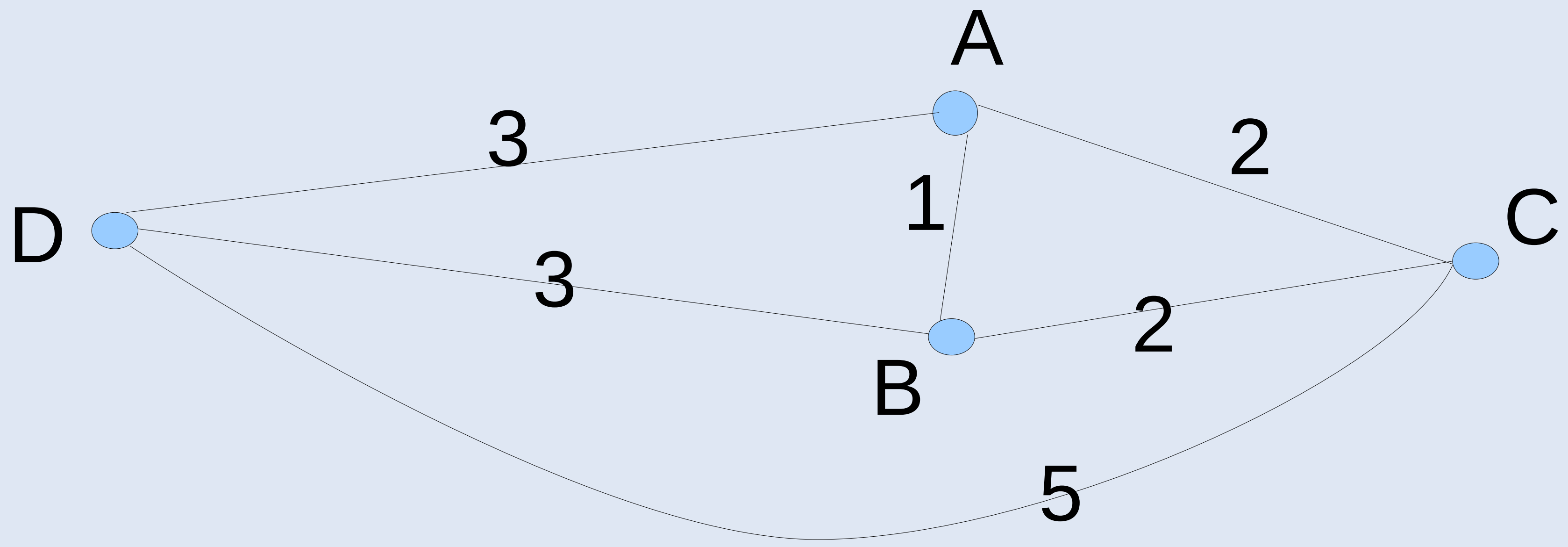
Traveling Salesman

Finding the best route by brute force has a time complexity of N*(N-1)*(N-2) … 3*2*1 which is larger than 2^N



Traveling Salesman

Try: start with a city, find the nearest city, and then repeatedly visit the closest next city that we have not yet seen, ending back in the starting city



Traveling Salesman

Start with at A, find the nearest city → B; cost = 1

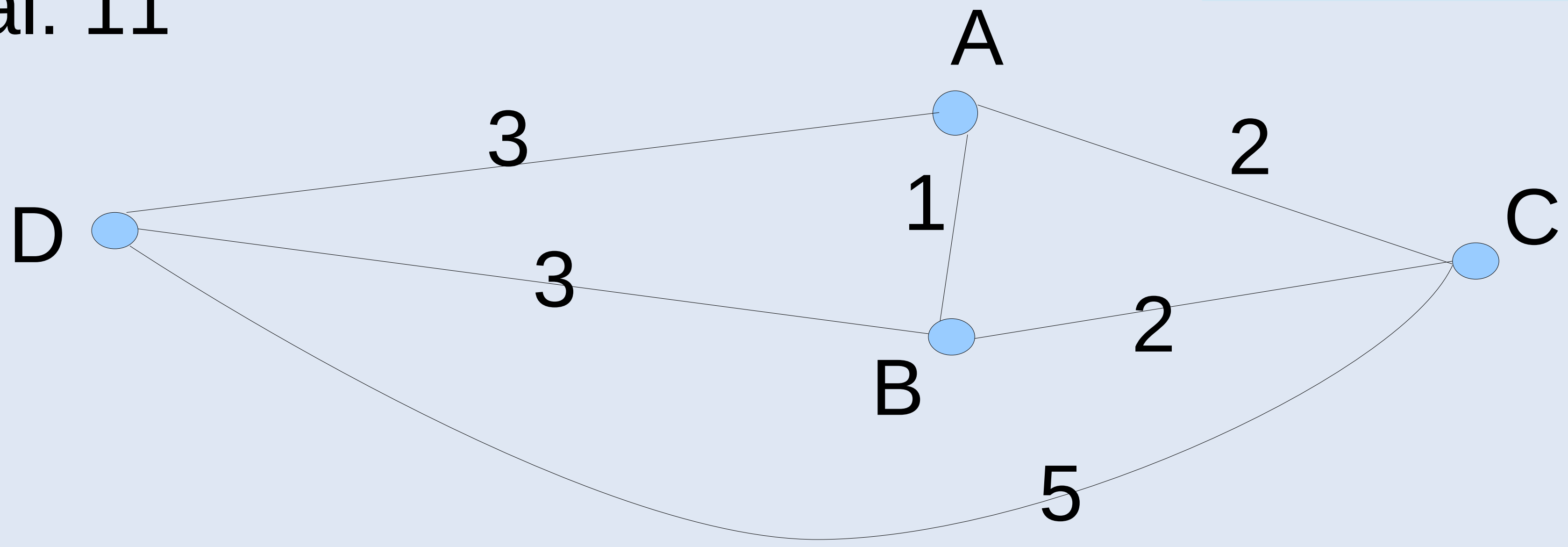
closest next city → C; cost = 2

closest next one → D; cost = 5

And back to A; cost = 3

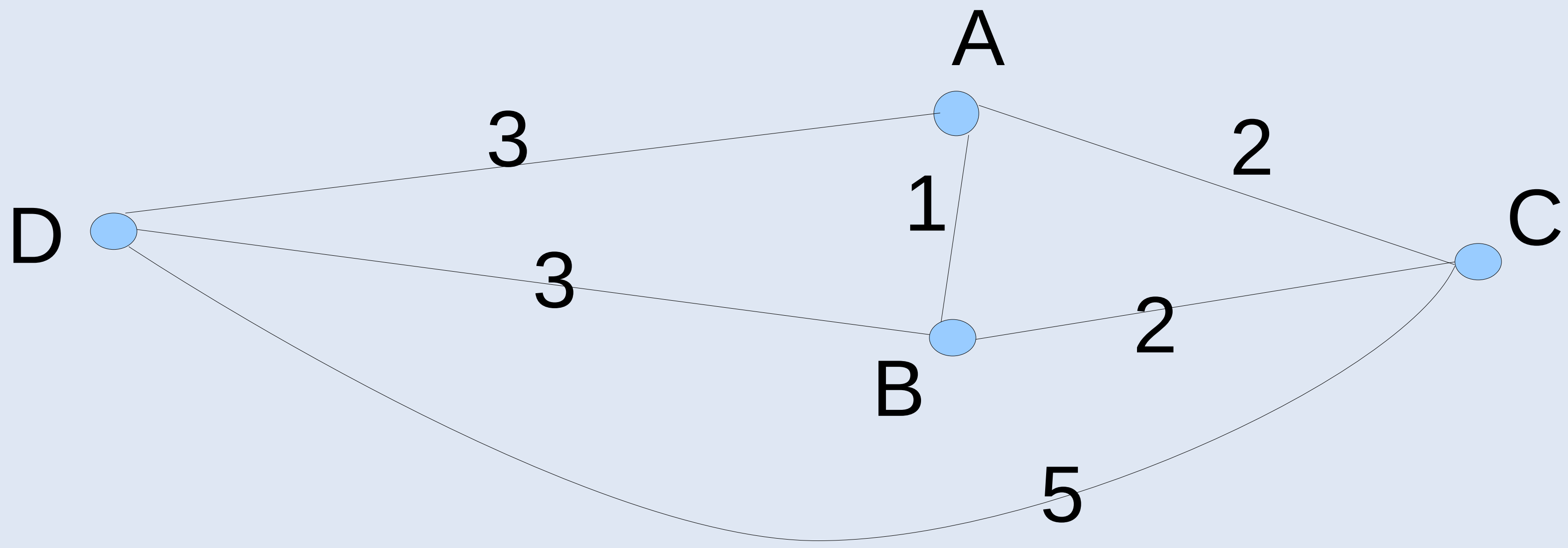
Total: 11

”Greedy” method:
take the lowest-cost
next step



Optimal Solution Traveling Salesman

Start from A, go to D; cost = 3
next city → B; cost = 3
next one → C; cost = 2
And back to A; cost = 2
Total: 10



Compare: "Greedy" method: take the lowest-cost next step
Downfall of greedy approach: mistakes made early on cannot be corrected later.

"Acceptable" Alignment with gaps

We want a reasonably good alignment – may not be the optimal one

Dynamic Programming Strategy: Try to find an optimal solution by by finding optimal solutions to smaller subproblems.

i.e. break a big problem into a bunch of smaller problems and ”remember” the best previous solutions.

Sequence Alignment Problem

Suppose we have the sequences

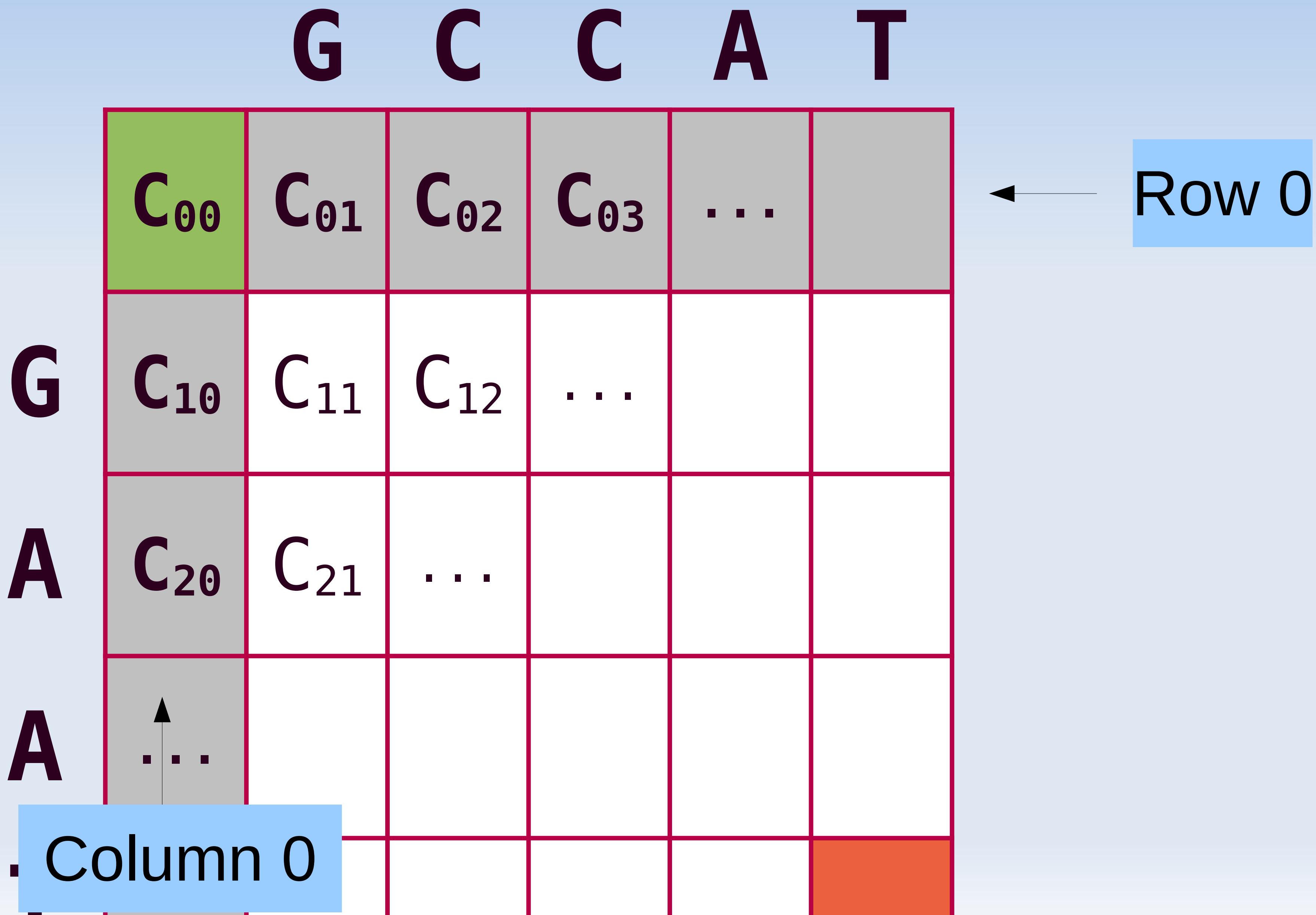
```
GCCAT

GAAT
```

We want to find a ”good” alignment of these two sequences using a **D**ynamic **P**rogramming (DP) algorithm

DP Path algorithm Step 1

Set up the matrix



$$
\begin{array}{c|c|c|c|c|c|c}
 & & G & C & C & A & T \\
\hline
 & C_{00} & C_{01} & C_{02} & C_{03} & \cdots & \\
G & C_{10} & C_{11} & C_{12} & \cdots & & \\
A & C_{20} & C_{21} & \cdots & & & \\
A & \cdots & & & & & \\
 & & & & & & \\
\end{array}
$$

DP Path algorithm Step 3

Set row 1 (C_{1j}) elements from left to right:

to the max of C_{0j}- 1, C_{1j-1}- 1, or C_{0j-1}+ (1 if C_{1j} = match)

For C_{11}, the

3 values are

C_{01}- 1 = -1

C_{10}- 1 = -1

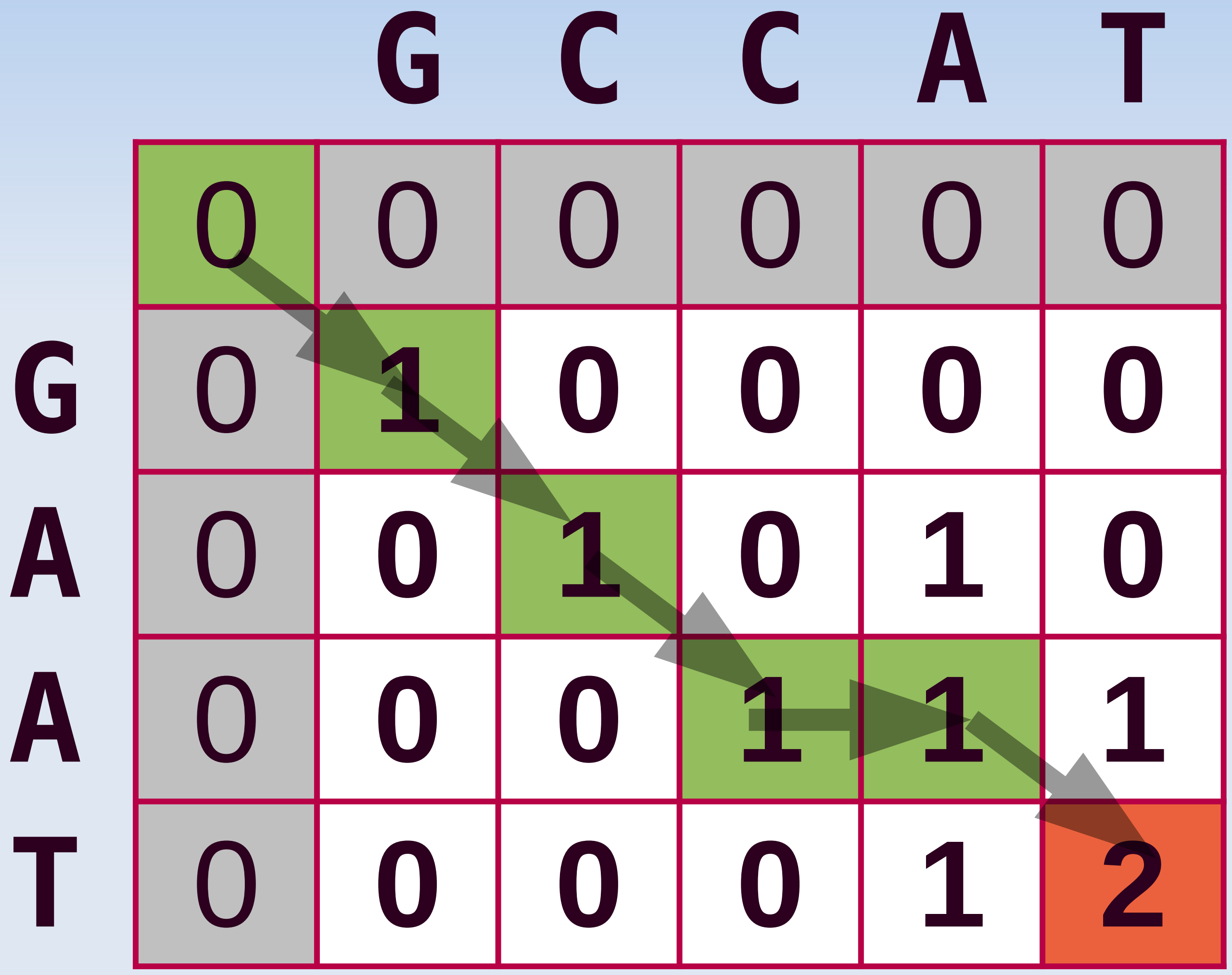
C_{00}+ 1 = 1

		G	C	C	A	T
	0	0	0	0	0	0
G	0	C_{11}	C_{12}	∵		
A	0					
A	0					
T	0					

DP Path algorithm - Align

Resulting path gives us this alignment:

```
GCCAT
+oo-+
GAA_T
```



		G	C	C	A	T
	0	0	0	0	0	0
G	0	1	0	0	0	0
A	0	0	1	0	1	0
A	0	0	0	1	1	1
T	0	0	0	0	1	2

Probably not optimal: one gap, two mismatches

BLAST: approximate DP Path

We saw that the Dynamic Programming approach in the "Smith-Waterman" algorithm is a short cut.

SW is expensive. Main problem is that "matrix". For two n-length sequences, we need n^2 time to set up a matrix and then more time to find the path.

BLAST and FASTA use "k-mers" – short k-length subsequences – and hash tables to find these k-mers in a larger sequence. Hash tables make search fast!

→ BLAST uses short cuts to solve a DP Path (which is already a short cut!)