The
CRIMSON
PERMANENT
ASSURANCE

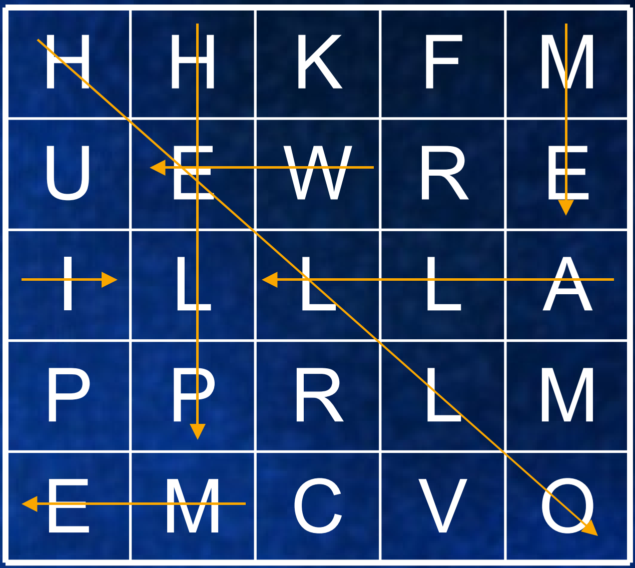
(search)

The problem

- Given a grid of letters and a list of words:
  - Find the words in the grid (either vertically, horizontally or diagonally)
    - Call all the places where we find words *placements*
  - Choose the subset of these placements such that:
    - No two placements overlap
    - The *score* of the subset of placements is as large as possible
      - Score is equal to the number of placements + the length of all the placements

Example

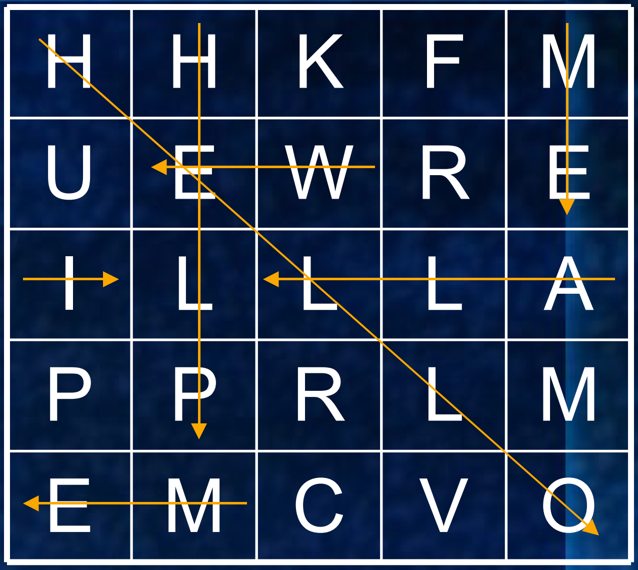
- HELLO
- ME
- ALL
- I
- HELP
- WE



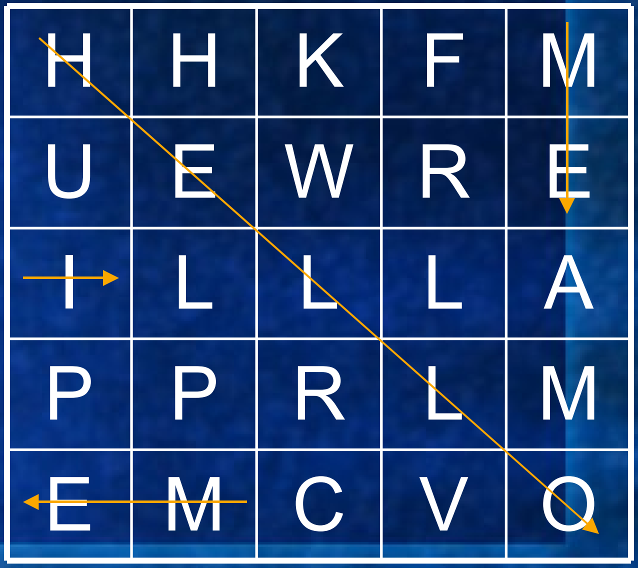
H	H	K	F	M
U	E	W	R	E
I	L	L	L	A
P	P	R	L	M
E	M	C	V	Q

The greedy approach

- The scoring formula favoured longer placements
  - Use the longest legal placement, and repeat until there are no more legal placements



H	H	K	F	M
U	E	W	R	E
I	L	L	L	A
P	P	R	L	M
E	M	C	V	O

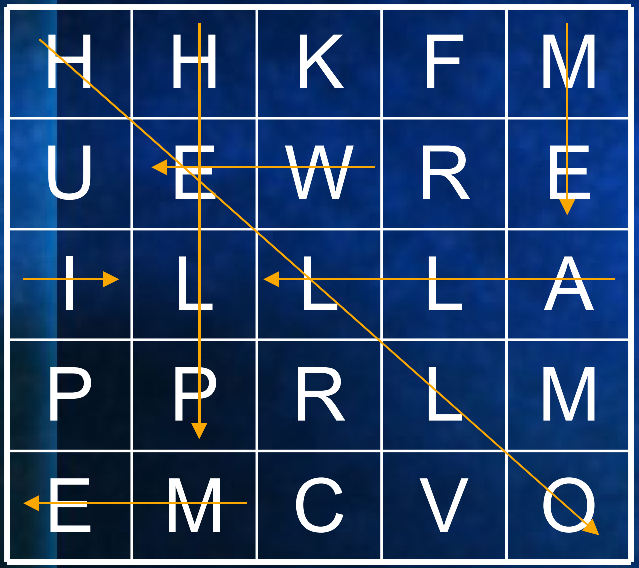


H	H	K	F	M
U	E	W	R	E
I	L	L	L	A
P	P	R	L	M
E	M	C	V	O

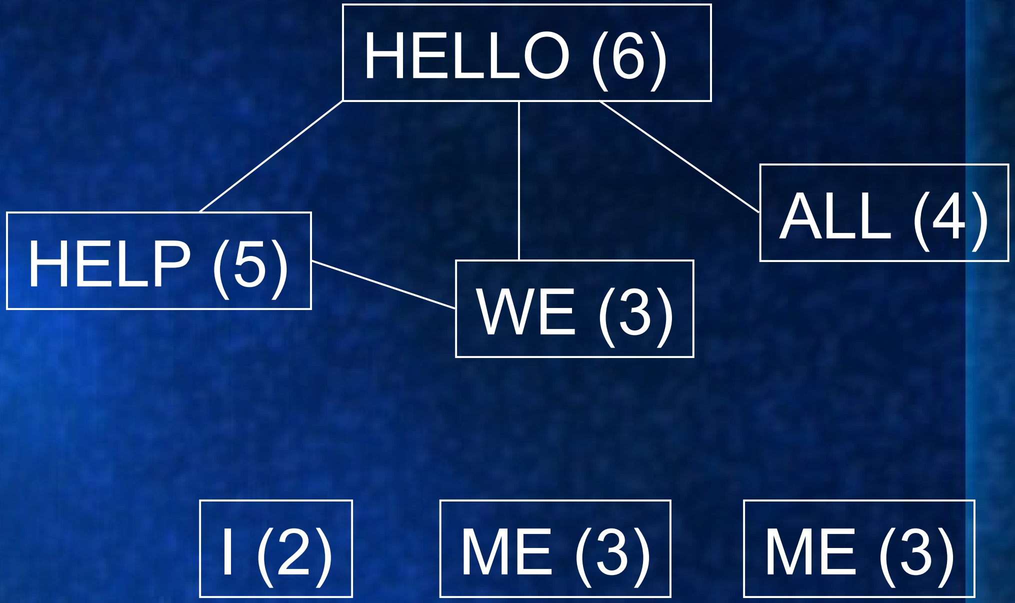
I have a cunning plan…

- Turn the set of placements into a *graph*
  - Create a node for each placement
  - Create an edge between nodes if their placements *overlap*

Converting to a graph



H	H	K	F	M
U	E	W	R	E
I	L	L	L	A
P	P	R	L	M
E	M	C	V	Q



HELLO (6)

HELP (5)

WE (3)

ALL (4)

I (2)

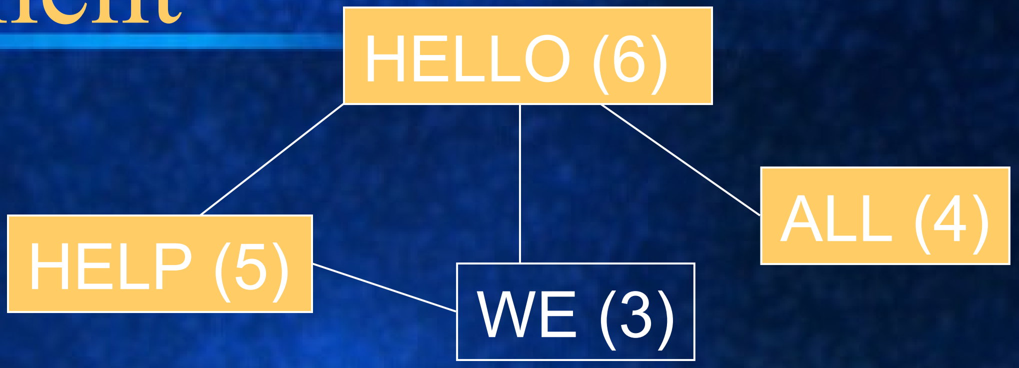
ME (3)

ME (3)

Using the graph

- Use every node that is not connected to anything else
  - These nodes correspond to placements that do not overlap with any other placements
    - We get them for free! :-)
- For the remaining nodes, split them up into what are called *connected components*
  - Each connected component corresponds to a smaller sub-problem

Solving within each connected component



- Imagine that we highlight nodes in the graph to indicate that we wish to use those nodes’ placements
- Our goal is then to highlight a subset of highlighted nodes such that:
  - No two highlighted nodes have an edge connecting them
  - The total score of the highlighted nodes is as large as possible
    - If each node had a score of 1, then this is known as the *maximum independent set problem* (which is NP-complete)

Solving within each connected component (2)

- Brute force
  - Give your placements some order
  - For the first placement, you try two options: either you use the placement, or you don’t
    - For the second placement:
      - If it conflicts with the first one (their nodes are connected by an edge), then you can’t use it — move onto the third placement
      - No conflict, so again you have two options: use it or lose it! Try each, and then…
        - For the third placement…

Carl’s quick intro to recursion

- Typing out all those different options is going to take a ***long*** time. There must be an easier way…

```
Function solve(n)
        if n = end
                calculate_score()
                if score > best_score
                        update_best_score()
                return
        if can_use(n)
                use(n)
                solve(n+1)
        don’t_use(n)
        solve(n+1)
```

Some other thoughts

- Look for chains
  - E.g. A connected to B connected to C connected to D connected to…
  - These can be solved using *dynamic programming* (DP)
- Look for loops
  - These can be dealt with in a similar way to chains, using DP
- Look for trees
  - Trees allow you to break up the problem into smaller problems, by considering each branch individually

Or just do it by hand ;-)

Questions, comments, death threats, large sums of money? ;-P