Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Dynamic Programming II

COMP4128 Programming Challenges

School of Computer Science and Engineering
UNSW Australia

Table of Contents

1 Finding DP solutions
- Tickets

2 DP on a tree

3 Example: Cutting Sticks
- Knuth's Optimisation

4 More example problems
- Longest Increasing Subsequence
- Rooks

Finding DP solutions

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- DP solutions are usually easy to understand once they are given to you, but coming up with them can be quite hard.
- In this section we will solve a few problems in different ways to give you an idea of how to approach DP problems.

Example Problem: Tickets

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Problem Statement** Jane has come to visit Australia, and soon finds out that the public transport is ridiculously expensive, meaning that she will actually have to plan her ticket purchases.
  There are two types of tickets in Australia. The first one costs C_1 thousand dollars and is valid for D_1 consecutive days. The second one costs C_2 thousand dollars and is valid for D_2 consecutive days.
  Jane knows in advance on which days she will be travelling, and wishes to plan which tickets she will need to buy and when, in order to minimise the total cost of her travel.
- **Constraints**
  - Jane will be in Australia for at most $100,000$ days.
  - Jane will travel on $N \leq 10,000$ different days.
  - $1 \leq C_1, C_2 \leq 1,000$ and $1 \leq D_1, D_2 \leq 100$.

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example Problem: Tickets

- **Example Input**
  - $C_1 = 4,\ D_1 = 3$
  - $C_2 = 7,\ D_2 = 5$
  - $N = 7$
  - Jane travels on days 1, 2, 4, 6, 8, 13, and 16.
- **Example Output** 18
- **Questions to Ask** to reduce the number of possibilities we have to think about:
  - When do we buy tickets?
  - Would we ever buy a ticket on a day when we are not travelling?
  - Would we ever buy a ticket before the one we already have has expired?

Example Problem: Tickets

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Start from the start**: When do we buy our *first* ticket? On the first day that we travel. There's no point buying it earlier, and we can't buy it later.
- **Decisions to be made**: Do we know which ticket to buy? No, we don't.
- **When you don't know what to choose**: Try both! This is the essence of Dynamic Programming.
- **What do we do after deciding?** Suppose we chose to buy ticket 1. What is the next day we will need a ticket? We will need to buy another ticket on the next day that we travel that is not covered by the ticket we just bought. *There's no point buying it earlier, and we can't buy it later.*
- **What information are we looking for?** Just the total minimum money we can spend on the tickets.

Example Problem: Tickets

- Try to formulate a state. **What 'situation' are we in every time we have to make a decision?**
  - It is a day on which we have to travel
  - We don't have a ticket
- Find a recurrence.
  1. **What do we have to do?** Buy a ticket.
  2. **Is there a decision to be made?** Which ticket to buy.
  3. **What information will we obtain after we decide?** We will know *the minimum amount of money we can spend between today and the end of our trip*.
  4. **What information do we need to make this decision?** We need to know the minimum amount of money we can spend between today and the end of our trip, for each outcome of the decision.
  5. **Can we get this information by asking the same question we are trying to answer right now**? Yes, we can ask this same question for the next day not covered by the ticket we buy, for both possibilities.

Example Problem: Tickets

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Summarise**. Write some maths to see what's going on.
- **State**: Recall that our state is simply "it is a day on which we travel and we don't have a ticket". What information is in this state?
  The state only depends on the day it is. The other information simply provides guarantees.
  Let our state be $f(i)$, the answer to our question.
- **The meaning of a state**: What question are we answering?
  Let $f(i) =$ the minimum amount of money we can spend on tickets between day i and our last day.
- **Recurrence**: How can we answer this question by asking more questions?

$$f(i) = \min \begin{cases} f(\text{first travel day after } D_1 \text{ days}) + C_1 \\ f(\text{first travel day after } D_2 \text{ days}) + C_2 \end{cases}$$

Example Problem: Tickets

Dynamic Programming II
Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- Don't forget the base cases!
- Every state for this problem represents a day on which we travel and don't have a ticket, and the recurrence finds the next such day.
- What's the easiest base case for this problem?
- *When we have run out of days.* Create a $n + 1st$ travel day occurring on day $101,000$ so that no ticket from an actual can cover it.
  Then, just return 0 for the cost.
- **General Trick** In general, trying to create "correct" base cases from the last actual item you consider results in more case bash, and more things to get wrong.
  If you instead create a fake item past the end that all legitimate decision sequences will arrive at, you can often return a simple constant value there and be done.

Example Problem: Tickets

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Implementation**

```
#include <iostream>

using namespace std;

int c1, d1, c2, d2, n, t[10001];

int next(int i, int d) {
    int to = t[i] + d;
    while (t[i] < to) ++i;
    return i;
}

int cache[10000];

int f(int i) {
    if (i == n) return 0;
    if (cache[i]) return cache[i];
    return cache[i] = min(f(next(i, d1)) + c1, f(next(i, d2)) + c2);
}

int main() {
    cin >> c1 >> d1 >> c2 >> d2 >> n;
    for (int i = 0; i < n; ++i) cin >> t[i];
    t[n] = 101000;
    cout << f(0) << '\n';
}
```

Table of Contents

1 Finding DP solutions
- Tickets

2 DP on a tree

3 Example: Cutting Sticks
- Knuth's Optimisation

4 More example problems
- Longest Increasing Subsequence
- Rooks

Example problem: Weighted MIS of a tree

Dynamic Programming II

Finding DP solutions

Tickets

DP on a tree

Example: Cutting Sticks

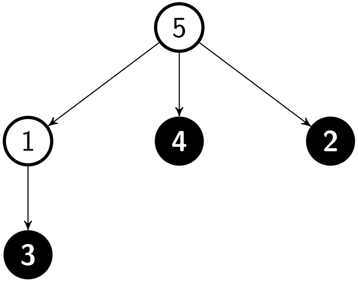
Knuth's Optimisation

More example problems

Longest Increasing Subsequence

Rooks

- **Problem statement** Given a description of a tree T ($1 \leq |T| \leq 1,000,000$), with vertex weights, what is the maximum sum of vertex weights of an independent subset of the tree? An independent set is a subset of the nodes of the tree, with the property that no two nodes in the subset share an edge.
- **Example**



Example problem: Weighted MIS of a tree

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- How do we define the states? We need to define them in such a way that there exists an easy way to solve our subproblems in the right order.
- To do this, we add some more structure to the tree that will not affect the answer: we *root* the tree somewhere (anywhere). In this way, we only need to consider subtrees pointing away from the root.
  - We only need to do this because the tree is given simply as a listed of undirected edges. The diagram on the previous slide has arrows for clarity (after we've already chosen a root).
- This is usually called "DP on a tree".

Example problem: Weighted MIS of a tree

Dynamic Programming II

Finding DP solutions

Tickets

DP on a tree

Example: Cutting Sticks

Knuth's Optimisation

More example problems

Longest Increasing Subsequence

Rooks

- **Subproblems** For each subtree R rooted at some vertex u of our tree T, what is the maximum weighted independent set contained in R?
- There are two cases: either the root of R is in the independent set, or it is not. We define two functions:
  - Let $f(u)$ be the size of the maximum weighted independent set in the subtree rooted at u that contains u.
  - Let $g(u)$ be the size of the maximum weighted independent set in the subtree rooted at u that does not contain u.

Example problem: Weighted MIS of a tree

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Recurrence** If we are considering an independent set that contains the root, then we must not take any of its children. However, if we don't have the root, then it doesn't matter whether we take the children or not. Hence, denoting the set of children of u by $N(u)$, we have:

$$f(u) = w_u + \sum_{v \in N(u)} g(v)$$
$$g(u) = \sum_{v \in N(u)} \max(f(v), g(v)).$$

- **Base case** If u is a leaf, then $f(u) = w_u$ and $g(u) = 0$.

Example problem: Weighted MIS of a tree

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Complexity** Since we have $O(n)$ values of f and g to calculate, each taking $O(n)$ time to calculate, it seems that the overall complexity is $O(n^2)$.
- However, if we observe that each vertex v only appears on the right hand side once (when u is its parent), it can be seen that the overall complexity is in fact only $O(n)$.
- **Implementation**

```
void calculate_wmis (int u) {
  f[u] = w[u];
  g[u] = 0;
  for (int i = 0; i < children[u].size(); i++) {
    int v = children[u][i];
    calculate_wmis(v);
    f[u] += g[v];
    g[u] += max(f[v],g[v]);
  }
}
```

Table of Contents

1 Finding DP solutions
- Tickets

2 DP on a tree

3 Example: Cutting Sticks
- Knuth's Optimisation

4 More example problems
- Longest Increasing Subsequence
- Rooks

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Cutting Sticks

- **Problem statement** The good King Elgar has decreed that the trunk of the mythical sword-tree Yggdrasill (which is $1 \leq L \leq 1,000,000$) metres long) be broken at exactly $1 \leq N \leq 200$ places along its length. Unfortunately, due to the sword-tree's incredible power, breaking a piece of the sword-tree of length X requires X units of magic power. Given the length L of the sword-tree and the N points where the sword-tree needs to be broken, what is the minimum number of units of magic power needed?

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Cutting Sticks

- **Example** With $L = 20$ and the cuts needing to be at positions 3, 8 and 10, the minimum number of magic units is 37.
- The first cut must take 20 units, and should be at position 10. The next cut should take 10 units, and be at position 3. The last cut that needs to be made will take 7 units, for a total of 37.

Example problem: Cutting Sticks

- We can imagine a recursive process which tries to figure out the optimal first break, by trying them all.
- If we make our first break at the ith breaking point, say at position p_i, then what will be the total cost to make all the breaks from here?
- It's L + the cost to break everything to the left of p_i + the cost to break everything to the right of p_i.

Example problem: Cutting Sticks

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Subproblems** Label the given breaking points as $p_1, p_2, \ldots, p_N$, and set $p_0 = 0$ and $p_{N+1} = L$. Let $f(i, j)$ be the minimum magic power needed to break at all the breaking points between the ith breaking point and the jth breaking point, inclusive.
- **Recurrence**

$$f(i,j) = p_j - p_i + \min_{i<k<j} (f(i,k) + f(k,j))$$

- **Base case** Whenever $j = i + 1$, then we have a single segment, so the answer is zero.

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Cutting Sticks

- **Complexity** We have $O(n^2)$ values of $f(i,j)$. To compute each value, we need to iterate over all k between i and j, of which there can be up to $O(n)$. Therefore the time complexity is $O(n^3)$ and the space complexity is $O(n^2)$.

Example problem: Cutting Sticks

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Implementation**

```
// shorter intervals should be processed before longer intervals
// length is the number of breaking points strictly between i and j
for (int length = 0; length <= n; length++) {
  for (int i = 0, j = length + 1; j <= n + 1; i++, j++) {
    // base case
    if (length == 0) f[i][j] = 0;
    else {
      f[i][j] = INF;
      for (int k = i+1; k < j; k++) {
        // cost returns p_j - p_i, the cost of cutting this segment
        f[i][j] = min(f[i][j], f[i][k] + f[k][j] + cost(i, j));
      }
    }
  }
}
// answer is f[0][n+1]
```

Example problem: Cutting Sticks

- Let's try to optimise.
- Let the optimal position for cutting a segment (i, j) be $A(i, j)$. That is, $f(i, j) = p_j - p_i + f(i, A(i, j)) + f(A(i, j), j)$.
- Now let's consider $A(i, j)$. How does it relate to $A(i, j-1)$ and $A(i+1, j)$?
- We can see an obvious relationship: $A(i, j-1) \leq A(i, j) \leq A(i+1, j)$.
- Since we've already found $f(i, j-1)$ and $f(i+1, j)$ by the time we get to $f(i, j)$, we can restrict the range we need to search in.
- The next state we will process is $(i+1, j+1)$, for which we have $A(i+1, j) \leq A(i+1, j+1) \leq A(i+2, j+1)$.
- Notice that this range doesn't overlap with the previous one. So to compute *all* states of a certain length, we only look at $O(n)$ other states.

Knuth's Optimisation

- **Improved Complexity** We still have $O(n^2)$ states to calculate, but now for each of $O(n)$ lengths, we only perform $O(n)$ work total to calculate all states of that length. This reduces the total complexity to $O(n^2)$ from $O(n^3)$.
- The optimisation we just applied is **Knuth's Optimisation**.
- In general, when you have a DP of the form $dp[i][j] = \min_{i<k<j}(dp[i][k] + dp[k][j]) + C[i][j]$ and $A[i, j-1] \leq A[i, j] \leq A[i+1, j]$, then this optimisation can be applied as above.
- Additionally, both of these two conditions being true imply the inequality above is satisfied:
  - $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ for $a \leq b \leq c \leq d$
  - $C[b][c] \leq C[a][d]$ for $a \leq b \leq c \leq d$

Table of Contents

1 Finding DP solutions
- Tickets

2 DP on a tree

3 Example: Cutting Sticks
- Knuth's Optimisation

4 More example problems
- Longest Increasing Subsequence
- Rooks

Example problem: LIS

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Problem statement** Given N ($1 \leq N \leq 3,000$) integers in a sequence S, what is the longest subsequence of S that is strictly increasing? If there are multiple solutions, print any one of them.
- **Example** For the sequence [1, 9, 5, 8, 7], the longest increasing subsequence has length 3. There are multiple solutions, such as [1, 5, 7].

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: LIS

- On this sort of sequence, the only way to define our state is to use a position in the sequence.
- We can ask the question, "What's the longest increasing subsequence that we can obtain, finishing at the ith element of the sequence?"
- The recurrence is also straightforward: we can try every previous element in our sequence as the last element of our subsequence to be included, and then take the best one.
- This leads to an $O(n^2)$ time, $O(n)$ space algorithm.

Example problem: LIS

Dynamic Programming II
Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- We aren't done yet!
- We need to find not only the length of a longest increasing subsequence, but we need to find the subsequence itself.
- How do we recover an actual answer from our dynamic programming algorithm?
- Each state represents an increasing subsequence, which we constructed by extending another one, also represented by a state. Then all we need to do is store the index of the state we came from.
- Then to build an optimal solution, we can just backtrack from some terminal state through the optimal move we've stored until we reach our initial state.

Example problem: LIS

- **Implementation (DP, $O(n^2)$)**

```
int best = 0;
for (int i = 1; i <= n; i++) {
  dp[i] = 1;
  for (int j = 1; j < i; j++) {
    // try j as the penultimate index
    if (s[j] < s[i] && dp[j] + 1 > dp[i]) {
      dp[i] = dp[j] + 1;
      from[i] = j;
    }
    // update answer
    if (dp[best] < dp[i])
      best = i;
  }
}
```

- **Implementation (solution building)**

```
vector<int> ans;
int u = best;
while (from[u]) {
  ans.push_back(s[u].second);
  u = from[u];
}
ans.push_back(s[u].second);
reverse(ans.begin(), ans.end());
```

Example problem: LIS

Dynamic Programming II

Finding DP solutions
Tickets

DP on a tree

Example: Cutting Sticks
Knuth's Optimisation

More example problems
Longest Increasing Subsequence
Rooks

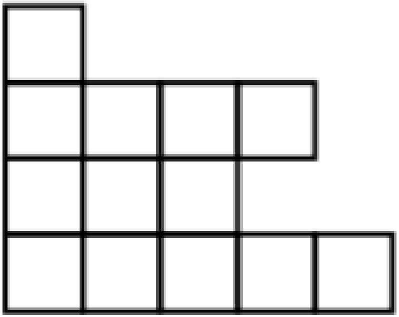
- This can be improved to $O(n \log n)$.
  - As i iterates through the sequence, maintain a sequence $M[j]$, the index of the minimum terminal value of an increasing subsequence of length j among the first i terms of S.
  - It can be seen that the elements $S[M[j]]$ always form an increasing subsequence.
  - To process a new element $S[i+1]$, we augment the longest of these subsequences whose terminal value is less than $S[i+1]$.
    That is, we find the largest j where $M[j] < S[i+1]$. This gives us the longest subsequence so far, so we extend that by updating $M[j+1]$.
  - Rather than trying all possibilities in $O(n)$, we can perform a binary search on the $S[M[j]]$.

Example problem: Rooks

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

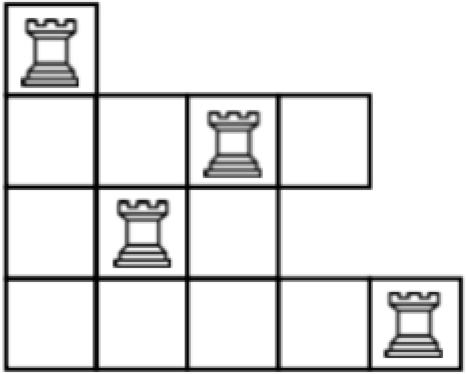
- **Problem statement** You have a set of N $(1 \leq N \leq 30)$ rows from a chessboard of N rows and M columns, with some of the squares cut out from the right. How many ways are there to place K rooks on this chessboard without any rook threatening any other rook?



Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Rooks



$N = 4, K = 4$

8 ways

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Rooks

- Rooks can threaten each other if they're on the same row or same column, even if there are rows between them where the board is missing
- Immediately, it looks like we can use one of these dimensions as part of our state, then compute our answer row by row, knowing that we can only put one rook down per row
- We also need to store how many rooks we still need to put down in our state

Example problem: Rooks

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- But how do we know which columns are free?
- We have no choice but to actually keep in our state exactly which columns are free, using a bit set
- But this is slow and somewhat difficult to reason about

Example problem: Rooks

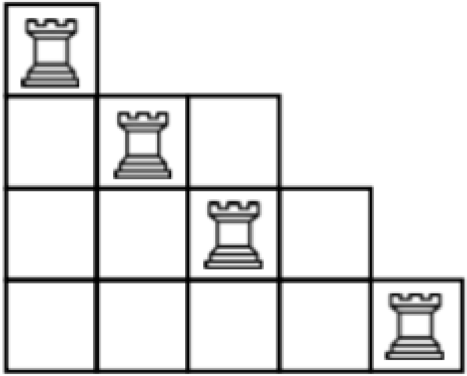
Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- Can we do something better?
- There doesn't seem to be an immediate way to optimise the DP state or recurrence, but what if we change the problem?
- The key thing is to notice that if we rearrange the order of the rows, we don't change the answer

Example problem: Rooks

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks



$N = 4, K = 4$
8 ways

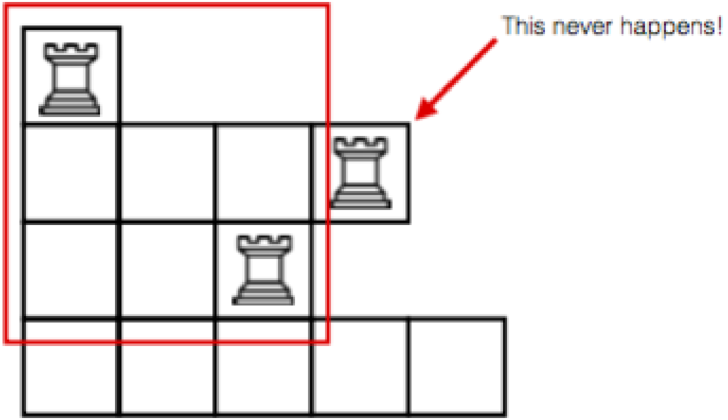
Example problem: Rooks

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- Let's sort the sequence, so that the size of each row is non-decreasing
- Now, we know that if we place a rook on a row, we know that we can assume that every previously placed rook is in a cell that our row covers

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

Example problem: Rooks



Example problem: Rooks

- Now, we can say that any rooks already placed will be either to the left or directly above us
- We can then formulate a recurrence that only needs to know about the current row and the number of rooks
- If we're on a row i on length L, then for every configuration of the rows above with k rooks already placed, we can place a rook for this row in $(L - k)$ places

Example problem: Rooks

Dynamic Programming II

Finding DP solutions
Tickets
DP on a tree
Example: Cutting Sticks
Knuth's Optimisation
More example problems
Longest Increasing Subsequence
Rooks

- **Subproblems** Let $f(i, j)$ be the number of ways to place j rooks on the first i rows, sorted by length.
- **Recurrence**

$$f(i,j) = f(i-1,j) + f(i-1,j-1) * (L_i - (j-1))$$

- **Base case** The number of ways to place 0 rooks on 0 rows is 1.

Example problem: Rooks

- **Implementation**

```
cache[0][0] = 1;
for (int i = 1; i <= n; i++) {
  for (int j = 0; j <= k; j++) {
    // can place no rooks in this row
    cache[i][j] = cache[i-1][j];
    // or place a rook in this row
    if (j > 0)
      cache[i][j] += cache[i-1][j-1] * (L[i-1] - (j-1))
  }
}
cout << cache[n][k] << endl;
```