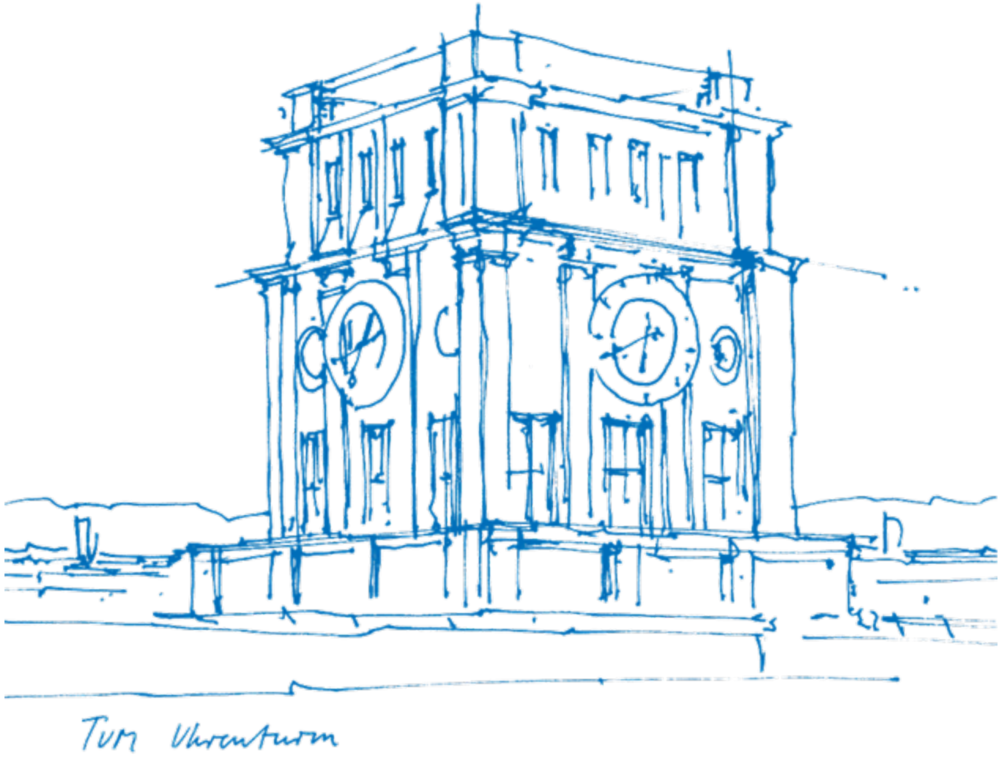
AACPP WiSe 2025/26

Class 1: Dynamic Programming

Mateusz Gienieczko

School of Computation, Information and Technology
Technical University of Munich

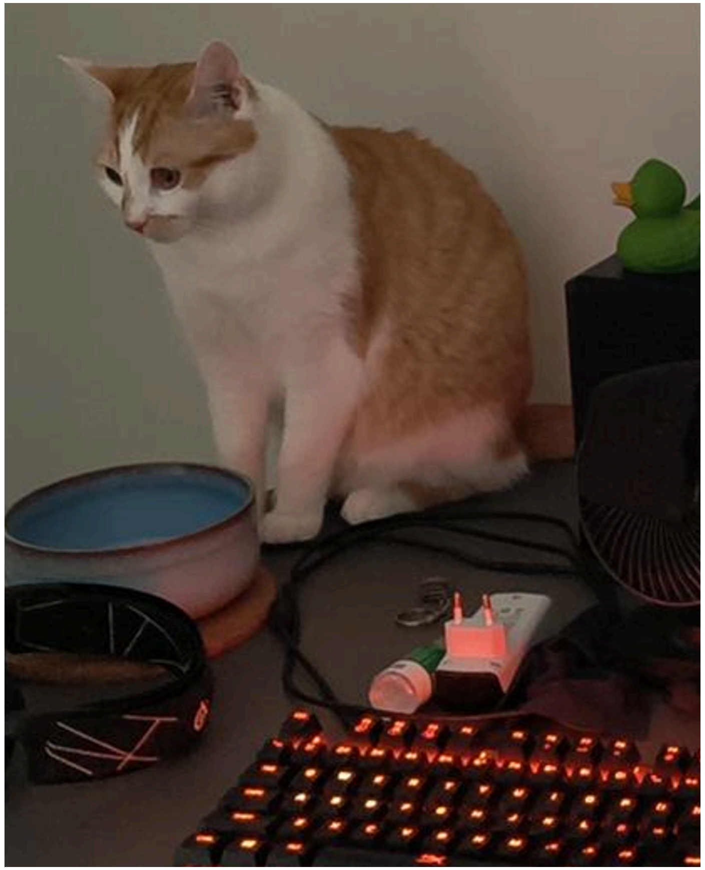
2025.10.14



First problem

First deadline – 29.10.2025, 12:00 PM.

Always try to get non-zero points :)



Optimisation problems

Make a number of decisions creating a strategy that produces an optimal (minimal, maximal, ...) result.

Can often be solved with *dynamic programming*.

Optimisation problems

Make a number of decisions creating a strategy that produces an optimal (minimal, maximal, ...) result.

Can often be solved with *dynamic programming*.

For DP to be applicable the problem has to have **optimal substructure**.

To solve a big problem we can use optimal solutions of smaller problems solved **independently**.

Dynamic programming

Generic format: we are filling a DP table.

The size of the table is the number of states.

Overall running time is number of states times the cost of computing one state.

Dynamic programming – examples

Discrete knapsack.

LCS-likes (e.g. editing distance)

Optimal ordering of matrices to multiply.

Join ordering.

Counting combinations.

Tasks of the form “path from top-left to bottom-right optimising some value”.

Shortest paths in graphs with negative weights.

Longest path **in a DAG**.

NOT optimal substructure

Shortest paths in a graph have optimal substructure.

For the shortest path $v \rightsquigarrow u$ pick a middle vertex w and consider $v \rightsquigarrow w \rightsquigarrow u$.

Picking the shortest path $v \rightsquigarrow w$ and $w \rightsquigarrow u$ works.

NOT optimal substructure

Shortest paths in a graph have optimal substructure.

For the shortest path $v \rightsquigarrow u$ pick a middle vertex w and consider $v \rightsquigarrow w \rightsquigarrow u$.

Picking the shortest path $v \rightsquigarrow w$ and $w \rightsquigarrow u$ works.

For **longest** simple paths this substructure does not exist.

The longest paths $v \rightsquigarrow w$ and $w \rightsquigarrow u$ might share vertices which cannot be picked again, so they're **not independent**.

This problem is actually NP-complete.

Case study: Aliens (IOI 2016)

Our satellite has just discovered an alien civilization on a remote planet. We have already obtained a low-resolution photo of a square area of the planet. The photo shows many signs of intelligent life. Our experts have identified points of interest in the photo. The points are numbered from 0 to $n - 1$. We now want to take high-resolution photos that contain all of those points.

Case study: Aliens (IOI 2016)

Internally, the satellite has divided the area of the low-resolution photo into an m by m grid of unit square cells. Both rows and columns of the grid are consecutively numbered from 0 to $m-1$ (from the top and left, respectively). We use (s,t) to denote the cell in row s and column t. The point number i is located in the cell (r_i,c_i). Each cell may contain an arbitrary number of these points.

Case study: Aliens (IOI 2016)

Our satellite is on a stable orbit that passes directly over the main diagonal of the grid. The main diagonal is the line segment that connects the top left and the bottom right corner of the grid. The satellite can take a high-resolution photo of any area that satisfies the following constraints:

- the shape of the area is a square,
- two opposite corners of the square both lie on the main diagonal of the grid,
- each cell of the grid is either completely inside or completely outside the photographed area.

The satellite is able to take at most k high-resolution photos.

Case study: Aliens (IOI 2016)

Once the satellite is done taking photos, it will transmit the high-resolution photo of each photographed cell to our home base (regardless of whether that cell contains some points of interest). The data for each photographed cell will only be transmitted once, even if the cell was photographed several times. Thus, we have to choose at most k square areas that will be photographed, assuring that:

- each cell containing at least one point of interest is photographed at least once, and
- the number of cells that are photographed at least once is minimized.

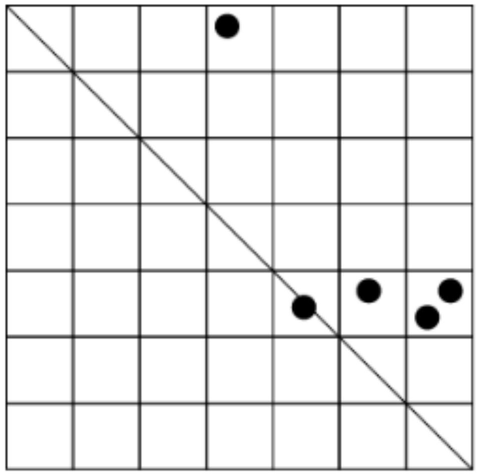
Your task is to find the smallest possible total number of photographed cells.

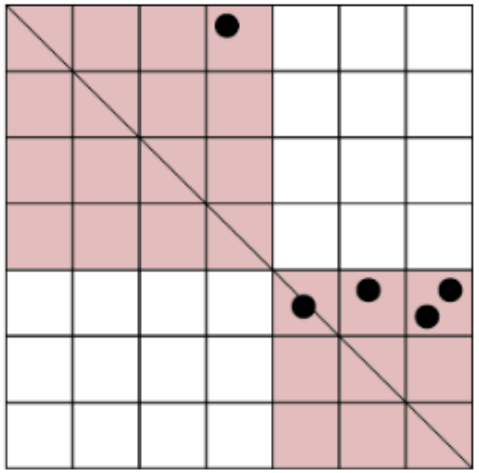
Case study: Aliens (IOI 2016)

Example:

```
5 7 2
0 3
4 4
4 6
4 5
4 6

25
```





Aliens – limits

$1 \leq k \leq n$

$1 \leq n \leq 100\,000$

$1 \leq m \leq 1\,000\,000$

Aliens – limits

$1 \leq k \leq n$

$1 \leq n \leq 100\,000$

$1 \leq m \leq 1\,000\,000$

Simpler subtasks:

- $n \leq 500, m \leq 1\,000$
- $n \leq 4\,000, m \leq 1\,000\,000$
- $n \leq 50\,000, k \leq 100, m \leq 1\,000\,000$

Aliens - simplify

What if $n = k$?

Aliens - simplify

What if $n = k$?

It’s always worth it to take a smaller photo for each point.

Aliens - simplify

What if $\forall_i . r_i = c_i$?

Aliens - simplify

What if $\forall_i . r_i = c_i$?

Optimally, each photo is taken at some r_i.

If a photo covers r_i and r_j then also all $r_i \leq r_x \leq r_j$.

Aliens - simplify

What if $\forall_i . r_i = c_i$?

Optimally, each photo is taken at some r_i.

If a photo covers r_i and r_j then also all $r_i \leq r_x \leq r_j$.

Cover n points on the line using k segments, such that sum of squares of segment lengths is minimal.

Aliens - DP

$\forall i \in [0,n]: r_i \leq r_{i+1}$

$$\mathrm{DP}_{i,j} := \text{minimum cost to cover first } i \text{ points with at most } j \text{ photos}$$

$$\mathrm{DP}_{0,j} = 0$$

$$\mathrm{DP}_{i,1} = (r_{i-1} - r_0 + 1)^2$$

$$\mathrm{DP}_{i,j} = \min_{1 \leq x < i} \mathrm{DP}_{x,j-1} + (r_{i-1} - r_x + 1)^2$$

Answer in $\mathrm{DP}_{n,k}$.

$\mathcal{O}(nk)$ states, calculating one costs $\mathcal{O}(n)$, total $\mathcal{O}(n^2k)$.

Aliens - DP

Now drop the $r_i = c_i$ simplification.

Aliens - DP

Now drop the $r_i = c_i$ simplification.

A photo from (x, x) to (y, y) $(x \leq y)$ covers a point (r_i, c_i) iff the segment $[\min(r_i, c_i), \max(r_i, c_i)]$ is fully contained in $[x, y]$.

So now we are covering segments instead of points.

Aliens - DP

First, convert (r_i, c_i) to segment $\overset{l_i}{\cancel{s}_i} = \min(r_i, c_i), \overset{r_i}{\cancel{e}_i} = \max(r_i, c_i)$. Sort by s_i ascending and then e_i descending.

Remove segments fully contained in other ones. Then:

$$\mathrm{DP}_{i,j} := \text{minimum cost to cover first } i \text{ points with at most } j \text{ photos}$$

$$\mathrm{DP}_{0,j} = 0$$

$$\mathrm{DP}_{i,1} = (r_{i-1} - \overset{l_0}{\cancel{r_0}} + 1)^2$$

$$\mathrm{DP}_{i,j} = \min_{1 \le x < i} \mathrm{DP}_{x,j-1} + (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2$$

Still $\mathcal{O}(n^2 k)$.

Knuth's optimization

Work by Knuth in 1971, improved by Yao (a.k.a. the Knuth-Yao Speedup).[1]

Given a recurrence relation (over $i \leq n, j \leq m$):

$$f_{i,j} = c_{i,j} + \min_{i<k\leq j}\left(f_{i,k-1} + f_{k,j}\right)$$

and assuming the *optimal* splitting point $k_{i,j}$ satisfies:

$$k_{i,j-1} \leq k_{i,j} \leq k_{i+1,j} \qquad \text{for } i \leq j$$

we can speed up computation by a linear factor ($\mathcal{O}\left(n^2 m\right)$ to $\mathcal{O}\left(n^2\right)$).

[1] You can find the proof of all this in Yao's *Efficient Dynamic Programming Using Quadrangle Inequalities*, STOC'80.

Knuth's optimization

$$f_{i,j} = c_{i,j} + \min_{i<k\leq j}\left(f_{i,k-1} + f_{k,j}\right)$$

Previous may be hard to prove. But, if the cost function c is *monotone* and satisfies the *quadrangle inequality*:

$$c_{i',j'} \leq c_{i,j}$$

$$c_{i,j} + c_{i',j'} \leq c_{i',j} + c_{i,j'} \quad \text{for } i \leq i' \leq j \leq j'$$

then it implies the splitting point property.

Knuth's optimization

$$f_{i,j} = c_{i,j} + \min_{i<k\le j}\left(f_{i,k-1} + f_{k,j}\right)$$

To calculate $k_{i,j}$, instead of testing values from $i + 1$ to j we only need to look at values between $k_{i,j-1}$ to $k_{i+1,j}$.

We need to change the loop directions of i for that.

It can be proven that this results in at most $\mathcal{O}(n)$ minimisation operations for each $\delta = j - i$, giving $\mathcal{O}\left(n^2\right)$ complexity.

Knuth's optimization – note

$$f_{i,j} = c_{i,j} + \min_{i<k\leq j}\left(f_{i,k-1} + f_{k,j}\right)$$

This is a general optimisation.

It of course also works for max, or for slightly different bounds (e.g. $\max_{0\leq k<j}$), etc.

The point is to understand the technique, not memorise an equation.

Aliens – Knuth’s optimization

Initialization of DP[0] & DP[1], S

We have code like:

```
for i in [1,n]
  for j in [2,k]
    DP[i][j] = DP[i][j-1]
    for ~~k~~ x in [1, i)
      let local = DP[x][j-1] +
        (S[i-1].r - S[x].l + 1)^2 - max(0, S[x-1].r - S[x].l + 1)^2
      DP[i][j] = min(DP[i][j], local)
```

Aliens – Knuth's optimization

Initialitation of DP, opt, S

Switching loop order and adding an additional table:

```
for j in [2,k]
  for i in [n,1]
    DP[i][j] = DP[i][j-1]
    opt[i][j] = i
    for ~~k~~ x in [opt[i][j-1], opt[i+1][j])
      let local = DP[x][j-1] +
        (S[i-1].r - S[x].l + 1)^2 - max(0, S[x-1].r - S[x].l + 1)^2
      if local <= DP[i][j]
        DP[i][j] = local
        opt[i][j] = k
```

Divide and Conquer

Divide and Conquer is a general algorithm construction technique.

Split into smaller subproblems, merge solutions.

Prime example – merge-sort.

Divide and Conquer DP optimisation

For recurrences over $i \leq n, j \leq m$ where we optimise something similar to

$$f_{i,j} = \min_{0 \leq k \leq j} f_{i-1,k} + c_{i,k}$$

and we have the previously established property:

$$k_{i,j} \leq k_{i,j+1}$$

we can apply divide-and-conquer to improve complexity from $\mathcal{O}(n^2 m)$ to $\mathcal{O}(nm \log n)$.

Divide and Conquer DP optimisation

Since $k_{i,j}$ for fixed i increases as j increases we know, after computing some $k_{i,j}$, that for $j' < j$ $k_{i,j'}$ is bound by $k_{i,j}$.

First calculate $k_{i,\frac{n}{2}}$. Using that bound find $k_{i,\frac{n}{4}}$ ($\leq k_{i,\frac{n}{2}}$) and $k_{i,\frac{3n}{4}}$ ($\geq k_{i,\frac{n}{2}}$), etc.

Creates a recursion tree of $\log n$ levels, the same value of k can occur only twice at the same level.

We get $\mathcal{O}(nm \log n)$, which might be better than Knuth's speedup if m is small.

Aliens – Divide and Conquer

Applying to our task is straightforward.

Easiest to implement recursively:

```
fn solve(j, i_lo, i_hi, x_lo, x_hi)
  if i_lo > i_hi return
  let i_mid = (i_hi + i_lo) / 2
  // find DP[i_mid, j], opt[i_mid, j] like before
  solve(j, i_lo, i_mid - 1, x_lo, opt[i_mid, j])
  solve(j, i_mid + 1, i_hi, opt[i_mid, j], x_hi)
```

Convex Hull Trick

Consider a problem where we have a set of linear functions $a_i x + b_i$ and want to answer queries – which function is minimal (or maximal) at given x?

E.g. minimising cost or maximising yield of an investment.

Naively, you have to look through all functions for each x.

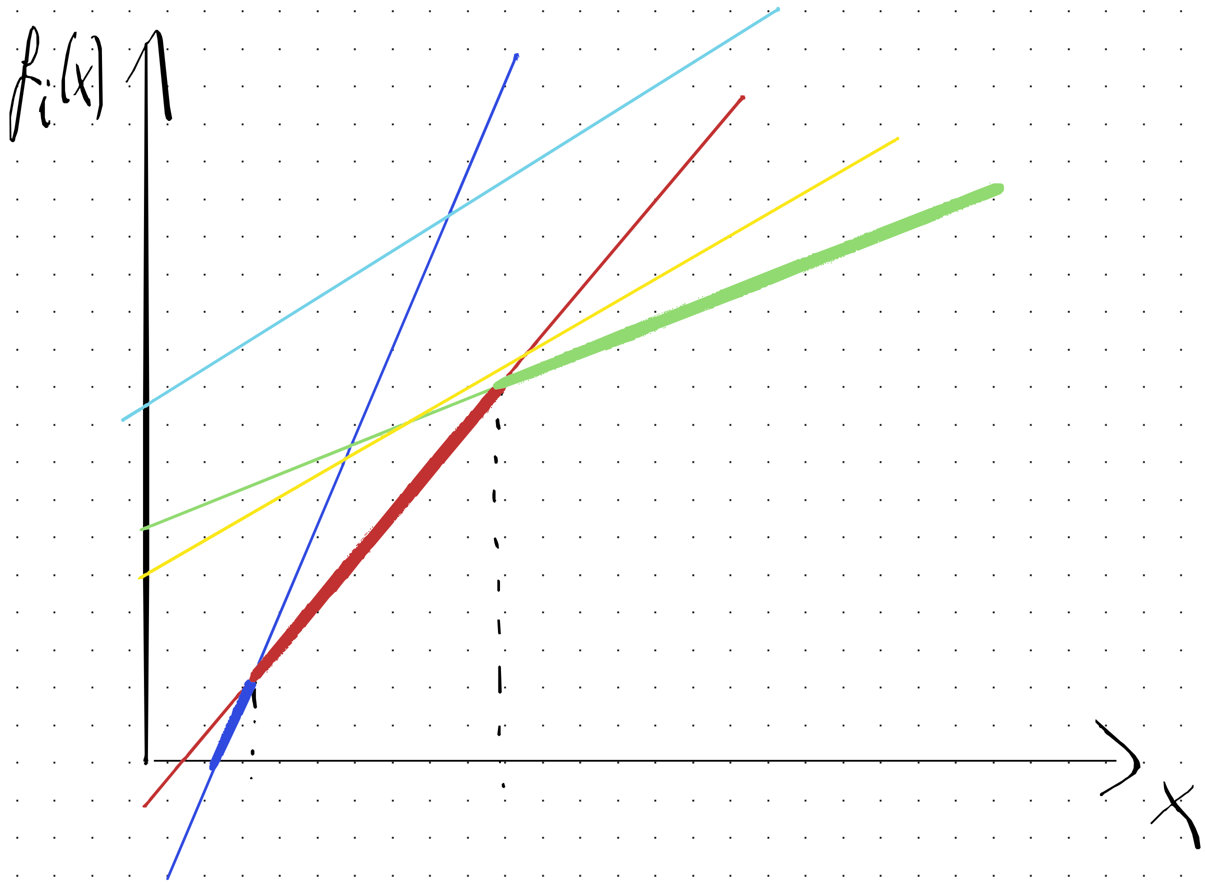
Convex Hull Trick

Sort the functions descending by a_i (for maximum, ascending).

Sort the queries ascending.

Key observation is that if at some point f_i is better than f_j, $i > j$, then it will always be better.

Convex Hull Trick



Convex Hull Trick

Keep a stack of functions.

When adding a function remove all from the stack that are dominated.

When querying, check the two top functions. If the top one is no longer the best, remove it.

Both operations are $\mathcal{O}(1)$ amortised.

Aliens – Convex Hull Trick

We can rewrite the DP equation:

$$\begin{aligned}
\mathrm{DP}_{i,j} &= \min_{1\le x<i} \mathrm{DP}_{x,j-1} + (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2 \\
&= \min_{1\le x<i} \mathrm{DP}_{x,j-1} + r_{i-1}^2 - 2(l_x - 1)r_{i-1} + (l_x - 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2 \\
&= C_i + \min_{1\le x<i} A_x r_{i-1} + B_{x,j}
\end{aligned}$$

Thus, we have linear functions with coefficients $A_x, B_{x,j}$, evaluated at r_{i-1}.

As i increases, A_x decreases and r_{i-1} increases.

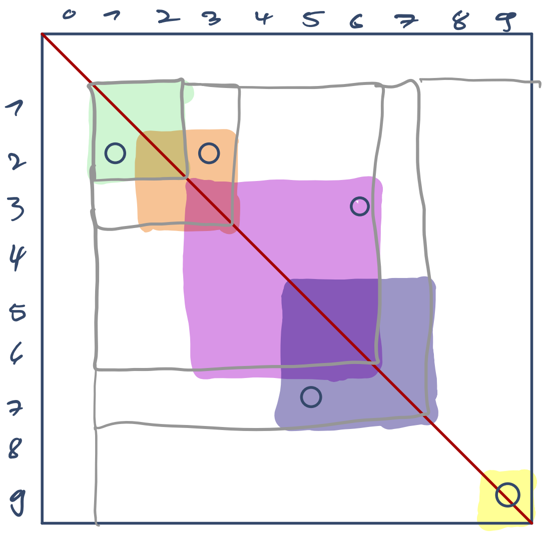
Aliens – Convex Hull Trick

Go over j first.

At each i first add the function $A_i x + B_{i,j}$ to the set and then query for minimum at r_{i-1}.

Total running time $\mathcal{O}(nk)$.

Aliens – Convex Hull Trick



i	l	r
0	1	2
1	2	3
2	3	6
3	5	7
4	9	9

$$\mathrm{DP}_{i,j} = \min_{1\le x<i} \mathrm{DP}_{x,j-1} + (r_{i-1} - l_x + 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2$$

$$= \min_{1\le x<i} \mathrm{DP}_{x,j-1} + r_{i-1}^2 - 2(l_x - 1) r_{i-1} + (l_x - 1)^2 - (\max(0, r_{x-1} - l_x + 1))^2$$

$$= C_i + \min_{1\le x<i} A_x r_{i-1} + B_{x,j} = \mathrm{DP}_{x,j-1} + (l_x - 1)^2 - \max(0, r_{x-1} - l_x + 1)^2$$

$j=1$, $i=[0,\dots 4]$: $\mathrm{DP}[1] = [4, 9, 36, 49, 81]$

$j=2$:

Parameter search

This optimisation is also known as *lambda optimisation*, or… *the trick from Aliens*.

This is a trick that reduces the number of dimensions in a DP as long as the DP functions is *convex*.

Parameter search – convexity

A function is convex if connecting two points on its graph doesn't lead outside of the graph.

In case of our DP problems over (i, j), this is equivalent to saying:

$$f_{i,j} - f_{i,j+1} \geq f_{i,j+1} - f_{i,j+2}$$

Intuitively, further increasing the use of resource j gives diminishing returns.

In the case of Aliens, taking the j-th photo decreases the result by more than taking the $j + 1$-st photo.

Parameter search – convexity

The idea is to transform our function from:

$$f_{i,j} = \min_{0 \le k \le j} f_{i-1,k} + c_{i,k}$$

into:

$$\tilde{f}_{i,j} = f_{i,j} + \lambda j$$

and an optimisation problem:

$$g_i = \min_{0 \le k \le n} \tilde{f}_{i,k} = \min_{0 \le k \le n} \left(f_{i,k} + \lambda \overset{k}{\cancel{j}} \right)$$

Intuitively, λ is a penalty for taking more of the j resource.

Parameter search – convexity

$$g_i = \min_{0 \le k \le n} \tilde{f}_{i,k} = \min_{0 \le k \le n} \left(f_{i,k} + \lambda \overset{k}{\cancel{i}} \right)$$

The trick here is that g_i is usually *much easier* to compute, as it has only one dimension.

Crucially, $\tilde{f}$ is also convex.

Let the function $k_{\mathrm{g}}(\lambda)$ return the optimal splitting point k above for given λ.

Since $\tilde{f}$ is convex, this is the minimum k such that $\tilde{f}_{n,k} - \tilde{f}_{n,k+1}$ is negative.

The $k_{\mathrm{g}}(\lambda)$ is unimodal, and thus we can ternary search λ and find such λ_{opt} that $k_{\mathrm{g}}\left(\lambda_{\mathrm{opt}}\right)$ satisfies the resource constraint in the task.

Parameter search – result

$$g_i = \min_{0 \le k \le n} \tilde{f}_{i,k} = \min_{0 \le k \le n} \left(f_{i,k} + \lambda \overset{k}{\cancel{j}} \right)$$

We have two points now: $k_1 = k_g\left(\lambda_{\text{opt}}\right)$ and $k_2 = k_g\left(\lambda_{\text{opt}} + 1\right)$. To find $f_{n,k}$, the answer to the task, we need to interpolate $f_{n,j}$ on this interval.

This can be done since all differences $\left(f_{n,k_2} - f_{n,k_2+1}\right), \left(f_{n,k_2+1} - f_{n,k_2+2}\right), \dots, \left(f_{n,k_1-1}, f_{n,k_1}\right)$ are equal.

Aliens – parameter search

For Aliens, the penalty λ is given to each additional photo:

$$g_i = \min_{0 \le x \le n} \tilde{f}_{i,x} = \min_{0 \le x \le n} \left(f_{i,x} + \lambda \overset{x}{\cancel{j}} \right)$$

$$g_i = \min_{0 \le x \le n} g_x + (r_{i-1} - l_x + 1)^2 - \max(0, r_{x-1} - l_x + 1)^2 + \lambda x$$

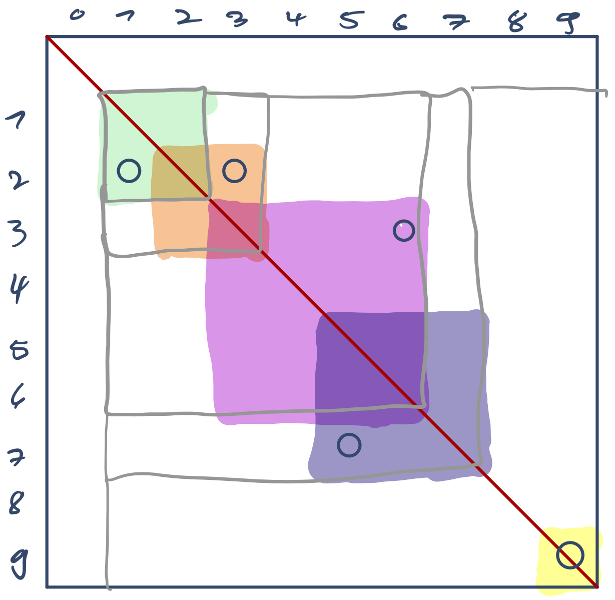
This can be computed using the convex hull trick in $\mathcal{O}(n)$.

Adding ternary search and sorting the segments from the start gives us $\mathcal{O}(n \log m + n \log n)$.

Aliens – parameter search

$$g_i = \min_{0 \le x \le n} \tilde{f}_{i,x} = \min_{0 \le x \le n} \left(f_{i,x} + \lambda \overset{x}{\cancel{x}}\right)$$

$$g_i = \min_{0 \le x \le n} g_x + (r_{i-1} - l_x + 1)^2 - \max(0, r_{x-1} - l_x + 1)^2 + \lambda x$$



i	l	r
1	1	2
2	2	3
3	3	6
4	5	7
5	9	9

$DP[1] = [4, 9, 36, 49, 81]$

λ-search over $\lambda \in [0, 81]$

$\lambda = 40$

$A_1 = -2(1-1) = 0$

$B_{40,1} = F(0) + (0)^2 - 0 + 40 = 40$

- $i=1$, $r_1 = 2$, only $x=1$. value $= 0 \cdot 2 + 40 = 40$
  $F(1) = 2^2 + 40 = 44$, $k = 1$
  $\Rightarrow x=2$, $A_2 = -2(2-1) = -2$, $B_{2,40} = F(1) + 1^2 - 1 + 40 = 84$
- $i=2$, $r_2=3$, CHT min $x=1$ $\Rightarrow$ value $0 \cdot 3 + 40 = 40$
  $F(2) = 3^2 + 40 = 49$, $k=1$
  $\Rightarrow x=3$, $A_3 = -2(3-1) = -4$, $B_{3,40} = F(2) + 2^2 - 1 + 40 = 104$
- $i=3$, $r_3 = 6$, CHT min $x=1$: $0 \cdot 6 + 40 = 40$
  $g_3 = 6^2 + 40 = 76$, $k=1$
  $\Rightarrow x=4$, $A_4 = -2(5-1) = -8$, $B_{4,40} = g_3 + 4^2 - 4 + 40 = 128$
- $i=4$, $r_4 = 7$, CHT min $x=1$: $0 \cdot 7 + 40 = 40$
  $g_4 = 7^2 + 40 = 89$, $k=1$
  $\Rightarrow x=5$, $A_5 = -2(9-1) = -16$, $B_{5,40} = g_4 + 8^2 - 0 + 40 = 193$
- $i=5$, $r_5 = 9$: CHT min $x=1$: $0 \cdot 9 + 40 = 40$
  $g_5 = 9^2 + 40 = 121$, $k=1$

$\Rightarrow k(40) = 1 < 3 \Rightarrow \lambda_{opt} < 40$

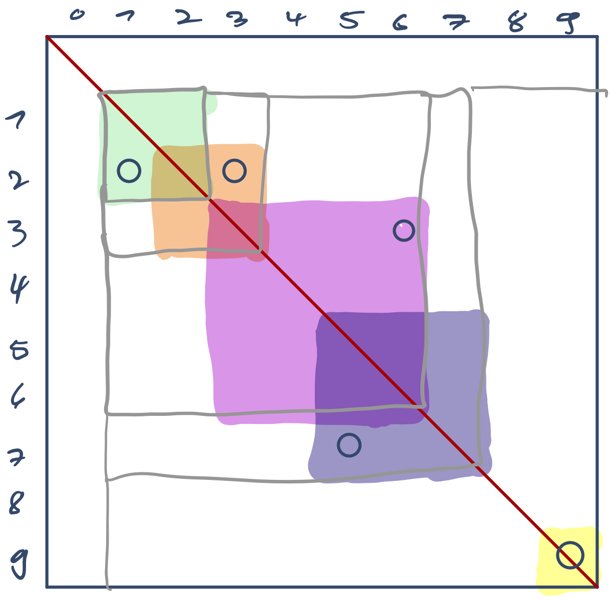
Aliens – parameter search

$$g_i = \min_{0 \le x \le n} \tilde{f}_{i,x} = \min_{0 \le x \le n} \left(f_{i,x} + \lambda \overset{x}{\cancel{x}}\right)$$

$$g_i = \min_{0 \le x \le n} g_x + (r_{i-1} - l_x + 1)^2 - \max(0, r_{x-1} - l_x + 1)^2 + \lambda x$$

DP[1] = [4, 9, 36, 49, 81]

λ-search over λ∈[0, 81]



i	l	r
1	1	2
2	2	3
3	3	6
4	5	7
5	9	9

λ =	gi	k
40	121	1
19	88	2
9	61	3
14	76	3
16	82	$\boxed{3}$ = j
17	84	2

⟹ no interpolation

$DP[5,3] = g_5(\lambda = 16) - 16 \cdot 3 = 82 - 48 = 34$