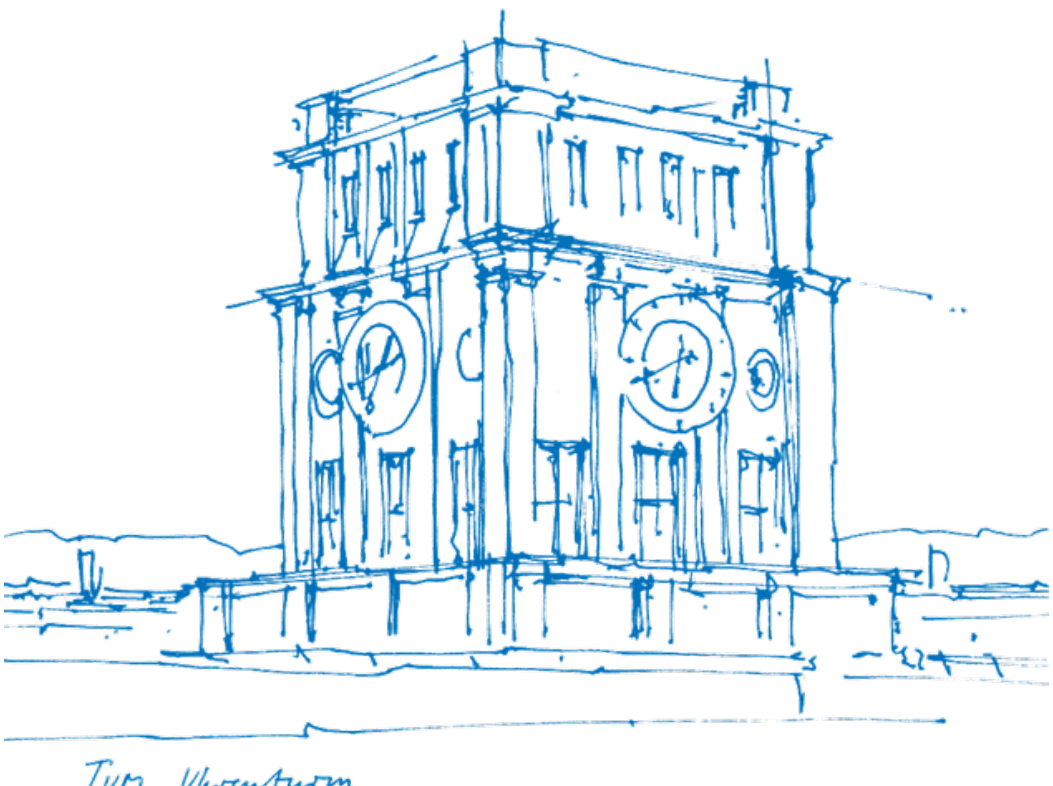
Please adjust group name in tumuser.sty.
Department of Electrical and Computer Engineering
Technical University of Munich

AACPP 26/27

Class 2: Dynamic Programming

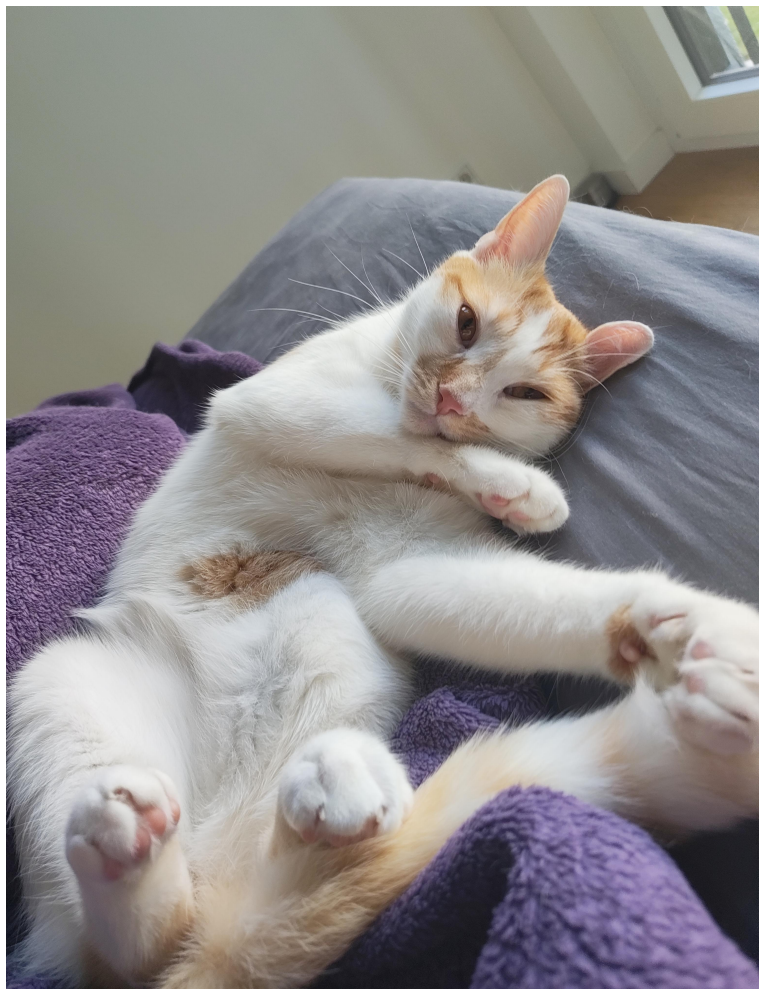
Julia Spindler
Mateusz Gienieczko

October 29, 2026



Second problem

Second deadline – 12.11.2026, 12:00 PM



DUN

Given a sequence of n pairs (a, b) answer q queries.
For query j, start with score x_j and for every pair between l_j and r_j either multiply score by a_i or add b_i. Return the result modulo $M = 10^9 + 7$.

DUN – brute force

Basic observation: this is a greedy problem.
At every event we should take the option that maximises the local score.

Once we exceed $10^9 + 7$ it's always worth it to multiply, unless $b_i = 1$. Call these moves *big-optimal*.
Keep an overflow flag to make decisions without keeping big numbers.

Naive solution: for every query go through all events, $\mathcal{O}(nq)$.

DUN – observations

Multiplying grows the score quickly – we reach M after $\log M$ multiplications.

Consecutive $b_i = 1$ events can be merged into a single block with the sum of their a_i.

In the end, we only need to process at most $2 \log M$ events to reach score $\geq M$.

DUN – model solution

We create two segment trees – one for small moves and one for big-optimal moves.

First, go through the events until reaching M. Query the first tree to find ends and aggregate sums of blocks with $b_i = 1$.

Then, query the second tree for the big-optimal score until r.

DUN – model solution

Question: how to compose big-optimal scores?

The score modifier is $\lambda x.px + q$ (represented as (p, q)). For leaves, we have either $\lambda x.b_i x + 0$ when $b_i > 1$, or $\lambda x.1x + a_i$ when $b_i = 1$.
Composition of (p, q) and (p', q'):

$$\begin{aligned}(p', q') \circ (p, q) &= \lambda x.p'x + q' \circ \lambda x.px + q \\ &= \lambda x.p'(px + q) + q' \\ &= \lambda x.p'px + p'q + q' \\ &= (p'p, p'q + q')\end{aligned} \tag{1}$$

All operations modulo M!

DUN – model solution

Time complexity: preprocessing in $\mathcal{O}(n)$, then each query in $\mathcal{O}(q(\log M + \log n))$.
Total: $\mathcal{O}(n + q(\log M + \log n))$.

Dynamic Programming

DP can be used to solve optimization problems (find strategy to achieve minimal/maximal result)

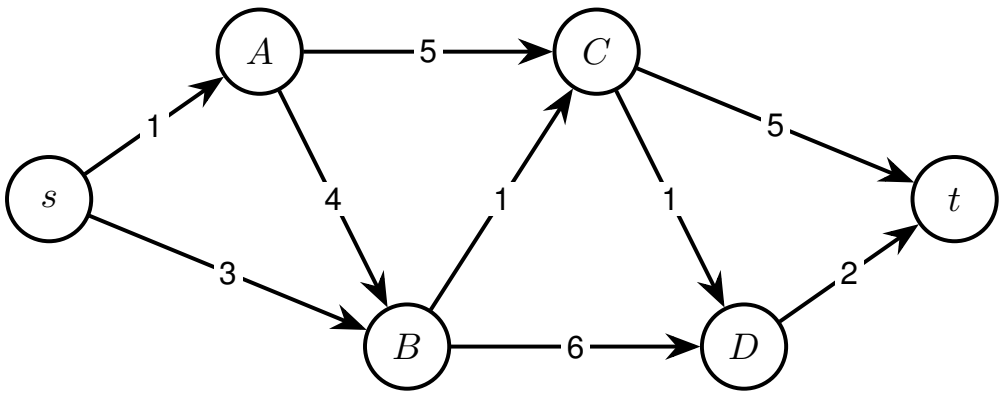
Problem needs an “inductive” structure to be solvable by DP

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

Dynamic Programming – Examples

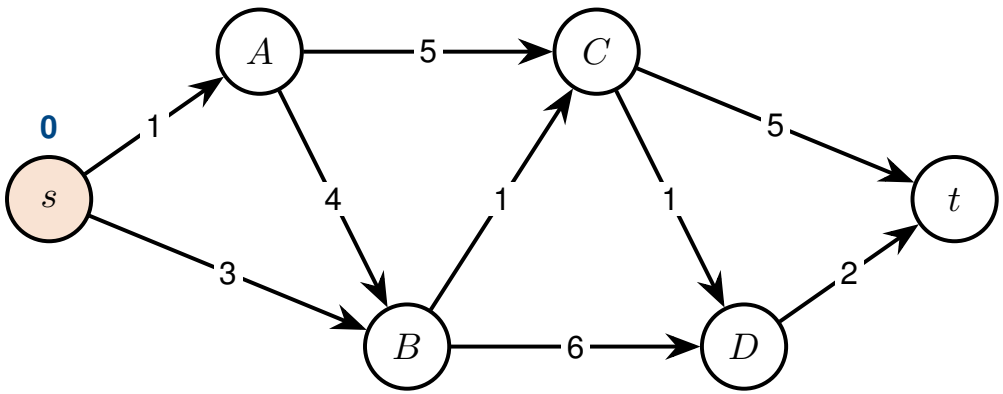
- Discrete Knapsack
- Longest Common Substring
- Edit Distance
- Shortest Path in a DAG

Shortest paths in a DAG



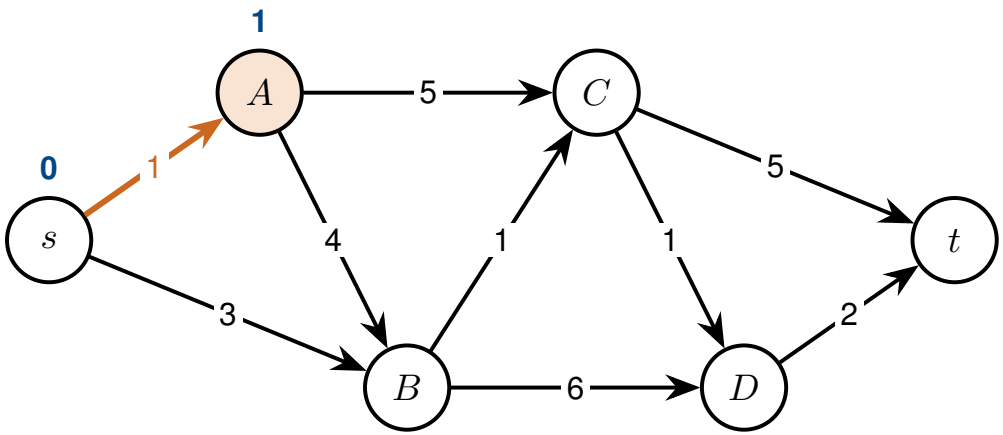
v	candidates $d(u) + w(u, v)$	$d(v)$
s		
A		
B		
C		
D		
t		

Shortest paths in a DAG



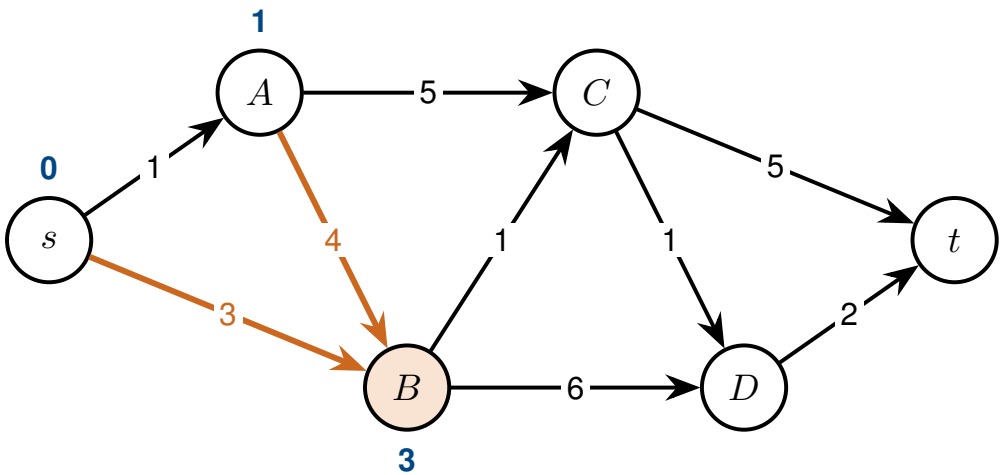
v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A		
B		
C		
D		
t		

Shortest paths in a DAG



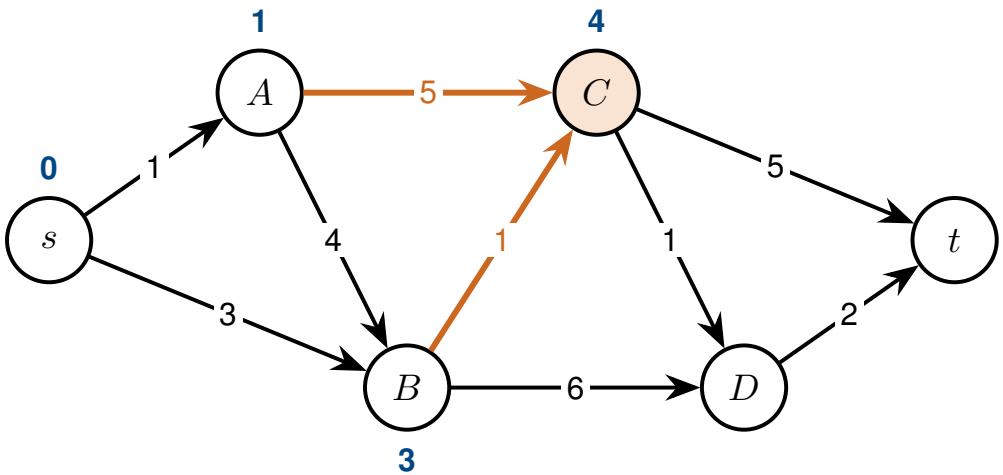
v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B		
C		
D		
t		

Shortest paths in a DAG



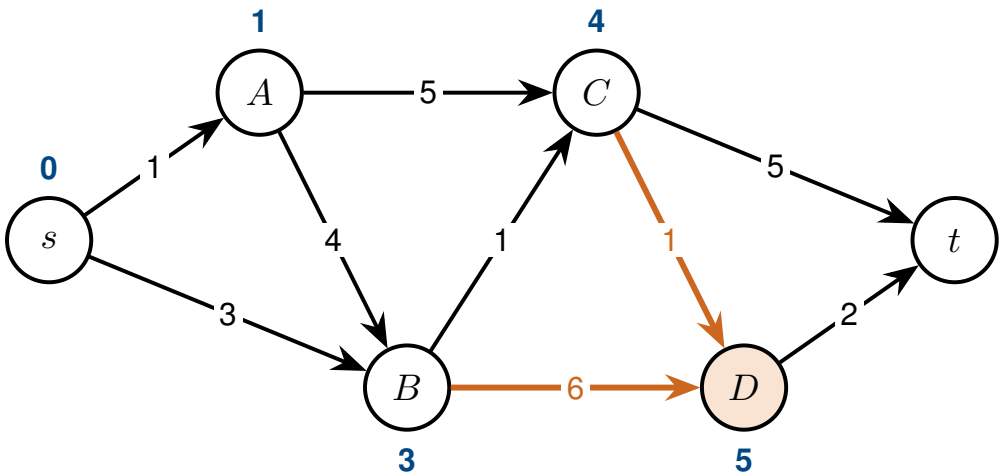
v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B	$\min(\underset{s}{\mathbf{0}} + \mathbf{3},\ 1 + \underset{A}{4})$	3
C		
D		
t		

Shortest paths in a DAG



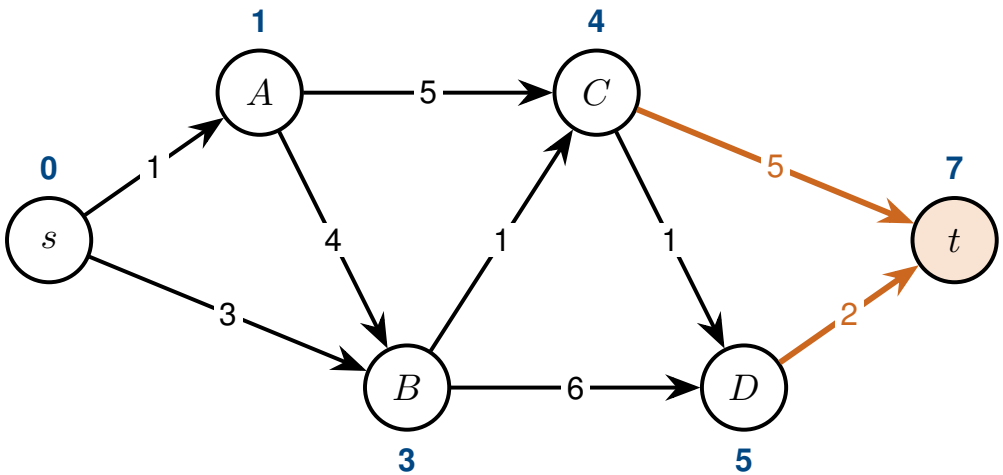
v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B	$\min(\underset{s}{\mathbf{0}} + \mathbf{3},\ 1 + \underset{A}{4})$	3
C	$\min(1 + \underset{A}{5},\ \underset{B}{\mathbf{3}} + \mathbf{1})$	4
D		
t		

Shortest paths in a DAG



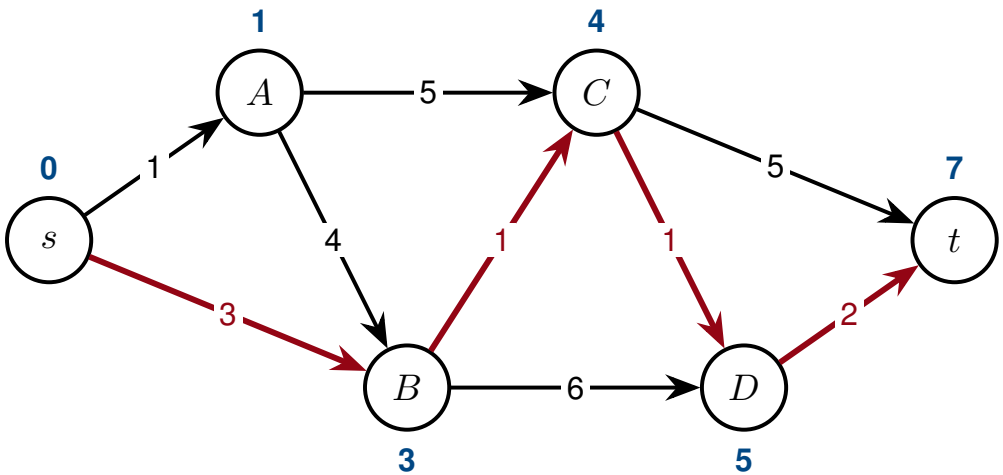
v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B	$\min(\underset{s}{\mathbf{0}} + \mathbf{3},\ 1 + \underset{A}{4})$	3
C	$\min(1 + \underset{A}{5},\ \underset{B}{\mathbf{3}} + \mathbf{1})$	4
D	$\min(3 + \underset{B}{6},\ \underset{C}{\mathbf{4}} + \mathbf{1})$	5
t		

Shortest paths in a DAG



v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B	$\min(\underset{s}{\mathbf{0}} + \mathbf{3},\ 1 + \underset{A}{4})$	3
C	$\min(1 + \underset{A}{5},\ \underset{B}{\mathbf{3}} + \mathbf{1})$	4
D	$\min(3 + \underset{B}{6},\ \underset{C}{\mathbf{4}} + \mathbf{1})$	5
t	$\min(4 + \underset{C}{5},\ \underset{D}{\mathbf{5}} + \mathbf{2})$	7

Shortest paths in a DAG



v	candidates $d(u) + w(u, v)$	$d(v)$
s	start	0
A	$\min(\underset{s}{\mathbf{0}} + \mathbf{1})$	1
B	$\min(\underset{s}{\mathbf{0}} + \mathbf{3},\ 1 + \underset{A}{4})$	3
C	$\min(1 + \underset{A}{5},\ \underset{B}{\mathbf{3}} + \mathbf{1})$	4
D	$\min(3 + \underset{B}{6},\ \underset{C}{\mathbf{4}} + \mathbf{1})$	5
t	$\min(4 + \underset{C}{5},\ \underset{D}{\mathbf{5}} + \mathbf{2})$	7

$$d(s) = 0, \qquad d(v) = \min_{(u,v)\in E} \big(d(u) + w(u, v)\big)$$

Following the bold minima backwards from t gives the path $s \to B \to C \to D \to t$.

Runtime

We have to go through all nodes and we look at all edges once $\Rightarrow O(n+m)$

Implemented as an adjacency matrix the runtime is $O(n^2)$

We can sometimes use the convex hull trick (CHT) to solve optimization problems in $O(n)$ or $O(n \cdot log(n))$

CHT

Consider a problem where we have to find the min/max at a certain x given a set of linear functions, i.e.:

$$f_i = c_i + \min_{j<i}(m_j \cdot x_i + k_j)$$

- c_i and x_i depend only on i
- m_j and k_j depend only on j (they may use f_j, which is already known)

Computing every f_i by trying all j takes $O(n^2)$.

The convex hull trick computes all of them in $O(n)$ (or $O(n \log n)$).

CHT

Often two things are sorted:

- Slopes: lines are added in order of (de/in)creasing slope m_j
- Queries: The x_i increase

$\Rightarrow$ If at some point f_i is better than f_j, $i > j$ then it will always be better

CHT

Keep a deque of functions

Add functions from the back and remove all those that are now dominated also starting from the back

When querying, check from the front. If the first function is no longer the best, remove it. Repeat.

Any function is at most added once and removed once

Both operations are amortized $O(1) \rightarrow$ the entire algorithm runs in $O(n)$.

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.

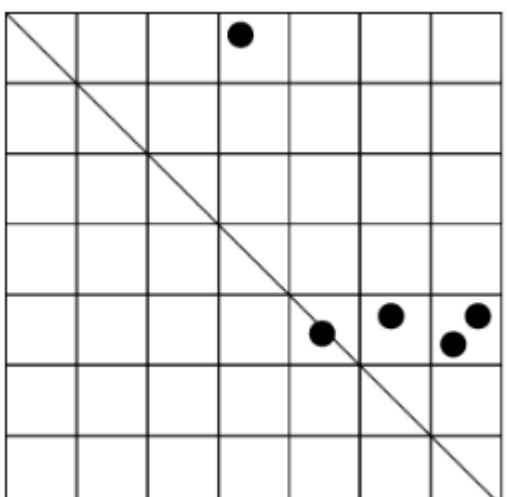
Your task is to find the smallest possible total number of photographed cells.

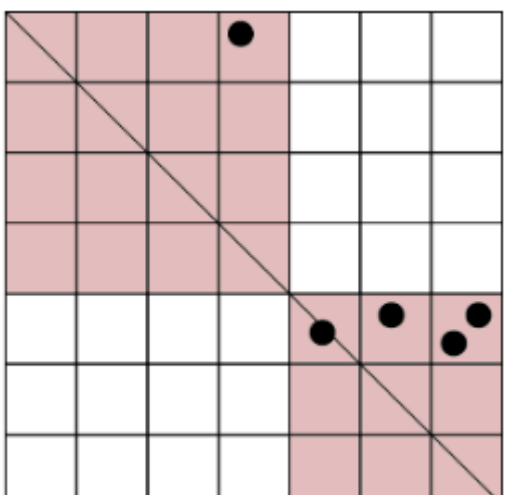
Case study: Aliens (IOI 2016)

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

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

$\mathrm{DP}_{i,j} :=$ minimum cost to cover first i points with at most j 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 $\mathcal{O}\left(n^2 k\right)$.

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 $l_i = \min(r_i, c_i) re_i = \max(r_i, c_i)$. Sort by l_i ascending and then r_i descending.

Remove segments fully contained in other ones. Then:

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

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

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

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

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 $i \leq n, j \leq m$):

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

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

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

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

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

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

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

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

then it implies the splitting point property.

Knuth's optimization

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

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

We need to change the loop directions 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

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

This is a general optimisation.

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

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

Aliens – Knuth's optimization

We have code like:

```
for i in [1,n]
  for j in [2,k]
    DP[i][j] = DP[i][j-1]
    for 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

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 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
```

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 $\mathcal{O}(nk)$.

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 – convexivity

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 – convexivity

The idea is to transform our function from:

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

into:

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

and an optimisation problem:

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

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

Parameter search – convexivity

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

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 $x_g(\lambda)$ return the optimal splitting point x above for given λ.

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

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

Parameter search – result

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

We have two points now: $x_1 = x_g\left(\lambda_{\text{opt}}\right)$ and $x_2 = x_g\left(\lambda_{\text{opt}} + 1\right)$. To find $f_{n,x}$, the answer to the task, we need to interpolate $f_{n,j}$ on this interval.

This can be done since all differences $\left(f_{n,x_2} - f_{n,x_2+1}\right), \left(f_{n,x_2+1} - f_{n,x_2+2}\right), \ldots, \left(f_{n,x_1-1}, f_{n,x_1}\right)$ are equal.

Aliens – parameter search

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

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

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

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 $\mathcal{O}(n \log m + n \log n)$.

See you next week

RAI: 12.11.2026, 12:00 PM

Good luck!

