Basic Greedy algorithm for a Job Scheduling problem

Jobs \( J_1, J_2, \ldots, J_n \) each with unit processing requirement

Deadlines \( d_1, d_2, \ldots, d_n \)

Penalty \( p_1, p_2, \ldots, p_n \)

Obj: Maximize the penalty of the scheduled jobs

A set of jobs \( J_1, J_2, \ldots, J_k \) is "feasible" if they can be scheduled without incurring any penalty.

Basic Greedy

Starting from the largest penalty job, keep adding the next highest penalty-incurring job (if feasible).

Does basic greedy yield optimal solution?
We will try to prove the exchange property.

\[ \begin{array}{ccccccc}
A_1 & A_2 & A_3 & \ldots & A_k & A_{k+1} \\
B_1 & B_2 & B_3 & \ldots & B_k & B_{k+1} \\
\end{array} \]

\[ A \cap B \neq \emptyset \]

**Case I** \( B_{k+1} \neq A \)

The jobs are given in order of some feasible schedule.

(It is an algorithmic problem to determine a feasible schedule)

\[ \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & \ldots & k \\
J_1 & J_2 & J_3 & J_4 & \ldots & \end{array} \]

feasible

gaps can be compressed to left

**Case II** \( B_{k+1} = A_i \) for some \( i < k+1 \)

Move \( A_i \) to the end, and \( k, k+1 \)

Look at the set of jobs ignoring the last col.
Can we add the next most profitable element and maintain feasibility?

\((S, M)\) — family of “feasible subsets” and \(M\) can be very large \(M \leq 2^S\), so maintaining \(M\) explicitly will be extremely inefficient.

Instead we characterise the subsets \(T\), \(M\) using some property

\[\text{Maximal Spanning Trees: } \text{no cycles}\]

(Does the matroid theorem extend to minimisation functions, specifically Minimal Spanning Trees?)

At any stage of the MSF problem we have a set \(y\) trees. We add the next edge if it doesn’t induce a cycle.
Obs We can add the next edge \((x, y)\) iff \(x\) and \(y\) belong to different connected components.

How quickly can we do this test? What is the right data structure?

\(C()\): defines the component

\(C(x) \neq C(y)\)?

If \(C(x) \neq C(y)\) then we must add the edge and combine the components.

Label the vertices with the component nos (initially, \(-1, 2, \ldots n\)). When you join, change the labels of one component.
Test \quad C(x) = C(y)

Find \quad O(1) \text{ time}

Join : \quad O(\min(|K_x|, |K_y|)) \quad \text{size of smaller component}

Union

What is the overall cost for \(2^m\) tests \(n-1\)Joins?

\(m = |E|\)

\(|V| = n\)

\(m \text{ Finds} \quad \text{and} \quad n \text{ unions} \quad O(m) \quad \Rightarrow \quad O(n^2) ?\)

How often does a specific vertex change its label.