Algorithms
COMP3121/3821/9101/9801
6. THE GREEDY METHOD

School of Computer Science and Engineering
University of New South Wales Sydney

The Greedy Method

Search for a solution of maximal reward (/minimal cost)

- Define a “quality” measure on problem’s elements
- Build a solution step by step by adding elements of highest quality

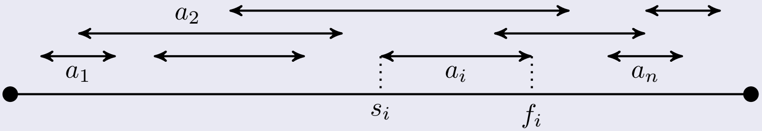
Optimality proof method (exchange argument)

- Pick any solution S
- morph it by swapping elements with higher quality ones
- show that any swap leads to a solution with higher reward
- stop when we arrive at the greedy solution G
- Conclude that the reward of G is larger than S

The Greedy Method

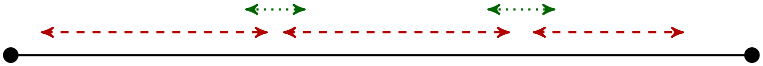
Activity selection problem

- **Setting:** A list of activities a_i, $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i. No two activities can take place simultaneously.
- **Task:** Find a *maximum size* subset of compatible activities.



Attempt 1: always choose the shortest activity which does not conflict with the previously chosen activities, remove the conflicting activities and repeat?

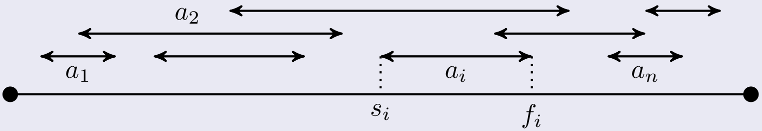
- Counter-example: chosen activities dotted, conflicting dashed



The Greedy Method

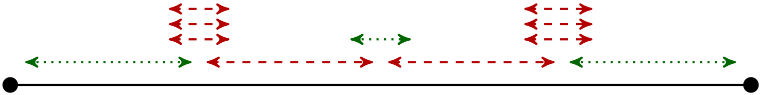
Activity selection problem

- **Setting:** A list of activities a_i, $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i. No two activities can take place simultaneously.
  - **Task:** Find a *maximum size* subset of compatible activities.



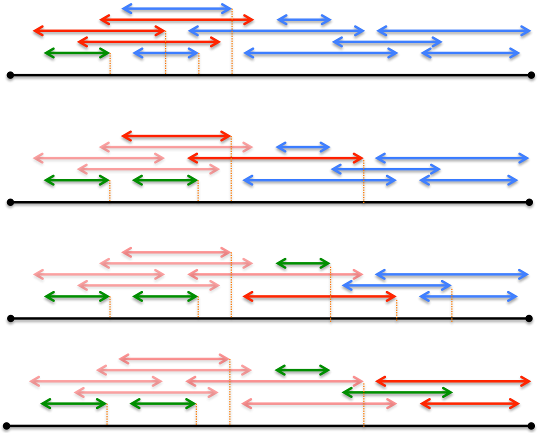
Attempt 2: Maybe we should always choose an activity which conflicts with the fewest possible number of the remaining activities? It may appear that in this way we minimally restrict our next choice....

- Counter-example: chosen activities dotted, conflicting dashed



The Greedy Method

The correct solution: Among the activities which do not conflict with the previously chosen activities always chose the one with the earliest end time.

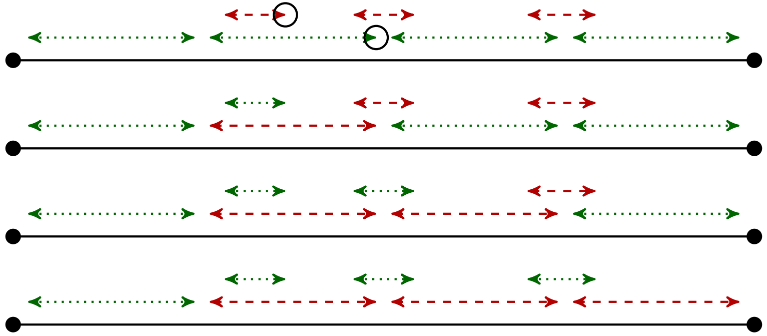


Activity selection problem

Optimality proof

Claim: any solution S has $\leq$ number of activities than the greedy solution G:

- Find the first place where the chosen activity violates the greedy choice.
- Show that replacing that activity with the greedy choice produces a non conflicting selection S' with the same number of activities.
- Continue until all activities match thoses in the greedy solution G



The Greedy Method

- What is the time complexity of the algorithm?
- We represent activities by ordered pairs of their starting and their finishing times and sort them in an increasing order of their finishing time (the second coordinate).
- This takes $O(n \log n)$ time.
- We go through such a sorted list in order, looking for the first activity whose starting time is after the finishing time of the last taken activity.
- Every activity is handled only once, so this part of the algorithm takes $O(n)$ time.
- Thus, the algorithm runs in total time $O(n \log n)$.

The Greedy Method

ACTIVITY SELECTION PROBLEM II

- **Instance:** A list of activities a_i, $(1 \leq i \leq n)$ with starting times s_i and finishing times $f_i = s_i + d$; thus, all activities are of the same duration. No two activities can take place simultaneously.
- **Task:** Find a subset of compatible activities of *maximal total duration*.

- **Solution:** Since all activities are of the same duration, this is equivalent to finding a selection with a largest number of non conflicting activities, i.e., the previous problem.

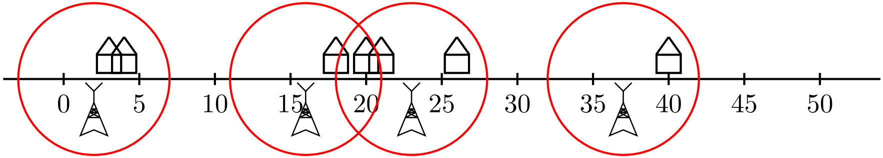
- **Question:** What happens if the activities are not all of the same duration?

- Greedy strategy no longer works - we will need a more sophisticated technique.

More practice problems for the Greedy Method

Placing base stations on a line

- **Setting:** Along the straight road from Loololong to Goolagong houses are scattered sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstra's cell base station is 5km.
- **Task:** Design an algorithm for placing the minimal number of base stations alongside the road, that is sufficient to cover all houses.



- Hint: The leftmost house needs to be covered. Where is best to put a tower which will cover this particular house?

More practice problems for the Greedy Method

- Once again, along the long, straight road from Loololong (on the West) to Goolagong (on the East) houses are scattered quite sparsely, sometimes with long gaps between two consecutive houses. Telstra must provide mobile phone service to people who live alongside the road, and the range of Telstra's cell base station is 5km.
- One of Telstra's engineer started with the house closest to Loololong and put a tower 5km away to the East. He then found the westmost house not already in the range of the tower and placed another tower 5 km to the East of it and continued in this way till he reached Goolagong.
- His junior associate did exactly the same but starting from the East and moving westwards and claimed that his method required fewer towers.
- Is there a placement of houses for which the associate is right?
- Hint: Both methods are greedy and are easy to prove that they produce optimal solutions.
- Can two optimal solutions have different number of elements?

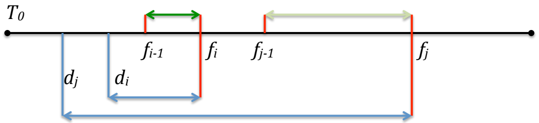
More practice problems for the Greedy Method

Minimising job lateness

- **Setting:** A start time T_0 and a list of jobs a_i, $(1 \leq i \leq n)$, with duration times t_i and deadlines d_i. Only one job can be performed at any time; all jobs have to be completed. If a job a_i is completed at a finishing time $f_i > d_i$ then we say that it has incurred lateness $l_i = f_i - d_i$.
- **Task:** Schedule all the jobs so that the lateness of the job with the largest lateness is minimised.

- **Solution:** Ignore job durations and schedule jobs in the increasing order of deadlines.

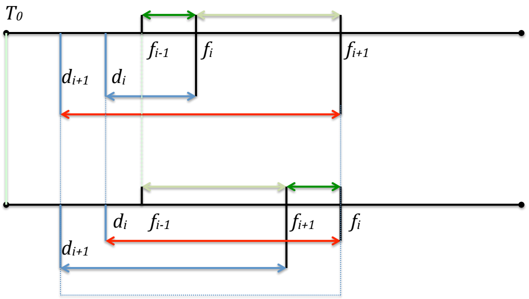
- **Optimality:** Consider any optimal solution. We say that jobs a_i and jobs a_j form an inversion if job a_i is scheduled before job a_j but $d_j < d_i$.



More practice problems for the Greedy Method

Minimising job lateness

- We will show that there exists a scheduling without inversions which is also optimal.
- Recall the `BubbleSort`: if we manage to eliminate all inversions between adjacent jobs, eventually all the inversions will be eliminated.



- Note that swapping adjacent inverted jobs reduces the larger lateness!

More practice problems for the Greedy Method

- Given two sequences of letters A and B, find if B is a subsequence of A in the sense that one can delete some letters from A and obtain the sequence B.
- There is a line of 111 stalls, some of which lack a roof and need to be covered with boards. You can use up to 11 boards, each of which may cover any number of consecutive stalls. Cover all the necessary stalls, while covering as few total stalls as possible.
    - One way: cover all the roofless stalls with a single piece of wood and see which part of it is best to excise;
    - Another way: cover each roofless stall with a separate, fitting piece of wood and then see what consecutive pieces should be replaced by a single larger piece.

More practice problems for the Greedy Method

Tape storage

- **Setting:** A list of n files f_i of lengths l_i which have to be stored on a tape. Each file is equally likely to be needed. To retrieve a file, one must start from the beginning of the tape and scan it until the tape is found and read.
- **Task:** Order the files on the tape so that the average (expected) retrieval time is minimised.

- **Solution:** If the files are stored in order $l_1, l_2, \dots l_n$, then the expected time is proportional to

$$\begin{aligned} l_1 + (l_1 + l_2) + (l_1 + l_2 + l_3) + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n) = \\ nl_1 + (n-1)l_2 + (n-2)l_3 + \ldots + 2l_{n-1} + l_n \end{aligned}$$

- This is minimised if $l_1 \leq l_2 \leq l_3 \leq \ldots \leq l_n$.

More practice problems for the Greedy Method

Tape storage II

- **Setting:** A list of n files f_i of lengths l_i and probabilities to be needed p_i, $\sum_{i=1}^{n} p_i = 1$, which have to be stored on a tape. To retrieve a file, one must start from the beginning of the tape and scan it until the tape is found and read.
- **Task:** Order the files on the tape so that the expected retrieval time is minimised.

- **Solution:** If the files are stored in order $l_1, l_2, \ldots l_n$, then the expected time is proportional to

$$p_1 l_1 + (l_1 + l_2)p_2 + (l_1 + l_2 + l_3)p_3 + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n)p_n$$

- We now show that this is minimised if the files are ordered in a decreasing order of values of the ratio p_i/l_i.

More practice problems for the Greedy Method

- Let us see what happens if we swap to adjacent files f_k and f_{k+1}.
- The expected time before the swap and after the swap are, respectively,

$$\begin{aligned} E =& l_1 p_1 + (l_1 + l_2)p_2 + (l_1 + l_2 + l_3)p_3 + \ldots + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_k)p_k \\ &+ (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_k + l_{k+1})p_{k+1} + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n)p_n \end{aligned}$$

and

$$\begin{aligned} E' =& l_1 p_1 + (l_1 + l_2)p_2 + (l_1 + l_2 + l_3)p_3 + \ldots + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_{k+1})p_{k+1} \\ &+ (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_{k+1} + l_k)p_k + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n)p_n \end{aligned}$$

- Thus, $E - E' = l_k p_{k+1} - l_{k+1} p_k$, which is positive just in case $l_k p_{k+1} > l_{k+1} p_k$, i.e., if $p_k / l_k < p_{k+1}/l_{k+1}$.

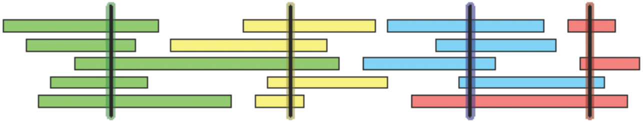
- Consequently, $E > E'$ if and only if $p_k/l_k < p_{k+1}/l_{k+1}$, which means that the swap decreases the expected time just in case $p_k/l_k < p_{k+1}/l_{k+1}$, i.e., if there is an inversion: a file f_{i+1} with a larger ratio p_{k+1}/l_{k+1} has been put after a file f_i with a smaller ratio p_k/l_k.

- For as long as there are inversions there will be inversions of consecutive files and swapping will reduce the expected time. Consequently, the optimal solution is the one with no inversions.

More practice problems for the Greedy Method

Interval stabbing

- **Setting:** Let X be a set of n intervals on the real line. We say that a set P of points stabs X if every interval in X contains at least one point in P; see the figure below. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X.
- **Task:** Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X.



A set of intervals stabbed by four points (shown here as vertical segments)

- Hint: the interval which ends the earliest has to be stabbed.
- What is the best place to stabb it?

More practice problems for the Greedy Method

- Assume denominations of your $n+1$ coins are $1, c, c^2, c^3, \ldots, c^n$ for some integer $c > 1$. Design a greedy algorithm which, given any amount, pays it with a minimal number of coins.

- Assume you have \$2, \$1, 50c, 20c, 10c and 5c coins to pay for your lunch. Design an algorithm that, given the amount that is a multiple of 5c, pays it with a minimal number of coins.

- Give an example of a set of denominations containing the single cent coin for which the greedy algorithm does not always produce an optimal solution.

The Greedy Method

0-1 knapsack problem

- **Setting:** A list of weights w_i and values v_i for **discrete items** a_i, $1 \leq i \leq n$, and a maximal weight limit W of your knapsack.
- **Task:** Find a subset S of all items available such that its weight does not exceed W and its value is maximal.

- Can we always choose the item with the highest value per unit weight?
- Assume there are just three items with weights and values: (10kg, \$60), (20kg, \$100), (30kg, \$120) and a knapsack of capacity W = 50kg.
- Greedy would choose (10kg, \$60) and (20kg, \$100), while the optimal solution is to take (20kg, \$100) and (30kg, \$120)!
- So when does the Greedy Strategy work??
- Unfortunately there is no easy rule...

More practice problems for the Greedy Method

Array Merging

- **Setting:** Assume you are given n sorted arrays of different sizes. You are allowed to merge any two arrays into a single new sorted array and proceed in this manner until only one array is left.
- **Task:** Design an algorithm that achieves this task and uses minimal total number of moves of elements of the arrays and give an informal justification why your algorithm is optimal.

- This problem is somewhat related to the next application of the Greedy method, which is, arguably, among the most important applications of the greedy method!

That's All, Folks!!