Algorithms: COMP3121/3821/9101/9801

Aleks Ignjatović

School of Computer Science and Engineering
University of New South Wales

TOPIC 4: THE GREEDY METHOD

Activity selection problem

- **Instance:** 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.

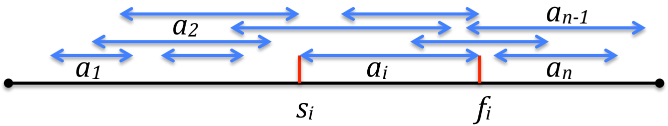
Activity selection problem

- **Instance:** 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.

Activity selection problem

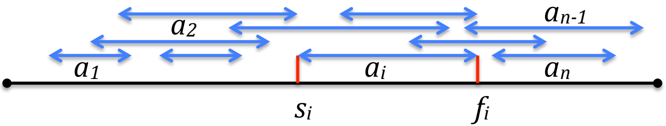
- **Instance:** A list of activities a_i, $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i. No two activities can take place simultaneously.
- **Task:** Find a *maximum size* subset of compatible activities.



Activity selection problem

- **Instance:** 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.

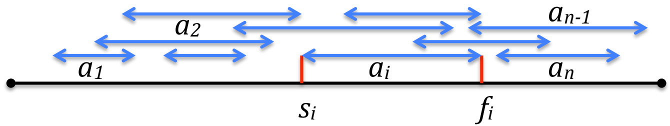
- **Task:** Find a *maximum size* subset of compatible activities.



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

Activity selection problem

- **Instance:** 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.

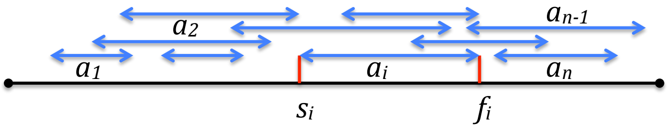


Attempt 1: always choose the shortest activity which does not conflict with the previously chosen activities, remove the conflicting activities and repeat?

Activity selection problem

- **Instance:** 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?

- The above figure shows this does not work...
(chosen activities in green, conflicting in red)

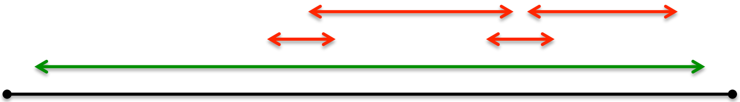
Activity selection problem

- **Instance:** 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.

Activity selection problem

- **Instance:** 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.

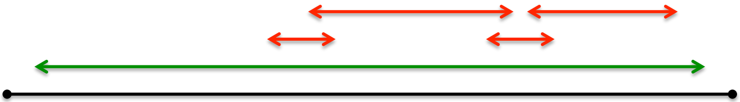
- **Attempt 2:** Maybe among the activities which do not conflict with the previously chosen activities we should always choose the one with the earliest possible start?



Activity selection problem

- **Instance:** 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 among the activities which do not conflict with the previously chosen activities we should always choose the one with the earliest possible start?



- The above figure shows this does not work either...

Activity selection problem.

- **Instance:** 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.

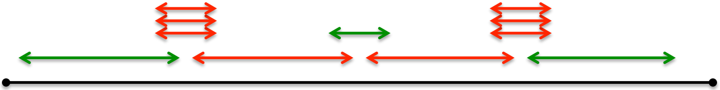
- **Attempt 3:** Maybe we should always choose an activity which conflicts with the fewest possible number of the remaining activities? It may appear that in this way we minimally restrict our next choice....

Activity selection problem.

- **Instance:** 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 3:** 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....

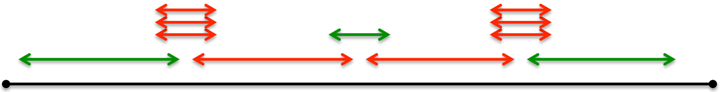


Activity selection problem.

- **Instance:** 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 3:** 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....



- As appealing the idea is, the above figure shows this again does not work ...

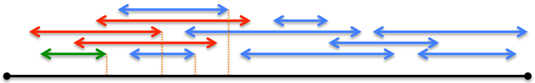
Activity selection problem

- **Instance:** 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.

Activity selection problem

- **Instance:** 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.

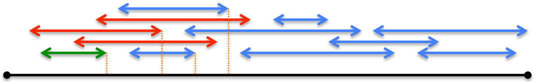
- **The correct solution:** Among the activities which do not conflict with the previously chosen activities always chose the one with the earliest end time.

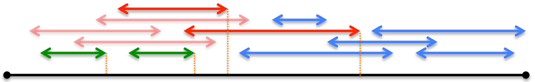


Activity selection problem

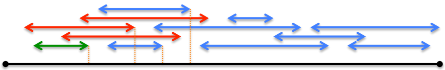
- **Instance:** A list of activities a_i, $(1 \leq i \leq n)$ with starting times s_i and finishing times f_i. No two activities can take place simultaneously.
- **Task:** Find a *maximum size* subset of compatible activities.

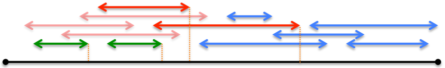
- **The correct solution:** Among the activities which do not conflict with the previously chosen activities always chose the one with the earliest end time.



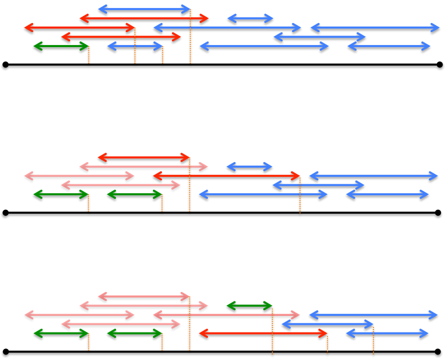


Activity selection problem

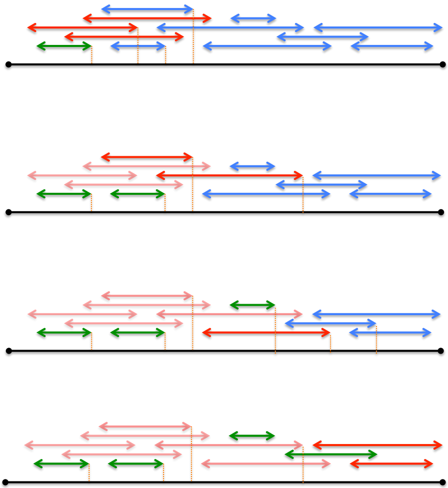




Activity selection problem

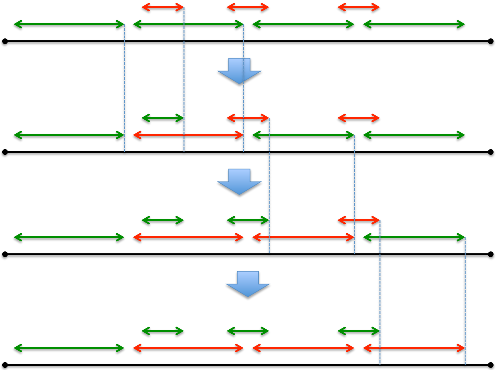


Activity selection problem



The Greedy Method - proving optimality of a solution

- Transforming any optimal solution to the greedy solution with equal number of activities: find the first place where the chosen activity violates the greedy choice and show that replacing that activity with the greedy choice produces a non conflicting selection with the same number of activities. Continue in this manner till you "morph" your optimal solution into the greedy solution, thus proving the greedy solution is also optimal.



The Greedy Method

- What is the time complexity of the algorithm?

- What is the time complexity of the algorithm?
- We make a copy of the list of all activities and sort the list in an increasing order of their finishing time; this takes $O(n \log n)$ time.

The Greedy Method

- What is the time complexity of the algorithm?
- We make a copy of the list of all activities and sort the list in an increasing order of their finishing time; this takes $O(n \log n)$ time.
- We go through such a sorted list in order; for each activity we encounter we use the original list to see if this activity has a starting time which is after the finishing time of the last taken activity.

The Greedy Method

- What is the time complexity of the algorithm?
- We make a copy of the list of all activities and sort the list in an increasing order of their finishing time; this takes $O(n \log n)$ time.
- We go through such a sorted list in order; for each activity we encounter we use the original list to see if this activity has a starting time which 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.

The Greedy Method

- What is the time complexity of the algorithm?
- We make a copy of the list of all activities and sort the list in an increasing order of their finishing time; this takes $O(n \log n)$ time.
- We go through such a sorted list in order; for each activity we encounter we use the original list to see if this activity has a starting time which 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 $O(n \log n)$.

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.

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*.

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.

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?

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.

Minimising job lateness

- **Instance:** 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 $l_i = f_i - d_i$.

Minimising job lateness

- **Instance:** 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.

Minimising job lateness

- **Instance:** 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.

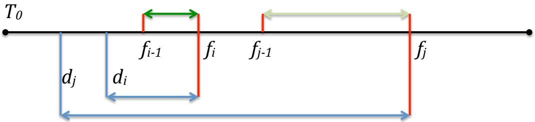
Minimising job lateness

- **Instance:** 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 $d_j < d_i$.



Minimising job lateness

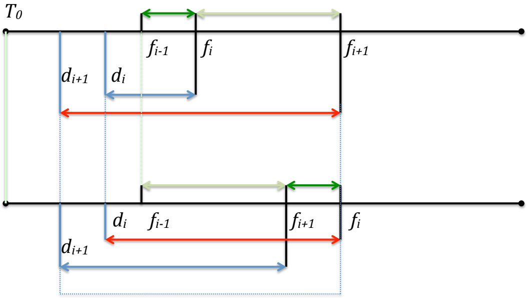
- We will show that there exists a scheduling without inversions which is also optimal.

Minimising job lateness

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

Minimising job lateness

- We will show that there exists a scheduling without inversions which is also optimal.
- Recall the Bubble Sort: 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!

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.

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 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.

Tape storage

- **Instance:** 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.

Tape storage

- **Instance:** 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.

Tape storage

- **Instance:** 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 $l_1, l_2, \ldots l_n$, then the expected time is proportional to

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

Tape storage

- **Instance:** 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 $l_1, l_2, \ldots l_n$, then the expected time is proportional to

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

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

Tape storrage II

- **Instance:** 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 fund and read.

Tape storrage II

- **Instance:** 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 fund and read.

- **Task:** Order the files on the tape so that the expected retrieval time is minimised.

Tape storrage II

- **Instance:** 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 fund 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

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

Tape storrage II

- **Instance:** 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 fund 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.

The Greedy Method

- Let us see what happens if we swap to adjacent files f_k and f_{k+1}.

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,

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,

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

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,

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

and

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

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,

$$
\begin{aligned}
E = & l_1p_1 + (l_1 + l_2)p_2 + (l_1 + l_2 + l_3)p_3 + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_k)p_k + \ldots \\
& + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_k + l_{k+1})p_{k+1} + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n)p_n
\end{aligned}
$$

and

$$
\begin{aligned}
E' = & l_1p_1 + (l_1 + l_2)p_2 + (l_1 + l_2 + l_3)p_3 + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_{k+1})p_{k+1} + \ldots \\
& + (l_1 + l_2 + l_3 + \ldots + l_{k-1} + l_{k+1} + l_k)p_k + \ldots + (l_1 + l_2 + l_3 + \ldots + l_n)p_n
\end{aligned}
$$

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

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,

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

and

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

- 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.

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,

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

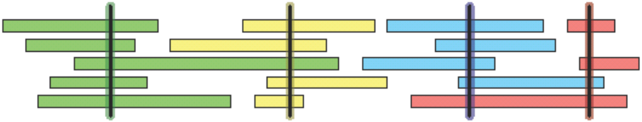
and

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

- 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.

The Greedy Method

- 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. Describe and analyse an efficient algorithm to compute the smallest set of points that stabs X. 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.



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

- 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. Design an algorithm that achieves this task and uses minimal total number of moves of elements of the arrays. Give an informal justification why your algorithm is optimal.

- Along the long, straight road from Loololong to Goolagong 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 Telstras cell base station is 5km. Design an algorithm for placing the minimal number of base stations alongside the road, that is sufficient to cover all houses.

Using the Greedy Method to derive various properties of the object constructed

- 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 Telstras cell base station is 5km.

Using the Greedy Method to derive various properties of the object constructed

- 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 Telstras 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.

Using the Greedy Method to derive various properties of the object constructed

- 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 Telstras 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.

Using the Greedy Method to derive various properties of the object constructed

- 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 Telstras 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?

- 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.

The Greedy Method

- 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.

- 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.

- 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.

- 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 - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.
- However this is not an economical way: all the symbols have codes of equal length but the symbols are not equally frequent.

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.
- However this is not an economical way: all the symbols have codes of equal length but the symbols are not equally frequent.
- One would prefer an encoding in which frequent symbols such as 'a', 'e', 'i' or 't' have short codes while infrequent ones, such as 'w','x' and 'y' can have longer codes.

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.
- However this is not an economical way: all the symbols have codes of equal length but the symbols are not equally frequent.
- One would prefer an encoding in which frequent symbols such as 'a', 'e', 'i' or 't' have short codes while infrequent ones, such as 'w','x' and 'y' can have longer codes.
- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.
- However this is not an economical way: all the symbols have codes of equal length but the symbols are not equally frequent.
- One would prefer an encoding in which frequent symbols such as 'a', 'e', 'i' or 't' have short codes while infrequent ones, such as 'w','x' and 'y' can have longer codes.
- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?
- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.

The Greedy Method - Huffman code

- Assume you are given a set of symbols, for example the English alphabet plus punctuation marks and a blank space (to be used between words).
- You want to encode these symbols using binary strings, so that sequences of such symbols can be decoded in an unambiguous way.
- One way of doing so is to reserve bit strings of equal and sufficient length, given the number of distinct symbols to be encoded; say if you have 26 letters plus 6 punctuation symbols, you would need strings of length 5 ($2^5 = 32$).
- To decode a piece of text you would partition the bit stream into groups of 5 bits and use a lookup table to decode the text.
- However this is not an economical way: all the symbols have codes of equal length but the symbols are not equally frequent.
- One would prefer an encoding in which frequent symbols such as 'a', 'e', 'i' or 't' have short codes while infrequent ones, such as 'w','x' and 'y' can have longer codes.
- However, if the codes are of variable length, then how can we partition a bitstream UNIQUELY into segments each corresponding to a code?
- One way of insuring unique readability of codes from a single bitstream is to ensure that no code of a symbol is a prefix of a code for another symbol.
- Codes with such property are called *the prefix codes*.

The Greedy Method - Huffman code

- We can now formulate the problem:

  Given the frequencies (probabilities of occurrences) of each symbol, design an optimal prefix code, i.e., a prefix code such that the expected length of an encoded text is as small as possible.

- Note that this amounts to saying that the *average* number of bits per symbol in an “average” text is as small as possible.

0-1 knapsack problem

- **Instance:** 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.

0-1 knapsack problem

- **Instance:** 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.

0-1 knapsack problem

- **Instance:** 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?

0-1 knapsack problem

- **Instance:** 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.

0-1 knapsack problem

- **Instance:** 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)!

0-1 knapsack problem

- **Instance:** 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??

0-1 knapsack problem

- **Instance:** 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...