Algorithms

Selection

Given an array A of n elements, a common task is to find the ith order statistic - that is, the ith largest element of A. To do this, we use a modified version of merge sort, combining two ideas:

- Firstly, we consider the merge sort partition algorithm. We select some element $x = A[0]$, then by holding two pointers we maintain the invariant $A[1..l) < x \leq A[r..n)$ until $l = r$. Thus we get that x is the $(l-1)$th order statistic of A. We then will have either solved the problem, or we will have found a subproblem containing the solution, of (ideally) half the size of the original problem. In the worst case this is still $O(n^2)$ if we choose the partitions badly however.
- Secondly, we find a method of getting an approximation to the median within a constant factor. To do this we use the median of medians method: divide A into $\lfloor \frac{n}{5} \rfloor$ subarrays, each of size around 5. For each we may find the median in constant time (e.g. run insertion sort on each), and then we find the median of this new list. This itself uses the algorithm described above.

By using the second algorithm within the first algorithm, we ultimately get a linear time algorithm. Note that we cannot use the master theorem here, because we have distinct recursive calls that may overlap.

Change-Making

Input: Positive integers $1 = x_1 < x_2 < \cdots < x_n$ and v Task: Given an unlimited supply of coins of denominations $x_1, \cdots, x_n$, find the minimum number of coins needed to sum up to v.

The solution to this comes from observing the optimal substructure that if we have a solution to make v with a minimal multiset of coins, then if x_i is used, we must also have a solution to make $v - x_i$ with a minimal multiset. Otherwise we would be able to improve by using x_i combined with an improved number.

Thus $C[u] = 1 + \min\{C[u - x_i] \mid i \in [1..n] \wedge u \geq x_i\}$.

Knapsack

Input: Positive integers $w_1, \cdots, w_n$, values $v_1, \cdots, v_n$, and W. Task: Find the collection of items that maximises the total value while remaining under the total weight limit.

In the case where we can only pick one of each item, any optimal solution for weight w on the first j items will be a solution for either weight w on the first

$j-1$ (if the jth item is not useful), or for $w-w_j$ on the first $j-1$ (if it is useful). $K[w,j] = \max(K[w-w_j, j-1] + v_j, K[w, j-1])$.

In the case where we have unlimited, we instead have $K[w] = \max\{K[w-w_j] + v_j \,|\, j \in [1..n], w_j \leq w\}$.

Longest Increasing Subsequence

Input: A sequence of numbers $\langle a_1, \cdots, a_n \rangle$. Task: Find a longest increasing subsequence of this sequence.

With $L[j]$ the length of the longest increasing subsequence ending at exactly j, $L[j] = 1 + \max\{L[i] \,|\, i < j, a_i < a_j\}$.

Edit Distance

Input: Two strings $x[1..n]$ and $y[1..m]$. Task: Find the shortest sequence of edits to transform x to y. An edit can be a deletion, insertion, or substitution.

The main observation to be made here is that with any alignment of x and y (a table with each character of x, or a blank character, over each character of y, or a blank character), we get a sequence of edits which transform one to the other:

- If a character of x is over a blank space, a deletion removes this.
- If a character of y is below a blank space, an insertion removes this.
- If the characters are different, a substitution fixes this.
- If the characters are the same, nothing must occur.

Thus we might consider the problem as simply minimising the number of mismatched columns. If $E[i,j]$ is the edit distance of $x[1..i]$ to $y[1..j]$, we may formulate the problem in terms of the final column:

- If $x[i]$ is over a blank space, then the cost is $1 + E[i-1, j]$.
- If $y[j]$ is under a blank space, then the cost is $1 + E[i, j-1]$.
- If $x[i] = y[j]$ and both are in the same column, the cost is $E[i-1, j-1]$.
- If $x[i] \neq y[j]$ and both are in the same column, the cost is $1 + E[i-1, j-1]$.

Longest Simple Path

Input: A weighted directed graph $G = (V, E)$ and two vertices $a, c \in V$. Task: Find the length of th elongest simple path from a to c.

This is an instance where optimal substructure fails. With $l[u,v]$ the longest path from u to v, we don't have that $l[a,c] = l[a,b] + l[b,c]$ for all $b \in V$. This is because we potentially violate simplicity if both paths used in the subproblems overlap. Simplicity entails that the path cannot visit the same vertex twice.