Divide and Conquer

Coin Puzzle (**Ternary Search**)
Problem:
Given 27 coins of the same denomination, we know that one of them is counterfeit and lighter than the others. Using a pan balance only three times, find the counterfeit coin.

Solution:
Split the coins into three groups of 9. Weigh two (arbitrarily).

- If the two groups are equal, then the counterfeit must be in the remaining group.
- If one of the groups are lighter, then the counterfeit is in that group.

This would reduce 27 to 9 to 3 to 1. There are three weightings.

Maximum Median (**Binary Search**)
Problem:
Given $2n-1$ sorted integers, and the ability to add 1 to one of her numbers for k days, find the largest possible median. Denote $A[n]$ as the original median.

Solution:
We know that our solution will lie somewhere in the range of $A[n] \rightarrow A[n]+k$. Note that once we test a value, it's true/false response creates a monotonic property. So we can binary search this range, with some validating function.

Inversions (**Mergesort**)
Problem:
Given some array A, we want to count the number of pairs (i, j), such that $j > i$, and that $A[j] < A[i]$

Solution:
During the combine step of mergesort, we can actually count how many elements we gather from the right hand side into the left hand side. This gives us the ability to count how many inversions we have.

Greedy

Frozen Yoghurt I
Problem:
There are n flavours of frozen yoghurt at your local shop. The ith flavour is dispensed from a machine of capacity c_i litres and contributes d_i/c_i deliciousness per litre. Maximise total deliciousness with a tub of capacity C litres.

Solution:
Greedily fill in the yoghurt with descending order of deliciousness. Very easy to stays ahead argument.

Fractional Knapsack
Problem:
There is a list of n items described by their weights w_i and values v_i. We have a knapsack of weight W. You can take any fractional amount of any item.

Solution:
Refer to above. Same as yoghurt question. Here, the greedy works because it's continuous amounts of items – if it were to be discrete, it would be invalid, as we don't exactly know locally, whether the local optimal choice leads to a global one.

Activity Selection
Problem:
A list of n activities, with starting times s_i and finishing times f_i. Find the maximum size subset of compatible activities (activity starting and finish times cannot overlap).

Solution:
We essentially want to always maintain the maximum size buffer remaining after choosing an activity. This leads to always choosing the activity with the earliest end time, such that the problem constraints hold.

Cell Towers:
Problem:
Given a list of houses which are d_i kilometres west of some point B, we must place cell towers (that cover 5km of range east and west), such as to minimise the amount of cell towers.

Solution:
Put a new cell tower 5km west of the first uncovered house. We can prove this using greedy stays ahead.

Tape Storage

Problem:

A list of n files of lengths l_i and probabilities to be needed p_i, 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 file is found and read.

Solution:

Consider two different storage methods, with an adjacent inversion. Considering the adjacent inversion will show you:

$$\frac{p_{k+1}}{l_{k+1}} > \frac{p_k}{l_k}$$

Thus, we order in descending ratio of p_i/l_i. *If you're given a problem with seemingly two elements you must order by, consider this problem.*

Minimising Job Lateness

Problem:

Given a start time T_0 and a list of n jobs, with duration time t_i and deadlines d_i, schedule jobs to minimise lateness.

Solution:

Lateness is only defined by the finishing time relative to the deadline. Sort by ascending order of deadlines, and complete. Can be proven by inversions.

k-clustering of maximum spacing

Problem:

Given a complete graph G with weighted edges representing distances between two vertices, partition the vertices of G into k disjoint subsets so that the minimal distance between two points belonging to two different sets is as large as possible.

Solution:

Perform Kruskal's Algorithm until you have k connected components. The optimality can be shown by saying that if there was a larger edge between two components a and b, it would have been chosen by Kruskal's already.

Flow Networks

Movie Rental

Problem:

Given k movies in stock, with m_i copies of movie i, and n customers who can only rent 5 movies at a time, which have a specified subset of the k movies they want to see, design an algorithm which dispatches the largest number of movies.

Solution:

- Source links to n vertices that represent customers. Link source with edge capacity 5 to these customers.
- Sink links to k movies with edge capacity m_i
- Customers link to their preferred movies with capacity 1.

Run Edmonds-Karp.

Cargo Allocation

Problem:

The storage space of a ship is in the form of n rows and n columns. Some cells are used by support pillars, and hence, cannot be used for storage. Given values for each row and column such that we cannot each these values, find the maximum possible weight of cargo we can put on the ship.

Solution:

- Connect source with row vertices, and sink with column vertices, for each n rows and m columns.
- Row to column connections are defined by capacity that exists within them.
- For example, for column 1 and row 1's edge, we should have $C(1,1)$, where C represents the capacity matrix.

Run Edmonds-Karp.

Vertex-Disjoint Paths

Problem:

Given n vertices and m edges, as well as r red vertices, b blue vertices and $n - r - b$ black vertices, find the maximum number of vertex-disjoint paths (non-intersecting paths)

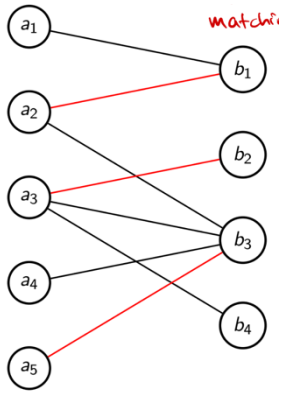
Solution:

Just create a flow network with red connected to super source, blue connected to super sink and then construct the rest of the graph as per usual. Max flow will represent the maximum amount of paths we can send.

Maximal Bipartite Matching

Problem:

Given a bipartite graph G, find the size of the maximum matching (where a matching is a subset of edges where each vertex of the graph only belongs to at most one edge in the matching).



Solution:

Super source the left-hand vertices, super sink the right-hand vertices, run Edmonds-Karp. Can be very useful for questions that are bipartite in nature, so look out for them.

Job Centre

Problem:

There are k recognised qualifications in Australia. There are n unemployed people, holding some set of available qualifications. There are also m jobs, which require some set of qualifications. Find the maximum amount of people that can be employed.

Solution:

Maximum bipartite matching – super source the unemployed, super sink the jobs, and create edge capacities of 1 with employees and jobs where the employees match the qualifications.

Dynamic Programming

This section assumes you have some experience with understanding problem definitions fast – I've only included recurrences for brevity.

Hopscotch

Problem:

Rui is playing hopscotch on a linear court. They start in square 0, and wish to get to square n. At any square, Rui can step one or two steps forward. Find how many different ways we can get to square n in linear time.

Solution:

Where i represents the $i-th$ square:

$$opt(i) = opt(i-1) + opt(i-2)$$

Longest Increasing Subsequence

Problem:

Given a sequence of real numbers $A[1, \dots, n]$, finding the longest increasing subsequence within this sequence.

Solution:

Where i represents the i-th index:

$$opt(i) = max\big(opt(j)\big) + 1$$

where $A[j] < A[i]$ and $j < i$.

Activity Selection

Problem:

Same as the greedy problem.

Solution:

With the input sorted by finishing time, and where i represents the $i-th$ activity:

$$opt(i) = (f_i - s_i) + max(t\,(j))$$

where $j < i, f_j < s_i$ for the max statement.

Making Change

Problem:

Given n types of coin denominations for integer values – make change for a given amount C, using the least number of coins possible.

Solution:

Where i represents trying to fill an amount i, with the least amount of coins:

$$\text{opt}(i) = \min\{\text{opt}(i - v_k) \mid 1 \le k \le n, v_k \le i\} + 1.$$

Integer Knapsack

Problem:

You have n items, given some weight w_i each, and value v_i. You have a knapsack of capacity C. Choose a combination of available items which all fit in the knapsack and whose value is as large as possible.

Solution:

Where i represents the i-th item, and where $opt(i)$ returns the maximum value of the combination using up to the i-th item:

$$\text{opt}(i) = \max_{1 \le k \le n, w_k \le i} \text{opt}\,(i - w_k) + v_k$$

0-1 Knapsack

Problem:

You have n items, the i-th of which has weight w_i and value v_i. All weights are *integers.* You also have a knapsack of capacity C. Given you are only allowed to take an item at most once, choose a combination that has the maximum value.

Solution:

Where i represents the capacity of our knapsack and k represents the usage up to the $k - th$ item, our recurrence is:

$$\text{opt}(i,k) = \max\left(\text{opt}(i,k-1), \text{opt}(i - w_k, k-1) + v_k\right)$$

Balanced Partition

Problem:

Given a set of n positive integers, partition these integers into two subsets such that the difference of the sum of the subsets is minimised.

Solution:

$$\Sigma_1 + \Sigma_2 = \Sigma$$

$$\Sigma_1 - \Sigma_2 = 2\left(\frac{\Sigma}{2} - \Sigma_2\right)$$

Thus, we find a subset S_2 such that the total sum is as close to $\frac{\Sigma}{2}$ as possible. (0-1 Knapsack)

Longest Common Subsequence

Problem:

Give two sequences S and S', find the length of the longest common subsequence of S and S'.

Solution:

For $0 \le i \le n$ and $0 \le j \le m$ let, and where I represents the i-th index of the sequence S and j represents the j-th index of the sequence S':

$$\text{opt}(i,j) = \begin{cases} \text{opt}(i-1,j-1) + 1 & \text{if } a_i = b_j \\ \max(\text{opt}(i-1,j), \text{opt}(i,j-1)) & \text{otherwise.} \end{cases}$$

The second case considers the best choice of discarding $j - 1$ or $i - 1$.

Shortest Common Subsequence

Problem:

Same as above, but shortest.

Solution:

Find the LCS, and then add back differing elements of the two subsequences in the right places, in any compatible order.

Edit Distance

Problem:

Given two text strings A of length n, and B of length m, we want to make A into B, using the least number of transformations possible. Each insertion, deletion and replacement have some cost c_I, c_D, c_R.

Solution:

For $0 \le i \le n$ and $0 \le j \le m$, where i represents the i-th letter of A and j represents the j-th letter of B:

$$\text{opt}(i,j) = \min \begin{cases} \text{opt}(i-1,j) + c_D \\ \text{opt}(i,j-1) + c_I \\ \begin{cases} \text{opt}(i-1,j-1) & \text{if } A[i] = B[j] \\ \text{opt}(i-1,j-1) + c_R & \text{if } A[i] \neq B[j]. \end{cases} \end{cases}$$

Deleting a word means we moved our i pointer forward, inserting a word means we moved our j pointer forward, and the two other cases are trivial.

Matrix Chain Multiplication

Problem:

For some sequences of matrices $A_1 A_2 A_3 \ldots A_n$, group them in such a way as to minimise the total number of multiplications needed to find the product matrix.

Solution:

For $0 \le i < j \le n$, where I and j represent indices of the matrices:

$$\text{opt}(i,j) = \min\{\text{opt}(i,k) + s_i s_k s_j + \text{opt}(k,j) \mid i < k < j\}.$$