3121 Problems

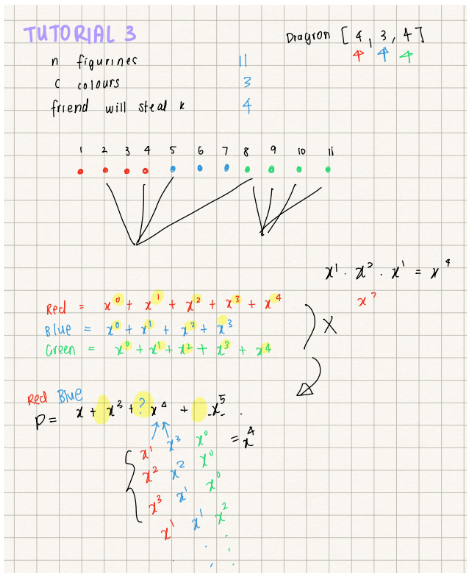
Chris Wong

1 Divide and Conquer

1.1 YearOfTheDragon

You have a collection of n coloured dragon figurines. Each figurine is coloured one of c colours. Design an $O(ck \log k)$ algorithm that finds the number of different ways of choosing k dragon figurines. Input is an array Dragon[1..c] and an integer k.

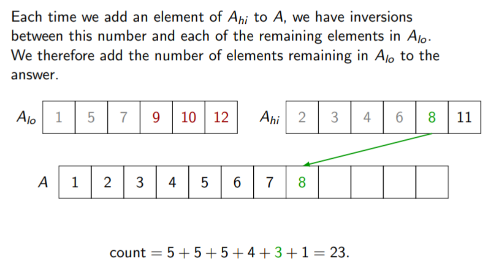
- **Intuition:** Recall that convolutions allows us to compute the sum of powers of $c_0 x^k$ that sum to k. Now we can map each colour as a polynomial and multiply all colours together. Now looking at $c_0 x^k$, c_0 will give us the number of ways to collect k figurines.


TUTORIAL 3
n figurines 11
c colours 3
friend will steal k 4
Dragon [4, 3, 4]
Red = x⁰ + x¹ + x² + x³ + x⁴
Blue = x⁰ + x¹ + x² + x³
Green = x⁰ + x¹ + x² + x³ + x⁴
x¹ · x² · x¹ = x⁴
Red Blue
P = x + x² + ? x⁴ + x⁵

1.2 CountingInversions

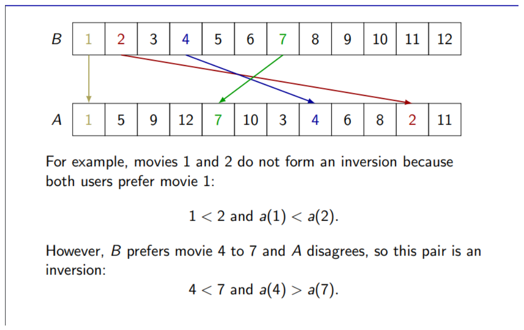
Suppose that you have m users ranking the same set of n movies. Count the number of inversions between two users A and B.

- **Inversion Definition:** An inversion is a pair (i, j) such that:
  - $i < j$ i.e B prefers i to j
  - $a(i) > a(j)$ i.e A prefers j to i
- **Brute Force Method:** Test each pair $i < j$ and add one to the total if $a(i) > a(j)$, this produces a quadratic time algorithm.
- **Merge Sort Modification:**
  - Split the array A into two equal parts. $A_{lo} = A[1..m]$ and $A_{hi} = A[m+1..n]$. The number of inversions is $I(A_{lo}) + I(A_{hi}) + I(A_{lo}, A_{hi})$
  - During our combine step, when we add an element of A_{hi} to A we add to the count the total of remaining elements in A_{lo}.



Each time we add an element of A_{hi} to A, we have inversions between this number and each of the remaining elements in A_{lo}. We therefore add the number of elements remaining in A_{lo} to the answer.

count = 5 + 5 + 5 + 4 + 4 + 3 + 1 = 23.



For example, movies 1 and 2 do not form an inversion because both users prefer movie 1:

$$1 < 2 \text{ and } a(1) < a(2).$$

However, B prefers movie 4 to 7 and A disagrees, so this pair is an inversion:

$$4 < 7 \text{ and } a(4) > a(7).$$

2 Greedy

2.1 ActivitySelection

Given a list of n activites, with starting times s_i and finishing times f_i. No two activities can take place simultaneously. Find a maximum size subset of compatible activities.

- **Intuition:** Since we want to pick as many activities that don't conflict, we can always pick the activity that ends the earliest and delete the conflicting ones.
- **Exchange Argument:** Consider a greedy solution $G = \{g_1, \ldots, g_n\}$ and an optimal solution $O = \{o_1, \ldots, o_n\}$. Assume that G differs from O at some index $k \leq n$ such that $g_k \neq$

o_k. Now exchanging g_k with o_k forms a new solution $O' = \{o_1, \dots, o_{k-1}, g_k, o_[k+1], \dots o_m\}$.

- O' stays a valid solution as g_k will not have any conflicting activities for o_1 to o_{k-1} since the G agrees with all of O except up to k.
- O' stays will not conflict with o_{k+1} since we choose g_k such that it is the activity that ends the earliest after o_{k-1}, thus if this did conflict with o_{k+1} then o_k would have also conflicted with o_{k+1} as there cannot be an earlier activity before g_k which contradicts our heuristic.
- Thus we can inductively apply this claim until O transforms into G.

2.2 RoadTrip

On a trip from Sydney to Melbourne, you can only drive 100km in a day. You have a list of n hotels sorted by its distance from Sydney and you need to stop at a hotel every night. Find a plan that stops for the fewest nights possible.

- **Intuition:** To stop for the fewest nights possible, we should get to Melbourne as fast as possible. We should then only stop at the hotel closest to 100km of where we will can go furthest.

- **Greedy Stays Ahead Argument:** We let g_i be the distance travelled from Sydney to the hotel on the ith night chosen by the greedy solution and similarly o_i by the optimal solution. We then label our solutions $G = \{g_1, \dots, g_{m'}\}$ and $O = \{o_1, \dots, o_m\}$, since O is optimal $m \leq m'$.
  - **Base case:** Since our greedy will choose the farthest hotel possible we denote that $g_1 \geq o_1$.
  - **Inductive Hypothesis:** Suppose, that for all $1 \leq i \leq k$ that $g_k \geq o_k$.
  - **Inductive Step:** Since $g_k \geq o_k$ we know that $g_k + 100 \geq o_k + 100$ such that the hotels between our greedy and optimal will be the same more importantly the optimal solution does not have access to a hotel further than the greedy. Therefore $g_{k+1} \geq o_{k+1}$.
  - We now conclude that $g_m \geq o_m$, such that G requires at most as many nights as O, therefore $m' \leq m$ and so $m = m'$

2.3 IntervalCover

You are given a set $I = \{I_1, \dots, I_n\}$ of n intervals on the real line. Find a subset $I' \subseteq I$ satisfying. If there is an interval $I_i \in I$ that covers x, then there is an interval $I_j \in I'$ that also covers x. Design an $O(n \log n)$ algorithm to find the minimum subset I'.

- **Intuition:** Start with the leftmost starting point and choose the interval whose ending point is as far right as possible out of all the intervals that includes the leftmost starting point.

- **Exchange Argument:** Let X be the interval that includes the leftmost starting point a_X and whose endpoint is as rightmost as possible. Let S be any optimal solution that does not include interval X, and let Y be an interval in S that includes point a_X but not b_X. Let a_Y be its starting point and b_Y be its endpoint. Since X is the interval with the left most starting point, we have that $a_X \leq a_Y$. Since Y includes the point a_X, we also have $a_Y \leq a_X$; therefore, $a_X = a_Y$. Since Y does not include the point b_X, we have that $b_Y \leq b_X$. Hence, Y is contained completely inside X. Now consider $S' = (S \setminus \{Y\}) \cup \{X\}$. Since $S \setminus \{Y\}$ covers every point, except for points in Y, all points in Y are covered by X, and $|S| = |S'|$ therefore S' is an optimal solution. The claim inductively transforms S into G.

3 Dynamic Programming

3.1 LongestIncreasingSubsequence

Given a sequence of n real numbers $A[1..n]$, determine a subsequence of maximum length, in which the values in the subsequence are strictly increasing.

- **Intuition:** We can think of the problem as which previous sequence of LIS do we want to extend off of.

Solution

Subproblems: for each $1 \leq i \leq n$, let $Q(i)$ be the problem of determining opt(i), the maximum length of an increasing subsequence of $A[1..i]$ which ends with $A[i]$.

Recurrence: for $i > 1$,

$$\text{opt}(i) = 1 + \max_{\substack{j<i \\ A[j]<A[i]}} \text{opt}(j).$$

Base case: opt$(1) = 1$.

3.2 ActivitySelection

Given a list of n activities with starting times s_i and finishing times f_i. No two activities can take place simultaneously, find the <u>maximal total duration</u> of a subset of compatible activities

- **Intuition:** Similar to LIS, we want to choose the right activity to extend off of, such that it is not over-lapping and ends with activity i and is of maximal total duration.

Solution

Subproblems: for each $1 \le i \le n$, let $P(i)$ be the problem of determining $t(i)$, the maximal duration of a non-overlapping subsequence of the first i activities which ends with activity i.

Recurrence: for $i > 1$,

$$t(i) = (f_i - s_i) + \max_{\substack{j<i \\ f_j<s_i}} t(j).$$

Base Case: $t(1) = f_1 - s_1$.

3.3 MakingChange

You are given n types of coin denominations of values $v_1 < v_2 < \cdots < v_n$. Make change for a given integer amount C, using as few coins as possible.

- **Intuition:** Similar to the previous think of this as a matching, once we've made i change where could we have come from? We want to come from a value that also minimises that amount of coins used.

Solution

Subproblems: for each $0 \le i \le C$, let $P(i)$ be the problem of determining $\text{opt}(i)$, the fewest coins needed to make change for an amount i.

Recurrence: for $i > 0$,

$$\text{opt}(i) = 1 + \min_{\substack{1\le k\le n \\ v_k\le i}} \text{opt}(i - v_k).$$

Base case: $\text{opt}(0) = 0$.

3.4 GridPaths

We are given an $n \times m$ grid, and start at $(1,1)$ and want to get to (m,n). You can only walk down or right. How many ways are there to get to (m,n)?

- **Intuition:** Think about I'm midway to the house, where could have I come from? I could've either come from the top or the left.

(a) For each $1 \le i,j \le n$, let $\text{GridPath}(i,j)$ denote the number of ways to start from $(1,1)$ and finish at cell (i,j) where we can only move right and down. The recurrence is given by

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

The base case is $\text{GridPath}(1,1) = 1$ with the final solution being $\text{GridPath}(m,n)$.

3.5 MaximumDotProduct

You are given two arrays $A[1..n]$ and $B[1..m]$. Compute the maximum dot product between two non-empty subsequences of A and B.

- **Intuition:** Think about the choices we make in the middle of the problem, we either choose to multiply $A[i]$ and $B[j]$ or we skip it and we could've came from considering the maximum dot product of i and $j-1$ or $i-1$ and j.

Solution. For $1 \le i,j \le n$, let $\text{MaxDotProduct}(i,j)$ denote the maximum dot product of non-empty subsequences of $A[1..i]$ and $B[1..j]$, of the same length. We have the following recurrence:

$$\text{MaxDotProduct}(i,j) = \max \begin{Bmatrix} \text{MaxDotProd}(i,j-1), \\ \text{MaxDotProduct}(i-1,j), \\ \text{MaxDotProduct}(i-1,j-1) + A[i] \cdot B[j] \end{Bmatrix}.$$

The base cases are $\text{MaxDotProduct}(0,j) = 0$ and $\text{MaxDotProduct}(i,0) = 0$ with the final solution occurring at $\text{MaxDotProduct}(n,m)$, and we fill the dynamic programming table in increasing order of i, followed by increasing order of j.

3.6 TwinPalindromes

Youa re given $S[1..n]$ a string of n characters. A *palindromic seubsequence* is a subsequence of S that also forms a palindrome. Find the length of the longest palindromic subsequence

- **Intuition:** A single parameter is not enough since we need some way of knowing two values are the same. Therefore we can do a range representing the index $1 \le i \le j \le n$.

Solution.

(a) For $1 \le i \le j \le n$, let $\text{MaxPalin}(i,j)$ denote the length of the longest palindromic subsequence from index i to index j. The subproblem satisfies the following recurrence:

$$\text{MaxPalin}(i,j) = \begin{cases} 2 + \text{MaxPalin}(i+1,j-1) & \text{if } S[i] = S[j], \\ \max\{\text{MaxPalin}(i+1,j), \text{MaxPalin}(i,j-1)\} & \text{if } S[i] \neq S[j]. \end{cases}$$

The base cases occur when $i = j$, in which case we have that $\text{MaxPalin}(i,i) = 1$. We also consider the edge case when $i > j$, in which case we return 0. The final solution occurs at $\text{MaxPalin}(1,n)$, where we fill the dynamic programming table in increasing order of $j - i$ (i.e. string length).

3.7 IntegerKnapsack

You are given n items, the ith of which has weight w_i and value v_i. You have a knapsack of capacity C, you can take each item any number of times. Choose a combination that will fit in the knapsack and maximise the total value.

- **Intuition**: If we already calculate the maximum value that can be gained at our previous capacity, we can simply choose which one to extend from at the ith item that will ensure our capacity isn't full.

Solution

Subproblems: for each $0 \le i \le C$, let $P(i)$ be the problem of determining

- $\text{opt}(i)$, the maximum value that can be achieved using *up to* i units of weight, and
- $m(i)$, the index of an item in such a collection.

Recurrence: for $i > 0$,

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

$$m(i) = \operatorname{arg\,max}_{1\le k\le n, w_k\le i} [v_k + \text{opt}(i - w_k)].$$

3.8 0-1 Knapsack

Same as integer knapsack however, you can only take an item at most once.

- **Intuition:** We can try our old recurrence however we don't have enough information to tell if the item have been used or not. This tell us to include a second parameter, which tell us the up to k items we've considered.

Subproblems: for $0 \leq i \leq C$ and $0 \leq k \leq n$, let $P(i,k)$ be the problem of determining

- $\mathrm{opt}(i,k)$, the maximum value that can be achieved using up to i units of weight *and* using only the first k items, and

Recurrence: for $i > 0$ and $1 \leq k \leq n$,

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

with $m(i,k) = k$ or $m(i,k) = m(i,k-1)$ respectively.

3.9 ExamRoomAssignment

You are given an array Students[1..n] of non-negative integers, where Students[i] denotes the maximum number of students that can fit in room i. Find the maximum number of students that can be assigned without using three consecutive rooms in $O(n)$

- **Intuition:** If we assume that $i+1$ will always be unoccupied, we can develop our recurrence such that for any room i we have three options. We can choose to ignore i, we can choose to take i, we can choose to take i and $i-2$.

- **Subproblem Definition:** Room(i) $1 \leq i \leq n$, maximum number of students that can be assigned to room 1 to i, assuming that room $i+1$ is not occupied.

- **Recurrence**: Room(i) =

$$\max\begin{cases}\mathrm{Room}(i-1)\\ \mathrm{Room}(i-2) + \mathrm{Students}[i]\\ \mathrm{Room}(i-3) + \mathrm{Students}[i] + \mathrm{Students}[i-1]\end{cases}$$

- **Base Cases:**
  - $\mathrm{Room}(1) = \mathrm{Students}[1]$
  - $\mathrm{Room}(2) = \mathrm{Students}[1] + \mathrm{Students}[2]$
  - $\mathrm{Room}(0) = 0$

- **Final Solution:** Compute in order of increasing i and answer Room(n).

- **Time Complexity**: There are $O(n)$ subproblems, each subproblem takes $O(1)$ time, hence overall time complexity is $O(n)$

4 Reductions

4.1 hamiltonianpath ⇔ 2024-leaf

Hamiltonian Path: Does a hamiltonian path exist on the graph $G = (V, E)$?

2024-Leaf: Does G contain a spanning tree with at most 2024 leaves?

- **Intuition:** A hamiltonian path of G is a spanning tree of at most two leaves. Force one of the leaf vertices to have 2023 more leaf vertices.

- **Reduction Function** Let $G_1 = (V_1, E_1)$ be the input graph. Construct a new graph $G_2 = (V_2, E_2)$ by constructing a new vertex z that has an edge to every vertex in G_1. From z, construct 2023 vertices $v_1, \ldots, v_{2023}$ and edges (z, v_i), for each $i \in \{1, \ldots, 2023\}$.

- ($\Rightarrow$) Suppose G_1 has a Hamiltonian path. Let the Hamiltonian path be the path $P = [x_{i_1}, \ldots x_{i_n}]$. Since z is adjacent to each vertex x_{i_k}, a spanning tree is formed by adding the edge (x_{i_k}, z) and all of the edges of the form (z, v_j) for each $1 \leq j \leq 2023$ onto the path P. This is a spanning tree of G_2 with exactly 2024 leaf vertices.

- ($\Leftarrow$) Suppose that T forms a spanning tree of G_2 with at most 2024 leaf vertices. Each vertex v_i is a leaf vertex of T and so, T must contain the 2023 edges (v_i, z) and a simple path from z to some vertex x_i in G_1. Let t be the only neighbour of z in T that is not a leaf vertex. Let P be the unique path from x_i to t. This path must visit every vertex in G_1 and therefore, P forms a Hamiltonian path of G_1. This path must visit every vertex in G_1 and therefore, P forms a Hamiltonian path of G_1.

4.2 Partition⇔ SubsetSum

Partition: Decide if there exist a way to partition A into two sets E_1 and E_2 such that

$$\sum_{i \in E_1} A[i] = \sum_{i \in E_2} A[i].$$

SubsetSum: Decide if there exist a subset of integers in A that sum to T.

- **Intuition:** If our subset sum is half of A's total value then $A_{\text{partition}}$ must have a subset E_1 of value exactly half of the array's value. Therefore $A \setminus E_1$ must also contain exactly half of the array's value. Hence, a valid partition $E_1 = E_2$.

- **Reduction:** Let T in subset sum equal to exactly half of A's value.

- ($\Rightarrow$) Suppose A has a partition where $E_1 = E_2$. This must mean that $E_1 + E_2$ make up the entire value of A. Hence E_1 is made up of exactly the sum of exactly half of A's value. Hence, our subset sum will detect a valid subset sum of $\frac{A}{2}$.

- ($\Leftarrow$) Suppose A has a valid subset sum of $\frac{A}{2}$. If we remove the following subset from our set A, we'll be left with a subset that is also of exactly $\frac{A}{2}$ such that a valid partition exists.