Dynamic Programming

CPE 349

Theresa Migler-VonDollen

Dynamic Programming

Definition

Dynamic programming is a very powerful algorithmic tool in which a problem is solved by identifying a collection of subproblems and solving them one at a time, smallest first, using the answers for the small problems to help figure out larger ones.

- Sometimes called a "sledgehammer" of algorithm craft.
- Once a solution to a subproblem has been found it is stored, or "memoized".

Dynamic Programming vs Divide and Conquer

- Similarities:
  - Both techniques solve a problem by combining solutions to subproblems.
- Dissimilarities:
  - In Dynamic Programming the subproblems are not independent.
  - Dynamic Programming stores the solutions to subproblems in a table.

Coin Row

Coin Row

Input: A list of n coins, $c_1, c_2, \ldots c_n$, whose values are some positive integers (not necessarily distinct), $v_1, v_2, v_3, \ldots v_n$.
Goal: Find a subset of coins with maximum value subject to the constraint that no two consecutive coins in the input list can be selected.

Example:

- Consider the coin row problem with 8 coins and values $3, 7, 8, 2, 3, 12, 11, 1$.
  - The max value is 25 using c_1, c_3, c_5, and c_7.
- Consider the coin row problem with 7 coins and values $5, 17, 10, 8, 40, 12, 30$.
  - The max value is 87 using c_2, c_5, and c_7.

Dynamic Programming Tables

We are going to create a dynamic programming table. We need to describe:

- The value that each cell contains (a precise definition - in English).
- How to fill in the first entries of the table (base cases).
- Which entry in the table is the solution.
- How to obtain this value from the values in previous cells (a formula).

Coin Row - DP

Coin Row

Input: A list of n coins, $c_1, c_2, \ldots c_n$, whose values are some positive integers (not necessarily distinct), $v_1, v_2, v_3, \ldots v_n$.
Goal: Find a subset of coins with maximum value subject to the constraint that no two consecutive coins in the input list can be selected.

- **Precise definition:**

  Let $CR[i]$ be the value of the maximum value subset of coins (with above constraint) drawing from the first i coins.

- **Base Cases:**

  $CR[0] = 0. CR[1] = v_1.$

- **Solution:** $CR[n]$

- **Formula:**

  $CR[i] = \max\{CR[i-1], v_i + CR[i-2]\}$

Coin Row - Running Time

What is the size of the table?

- $1 \times n$.

How long does it take to fill in each cell?

- Constant.

Therefore, our dynamic program has a running time of $O(n)$.

Coin Row - Example Table

Suppose your input is $18, 29, 17, 5, 12, 19, 6$.

i	1	2	3	4	5	6	7
v_i	18	29	17	5	12	19	6
$CR[i]$	18	29	35	35	47	54	54

Longest Increasing Subsequence

Definition

Given a sequence of numbers, $a_1, a_2, a_3, \ldots a_n$, a *subsequence* is any subset of these numbers taken in order, of the form $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_k}$ where $1 \leq i_1 < i_2 < \cdots i_k \leq n$.

Example:

- Sequence: $\{4, 8, 5, 6, 12, 12, 20, 18, 19, 21\}$
- Subsequences: $\{8, 12, 20, 18, 19\}$, $\{4, 12, 12, 20, 18, 21\}$, many others...

Longest Increasing Subsequence

Definition

Given a sequence of numbers, $a_1, a_2, a_3, \ldots a_n$, an *increasing subsequence* is a subsequence $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_k}$ where the numbers are getting strictly larger.

Example:

- Increasing Subsequences: $\{4, 8, 12\}$, $\{4, 8, 12, 18, 19, 21\}$, $\{12, 19\}$, many others...

Longest Increasing Subsequence

Longest Increasing Subsequence

Input: A sequence of numbers, $a_1, a_2, a_3, \ldots a_n$.
Goal: Find an increasing subsequence of greatest length.

Example:

- Sequence: $\{4, 8, 5, 6, 12, 12, 20, 18, 19, 21\}$
- Subsequences: $\{8, 12, 20, 18, 19\}$, $\{4, 12, 12, 20, 18, 21\}$, many others...
- Increasing Subsequences: $\{4, 8, 12\}$, $\{4, 8, 12, 18, 19, 21\}$, $\{12, 19\}$, many others...
- Longest Increasing Subsequence: $\{4, 5, 6, 12, 18, 19, 21\}$.

Longest Increasing Subsequence - DP Solution

Big Question

How can we use solutions for the longest increasing subsequence problem on smaller versions of the input sequence to come up with a solution for the original input sequence?

- Suppose I have a solution for the longest increasing subsequence problem on $\{4, 8, 5, 6, 12, 12, 20, 18, 19\}$.
- How could I use that information to find a solution for the longest increasing subsequence problem on $\{4, 8, 5, 6, 12, 12, 20, 18, 19, 21\}$?
  Or $\{4, 8, 5, 6, 12, 12, 20, 18, 19, 11\}$?

Longest Increasing Subsequence - DP Solution

Suppose we are given the sequence $a_1, a_2, a_3, \ldots a_n$.

- **Precise definition:** Let $LIS[i]$ be the length of the longest increasing subsequence of the sequence $a_1, a_2, \ldots a_i$ **including** a_i.
- **Base Cases:** $LIS[1] = 1$.
- **Solution:** The maximum value in the table.
- **Formula:** $LIS[i] = \max_{a_j < a_i}\{LIS[j]\} + 1$
  - If no such a_j exists $LIS[i] = 1$.

Longest Increasing Subsequence - Running Time

What is the size of the table?

- $1 \times n$

How long does it take to fill in each cell?

- n

Therefore, our dynamic program has a running time of $O(n^2)$.

Longest Increasing Subsequence - Example Table

Suppose your input is $7, 12, 14, 2, 4, 6, 17, 5$.

i	1	2	3	4	5	6	7	8
a_i	7	12	14	2	4	6	17	5
$LIS[i]$	1	2	3	1	2	3	4	3
pointers	$\emptyset$	1	2	$\emptyset$	4	5	6 OR 3	5

Levenshtein (Edit) Distance

When a spell checker encounters a possible misspelling, it looks in its dictionary for other words that are close by.

What is the appropriate notion of closeness in this case?

A natural measure of the distance between two strings is the extent to which they can be aligned, or matched up.

Technically, an alignment is simply a way of writing the strings one above the other.

For instance, here are two possible alignments of SNOWY and SUNNY:

S	-	N	O	W	Y
S	U	N	N	-	Y

-	S	N	O	W	-	Y
S	U	N	-	-	N	Y

Levenshtein (Edit) Distance

S	-	N	O	W	Y	
S	U	N	N	-	Y	
-	S	N	O	W	-	Y
S	U	N	-	-	N	Y

The "-" indicates a gap; any number of these can be placed in either string.

The cost of an alignment is the number of columns in which the letters differ (this includes when a letter is matched with a gap). Thus the cost of the first alignment above is 3 and the second is 5.

Levenshtein (Edit) Distance

The Levenshtein (edit) distance between two strings is the cost of their best (minimum cost) alignment.

Levenshtein Distance

Input: Two strings, a and b.
Goal: Find the Levenshtein distance between a and b.

Levenshtein (Edit) Distance

- **Precise definition:**
  Let $L[i,j]$ be the Levenshtein distance between $a[1,2,\ldots,i]$ and $b[1,2,\ldots,j]$
- **Base Cases:**
  $L[i,0] = i$.
  $L[0,j] = j$.
- **Solution:** $L[|a|,|b|]$
- **Formula:**

$$L[i,j] = min \begin{cases} L[i,j-1]+1 \\ L[i-1,j]+1 \\ L[i-1,j-1]+\chi_{i,j} \end{cases}$$

Here $\chi_{i,j} = 0$ if $a[i] = b[j]$ and $\chi_{i,j} = 1$ otherwise.

Levenshtein (Edit) Distance - Running Time

What is the size of the table?

- $|a| \times |b|$.

How long does it take to fill in each cell?

- Constant

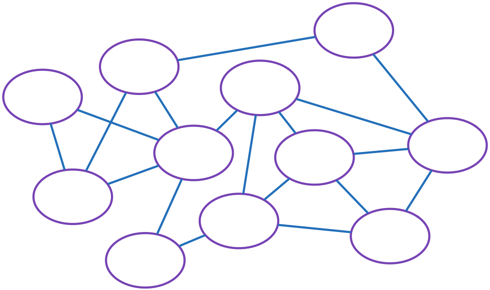
Therefore, our dynamic program has a running time of $O(|a||b|)$.

Independent Set

Definition

A subset $S \subset V$ of vertices forms an *independent set* of a graph $G = (V, E)$ if there are no edges between any two vertices in S.

Example:

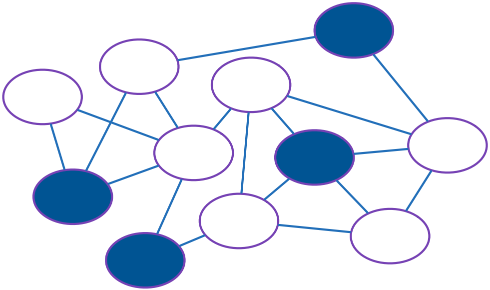


Independent Set

Definition

A subset $S \subset V$ of vertices forms an *independent set* of a graph $G = (V, E)$ if there are no edges between any two vertices in S.

Example:

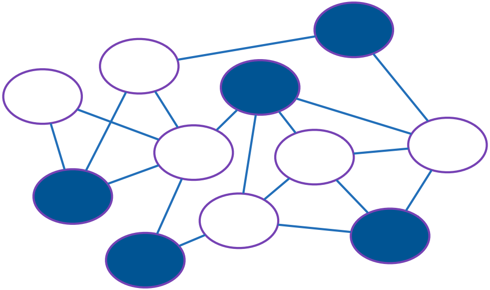


Independent Set

Independent Set

Input: A graph $G = (V, E)$.
Goal: Find a largest independent set (an independent set with the most vertices).

Example:



Independent Set

Independent Set

Input: A graph $G = (V, E)$.
Goal: Find a largest independent set (an independent set with the most vertices).

Problem. Independent Set is an NP-hard optimization problem.

- We focus our attention on independent set in trees.
- We can solve this problem using dynamic programming.

Independent Set on Trees - DP Solution

Independent Set on Trees

Input: A tree $T = (V, E)$.
Goal: Find a largest independent set (an independent set with the most vertices).

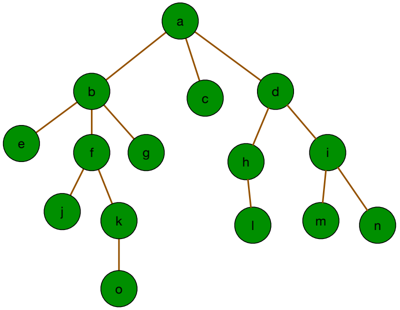
- What are the subproblems?
- Subtrees.

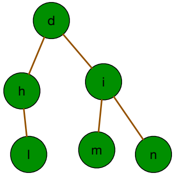
Subtrees

Definition

A *subtree* of a tree $T = (V, E)$, rooted at a vertex $a \in V$ is a tree consisting of a and all of a's descendants in T.

Example:





Independent Set on Trees - DP Solution

Suppose we are given a tree, $T = (V, E)$. Our table will be built with respect to the vertices.

- **Precise definition:** Let $IST[u]$ be the size of the largest independent set for the subtree rooted at u.
- **Base Cases:** $IST[leaves] = 1$.
- **Solution:** $IST[root]$
- **Formula:**
  $IST[u] = \max\{1 + \sum_{\text{w a grandchild of u}} IST[w], \sum_{\text{w a child of u}} IST[w]\}$

Independent Set on Trees - Running Time

What is the size of the table?

- $1 \times n$, where n is the number of vertices.

How long does it take to fill in each cell?

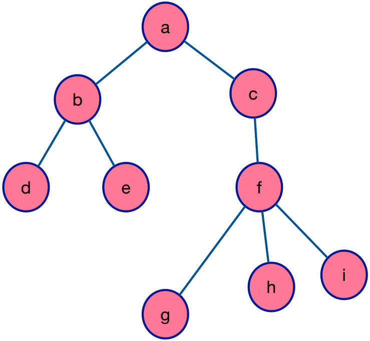
- n

Therefore, our dynamic program has a running time of $O(n^2)$.

- This is a fine running time. There is however a clever argument that shows that the running time is $O(n)$. This can be shown by noting that each vertex, v, is only considered 3 times:
  - Once when processing vertex v.
  - Once when processing v's parent.
  - And once when processing v's grandparent.

Independent Set - Example Table

Suppose your input is the following graph:



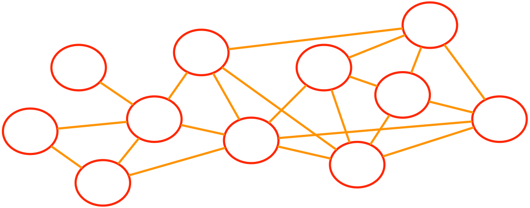
v	d	e	b	g	h	i	f	c	a
$IST[v]$	1	1	2	1	1	1	3	4	6

Vertex Cover

Definition

A *vertex cover* is a subset of vertices of a graph such that every edge is incident to at least one vertex in the set.

Example:

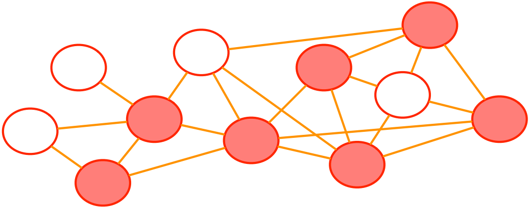


Vertex Cover

Definition

A *vertex cover* is a subset of vertices of a graph such that every edge is incident to at least one vertex in the set.

Example:

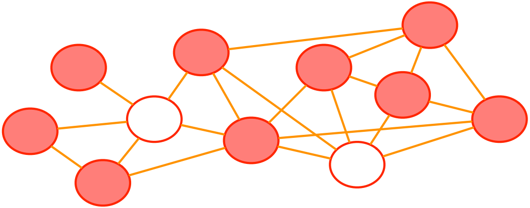


Vertex Cover

Definition

A *vertex cover* is a subset of vertices of a graph such that every edge is incident to at least one vertex in the set.

Example:



Vertex Cover

Vertex Cover

Input: A graph $G = (V, E)$.
Goal: Find a vertex cover of minimum size.

Problem. Vertex Cover is one of Karp's original NP-hard problems.

- Again, we focus our attention on vertex in trees.
- We can solve this problem using dynamic programming.

Vertex Cover on Trees - DP Solution

Vertex Cover on Trees

Input: A tree $T = (V, E)$.
Goal: Find a vertex cover of minimum size.

- Again, the subproblems are subtrees.

Vertex Cover on Trees - DP Solution

Suppose we are given a tree, $T = (V, E)$. Our table will be built with respect to the vertices.

- **Precise definition:**
  Let $VC[v, 0]$ be the size of the smallest vertex cover for the subtree rooted at v, not including v.
  Let $VC[v, 1]$ be the size of the smallest vertex cover for the subtree rooted at v, including v.
- **Base Cases:**
  $VC[leaves, 1] = 1. VC[leaves, 0] = 0.$
- **Solution:** $min\{VC[root, 0], VC[root, 1]\}$
- **Formula:**
  $VC[u, 0] = \sum_{\text{w a child of u}} VC[w, 1]$
  $VC[u, 1] = 1 + \sum_{\text{w a child of u}} \min\{VC[w, 0], VC[w, 1]\}$

Vertex Cover on Trees - Running Time

What is the size of the table?

- $2 \times n$, where n is the number of vertices.

How long does it take to fill in each cell?

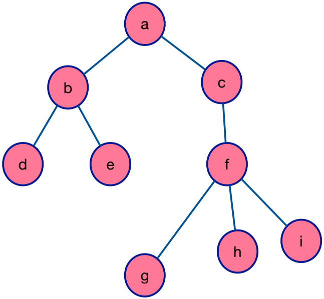
- n

Therefore, our dynamic program has a running time of $O(n^2)$.

- Again, this is a fine running time. There is however a clever argument that shows that the running time is $O(n)$. This can be shown by noting that each vertex, v, is only considered 4 times:
  - Once when processing vertex v in $VC[v, 0]$.
  - Once when processing vertex v in $VC[v, 1]$.
  - Once when processing v's parent, p, in $VC[p, 0]$.
  - Once when processing v's parent, p, in $VC[p, 1]$.

Vertex Cover - Example Table

Suppose your input is the following graph:



v	d	e	b	g	h	i	f	c	a
$VC[v, 0]$	0	0	2	0	0	0	3	1	3
$VC[v, 1]$	1	1	1	1	1	1	1	2	3

Knapsack

Knapsack

Input: A set, N, of n items, each with weight $w_1, w_2, \ldots, w_n$ and value $v_1, v_2, v_3, \ldots, v_n$, and a threshold W.
Goal: Find a subset, $S \subset N$ of the items such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

There are two versions of this problem:

- Unlimited quantities of each item (with repetition)
- Each item is unique (without repetition)

Knapsack

Example: Suppose $W = 14$

item	weight	value
1	7	4
2	3	10
3	4	5
4	2	2

- With repetition, the optimal knapsack contains four of item 2 and one of item 4. This knapsack has value 42.
- Without repetition, the optimal knapsack contains one of item 1, one of item 2, and one of item 3. This knapsack has value 19.

Knapsack with Repetition - DP

Knapsack

Input: A set, N, of n items, each with weight $w_1, w_2, \ldots, w_n$ and value $v_1, v_2, v_3, \ldots, v_n$, and a threshold W.
Goal: Find a subset (with possible repetitions), $S \subset N$ of the items such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

- **Precise definition:**
  Let $K[w]$ be the maximum value achievable with a knapsack of capacity w.
- **Base Case:**
  $K[0] = 0$.
- **Solution:** $K[W]$
- **Formula:**
  $K[w] = \max_{w \geq w_i} \{K[w - w_i] + v_i\}$

Knapsack with Repetition - Running Time

What is the size of the table?

- $1 \times W$.

How long does it take to fill in each cell?

- n

Therefore, our dynamic program has a running time of $O(W \times n)$.

- Is this polynomial?

Knapsack with Repetition - Example Table

Suppose your input is: $W = 8$

item	weight	value
1	2	1
2	2	3
3	3	4
4	5	3

w	1	2	3	4	5	6	7	8
$K[w]$	0	3	4	6	7	9	10	12

Knapsack without Repetition - DP

Knapsack

Input: A set, N, of n items, each with weight $w_1, w_2, \ldots, w_n$ and value $v_1, v_2, v_3, \ldots, v_n$, and a threshold W.
Goal: Find a subset, $S \subset N$ of distinct items such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

- **Precise definition:**
  Let $K[w, i]$ be the maximum value achievable with a knapsack of capacity w drawing from the first i items.
- **Base Cases:**
  $K[w, 0] = 0.$
  $K[0, i] = 0.$
- **Solution:** $K[W, n]$
- **Formula:**
  $K[w, i] = \max_{w \geq w_i} \{K[w - w_i, i - 1] + v_i, K[w, i - 1]\}$

Knapsack without Repetition - Running Time

What is the size of the table?

- $n \times W$.

How long does it take to fill in each cell?

- Constant.

Therefore, our dynamic program has a running time of $O(W \times n)$.

- Is this polynomial?

Knapsack without Repetition - Example Table

Suppose your input is: $W = 8$

item	weight	value
1	2	1
2	2	3
3	3	4
4	5	3

(i, w)	1	2	3	4	5	6	7	8
1	0	1	1	1	1	1	1	1
2	0	3	3	4	4	4	4	4
3	0	3	4	4	7	7	8	8
4	0	3	4	4	7	7	8	8

Change Making

Suppose we are given an unlimited supply of coins of denominations $x_1, x_2, x_3, \ldots x_n$, we wish to make change for a value V. That is, we wish to find a set of coins whose total value is V.

Note: This might not be possible...

- Suppose the denominations are $5, 12, 19, 50$.
  - Is it possible to make change for 36?
  - Is it possible to make change for 7?

Change Making

Input: $x_1, x_2, x_3, \ldots x_n$, V.

Goal: Determine if it is possible to make change for V using denominations $x_1, x_2, x_3, \ldots x_n$.

Change Making - DP

Change Making

Input: $x_1, x_2, x_3, \dots x_n$, V.
Goal: Determine if it is possible to make change for V using denominations $x_1, x_2, x_3, \dots x_n$.

- **Precise definition:**
  Let $CM[v, i] = 1$ if it is possible to make change for v using the first i denominations. $CM[v, i] = 0$ if it is not possible.
- **Base Cases:**
  $CM[v, 0] = 0$.
  $CM[0, i] = 1$.
- **Solution:** $CM[V, n]$
- **Formula:**
  $CM[v, i] = \max\{CM[v - x_i, i], CM[v, i-1]\}$

Change Making - Running Time

What is the size of the table?

- $V \times n$.

How long does it take to fill in each cell?

- Constant.

Therefore, our dynamic program has a running time of $O(V \times n)$.

Change Making - Variations

Change Making

Input: $x_1, x_2, x_3, \ldots x_n$, V.
Goal: Determine if it is possible to make change for V using denominations $x_1, x_2, x_3, \ldots x_n$.

- Suppose that you can use each denomination at most once. (homework)
  - How would this change the dynamic program?
- Suppose that you must make change using at most k coins.
  - How would this change the dynamic program?

Change Making - Example Table

Is it possible to make change for 9 with coin denominations 2, 5, 6, and 8?

(i, v)	1	2	3	4	5	6	7	8	9
2	0	1	0	1	0	1	0	1	0
5	0	1	0	1	1	1	1	1	1
6	0	1	0	1	1	1	1	1	1
8	0	1	0	1	1	1	1	1	1

Cookie Collecting

Cookie Collecting

Input: There are several cookies placed in cells of a $n \times m$ board, no more than one per cell.
A robot, which is located in the upper left corner of the board wants to collect as many cookies as possible and bring them to the bottom right cell. The robot can only move one cell to the right or one cell down on each move.
Goal: Find the maximum amount of cookies that the robot can collect.

Cookie Collecting - DP

Cookie Collecting

Input: There are several cookies placed in cells of a $n \times m$ board, no more than one per cell.
Goal: Find the maximum amount of cookies that the robot can collect.

- **Precise definition:**
  Let $CC[i, j]$ be the maximum amount of cookies that the robot can collect on the board restricted to the first i rows and j columns.
- **Base Cases:** $CC[i, 0] = 0. CC[0, j] = 0.$
- **Solution:** $CC[n, m]$
- **Formula:**
  Let $\chi_{i,j} = 1$ if there is a cookie in cell (i, j) and $\chi_{i,j} = 0$ if there is no cookie in cell (i, j).
  $CC[i, j] = \max\{CC[i-1, j], CC[i, j-1]\} + \chi_{i,j}$

Cookie Collecting - Running Time

What is the size of the table?

- $n \times m$.

How long does it take to fill in each cell?

- Constant.

Therefore, our dynamic program has a running time of $O(n \times m)$.

Cookie Collecting - Example Table

Consider the following game board:

$\star$		$\star$
	$\star$	
		$\star$
$\star$		$\star$

	0	1	2	3
0	0	0	0	0
1	0	1	1	2
2	0	1	2	2
3	0	1	2	3
4	0	2	2	4