

CS2104 Problem Solving in Computer Science

*Margaret Ellis,
Naren Ramakrishnan,
Sehrish Basir Nizamani*

Algorithms

Virginia Tech CS Alumni Quote

"Most tech companies have their own internal tooling so you can't just use stackoverflow or ChatGPT to figure things out, you need to be able to master the fundamentals to be productive and successful. The process of constantly learning new languages and concepts during my education prepared me for new frontiers." - VT CS BS 2023, Meng 2024, Amazon AWS Cloud Engineer, Seattle, WA

Today's Objectives

1. Understand and use python string matching
2. Analyze the use of LLMS to code and test
3. Understand and use python regular expressions

Recall Python Strings

- Strings are sequences of characters
- Strings are immutable
 - Once created, strings cannot be modified - any operation that appears to modify a string actually creates a new one
- Here's a simple program to print each character in a string

```
def print_characters(string):
```

```
    for char in string:
```

```
        print(char)
```

```
example_string = "Hello, World!"
```

```
print_characters(example_string)
```

Concepts for HW4 ...



Concepts for HW4 ...

Indexing Python Strings

- Characters in a string can be indexed from 0 till the length of the string - 1
- *For instance, given a string in variable "text"*
 - `print(text[0])` # First character
 - `print(text[1])` # Second character
 - `print(text[-1])` # Last character
- You can index a string from 0 till `len(string)-1`
 - Or from `-len(string)` to -1

Find repeated letter in a given string

GPT-4o generated code →

Don't understand the code?

Demo:

- 1) *setting breakpoint with debugger to step through in vs code*
- 2) *python tutor*
- 3) *asking LLM to explain*
- 4) *trace by hand!*

```
def find_repeated_letters(input_string):  
    repeated_letters = []  
    for i in range(len(input_string)):  
        if input_string[i] in input_string[:i] and input_string[i] not in repeated_letters:  
            repeated_letters.append(input_string[i])  
    return repeated_letters
```

Example usage

```
input_string = "programming"  
print(find_repeated_letters(input_string))
```

Output will be

```
#[ 'r', 'm', 'g' ]
```


Palindrome

- Sequence of characters that reads same as backwards
 - E.g., “level”, “radar”
- Given a string, check if that string is palindrome
- GPT-4o generated code→

Don't understand the code? *Demo:*
1) *setting breakpoint with debugger to step through in vs code*
2) *python tutor*
3) *asking LLM to explain*
4) *trace by hand!*

```
def is_palindrome(string):  
    # Remove non-alphanumeric characters and convert to lowercase  
    clean_string = ''.join(char.lower() for char in string if  
    char.isalnum())  
    # Check if the string is equal to its reverse  
    return clean_string == clean_string[::-1]  
  
# Example usage  
  
string = input("Enter a string: ")  
  
if is_palindrome(string):  
    print("The string is a palindrome!")  
else:  
    print("The string is not a palindrome.")
```

Week 4 HW - Substring Wannabe Palindrome Problem

In this assignment, you will solve the following problem:

Given a string s , find the longest "wannabe" palindromic substring. A wannabe palindrome allows for one mismatch within the substring (e.g., "abcbda" is a wannabe palindrome since "abcba" is the core palindrome). Return the length of the longest wannabe palindromic substring.

Constraints:

1. s consists only of lowercase English letters.
2. A substring with more than one mismatch does not qualify
3. A substring with no mismatches qualifies

Examples:

1. $s = \text{"abcba"}$, Expected output: 5 (abcba)
2. $s = \text{"abcbda"}$, Expected output: 6 (abcbda)

Regular expression

Regular Expressions

- Regex is a sequence of characters that forms a search pattern
- Used in many programming languages, text editors
- Consist of a combination of normal characters (letters and digits) and special characters (*, +, ?) that have special meanings
- Finding specific words or patterns in a document, validating user input in a form, and transforming text into a different format

Regular Expressions Symbols Review

Special Characters

Symbol	Description	Example	Matches
.	Matches any single character except newline.	a.b	aab, acb
^	Matches the start of a string.	^abc	abc at the start of the string
\$	Matches the end of a string.	abc\$	abc at the end of the string
*	Matches 0 or more repetitions of the preceding token.	ab*c	ac, abc, abbc
+	Matches 1 or more repetitions of the preceding token.	ab+c	abc, abbc
?	Matches 0 or 1 repetition of the preceding token.	ab?c	ac, abc
{n}	Matches exactly n occurrences of the preceding token.	a{3}	aaa
{n,}	Matches n or more occurrences of the preceding token.	a{2,}	aa, aaa, aaaa
{n,m}	Matches between n and m occurrences of the preceding token.	a{2,4}	aa, aaa, aaaa

Should be
familiar from
python
tutorial

Regular Expressions Symbols Sets

Syntax	Matches	Explained
[rEsZ]	Regular Expression	Square brackets [] define a set, where each character is matched independently. A match occurs if any character from the set appears in the text.
[a-z]	1 Fig, 2 NewTons	The - in [m-n] is a range operator, matching any character from m to n.
[a\z]	a to z is not = A-Z	The \ escapes the -, treating it as a literal character instead of a range operator. This set matches a, z, and - only.
[a-]	regular-expression	Matches a and the literal - because - is treated as a character when placed at the start or end of a set.
[-a]	regular-expression	As above, matches a or -.
[a-z0-9]	396 ExpressionS	Matches characters from a to z and also from 0 to 9.
[(+*)]	(valid)*expressions+words	Special characters become literal inside a set, so this matches (, +, *, and).
[^ers]	regular expression	The ^ negates the set, matching any character not in the set. Here, it matches characters that are not e, r, or s.

Regular Expressions Symbols

Special Sequences

Character Class	Description	Example/Matches
<code>\w</code>	Matches all alphanumeric characters (a-z, A-Z, and 0-9) and underscore (<code>_</code>)	<code>Ch4racter_Class3s</code>
<code>\W</code>	Matches any non-word character (symbols, punctuation, spaces)	<code>!@#\$%^&*()</code>
<code>\d</code>	Matches all digits 0-9	<code>1a2b3c</code>
<code>\D</code>	Matches any non-digits	<code>1a2b3c</code>
<code>\s</code>	Matches whitespace characters including <code>\t</code> , <code>\n</code> , <code>\r</code> , and spaces	<code>Character classes</code>
<code>\S</code>	Matches non-whitespace characters	<code>Character classes</code>
<code>\b</code>	Matches word boundaries i.e. the start and end of the word	<code>\bfox\b</code> The <code>fox</code> tried to trick the hunter, but he remained unfoxed.
<code>\B</code>	Matches non-word boundaries. match characters in the middle of other non-space characters	<code>\Bfox\B</code> The fox tried to trick the hunter, but he remained unfoxed.

Should be familiar from python tutorial

Python re Module Functions

Syntax	Explained
<code>re.findall(A, B)</code>	Finds all non-overlapping matches of the pattern A in string B and returns them as a list. If no matches are found, it returns an empty list.
<code>re.search(A, B)</code>	Searches string B for the first occurrence of the pattern A and returns a match object. If no match is found, it returns None.
<code>re.split(A, B)</code>	Splits string B into a list at each occurrence of the pattern A. If no match is found, it returns the original string as a single-element list.
<code>re.sub(A, B, C)</code>	Replaces all occurrences of the pattern A in string C with the string B and returns the modified string. The original string C remains unchanged.
<code>re.match(A, B)</code>	Attempts to match the pattern A starting strictly at position 0 in string B. If the pattern doesn't match at the start, it returns None. Unlike <code>re.search()</code> , it does not evaluate the rest of the string.

EXAMPLE: re.search()

```
import re
import unittest

def findmatch(str):
    return re.search('word:\w\w\w',str)

#print(findmatch('an example word:cat!!').group())

class TestFindMatch(unittest.TestCase):
    def test_findmatch(self):
        # Test cases for findmatch
        self.assertEqual('word:cat',findmatch('an example word:cat!!').group(), "not matched")
)

if __name__ == '__main__':
    unittest.main(argv=[""], exit=False)
```

VT CS Alumni Quote

"GenAI tools are changing how we work in the software industry and there's certainly a time and place to use them! I'd encourage students to use those tools when appropriate and when explicitly allowed in their course syllabi as a way to find new ideas, generate boilerplate code, or try to explain a concept. However, it is also obvious when someone presents work that is incorrect and written by a tool such as ChatGPT. Oftentimes these tools will produce code (or explanations of code) that look "correct" but make fundamental errors such as not following the syntax of a given programming language.

Virginia Tech CS prepared me for my current role with a solid understanding of Computer Science fundamentals. Generative AI is not a substitute for this foundational knowledge. To be able to work effectively with colleagues in the industry you need to be able to explain and justify your work." VT CS BS 2023, Red Hat Consultant, Raleigh, NC

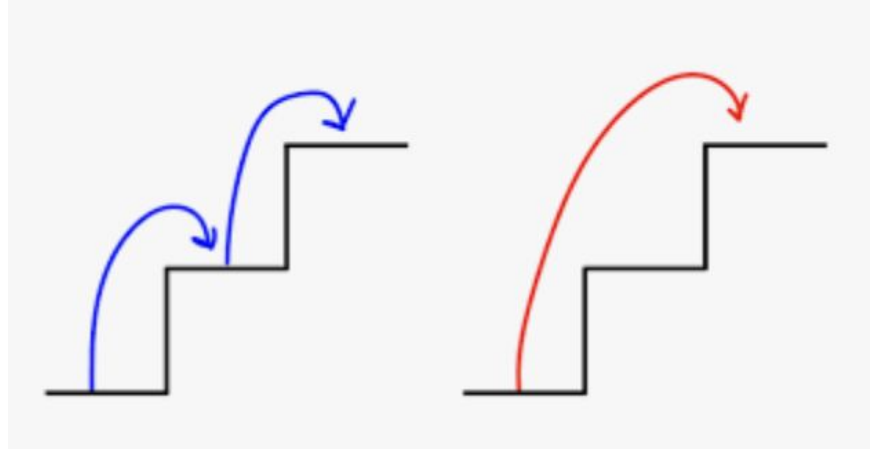
Recursion Algorithms

Objectives

- **Understand and use recursion and dynamic programming in Python**
 - i. Understanding Recursion
 - ii. Recursive vs Iterative
 - iii. Use of Helper Functions
 - iv. Recursions with Backtracking
- **Analyze the use of LLMs to code and test**

Staircase Problem

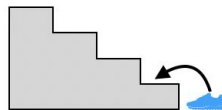
- Given a staircase with n stairs and that you can only take 1 or 2 steps at a time, how many ways you can walk your way up to reach the n th stair.



Staircase Problem

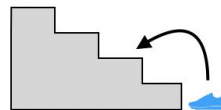
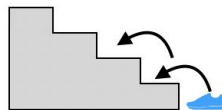
Stair 1

There is only 1 way to climb stair 1



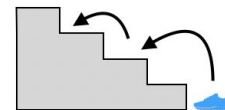
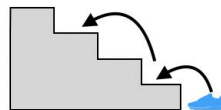
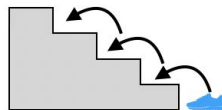
Stair 2

There are 2 ways to climb stair 2 => (1,1) and (2)



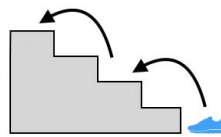
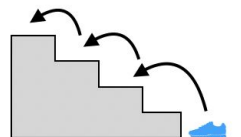
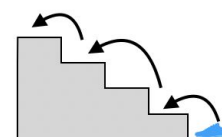
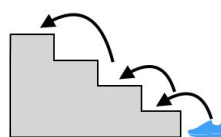
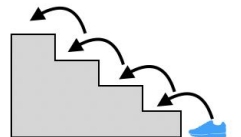
Stair 3

There are 3 s to climb stair 3 => (1,1,1) , (1,2) and (2,1)



Stair 4

There are 5 ways to climb stair 4 => (1,1,1,1) , (1,1,2) , (1,2,1) , (2,1,1) and (2,2)



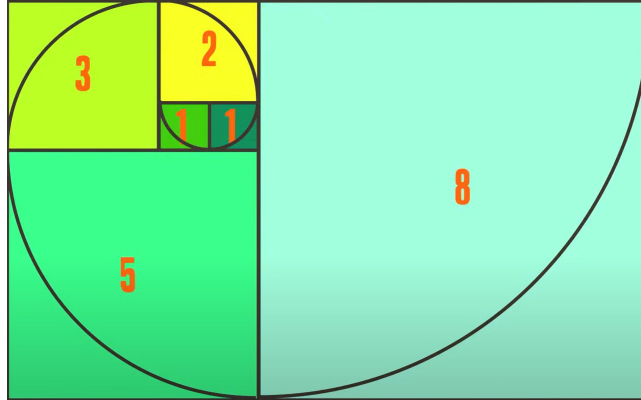
Staircase Problem (Continue)

$$F(n) = F(n-1) + F(n-2)$$

1 step from (n-1)th stair

2 steps from (n-2)th stair

The Fibonacci Numbers



Recursive Algorithm

```
# Computes the value of the n-th Fibonacci number.  
# Pre: n is a non-negative integer.  
def fibonacci(n):  
    if n < 2:                # base case  
        return 1  
    return fibonacci(n-1) + fibonacci(n-2) # recursive case
```

Understanding Recursion

- A process in which the result of each repetition is dependent upon the result of the next repetition.
- Simplifies program structure at a cost of function calls

Recursive Definitions (Fibonacci sequence)

$$\text{fibonacci}(n) = \begin{cases} 1 & n = 0, 1 \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & n > 1 \end{cases}$$

base case
 recursive case

$$\begin{aligned}
 \text{fibonacci}(4) &= \text{fibonacci}(3) + \text{fibonacci}(2) \\
 &= \text{fibonacci}(2) + \text{fibonacci}(1) + \\
 &\quad \text{fibonacci}(1) + \text{fibonacci}(0) \\
 &= \text{fibonacci}(1) + \text{fibonacci}(0) + \\
 &\quad 1 + 1 + 1 \\
 &= 1 + 1 + 1 + 1 + 1 \\
 &= 5
 \end{aligned}$$



Recursion

- ❖ a base case in which no recursion occurs
- ❖ a recursive case



```
int factorial(int);  
void main()  
{  
    int fact, n;  
    printf("Enter any positive integer: ");  
    scanf("%d", &n);  
    fact = factorial(n);  
    printf("Factorial of %d is %d", n, fact);  
}
```

factorial(3);

6

```
int factorial(int n)  
{  
    int temp;  
    if(n == 0)  
        return 1;  
    else  
        temp = n * factorial(n-1);  
    return temp;  
}
```

3*factorial(2);

2

```
int factorial(int n)  
{  
    int temp;  
    if(n == 0)  
        return 1;  
    else  
        temp = n * factorial(n-1);  
    return temp;  
}
```

2*factorial(1);

1

```
int factorial(int n)  
{  
    int temp;  
    if(n == 0)  
        return 1;  
    else  
        temp = n * factorial(n-1);  
    return temp;  
}
```

1*factorial(0);

1

```
int factorial(int n)  
{  
    int temp;  
    if(n == 0)  
        return 1;  
    else  
        temp = n * factorial(n-1);  
    return temp;  
}
```

Recursion animation

```
factorial( n ):
```

```
    if n == 1:  
        return 1
```

```
    else:
```

```
        return n * factorial(n-1):
```

```
            if n == 1:  
                return 1
```

```
            else:
```

factorial(n) =

Recursive Definitions

recursion a method of defining functions in which the function being defined is applied within its own definition

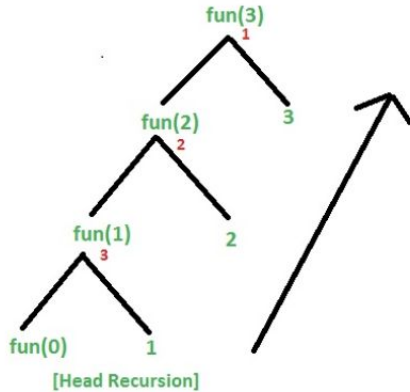
$$factorial(n) = \begin{cases} 1 & n = 0 \\ n \cdot factorial(n-1) & n > 0 \end{cases}$$

base case
←
recursive case

```
factorial(5)  =  5 * factorial(4)
               =  5 * (4 * factorial(3))
               =  5 * (4 * (3 * factorial(2)))
               =  5 * (4 * (3 * (2 * factorial(1))))
               =  5 * (4 * (3 * (2 * 1)))
               =  120
```

Head and Tail Recursion

Tracing Tree Of Recursive Function



Output: 1 2 3

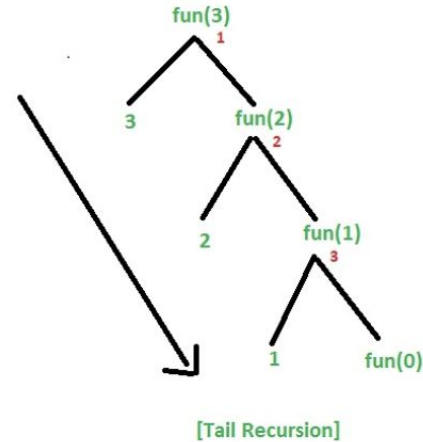
*Digits in red showing that the order in which the calls are made and note that printing done at returning time. And it does nothing at calling time.

#head recursion

#pre: n is an N+

```
def fun(n):
    if n == 1:
        print(1)
    else:
        fun(n - 1)
    print(n)
```

Tracing Tree Of Recursive Function



Output: 3 2 1

*Digits in red showing that the order in which the calls are made and according to the order of calling the output are printed on the screen. Note that for `fun(0)` it gives nothing as output.

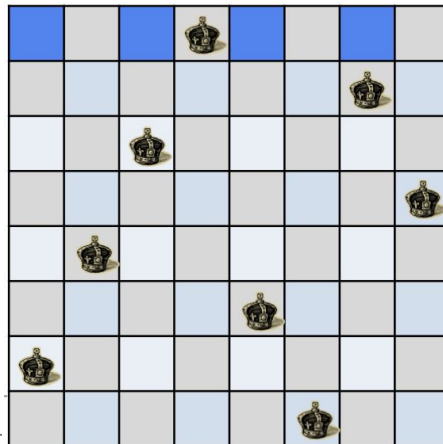
#tail recursion

#pre: n is N+

```
def fun(n):
    if n == 1:
        print(1)
    else:
        print(n)
        fun(n - 1)
```

K Queens Problem

Given a $K \times K$ chessboard, find a way to place K queens on the board so that no queen can attack another queen. A queen can move an arbitrary number of squares vertically, horizontally or diagonally.



Here is one solution: Turns out that there must be one queen in every row and one queen in every column.

There are over 4 billion different ways to drop 8 queens onto an 8x8 board.

It's known that there are exactly 92 distinct solutions to the problem.

4 Queens Problem

Solve a
simpler
version

Let's consider a variant on a 4x4 board... how to start?

Let's flag squares that are under attack with Xs, since we cannot put a queen there.

Let's process the board row by row, from the top down.

Let's start by putting a queen in the first square in row 1:

	X	X	X
X	X		
X		X	
X			X

Now for row 2... we have two choices, let's try the first one:

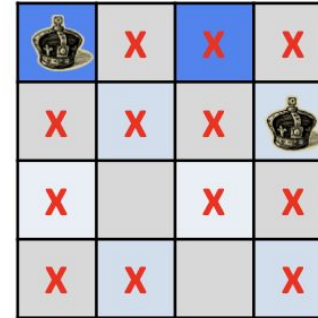
	X	X	X
X	X		X
X	X	X	X
X		X	X

Oops... now all the squares in row 3 are under attack, so this cannot lead to a solution...

4 Queens Problem (2)

What to try next?

Let's *backtrack*... take back the last move and try a different one:



	X	X	X
X	X	X	
X		X	X
X	X		X

OK, now we have possibilities... let's fill the free square in row 3:

Rats! Now there are no free squares left in row 4.

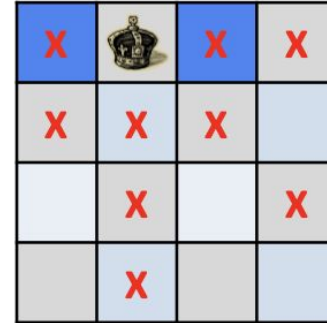
We can backtrack again, but that means we must now remove the 2nd and 3rd queens, since we've already tried all the possibilities for the 2nd one, and then we must consider a different spot for the 1st one...



	X	X	X
X	X	X	
X		X	X
X	X	X	X

4 Queens Problem (3)

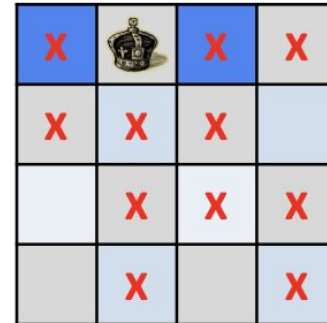
So, we'll try the 1st queen in column 2:



A 4x4 grid representing a chessboard. The first row has a queen in column 2 (indicated by a crown icon). The first, third, and fourth columns of the first row are marked with a red 'X'. The second, third, and fourth columns of the second row are marked with a red 'X'. The second and fourth columns of the third row are marked with a red 'X'. The second column of the fourth row is marked with a red 'X'.

X		X	X
X	X	X	
	X		X
	X		

That leaves just one place for a queen in row 2:



A 4x4 grid representing a chessboard. The first row has a queen in column 2 (indicated by a crown icon). The first, third, and fourth columns of the first row are marked with a red 'X'. The first, second, and third columns of the second row are marked with a red 'X'. The second, third, and fourth columns of the third row are marked with a red 'X'. The second and fourth columns of the fourth row are marked with a red 'X'.

X		X	X
X	X	X	
	X	X	X
	X		X

K Queens Problem Algorithm

Let's suppose we have some way to represent a board configuration (size, location of queens, number of queens, etc.)

K Queens Algorithm

Try_config takes configuration C, number m

if C contains K queens

display C

halt

endif

for each square in row m of C

if square is free

place a queen in square

Try_config(C, m + 1) # leads to soln?

remove queen from square # no, backtrack

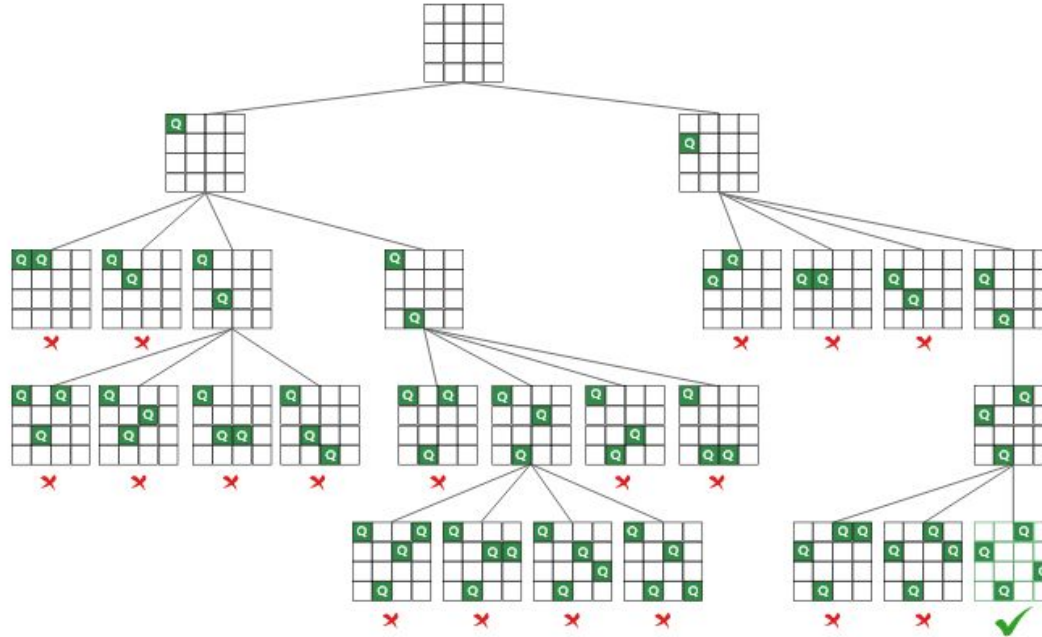
endif

Backtracking

The general steps of backtracking are:

- start with a sub-solution
- check if this sub-solution will lead to the solution or not
- If not, then come back and change the sub-solution and continue again

Geeks for Geeks (Working through columns)



Knapsack

- What is Knapsack?
 - A combinatorial optimization problem where you maximize the total value of items placed in a knapsack without exceeding its weight capacity.
 - Think of a traveler with a fixed-size backpack trying to carry the most valuable combination of items.
- Real-world applications
 - **Resource Allocation:** Budget optimization, investment decisions.
 - **Logistics:** Shipping container capacity optimization.
 - **Computer Science:** Memory allocation, load balancing.
 - **Cryptography:** Basis for some encryption schemes.

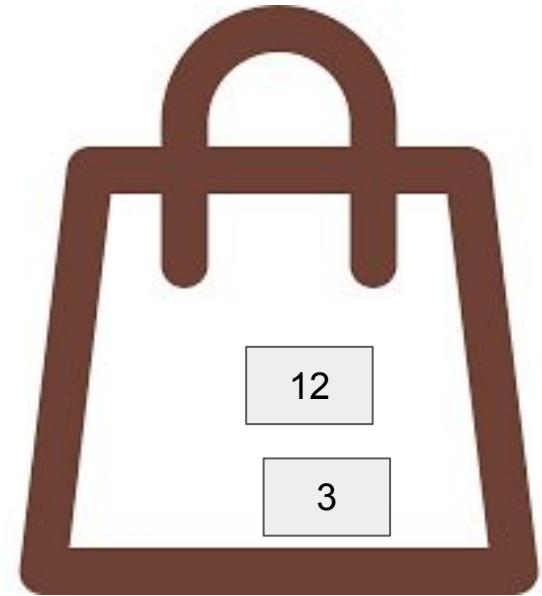


https://en.wikipedia.org/wiki/Knapsack_problem

Knapsack Problem

Given an integer total, and an integer list, determine if any collection of list elements within a subset of the list sum up to total.

Your goal is to fill your knapsack with 15 weight capacity, choose items from the list [17, 3, 4, 12, 2], how to pick the items such that they will mount up to 15?



17

2

4

Knapsack Problem - Algorithm

Algorithm:

Check if a collection exists containing the first subset element,

(i.e. does collection exist for the remaining elements in the list for the total reduced by the first element)?

If no collection exists containing the first subset element check for a collection for total from subset start + 1 to the end of the subset.

Knapsack Problem (Recursive Solution)

```
# ray   List of integers
# sum   Subset sum goal
# start First subset index
# end   Last subset index
# Prerequisite: ray is a list of positive integers,
#               sum is a positive integer
#               start, end are valid list indexes
def knapsack(ray,sum,start,end):
    if ( sum == 0 ): #empty collection sums to zero
        return True
    if ( (sum < 0) or (start > end) ): #no solution
        return False
    #check for soln with first element
    if (knapsack(ray, sum-ray[start], start+1, end) ):
        return True
    #any possible soln cannot contain first element
    return knapsack(ray, sum, start+1, end)
```

[17, 3, 4, 12, 2]

Knapsack Problem Recursive Solution - Code Trace

knapsack code trace of [17, 3, 4, 12, 2] looking for sum of 15 between indices 0 and 4 (diagram on next slide)

call knapsack(ray, 15 - 17, 1, 4)

looking for sum of -2 which returns FALSE

call knapsack(ray, 15, 1, 4) looking for sum of 15 between 1 and 4

call knapsack(ray, 15 - 3, 2, 4)

call knapsack(ray, 12 - 4, 3, 4)

call knapsack (ray, 8 - 12, 4, 4)

looking for sum of -4 which returns **false**

call knapsack(ray, 8, 4, 4)

call knapsack(ray, 8 - 2, 5, 4)

start > end which returns false

call knapsack(ray, 8, 5, 4)

returns false

call knapsack(ray, 12, 3, 4) # backtrack happens

call knapsack(ray, 12 - 12, 4, 4)

returns TRUE!!

The values 3 and 12 sum to 15, it skipped 17 and 4 by backtracking

Knapsack Problem Approaches

- Recursive with backtracking
- Dynamic programming uses book-keeping (remembering subsolutions)
 - Dynamic programming is useful for online coding challenges and interview questions

VT CS Alumni Quotes

"We have daily stand ups and regular code reviews so being able to communicate about your work and the work of others is a vital part of the job. It's fun to help design new features and it requires a lot of time and work to be prepared to contribute to a project. My courses, undergraduate research, and student organizations at VT CS gave me experience to get and be successful in my web development positions." -VT CS BS 2019, Google, Software Engineer, Seattle, WA

"Employers want to know you can problem solve and work in a team. It's important to be able to communicate about your decision making and coding process and also be able to ask your colleagues questions so you can learn and advance in your career." -VT CS BS 2023, Manhattan, Cloud and Security Engineer, Atlanta, GA

Knapsack and Variations

Objectives:

- understand and use recursion and dynamic programming in python
- analyze the use of LLMs to code and test

Types of Knapsack Problems

- **0/1 Knapsack Problem:** Either include or exclude an item (binary choice).
- **Fractional Knapsack Problem:** Items can be broken into fractions (solved using greedy algorithms).
- **Unbounded Knapsack Problem:** Unlimited copies of each item are allowed.

Consider

- Example:

- Items:

Item	Weight w	Value v
1	2	10
2	3	20
3	4	30

- Knapsack Capacity: $W = 5$
 - **Solution (0/1 Knapsack):** Include items 1 and 2 for a total value of 30.

How do we solve the knapsack problem?

- One strategy is to do “dynamic programming”
 - Fancy term for “book-keeping”
- For every item
 - We can either keep it, or discard it
 - Lets create a lookup table $dp[i][w]$ where
 - i is the index of the current item (0 to n).
 - w is the current weight capacity (0 to W).
 - $dp[i][w]$ represents the **maximum value obtainable** using the first i items and weight capacity w .
 - We will keep filling this table from one corner so as to make sure we have considered all possibilities

Knapsack worked out

$dp[i][w]$ represents the **maximum value obtainable** using the first i items and weight capacity w .

- We have 3 items and capacity 5, so we will actually create a table of 4 (3+1) rows and 5 columns

		weight					
$dp[i][w]$		0	1	2	3	4	5
Number of items	$i = 0$	0	0	0	0	0	0
	$i = 1$	0	0	10	10	10	10
	$i = 2$	0	0	10	20	20	30
	$i = 3$	0	0	10	20	30	30

Item	Weight w	Value v
1	2	10
2	3	20
3	4	30

$dp[3][4]$ could just be $dp[2][4]$ which is 20, but get more value by removing what was there and adding weight of 4 with value of 30

More examples of 0/1 knapsack

Given a set of **N** items, each with a weight and a value, determine the maximum total value of items you can include in a knapsack of capacity **W**. Remember that you cannot break items; you either take an item or not (0/1 Knapsack problem).

N = 3 W = 50 Weights = [10, 20, 30] Values = [60, 100, 120]	220	N = 5 W = 15 Weights = [12, 7, 11, 8, 9] Values = [24, 13, 23, 15, 16]	28	N = 4 W = 5 Weights = [5, 5, 5, 5] Values = [10, 40, 30, 50]	50
N = 4 W = 10 Weights = [5, 4, 6, 7] Values = [10, 40, 30, 50]	70	N = 6 W = 100 Weights = [77, 22, 29, 50, 99, 1] Values = [92, 22, 87, 46, 90, 1]	134		

Lets use GPT-4o to generate code for 0/1 Knapsack

```
def knapsack(N, W, weights, values):  
    # Initialize a 2D DP array  
    dp = [[0] * (W + 1) for _ in range(N + 1)]  
  
    # Fill the DP table  
    for i in range(1, N + 1):  
        for w in range(W + 1):  
            if weights[i - 1] <= w:  
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])  
            else:  
                dp[i][w] = dp[i - 1][w]  
  
    return dp[N][W]
```

Given the knapsack algorithm in python ...] can you explain $dp[i - 1][w - weights[i - 1]] + values[i - 1]$? (co-pilot)

Sure! Let's break down the expression $dp[i - 1][w - weights[i - 1]] + values[i - 1]$ in the context of the knapsack algorithm.

Context

In the knapsack problem, we aim to maximize the total value of items that can be placed in a knapsack of capacity (W), given (N) items, each with a specific weight and value.

Dynamic Programming Table (dp)

$dp[i][w]$ represents the maximum value that can be achieved with the first i items and a knapsack capacity of w .

Expression Breakdown

$dp[i - 1][w - weights[i - 1]] + values[i - 1]$

This expression is used to determine the value if we include the (i)-th item in the knapsack.

1. $dp[i - 1][w - weights[i - 1]]$:

- This part looks at the maximum value achievable with the first $i - 1$ items and a reduced knapsack capacity of $w - weights[i - 1]$.
- Essentially, it checks the value of the knapsack if we were to include the (i)-th item, thus reducing the capacity by the weight of the (i)-th item.

2. $values[i - 1]$:

- This is the value of the (i)-th item itself.

Combining Both Parts

- By adding $values[i - 1]$ to $dp[i - 1][w - weights[i - 1]]$, we get the total value if the (i)-th item is included in the knapsack.
- The max function then compares this value with $dp[i - 1][w]$ (the value without including the (i)-th item) to decide whether to include the (i)-th item or not.

Example

Let's say we have:

- $weights = [2, 3, 4]$
- $values = [3, 4, 5]$
- $W = 5$

For $i = 2$ (second item) and $w = 5$:

- $weights[1] = 3$
- $values[1] = 4$

The expression $dp[1][5 - 3] + 4$ translates to:

- $dp[1][2] + 4$

If $dp[1][2]$ (maximum value with the first item and capacity 2) is 3, then:

- $dp[1][2] + 4 = 3 + 4 = 7$

So, $dp[2][5]$ will be the maximum of $dp[1][5]$ (not including the second item) and 7 (including the second item).

Basic Analysis

0/1 Knapsack is the most basic dynamic programming problem

- Recall that there is only one item of each kind, and you can choose to put it in or not.
- Use subproblems to define the state
 - $F[i, v]$ represents the maximum value that can be obtained by putting the first i items into a knapsack with a capacity of v
 - Then its state transition equation is:
- Dynamic Programming characteristics
 - Optimally solve subproblems

$$F[i, v] = \max\{F[i - 1, v], F[i - 1, v - C_i] + W_i\}$$

$F[0, 0..V] \leftarrow 0$

for $i \leftarrow 1$ to N

 for $v \leftarrow C_i$ to V

$F[i, v] \leftarrow \max\{F[i - 1, v], F[i - 1, v - C_i] + W_i\}$

Testing Approaches

- Think through all possibilities and edge cases
- Full coverage (all paths tested)
- Write a script to cover testing possibilities/variations
- Ask LLM for help with the above
- Latest research suggests generating code with one LLM and testing it with another

Imagine real-world scenarios

Increased complexity when optimizing

- Amazon airplane carrying packages/Fed Ex delivery truck
 - Space needed
 - Weight allowed (including weight distribution)
 - Destination
 - Expected delivery date
 - Fragile or hazardous item considerations
- Marketing campaigns optimizing reach
 - Which types of channels (TV, online ads, print media)
 - Cost of channels
 - Reach of channels
 - Effectiveness of combinations of channels
 - Time/Length of advertisements
 - Target audience

Week 5 HW - a more complicated knapsack

You are given a set of N items, each with a weight and a value. Your goal is to select a subset of these items to maximize the total value without exceeding a given capacity W . However, the problem has additional complexities:

1. **Mandatory Items:** Some items are mandatory and must be included in the knapsack if the total weight does not exceed W . If including all mandatory items exceeds W , it's impossible to satisfy the constraints.
2. **Conflicting Items:** Certain pairs of items conflict with each other. If you select one item from a conflicting pair, you cannot select the other.
3. **Grouped Items Constraint:** Items are divided into groups. You can select at most K items from each group.
4. **Bonuses for Item Sets:** Selecting specific combinations of items grants additional bonus value.

Your task is to determine the maximum total value achievable under these constraints.

$N = 5$
 $G = 2$
 $K = 2$
 $W = 15$
 Items:
 Group: [1, 1, 2, 2, 2]
 Weights: [5, 7, 4, 3, 2]
 Values: [10, 13, 8, 6, 4]
 $M = 1$
 Mandatory Items: [1]
 $C = 1$
 Conflicting Pairs: [(2, 3)]
 $B = 1$
 Bonus Sets: [(1, 4), 5])

Items Selected: 1, 2, 4 (Initial selection)
 Total Weight: $5 + 7 + 3 = 15$ (equals W)
 Total Value: $10 + 13 + 6 + 5$ (Bonus) = 34