**ĐẠI HỌC HUẾ**

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ



🙠🙟🕮🙝🙢

**Icon

Description automatically generated**

**BÁO CÁO**

**ĐỒ ÁN/TIỂU LUẬN/BÀI TẬP LỚN**

**NĂM HỌC 2020-2021**

**Giảng viên hướng dẫn: Hồ Quốc Dũng**

**Lớp: K1 KHDL&TTNT**

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày 12 tháng 7 năm 2021**

**ĐẠI HỌC HUẾ**

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ



🙠🙟🕮🙝🙢

**Icon

Description automatically generated**

(MẪU BÌA PHỤ)

**BÁO CÁO**

**ĐỒ ÁN/TIỂU LUẬN/BÀI TẬP LỚN**

**NĂM HỌC 2020-2021**

**Giảng viên hướng dẫn: Hồ Quốc Dũng**

**Lớp: K1 KHDL&TTNT**

**Sinh viên thực hiện: Đặng Văn Thắng**

*(ký tên và ghi rõ họ tên)*

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày 12 tháng 7 năm 2021**

1. Giải thuật đệ quy  
BT01. Tháp Hà Nội  
BT02. Ước số chung lớn nhất  
BT03. Tính giai thừa của 1 số  
BT04. Bài toán mã đi tuần  
BT05. Bài toán 8 quân hậu  
2. Danh sách liên kết  
BT06. Cài đặt danh sách liên kết đơn  
BT07. Cài đặt danh sách liên kết kép  
BT08. Cài đặt ngăn xếp - stack  
BT09. Cài đặt hàng đợi - queue  
3. Cây  
BT10. Cài đặt cây - duyệt cây theo thứ tự trước  
BT11. Cài đặt cây - duyệt cây theo thứ tự sau  
4. Đồ Thị  
BT12. Cài đặt đồ thị vô hướng  
BT13. Cài đặt đồ thị có hướng  
5. Sắp xếp và tìm kiếm  
BT14.  Cài đặt thuật toán sắp xếp chọn  
BT15.  Cài đặt thuật toán sắp xếp chèn  
BT16.  Cài đặt thuật toán sắp xếp nổi bọt  
BT17.  Cài đặt thuật toán sắp xếp nhanh - quick sort  
BT18.  Cài đặt thuật toán heap sort  
BT19.  Cài đặt thuật toán sắp xếp trộn - merge sort

Link github: *https://github.com/dvthang774/GIAI\_THUAT*

# Chương 1: Cấu Trúc Và Dữ Liệu Và Giải Thuật Python

1. Giải thuật đệ quy
   1. BT01: Tháp Hà Nội:

**def** thapHaNoi(n, toaMot, toaHai, toaBa):

**if** n == 1:

**print**("Chuyen tu", toaMot, "sang", toaBa)

**else**:

thapHaNoi(n-1,toaMot,toaBa ,toaHai)

**print**("Chuyen tu", toaMot, "sang", toaBa)

thapHaNoi(n-1,toaHai,toaMot, toaBa)

* 1. BT02: Ước số chung lớn nhất

**def** ucln(a, b):

**if** (b == 0):

**return** a

**return** uscln(b, a % b)

* 1. BT03: Tính giai thừa của 1 số:

**def** giai\_thua(n):

**if** n == 0:

**return** 1

**else**:

**return** n \* giai\_thua(n - 1)

print(giai\_thua(9))

* 1. BT04: Bài toán mã đi tuần

n = 8

**def** isSafe(x, y, board):

*'''*

*A utility function to check if i,j are valid indexes*

*for N\*N chessboard*

*'''*

**if**(x >= 0 **and** y >= 0 **and** x < n **and** y < n **and** board[x][y] == -1):

**return** **True**

**return** **False**

**def** printSolution(n, board):

*'''*

*A utility function to print Chessboard matrix*

*'''*

**for** i **in** range(n):

**for** j **in** range(n):

print(board[i][j], end=' ')

print()

**def** solveKT(n):

*'''*

*This function solves the Knight Tour problem using*

*Backtracking. This function mainly uses solveKTUtil()*

*to solve the problem. It returns false if no complete*

*tour is possible, otherwise return true and prints the*

*tour.*

*Please note that there may be more than one solutions,*

*this function prints one of the feasible solutions.*

*'''*

*# Initialization of Board matrix*

board = [[-1 **for** i **in** range(n)]**for** i **in** range(n)]

*# move\_x and move\_y define next move of Knight.*

*# move\_x is for next value of x coordinate*

*# move\_y is for next value of y coordinate*

move\_x = [2, 1, -1, -2, -2, -1, 1, 2]

move\_y = [1, 2, 2, 1, -1, -2, -2, -1]

*# Since the Knight is initially at the first block*

board[0][0] = 0

*# Step counter for knight's position*

pos = 1

*# Checking if solution exists or not*

**if**(**not** solveKTUtil(n, board, 0, 0, move\_x, move\_y, pos)):

print("Solution does not exist")

**else**:

printSolution(n, board)

**def** solveKTUtil(n, board, curr\_x, curr\_y, move\_x, move\_y, pos):

*'''*

*A recursive utility function to solve Knight Tour*

*problem*

*'''*

**if**(pos == n\*\*2):

**return** **True**

*# Try all next moves from the current coordinate x, y*

**for** i **in** range(8):

new\_x = curr\_x + move\_x[i]

new\_y = curr\_y + move\_y[i]

**if**(isSafe(new\_x, new\_y, board)):

board[new\_x][new\_y] = pos

**if**(solveKTUtil(n, board, new\_x, new\_y, move\_x, move\_y, pos+1)):

**return** **True**

*# Backtracking*

board[new\_x][new\_y] = -1

**return** **False**

*# Driver Code*

**if** \_\_name\_\_ == "\_\_main\_\_":

*# Function Call*

solveKT(n)

* 1. BT05: Bài toán 8 quân hậu:

**class** **NQueens**:

*"""Generate all valid solutions for the n queens puzzle"""*

**def** \_\_init\_\_(self, size):

*# Store the puzzle (problem) size and the number of valid solutions*

self.size = size

self.solutions = 0

self.solve()

**def** solve(self):

*"""Solve the n queens puzzle and print the number of solutions"""*

positions = [-1] \* self.size

self.put\_queen(positions, 0)

print("Found", self.solutions, "solutions.")

**def** put\_queen(self, positions, target\_row):

*"""*

*Try to place a queen on target\_row by checking all N possible cases.*

*If a valid place is found the function calls itself trying to place a queen*

*on the next row until all N queens are placed on the NxN board.*

*"""*

*# Base (stop) case - all N rows are occupied*

**if** target\_row == self.size:

self.show\_full\_board(positions)

*# self.show\_short\_board(positions)*

self.solutions += 1

**else**:

*# For all N columns positions try to place a queen*

**for** column **in** range(self.size):

*# Reject all invalid positions*

**if** self.check\_place(positions, target\_row, column):

positions[target\_row] = column

self.put\_queen(positions, target\_row + 1)

**def** check\_place(self, positions, ocuppied\_rows, column):

*"""*

*Check if a given position is under attack from any of*

*the previously placed queens (check column and diagonal positions)*

*"""*

**for** i **in** range(ocuppied\_rows):

**if** positions[i] == column **or** \

positions[i] - i == column - ocuppied\_rows **or** \

positions[i] + i == column + ocuppied\_rows:

**return** **False**

**return** **True**

**def** show\_full\_board(self, positions):

*"""Show the full NxN board"""*

**for** row **in** range(self.size):

line = ""

**for** column **in** range(self.size):

**if** positions[row] == column:

line += "Q "

**else**:

line += ". "

print(line)

print("**\n**")

**def** show\_short\_board(self, positions):

*"""*

*Show the queens positions on the board in compressed form,*

*each number represent the occupied column position in the corresponding row.*

*"""*

line = ""

**for** i **in** range(self.size):

line += str(positions[i]) + " "

print(line)

**def** main():

*"""Initialize and solve the n queens puzzle"""*

NQueens(8)

**if** \_\_name\_\_ == "\_\_main\_\_":

*# execute only if run as a script*

main()

1. Danh sách liên kết:
   1. BT06. Cài đặt danh sách liên kết đơn:

# Node class

**class** Node:

# Function to initialize the node object

**def** \_\_init\_\_(self, data):

self.data = data # Assign data

self.next = None # Initialize

# next as null

# Linked List class

**class** LinkedList:

# Function to initialize the Linked

# List object

**def** \_\_init\_\_(self):

self.head = None

* 1. BT07. Cài đặt danh sách liên kết kép:

# Node of a doubly linked list

**class** Node:

**def** \_\_init\_\_(self, next=None, prev=None, data=None):

self.next = next # reference to next node in DLL

self.prev = prev # reference to previous node in DLL

self.data = data

* 1. BT08. Cài đặt ngăn xếp – stack

**class** Stack:

**def** \_\_init\_\_(self):

self.stack = []

**def** pop(self):

**if** len(self.stack) < 1:

**return** None

**return** self.stack.pop()

**def** push(self, item):

self.stack.append(item)

**def** size(self):

**return** len(self.stack)

* 1. BT09. Cài đặt hàng đợi – queue

**class** Queue:

**def** \_\_init\_\_(self):

self.queue = []

**def** enqueue(self, item):

self.queue.append(item)

**def** dequeue(self):

**if** len(self.queue) < 1:

return None

**return** self.queue.pop(0)

**def** size(self):

**return** len(self.queue)

1. Cây:
   1. BT10. Cài đặt cây - duyệt cây theo thứ tự trước

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

# Insert Node

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

# Print the Tree

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

# Inorder traversal

# Left -> Root -> Right

def inorderTraversal(self, root):

res = []

if root:

res = self.inorderTraversal(root.left)

res.append(root.data)

res = res + self.inorderTraversal(root.right)

return res

root = Node(27)

root.insert(14)

root.insert(35)

root.insert(10)

root.insert(19)

root.insert(31)

root.insert(42)

print(root.inorderTraversal(root))

* 1. BT11. Cài đặt cây - duyệt cây theo thứ tự sau:

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

# Insert Node

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

# Print the Tree

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

# Preorder traversal

# Root -> Left ->Right

def PreorderTraversal(self, root):

res = []

if root:

res.append(root.data)

res = res + self.PreorderTraversal(root.left)

res = res + self.PreorderTraversal(root.right)

return res

root = Node(27)

root.insert(14)

root.insert(35)

root.insert(10)

root.insert(19)

root.insert(31)

root.insert(42)

print(root.PreorderTraversal(root))

1. Đồ thị
   1. BT12. Cài đặt đồ thị vô hướng

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**def** veNgangTren():

**for** i **in** range(1,4):

df[i][0] = '-'

veNgangTren()

**def** veNgangDuoi():

**for** i **in** range(1,4):

df[i][4] = '-'

veNgangDuoi()

def veCheoPhaiTrai():

j = 1

for i in reversed(range(1,4)):

df[i][j] = "/"

j +=1

**def** veCheoTraiPhai():

**for** i **in** reversed(range(1,4)):

df[i][i] = "**\\**"

**def** veDuongThangXuongTrai():

**for** i **in** range(1,4):

df[0][i] = "|"

**def** veDuongThangXuongPhai():

**for** i **in** range(1,4):

df[4][i] = "|"

veDuongThangXuongPhai()

A = np.zeros( (5, 5) )

df = pd.DataFrame(data=A)

lien\_ket = {("A","B"),("B","C"),("C","D"),("B","D"),("D","D")}

lien\_ket

**if** ("A","C") **in** lien\_ket:

veCheoTraiPhai()

**if** ("A","B") **in** lien\_ket:

veNgangTren()

**if** ("B","C") **in** lien\_ket:

veDuongThangXuongPhai()

**if** ("C","D") **in** lien\_ket:

veNgangDuoi()

**if** ("B","D") **in** lien\_ket:

veCheoPhaiTrai()

**if** ("A","D") **in** lien\_ket:

veDuongThangXuongTrai()

**if** ("A","C") **in** lien\_ket:

veCheoTraiPhai()

* 1. BT13. Cài đặt đồ thị có hướng

**import** **networkx** **as** **nx**

**import** **matplotlib.pyplot** **as** **plt**

G = nx.DiGraph()

G.add\_edges\_from([('A','B'),('B','C'),('C','D'),('A','C'),('D','D')])

red\_edges = [('A', 'C'), ('E', 'C')]

edge\_colours = ['black' **if** **not** edge **in** red\_edges **else** 'red'

**for** edge **in** G.edges()]

black\_edges = [edge **for** edge **in** G.edges() **if** edge **not** **in** red\_edges]

pos = nx.spring\_layout(G)

nx.draw\_networkx\_nodes(G, pos, cmap=plt.get\_cmap('jet'),

node\_size = 500)

nx.draw\_networkx\_labels(G, pos)

nx.draw\_networkx\_edges(G, pos, edgelist=black\_edges, arrows=**True**)

plt.show()

1. Sắp xếp và tìm kiếm:
   1. BT14. Cài đặt thuật toán sắp xếp chọn

def selection\_sort(nums):

# This value of i corresponds to how many values were sorted

for i in range(len(nums)):

# We assume that the first item of the unsorted segment is the smallest

lowest\_value\_index = i

# This loop iterates over the unsorted items

for j in range(i + 1, len(nums)):

if nums[j] < nums[lowest\_value\_index]:

lowest\_value\_index = j

# Swap values of the lowest unsorted element with the first unsorted

# element

nums[i], nums[lowest\_value\_index] = nums[lowest\_value\_index], nums[i]

# Verify it works

random\_list\_of\_nums = [12, 8, 3, 20, 11]

selection\_sort(random\_list\_of\_nums)

print(random\_list\_of\_nums)

* 1. BT15. Cài đặt thuật toán sắp xếp chèn:

def insertion\_sort(nums):

# Start on the second element as we assume the first element is sorted

for i in range(1, len(nums)):

item\_to\_insert = nums[i]

# And keep a reference of the index of the previous element

j = i - 1

# Move all items of the sorted segment forward if they are larger than

# the item to insert

while j >= 0 and nums[j] > item\_to\_insert:

nums[j + 1] = nums[j]

j -= 1

# Insert the item

nums[j + 1] = item\_to\_insert

# Verify it works

random\_list\_of\_nums = [9, 1, 15, 28, 6]

insertion\_sort(random\_list\_of\_nums)

print(random\_list\_of\_nums)

* 1. BT16. Cài đặt thuật toán sắp xếp nổi bọt:

def bubble\_sort(nums):

# We set swapped to True so the loop looks runs at least once

swapped = True

while swapped:

swapped = False

for i in range(len(nums) - 1):

if nums[i] > nums[i + 1]:

# Swap the elements

nums[i], nums[i + 1] = nums[i + 1], nums[i]

# Set the flag to True so we'll loop again

swapped = True

# Verify it works

random\_list\_of\_nums = [5, 2, 1, 8, 4]

bubble\_sort(random\_list\_of\_nums)

print(random\_list\_of\_nums)

* 1. BT17. Cài đặt thuật toán sắp xếp nhanh - quick sort:

# There are different ways to do a Quick Sort partition, this implements the

# Hoare partition scheme. Tony Hoare also created the Quick Sort algorithm.

def partition(nums, low, high):

# We select the middle element to be the pivot. Some implementations select

# the first element or the last element. Sometimes the median value becomes

# the pivot, or a random one. There are many more strategies that can be

# chosen or created.

pivot = nums[(low + high) // 2]

i = low - 1

j = high + 1

while True:

i += 1

while nums[i] < pivot:

i += 1

j -= 1

while nums[j] > pivot:

j -= 1

if i >= j:

return j

# If an element at i (on the left of the pivot) is larger than the

# element at j (on right right of the pivot), then swap them

nums[i], nums[j] = nums[j], nums[i]

def quick\_sort(nums):

# Create a helper function that will be called recursively

def \_quick\_sort(items, low, high):

if low < high:

# This is the index after the pivot, where our lists are split

split\_index = partition(items, low, high)

\_quick\_sort(items, low, split\_index)

\_quick\_sort(items, split\_index + 1, high)

\_quick\_sort(nums, 0, len(nums) - 1)

# Verify it works

random\_list\_of\_nums = [22, 5, 1, 18, 99]

quick\_sort(random\_list\_of\_nums)

print(random\_list\_of\_nums)

* 1. BT18. Cài đặt thuật toán heap sort:

def heapify(arr, n, i):

largest = i # Initialize largest as root

l = 2 \* i + 1 # left = 2\*i + 1

r = 2 \* i + 2 # right = 2\*i + 2

# See if left child of root exists and is

# greater than root

if l < n and arr[i] < arr[l]:

largest = l

# See if right child of root exists and is

# greater than root

if r < n and arr[largest] < arr[r]:

largest = r

# Change root, if needed

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i] # swap

# Heapify the root.

heapify(arr, n, largest)

# The main function to sort an array of given size

def heapSort(arr):

n = len(arr)

# Build a maxheap.

# Since last parent will be at ((n//2)-1) we can start at that location.

for i in range(n // 2 - 1, -1, -1):

heapify(arr, n, i)

# One by one extract elements

for i in range(n - 1, 0, -1):

arr[i], arr[0] = arr[0], arr[i] # swap

heapify(arr, i, 0)

# Driver code to test above

arr = [1, -1, 13, 5, 6, 7]

heapSort(arr)

n = len(arr)

print("Sorted array is")

for i in range(n):

print("%d" % arr[i])

* 1. BT19. Cài đặt thuật toán sắp xếp trộn - merge sort:

def merge(left\_list, right\_list):

sorted\_list = []

left\_list\_index = right\_list\_index = 0

# We use the list lengths often, so its handy to make variables

left\_list\_length, right\_list\_length = len(left\_list), len(right\_list)

for \_ in range(left\_list\_length + right\_list\_length):

if left\_list\_index < left\_list\_length and right\_list\_index < right\_list\_length:

# We check which value from the start of each list is smaller

# If the item at the beginning of the left list is smaller, add it

# to the sorted list

if left\_list[left\_list\_index] <= right\_list[right\_list\_index]:

sorted\_list.append(left\_list[left\_list\_index])

left\_list\_index += 1

# If the item at the beginning of the right list is smaller, add it

# to the sorted list

else:

sorted\_list.append(right\_list[right\_list\_index])

right\_list\_index += 1

# If we've reached the end of the of the left list, add the elements

# from the right list

elif left\_list\_index == left\_list\_length:

sorted\_list.append(right\_list[right\_list\_index])

right\_list\_index += 1

# If we've reached the end of the of the right list, add the elements

# from the left list

elif right\_list\_index == right\_list\_length:

sorted\_list.append(left\_list[left\_list\_index])

left\_list\_index += 1

return sorted\_list

def merge\_sort(nums):

# If the list is a single element, return it

if len(nums) <= 1:

return nums

# Use floor division to get midpoint, indices must be integers

mid = len(nums) // 2

# Sort and merge each half

left\_list = merge\_sort(nums[:mid])

right\_list = merge\_sort(nums[mid:])

# Merge the sorted lists into a new one

return merge(left\_list, right\_list)

# Verify it works

random\_list\_of\_nums = [120, 45, 68, 250, 176]

random\_list\_of\_nums = merge\_sort(random\_list\_of\_nums)

print(random\_list\_of\_nums)

# Chương 2: Cấu Trúc Và Dữ Liệu Và Giải Thuật R

1. Giải thuật đệ quy
   1. BT01. Tháp Hà Nội

thap\_ha\_noi <- function(n, toa1, toa2, toa3){

if (n==1){

print(toa1)

print('|')

print(toa3)

print('')

} else {

thap\_ha\_noi(n-1,toa1, toa3, toa2)

print(toa1)

print('|')

print(toa3)

print('')

thap\_ha\_noi(n-1,toa2, toa1, toa3)

}

}

thap\_ha\_noi(3,'A','B','C')

* 1. BT02. Ước số chung lớn nhất

ucln <- function(a, b){

if(b == 0){

return(a)

} else {

return(ucln(b, a%%b))

}

}

ucln(15,10)

* 1. BT03. Tính giai thừa của 1 số

#Giai Thua

giai\_thua <- function(n){

if(n == 0){

return(1)

} else {

return(n\*giai\_thua(n-1))

}

}

giai\_thua(3)

* 1. BT04. Bài toán mã đi tuần:

install.packages("purrr")

install.packages("dplyr")

install.packages("tidyverse")

library(purrr)

library(dplyr)

library(tidyverse)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Knight offsets i.e. the possible movements of a knight from the current location

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

knight\_offsets <- matrix(c(

1, 2,

2, 1,

-2, 1,

-1, 2,

2, -1,

1, -2,

-1, -2,

-2, -1

), ncol = 2, byrow = TRUE)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#' Recurisvely calculate moves for a knight to complete a tour

#'

#' @param this\_move proposed next move. 2 element numeric vector of (row, col)

#' position at which to place the knight next

#' @param moves list of vectors. Each vector is length=2 and indicates the

#' row/column locations of the knight's tour so far

#' @param visited 8x8 logical matrix which indicates whether or not a square has been

#' visited by the knight already. When called by the user, this matrix

#' must only contain FALSE

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

move\_knight <- function(this\_move, moves, visited) {

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Mark the move as visited

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

moves <- append(moves, list(this\_move))

visited[this\_move[1] + (this\_move[2] - 1)\*8] <- TRUE

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# termination if all visited

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

if (all(visited)) {

return(moves)

}

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Find all possible moves from this position

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

next\_move <- cbind(knight\_offsets[,1] + this\_move[1], knight\_offsets[,2] + this\_move[2])

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# keep only moves that remain on the board

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

on\_board <- next\_move[,1] %in% 1:8 & next\_move[,2] %in% 1:8

next\_move <- next\_move[on\_board,,drop=FALSE]

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Keep only moves that target a location that has not yet been visited

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

not\_yet\_visited <- !visited[next\_move]

next\_move <- next\_move[not\_yet\_visited,, drop = FALSE]

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Recurse over every possible next move

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

for (i in seq\_len(nrow(next\_move))) {

res <- move\_knight(next\_move[i,, drop = FALSE], moves, visited)

if (!is.null(res)) {

return(res)

}

}

NULL

}

system.time({

moves <- move\_knight(c(4, 8), moves = list(), visited = matrix(FALSE, 8, 8))

})

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

#' Convert results to a data.frame for ggplot

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

moves\_df <- as.data.frame(do.call(rbind, moves))

moves\_df <- set\_names(moves\_df, c('x', 'y'))

moves\_df$idx <- 1:nrow(moves\_df)

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Plot the knight's tour

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ggplot(moves\_df, aes(x, y)) +

geom\_tile(aes(fill=as.logical((x+y)%%2)), colour = 'black') +

geom\_path(alpha = 0.7, linetype = 1, size = 0.25) +

geom\_text(aes(label = idx)) +

scale\_fill\_manual(values = c('grey70', 'white')) +

theme\_void() +

theme(legend.position = 'none') +

coord\_equal() +

labs(

title = "A knight's tour in #RStats"

)

* 1. BT05. Bài toán 8 quân hậu

library(tidyverse)

#-----------------------------------------------------------------------------

#' Try and place a Queen given a vector of positions of the current Queens

#'

#' This function calls itself recursively for every valid placement of the

#' next queen.

#'

#' @param queens A vector of integers representing the column placement

#' of queens so far. The index within this list

#' is the row, and the value is the column.

#' To generate all solutions, pass in an empty vector (the default)

#'

#' e.g. queens = c(1, 4, 7) corresponds to queens placed at c(1, 1), c(2, 4) and

#' c(3, 7)

#'

#'

#' ---------------------------------

#' | | | | | | | | |

#' ---------------------------------

#' | | | | | | | | |

#' ---------------------------------

#' | | | | | | | | |

#' ---------------------------------

#' | | | | | | | | |

#' ---------------------------------

#' | | | | | | | Q | | 3rd row, 7th column

#' ---------------------------------

#' | | | | Q | | | | | 2nd row, 4th column

#' ---------------------------------

#' | Q | | | | | | | | 1st row, 1st column

#' ---------------------------------

#'

#'

#'

#'

#' @return a list where each element is a vector of 8 integers

#' i.e. a solution to the 8 queens problem

#-----------------------------------------------------------------------------

place\_queen <- function(queens=c()) {

#---------------------------------------------------------------------------

# If there are 8 queens placed, then this must be a solution.

#---------------------------------------------------------------------------

if (length(queens) == 8) {

return(list(queens))

}

#---------------------------------------------------------------------------

# Figure out where a queen can be placed in the next row.

# Drop all columns that have already been taken - since we

# can't place a queen below an existing queen

#---------------------------------------------------------------------------

possible\_placements <- setdiff(1:8, queens)

#---------------------------------------------------------------------------

# For each queen already on the board, find the diagonal

# positions that it can see in this row.

#---------------------------------------------------------------------------

diag\_offsets <- seq.int(length(queens), 1)

diags <- c(queens + diag\_offsets, queens - diag\_offsets)

diags <- diags[diags > 0 & diags < 9]

#---------------------------------------------------------------------------

# Drop these diagonal columns from possible placements

#---------------------------------------------------------------------------

possible\_placements <- setdiff(possible\_placements, diags)

#---------------------------------------------------------------------------

# For each possible placement, try and place a queen

#---------------------------------------------------------------------------

possible\_placements %>%

map(~place\_queen(c(queens, .x))) %>%

keep(~length(.x) > 0) %>%

flatten()

}

#-----------------------------------------------------------------------------

#' Plot a single solution

#' @param queens a vector of 8 integers giving the column positions of 8 queens

#-----------------------------------------------------------------------------

plot\_single\_8queens <- function(queens, title = NULL) {

queens\_df <- tibble(cols = queens, rows=1:8)

board\_df <- expand.grid(cols = 1:8, rows = 1:8) %>%

mutate(check = (cols + rows) %%2 == 1)

p <- ggplot(queens\_df, aes(rows, cols)) +

geom\_tile(data=board\_df, aes(fill=check), colour='black') +

geom\_text(label='\u2655', family="Arial Unicode MS", size = 8) +

theme\_void() +

coord\_equal() +

scale\_fill\_manual(values = c('TRUE'='white', 'FALSE'='grey70')) +

theme(

legend.position = 'none'

)

if (is.null(title)) {

p <- p + labs(title = paste("Queens:", deparse(as.numeric(queens))))

} else {

p <- p + labs(title = title)

}

}

#-----------------------------------------------------------------------------

# Start with no queens placed and generate all solutions.

#-----------------------------------------------------------------------------

solutions <- place\_queen()

v=1:8

f=function(q){L=length(q)

if(L==8){q}else{flatten(map(setdiff(v,c(q,q+L:1,q-L:1)),~f(c(q,.))))}}

s=data.frame(c=unlist(f(c())),r=v,x=rep(1:92,e=8),z=3)

b=mutate(crossing(c=v,r=v),z=(c+r)%%2)

g=geom\_tile

ggplot(s,aes(r,c,fill=z))+g(d=b)+g()+facet\_wrap(~x)

1. Danh sách liên kết
   1. BT06. Cài đặt danh sách liên kết đơn

# danh sach lien ket

lst <- list() # creates an empty (length zero) list

lst[[1]] <- 1 # automagically extends the lst

lst[[2]] <- 2 # ditto

lst

lst <- list(1, 2, 3, 4, 5) # a list of 5 items

lst <- vector("list", 10000) # 10000 NULLs

lst[[1]] <- 1

lst[[10000]] <- 10000 # lst now contains 1, NULL, ..., NULL, 10000

lst

* 1. BT07. Cài đặt danh sách liên kết kép:

# danh sach lien ket

lst <- list() # creates an empty (length zero) list

lst[[1]] <- 1 # automagically extends the lst

lst[[2]] <- 2 # ditto

lst

lst <- list(1, 2, 3, 4, 5) # a list of 5 items

lst <- vector("list", 10000) # 10000 NULLs

lst[[1]] <- 1

lst[[10000]] <- 10000 # lst now contains 1, NULL, ..., NULL, 10000

lst

* 1. BT08. Cài đặt ngăn xếp – stack

install.packages('dequer')

library(dequer)

# Stack

s <- stack()

for (i in 1:3) push(s, i)

str(s)

pop(s)

str(s)

pop(s)

str(s)

pop(s)

str(s)

* 1. BT09. Cài đặt hàng đợi – queue:

install.packages('dequer')

library(dequer)

# Queue

q <- queue()

for (i in 1:3) pushback(q, i)

str(q)

## queue of 3

## $ : int 1

## $ : int 2

## $ : int 3

pop(q)

## [1] 1

str(q)

## queue of 2

## $ : int 2

## $ : int 3

pop(q)

## [1] 2

str(q)

## queue of 1

## $ : int 3

pop(q)

## [1] 3

str(q)

## queue()

1. Cây
   1. BT10. Cài đặt cây - duyệt cây theo thứ tự trước

install.packages('data.tree')

library(data.tree)

# thiet lap cay

acme <- Node$new("A")

accounting <- acme$AddChild("B")

software <- accounting$AddChild("C")

standards <- accounting$AddChild("D")

research <- acme$AddChild("E")

newProductLine <- research$AddChild("F")

newLabs <- research$AddChild("G")

it <- acme$AddChild("H")

outsource <- it$AddChild("I")

agile <- it$AddChild("J")

goToR <- it$AddChild("K")

print(acme)

# Duyet cay Truoc

acme$Get('level')

# Duyet Cay sau

acme$Get('level', traversal = "post-order")

* 1. BT11. Cài đặt cây - duyệt cây theo thứ tự sau

install.packages('data.tree')

library(data.tree)

# thiet lap cay

acme <- Node$new("A")

accounting <- acme$AddChild("B")

software <- accounting$AddChild("C")

standards <- accounting$AddChild("D")

research <- acme$AddChild("E")

newProductLine <- research$AddChild("F")

newLabs <- research$AddChild("G")

it <- acme$AddChild("H")

outsource <- it$AddChild("I")

agile <- it$AddChild("J")

goToR <- it$AddChild("K")

print(acme)

# Duyet cay Truoc

acme$Get('level')

# Duyet Cay sau

acme$Get('level', traversal = "post-order")

1. Đồ thị:
   1. BT12. Cài đặt đồ thị vô hướng

# DO THI

install.packages('igraph')

library(igraph)

# Vo Huong

help(graph)

vo\_huong <- make\_graph( ~ A-B-C-D-A, E-A:B:C:D, directed = FALSE)

plot(vo\_huong)

* 1. BT13. Cài đặt đồ thị có hướng:

# DO THI

install.packages('igraph')

library(igraph)

#Co huong

help(make\_directed\_graph)

a = make\_graph(c(1, 2, 2, 3, 3, 4, 5, 6, 4,1, 2,4, 4,5, 6,1), directed = TRUE)

plot(a)

1. Sắp xếp và tìm kiếm
   1. BT14. Cài đặt thuật toán sắp xếp chọn:

# sap xep chon

vec <- sample(1:100)

vec

sort <- function(x) {

n<- length(x)

for (i in 1:(n-1)) {

for (j in (i+1):n) {

if(x[j] < x[i]) {

temp <-x[i]

x[i] <- x[j]

x[j] <- temp

}

}

}

return(x)

}

sort(vec)

selfsort <- function(x) {

if (length(x)>1) {

min <- which.min(x)

c(x[min], selfsort(x[-min]))

} else x

}

selfsort(vec)

* 1. BT15. Cài đặt thuật toán sắp xếp chèn

# sap xep chen

insertionsort\_function <- function(A){

for (j in 2:length(A)) {

key = A[j]

# insert A[j] into sorted sequence A[1,...,j-1]

i = j - 1

while (i > 0 && A[i] > key) {

A[(i + 1)] = A[i]

i = i - 1

}

A[(i + 1)] = key

}

A

}

insertionsort\_function(c(5, 2, 4, 6, 1, 3))

* 1. BT16. Cài đặt thuật toán sắp xếp nổi bọt

#sap xep noi bot

vec = c(1,-1,3,2,10,9)

bubble <- function(x){

n<-length(x)

for(j in 1:(n-1)){

for(i in 1:(n-j)){

if(x[i]>x[i+1]){

temp<-x[i]

x[i]<-x[i+1]

x[i+1]<-temp

}

}

}

return(x)

}

bubble(vec)

* 1. BT17. Cài đặt thuật toán sắp xếp nhanh - quick sort:

# quick sort

vec = c(1,-2,3,6,2,1,9)

quickSort <- function(arr) {

mid <- sample(arr, 1)

left <- c()

right <- c()

lapply(arr[arr != mid], function(d) {

if (d < mid) {

left <<- c(left, d)

}

else {

right <<- c(right, d)

}

})

if (length(left) > 1) {

left <- quickSort(left)

}

if (length(right) > 1) {

right <- quickSort(right)

}

c(left, mid, right)

}

quickSort(vec)

* 1. BT18. Cài đặt thuật toán heap sort:

#### heap sort

#### min-heap

#### bottom-top algorithm used

heap.building<-function(vec)

{

len=length(vec)

heap=vec

for (j in len:1)

{

heap=modify.heap(heap,j)

}

return(heap)

}

is.heap<-function(heap,root\_i)

{

i=root\_i

res=T

while(2\*i<=length(heap)&res)

{

son=c(heap[2\*i],heap[2\*i+1])

son=son[!is.na(son)]

res=all(heap[i]<=son)

i=i+1

}

return(res)

}

modify.heap<-function(heap,root\_i)

{

len=length(heap)

flag=1

while (root\_i\*2<=len&&flag==1)

{

left\_i=root\_i\*2

right\_i=root\_i\*2+1

flag=0

son=c(heap[left\_i],heap[right\_i])

son=son[!is.na(son)]

min\_ind=which.min(son)

if (heap[root\_i]>son[min\_ind])

{

flag=1

heap\_ind=c(left\_i,right\_i)[min\_ind]

tmp=heap[heap\_ind]

heap[heap\_ind]=heap[root\_i]

heap[root\_i]=tmp

root\_i=heap\_ind

}

}

return(heap)

}

heap.sort<-function(heap)

{

sorted=NULL

len=length(heap)

while(len>0)

{

sorted=c(sorted,heap[1])

len=length(heap)

heap[1]=heap[len]

heap=heap[1:(len-1)]

heap=modify.heap(heap,root\_i=1)

len=len-1

}

return(sorted)

}

#vec=sample(1:100)

vec = c(-1,2,1,10,3,9)

heap=heap.building(vec)

heap\_sort=heap.sort(heap)

heap\_sort

* 1. BT19. Cài đặt thuật toán sắp xếp trộn - merge sort:

mmerge<-function(a,b) {

r<-numeric(length(a)+length(b))

ai<-1; bi<-1; j<-1;

for(j in 1:length(r)) {

if((ai<=length(a) && a[ai]<b[bi]) || bi>length(b)) {

r[j] <- a[ai]

ai <- ai+1

} else {

r[j] <- b[bi]

bi <- bi+1

}

}

r

}

mmergesort<-function(A) {

if(length(A)>1) {

q <- ceiling(length(A)/2)

a <- mmergesort(A[1:q])

b <- mmergesort(A[(q+1):length(A)])

mmerge(a,b)

} else {

A

}

}

x<- sample(1:100)

mmergesort(x)