
**SCHOOL OF COMPUTER SCIENCES**

**UNIVERSITI SAINS MALAYSIA**

**CPT443 Automata Theory and Formal Languages**

**Semester 2 2021/2022**

**Title: Assignment 1 – DFA Place Finder**

**Student: Teh Zhen Rong (143955)**

# Table of Contents

# 1 Introduction

The language will accept characters ‘abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ’.

$$L = \{w: w\ consist\ of\ strings\ in\ dictionary\}$$

The language consists of district names and states in Malaysia like Butterworth, Alor Setar, Balik Pulau, Kedah, Penang, and Perak. The language also consists of country names that are member states of the United Nations. Moreover, the language also consists of continents of Earth following Wikipedia. The language will accept strings that only contain in the dictionary which are the district names in Malaysia, country name and continents of Earth. There are about 450 district strings in the dictionary that is created.

The DFA will accept the string that consists of the string in the dictionary as a whole and give strings that is otherwise trap state or not accepting state. The simple illustration of the DFA is shown in the following figure.

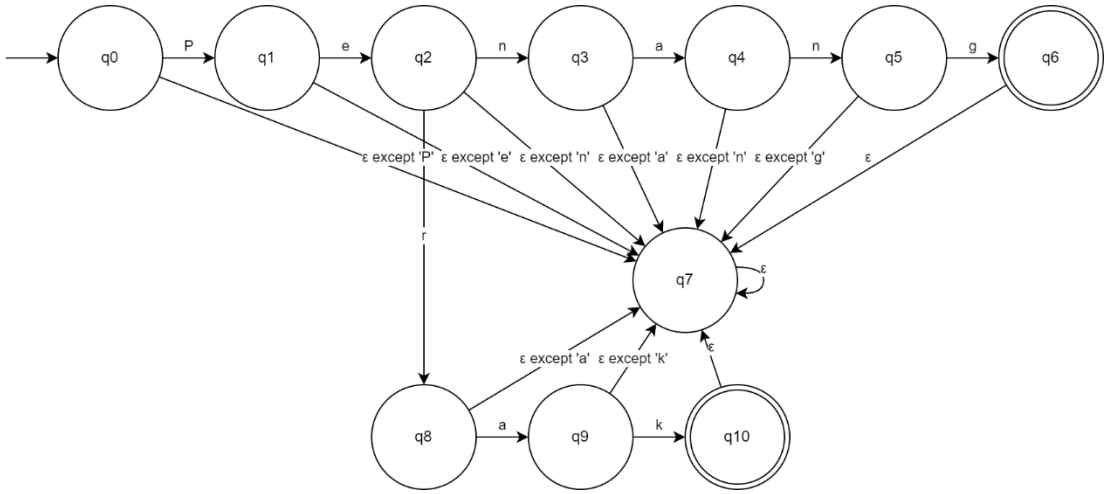


Figure 1 Sample of DFA

In Figure 1, the language accepts only string ‘Penang’ and ‘Perak’ and reject other strings.

# 2 Implementation Information

The DFA machine is implemented using Python. This is because of Python is great for fast prototyping and having large numbers of libraries.

## 2.1 Reading of input string

The input of strings can be two form which are as keyboard inputs and txt files. Both forms are read by the program and store a string. The string is then gone through a for loop to get character by character out from the string starting from left to right, top to bottom. Each character is inputted into the DFA machine and get the results of each character that is inputted. The inputted strings are then determined accepted or rejected when the character is whitespace, commas, and full stop. The inputted strings consist of many words. The DFA will accept character by character of the first word and determine whether to continue forward to next word if there are possible state that come after the first word. If the first two words are accepted and there still have possible state come after the first two word the DFA will

continue to get input from the third word. These all are done character by character. If the process breaks at midway of second word, the first word that is accepted by DFA will be treat as accepted by DFA and the DFA is initiated for inputting the second word character by character again.

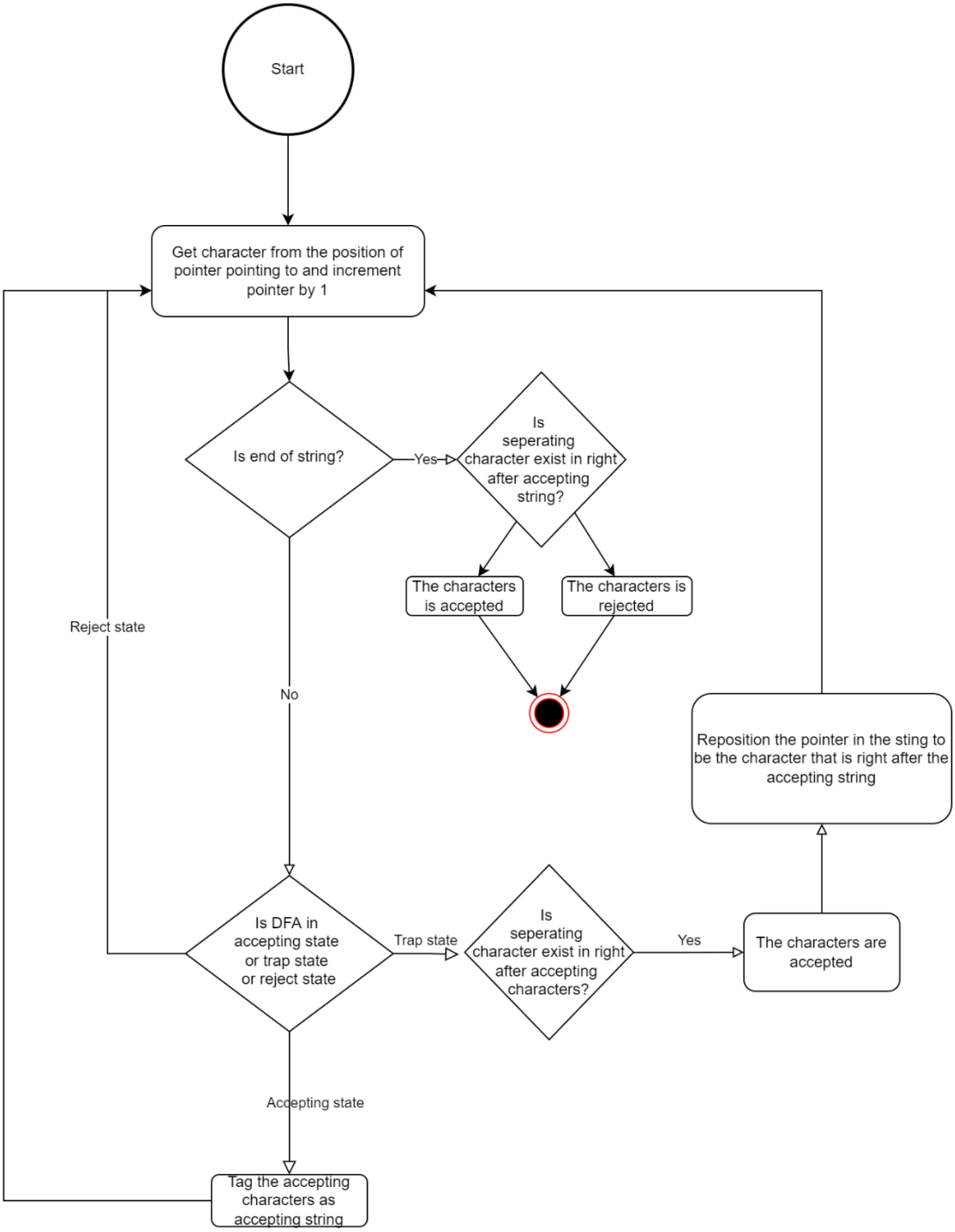


Figure 2 Flowchart of Reading Input String

## 2.2 Overview of programming construct

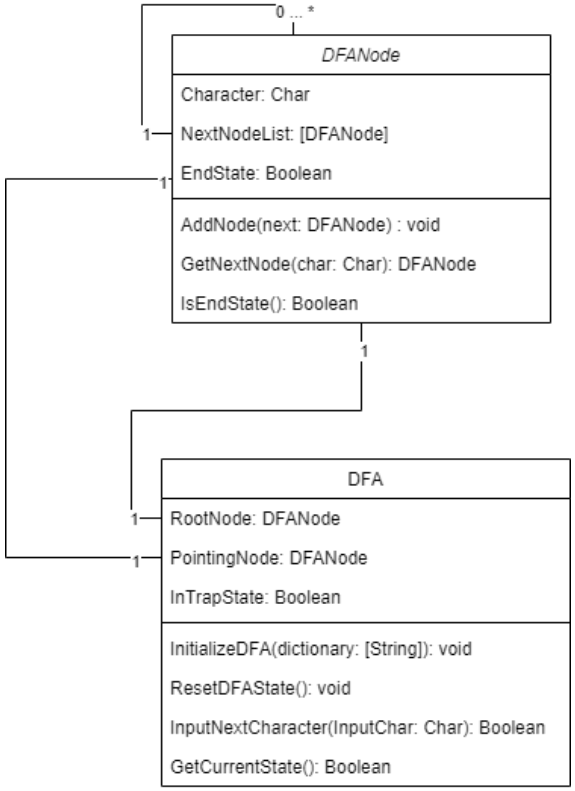


Figure 3 Class Diagram of DFA

The overall DFA consists of 2 parts which are DFA class and DFANode class. The other module like the GUI does not discuss in this part.

The DFA class has a RootNode which act as a pointer that pointing at the start state of DFA. The PointingNode act as a pointer that pointing at the current state of the DFA which will move from one state to another state according to the input character.

A DFANode consists of a list of DFANode inside it which act as a transition function of a particular state in DFA. DFANode can only move from one state to another using the NextNodeList only.

# 3 Conclusion

Throughout the assignment, I have learnt into depth of building DFA from scratch using Python. The scope of this assignment is also limited due to the amount of clean data I able to manage and get and the computational resources required by the program to run a larger scope.

Due to nature of Python I unable to have a better solution than the current one that I implemented where the pointer is unable to be implemented. In the end, the Python program of DFA is looked like a NFA where not every character of the character consider in language is stated in the transition function but I managed to make into mocking a DFA machine by assuming all character that in not stated in the transition function are moved to a trap state.

# 4 Appendix – Sample/Full programs

## 4.1 DFA

```
allElement = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ "


class dfa_node:
    def __init__(self, char=""):
        self.char = char
        self.nextNodeCharList = []
        self.nextNodeList = []
        self.isEndState = False

    def add_next_node(self, node):
        self.nextNodeList.append(node)
        self.nextNodeCharList.append(node.char)

    def is_in_next_node_list(self, char):
        return char in self.nextNodeCharList

    def get_next_node_index(self, char):
        if self.is_in_next_node_list(char):
            return self.nextNodeCharList.index(char)
        return None

    def get_next_node_list(self):
        return self.nextNodeList

    def return_relation_array(self, ancestor=""):
        ancestor = ancestor + self.char
        relationArray = []
        if self.nextNodeList:
            for i in self.nextNodeList:
                relationArray.append((ancestor, ancestor + i.char))
                if i.nextNodeCharList:
                    relationArray.extend(i.return_relation_array(ancestor))
        return relationArray


class Dfa:
    def __init__(self):
        self.root = dfa_node("")
        self.pointingNode = self.root
        self.inTrapState = False

    def initializeDfa(self, key):
        pCrawl = self.root
        length = len(key)
        for level in range(length):
            index = pCrawl.get_next_node_index(key[level])
            if index is None:
                pCrawl.add_next_node(dfa_node(char=key[level]))
                index = len(pCrawl.nextNodeList) - 1
            pCrawl = pCrawl.nextNodeList[index]
        pCrawl.isEndState = True

    def search(self, key):
        pCrawl = self.root
        length = len(key)
        for level in range(length):
            index = pCrawl.get_next_node_index(key[level])
            if index is None:
                print(key[level] + "Not found")
                return False
            print(key[level] + "Found")
            pCrawl = pCrawl.nextNodeList[index]
        if pCrawl.isEndState:
            print("Found end state. String is valid")
        return pCrawl.isEndState

    def resetDfaState(self):
        self.pointingNode = self.root
        self.inTrapState = False

    def inputNextChar(self, key):
        if len(key) == 1 and key in allElement and self.inTrapState is False:
            pCrawl = self.pointingNode
            index = pCrawl.get_next_node_index(key)
            if index is None:
                self.inTrapState = True
                return None
            self.pointingNode = pCrawl.nextNodeList[index]
            return self.getCurrentState()
        else:
            self.inTrapState = True
            return None

    def getCurrentNodePointingTo(self):
        return self.pointingNode.nextNodeCharList

    def getCurrentState(self):
        if self.inTrapState is False:
            return self.pointingNode.isEndState
        else:
            return None

    def return_relation_array(self):
        return self.root.return_relation_array()
```

## 4.2 GUI

```
import tkinter as tk

from tkinter import filedialog as fd
from tkinter import ttk

import src.DFA as DFA
import src.source as fileManagement


class UI:
    seperatorChar = [" ", ",", ".", "\n"]

    def __init__(self, root):
        dictionaryList = fileManagement.readFile("data\ListOfPlace.txt")
        self.t = DFA.Dfa()
        self.patternFound = {}
        for key in dictionaryList:
            self.t.initializeDfa(key)
        root.title("DFA Simple Machine")
        root.columnconfigure(0, weight=1)
        root.rowconfigure(0, weight=1)

        mainframe = ttk.Frame(root, padding="3 3 12 12")
        mainframe.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        # Input Text
        sampleTextFrame = ttk.Frame(mainframe, padding="3 3 12 12")
        sampleTextFrame.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))
        sampleTextLabel = ttk.Label(sampleTextFrame, text="Input Text")
        sampleTextLabel.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))
        self.sampleText = tk.Text(sampleTextFrame)
        self.sampleText.grid(column=0, row=1, sticky=(tk.N, tk.S, tk.W, tk.E))

        # Output Text
        resultTextFrame = ttk.Frame(mainframe, padding="3 3 12 12")
        resultTextFrame.grid(column=0, row=1, sticky=(tk.N, tk.S, tk.W, tk.E))
        resultTextLabel = ttk.Label(resultTextFrame, text="Output Text")
        resultTextLabel.grid(column=0, row=2, sticky=(tk.N, tk.S, tk.W, tk.E))
        self.resultText = tk.Text(resultTextFrame, bg="#f5f5f5")
        self.resultText.grid(column=0, row=3, sticky=(tk.N, tk.S, tk.W, tk.E))
        # file button
        buttonFrame = ttk.Frame(mainframe, padding="3 3 12 12")
        buttonFrame.grid(column=1, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))
        self.fileButton = ttk.Button(
            buttonFrame, text="Open txt File", command=self.fileButtonClicked
        )
        self.fileButton.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        self.dictionaryButton = ttk.Button(
            buttonFrame, text="Choose dictionary", command=self.dictionaryButtonClicked
        )
        self.dictionaryButton.grid(column=1, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        self.rerunButton = ttk.Button(
            buttonFrame, text="Run DFA", command=self.rerunButtonClicked
        )
        self.rerunButton.grid(column=2, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        self.clearLogButton = ttk.Button(
            buttonFrame, text="Clear Info Log", command=self.clearLogButtonClicked
        )
        self.clearLogButton.grid(column=3, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        self.clearInputTextButton = ttk.Button(
            buttonFrame, text="Clear Input Tex", command=self.clearInputTextClicked
        )
        self.clearInputTextButton.grid(column=4, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))

        infoTextFrame = ttk.Frame(mainframe, padding="3 3 12 12")
        infoTextFrame.grid(column=1, row=1, sticky=(tk.N, tk.S, tk.W, tk.E))
        infoTextFrame.columnconfigure(0, weight=3)
        infoTextFrame.rowconfigure(0, weight=3)
        infoTextLabel = ttk.Label(infoTextFrame, text="Info")
        infoTextLabel.grid(column=0, row=0, sticky=(tk.N, tk.S, tk.W, tk.E))
        self.infoText = tk.Text(infoTextFrame)
        self.infoText.grid(column=0, row=1, sticky=(tk.S, tk.E))

    def executeOnText(self, func):
        self.resultText.config(state="normal")
        func()
        self.resultText.config(state="disabled")

    def executeOnInfoText(self, func):
        self.infoText.config(state="normal")
        func()
        self.infoText.config(state="disabled")

    def dictionaryButtonClicked(self):
        filetypes = (("text files", "*.txt"), ("All files", "*.*"))
        filename = fd.askopenfilename(
            title="Open a file", initialdir="./", filetypes=filetypes
        )
        dictionaryList = fileManagement.readFile(filename)
        self.t = DFA.Dfa()
        for key in dictionaryList:
            self.t.initializeDfa(key)
        # Prompt user for success
        self.executeOnInfoText(lambda: self.infoText.delete(1.0, tk.END))
        self.executeOnInfoText(
            lambda: self.infoText.insert(tk.END, "Dictionary loaded")
        )
        # Check if result is empty
        self.rerunButtonClicked()

    def rerunButtonClicked(self):
        text = self.sampleText.get(1.0, tk.END)
        if text != "":
            self.executeOnText(lambda: self.resultText.delete(1.0, tk.END))
            self.DfaStart(text)

    def clearInputTextClicked(self):
        self.sampleText.delete(1.0, tk.END)
        self.executeOnText(lambda: self.resultText.delete(1.0, tk.END))

    def clearLogButtonClicked(self):
        self.executeOnInfoText(lambda: self.infoText.delete(1.0, tk.END))
```

```
    def fileButtonClicked(self):

        filetypes = (("text files", "*.txt"), ("All files", "*.*"))

        filename = fd.askopenfilename(
            title="Open a file", initialdir="./", filetypes=filetypes
        )
        if filename:

            # read file
            with open(filename, "r") as f:
                # read lines
                lines = f.readlines()
                # merge list to strings
                text = "".join(lines)
                self.clearInputTextClicked()
                self.sampleText.insert(tk.END, text)
                self.DfaStart(text)

    def highlightText(self, text):
        self.resultText.tag_configure("highlight", background="yellow")
        highlightSwitch = False
        patternFound = []
        for char in text:
            if char == "<":
                highlightSwitch = True
                patternFound.append("")
            elif char == ">":
                highlightSwitch = False
            if char not in "<>":
                if highlightSwitch:
                    self.executeOnText(
                        lambda: self.resultText.insert(tk.END, char, ("highlight"))
                    )
                    patternFound[-1] += char
                else:
                    self.executeOnText(lambda: self.resultText.insert(tk.END, char))
        self.patternFoundCounter(patternFound)

    def patternFoundCounter(self, patternArray):
        for pattern in patternArray:
            if pattern in self.patternFound:
                self.patternFound[pattern] += 1
            else:
                self.patternFound[pattern] = 1

    def patternFoundPrint(self):

        if self.patternFound:
            self.infoText.tag_configure("important", background="#90ee90")
            self.executeOnInfoText(
                lambda: self.infoText.insert(tk.END, "Pattern Found:\n", ("important"))
            )
            self.infoText.tag_configure("patternFound", background="#90ee90")
            for key, value in self.patternFound.items():
                self.executeOnInfoText(
                    lambda: self.infoText.insert(
                        tk.END, key + " : " + str(value) + "\n", ("patternFound")
                    )
                )
        else:
            self.infoText.tag_configure("important", background="red")
            self.executeOnInfoText(
                lambda: self.infoText.insert(
                    tk.END, "Pattern Not Found\n", ("important")
                )
            )

    def DfaStart(self, text):
        self.patternFound = {}
        stack = self.DfaInput(text)
        if stack[-1] != "" or len(stack) > 1:
            self.highlightText("".join(stack))
        self.patternFoundPrint()

    def DfaInput(self, text):
        self.t.resetDfaState()

        stack = [""]
        for char in text:
            # char = text[pIndex]
            stateOfDfa = self.t.inputNextChar(char)
            stack[-1] += char
            if stateOfDfa is None and char in self.seperatorChar:
                if len(stack) > 1:
                    if stack[1][0] not in self.seperatorChar and stack[0][-1] != " ":
                        stack[0] = stack[0].strip("<>") + stack[1]
                        stack.pop(1)
                self.highlightText(stack[0])
                stack.pop(0)
                stack = self.DfaInput("".join(stack))

            elif stateOfDfa is True:
                if len(stack) == 1:
                    stack = ["<" + stack[-1] + ">", ""]
                else:
                    stack[0] = stack[0].strip("<>")
                    result = "<" + "".join(stack) + ">"
                    stack = [result, ""]
            elif stateOfDfa is False:
                if char in self.seperatorChar:
                    stack.append("")

        return stack


root = tk.Tk()
UI(root)
root.mainloop()
```

## 4.3 Dictionary Used

Alor Gajah
Alor Setar
Asajaya
Bachok
Bagan Datuk
Bakong
Balik Pulau
Baling
Bandar Baharu
Bandar Baru Bangi
Bandar Baru Selayang
Bandar Bera
Bandar Permaisuri
Bandar Seri Jempol
Batang Padang
Batu Gajah
Batu Pahat
Bau
Beaufort
Belaga
Belawai
Beluran
Beluru
Bentong
Bera
Besut
Betong
Bintangor
Bintulu
Bukit Mabong
Bukit Mabong[6]
Bukit Mertajam
Cameron Highlands
Central Seberang Perai
Chukai
Dalat
Daro
Dungun
Federal Territory (Malaysia)
George Town
Gerik
Gombak
Gua Musang
Hilir Perak
Hulu Langat
Hulu Perak
Hulu Selangor
Hulu Terengganu
Interior
Jasin
Jelebu
Jeli
Jempol
Jerantut
Jitra
Johor
Johor Bahru
Julau
Kabong
Kalabakan
Kampar
Kampung Raja
Kangar
Kanowit
Kapit
Kedah
Kelantan
Kemaman
Keningau
Kepala Batas
Kerian
Kinabatangan
Kinta
Klang
Kluang
Kota Belud
Kota Bharu
Kota Kinabalu
Kota Marudu
Kota Setar
Kota Tinggi
Kuah
Kuala Berang
Kuala Dungun
Kuala Kangsar
Kuala Klawang
Kuala Krai
Kuala Kubu Bharu
Kuala Langat
Kuala Lipis
Kuala Lumpur
Kuala Muda
Kuala Nerang
Kuala Nerus
Kuala Penyu
Kuala Pilah
Kuala Rompin
Kuala Selangor
Kuala Terengganu
Kuantan
Kubang Pasu
Kuching
Kudat
Kulai
Kulim
Kunak
Labuan
Lahad Datu
Langkawi
Larut, Matang and Selama
Lawas
Limbang
Lipis
Long Lama
Lubok Antu
Lundu
Machang
Malacca
Malacca City
Malaysia
Manjung
Maran
Marang
Marudi
Matu
Melaka Tengah
Meradong
Mersing
Miri

Muallim
Muar
Mukah
Nabawan
Negeri Sembilan
North Seberang Perai
Northeast Penang Island
Padang Terap
Pahang
Pakan
Papar
Parit Buntar
Pasir Mas
Pasir Puteh
Pekan
Penampang
Penang
Pendang
Perak
Perak Tengah
Perlis
Petaling
Pitas
Pokok Sena
Pontian
Pontian Kechil
Port Dickson
Pusa
Putatan
Putrajaya
Ranau
Raub
Rembau
Rompin
Sabah
Sabak
Sabak Bernam
Salak Tinggi
Samarahan
Sandakan
Saratok
Sarawak
Sarikei
Sebauh
Segamat
Selangau
Selangor
Semporna
Sepang
Serdang
Seremban
Seri Iskandar
Seri Manjung
Serian
Setiu
Sibu
Sik
Simanggang
Simunjan
Sipitang
Song
South Seberang Perai
Southwest Penang Island
Sri Aman
Subang
Subis
Sungai Jawi
Sungai Petani
Taiping
Tambunan
Tampin
Tanah Merah
Tanah Rata
Tangkak
Tanjung Malim
Tanjung Manis
Tapah
Tatau
Tawau
Tebedu
Telang Usan
Teluk Datok
Teluk Intan
Telupid
Temerloh
Tenom
Terengganu
Tongod
Tuaran
Tumpat
Victoria
West Coast
Yan
Yan Besar
Afghanistan
Albania
Algeria
Andorra
Angola
Antigua and Barbuda
Argentina
Armenia
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bhutan
Bolivia
Bosnia and Herzegovina
Botswana
Brazil
Brunei
Bulgaria
Burkina Faso
Burundi
Côte d'Ivoire
Cabo Verde
Cambodia
Cameroon
Canada
Central African Republic
Chad
Chile
China
Colombia

Comoros
Congo
Congo-Brazzaville
Costa Rica
Croatia
Cuba
Cyprus
Czechia
Czech Republic
Democratic Republic of the Congo
Denmark
Djibouti
Dominica
Dominican Republic
Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Eswatini
Ethiopia
Fiji
Finland
France
Gabon
Gambia
Georgia
Germany
Ghana
Greece
Grenada
Guatemala
Guinea
Guinea-Bissau
Guyana
Haiti
Holy See
Honduras
Hungary
Iceland
India
Indonesia
Iran
Iraq
Ireland
Israel
Italy
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Kiribati
Kuwait
Kyrgyzstan
Laos
Latvia
Lebanon
Lesotho
Liberia
Libya
Liechtenstein
Lithuania
Luxembourg
Madagascar
Malawi
Maldives
Mali
Malta
Marshall Islands
Mauritania
Mauritius
Mexico
Micronesia
Moldova
Monaco
Mongolia
Montenegro
Morocco
Mozambique
Myanmar
Namibia
Nauru
Nepal
Netherlands
New Zealand
Nicaragua
Niger
Nigeria
North Korea
North Macedonia
Norway
Oman
Pakistan
Palau
Palestine State
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Poland
Portugal
Qatar
Romania
Russia
Rwanda
Saint Kitts and Nevis
Saint Lucia
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Korea
South Sudan
Spain
Sri Lanka
Sudan

Suriname
Sweden
Switzerland
Syria
Tajikistan
Tanzania
Thailand
Timor-Leste
Togo
Tonga
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
United States of America
Uruguay
Uzbekistan
Vanuatu
Venezuela
Vietnam
Yemen
Zambia
Zimbabwe
Africa
Antarctica
Asia
Europe
North America
South America
Afro-Eurasia
Americas
Eurasia
Oceania
Amazonia
Arctica
Asiamerica
Atlantica
Avalonia
Baltica
Cimmeria
Congo Craton
Euramerica
Kalaharia
Kazakhstania
Laramidia
Laurentia
North China
Siberia
South China
East Antarctica
Pampia
Cuyania
Chilenia
Southeast Asia