­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­



Контрольна робота

з курсу «Дискретні моделі в САПР»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 11

Виконав студент гр. КНз-41

Чалий Михайло

­­

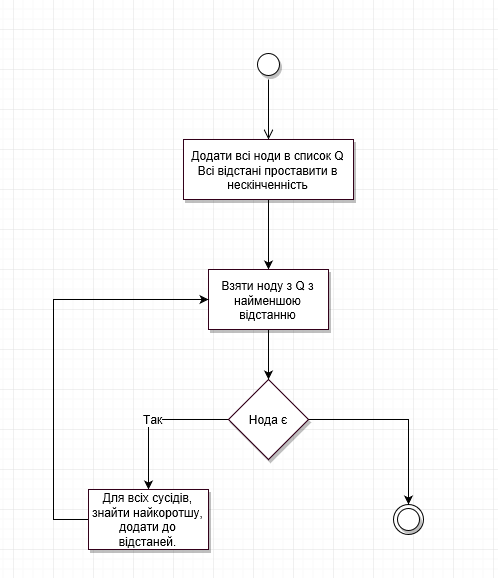
Львів 2016

# Алгоритм Дейкстри

**Алгоритм Дейкстри** — алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин. Класичний алгоритм Дейкстри працює тільки для графів без циклів від'ємної довжини.

## Блок схема алгоритму

Дуже спрощенна блок схема, на виході матриця з відстаней, де відсортувавшм можна зайти найкоротший шлях.



## Код алгоритму

F# - <https://github.com/chaliy/studies-octo-adventure/blob/master/lp/c1/dm/dijkstra.fsx>

open System

open System.Collections.Generic

let graph = [

( ('A', 'B'), 4.0)

( ('A', 'H'), 3.0)

( ('A', 'G'), 15.0)

( ('B', 'C'), 5.0)

( ('B', 'D'), 7.0)

( ('B', 'H'), 5.0)

( ('C', 'D'), 3.0)

( ('D', 'F'), 6.0)

( ('D', 'E'), 10.0)

( ('E', 'G'), 13.0)

( ('F', 'G'), 1.0)

( ('F', 'H'), 2.0)

( ('G', 'H'), 4.0)

]

let distance g e =

g

|> List.tryFind (fun (xe, \_) -> e = xe || e = (snd xe, fst xe))

|> function | Some(xe, xd) -> xd

| None -> Double.PositiveInfinity

let vertices g =

g

|> Seq.collect (fun (xe, \_) -> [fst xe; snd xe])

|> Seq.distinct

let edges g =

g

|> List.map (fun (xe, \_) -> xe)

let neighbors g x =

g

|> List.choose(fun (xe, \_) ->

match xe with

| (xs, xt) when xs = x -> Some(xt)

| (xs, xt) when xt = x -> Some(xs)

| \_ -> None)

let dijkstra s =

let unprocessed = List<char>(vertices graph)

let distances = Dictionary<char, float> (dict (seq {

for v in (vertices graph) -> (v, Double.PositiveInfinity)

}))

let previous = Dictionary<char, char option> (dict (seq {

for v in (vertices graph) -> (v, None)

}))

distances.[s] <- 0.0 // Start

while (unprocessed.Count <> 0) do

// Vertex with smallest distance

let cv = unprocessed

|> Seq.map(fun v -> (v, distances.[v]))

|> Seq.sortBy(fun (v,d) -> d)

|> Seq.map(fun (v,d) -> v)

|> Seq.head

let cd = distances.[cv] // Current distance

printfn "Vertex %A with distance %A" cv cd

let nn = neighbors graph cv

for n in nn do

let nd = cd + (distance graph (cv,n)) // New distance

if (nd < distances.[n]) then

printfn "\tNeighbor %A new distance %A, previous %A" n nd (distances.[n])

distances.[n] <- nd

previous.[n] <- Some(cv)

printfn "\tRemove vertex %A" cv

unprocessed.Remove(cv) |> ignore

(distances, previous)

let getPath (pp:IDictionary<char, char option>) tv =

let rec nextItem ii =

match pp.[ii |> List.head] with

| Some (nv) ->

nextItem (ii |> List.append([nv]))

| None -> ii

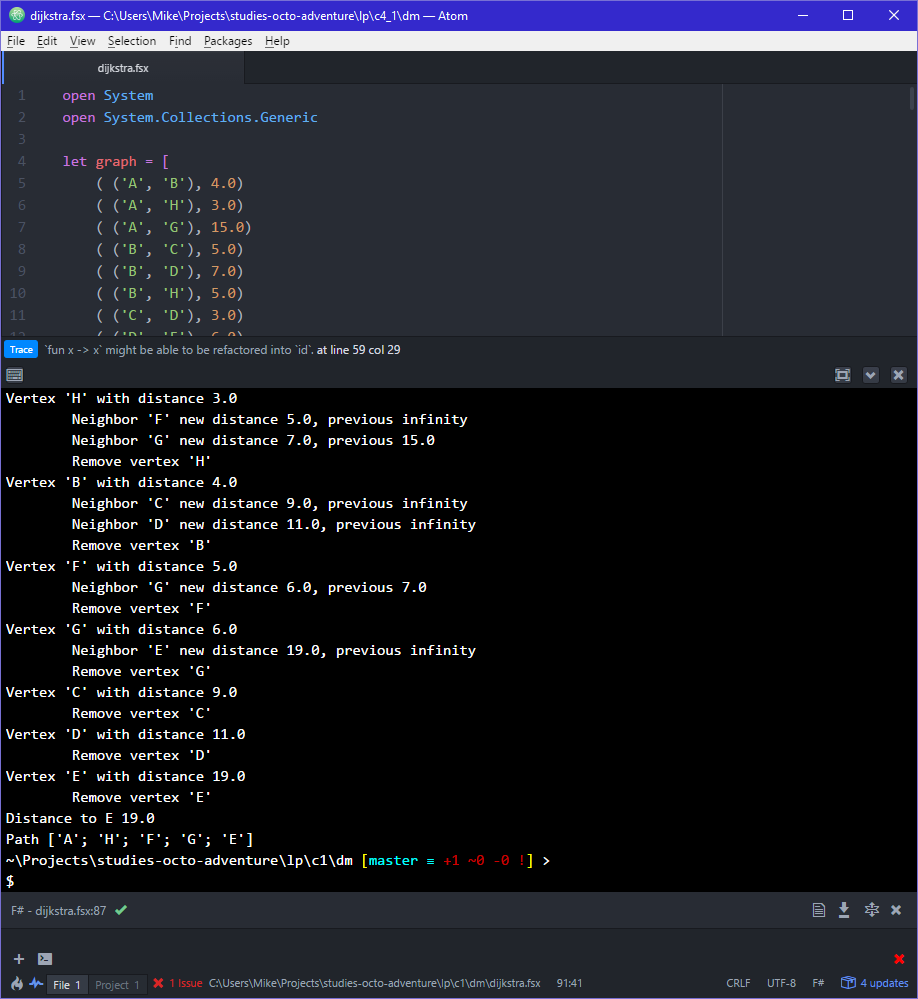
nextItem [tv]

let distances, previous = dijkstra 'A'

printfn "Distance to E %A" distances.['E']

printfn "Path %A" (getPath previous 'E')

## Результати роботи програми



## Аналіз результатів та висновки

Алгоритм Дейкстри легкий в реалізаціі і зрозумілий алгоритм для пошуку найкоротшого шляху.