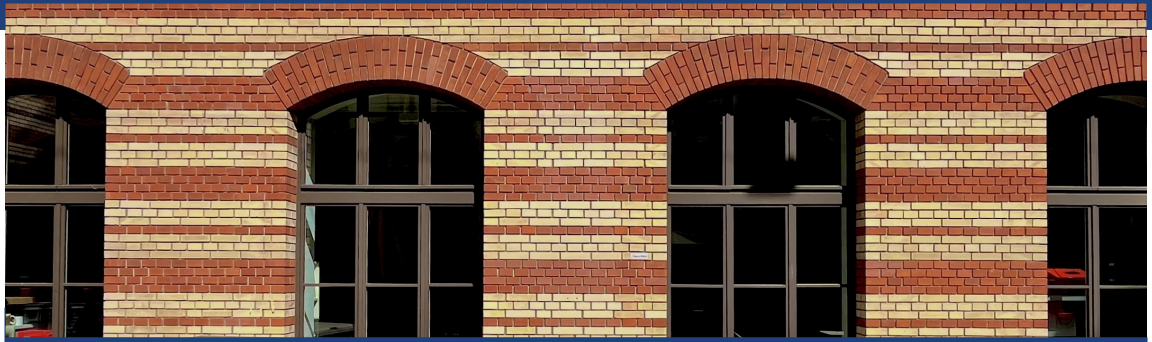


Datastructures and Algorithms

DFS, BFS, Dijkstra, “Real World” Shortest Path Problems

Adel Gavranović — ETH Zürich — 2025

Overview

Learning Objectives
Repetition Theory Graphs
- Graphs: DFS and BFS
- Appendix: Real World Shortest Path Problems
- Shortest Paths Quiz

Code-Expert Exercise
Repetition Theory Dijkstra
- Dijkstra

Past Exam Questions

n.ethz.ch/~agavranovic

Material
Webpage
Mail

1. Follow-up

Follow-up from last session

- Wasn’t able to cover any of the questions from last time, due to an already very busy TA meeting and too little time overall. Sorry.

2. Learning Objectives

Objectives

- ☐ Understand and be able to manually execute all of the below
  - ☐ Breadth-First Search (BFS)
  - ☐ Depth-First Search (DFS)
  - ☐ Dijkstra's Shortest Path Algorithm

3. Summary

Getting on the same page

- What did you cover in the lectures?

4. Repetition Theory Graphs

4. Repetition Theory Graphs

4.1. Graphs: DFS and BFS

Quiz: Runtimes of simple Operations

Operation	Matrix	List
$(v, u) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Find neighbours/successors of $v \in V$	$\Theta(n)$	$\Theta(\deg^+ v)$
find $v \in V$ without neighbour/successor	$\Theta(n^2)$	$\Theta(n)$
find all edges $e \in E$	$\Theta(n^2)$	$\Theta(n + m)$
Insert edge	$\Theta(1)$	$\Theta(1)$
Delete edge	$\Theta(1)$	$\Theta(\deg^+ v)$

Quiz #1

Question

Which graph representation, adjacency matrix or adjacency list, is more suitable for representing a graph with a high number of edges compared to vertices?

Answer

For very dense graphs, when the number of edges is close to n^2, an adjacency matrix is more suitable; the space complexity of an adjacency matrix is $\Theta(n^2)$, which is independent of the number of edges.

Quiz #2

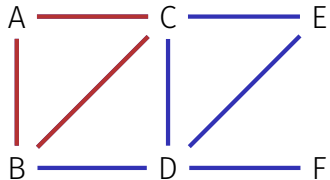
Question

When would it be more appropriate to use an adjacency matrix representation rather than an adjacency list representation? Provide another example scenario.

Answer

For example, in a scenario where you need to frequently check the presence of an edge or update edges between vertices, an adjacency matrix would be more suitable due to its $\Theta(1)$ edge lookup, insertion, and deletion time complexity.

Quiz #3



We want to count the number of triangles (cycles with 3 nodes and edges) in a graph G.

In what time can we do this with an adjacency matrix? How about an adjacency list?

Quiz #3 Solution

Adjacency matrix: $\Theta(n^2 + m \cdot n)$

Naively: $\Theta(n^3)$: check for each of the $\binom{n}{3}$ combinations of 3 nodes whether the corresponding 3 edges are there.

Efficient: for every edge and every additional node, check whether the two additional edges are there.

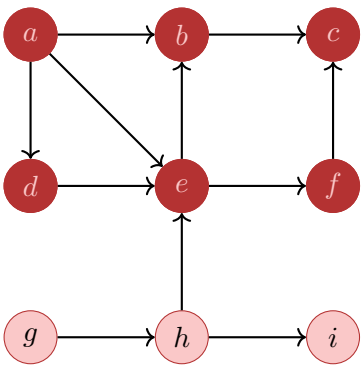
Adjacency list: $\Theta(n \cdot m)$ with $\Theta(n)$ additional memory or $\Theta(n^2 \cdot m)$

Naively: $\Theta(n^2 \cdot m)$: for every edge $e = \{u, v\}$ and every potential third node w, we go through the two lists $A[u]$ and $A[v]$ to see whether w is a neighbor of both.

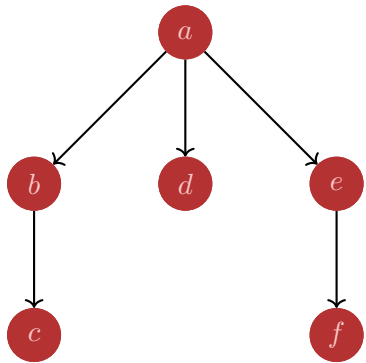
Efficient: go through $A[u]$, store the neighbors in a bitmap of length n, then for each neighbor v construct the bitmap of v and compare. So we are effectively comparing $\Theta(m)$ bitmaps of length n.

Breadth-First-Search BFS

BFS starting from a:



BFS-Tree: Distances and Parents



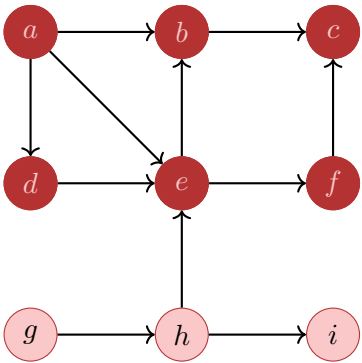
Distance 0

Distance 1

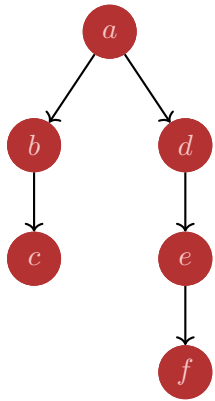
Distance 2

Depth-First-Search DFS

DFS starting from a:



DFS-Tree: Distances and Parents



Distance 0

Distance 1

Distance 2

Distance 3

Detect Cycles

Cycle Detection

How can you detect cycles in a graph? Explain the process for undirected and directed graphs.

Detect Cycles

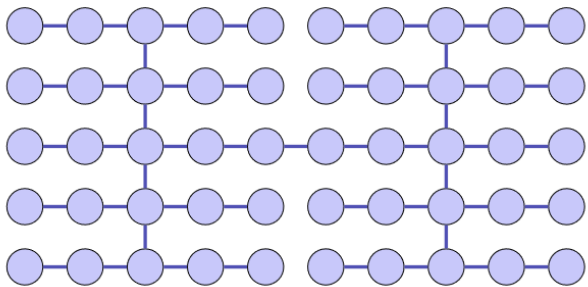
DFS Cycle Detection

- Start DFS traversal from an arbitrary node
- undirected: If a visited node is encountered again (excluding the immediate parent), a cycle exists.
- directed: If an edge to a grey node is found, a directed cycle exists.

Exam Question Example

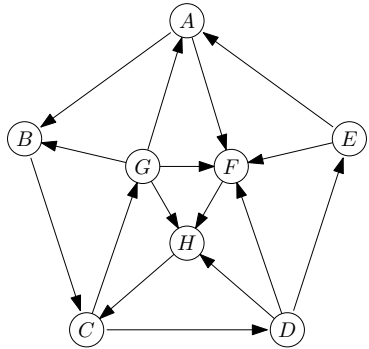
Was ist die maximale Rekursionstiefe der (rekursiv implementierten) Funktion DFS angewendet auf folgenden Graphen. Der erste Aufruf wird mitgezählt.

What is the maximum recursion depth of the (recursively implemented) function DFS in the following graph. The first call is counted.



Answer: 14

Depth-first-search and Breadth-first-search



Starting at A

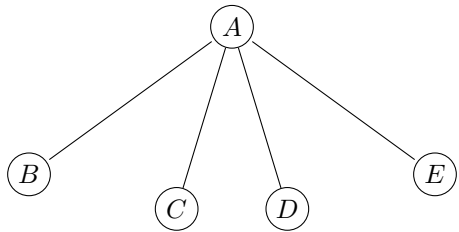
DFS: A, B, C, D, E, F, H, G

BFS: A, B, F, C, H, D, G, E

There is no starting vertex where the DFS ordering equals the BFS ordering.

Depth-first-search and Breadth-first-search

Star: DFS ordering equals BFS ordering

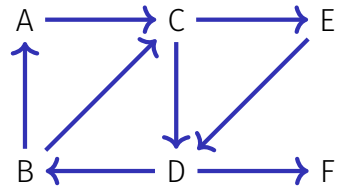


Starting at A
DFS: A, B, C, D, E
BFS: A, B, C, D, E

Starting at C
DFS: C, A, B, D, E
BFS: C, A, B, D, E

Quiz (from an old exam): BFS/DFS

The following graph is visited with a breadth-first search and a depth-first search algorithm starting at node A. If there are several possibilities for a visiting order of the neighbours, the alphabetical order is chosen. Provide both visiting orders.



Breadth First Search: ?

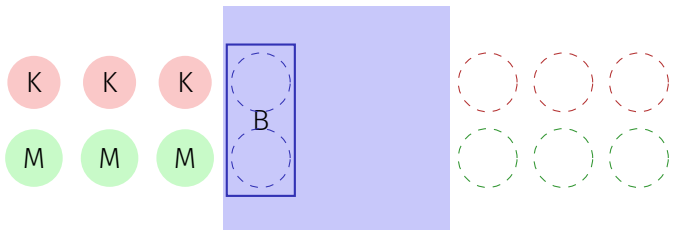
Depth First Search: ?

4.2. Appendix: Real World Shortest Path Problems

Modeling

River Crossing (Missionaries and Cannibals)

Problem: Three cannibals and three missionaries are standing at a river bank. The available boat can carry two people. At no time may at any place (banks or boat) be more cannibals than missionaries. How can the missionaries and cannibals cross the river as fast as possible? [1]



[1] There are slight variations of this problem. It is equivalent to the jealous husbands problem.

Problem as Graph

Enumerate permitted configurations as nodes and connect them with an edge, when a crossing is allowed. The problem then becomes a shortest path problem.

Example

	left	right
missionaries	3	0
cannibals	3	0
boat	x	

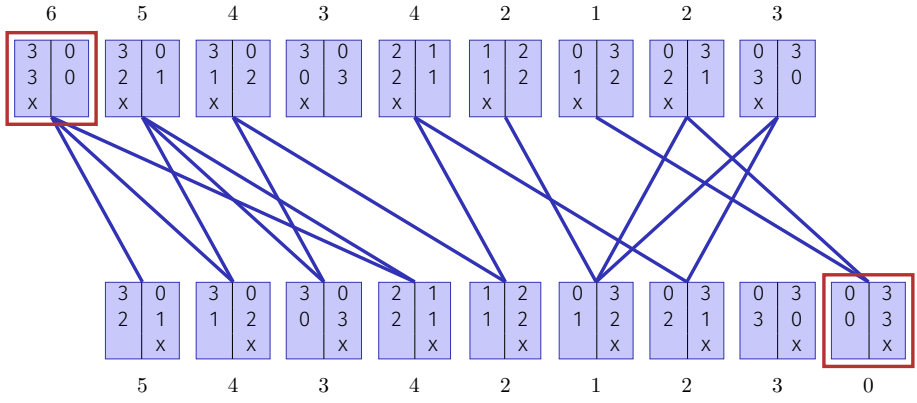
6 People on the left bank

Possible crossing

	left	right
missionaries	2	1
cannibals	2	1
boat		x

4 People on the left bank

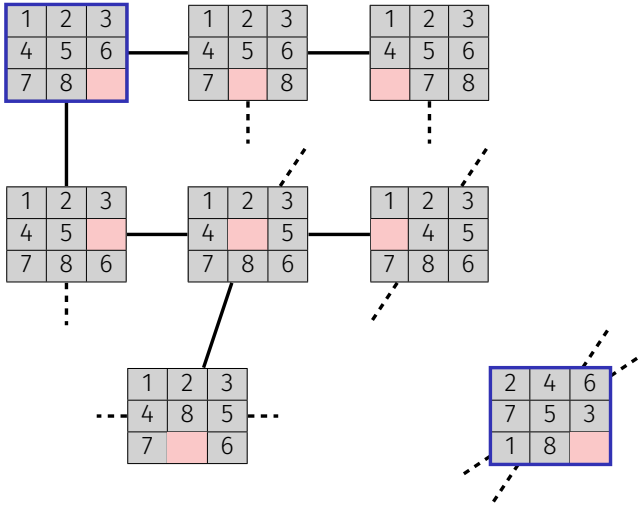
The whole problem as a graph



Real-World Example: Mystic Square

Fastest solution for

Problem as Graph



Shortest Paths

Given: $G = (V, E, c)$, $c : E \to \mathbb{R}$, $s, t \in V$.

Path: $s \overset{p}{\rightsquigarrow} t : \langle s = v_1, v_2, \ldots, v_{k+1} = t \rangle$, $(v_i, v_{i+1}) \in E \; (1 \leq i \leq k)$
Weight: $c(p) := \sum_{i=1}^{k} c((v_i, v_{i+1}))$.

Weight of a shortest path from u to v:

$$\delta(u, v) = \begin{cases} \infty & \text{no path from } u \text{ to } v, \\ \min\{c(p) : u \overset{p}{\rightsquigarrow} v\} & \text{otherwise.} \end{cases}$$

Quiz 1

You are given a weighted directed graph $G = (V, E)$ as well as two designated nodes s, t.

Let $p_1 = \langle s, v_1, \ldots, v_k, u \rangle$ and $p_2 = \langle u, w_1, \ldots, w_l, t \rangle$ be two shortest paths, from s to u and from u to t, respectively.

Is the following statement correct?

The concatenation $\langle s, v_1, \ldots, v_k, u, w_1, \ldots, w_l, t \rangle$ is a shortest path from s to t.

Quiz 2

Given any shortest path $p = \langle s, v_1, \ldots, v_\ell, t \rangle$ from s to t.

Is the following statement correct (v_k is on the path p)?

The two paths $\langle s, v_1, \ldots, v_k \rangle$ and $\langle v_k, \ldots, v_\ell, t \rangle$ must be shortest paths from s to v_k, and from v_k to t, respectively.

5. Code-Expert Exercise

Code-Example 1

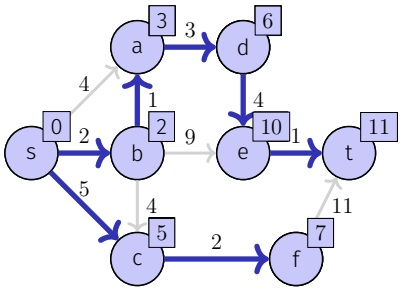
’BFS on a Tree’ on Code-Expert

6. Repetition Theory Dijkstra

6. Repetition Theory Dijkstra

6.1. Dijkstra

Example



Known shortest paths from s:

$s \rightsquigarrow s$: 0 $\quad$ $s \rightsquigarrow d$: 6

$s \rightsquigarrow b$: 2 $\quad$ $s \rightsquigarrow f$: 7

$s \rightsquigarrow a$: 3 $\quad$ $s \rightsquigarrow e$: 10

$s \rightsquigarrow c$: 5 $\quad$ $s \rightsquigarrow t$: 11

$\mathbf{S} = \{s, b, a, c, d, f, e, t\}$

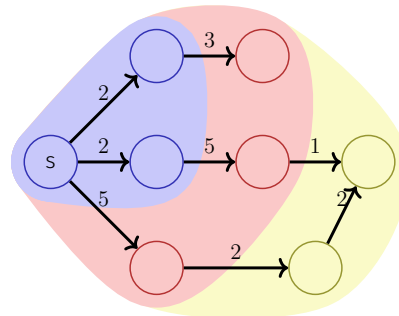
$\mathbf{U} = \{\}$

$\mathbf{R} = \{\}$

Dijkstra (positive edge weights)

Set V of nodes is partitioned into

- the set S of nodes for which a shortest path from s is already known,
- the set $U = \bigcup_{v \in S} N^+(v) \setminus S$ of nodes where a shortest path is not yet known but that are accessible directly from S,
- the set $R = V \setminus (S \cup U)$ of nodes that have not yet been considered.



Algorithm Dijkstra(G, s)

Input: Positively weighted Graph $G = (V, E, c)$, starting point $s \in V$,
Output: Minimal weights d of the shortest paths and corresponding predecessor node for each node.

foreach $u \in V$ **do**
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow null$
$d_s[s] \leftarrow 0$; $U \leftarrow \{s\}$
while $U \neq \emptyset$ **do**
    $u \leftarrow$ ExtractMin(U)
    foreach $v \in N^+(u)$ **do**
        if $d_s[u] + c(u, v) < d_s[v]$ **then**
            $d_s[v] \leftarrow d_s[u] + c(u, v)$
            $\pi_s[v] \leftarrow u$
            $U \leftarrow U \cup \{v\}$

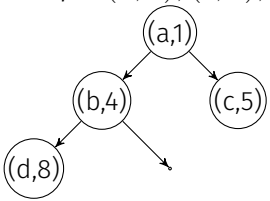
Implementation: Data Structure for U?

Relax for Dijkstra:

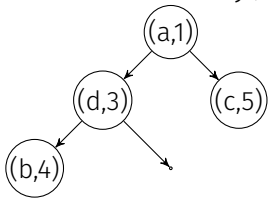
if $d_s[u] + c(u, v) < d_s[v]$ **then**
    $d_s[v] \leftarrow d_s[u] + c(u, v)$
    $\pi_s[v] \leftarrow u$
    if $v \notin U$ **then**
        Add(U, v) // Insertion of a new $(v, d(v))$ in the heap of U
    else
        DecreaseKey(U, v) // Update of a $(v, d(v))$ in the heap of U

DecreaseKey ?

Heap ($(a,1),(b,4),(c,5),(d,8)$) =



after DecreaseKey($d,3$):

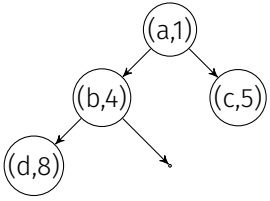


2 problems:

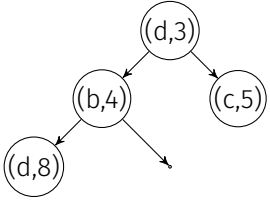
- Position of d unknown at first. Search: $\Theta(n)$
- Positions of the nodes can change during DecreaseKey

Lazy Deletion !

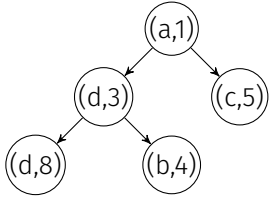
Heap ($(a,1),(b,4),(c,5),(d,8)$) =



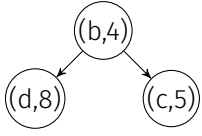
ExtractMin() $\rightarrow (a,1)$



Insert$(d,3)$:



ExtractMin() $\rightarrow (d,3)$



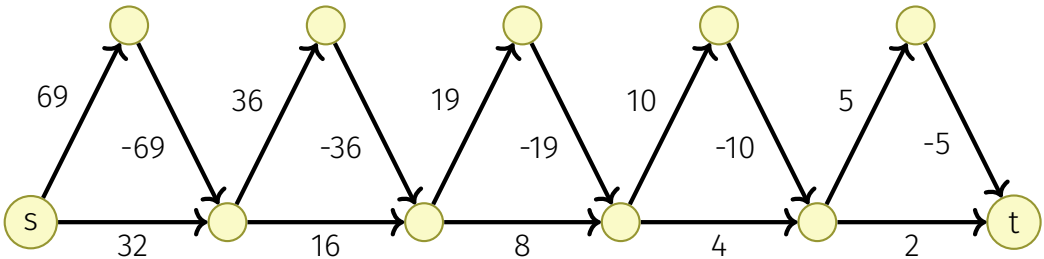
Later ExtractMin() $\rightarrow (d,8)$ must be ignored

Runtime Dijkstra

$n := |V|, m := |E|$

- $n\times$ ExtractMin: $\mathcal{O}(n \log n)$ ($m\times$ ExtractMin with Lazy Deletion)
- $m\times$ Insert or DecreaseKey: $\mathcal{O}(m \log n)$
- $1\times$ Init: $\mathcal{O}(n)$
- Overall: $\mathcal{O}((n + m) \log n)$. For connected graphs: $\mathcal{O}(m \log n)$.

Quiz: An Interesting Graph



Does Dijkstra work?

7. Past Exam Questions

Past Exam 2022: Task 1a)

In einem gewichteten Graphen mit negativen Gewichten, aber ohne negative Zyklen, kann der Dijkstra-Algorithmus verwendet werden, um kürzeste Pfade in polynomieller Zeit zu finden. / *In a weighted graph with negative-weight edges but no negative- weight cycles, Dijkstra's algorithm can be used to find shortest paths in polynomial time.*

◯Wahr / *True*

◯Falsch / *False*

Past Exam 2022: Task 1a) – Solution

In einem gewichteten Graphen mit negativen Gewichten, aber ohne negative Zyklen, kann der Dijkstra-Algorithmus verwendet werden, um kürzeste Pfade in polynomieller Zeit zu finden. / *In a weighted graph with negative-weight edges but no negative- weight cycles, Dijkstra's algorithm can be used to find shortest paths in polynomial time.*

◯Wahr / *True*

√Falsch / *False*

8. Outro

General Questions?

See you next time!

Have a nice week!