ETH zürich

Exercise Session 09 – Graph Algorithms

Data Structures and Algorithms

These slides are based on those of the lecture, but were adapted and extended by the teaching assistant Adel Gavranović

Today's Schedule

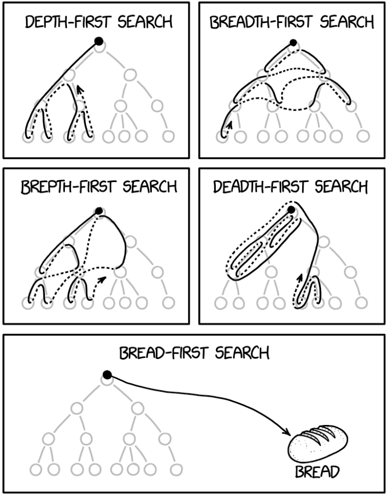
Intro
Feedback for **code** expert
Learning Objectives
Repetition Theory
- Graphs: DFS and BFS
- Topological Sorting
- Dijkstra

Code-Expert Exercise
Red-Black Trees (again)
Old Exam Question
Outro

`n.ethz.ch/~agavranovic`

- Exercise Session Material
- Adel's Webpage
- Mail to Adel

Comic of the Week



▸ xkcd

1. Intro

Intro

- Welcome back!

2. Feedback for **code** expert

General things regarding **code** expert

- The subtask pertaining to Red-Black trees in the exercise "Trees" went pretty bad, so we're going over it again today (if time allows)
  - Some gave 2-3 Trees instead of Red-Black trees (which is impressive, but *not* what was asked for)
  - Some trees were simply wrong
  - What went wrong? How can we improve?
- The current Master Solution for this exercise is useless (imho) and I'm working on a *very* detailed one that is going to be available "soon"

Task "Binary Search Tree"

- If you didn't get 100% for this exercise: **try again**
- This is a classic coding exercise

Questions regarding **code** expert from your side?

3. Learning Objectives

Learning Objectives

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

4. Repetition Theory

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 a graph with a high number of edges compared to vertices, 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

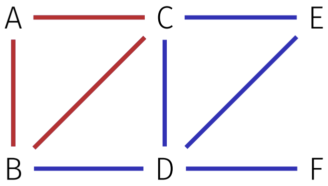
Question

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

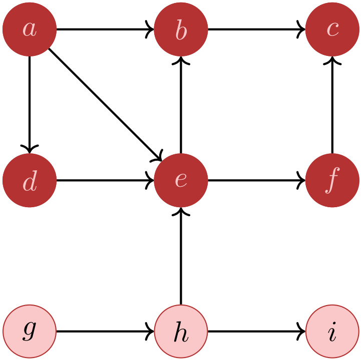
Adjacency list: $\Theta(n \cdot m)$ with $\Theta(n)$ additional memory or $\Theta(n^2 \cdot m)$

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.

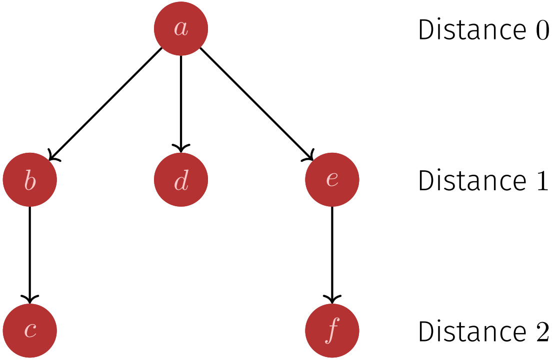
Efficient: go through $A[u]$, store the neighbors in a bitmap of length n, then for each neighbor v construct the bitmap of v and compare. So we are effectively comparing $\Theta(m)$ bitmaps of length n.

Breadth-First-Search BFS

BFS starting from a:

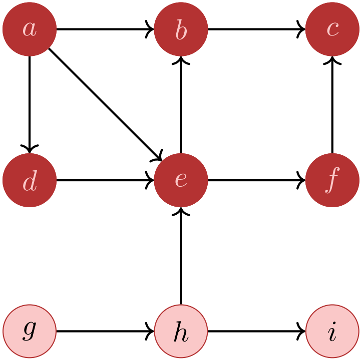


BFS-Tree: Distances and Parents

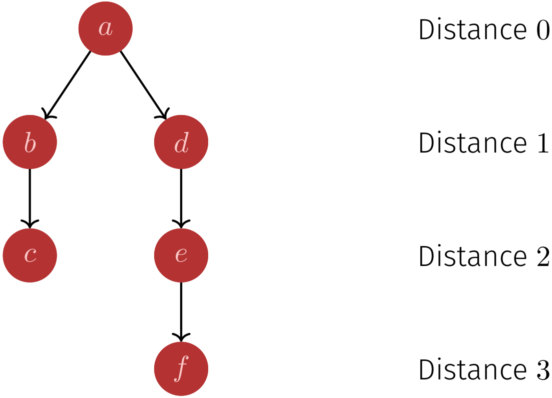


Depth-First-Search DFS

DFS starting from a:



DFS-Tree: Distances and Parents



Detect Cycles

Cycle Detection

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

Detect Cycles

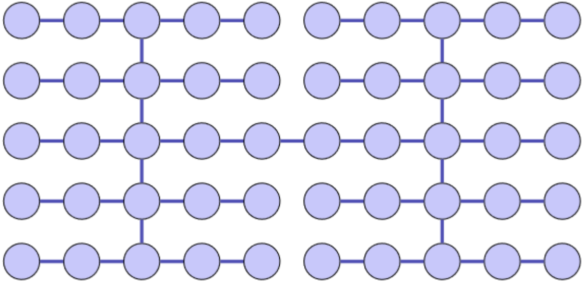
DFS Cycle Detection

- 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.

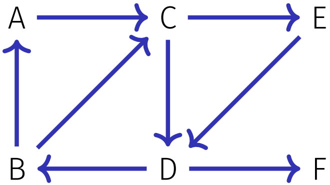
What is the maximum recursion depth of the (recursively implemented) function DFS in the following graph. The first call is counted.



Answer: 14

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.

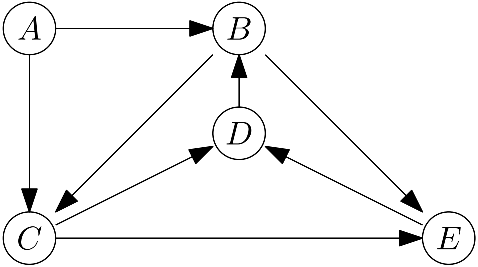


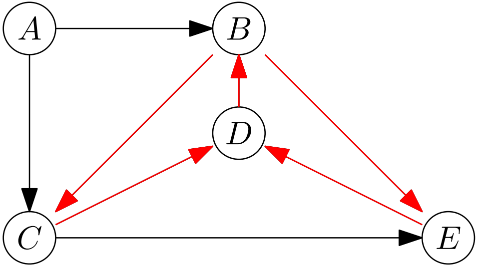
Breadth First Search: A C D E B F

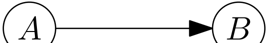
Depth First Search: A C D B F E

4.2 Topological Sorting

Topological Sorting

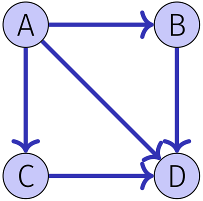






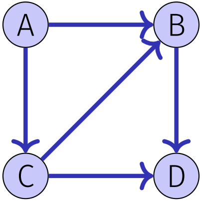
Quiz 1: Topological Sorting

In how many ways can the following directed graphs be topologically sorted each?



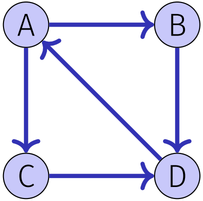
number sortings

2



number sortings

1

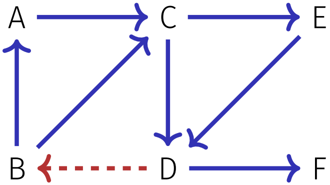


number sortings

0

Quiz 2: Topological Sorting

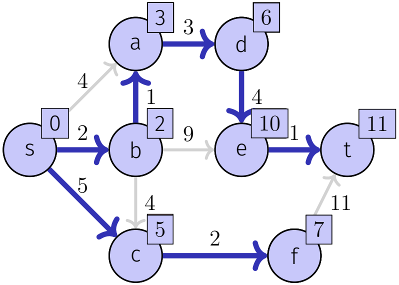
In the following graph, cross out the smallest possible set of edges such that the remaining graph can be topologically sorted. Then provide a sorting.



Sorting: B A C E D F

4.3 Dijkstra

Example



Known shortest paths from s:

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

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

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

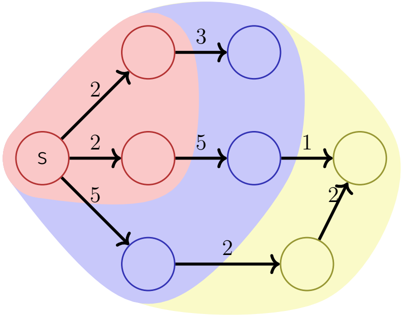
$s \rightsquigarrow c$: 5 | $s \rightsquigarrow t$: 11

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

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**
            $d_s[v] \leftarrow d_s[u] + c(u, v)$
            $\pi_s[v] \leftarrow u$
            $U \leftarrow U \cup \{v\}$

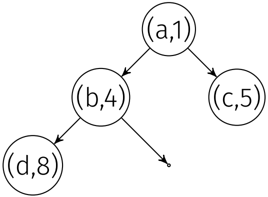
Implementation: Data Structure for U?

Relax for Dijkstra:

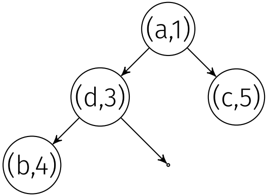
```
if d_s[u] + c(u,v) < d_s[v] then
    d_s[v] ← d_s[u] + c(u,v)
    π_s[v] ← u
    if v ∉ U then
        Add(U, v)              // Insertion of a new (v, d(v)) in the heap of U
    else
        DecreaseKey(U, v)      // Update of a (v, d(v)) in the heap of U
```

DecreaseKey ?

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



after DecreaseKey$(d,3)$:

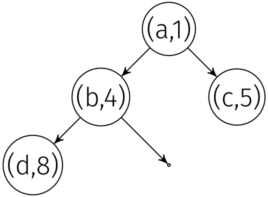


2 problems:

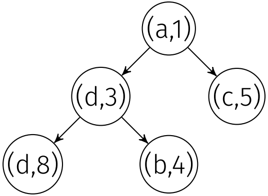
- Position of d unknown at first. Search: $\Theta(n)$
- Positions of the nodes can change during DecreaseKey

Lazy Deletion !

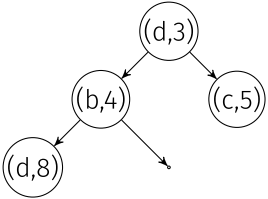
Heap ($(a,1),(b,4),(c,5),(d,8)$) =



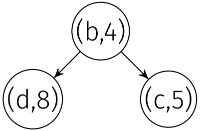
Insert$(d,3)$:



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



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



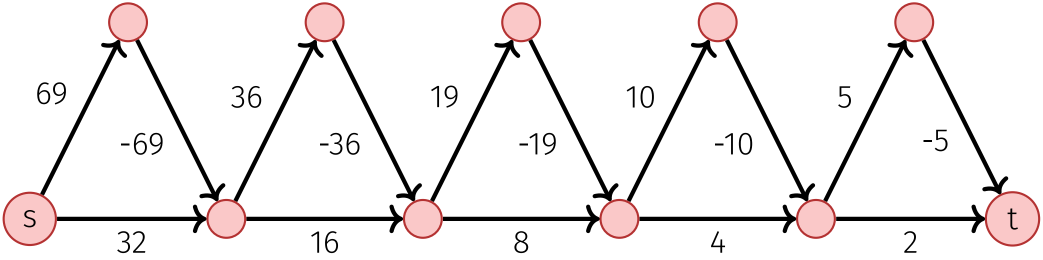
Later ExtractMin() $\rightarrow (d,8)$ must be ignored

Runtime Dijkstra

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

- $n\times$ ExtractMin: $\mathcal{O}(n \log n)$
- $m\times$ 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?

Answer

Dijkstra works also for graphs with negative edge weights (with the modification that nodes can be added to and removed from U repeatedly), if no negative weight cycles are present. But Dijkstra may then exhibit exponential running time!

5. Code-Expert Exercise

Code-Example 1

’BFS on a Tree’ on Code-Expert

6. Red-Black Trees (again)

Insert: $9, 5, 14, 7, 3, 16, 1, 4$ into Red-Black Tree

Insert: $9, 5, 14, 7, 3, 16, 1, 4$ into Red-Black Tree

Insert: $9, 5, 14, 7, 3, 16, 1, 4$ into Red-Black Tree

7. Old Exam Question

Dijkstra Exam Question

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*

Dijkstra Exam Question – 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*

But why? Because the Dijkstra algorithm can have an **exponential runtime** if negative edges are included!

8. Outro

General Questions?

See you next time

Have a nice week!