Datenstrukturen und Algorithmen

Exercise 9

FS 2020

Program of today

1. Feedback of last exercises
2. Recap Theory
3. In-Class Exercise

1. Feedback of last exercises

Levenshtein Distance

```
// D[n,m] = distance between x and y
// D[i,j] = distance between strings x[1..i] and y[1..j]
vector<vector<unsigned>> D(n+1,vector<unsigned>(m+1,0));
for (unsigned j = 0; j <=m; ++j)
  D[0][j] = j;
for (unsigned i = 1; i <= n; ++i){
  D[i][0] = i;
  for (unsigned j = 1; j <=m; ++j){
    unsigned q = D[i-1][j-1] + (x[i-1]!=y[j-1]);
    q = std::min(q,D[i][j-1]+1);
    q = std::min(q,D[i-1][j]+1);
    D[i][j] = q;
  }
}
return D[n][m];
```

Traveling Salesman

see master solution with detailed comments

Huffman Code- Frequencies: Hashmap!

```
std::map<char, int> m;
char x; int n = 0;
while (in.get(x)){
       ++m[x]; ++n;
}
std::cout << "n = " << n << " characters" << std::endl;
```

Huffman Code - Nodes: SharedPointers on a Heap

```
struct comparator {
 bool operator()(const SharedNode a, const SharedNode b) const {
        return a->frequency > b->frequency;
 }
};
...

// build heap
std::priority_queue<SharedNode, std::vector<SharedNode>, comparator>
for (auto y: m){
        q.push(std::make_shared<Node>(y.first, y.second));
}
```

Huffman Code – Tree: SharedPointers in Tree

```
// build code tree
SharedNode left;
while (!q.empty()){
        left = q.top();q.pop();
        if (!q.empty()){
                auto right = q.top();q.pop();
                q.push(std::make_shared<Node>(left, right));
        }
}
```

2. Recap Theory

Quiz: Runtimes of simple Operations

Operation	Matrix	List
Find neighbours/successors of $v \in V$		
find $v \in V$ without neighbour/successor		
$(u, v) \in E$?		
Insert edge		
Delete edge		

Quiz: Runtimes of simple Operations

Operation	Matrix	List
Find neighbours/successors of $v \in V$	$\Theta(n)$	
find $v \in V$ without neighbour/successor		
$(u, v) \in E$?		
Insert edge		
Delete edge		

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

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

Quiz: Runtimes of simple Operations

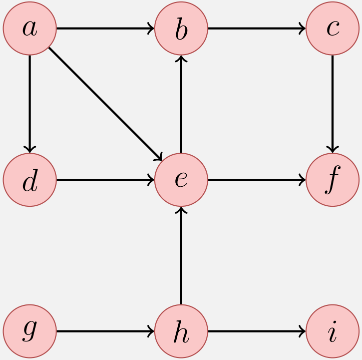
Operation	Matrix	List
Find neighbours/successors of $v \in V$	$\Theta(n)$	$\Theta(\deg^+ v)$
find $v \in V$ without neighbour/successor	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Insert edge	$\Theta(1)$	$\Theta(1)$
Delete edge	$\Theta(1)$	

Quiz: Runtimes of simple Operations

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

Breadth-First-Search BFS

BFS starting from a:

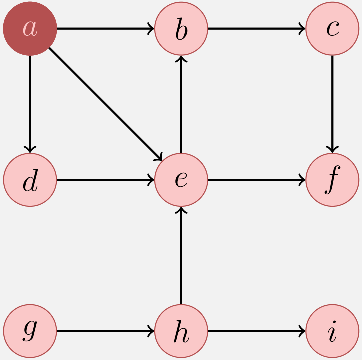


BFS-Tree: Distances and Parents

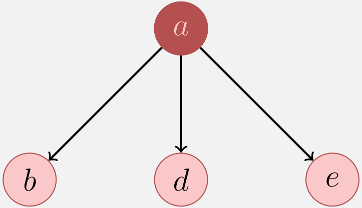
distance 0

Breadth-First-Search BFS

BFS starting from a:



BFS-Tree: Distances and Parents

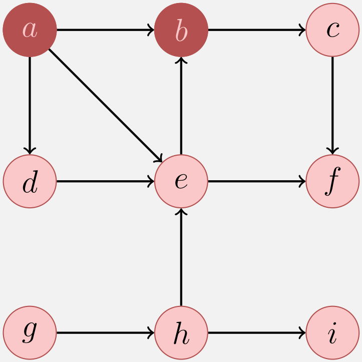


distance 0

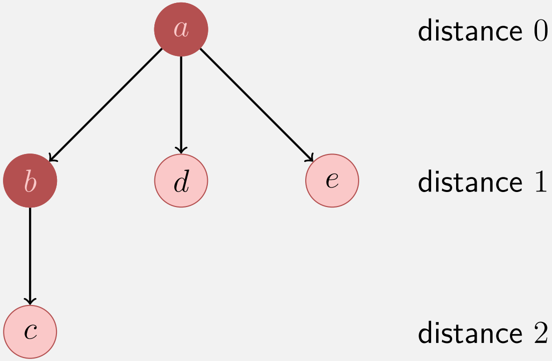
distance 1

Breadth-First-Search BFS

BFS starting from a:

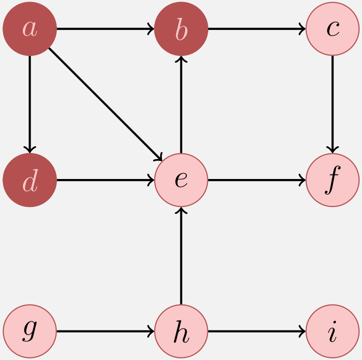


BFS-Tree: Distances and Parents

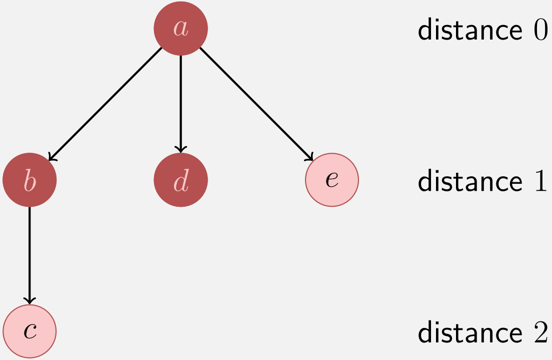


Breadth-First-Search BFS

BFS starting from a:

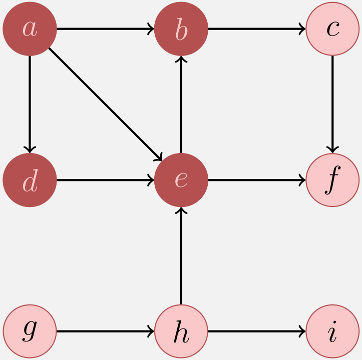


BFS-Tree: Distances and Parents

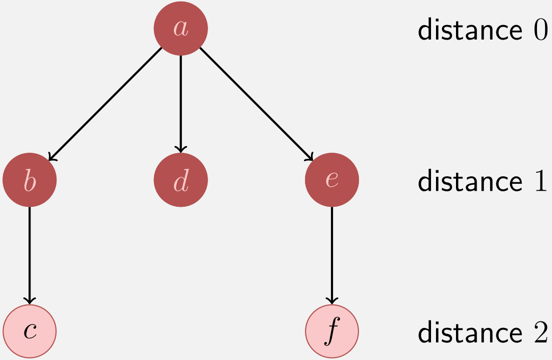


Breadth-First-Search BFS

BFS starting from a:

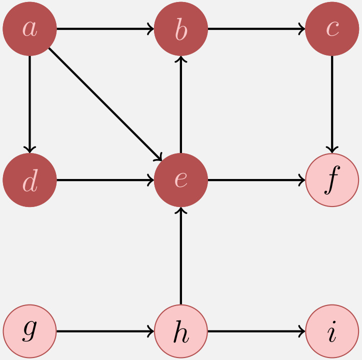


BFS-Tree: Distances and Parents

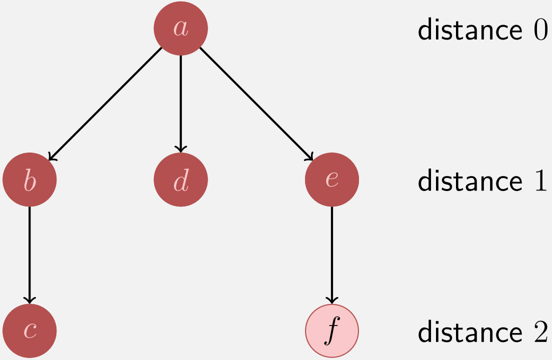


Breadth-First-Search BFS

BFS starting from a:

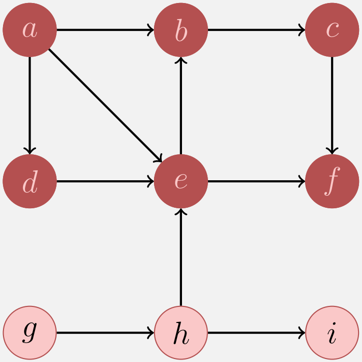


BFS-Tree: Distances and Parents

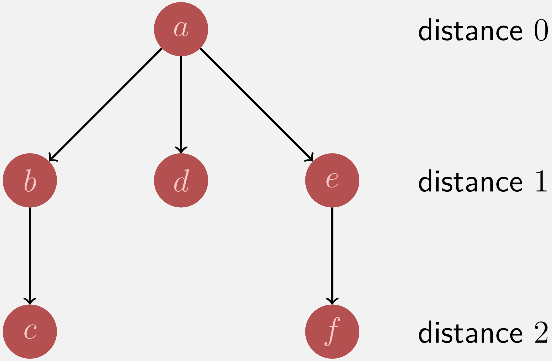


Breadth-First-Search BFS

BFS starting from a:

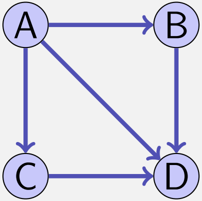


BFS-Tree: Distances and Parents



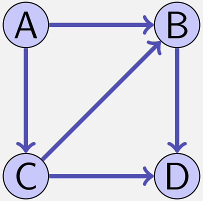
Quiz: Topological Sorting

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



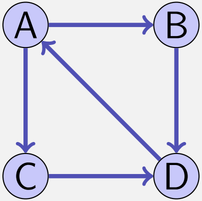
number sortings

?



number sortings

?

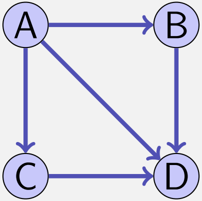


number sortings

?

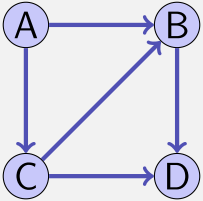
Quiz: Topological Sorting

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



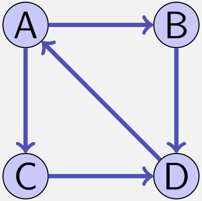
number sortings

2



number sortings

1



number sortings

0

Shortest Paths: General Algorithm

1. Initialise d_s and π_s: $d_s[v] = \infty$, $\pi_s[v] =$ null for each $v \in V$
2. Set $d_s[s] \leftarrow 0$
3. Choose an edge $(u, v) \in E$

   Relaxiere (u, v):

   if $d_s[v] > d[u] + c(u, v)$ then

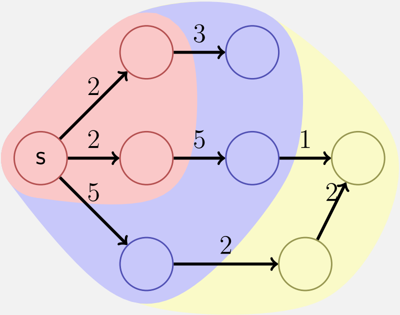
   $d_s[v] \leftarrow d_s[u] + c(u, v)$

   $\pi_s[v] \leftarrow u$
4. Repeat 3 until nothing can be relaxed any more.
   (until $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Dijkstra ShortestPath Basic Idea

Set V of nodes is partitioned into

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

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

General Weighted Graphs

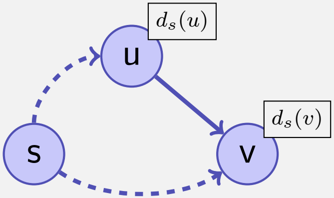
Relax(u, v) $(u, v \in V,\ (u, v) \in E)$

if $d_s(v) > d_s(u) + c(u, v)$ **then**

  $d_s(v) \leftarrow d_s(u) + c(u, v)$

  return true

return false



Problem: cycles with negative weights can shorten the path, a shortest path is not guaranteed to exist.

Dynamic Programming Approach (Bellman)

Induction over number of edges $d_s[i, v]$: Shortest path from s to v via maximally i edges.

$$d_s[i, v] = \min\{d_s[i-1, v], \min_{(u,v)\in E}(d_s[i-1, u] + c(u, v))\}$$

$$d_s[0, s] = 0, d_s[0, v] = \infty \; \forall v \neq s.$$

Algorithm Bellman-Ford(G, s)

Input: Graph $G = (V, E, c)$, starting point $s \in V$
Output: If return value true, minimal weights d for all shortest paths from s, otherwise no shortest path.

foreach $u \in V$ **do**
    $d_s[u] \leftarrow \infty$; $\pi_s[u] \leftarrow$ null
$d_s[s] \leftarrow 0$;
for $i \leftarrow 1$ **to** $|V|$ **do**
    $f \leftarrow$ false
    foreach $(u, v) \in E$ **do**
        $f \leftarrow f \vee \mathrm{Relax}(u, v)$
    if $f =$ false **then return** true
return false;

3. In-Class Exercise

Maze Solver (BFS, DFS, Dijkstra) on code-expert

Colors

Conceptual coloring of nodes

- **white:** node has not been discovered yet.
- **grey:** node has been discovered and is marked for traversal / being processed.
- **black:** node was discovered and entirely processed.

Interpretation of the Colors

When traversing the graph, a tree (or Forest) is built. When nodes are discovered there are three cases

- White node: new tree edge
- Grey node: Zyklus (“back-egde”)
- Black node: forward- / cross edge

Questions?