Datenstrukturen und Algorithmen

Übung 9

FS 2020

Programm von heute

1. Feedback letzte Übungen
2. Wiederholung Theorie
3. In-Class Übung

1. Feedback letzte Übungen

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

siehe Musterlösung mit detaillierten Kommentaren

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. Wiederholung Theorie

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden		
$v \in V$ ohne Nachbar/Nachfolger finden		
$(u, v) \in E$?		
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	
$v \in V$ ohne Nachbar/Nachfolger finden		
$(u, v) \in E$?		
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden		
$(u, v) \in E$?		
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	
$(u, v) \in E$?		
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?		
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen		
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante löschen		

Quiz: Laufzeiten einfacher Operationen

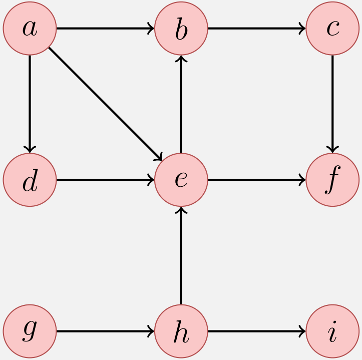
Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante löschen	$\Theta(1)$	

Quiz: Laufzeiten einfacher Operationen

Operation	Matrix	Liste
Nachbarn/Nachfolger von $v \in V$ finden	$\Theta(n)$	$\Theta(\deg^+ v)$
$v \in V$ ohne Nachbar/Nachfolger finden	$\Theta(n^2)$	$\Theta(n)$
$(u, v) \in E$?	$\Theta(1)$	$\Theta(\deg^+ v)$
Kante einfügen	$\Theta(1)$	$\Theta(1)$
Kante löschen	$\Theta(1)$	$\Theta(\deg^+ v)$

Breitensuche BFS

BFS von a aus:

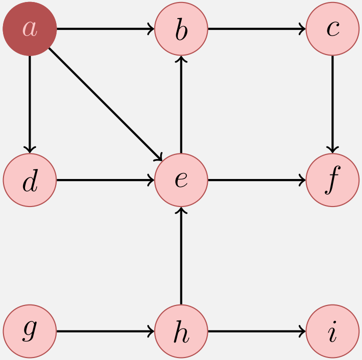


BFS-Baum: Distanzen und Vorgänger

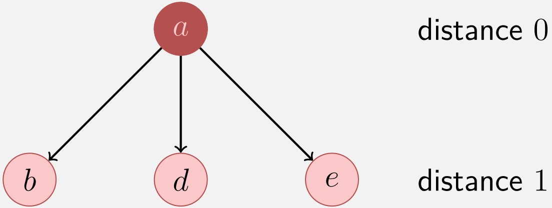
distance 0

Breitensuche BFS

BFS von a aus:

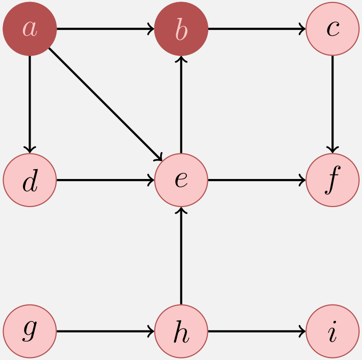


BFS-Baum: Distanzen und Vorgänger

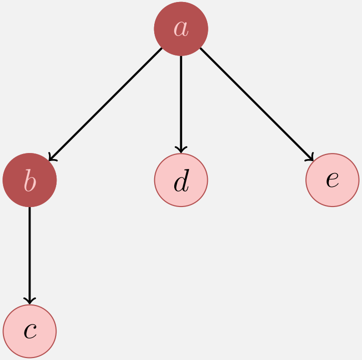


Breitensuche BFS

BFS von a aus:



BFS-Baum: Distanzen und Vorgänger



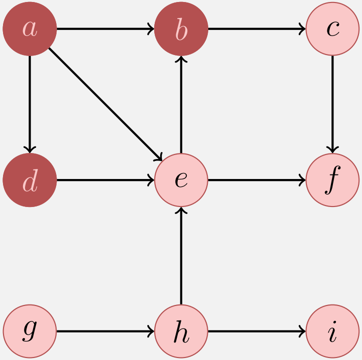
distance 0

distance 1

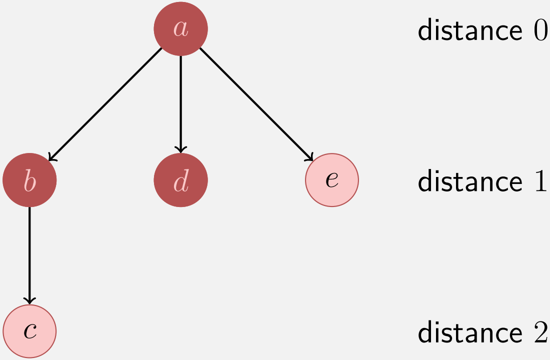
distance 2

Breitensuche BFS

BFS von a aus:

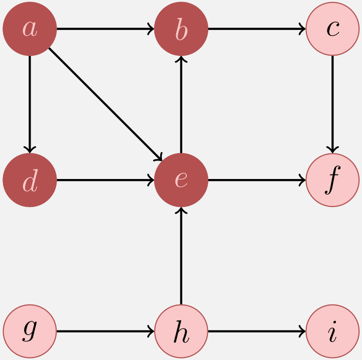


BFS-Baum: Distanzen und Vorgänger

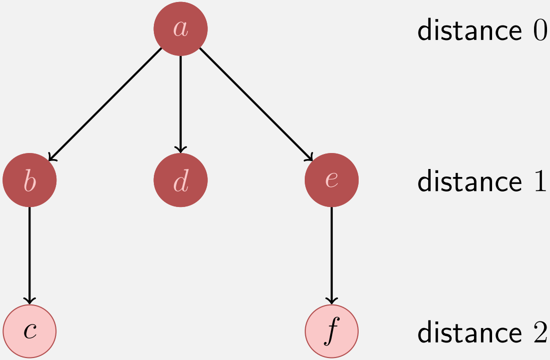


Breitensuche BFS

BFS von a aus:

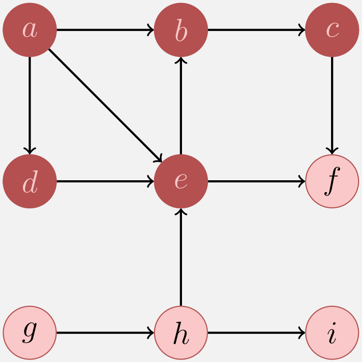


BFS-Baum: Distanzen und Vorgänger

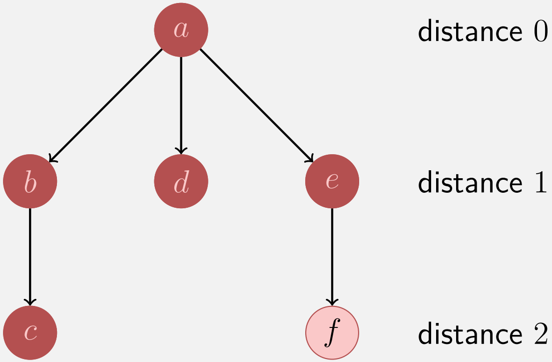


Breitensuche BFS

BFS von a aus:

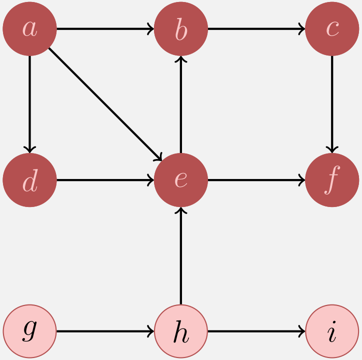


BFS-Baum: Distanzen und Vorgänger

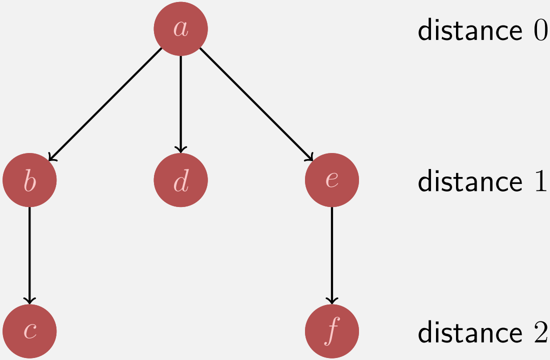


Breitensuche BFS

BFS von a aus:

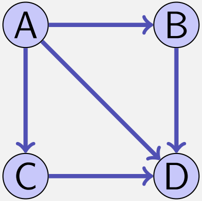


BFS-Baum: Distanzen und Vorgänger



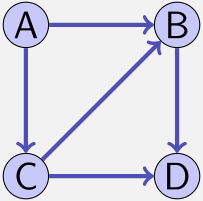
Quiz: Topologisch Sortieren

Auf wie viele Arten können die folgenden gerichteten Graphen jeweils topologisch sortiert werden?



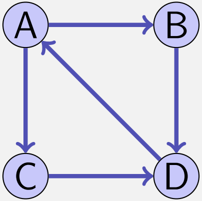
Anzahl Sortierungen

?



Anzahl Sortierungen

?

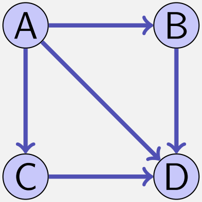


Anzahl Sortierungen

?

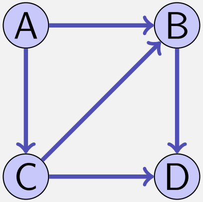
Quiz: Topologisch Sortieren

Auf wie viele Arten können die folgenden gerichteten Graphen jeweils topologisch sortiert werden?



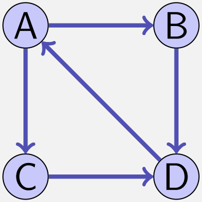
Anzahl Sortierungen

2



Anzahl Sortierungen

1



Anzahl Sortierungen

0

Kürzeste Pfade: Allgemeiner Algorithmus

1. Initialisiere d_s und π_s: $d_s[v] = \infty$, $\pi_s[v] =$ null für alle $v \in V$
2. Setze $d_s[s] \leftarrow 0$
3. Wähle eine Kante $(u, v) \in E$

   Relaxiere (u, v):

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

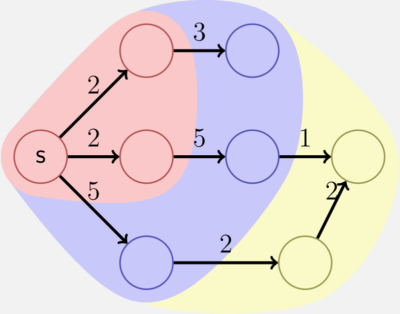
   $d_s[v] \leftarrow d_s[u] + c(u, v)$

   $\pi_s[v] \leftarrow u$
4. Wiederhole 3 bis nichts mehr relaxiert werden kann.
   (bis $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Dijkstra ShortestPath Grundidee

Menge V aller Knoten wird unterteilt in

- die Menge M von Knoten, für die schon ein kürzester Weg von s bekannt ist
- die Menge $R = \bigcup_{v \in M} N^+(v) \setminus M$ von Knoten, für die kein kürzester Weg bekannt ist, die jedoch von M direkt erreichbar sind.
- die Menge $U = V \setminus (M \cup R)$ von Knoten, die noch nicht berücksichtigt wurden.



Algorithmus Dijkstra(G, s)

Input: Positiv gewichteter Graph $G = (V, E, c)$, Startpunkt $s \in V$
Output: Minimale Gewichte d der kürzesten Pfade und Vorgängerknoten für jeden Knoten.

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

Allgemeine Bewertete Graphen

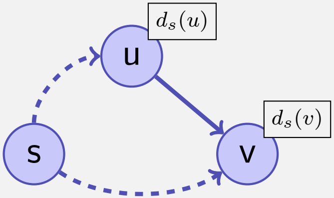
Relax(u, v) $(u, v \in V, (u, v) \in E)$

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: Zyklen mit negativen Gewichten können Weg verkürzen: es muss keinen kürzesten Weg mehr geben

Dynamic-Programming-Ansatz (Bellman)

Induktion über Anzahl Kanten. $d_s[i, v]$: Kürzeste Weglänge von s nach v über maximal i Kanten.

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

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

Algorithmus Bellman-Ford(G, s)

Input: Graph $G = (V, E, c)$, Startpunkt $s \in V$
Output: Wenn Rückgabe true, Minimale Gewichte d der kürzesten Pfade zu jedem Knoten, sonst kein kürzester Pfad.

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 Übung

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

Farben

Konzeptuelle Färbung der Knoten

- **Weiss:** Knoten wurde noch nicht entdeckt.
- **Grau:** Knoten wurde entdeckt und zur Traversierung vorgemerkt / in Bearbeitung.
- **Schwarz:** Knoten wurde entdeckt und vollständig bearbeitet

Interpretation der Farben

Beim Traversieren des Graphen wird ein Baum (oder Wald) aufgebaut.
Beim Entdecken von Knoten gibt es drei Fälle

- Weisser Knoten: neue Baumkante
- Grauer Knoten: Zyklus („Rückwärtskante“)
- Schwarzer Knoten: Vorwärts-/Seitwärtskante

Fragen?