Plan

- Discussion Sheet 5 &6
- Preview Assignment 8
- Recap Graphs (cheatsheet on polybox)
- Exercise 7.3
- Exercise 7.4

Peer Grading is 7.1

Common mistakes with sheets 5,6

5.2:
- not justifying hints
- not defining K_n
- incorrect vords for depth (levels, layers might be used in certain context)

5.3:
- wrong base case (1, but should be 0)
- missing the final conclusion in subtask (c)

5.5: barely any mistakes! ☺

6.1:
- was solved very well!!
- children are only the direct successors of a node (not all nodes belov)
- weird correction guideline that depth $=O(\log n)$ must've been mentioned explicitly

6.3: Stating that $T(n)=T(n-1)+T(n-3)+2\cdot T(n-4)+c$, instead of $T(n-1)+T(n-3)+T(n-4)+c$

6.4: Barely any mistakes

Preview Assignment 8

- Only graph exercises
  - ↳ try to do all of the non-bonus, especially 8.3

Graph Cheatsheet für Algorithmen und Datenstrukturen

Kenji Nakano, HS23, Stand 11.11.2023

Keine Garantie für Vollständigkeit oder Korrektheit

Definition Graph

Ein **Graph** ist ein Tupel $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ wobei

- $V :=$ Knotenmenge (vertices)
- $E :=$ Kantenmenge (edges)

jede Kante ist ein ungeordnetes Paar zweier Knoten $u \neq v$, $e = \{u, v\} \in E$ (Kurzform: uv)

Weg, Pfad, Zyklus

- **Weg:** Folge von benachbarten Knoten (engl. walk)
- **Pfad:** Weg ohne wiederholte Knoten
- **Zyklus:** Weg mit $v_0 = v_l$, $l \geq 2$ (engl. closed walk)

Die Länge einers Wegs bzw. Pfads ist die Anzahl an Kanten, nicht die Anzahl an Knoten

Begriffe

- u, v adjazent/benachbart $\Leftrightarrow e = \{u, v\} \in E$
- $e \in E$ inzident/anliegend zu $v \Leftrightarrow \exists u \in V$, so dass $e = \{u, v\}$
- $deg(u)$ = Knotengrad von u (Anzahl Nachbarn)
- u erreicht $v \Leftrightarrow \exists$ Weg zwischen u und v (engl. reachable)
  Äquivalenzrelation (symmetrisch, reflexiv, transitiv)
- Zusammenhangskomponente (ZHK): Äquivalenzklasse der "erreichbar"-relation (engl. connected component)
- Graph ist zusammenhängend $\Leftrightarrow$ es gibt gibt genau eine ZHK

$$\text{Handschlag Lemma: } \sum_{v \in V} deg(v) = 2 \cdot |E|$$

Eulerweg, Hamiltonpfad, Eulerzyklus

- **Eulerweg:** Weg welcher jede Kante **genau** einmal enthält (engl. Eulerian walk)
- **Hamiltonpfad:** Pfad der jeden Knoten **genau** einmal enthält
- **Eulerzyklus:** Zyklus welcher jede Kante **genau** einmal enthält

$\exists$ Eulerzyklus $\Leftrightarrow$ alle Knotengrade gerade und alle Kanten in einer ZHK

Algorithmus Eulertour / Eulerwalk

> **Euler(G):**
>
> - Input: Graph $G = (V, E)$
> - Output: Liste Z mit Eulerzyklus, falls existiert.
> - Laufzeit: $\mathcal{O}(m)$
>
> **EulerWalk(u):**
>
> - Input: Knoten $u \in V$
> - Output: Keiner
> - Laufzeit: $\mathcal{O}(m)$

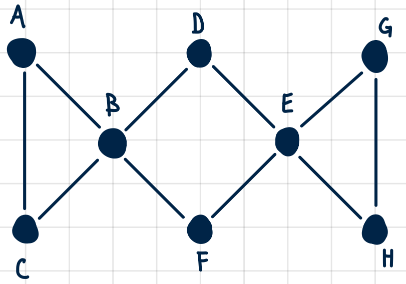
Algorithm 1 Euler(G)

Require: Alle Kanten unmarkiert

$Z \leftarrow$ Leere Liste
EulerWalk(u_0) ▷ für $u_0 \in V$ beliebig
return Z

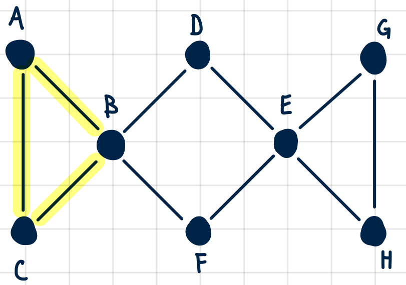
Algorithm 2 EulerWalk(u)

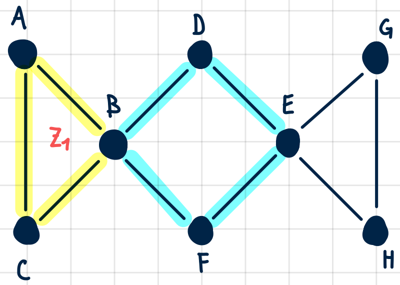
for $uv \in E$, nicht markiert **do**
    markiere Kante uv
    EulerWalk(v)
$Z \leftarrow Z \cup \{u\}$

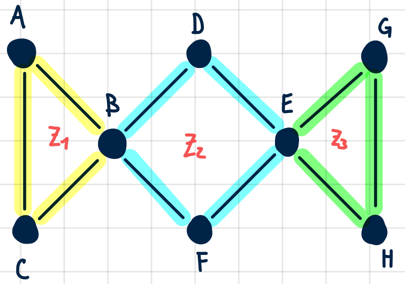


Algorithm 2 EulerWalk(u)

for $uv \in E$, nicht markiert **do**
    markiere Kante uv
    EulerWalk(v)
$Z \leftarrow Z \cup \{u\}$

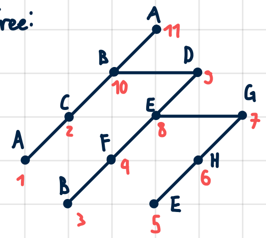




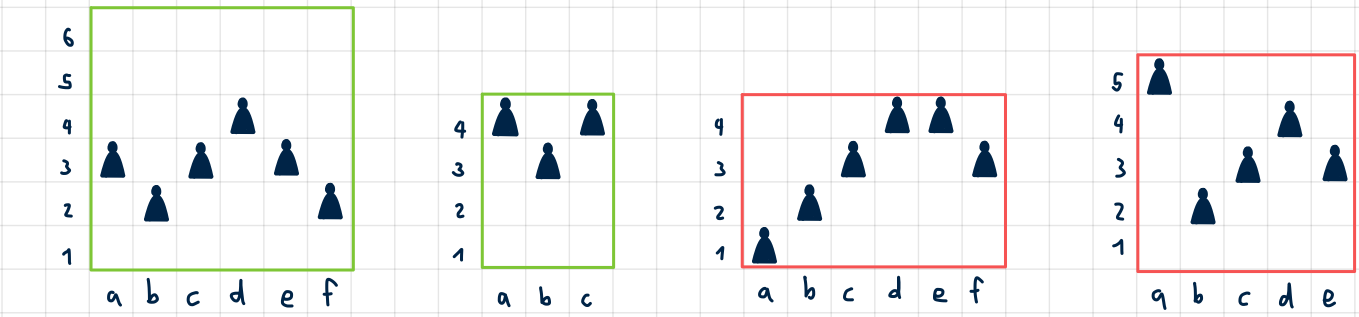


A B C A
B D E F B
E G H E

Recursion Tree:



7.3.) Safe Pawn Lines



Input: $N, M > 0$
Output: Number of safe pawn lines on an $N \times M$ chessboard
Desired Runtime: $O(NM)$

1.) **Dimensions of the DP-Table:** DP-Table is 2-dimensional with indices $\{1,\dots,N\} \times \{1,\dots,M\}$. The number of entries is $N \cdot M$

2.) **Subproblems:** DP[i][j] counts the number of distinct safe pawn lines on a $N \times j$ chessboard with the pawn in the last column located in row i.

3.) **Recursion:**

$$DP[i][j] = \begin{cases} 1, & \text{if } j=1 \quad \text{(base cases)} \\ DP[i+1][j-1], & \text{if } i=1,\ j>1 \quad \text{(top row)} \\ DP[i-1][j-1], & \text{if } i=N,\ j>1 \quad \text{(bottom row)} \\ DP[i+1][j-1]+DP[i-1][j-1], & \text{if } 1<i<N,\ j>1 \quad \text{(all other cases)} \end{cases}$$

4.) **Calculation Order:** We compute one column at a time from left to right, where in each column we go top to bottom   e.g for $j=1,\dots,M$
for $i=1,\dots,N$
DP[i][j] = ...

5.) **Extracting the solution:** The solution is the sum of the last column $\Rightarrow \sum_{i=1}^{N} DP[i][M]$

6.) **Runtime:** Computation: Each entry takes $O(1)$. We have $N \cdot M$ entries $\Rightarrow O(NM)$
Extracting the solution requires $O(1)$ per entry, we look at N entries $\Rightarrow O(N)$
$\Rightarrow$ Total runtime is $O(NM)+O(N) = O(NM)$

7.4.) String Counting

Input: $n,k \in \mathbb{N}$, $n,k > 0$ with $k < n$
Output: Number of binary strings S of length n with $f(S)=k$
Desired Runtime: $O(nk)$

1.) Dimensions of DP-Table: DP-Table is 3-dimensional with indices $\{1,\dots,n\} \times \{0,\dots,k\} \times \{0,1\}$. The number of entries is $n\cdot(k+1)\cdot 2 = O(nk)$

2.) Subproblems: DP[i][j][r] counts the number of binary strings S of length i, with $f(S)=j$ (i.e. j occurences of "11" in S), that end in r

3.) Recursion:

$$DP[i][j][r] = \begin{cases} 1 & \text{if } i=1,\ j=0,\ r\in\{0,1\} & \text{(base cases)} \\ 0 & \text{if } i=1,\ 1\le j\le k,\ r\in\{0,1\} & \text{(base cases)} \\ DP[i-1][j][0] + DP[i-1][j][1] & \text{if } 2\le i\le n,\ 0\le j\le k,\ r=0 & \text{(if string ends in 0)} \\ DP[i-1][j][0] + DP[i-1][j-1][1] & \text{if } 2\le i\le n,\ 1\le j\le k,\ r=1 & \text{(if string ends in 1 and >0 occurences)} \\ DP[i-1][0][0] & \text{if } 2\le i\le n,\ j=0,\ r=1 & \text{(special case (0 occurences, ends in 1))} \end{cases}$$

4.) Calculation order: Increasing order of i, the others don't matter since there is no dependency of entries with the same i. e.g.

```
for i=1,...n
    for j=0,...,k
        for r=0,1
            DP[i][j][r]=...
```

5.) Extracting the solution: The solution is DP[n][k][0]+DP[n][k][1]

6.) Runtime: Computation: We have $n\cdot(k+1)\cdot 2 = O(n\cdot k)$ entries, each computation takes $O(1) \Rightarrow O(n\cdot k)$
Extracting the solution: We look at 2 entries, each takes $O(1) \Rightarrow 2\cdot O(1) = O(1)$
$\Rightarrow$ Total runtime is $O(nk)+O(1) = O(nk)$