Algorithms and Data Structures

Exercise Session 8

https://n.ethz.ch/~ahmala/and

Graph Theory

$G = (V,E)$

Walk(Weg)

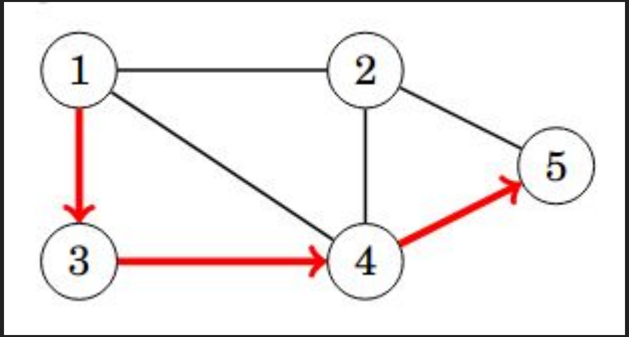
- The sequence v_1, v_2, v_3...,v_n is a **walk** iff there are edges between v_i and v_{i+1} for all $i \in \{1,...,n-1\}$
- Length of the walk?

Closed Walk(Zyklus)

- Walk that ends at the point where it started

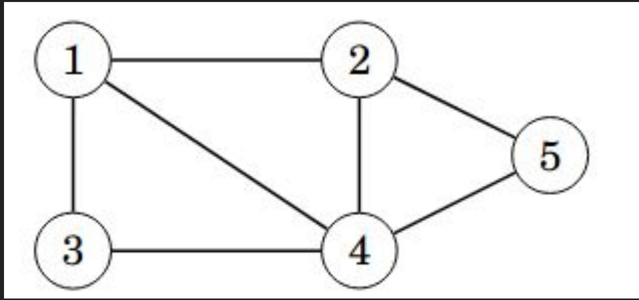
Path(Pfad)

- walk whose vertices are all distinct
- length?



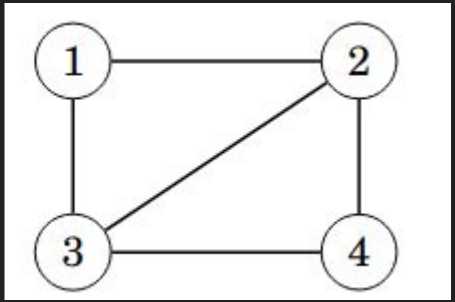
Cycle(Kreis)

- a path is a cycle if the first and last node is the same
- $1 \rightarrow 3 \rightarrow 4 \rightarrow 1$
- If we repeat vertices, then it’s called a circuit, not a cycle.
- Acyclic graph: no cycle in the graph



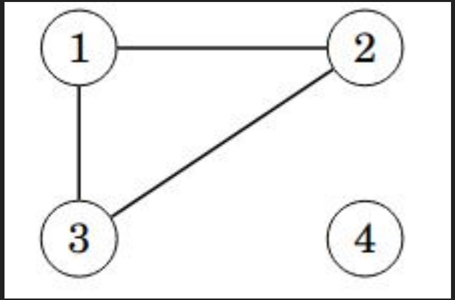
Connectivity

graph is **connected** if there is a path between any two nodes



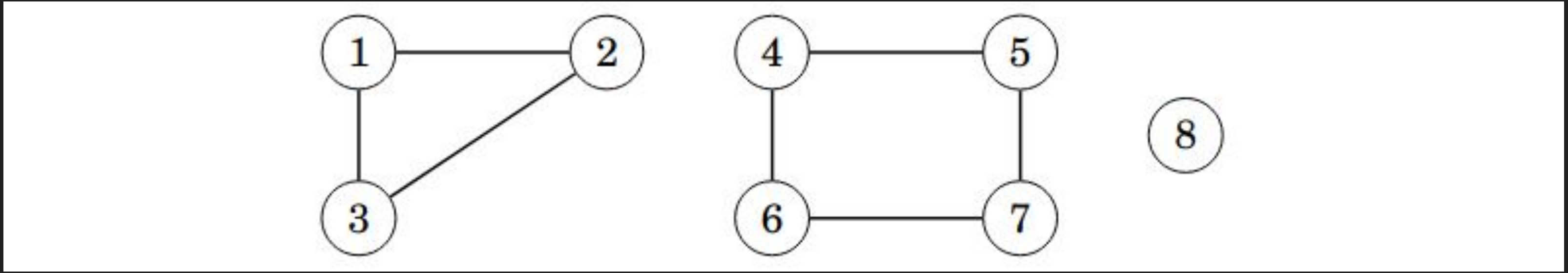
Disconnected

Iff there exists two nodes, which do not have a path between



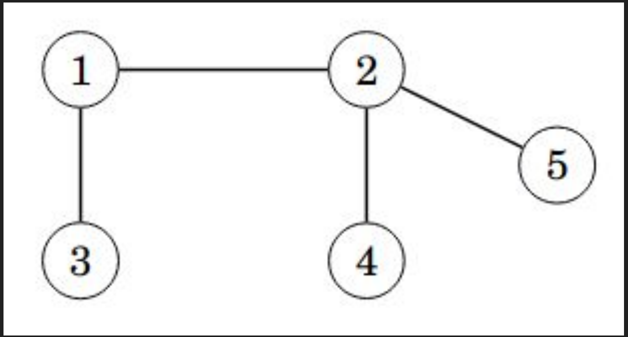
Connected Component (Zusammenhangskomponente ZHK)

- connected parts of a graph are called its components
- following graph contains three components: {1, 2, 3}, {4, 5, 6, 7} and {8}

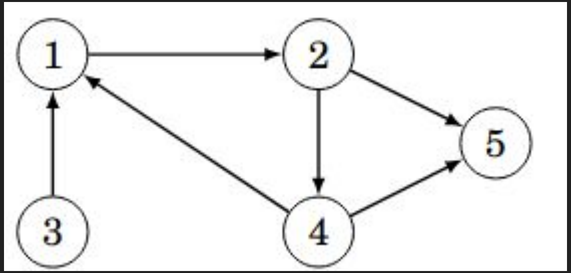


Tree(Baum)

- tree is a connected graph that consists of n nodes and n - 1 edges
    - Alternative definition: connected acyclic graph
- unique path between any two nodes of a tree

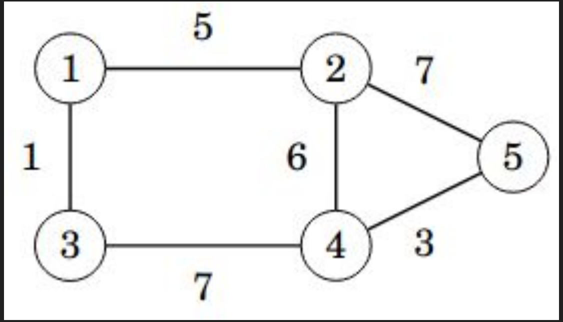


Directed Graph



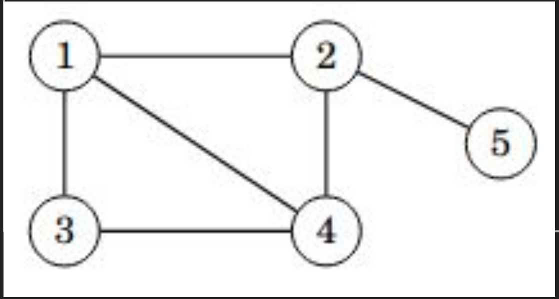
Weighted Graph

- Each edge is assigned a weight



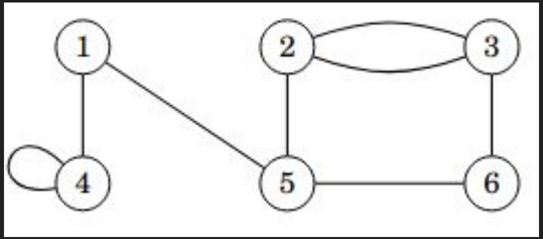
Neighbors and Degrees

- Two nodes are neighbors or adjacent if there is an edge between them
- Degree of a node is the number of its neighbors
- sum of degrees in a graph is always 2m, where m is the number of edges
  - because each edge increases the degree of exactly two nodes by one. For this reason, the sum of degrees is always even.
- N(v): Neighborhood of v.
- deg(v) = |N(v)|



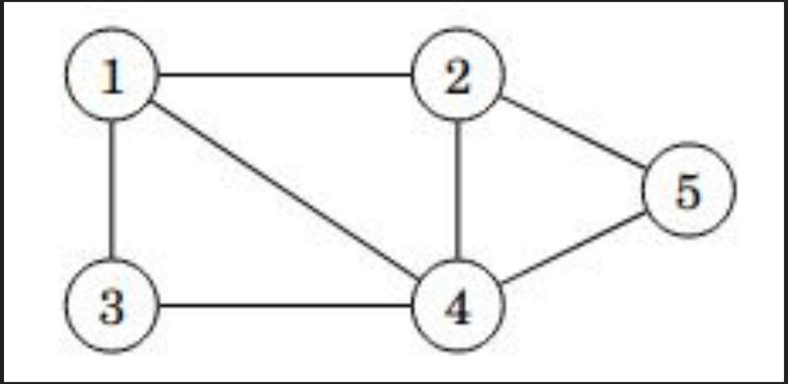
Simplicity

- A graph is simple if no edge starts and ends at the same node, and there are no multiple edges between two node
- Multigraphs may contain several parallel edges between the same pair of nodes—allowing this to happen would make our graph a multigraph (Multigraph)



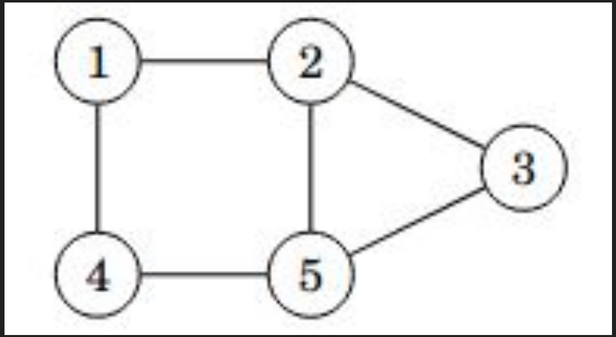
Graph Representation

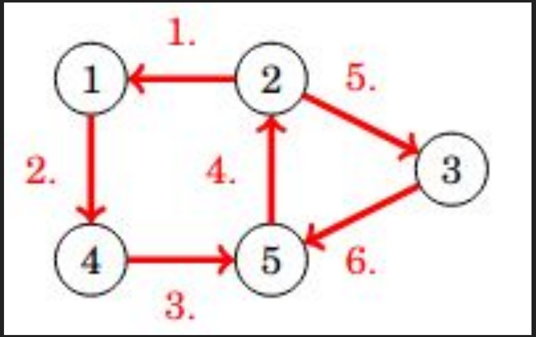
- Matrix Representation
- Adjacency List Representation



Eulerian Walk(Eulerweg)

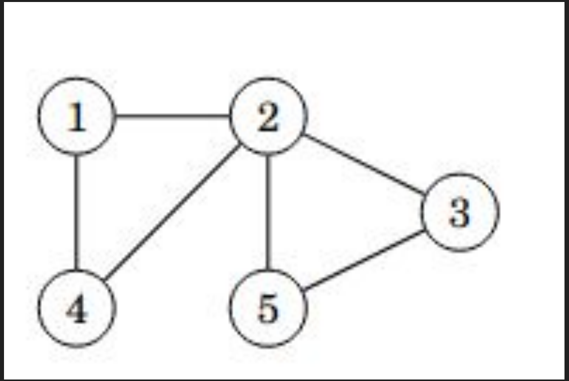
A walk that goes through each edge exactly once

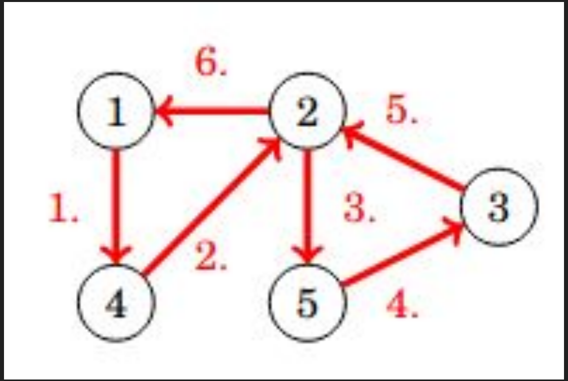




Eulerian Circuit(Eulerkreis)

an Eulerian path that starts and ends at the same node



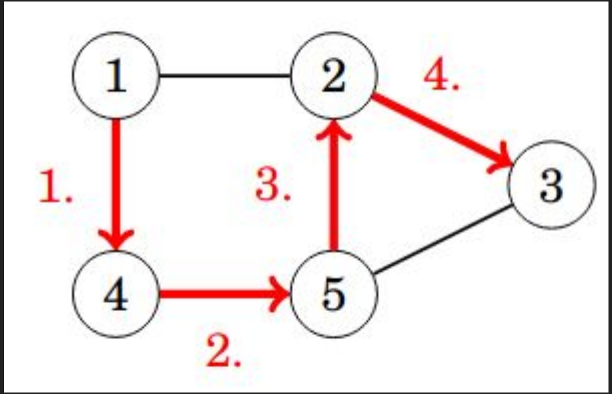


Existence of Eulerian Walk and Circuit

- Degree of each node is even, then eulerian path and circuit
- Degree of exactly two nodes is odd & degree of all the other nodes is even, then eulerian walk

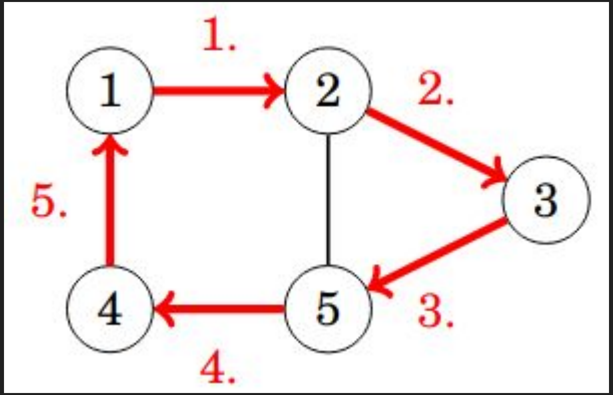
Hamiltonian Path(Hamiltonpfad)

Path that visits all vertices exactly once



Hamiltonian Circuit (Hamiltonkreis)

Cycle that visits all vertices



Exercise Sheet 7

Exercise 7.1 *Subset sums with duplicates* **(1 point)**.

Let $A[1 \dots n]$ be an array containing n positive integers and let $b \in \mathbb{N}$. We want to know if we can write b as a subset sum of A where each element of A is allowed to be used once, twice or not at all. If this is possible, we say b is a subset sum of A with duplicates.

For example, consider $A = [3, 4, 2, 7]$ and $b = 22$. Then b is a subset sum of A with duplicates, as we can use 3 not at all, use 4 once, and use 2 and 7 twice. In other words, we can write b as $0 \cdot A[1] + 1 \cdot A[2] + 2 \cdot A[3] + 2 \cdot A[4]$.

Describe a DP algorithm that, given an array $A[1 \dots n]$ of positive integers and a positive integer b, returns True if and only if b is a subset sum of A with duplicates. Your algorithm should have asymptotic runtime complexity at most $O(b \cdot n)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.2 *Road trip.*

You are planning a road trip for your summer holidays. You want to start from city C_0, and follow the only road that goes to city C_n from there. On this road from C_0 to C_n, there are $n-1$ other cities $C_1, \dots, C_{n-1}$ that you would be interested in visiting (all cities $C_1, \dots, C_{n-1}$ are on the road from C_0 to C_n). For each $0 \le i \le n$, the city C_i is at kilometer k_i of the road for some given $0 = k_0 < k_1 < \dots < k_{n-1} < k_n$.

You want to decide in which cities among $C_1, \dots, C_{n-1}$ you will make an additional stop (you will stop in C_0 and C_n anyway). However, you do not want to drive more than d kilometers without making a stop in some city, for some given value $d > 0$ (we assume that $k_i < k_{i-1} + d$ for all $i \in [n]$ so that this is satisfiable), and you also don't want to travel backwards (so from some city C_i you can only go forward to cities C_j with $j > i$).

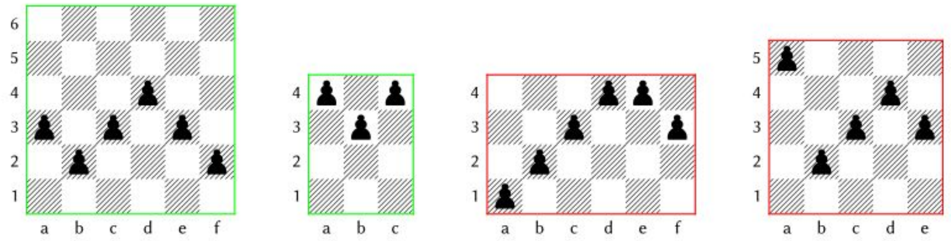
(a) Provide a *dynamic programming* algorithm that computes the number of possible routes from C_0 to C_n that satisfy these conditions, i.e., the number of allowed subsets of stop-cities. Your algorithm should have $O(n^2)$ runtime.

(b) If you know that $k_i > k_{i-1} + d/10$ for every $i \in [n]$, how can you turn the above algorithm into a linear time algorithm (i.e., an algorithm that has $O(n)$ runtime) ?

Exercise 7.3 *Safe pawn lines.*

On an $N \times M$ chessboard (N being the number of rows and M the number of columns), a *safe pawn line* is a set of M pawns with exactly one pawn per column of the chessboard, and such that every two pawns from adjacent columns are located diagonally to each other. When a pawn line is not safe, it is called *unsafe.*

The first two chessboards below show safe pawn lines, the latter two unsafe ones. The line on the third chessboard is unsafe because pawns d4 and e4 are located on the same row (rather than diagonally); the line on the fourth chessboard is unsafe because pawn a5 has no diagonal neighbor at all.



Describe a DP algorithm that, given $N, M > 0$, counts the number of safe pawn lines on an $N \times M$ chessboard. Your solution should have complexity at most $O(NM)$.

Exercise 7.4 *Weight and volume knapsack* **(1 point).**

Consider a knapsack problem with n items with profits being positive integers in the array $P[1, \ldots, n]$, and weights being positive integer in the array $W[1, \ldots, n]$. The knapsack has a weight limit $W_{max} \in \mathbb{N}$. Furthermore, each item has a volume of 1 and the knapsack has a volume limit V_{max}.

Describe a DP algorithm that, given the arrays $P[1, \ldots, n]$, $W[1, \ldots, n]$, of positive integers and positive integers W_{max}, V_{max}, returns the total profit of the largest subset of items such that the items respect both the weight limit and volume limit of W_{max} and V_{max}. Your algorithm should have asymptotic runtime complexity at most $O(n \cdot W_{max} \cdot V_{max})$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.5 *Zebra arrays* **(1 point)**.

A square two-dimensional array $Z[1 \ldots k][1 \ldots k]$ with entries in $\{0, 1\}$ is called a *zebra array* if no two adjacent entries of Z are equal. We say two distinct entries $Z[i_1][j_1]$ and $Z[i_2][j_2]$ in Z are adjacent if

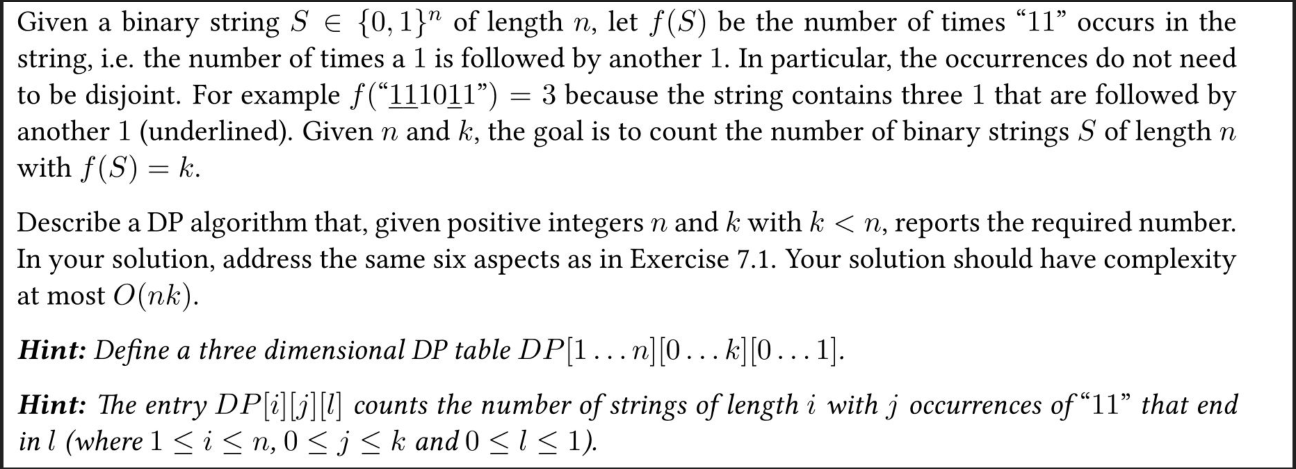
- $i_1 = i_2$ and $|j_1 - j_2| \leq 1$; or
- $|i_1 - i_2| \leq 1$ and $j_1 = j_2$.

Describe a DP algorithm that, given a two-dimensional array $A[1 \ldots n][1 \ldots m]$ with entries in $\{0, 1\}$, outputs the size of a largest zebra array contained in A. That is, the largest k such that, for some

$1 \leq i \leq n - k + 1, 1 \leq j \leq m - k + 1$, the array $A[i \ldots i + k - 1][j \ldots j + k - 1]$ is a zebra array. Your algorithm should have asymptotic runtime complexity at most $O(nm)$.

Extra Tasks

DP Task-1



Given a binary string $S \in \{0,1\}^n$ of length n, let $f(S)$ be the number of times “11” occurs in the string, i.e. the number of times a 1 is followed by another 1. In particular, the occurrences do not need to be disjoint. For example $f(\text{“}\underline{1}\underline{1}10\underline{1}1\text{”}) = 3$ because the string contains three 1 that are followed by another 1 (underlined). Given n and k, the goal is to count the number of binary strings S of length n with $f(S) = k$.

Describe a DP algorithm that, given positive integers n and k with $k < n$, reports the required number. In your solution, address the same six aspects as in Exercise 7.1. Your solution should have complexity at most $O(nk)$.

Hint: *Define a three dimensional DP table $DP[1 \dots n][0 \dots k][0 \dots 1]$.*

Hint: *The entry $DP[i][j][l]$ counts the number of strings of length i with j occurrences of “11” that end in l (where $1 \leq i \leq n, 0 \leq j \leq k$ and $0 \leq l \leq 1$).*

Solution:

1. *Dimensions of the DP table*: $DP[1 \dots n][0 \dots k][0 \dots 1]$.
2. *Subproblems*: The entry $DP[i][j][l]$ describes the number of strings of length i with j occurrences of "11" that end in l.
3. *Recursion*: The base cases for $i = 1$ are given by $DP[1][0][0] = 1$ (the string "0"), $DP[1][0][1] = 1$ (the string "1"), $DP[1][j][0] = 0$ and $DP[1][j][1] = 0$ for $1 \leq j \leq k$. The update rule is as follows: For $1 < i \leq n$ and $0 \leq j \leq k$, to get a string of length i with j occurrences of "11" ending in 0, we can append "0" to a string of length $i - 1$ with j occurrences of "11" ending in 0 or 1, which gives

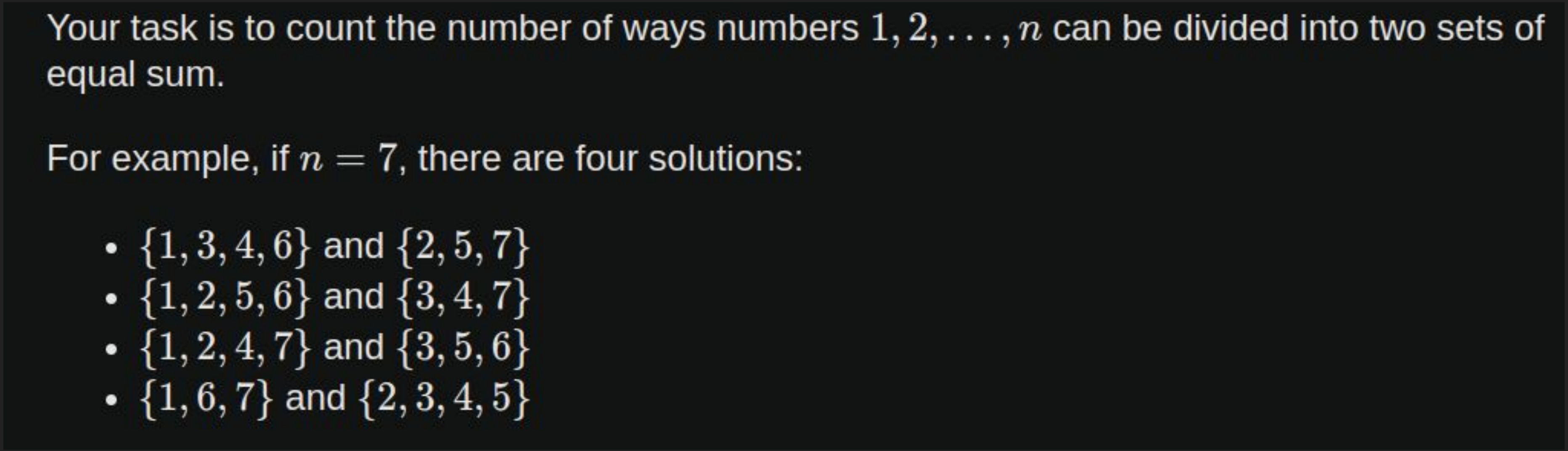
$$DP[i][j][0] = DP[i-1][j][0] + DP[i-1][j][1].$$

   To get a string of length i with j occurrences of "11" ending in 1, we can append "1" to a string of length $i-1$ with j occurrences of "11" ending in 0 or to a string of length $i-1$ with $j-1$ occurrences of "11" ending in 1 (if $j > 0$). Thus,

$$DP[i][j][1] = \begin{cases} DP[i-1][j][0], & \text{if } j = 0 \\ DP[i-1][j][0] + DP[i-1][j-1][1], & \text{if } j > 0. \end{cases}$$

4. *Calculation order*: The entries can be calculated in order of increasing i. There is no interaction between entries with the same i, hence the order within the same value of i can be arbitrary (e.g. increasing in j and l).
5. *Extracting the solution*: The solution is $DP[n][k][0] + DP[n][k][1]$.
6. *Running time*: The running time of the solution is $O(nk)$ as there are $O(nk)$ entries in the table, each of which is processed in $O(1)$ time, and the solution is extracted in $O(1)$.

DP Task-2



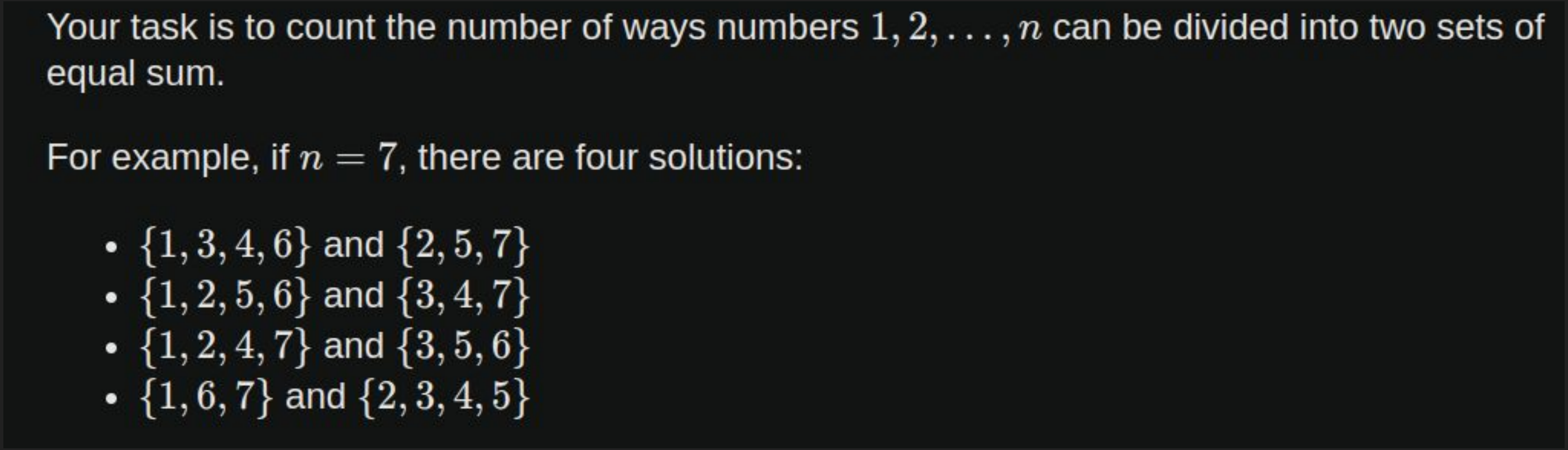
Your task is to count the number of ways numbers $1, 2, \ldots, n$ can be divided into two sets of equal sum.

For example, if $n = 7$, there are four solutions:

- $\{1, 3, 4, 6\}$ and $\{2, 5, 7\}$
- $\{1, 2, 5, 6\}$ and $\{3, 4, 7\}$
- $\{1, 2, 4, 7\}$ and $\{3, 5, 6\}$
- $\{1, 6, 7\}$ and $\{2, 3, 4, 5\}$

O(n^3) solution expected!

Source

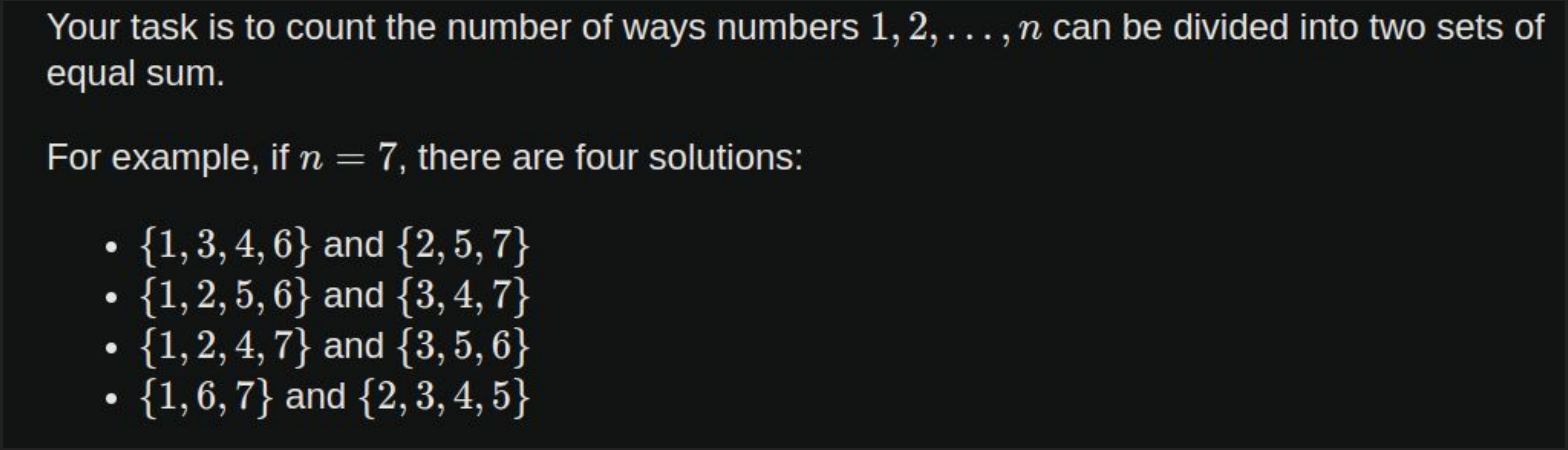

Your task is to count the number of ways numbers $1, 2, \ldots, n$ can be divided into two sets of equal sum.

For example, if $n = 7$, there are four solutions:

- $\{1, 3, 4, 6\}$ and $\{2, 5, 7\}$
- $\{1, 2, 5, 6\}$ and $\{3, 4, 7\}$
- $\{1, 2, 4, 7\}$ and $\{3, 5, 6\}$
- $\{1, 6, 7\}$ and $\{2, 3, 4, 5\}$

dp[i][x] = number of ways to make sum x using subsets of the numbers 1..i

Source



Your task is to count the number of ways numbers $1, 2, \ldots, n$ can be divided into two sets of equal sum.

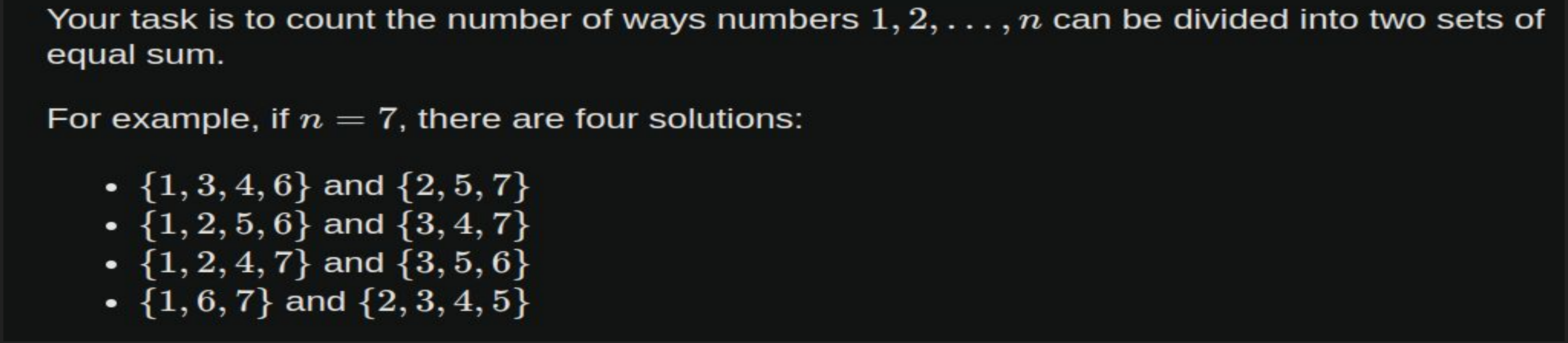
For example, if $n = 7$, there are four solutions:

- $\{1, 3, 4, 6\}$ and $\{2, 5, 7\}$
- $\{1, 2, 5, 6\}$ and $\{3, 4, 7\}$
- $\{1, 2, 4, 7\}$ and $\{3, 5, 6\}$
- $\{1, 6, 7\}$ and $\{2, 3, 4, 5\}$

dp[i][x] = number of ways to make sum x using subsets of the numbers 1..i

dp[i][x] = dp[i-1][x] + dp[i-1][x-i]

Source



Your task is to count the number of ways numbers $1, 2, \ldots, n$ can be divided into two sets of equal sum.

For example, if $n = 7$, there are four solutions:

- $\{1, 3, 4, 6\}$ and $\{2, 5, 7\}$
- $\{1, 2, 5, 6\}$ and $\{3, 4, 7\}$
- $\{1, 2, 4, 7\}$ and $\{3, 5, 6\}$
- $\{1, 6, 7\}$ and $\{2, 3, 4, 5\}$

dp[i][x] = number of ways to make sum x using subsets of the numbers 1..i

dp[i][x] = dp[i-1][x] + dp[i-1][x-i]
The answer is stored at dp[7][14]/2 OR dp[6][14] for n=7

Since you aren't having 'n' in your first set, you'll never have a first set that consists of a combination involving 'n' i.e. all the times a combination requires 'n' you are simply not counting it... thus you end up counting things only once.
Source

Peer Grading

Exercise 7.5