Intro RWTH-BIT

Walter Unger

Lehrstuhl für Informatik I

12. Oktober 2005

Page

http://www.b-it-center.de/Wob/en/view/class211_id329.html

Contents

- data structures

Contents

- data structures
- algorithmic design principles

Contents

- data structures
- algorithmic design principles
- basics of complexity theory

Contents

- data structures
- algorithmic design principles
- basics of complexity theory
- basics of information theory

Contents

- data structures
- algorithmic design principles
- basics of complexity theory
- basics of information theory

- Question: What do you know about these subjects?

Data Structures

- Stacks and queues

Data Structures

- Stacks and queues
- Linked lists

Data Structures

- Stacks and queues
- Linked lists
- Hash tables

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees
- B-Trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees
- B-Trees
- Heaps

Algorithmic design principles

- Dynamic Programming

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms
- Divide and conquer

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms
- Divide and conquer
- Sweep lines

Algorithms

- Sorting

Algorithms

- Sorting
- Single-Source Shortest Paths

Algorithms

- Sorting
- Single-Source Shortest Paths
- All-Pair Shortest Paths

Algorithms

- Sorting
- Single-Source Shortest Paths
- All-Pair Shortest Paths
- Matchings

Algorithms

- Sorting
- Single-Source Shortest Paths
- All-Pair Shortest Paths
- Matchings
- Maximum Flow

Algorithms

- Sorting
- Single-Source Shortest Paths
- All-Pair Shortest Paths
- Matchings
- Maximum Flow
- Approximation Algorithms

Complexity theory: Introduction

- $\mathcal{NP}$-Complete Problems

Complexity theory: Introduction

- $\mathcal{NP}$-Complete Problems
- The Class $\mathcal{P}$

Complexity theory: Introduction

- $\mathcal{NP}$-Complete Problems
- The Class $\mathcal{P}$
- The Class $\mathcal{NC}$

Complexity theory: Introduction

- $\mathcal{NP}$-Complete Problems
- The Class $\mathcal{P}$
- The Class $\mathcal{NC}$
- $\mathcal{P}$-Complete Problems

Complexity theory: Introduction

- $\mathcal{NP}$-Complete Problems
- The Class $\mathcal{P}$
- The Class $\mathcal{NC}$
- $\mathcal{P}$-Complete Problems
- The Halting Problem

Contents (Revisited)

- data structures
- algorithmic design principles
- basics of complexity theory
- basics of information theory

- Question: What do you know about these subjects?
- We start with: basics of complexity theory

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...
- There are Problems where no “fast” solution is known.

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...
- There are Problems where no “fast” solution is known.
- How to see, that a Problem is hard?

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...
- There are Problems where no “fast” solution is known.
- How to see, that a Problem is hard?
  - Start with a machine independent model.

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...
- There are Problems where no “fast” solution is known.
- How to see, that a Problem is hard?
  - Start with a machine independent model.
  - Find the first hard problem for this model.

Complexity theory: Introduction

- There are Problems with fast solutions, i.e. Sorting, ...
- There are Problems where no “fast” solution is known.
- How to see, that a Problem is hard?
  - Start with a machine independent model.
  - Find the first hard problem for this model.
  - Reduce any other hard problem to that first hard problem.

The Machine Model

- The Turing Machine Model

The Machine Model

- The Turing Machine Model
- The Coding of the input

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable
- The Classes $\mathcal{P}$ and $\mathcal{NP}$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable
- The Classes $\mathcal{P}$ and $\mathcal{NP}$
  - $\mathcal{P} = \{L \mid \exists$D-TM which halts in polynomial time on $L\}$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable
- The Classes $\mathcal{P}$ and $\mathcal{NP}$
  - $\mathcal{P} = \{L \mid \exists$D-TM which halts in polynomial time on $L\}$
  - $\mathcal{NP} = \{L \mid \exists$N-TM which halts in polynomial time on $L\}$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable
- The Classes $\mathcal{P}$ and $\mathcal{NP}$
  - $\mathcal{P} = \{L \mid \exists$D-TM which halts in polynomial time on $L\}$
  - $\mathcal{NP} = \{L \mid \exists$N-TM which halts in polynomial time on $L\}$
  - $co - \mathcal{NP} = \{L \mid \overline{L} \in \mathcal{NP}\}$

The Machine Model

- The Turing Machine Model
- The Coding of the input
  - As a language $L \subset \Sigma^*$
  - As a question:
    - Given $e \in \Sigma^*$
    - Question $e \stackrel{?}{\in} L$
- Deterministic and Non-deterministic programs
- The Running Time: Polynomial-time solvable
- The Classes $\mathcal{P}$ and $\mathcal{NP}$
  - $\mathcal{P} = \{L \mid \exists$D-TM which halts in polynomial time on $L\}$
  - $\mathcal{NP} = \{L \mid \exists$N-TM which halts in polynomial time on $L\}$
  - $co - \mathcal{NP} = \{L \mid \overline{L} \in \mathcal{NP}\}$
- Big open problem: $\mathcal{P} \stackrel{?}{\neq} \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$
- There exists $k \in \mathbb{N}$: for any language $L \in \mathcal{NP}$ exists a TM which decides L in time $2^{O(n^k)}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$
- There exists $k \in \mathbb{N}$: for any language $L \in \mathcal{NP}$ exists a TM which decides L in time $2^{O(n^k)}$
- $\mathcal{P} \subseteq \mathcal{NP} \cap co - \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$
- There exists $k \in \mathbb{N}$: for any language $L \in \mathcal{NP}$ exists a TM which decides L in time $2^{O(n^k)}$
- $\mathcal{P} \subseteq \mathcal{NP} \cap co - \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$
- There exists $k \in \mathbb{N}$: for any language $L \in \mathcal{NP}$ exists a TM which decides L in time $2^{O(n^k)}$
- $\mathcal{P} \subseteq \mathcal{NP} \cap co - \mathcal{NP}$

Open are still:

- $\mathcal{NP} \stackrel{?}{=} co - \mathcal{NP}$

First Observations and open Problems:

- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} \subseteq co - \mathcal{NP}$
- There exists $k \in \mathbb{N}$: for any language $L \in \mathcal{NP}$ exists a TM which decides L in time $2^{O(n^k)}$
- $\mathcal{P} \subseteq \mathcal{NP} \cap co - \mathcal{NP}$

Open are still:

- $\mathcal{NP} \stackrel{?}{=} co - \mathcal{NP}$
- $\mathcal{P} \stackrel{?}{=} \mathcal{NP} \cap co - \mathcal{NP}$

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$
- $\forall x \in \{0,1\}^* : x \in L_1 \iff f(x) \in L_2$

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$
- $\forall x \in \{0,1\}^* : x \in L_1 \iff f(x) \in L_2$

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$
- $\forall x \in \{0,1\}^* : x \in L_1 \iff f(x) \in L_2$

Notation: $L_1 \leqslant_p L_2$

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$
- $\forall x \in \{0,1\}^* : x \in L_1 \iff f(x) \in L_2$

Notation: $L_1 \leqslant_p L_2$

If we are able to "solve" L_2 we will also be able to solve L_1.

Reductions

Definition (polynomial-time reducible)

Given languages L_1 and L_2. We say: L_1 is polynomial-time reducible to L_2 iff:

- there exists a polynomial-time computable function f with:
- $f : \{0,1\}^* \mapsto \{0,1\}^*$
- $\forall x \in \{0,1\}^* : x \in L_1 \iff f(x) \in L_2$

Notation: $L_1 \leqslant_p L_2$

If we are able to "solve" L_2 we will also be able to solve L_1.

Lemma

If $L_1 \leqslant_p L_2$ and $L_2 \in \mathcal{P}$ then $L_1 \in \mathcal{P}$.

$\mathcal{NP}$-Completeness

Definition ($\mathcal{NP}$-Completeness)

A language L is $\mathcal{NP}$-Complete iff:

- $L \in \mathcal{NP}$

$\mathcal{NP}$-Completeness

Definition ($\mathcal{NP}$-Completeness)

A language L is $\mathcal{NP}$-Complete iff:

- $L \in \mathcal{NP}$
- $\forall L' \in \mathcal{NP} : L' \leqslant_p L$

$\mathcal{NP}$-Completeness

Definition ($\mathcal{NP}$-Completeness)

A language L is $\mathcal{NP}$-Complete iff:

- $L \in \mathcal{NP}$
- $\forall L' \in \mathcal{NP} : L' \leqslant_p L$

$\mathcal{NP}$-Completeness

Definition ($\mathcal{NP}$-Completeness)

A language L is $\mathcal{NP}$-Complete iff:

- $L \in \mathcal{NP}$
- $\forall L' \in \mathcal{NP} : L' \leqslant_p L$

Notation: $L \in \mathcal{NPC}$

$\mathcal{NP}$-Completeness

Definition ($\mathcal{NP}$-Completeness)

A language L is $\mathcal{NP}$-Complete iff:

- $L \in \mathcal{NP}$
- $\forall L' \in \mathcal{NP} : L' \leqslant_p L$

Notation: $L \in \mathcal{NPC}$

These are the "hardest" problems in $\mathcal{NP}$.

Definition ($\mathcal{NP}$-Hardness)

A language L is $\mathcal{NP}$-Hard iff: $\forall L' \in \mathcal{NP} : L' \leqslant_p L$

Observations

Lemma

$L_1 \leqslant_p L_2$ and $L_2 \leqslant_p L_3$ then $L_1 \leqslant_p L_3$.

Observations

Lemma

$L_1 \leqslant_p L_2$ and $L_2 \leqslant_p L_3$ then $L_1 \leqslant_p L_3$.

Lemma

- *If $L' \leqslant_p L$ and*

Observations

Lemma

$L_1 \leqslant_p L_2$ and $L_2 \leqslant_p L_3$ then $L_1 \leqslant_p L_3$.

Lemma

- *If $L' \leqslant_p L$ and*
- *$L' \in \mathcal{NPC}$ then*

Observations

Lemma

$L_1 \leqslant_p L_2$ *and* $L_2 \leqslant_p L_3$ *then* $L_1 \leqslant_p L_3$.

Lemma

- *If* $L' \leqslant_p L$ *and*
- $L' \in \mathcal{NPC}$ *then*
- *is* L *NP-Hard.*

Observations

Lemma

$L_1 \leqslant_p L_2$ and $L_2 \leqslant_p L_3$ then $L_1 \leqslant_p L_3$.

Lemma

- *If $L' \leqslant_p L$ and*
- *$L' \in \mathcal{NPC}$ then*
- *is L NP-Hard.*
- *If $L \in \mathcal{NP}$ then $L \in \mathcal{NPC}$ holds.*

Circuit-SAT

Definition (Circuit-SAT)

Given (input): A boolean Formula $\mathcal{F}$
Question: is that formula satisfiable?

Circuit-SAT

Definition (Circuit-SAT)

Given (input): A boolean Formula $\mathcal{F}$
Question: is that formula satisfiable?

Definition (Circuit-SAT)

$L_{C-SAT} = \{L \mid L \text{is a coding of a satisfiable boolean formula}$

Circuit-SAT

Definition (Circuit-SAT)

Given (input): A boolean Formula $\mathcal{F}$
Question: is that formula satisfiable?

Definition (Circuit-SAT)

$L_{C-SAT} = \{L \mid L$is a coding of a satisfiable boolean formula

Theorem

$L_{C-SAT} \in \mathcal{NP}$, i.e the language Circuit-SAT is in $\mathcal{NP}$.

Theorem

$L_{C-SAT} \in \mathcal{NPC}$, i.e the language Circuit-SAT is in $\mathcal{NPC}$.

SAT

Definition (SAT)

Given (input): A boolean Formula $\mathcal{F}$ in CNF
Question: is that formula satisfiable?

SAT

Definition (SAT)

Given (input): A boolean Formula $\mathcal{F}$ in CNF
Question: is that formula satisfiable?

Definition (SAT)

$L_{SAT} = \{L \mid L$is a coding of a satisfiable boolean formula in CNF

SAT

Definition (SAT)

Given (input): A boolean Formula $\mathcal{F}$ in CNF
Question: is that formula satisfiable?

Definition (SAT)

$L_{SAT} = \{L \mid L$is a coding of a satisfiable boolean formula in CNF

Theorem

$L_{SAT} \in \mathcal{NP}$, i.e the language SAT is in $\mathcal{NP}$.

Theorem

$L_{SAT} \in \mathcal{NPC}$, i.e the language SAT is in $\mathcal{NPC}$.

3-SAT

Definition (3-SAT)

Given (input): A boolean Formula $\mathcal{F}$ in CNF where each clause contains at most three variables
Question: is that formula satisfiable?

3-SAT

Definition (3-SAT)

Given (input): A boolean Formula $\mathcal{F}$ in CNF where each clause contains at most three variables
Question: is that formula satisfiable?

Theorem

The language 3-SAT is NP-complete.

Clique

Definition (Clique)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Clique?

Clique

Definition (Clique)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Clique?

Theorem

The Clique Problem is NP-complete.

k-Clique

Definition (k-Clique)

Given: A graph $G = (V, E)$
Question: does G contain a k-Clique?

k-Clique

Definition (k-Clique)

Given: A graph $G = (V, E)$
Question: does G contain a k-Clique?

Theorem

The k-Clique Problem is in P.

Independent Set

Definition (Independent Set)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Independent Set?

Independent Set

Definition (Independent Set)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Independent Set?

Theorem

The Independent Set Problem is NP-complete.

Dominating Set

Definition (Dominating Set)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Dominating Set?

Dominating Set

Definition (Dominating Set)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Dominating Set?

Theorem

The Dominating Set Problem is NP-complete.

Vertex-Cover

Definition (Vertex-Cover)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Vertex-Cover?

Vertex-Cover

Definition (Vertex-Cover)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$
Question: does G contain a k-Vertex-Cover?

Theorem

The Vertex-Cover Problem is NP-complete.

3-Coloring

Definition (3-Coloring)

Given: A graph $G = (V, E)$
Question: does G have a 3-Coloring?

3-Coloring

Definition (3-Coloring)

Given: A graph $G = (V, E)$
Question: does G have a 3-Coloring?

Theorem

The 3-Coloring Problem is NP-complete.

2-Coloring

Definition (2-Coloring)

Given: A graph $G = (V, E)$
Question: does G have a 2-Coloring?

2-Coloring

Definition (2-Coloring)

Given: A graph $G = (V, E)$
Question: does G have a 2-Coloring?

Theorem

The 2-Coloring Problem is in P.

3-Coloring-Planar

Definition (3-Coloring-Planar)

Given: A planar graph $G = (V, E)$
Question: does G have a 3-Coloring?

3-Coloring-Planar

Definition (3-Coloring-Planar)

Given: A planar graph $G = (V, E)$
Question: does G have a 3-Coloring?

Theorem

The 3-Coloring-Planar Problem is NP-complete.

3-Coloring-Planar-4

Definition (3-Coloring-Planar-4)

Given: A planar graph $G = (V, E)$ of degree 4
Question: does G have a 3-Coloring?

3-Coloring-Planar-4

Definition (3-Coloring-Planar-4)

Given: A planar graph $G = (V, E)$ of degree 4
Question: does G have a 3-Coloring?

Theorem

The 3-Coloring-Planar-4 Problem is NP-complete.

Hamilton Cycle

Definition (Hamilton Cycle)

Given: A graph $G = (V, E)$
Question: does G have a Hamilton Cycle?

Hamilton Cycle

Definition (Hamilton Cycle)

Given: A graph $G = (V, E)$
Question: does G have a Hamilton Cycle?

Theorem

The Hamilton Cycle Problem is NP-complete.

Traveling Salesman

Definition (Traveling Salesman)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$ with weights on the edges
Question: does G contain a Hamilton Cycle with costs at most k.

Traveling Salesman

Definition (Traveling Salesman)

Given: $k \in \mathbb{N}$ and a graph $G = (V, E)$ with weights on the edges
Question: does G contain a Hamilton Cycle with costs at most k.

Theorem

The Traveling Salesman Problem is NP-complete.

More Problems in $\mathcal{NPC}$

- Subset Sum Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem
- Page Number Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem
- Page Number Problem
- Subgraph Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem
- Page Number Problem
- Subgraph Problem
- Piano Movers Problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem
- Page Number Problem
- Subgraph Problem
- Piano Movers Problem
- Three node disjoint path problem

More Problems in $\mathcal{NPC}$

- Subset Sum Problem
- Coloring Circular Arc Graphs
- Coloring Cycle Graphs
- Edge Coloring Problem
- Planar 3-SAT Problem
- Guardsman Problem
- Search number Problem
- Page Number Problem
- Subgraph Problem
- Piano Movers Problem
- Three node disjoint path problem
- And about 9652 more Problems

P-Completeness

Definition

A problem L is P-Complete iff:

- L ist in $\mathcal{P}$.

P-Completeness

Definition

A problem L is P-Complete iff:

- L ist in $\mathcal{P}$.
- $\forall L' \in \mathcal{P}$ L' may be reduced to L with poly-logarithmic storage:

P-Completeness

Definition

A problem L is P-Complete iff:

- L ist in $\mathcal{P}$.
- $\forall L' \in \mathcal{P}$ L' may be reduced to L with poly-logarithmic storage:
  - a function f computable with poly-logarithmic memory, such that:

P-Completeness

Definition

A problem L is P-Complete iff:

- L ist in $\mathcal{P}$.
- $\forall L' \in \mathcal{P}$ L' may be reduced to L with poly-logarithmic storage:
  - a function f computable with poly-logarithmic memory, such that:
  - $\forall w \in \Sigma^* : w \in L \Leftrightarrow f(w) \in L'$

The Maximum Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

The Maximum Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

Definition (Maximum Matching Problem)

Given: Graph $G = (V, E)$
Output: matching M with: $\forall M' : M \subset M' \subseteq E : M'$ is not a matching.

The Maximum Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

Definition (Maximum Matching Problem)

Given: Graph $G = (V, E)$
Output: matching M with: $\forall M' : M \subset M' \subseteq E : M'$ is not a matching.

Theorem (Maximum Matching Problem)

The Maximum Matching Problem is in $\mathcal{P}$.

Method: Greedy

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$
- Let $M = \emptyset$

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$
- Let $M = \emptyset$
- While $E \neq \emptyset$ do

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$
- Let $M = \emptyset$
- While $E \neq \emptyset$ do
  - Choose $e \in E$

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$
- Let $M = \emptyset$
- While $E \neq \emptyset$ do
  - Choose $e \in E$
  - Let $E := E \setminus \{f \in E \mid e \cap f \neq \emptyset\}$

The Maximal Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

The Maximal Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

Definition (Maximal Matching Problem)

Given: Graph $G = (V, E)$
Output: matching M with: $\forall M' : M'$ is a matching $\Longrightarrow |M'| \leqslant |M|$.

The Maximal Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

Definition (Maximal Matching Problem)

Given: Graph $G = (V, E)$
Output: matching M with: $\forall M' : M'$ is a matching $\Longrightarrow |M'| \leqslant |M|$.

Theorem (Maximal Matching Problem)

The Maximum Matching Problem is in $\mathcal{P}$ for bipatite Graphs.

The Maximal Matching Problem

Definition (Matching)

Let $G = (V, E)$ be a graph.
$M \subseteq E$ is called a matching iff $\forall e, f \in M : e \cap f \neq \emptyset$.

Definition (Maximal Matching Problem)

Given: Graph $G = (V, E)$
Output: matching M with: $\forall M' : M'$ is a matching $\Longrightarrow |M'| \leqslant |M|$.

Theorem (Maximal Matching Problem)

The Maximum Matching Problem is in $\mathcal{P}$ for bipatite Graphs.

Theorem (Maximal Matching Problem)

The Maximum Matching Problem is in $\mathcal{P}$.

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph
- Let $M = \emptyset$

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph
- Let $M = \emptyset$
- While there exists a alternating path $(a_0, a, 1, a_2, \cdots a_l)$ in G with $\{a_{2 \cdot i}, a_{2 \cdot i+1}\} \notin M$ and $\{a_{2 \cdot i+1}, a_{2 \cdot i}\} \in M$ do

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph
- Let $M = \emptyset$
- While there exists a alternating path $(a_0, a, 1, a_2, \cdots a_l)$ in G with $\{a_{2 \cdot i}, a_{2 \cdot i+1}\} \notin M$ and $\{a_{2 \cdot i+1}, a_{2 \cdot i}\} \in M$ do
  - Exchange the edges in P:

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph
- Let $M = \emptyset$
- While there exists a alternating path $(a_0, a, 1, a_2, \cdots a_l)$ in G with $\{a_{2\cdot i}, a_{2\cdot i+1}\} \notin M$ and $\{a_{2\cdot i+1}, a_{2\cdot i}\} \in M$ do
  - Exchange the edges in P:
    - Add the edges of the form $\{a_{2\cdot i}, a_{2\cdot i+1}\}$ to M and

The Maximum Matching Problem

Algorithm:

- Input $G = (V < E)$ bipartite graph
- Let $M = \emptyset$
- While there exists a alternating path $(a_0, a, 1, a_2, \cdots a_l)$ in G with $\{a_{2\cdot i}, a_{2\cdot i+1}\} \notin M$ and $\{a_{2\cdot i+1}, a_{2\cdot i}\} \in M$ do
  - Exchange the edges in P:
    - Add the edges of the form $\{a_{2\cdot i}, a_{2\cdot i+1}\}$ to M and
    - Delete the edges of the form $\{a_{2\cdot i+1}, a_{2\cdot i}\}$ from M

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{e=(v,w)\in E} f(e) = \sum_{e=(w,v)\in E} f(e)$

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{e=(v,w)\in E} f(e) = \sum_{e=(w,v)\in E} f(e)$
- The value of the flow is: $\sum_{e=(s,w)\in E} f(e) - \sum_{e=(w,s)\in E} f(e)$

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{e=(v,w)\in E} f(e) = \sum_{e=(w,v)\in E} f(e)$
- The value of the flow is: $\sum_{e=(s,w)\in E} f(e) - \sum_{e=(w,s)\in E} f(e)$

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{e=(v,w)\in E} f(e) = \sum_{e=(w,v)\in E} f(e)$
- The value of the flow is: $\sum_{e=(s,w)\in E} f(e) - \sum_{e=(w,s)\in E} f(e)$

Definition (Maximal Flow Problem)

Given: Graph $G = (V, E)$, $s, t \in V$ and $c : E \mapsto \mathbb{N}$
Output: maximal flow function f.

The Maximum Flow Problem

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- A function $f : E \mapsto \mathbb{N}$ is call a flow function iff:
- $\forall e \in E : 0 \leqslant f(e) \leqslant c(e)$
- $\forall v \in V \setminus \{s, t\} : \sum_{e=(v,w)\in E} f(e) = \sum_{e=(w,v)\in E} f(e)$
- The value of the flow is: $\sum_{e=(s,w)\in E} f(e) - \sum_{e=(w,s)\in E} f(e)$

Definition (Maximal Flow Problem)

Given: Graph $G = (V, E)$, $s, t \in V$ and $c : E \mapsto \mathbb{N}$
Output: maximal flow function f.

Theorem (Maximal Flow)

The Maximum Flow Problem is in $\mathcal{P}$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do
  - increase the total flow by the minimal value $x(a, b)$ on the path.

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do
  - increase the total flow by the minimal value $x(a, b)$ on the path.
  - changes on forward edges: $f(a, b) := f(a, b) + x(a, b)$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do
  - increase the total flow by the minimal value $x(a, b)$ on the path.
  - changes on forward edges: $f(a, b) := f(a, b) + x(a, b)$
  - changes on backward edges: $f(a, b) := f(a, b) - x(a, b)$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do
  - increase the total flow by the minimal value $x(a, b)$ on the path.
  - changes on forward edges: $f(a, b) := f(a, b) + x(a, b)$
  - changes on backward edges: $f(a, b) := f(a, b) - x(a, b)$
- Running Time if DFS is used: $O(|F| \cdot |E| + |V|)$

Maximal Flow

Algorithm of Ford Fulgerson:

- An edge (a, b) is usable as an forward edge (a, b) if $c((a, b)) - f((a, b)) > 0$
- An edge (a, b) is usable as an backward edge (b, a) if $f((a, b)) > 0$
- An forward edge (a, b) may take the additional flow $x(a, b) = c((a, b)) - f((a, b))$
- An backward edge (b, a) may "take" a flow $x(b, a) = f((a, b))$
- While there is a path of usable edges from s to t do
  - increase the total flow by the minimal value $x(a, b)$ on the path.
  - changes on forward edges: $f(a, b) := f(a, b) + x(a, b)$
  - changes on backward edges: $f(a, b) := f(a, b) - x(a, b)$
- Running Time if DFS is used: $O(|F| \cdot |E| + |V|)$
- Running Time if BFS is used: $O(|V| \cdot |E|^2 + |V|)$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff
- $s \in A$ and $t \in B$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff
- $s \in A$ and $t \in B$
- $A \cap B = \emptyset$ and $A \cup B = V$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff
- $s \in A$ and $t \in B$
- $A \cap B = \emptyset$ and $A \cup B = V$
- The capacity of a A, B cut is: $\sum_{e=(v,w) \in E, v \in A, w \in B} c(e)$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff
- $s \in A$ and $t \in B$
- $A \cap B = \emptyset$ and $A \cup B = V$
- The capacity of a A, B cut is: $\sum_{e=(v,w) \in E, v \in A, w \in B} c(e)$

The Minimal Cut

Definition (Flow)

- Let $G = (V, E)$ be a directed graph with cost function $c : E \mapsto \mathbb{N}$
- Let $s, t \in V$ be the source and sink.
- $A, B \subset V$ is called a cut iff
- $s \in A$ and $t \in B$
- $A \cap B = \emptyset$ and $A \cup B = V$
- The capacity of a A, B cut is: $\sum_{e=(v,w) \in E, v \in A, w \in B} c(e)$

Theorem (Min-Cut-Max-Flow)

The value of the Maximal Flow is the same as the value of the Minimal Cut.

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l + 1$

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l + 1$
  - Compute a maximal flow on the level network

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l + 1$
  - Compute a maximal flow on the level network
    - while there is a path from s to t which may enlarge the flow

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l + 1$
  - Compute a maximal flow on the level network
    - while there is a path from s to t which may enlarge the flow
    - enlarge the flow

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l + 1$
  - Compute a maximal flow on the level network
    - while there is a path from s to t which may enlarge the flow
    - enlarge the flow
  - "add" the flow from the level network to the graph

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l+1$
  - Compute a maximal flow on the level network
    - while there is a path from s to t which may enlarge the flow
    - enlarge the flow
  - "add" the flow from the level network to the graph
- Running time for level network: $O(|E|^2)$ .... $O(|V| \cdot |E|)$

Maximal Flow

Algorithm of Dinic:

- While there is a Path of usable edges from s to t do
  - Compute a level network of usable edges:
    - Execute a BFS on the usable edges
    - The level network exits only of edges from a level l to $l+1$
  - Compute a maximal flow on the level network
    - while there is a path from s to t which may enlarge the flow
    - enlarge the flow
  - "add" the flow from the level network to the graph
- Running time for level network: $O(|E|^2)$ .... $O(|V| \cdot |E|)$
- Running time: $O((|V|^2 \cdot |E|))$

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network
  - Take the node with positive and minimal potential

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network
  - Take the node with positive and minimal potential
  - propagate that potential forward to t

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network
  - Take the node with positive and minimal potential
  - propagate that potential forward to t
  - propagate that potential backward to s

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network
  - Take the node with positive and minimal potential
  - propagate that potential forward to t
  - propagate that potential backward to s
- Running time for level network: $O(|V|^2)$

Maximal Flow

Algorithm with Forward/Backward Propagation:

- Potential of an edge (a, b): $pot((a, b)) := c((a, b)) - f((a, b))$
- Potential of an node v:
  $\min(\sum_{(v,w)\in E} pot((v, w)), \sum_{(w,v)\in E} pot((v, w)))$
- Compute a maximal flow on the level network
  - Take the node with positive and minimal potential
  - propagate that potential forward to t
  - propagate that potential backward to s
- Running time for level network: $O(|V|^2)$
- Running time: $O((|V|^3)$

The Euler Tour

Definition (Edge Graph)

- Let $G = (V, E)$ be a directed graph, then

The Euler Tour

Definition (Edge Graph)

- Let $G = (V, E)$ be a directed graph, then
- $E(G) = (E, \{((a, b), (b, c)) \mid (a, b), (b, c) \in E\})$ is called the edge graph of G.

The Euler Tour

Definition (Edge Graph)

- Let $G = (V, E)$ be a directed graph, then
- $E(G) = (E, \{((a, b), (b, c)) \mid (a, b), (b, c) \in E\})$ is called the edge graph of G.

The Euler Tour

Definition (Edge Graph)

- Let $G = (V, E)$ be a directed graph, then
- $E(G) = (E, \{((a, b), (b, c)) \mid (a, b), (b, c) \in E\})$ is called the edge graph of G.

Definition (Euler Tour Problem)

Given: Graph $G = (V, E)$
Output: a Hamilton Cycle in $E(G)$

The Euler Tour

Definition (Edge Graph)

- Let $G = (V, E)$ be a directed graph, then
- $E(G) = (E, \{((a, b), (b, c)) \mid (a, b), (b, c) \in E\})$ is called the edge graph of G.

Definition (Euler Tour Problem)

Given: Graph $G = (V, E)$
Output: a Hamilton Cycle in $E(G)$

Theorem (Euler Tour)

The Euler Tour Problem is in $\mathcal{P}$.

Approximation of Metric TSP

Definition (Metric Graph)

- Let $G = (V, E)$ be a graph with cost function $c : E \mapsto \mathbb{N}$

Approximation of Metric TSP

Definition (Metric Graph)

- Let $G = (V, E)$ be a graph with cost function $c : E \mapsto \mathbb{N}$
- G is metric iff $\forall a, b, c \in V : c(\{a, b\}) + c(\{a, b\}) \leqslant c(\{a, b\})$

Approximation of Metric TSP

Definition (Metric Graph)

- Let $G = (V, E)$ be a graph with cost function $c : E \mapsto \mathbb{N}$
- G is metric iff $\forall a, b, c \in V : c(\{a, b\}) + c(\{a, b\}) \leqslant c(\{a, b\})$

Approximation of Metric TSP

Definition (Metric Graph)

- Let $G = (V, E)$ be a graph with cost function $c : E \mapsto \mathbb{N}$
- G is metric iff $\forall a, b, c \in V : c(\{a, b\}) + c(\{a, b\}) \leqslant c(\{a, b\})$

Theorem

The TSP Problem may be approximated within a factor of 1.5 for metric graphs.

Data Structures

- Stacks and queues

Data Structures

- Stacks and queues
- Linked lists

Data Structures

- Stacks and queues
- Linked lists
- Hash tables

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees
- B-Trees

Data Structures

- Stacks and queues
- Linked lists
- Hash tables
- Binary search trees
- Red-Black Trees
- B-Trees
- Heaps

Algorithmic design principles

- Dynamic Programming

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms
- Divide and conquer

Algorithmic design principles

- Dynamic Programming
- Greedy Algorithms
- Divide and conquer
- Sweep lines