Outline

Lecture Content

1. Preface
2. Basics
3. Object orientation
4. Data Structures
5. Algorithms
   5.1. Introduction
   5.2. Characteristics
   5.3. Complexity
   5.4. Design Methods
   5.5. Examples of Algorithms
      5.5.1 Sorting procedures
      5.5.2 Voronoi Diagrams
      5.5.3 Convex Cover
      5.5.4 Network flow
Bubblesort

Principle:

- Bubblesort is one of the simplest sorting algorithms.
- In the first passage, the smallest element is sought for, in the second passage, the second smallest, etc.
- The array is traversed from back to front. First, the last element is compared to the second last element. If the last element is smaller than the second last, they switch. The same operation is performed with the second last and the third last element. By this, the larger elements stay in place, while the smaller elements are forwarded.
- Depending on the sorting direction (upwards or downwards), the bigger or the smaller elements ascent like bubbles in the water.
Principle:

- Quicksort chooses an element (pivot element) from the list to be sorted and divides the list into two sublists, a lower one, that contains all elements smaller and an upper one, that contains all elements bigger or equal to the pivot element.

- For this purpose, an element is sought in the lower list that is bigger than (or equal to) the pivot element. This element is too big for the lower list and mustn’t stay there. Accordingly, an element smaller than the pivot element is sought in the upper list. These elements switch places and are thus placed in the correct sublist. The procedure is repeated, until the upper and the lower search meet. Thus two correct sublists (see above) are created in one run.
Mergesort

Principle:

- Similar to quicksort, the method is based on the Divide-and-Conquer-strategy.
- The list to be sorted is subdivided into two sublists of equal size.
- The corresponding lists are subdivided continuously, until they reach a length of 1.
- Merging is performed by comparing the first elements of two sublists: the smaller one is deleted and added to the new (sorted) list.
Example: Mergesort (1)

5.5. Examples for Algorithms

5.5.1. Sorting Procedures

Unsorted Data Set

Part A

Part B

Part A (sorted)

Part B (sorted)

Sorted Data Set
Example: Mergesort (2)

Procedure: Divide the list into two lists (of the same size) and divide these again and again until the sub-lists contain only one single element.
Example: Mergesort (3)

Now, two sublists are sorted into one list that is twice as long. For this, the first elements of each list are compared and the smaller one is transferred into the new list, until all the elements are sorted.

5.5. Examples for Algorithms
5.5.1. Sorting Procedures
Example: Mergesort (4)

- The last step consists of merging two lists of length \( n/2 \) to one list of length \( n \) by sorting them.
  - Compare both halves with an index \( i \) and an index \( j \) for each element and copy the next largest element in the results list.

```
List 1
2 3 4 5

List 2
0 1 6 7

index i

index j

Results list (empty)
```

5.5. Examples for Algorithms
5.5.1. Sorting Procedures
Example: Mergesort (5)

• Compare the first element of the first list with the first element of the second list.
• Copy the smallest element into the results list.

List 1

\[
\begin{array}{cccc}
2 & 3 & 4 & 5 \\
\end{array}
\]

List 2

\[
\begin{array}{cccc}
0 & 1 & 6 & 7 \\
\end{array}
\]

Results list

\[
\begin{array}{cccccc}
0 & & & & & \\
\end{array}
\]
Example: Mergesort (6)

- Compare the first element of the first list with the second element of the second list.
- Copy the smallest element into the results list.

List 1:
2 3 4 5

List 2:
0 1 6 7

Results list:
0 1
Example: Mergesort (7)

- Compare the first element of the first list with the third element of the second list.
- Copy the smallest element into the results list.

List 1: 2 3 4 5

Index i: 0 1 2

List 2: 0 1 6 7

Index j: 0 1 2 3

Results list: 0 1 2
Example: Mergesort (8)

- Compare the second element of the first list with the third element of the second list.
- Copy the smallest element into the results list.
Example: Mergesort (9)

- Compare the third element of the first list with the third element of the second list.
- Copy the smallest element into the results list.

List 1

2 3 4 5

List 2

0 1 6 7

Results list

0 1 2 3 4
Example: Mergesort (10)

- Compare the fourth element of the first list with the third element of the second list.
- Copy the smallest element into the results list.

List 1

- 2 3 4 5

List 2

- 0 1 6 7

Results list

- 0 1 2 3 4 5
Example: Mergesort (11)

- Copy the next element from the second list into the results list.

List 1: 5 4 3 2

List 2: 7 6 1 0

Results list: 0 1 2 3 4 5 6
Example: Mergesort (12)

- Copy the last element from the second list in the results list.

List 1
2 3 4 5

List 2
0 1 6 7

Results list
0 1 2 3 4 5 6 7
Example: Mergesort (13)

5.5. Examples for Algorithms

5.5.1. Sorting Procedures

Prof. Dr. Dr.-Ing. Jivka Ovtcharova – CSE-Lecture – Ch. 5 - WS 08/09 - Slide 18
Example: Mergesort – Java Implementation

```java
public class MergeSorter {
    private static int[] a, b; // Help array b
    private static void merge(int lo, int m, int hi) {
        int i, j, k;
        // copy both halves of a in the help array b
        for (i=lo; i<=hi; i++)
            b[i]=a[i];
        i=lo; j=m+1; k=lo;
        // copy back the next largest element
        while (i<=m && j<=hi)
            if (b[i]<=b[j])
                a[k++] = b[i++];
            else
                a[k++] = b[j++];
        // copy the rest of the first half if still there
        while (i<=m)
            a[k++] = b[i++];
        while (j<=hi)
            a[k++] = b[j++];
    }

    private static void mergesort(int lo, int hi) {
        if (lo<hi) {
            int m=(lo+hi)/2;
            mergesort(lo, m);
            mergesort(m+1, hi);
            merge(lo, m, hi);
        }
    }

    public static void sort(int[] a0) {
        a = a0; System.out.println("unsorted order:");
        for (int i=0; i<a.length; i++)
            System.out.print(a[i] + " ");
        int n=a.length; b=new int[n]; mergesort(0, n-1);
        System.out.println("sorted order:");
        for (int i=0; i<a.length; i++)
            System.out.print(a[i] + " ");
    }
}
```

// The function mergesort sorts an order a
// from the lowest index lo to the upper index hi

```java
private static void mergesort(int lo, int hi) {
    if (lo<hi) {
        int m=(lo+hi)/2;
        mergesort(lo, m);
        mergesort(m+1, hi);
        merge(lo, m, hi);
    }
}
```

5.5. Examples for Algorithms
5.5.1. Sorting Procedures
// The TestMergeSorter is a test class
public class TestMergeSorter {
    public static void main(String[] args) {
        int array[] = {2, 5, 4, 3, 1, 7, 0, 6};
        MergeSorter.sort(array);
    }
}

Output on the screen:
The unsorted order: 2 5 4 3 1 7 0 6
The sorted order: 0 1 2 3 4 5 6 7
• Bubble sort
  - Expense:
    - Best case: field is already sorted: \( T(n) = O(n) \)
    - Worst case: field is decreasingly sorted: \( T(n) = O(n^2) \)
  - Advantages and disadvantages:
    - 😊 Easy to program and understand
    - 😊 Can easily be modified to be very fast by sorting data
    - 😊 Doesn’t need additional storage, stable
    - 🙁 Slow with random data, because there are many comparisons
    - 🙁 Still slow when the data is partially sorted.
• Quick sort

  - Expense:
    - Best case: \( T(n) = \Theta(n \log(n)) \)
    - Worst case: \( T(n) = \Theta(n^2) \)

  - Advantages and disadvantages:
    - 🎈 Doesn’t need additional storage
    - 🎈 As a rule, generally faster than Merge sort by a factor of 2
    - 🎈 In practice, often the fastest algorithm
    - ☹️ Not stable
    - ☹️ Worst case \( \Theta(n^2) \)
• Merge Sort

- Expense: \( T(n) = \Theta(n \log(n)) \)

- Advantages and disadvantages:
  - 😊 Can be bettered in the sense that you can separately treat simple cases with 2 or 3 elements.
  - 😊 Expense for all cases is \( T(n) = \Theta(n \log(n)) \)
  - 😞 Algorithm requires additional storage space.
  - 😞 Not stable

Note: The consistent generation of help fields is ineffective. Therefore, an algorithm is faster when you create a large help field and apply this over and over again.
As an interdisciplinary subject area of technical computer science, robotics deals with the navigation processes for a mobile and sensor-led robot, among others.

The navigation of a mobile robot is outlined in the following steps:

1. **Mapping and localization**
   Generate a map of the environment and determine your position

2. **Path design**
   Calculate the optimal path from your current point to the goal point

3. **Rules of motion**
   Follow the previously calculated path with help from sensors

*Source: Fraunhofer IPA*
As an example to the practical application of algorithms, we will take a closer look at the path design.

The goal of path design in robotics is to give robots the ability to independently design their own movements and actions starting from the start situation to the target situation.

This is essential for being able to develop autonomous robots.

Even when such autonomy is not required, these processes can serve as programming relief for users, in that he only has to define the task, since the robot can then carry out the rest of the task.

Complex tasks in a dynamic environment make manual path specification almost impossible.
Example (1)

• Exploratory robots
  - Steering exploratory robots on Mars is not possible due to the long signal run time between earth and Mars.
  - If a controller were to send a steering signal to the vehicle, he would have to wait 40 minutes until he sees from what cliff he is going to fall down.

source: NASA
Example (2)

- **Service robots**
  - Service robots perform tasks for people that are, for instance, not able to be performed by people or are too dangerous.
  - This type of robot is still being developed, and it just now starting to be seen as a possibility for performing household and office tasks.
  - Possible application areas are: assistance systems, museum robots, autonomous road sweepers, and production assistants.

*Source: Fraunhofer IPA*
Example (3)

- Game robots
  - The fast progressive development and potential of robots is shown every year in RoboCup
  - The latest processes and constructions are demonstrated in a fun and competitive manner against each other.

source: RoboCup 2005, Noriaki Mitsunaga
Problem Area: Path design

- **Problem of path design:**
  Starting from a start position and orientation, a target position and orientation should be achieved. Determine the path $\pi$, that defines the complete order of positions and orientations, so that no barrier is touched and the goal is reached.

**Example:** The parallel parking problem

*source: Marco Loh, TU München*
As a possible path design strategy, a Voronoi diagram can be used.

In the context of path design, the Voronoi diagram shows those paths that maintain a maximum distance to the barriers (points).
One possibility for constructing Voronoi Diagrams is the following recursive algorithm, based on the principle of the „divide and conquer“ approach.

**Requirements**

A number \( P \) of \( n \) points are given in a plane.

**Algorithm**

1. Divide \( P \) into 2 equally sized heaps \( P_1 \) and \( P_2 \)
2. Calculate the Voronoi Diagram for \( P_1 \) and \( P_2 \)
3. Merge both Voronoi Diagrams for \( P_1 \) and \( P_2 \) to the Voronoi Diagram for \( P \)
Construction of Voronoi Diagrams – Example (1)

Given heap of points $P$

$p_1$

$p_2$

$p_3$

$p_4$

$p_5$
1. Divide $P$ into 2 equally sized heaps $P_1$ and $P_2$
2. Calculate the Voronoi Diagram for $P_1$ and $P_2$

By the recursive break down of the point heap $P$, the construction problem area of a Voronoi Diagram can be reduced to 2 simple cases.

**Case 1:** Voronoi Diagram construction with **two points**

The **vertical line** sets up a Voronoi Diagram with 2 points.
2. Calculate the Voronoi Diagram for $P_1$ and $P_2$

By the recursive break down of the point heap $P$, the construction problem area of a Voronoi Diagram can be reduced to 2 simple cases.

**Case 2: Voronoi Diagram construction with three points**

The vertical lines of all point pairs are cut at a common intersection. They set up a Voronoi diagram with 3 points.
2. Calculate the Voronoi Diagram for $P_1$ and $P_2$
3. Merge both Voronoi Diagrams for $P_1$ and $P_2$

3.1. Connect the closests neighbour along the dividing line

![Diagram of Voronoi Diagrams with points $p_1$, $p_2$, $p_3$, $p_4$, $p_5$, $P_1$, and $P_2$.]
3. Merge both Voronoi Diagrams for $P_1$ and $P_2$

3.2. Construct the new vertical lines
3. Merge both Voronoi Diagrams for $P_1$ and $P_2$

3.3. Cut the vertical lines at the intersecting points
Result: The finished Voronoi Diagram for $P$
Voronoi Diagrams: An Application

5.5. Examples for Algorithms

5.5.2. Voronoi Diagrams

• Example of a complex environment and the corresponding Voronoi Diagram and the chosen path of a mobile robot

Voronoi Diagrams: Advantages and Disadvantages

- **Advantages:**
  - Maximum distance to the barriers
  - The robot can easily test-with help from the distancing sensors-if the correct path will be followed and if it’s not to be followed, correct itself.

- **Disadvantages:**
  - The path is normally not the shortest way
  - When there are only a few barriers, then only a few paths can be generated

source: Fraunhofer IPA
The Convex Cover (1)

- Definition: A set of plane points is called convex if it contains for every two points also the connecting line between these points.

convex Set A

non-convex Set B

- Definition: The convex cover of a set of n points is the smallest convex polygon containing all n points.

5.5. Examples for Algorithms

5.5.3. Convex Cover
• Motivation: Calculate a route for a robot

- Situation:
  - A mobile robot must trace back the way from A to B
  - There is an obstacle on the way (in form of a polygon)

- Problem: Determining the shortest way from A to B with a detour around the obstacle

- Solution: Ride along the convex cover of
  - Start Point A
  - Points in Obstacle (H)
  - Destination Point B
The Convex Cover (3)

- **Motivation: CAD-Modeling**
  - In tests of insertion or collision of parts, the curves and surfaces are approximated as polygons.
  - In intersection-tests, the convex covers of the point sets to be tested are first checked for intersection.
  - If the intersection of the convex sets is empty, the curves do not intersect as well.
Calculation of the Convex Cover: „Divide and Conquer“ - Algorithm

- Given: A point set in a two-dimensional coordinate system. For simplicity, we will assume that no three points are in one line, and that no two points have the same x-coordinate.

- Goal: Find the convex cover

- Principle of the algorithm:
  - Sort points based on their x-coordinate in ascending order
  - Divide point set in two partial sets L and R, with L containing the $n/2$ left and R the $n/2$ right points
  - Determine recursively the convex cover for every partial set ($CS_L$ or $CS_R$).
  - Merge the convex covers of both partial sets and determine the convex cover for the entire set (the so-called Merge-Step). The recursion breaks when $n < 3$ or $n = 3$. 
Divide and Conquer-Approach: Part 1 (Division)

- Given: n points
- Sort the points according to the x axis in ascending order.
- Divide the point set recursively until one point set contains 3 points max.
- Since the convex cover of 3 points is a triangle, build convex covers for each of the smallest partial sets
- With 15 points, 3 division steps are necessary!
• Determine the convex cover for the partial sets. These are either lines or triangles

• Then, merge the convex polygons recursively
Example: Combination Level 3

- Merge the convex polygons recursively

5.5. Examples for Algorithms

5.5.3. Convex Cover

1. Division

2. Division

3. Division
Example: Combination Level 2

- Merge the convex polygons recursively
Example: Combination Level 1

- Merge the convex polygons recursively

1. Division
Merge-Step: Define the Upper and Lower Tangent

5.5. Examples for Algorithms
5.5.3. Convex Cover

upper Tangent

CS_L

CS_R

lower Tangent
The upper and the lower extreme points divide the border of the CC in two border lines.
• **P2** is the follower of **P1** on the Border of CC exactly if the polar angle belonging to it is **minimal** on P2.
If \( CS_L = \{P_1, \ldots, P_n\} \) and \( CS_R = \{Q_1, \ldots, Q_m\} \).

1. View the upper extreme points \( P_1 \) and \( Q_1 \) and the successors \( P_2 \) and \( Q_2 \) clockwise, and let \( P_1 \) be higher than \( Q_1 \).

2. Define the minimum of the angle associated with \( P_1P_2, P_1Q_1 \) and \( P_1Q_2 \).

3. Cases:
   - \( P_1Q_1 \) is minimal: tangent found, ready
   - \( P_1P_2 \) minimal: replace \( P_1 \) by \( P_2 \) and \( P_2 \) by \( P_3 \) (walk clockwise on the left convex cover)
   - \( P_1Q_2 \) minimal: replace \( Q_1 \) by \( Q_2 \) and \( Q_2 \) by \( Q_3 \) (walk clockwise on the right convex cover)


The case of the lower tangent is symmetrical.
Finding the Upper Tangent (1)

Check $P_1$ and $Q_1$

Angle not minimal

$CS_L$

$P_1$

$P_2$

$Q_1$

$Q_2$

$CS_R$

5.5. Examples for Algorithms

5.5.3. Convex Cover
Finding the Upper Tangent (2)

Check $P_1$ and $Q_1$

$P_1P_2$ minimal $\Rightarrow$ replace $P_1$ by $P_2$

$\text{Angle minimal}$

$$CS_L$$

$$CS_R$$
Finding the Upper Tangent (3)

Check $P_2$ and $Q_1$

$P_2Q_2$ minimal $\rightarrow$ replace $Q_1$ by $Q_2$

Angle minimal
Finding the Upper Tangent (4)

Check $P_2$ and $Q_2$

$P_2Q_2$ minimal $\Rightarrow$ Tangent found
Network Flow (1)

• Motivation: Solution of logistical problems

• Examples:
  - Number of vehicles that can drive through a street net
  - Amount of electric power that can flow through a power lines net
  - Package handling in computer networks: When as many packages as possible (or even real-time) must be transmitted from the source computer to the target computer, the question about the maximal number of transmissible packages arises.
Network Flow (2)

5.5. Examples for Algorithms

5.5.4. Network Flow

- Problem description:

  1. Set of computers (= nodes of a graph), one being the source of a data set to be transmitted, one the target

  2. Two different computers are connected through singular connections (= edges) and can transmit data through this connection
     - Data is transmitted as packages of fixed length
     - Every connection has a capacity (maximal amount of transmitted data packages per unit of time)

  3. The networking is such that different paths are created by which the data can be transported from source to target.

- Question: how many data packages can be transmitted at most per unit of time from source to target?
A network is a directed graph $G = (V, E, c)$ with nodes $q$ (source) and $s$ (target), as well as a capacity function, $c : E \rightarrow \mathbb{R}^+$

$V$ is the node set and $E$ the edge set

A flow for the network is a function $f : E \rightarrow \mathbb{R}$, consisting of:

- Capacity limitation

  \[ \forall u, v \in V : f(u, v) \leq c(u, v) \quad e(u, v) \in E \]

- Flow consistency

  \[ \forall u, v \in V : f(u, v) = -f(v, u) \quad e(u, v) \in E \]

The value of $f(u, v)$ can be positive or negative

The value of a flow is described by

\[ w(f) = \sum_{v \in V} f(q, v) \quad e(q, v) \in E \]

The maximal flow is defined as maximal allowed flow in $G$ regarding $q$ and $s$
• Assume f is an allowable flow for G = (V,E).

- The **rest capacity** of an edge (u,v) is defined as follows:

\[ \text{rest}(u,v) = c(u,v) - f(u,v) \]

- The **rest graph** of G (reg. \( f \)) is defined by \( G_f = (V, E_f), e(u, v) \in E \), while it is true that:

\[ E_f = \{(u,v) \in V \times V : \text{rest}(u,v) > 0\} \quad e(u, v) \in E \]

- Every ordered path from q to s in rest graphs is called an **increasing way**.

- An **intersection** (A, B) of a network is a separation of V in 2 disjunct partial sets A and B, so that q \( \in A \) and s \( \in B \). The **capacity of the intersection** is

\[ c(A, B) = \sum_{u \in A, v \in B} c(u, v), \quad e(u, v) \in E \]
• The following statements are equivalent

  - \( f \) is maximal flow in \( G \)

  - The rest graph of \( f \) contains no increasing (augmenting) path

  - \( w(f') = c(A, B) \) for an intersection \((A, B)\) of \( G \)

if \( f \) is an allowable flow for \( G \).
Network Flow: The Ford-Fulkerson - Algorithm

• Principle of the algorithm:
  The main algorithm of Ford-Fulkerson is based on the Idea of the Max-Flow Min-Cut – theorem, e.g., the maximal flow is found when there are no more augmenting Paths.

• Pseudocode:

```plaintext
for(all (u,v) in E ) f(u,v) = 0; // Initialising

while( there is an increasing path p in Rest Graphs G_f) {
  r = min{rest(u,v) | (u,v) lies in p};
  for(all (u,v) on Path p ){
    f(u,v) = f(u,v) + r;
    f(v,u) = f(v,u) - r;
  }
}
```
Example: Determining the max. Flow in Network

Edges indicate the capacity:
Network Flow: The Ford-Fulkerson – Algorithm: Example (1)

- Step 1: Graph is initialized with flow = 0 edge notation: \( f/c/r \)
  - \( f \): flow
  - \( c \): initial capacity
  - \( r \): rest capacity

\[
\begin{array}{c}
\text{q} \\
\text{c} \\
\text{a} \\
\text{b} \\
\text{d} \\
\text{e} \\
\text{s}
\end{array}
\]

- Edges with capacities:
  - \( 0/3/3 \) from \( q \) to \( a \)
  - \( 0/4/4 \) from \( q \) to \( c \)
  - \( 0/2/2 \) from \( q \) to \( e \)
  - \( 0/2/2 \) from \( a \) to \( c \)
  - \( 0/2/2 \) from \( a \) to \( e \)
  - \( 0/1/1 \) from \( b \) to \( c \)
  - \( 0/2/2 \) from \( b \) to \( d \)
  - \( 0/2/2 \) from \( b \) to \( e \)
  - \( 0/3/3 \) from \( d \) to \( s \)
  - \( 0/3/3 \) from \( e \) to \( s \)
Iteration 1: Choosing the first path: \( q \rightarrow c \rightarrow e \rightarrow s \)

with \( r = \min\{\text{rest}(u,v) \mid (u,v) \text{ lies in } p\} = 2 \).
• Iteration 2: make corresponding edge descriptions and choose the second path $q \rightarrow e \rightarrow d \rightarrow s$ with $r = \min\{\text{rest}(u,v) \mid (u,v) \text{ lies in } p\} = 1$

Note: rest capacity of (c,e) is used up after the first iteration
• Iteration 3: make corresponding edge descriptions and choose third path $q \rightarrow a \rightarrow b \rightarrow s$ with $r = \min\{\text{rest}(u,v) \mid (u,v) \text{ lies in } p\} = 2$

Note:
- Rest capacity of $(e,d)$ is used up after the second iteration
- When making a corresponding edge description $(e,d)$, the back edge $(d,e)$ must be taken in consideration: $f(v,u) = f(v,u) - r$ (see Pseudocode)
• Iteration 4: Make corresponding edge descriptions and choose the fourth path with \( r = \min\{\text{rest}(u,v) \mid (u,v) \text{ lies in } p\} = 1 \)

\[ q \rightarrow a \rightarrow d \rightarrow e \rightarrow s \]

Note: Rest capacities of (e,d) and (b,s) are used up after the third iteration.
• Iteration 5: Make corresponding edge descriptions and choose the fifth path with \( r = \min\{\text{rest}(u,v) \mid (u,v) \text{ lies in } p\} = 2 \)

\[ q \rightarrow c \rightarrow d \rightarrow s \]

Note:
- Rest capacities of (a,d) and (e,s) are used up after the fourth iteration
- When making a corresponding edge description for the border (d,e), the back edge must be taken into consideration
• Result:
  - No other paths are possible
  - The calculation of the maximal flow is thus ended!
  - Note:
    - Which path is chosen in every iteration step remains open!
    - The number of iterations depends on the choice of paths
Problem

Given: The following network with the source node $q$, the target node $s$, the further nodes $v_1$, ..., $v_6$ and the inscriptions on the edges $f/c/r$ with $f =$ flow, $c =$ initial capacity and $r =$ rest capacity.

a.) Use the Ford-Fulkerson algorithm on the above mentioned network. List in a table all possible paths for every step through the rest capacities network, as well as their length and capacity. Choose as the next path the shortest, and - if equal length appears - the one with highest capacity.
Problem: Initial Determination of a Flow Network (determine Flow with 0)

\[ f: \text{Flow} \]
\[ c: \text{Initial Cap.} \]
\[ r: \text{Rest Capacity} \]

![Flow Network Diagram](image)

5.5. Examples for Algorithms
5.5.4. Network Flow
Problem (Step 1)

- Define all possible flows from source to target
- Choose flow with minimal length. If there is more than one flow with minimal length, choose the one with max. flow
  - Choose q, v₁, v₄, s
  - Flow Increase = 3

Path | f = 0 | Length | Flow
--- | --- | --- | ---
q, v₁, v₄, s | 4 | 3 |
q, v₁, v₄, v₅, v₆, s | 6 | 2 |
q, v₁, v₅, v₆, s | 5 | 2 |
q, v₁, v₂, v₅, v₆, s | 6 | 2 |
q, v₁, v₂, v₃, v₅, v₆, s | 7 | 2 |
q, v₁, v₂, v₃, v₆, s | 6 | 2 |
q, v₂, v₅, v₆, s | 5 | 2 |
q, v₂, v₃, v₅, v₆, s | 6 | 2 |
q, v₂, v₃, v₆, s | 5 | 4 |
q, v₃, v₅, v₆, s | 5 | 2 |
q, v₃, v₆, s | 4 | 2 |
Determination of the Flow Network after Step 1

5.5. Examples for Algorithms
5.5.4. Network Flow
Problem (Step 2)

- Update the flows
- Choose the flow with min. length
  - Choose q, v₃, v₆, s
  - Flow increase = 2

<table>
<thead>
<tr>
<th>Path</th>
<th>f</th>
<th>Length</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>q, v₁, v₄, s</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₄, v₅, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₅, v₆, s</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₅, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₃, v₅, v₆, s</td>
<td>/</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₃, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₂, v₅, v₆, s</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v₂, v₃, v₅, v₆, s</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v₂, v₃, v₆, s</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>q, v₃, v₅, v₆, s</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v₃, v₆, s</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Determination of the Flow Network after Step 2

5.5. Examples for Algorithms

5.5.4. Network Flow

f: Flow
c: Initial Cap.
r: Rest Capacity
Problem (Step 3)

- Update the flows
- Choose flow with min. length
  - Choose q, v_2, v_3, v_6, s
  - The flow q, v_2, v_5, v_6, s may also be possible
  - flow increase = 2

<table>
<thead>
<tr>
<th>Path</th>
<th>f</th>
<th>Length</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>q, v_1, v_4, s</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>q, v_1, v_4, v_5, v_6, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v_1, v_5, v_6, s</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v_1, v_2, v_5, v_6, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v_1, v_2, v_3, v_5, v_6, s</td>
<td>/</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>q, v_1, v_2, v_3, v_6, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v_2, v_5, v_6, s</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v_2, v_3, v_5, v_6, s</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v_2, v_3, v_6, s</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>q, v_3, v_5, v_6, s</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v_3, v_6, s</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Determination of the Flow Network after Step 3

5.5. Examples for Algorithms

5.5.4. Network Flow

f: Flow  c: Initial Cap.  r: Rest Capacity
Problem (Step 4)

- Update the flows
- choose flow with min. length
  - Choose q, v₂, v₅, v₆, s
  - Flow increase f=1

<table>
<thead>
<tr>
<th>Path</th>
<th>f</th>
<th>Length</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>q, v₁, v₄, s</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₄, v₅, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₅, v₆, s</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₅, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₃, v₅, v₆, s</td>
<td>/</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>q, v₁, v₂, v₃, v₆, s</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>q, v₂, v₅, v₆, s</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>q, v₂, v₃, v₅, v₆, s</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>q, v₂, v₃, v₆, s</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v₃, v₅, v₆, s</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>q, v₃, v₆, s</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Determination of the Flow Network after Step 4

\[ \begin{align*}
&v_1 \quad 3/4/1 \\
&v_2 \quad 1/3/2 \\
&v_3 \quad 4/4/0 \\
&v_4 \quad 0/4/4 \\
&v_5 \quad 3/4/1 \\
&v_6 \quad 1/2/1 \\
\end{align*} \]

\[ \begin{align*}
&f/c/r \quad v_q \\
&3/3/0 \quad v_1 & 0/2/2 & v_4 \\
&3/6/3 \quad v_2 & 0/3/3 & v_5 \\
&2/5/3 \quad v_3 & 1/2/1 & v_6 \\
&2/2/0 \quad v_q & 2/5/3 & v_s \\
\end{align*} \]

- \( f \): Flow
- \( c \): Initial Cap.
- \( r \): Rest Capacity
Problem (after Step 4)

- No further flow increase is possible $\Rightarrow$ Ford-Fulkerson algorithm terminates

<table>
<thead>
<tr>
<th>Path</th>
<th>f</th>
<th>Length</th>
<th>Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q, v_1, v_4, s$</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_1, v_4, v_5, v_6, s$</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_1, v_5, v_6, s$</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_1, v_2, v_5, v_6, s$</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_1, v_2, v_3, v_5, v_6, s$</td>
<td>/</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_1, v_2, v_3, v_6, s$</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_2, v_5, v_6, s$</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_2, v_3, v_5, v_6, s$</td>
<td>/</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_2, v_3, v_6, s$</td>
<td>/</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_3, v_5, v_6, s$</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q, v_3, v_6, s$</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
b) After solving the previous partial problem, any intersection of graphs can be done. The resulting maximal flow equals 8.

\[ f_{\text{max}} = 3 + 0 + 1 + 0 + 4 = 8 \]
Literature


[Cott01]  Vorlesungen TU Cottbus (2001)


Java Programm mit konvexer Hülle und Voronoi Diagrammen:

http://www.pi6.fernuni-hagen.de/GeomLab/VoroGlide/


Weitere Links:

http://www-ti.informatik.tu-cottbus.de/HTML/sortieren.html

http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/

http://www.inf.fh-flensburg.de/lang/algorithmen/geo/convex.htm