1. Felder (Arrays)

Feldtypen, Sieb des Eratosthenes,
Speicherlayout, Iteration, Vektoren, Zeichen,
Texte, Caesar-Code, String Matching,
Mehrdimensionale Felder, Vektoren von Vektoren,
Kürzeste Wege

Felder: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```

Felder: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```

- Oft muss man aber über *Daten* iterieren (Beispiel: Finde ein Kino in Zürich, das heute „L'Expérience C++" zeigt)

Felder: Motivation

- Wir können jetzt über Zahlen iterieren

```
for (int i=0; i<n ; ++i) ...
```

- Oft muss man aber über *Daten* iterieren (Beispiel: Finde ein Kino in Zürich, das heute „L'Expérience C++" zeigt)
- Felder dienen zum Speichern *gleichartiger* Daten (Beispiel: Spielpläne aller Zürcher Kinos)

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

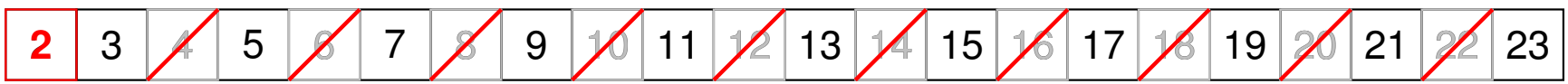
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Streiche alle echten Vielfachen von 2 ...

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

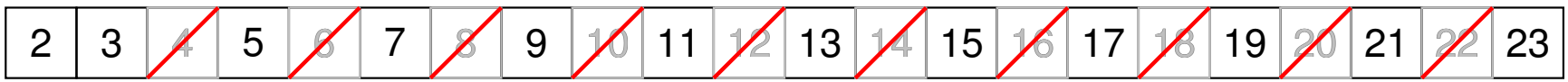


Streiche alle echten Vielfachen von 2 ...

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

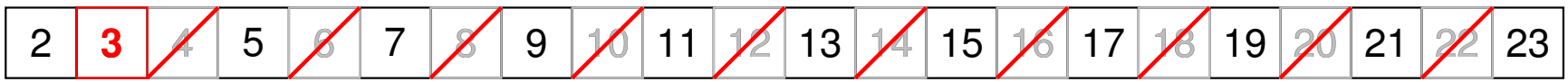


... und gehe zur nächsten Zahl

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen



Streiche alle echten Vielfachen von 3 ...

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

Streiche alle echten Vielfachen von 3 ...

Felder: erste Anwendung

Das Sieb des Eratosthenes

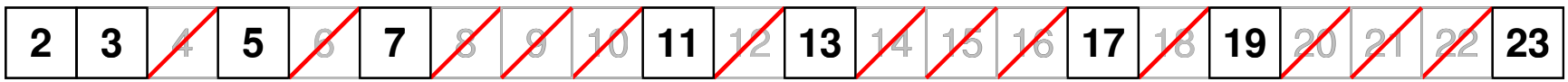
- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

... und gehe zur nächsten Zahl

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

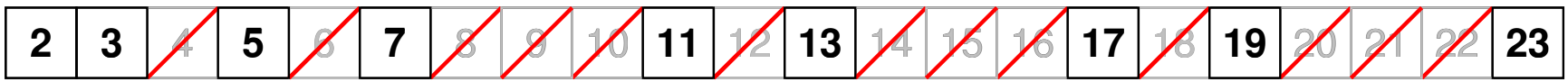


Am Ende des Streichungsprozesses bleiben nur die Primzahlen übrig.

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen

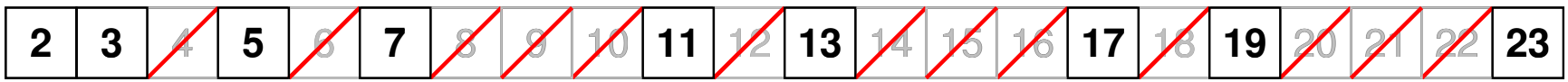


- Frage: wie streichen wir Zahlen aus ??

Felder: erste Anwendung

Das Sieb des Eratosthenes

- berechnet alle Primzahlen $< n$
- Methode: Ausstreichen der Nicht-Primzahlen



- Frage: wie streichen wir Zahlen aus ??
- Antwort: mit einem *Feld.*

Implementierung des Sieb des Eratosthenes

```
int main()
{
  const unsigned int n = 1000;
  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool crossed_out[n];
  for (unsigned int i = 0; i < n; ++i)
    crossed_out[i] = false;

  // computation and output
  std::cout << "Prime numbers in 2,..., " << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) {
      // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";

  return 0;
}
```

Implementierung des Sieb des Eratosthenes

Berechnet alle Primzahlen < 1000

```
int main()
{
  const unsigned int n = 1000;
  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool crossed_out[n];
  for (unsigned int i = 0; i < n; ++i)
    crossed_out[i] = false;

  // computation and output
  std::cout << "Prime numbers in 2,..., " << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) {
      // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";

  return 0;
}
```

Implementierung des Sieb des Eratosthenes

```
int main()
{
  const unsigned int n = 1000;
  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool crossed_out[n];
  for (unsigned int i = 0; i < n; ++i)
    crossed_out[i] = false;

  // computation and output
  std::cout << "Prime numbers in 2,..., " << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) {
      // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";

  return 0;
}
```

`n` ist eine Konstante.

Implementierung des Sieb des Eratosthenes

```
int main()
{
  const unsigned int n = 1000;
  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool crossed_out[n];
  for (unsigned int i = 0; i < n; ++i)
    crossed_out[i] = false;

  // computation and output
  std::cout << "Prime numbers in 2,..., " << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) {
      // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";

  return 0;
}
```

`n` ist eine Konstante.

Feld: `crossed_out[i]` gibt an, ob i schon ausgestrichen wurde.

Implementierung des Sieb des Eratosthenes

```
int main()
{
  const unsigned int n = 1000;
  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool crossed_out[n];
  for (unsigned int i = 0; i < n; ++i)
    crossed_out[i] = false;

  // computation and output
  std::cout << "Prime numbers in 2,..., " << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) {
      // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";

  return 0;
}
```

`n` ist eine Konstante.

Feld: `crossed_out[i]` gibt an, ob i schon ausgestrichen wurde.

Das Sieb: gehe zur jeweils nächsten nicht-gestrichenen Zahl i (diese ist Primzahl), gib sie aus und streiche alle echten Vielfachen von i aus.

Felder: Definition

Deklaration einer Feldvariablen (array):

T a [expr]

Felder: Definition

Deklaration einer Feldvariablen (array):

T a [expr]

Zugrundeliegender Typ

Variable des Feld-Typs

Konstanter ganzzahliger Ausdruck
Wert gibt die Länge des Feldes an

Felder: Definition

Deklaration einer Feldvariablen (array):

T a [expr]

Zugrundeliegender Typ

Variable des Feld-Typs

Konstanter ganzzahliger Ausdruck
Wert gibt die Länge des Feldes an

Typ von **`a`**: „T[k]"
Wertebereich von **`a`**: T^k

Wert ist bei Kompilierung bekannt.
Z.B. Literal, Konstante

Felder: Definition

Deklaration einer Feldvariablen (array):

T a [expr]

Zugrundeliegender Typ

Variable des Feld-Typs

Konstanter ganzzahliger Ausdruck
Wert gibt die Länge des Feldes an

Typ von **`a`**: „T[k]"
Wertebereich von **`a`**: T^k

Wert ist bei Kompilierung bekannt.
Z.B. Literal, Konstante

Beispiel: **`bool crossed_out[n]`**

Feld-Initialisierung

- `int a[5];`

  Die 5 Elemente von `a` bleiben uninitialisiert (können später Werte zugewiesen bekommen)

Feld-Initialisierung

- `int a[5];`

  Die 5 Elemente von `a` bleiben uninitialisiert (können später Werte zugewiesen bekommen)

- `int a[5] = {4, 3, 5, 2, 1};`

  Die 5 Elemente von `a` werden mit einer *Initialisierungsliste* initialisiert.

Feld-Initialisierung

- `int a[5];`

  Die 5 Elemente von `a` bleiben uninitialisiert (können später Werte zugewiesen bekommen)

- `int a[5] = {4, 3, 5, 2, 1};`

  Die 5 Elemente von `a` werden mit einer *Initialisierungsliste* initialisiert.

- `int a[] = {4, 3, 5, 2, 1};`

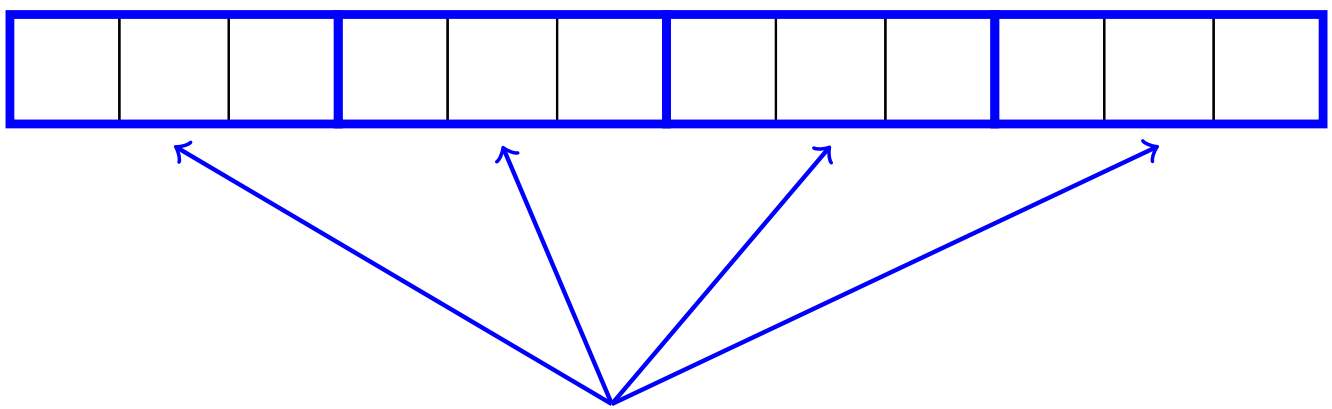
  Auch ok: Länge wird vom Compiler deduziert

Speicherlayout eines Feldes

- Ein Feld belegt einen *zusammenhängenden* Speicherbereich

Speicherlayout eines Feldes

- Ein Feld belegt einen *zusammenhängenden* Speicherbereich

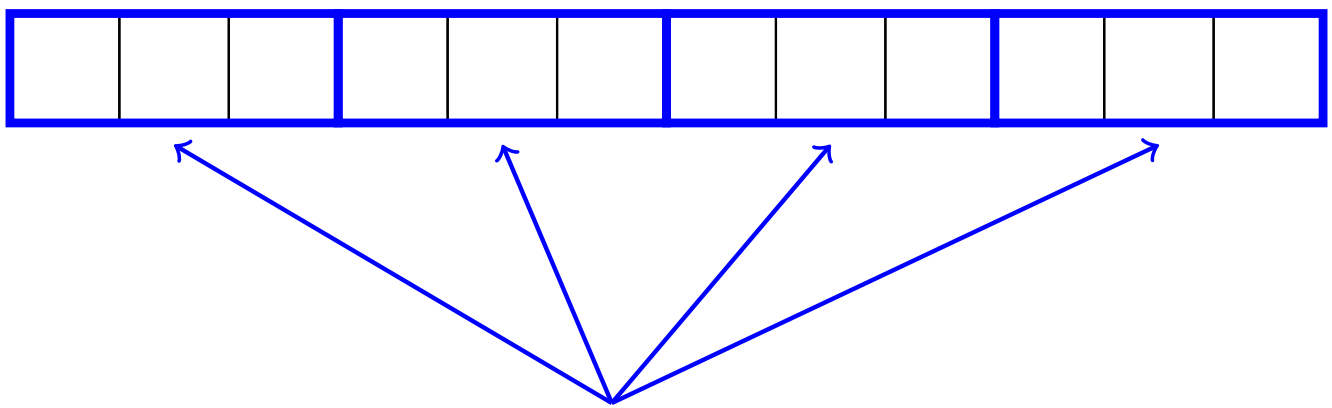


Speicherzellen für jeweils einen Wert vom Typ `T`

Speicherlayout eines Feldes

- Ein Feld belegt einen *zusammenhängenden* Speicherbereich

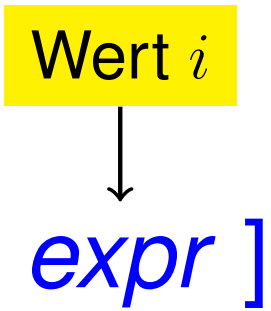
Beispiel: ein Feld mit 4 Elementen.



Speicherzellen für jeweils einen Wert vom Typ `T`

Wahlfreier Zugriff (Random Access)

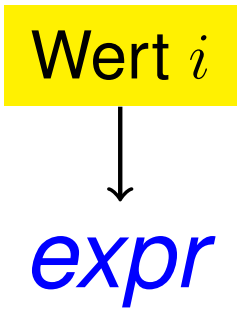
Der L-Wert



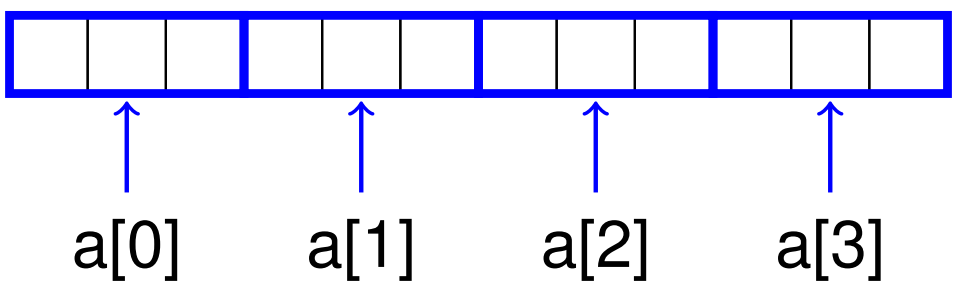
hat Typ *T* und bezieht sich auf das i-te Element des Feldes *a* (Zählung ab 0!)

Wahlfreier Zugriff (Random Access)

Der L-Wert



hat Typ *T* und bezieht sich auf das i-te Element des Feldes *a* (Zählung ab 0!)



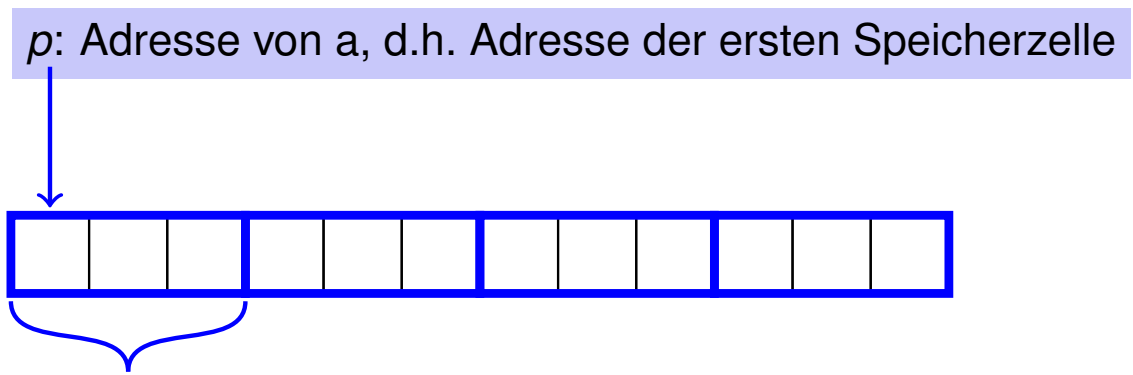
Wahlfreier Zugriff (Random Access)

a [*expr*]

Der Wert i von *expr* heisst *Feldindex*.
`[]`: Subskript-Operator

Wahlfreier Zugriff (Random Access)

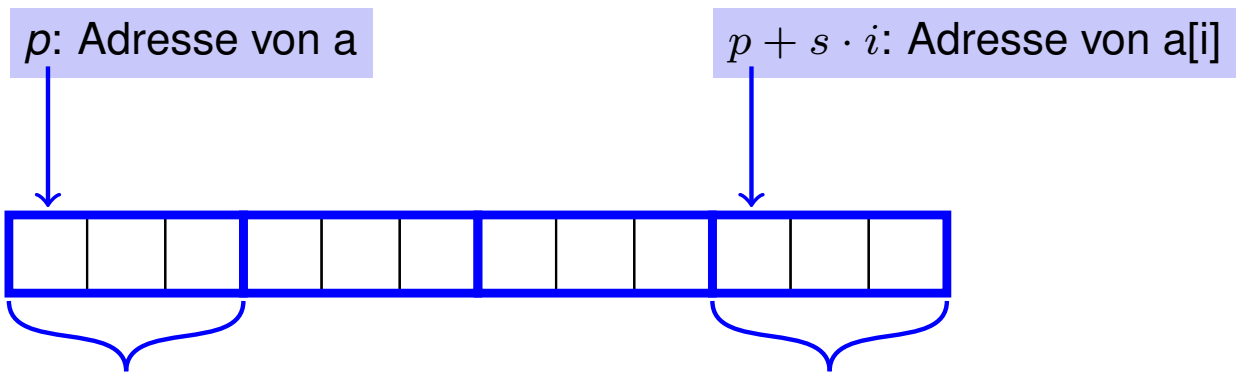
- Wahlfreier Zugriff ist sehr effizient:



s: Speicherbedarf von T (in Zellen)

Wahlfreier Zugriff (Random Access)

- Wahlfreier Zugriff ist sehr effizient:



Felder sind primitiv

- Der Zugriff auf Elemente ausserhalb der gültigen Grenzen eines Feldes führt zu undefiniertem Verhalten.

```
int arr[10];
for (int i=0; i<=10; ++i)
  arr[i] = 30;
```

Felder sind primitiv

- Der Zugriff auf Elemente ausserhalb der gültigen Grenzen eines Feldes führt zu undefiniertem Verhalten.

```
int arr[10];
for (int i=0; i<=10; ++i)
  arr[i] = 30;// Fehler: Zugriff auf arr[10]!
```

Felder sind primitiv

- Der Zugriff auf Elemente ausserhalb der gültigen Grenzen eines Feldes führt zu undefiniertem Verhalten.

```
int arr[10];
for (int i=0; i<=10; ++i)
  arr[i] = 30;// Fehler: Zugriff auf arr[10]!
```

Prüfung der Feldgrenzen

In Abwesenheit spezieller Compiler- oder Laufzeitunterstützung ist es die alleinige *Verantwortung des Programmierers*, die Gültigkeit aller Elementzugriffe zu prüfen.

Felder sind primitiv (II)

- Man kann Felder nicht wie bei anderen Typen initialisieren und zuweisen:

```
int a[5] = {4,3,5,2,1};
int b[5];
b = a;                  // Fehler !
int c[5] = a;           // Fehler !
```

Felder sind primitiv (II)

- Man kann Felder nicht wie bei anderen Typen initialisieren und zuweisen:

```
int a[5] = {4,3,5,2,1};
int b[5];
b = a;                  // Fehler !
int c[5] = a;           // Fehler !
```

Warum?

Felder sind primitiv

- Felder sind „Erblast” der Sprache C und aus heutiger Sicht primitiv.

Felder sind primitiv

- Felder sind „Erblast” der Sprache C und aus heutiger Sicht primitiv.
- In C sind Felder sehr maschinennah und effizient, bieten aber keinen „Luxus” wie eingebautes Initialisieren und Kopieren.

Felder sind primitiv

- Felder sind „Erblast” der Sprache C und aus heutiger Sicht primitiv.
- In C sind Felder sehr maschinennah und effizient, bieten aber keinen „Luxus” wie eingebautes Initialisieren und Kopieren.
- Fehlendes Prüfen der Feldgrenzen hat weitreichende Konsequenzen. Code mit unerlaubten aber möglichen Index-Zugriffen wurde von Schadsoftware schon (viel zu) oft ausgenutzt.

Felder sind primitiv

- Felder sind „Erblast” der Sprache C und aus heutiger Sicht primitiv.
- In C sind Felder sehr maschinennah und effizient, bieten aber keinen „Luxus” wie eingebautes Initialisieren und Kopieren.
- Fehlendes Prüfen der Feldgrenzen hat weitreichende Konsequenzen. Code mit unerlaubten aber möglichen Index-Zugriffen wurde von Schadsoftware schon (viel zu) oft ausgenutzt.
- Die Standard-Bibliothek bietet komfortable Alternativen (mehr dazu später)

Vektoren

- Offensichtlicher Nachteil statischer Felder: *konstante Feldlänge*

```
const unsigned int n = 1000;
bool crossed_out[n];
...
```

Vektoren

- Offensichtlicher Nachteil statischer Felder: *konstante Feldlänge*

```
const unsigned int n = 1000;
bool crossed_out[n];
...
```

- Abhilfe: Verwendung des Typs **`Vector`** aus der Standardbibliothek

```
#include <vector>
...
std::vector<bool> crossed_out (n, false);
```

Initialisierung mit n Elementen Initialwert **`false`**.

Elementtyp, in spitzen Klammern

Sieb des Eratosthenes mit Vektoren

```
#include <iostream>
#include <vector>  // standard containers with array functionality
int main() {
  // input
  std::cout << "Compute prime numbers in 2,...,n-1 for n =? ";
  unsigned int n;
  std::cin >> n;

  // definition and initialization: provides us with Booleans
  // crossed_out[0],..., crossed_out[n-1], initialized to false
  std::vector<bool> crossed_out (n, false);

  // computation and output
  std::cout << "Prime numbers in 2,...," << n-1 << ":\n";
  for (unsigned int i = 2; i < n; ++i)
    if (!crossed_out[i]) { // i is prime
      std::cout << i << " ";
      // cross out all proper multiples of i
      for (unsigned int m = 2*i; m < n; m += i)
        crossed_out[m] = true;
    }
  std::cout << "\n";
  return 0;
}
```

Zeichen und Texte

- Texte haben wir schon gesehen:

```
std::cout << "Prime numbers in {2,...,999}:\n";
```

Zeichen und Texte

- Texte haben wir schon gesehen:

```
std::cout << "Prime numbers in {2,...,999}:\n";
```

String-Literal

Zeichen und Texte

- Texte haben wir schon gesehen:

```
std::cout << "Prime numbers in {2,...,999}:\n";
```

String-Literal

- Können wir auch „richtig” mit Texten arbeiten?

Zeichen und Texte

- Texte haben wir schon gesehen:

```
std::cout << "Prime numbers in {2,...,999}:\n";
```

String-Literal

- Können wir auch „richtig” mit Texten arbeiten?
  Ja:

Zeichen:	Wert des fundamentalen Typs **`char`**
Text:	Feld mit zugrundeliegendem Typ **`char`**

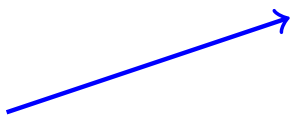
Der Typ `char` (“character”)

- repräsentiert druckbare Zeichen (z.B. `'a'`) und *Steuerzeichen* (z.B. `'\n'`)

Der Typ `char` (“character”)

- repräsentiert druckbare Zeichen (z.B. `'a'`) und *Steuerzeichen* (z.B. `'\n'`)

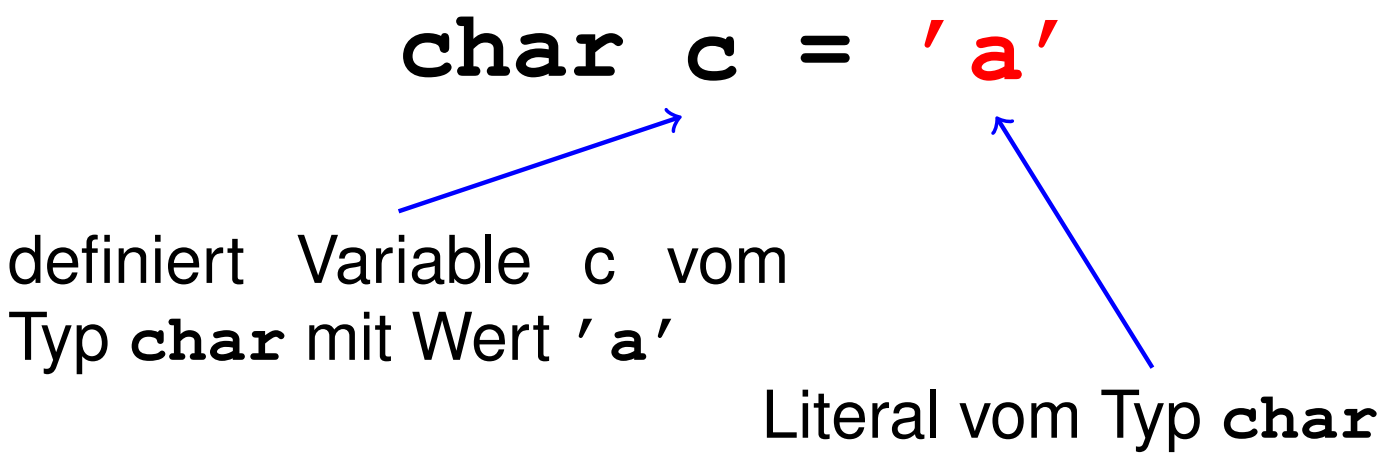
```
char c = 'a'
```



definiert Variable c vom Typ **`char`** mit Wert **`'a'`**

Der Typ `char` (“character”)

- repräsentiert druckbare Zeichen (z.B. `'a'`) und *Steuerzeichen* (z.B. `'\n'`)



Der Typ `char` (“character”)

- ist formal ein ganzzahliger Typ
  - Werte konvertierbar nach **`int`** / **`unsigned int`**
  - Alle arithmetischen Operatoren verfügbar (Nutzen zweifelhaft: was ist **`'a'/'b'`** ?)

Der Typ `char` (“character”)

- ist formal ein ganzzahliger Typ
  - Werte konvertierbar nach **`int`** / **`unsigned int`**
  - Alle arithmetischen Operatoren verfügbar (Nutzen zweifelhaft: was ist **`'a'/'b'`** ?)
  - Werte belegen meistens 8 Bit

Wertebereich:
$\{-128, \ldots, 127\}$ oder $\{0, \ldots, 255\}$

Der ASCII-Code

- definiert konkrete Konversionsregeln
  `char` $\longrightarrow$ **`int / unsigned int`**
- wird von fast allen Plattformen benutzt

Der ASCII-Code

- definiert konkrete Konversionsregeln
  `char` $\longrightarrow$ **`int / unsigned int`**
- wird von fast allen Plattformen benutzt

Zeichen $\longrightarrow \{0, \ldots, 127\}$

`'A', 'B', ... , 'Z'` $\longrightarrow 65, 66, ..., 90$

`'a', 'b', ... , 'z'` $\longrightarrow 97, 98, ..., 122$

`'0', '1', ... , '9'` $\longrightarrow 48, 49, ..., 57$

Der ASCII-Code

- definiert konkrete Konversionsregeln
  `char` $\longrightarrow$ **`int / unsigned int`**
- wird von fast allen Plattformen benutzt

Zeichen $\longrightarrow \{0, \ldots, 127\}$

`'A', 'B', ... , 'Z'` $\longrightarrow 65, 66, ..., 90$

`'a', 'b', ... , 'z'` $\longrightarrow 97, 98, ..., 122$

`'0', '1', ... , '9'` $\longrightarrow 48, 49, ..., 57$

```
for (char c = 'a'; c <= 'z'; ++c)
    std::cout << c;
```

Ausgabe: **`abcdefghijklmnopqrstuvwxyz`**

Anwendung: Caesar-Code

Ersetze jedes Zeichen in einem Text durch das jeweils nächste Zeichen:

- Wir behandeln nur die 96 druckbaren ASCII-Zeichen ’ ’ (32) bis ’$\sim$’ (126).
- Ersetze ’ ’ (32) durch ’!’ (33) usw.
- Ersetze ’$\sim$’ (126) durch ’ ’ (32)

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next) {
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127) {
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next) {
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127) {
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Leerzeichen und Zeilenumbrüche sollen *nicht* ignoriert werden

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next){
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127) {
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Leerzeichen und Zeilenumbrüche sollen *nicht* ignoriert werden

Konversion nach **`bool`**: liefert *false* genau dann, wenn die Eingabe leer ist.

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next){
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127){
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Leerzeichen und Zeilenumbrüche sollen *nicht* ignoriert werden

Konversion nach **bool**: liefert *false* genau dann, wenn die Eingabe leer ist.

Behandle nur die druckbaren Zeichen.

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next){
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127){
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Leerzeichen und Zeilenumbrüche sollen *nicht* ignoriert werden

Konversion nach **bool**: liefert *false* genau dann, wenn die Eingabe leer ist.

Behandle nur die druckbaren Zeichen.

Eigentliche Ersetzung.

Anwendung: Caesar-Code

```
// Program: caesar_encrypt.cpp
// encrypts a text by applying a cyclic shift of one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next){
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127){
      if (next < 126)
         ++next;
      else
         next = 32;
    }
    std::cout << next;
  }
  return 0;
}
```

Leerzeichen und Zeilenumbrüche sollen *nicht* ignoriert werden

Konversion nach **`bool`**: liefert *false* genau dann, wenn die Eingabe leer ist.

Behandle nur die druckbaren Zeichen.

Eigentliche Ersetzung.

Eingabe nicht von der Tastatur, sondern aus angegebener Datei mit Umlenkung der Eingabe

```
./caesar_encrypt < caesar_ecnrypt.cpp
```

./caesar_encrypt < caesar_ecnrypt.cpp

```
00!Qsphsbn;!dbftbs`fodszqu/dqq
00!fodszqut!b!ufyu!cz!bqqmzjoh!b!dzdmjd!tijgu!pg!pof!

$jodmvef=jptusfbn?
$jodmvef=jpt?!00!gps!tue;;optljqxt

jou!nbjo!)*
|
!!tue;;djo!??!tue;;optljqxt<!00!epo(u!tljq!xijuftqbdft"

!!00!fodszqujpo!mppq
!!dibs!ofyu<!
!!xijmf!)tue;;djo!??!ofyu*!|
!!!!00!tijgu!uif!:6!qsjoubcmf!dibsbdufst!pomz-!bttvnjoh!BTDJJ
!!!!jg!)ofyu!?!42!”!ofyu!=!238*!|
!!!!!!jg!)ofyu!=!237*!
„ofyu<
!!!!!!fmtf!
ofyu!>!43<
!!!!~
!!!!tue;;dpvu!==!ofyu<
!!~
!!sfuvso!1<
~
```

Caesar-Code: Entschlüsselung

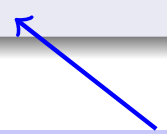
```
// Program: caesar_decrypt.cpp
// decrypts a text by applying a cyclic shift of minus one

#include<iostream>
#include<ios> // for std::noskipws

int main ()
{
  std::cin >> std::noskipws; // don't skip whitespaces!

  // encryption loop
  char next;
  while (std::cin >> next) {
    // shift the 95 printable characters only, assuming ASCII
    if (next > 31 && next < 127) {
      if (next > 32)
         --next;
      else
         next = 126;
    }
    std::cout << next;
  }
  return 0;
}
```

Ersetzung: Verschiebung nach links

Texte

- Texte können mit dem Typ **`std::string`** aus der Standardbibliothek repräsentiert werden.
- Ein String ist im Prinzip ein Feld mit zugrundeliegendem Typ **`char`**, plus Zusatzfunktionalität
- Benutzung benötigt **`#include <string>`**
- Beispiel: Initialisierung mit String-Literal

```
std::string text = "bool";
```



definiert ein String der Länge 4

Typ `std::string` ist nicht primitiv

- kennt seine Länge

```
text.length()
```

gibt Länge als `int` zurück (Aufruf einer Mitglieds-Funktion; später in der Vorlesung)

- kann mit variabler Länge initialisiert werden

```
std::string text (n, 'a')
```

`text` wird mit n `'a'`s gefüllt

- „versteht" Vergleiche

```
if (text1 == text2) ...
```

`true` wenn `text1` und `text2` übereinstimmen

Anwendung: String Matching

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

- „Trivialer” Algorithmus:

  Gallia est omnis divisa in partes tres

Anwendung: String Matching

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

- „Trivialer” Algorithmus:

  Gallia est omnis divisa in partes tres
  $\neq$
  visa

Anwendung: String Matching

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

- „Trivialer” Algorithmus:

  G**alli**a est omnis divisa in partes tres
  $\neq$
  visa

Anwendung: String Matching

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

- „Trivialer” Algorithmus:

  Ga**llia** est omnis divisa in partes tres
  $\neq$
  visa

Anwendung: String Matching

Finde das erste (oder alle) Vorkommen eines Musters (meist kurz) in einem gegebenen Text (meist lang)!

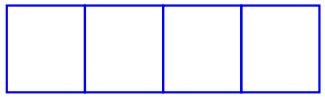
- „Trivialer” Algorithmus:

  Gallia est omnis di**visa** in partes tres
  =
  visa
  gefunden!

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

Inhalt von **window** wird nach links verschoben, dann wird next angehängt

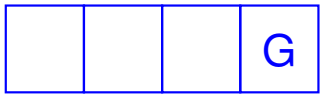


next='G'

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

Inhalt von **window** wird nach links verschoben, dann wird next angehängt



next='a'

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

Inhalt von **window** wird nach links verschoben, dann wird next angehängt

next='l'

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

Inhalt von **window** wird nach links verschoben, dann wird next angehängt

next='l'

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

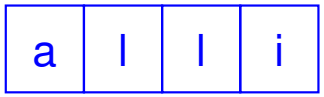
Inhalt von **window** wird nach links verschoben, dann wird next angehängt

next='i'

Anwendung: String Matching

```
// Program: string_matching.cpp
// find the first occurrence of a fixed pattern within the
// input text, and output the text read so far

#include<iostream>
#include<string>    // strings are sequences of characters
                    // with array functionality
#include<ios>       // for std::noskipws

// POST: returns window, after removing the first character and
//       appending next as a new last character
std::string advance (std::string window, const char next)
{
  int last_index = window.length()-1;
  for (int i=0; i<last_index; ++i)
     window[i] = window[i+1];
  window[last_index] = next;
  return window;
}
```

Inhalt von **window** wird nach links verschoben, dann wird next angehängt



next='a'

Anwendung: String Matching

```
int main ()
{
  // search pattern
  const std::string pattern = "bool";

  // "empty" text window of pattern length
  std::string window (pattern.length(), '\0');

  // read characterwise from standard input until
  // search pattern was found
  std::cin >> std::noskipws; // don't skip whitespaces!
  char c;                    // next character to be read
  while (std::cin >> c) {
    std::cout << c;
    window = advance (window, c);
    if (window == pattern) break;
  }
  std::cout << "\n";

  return 0;
}
```

Anwendung: String Matching

Pattern als konstanter String

```
int main ()
{
  // search pattern
  const std::string pattern = "bool";

  // "empty" text window of pattern length
  std::string window (pattern.length(), '\0');

  // read characterwise from standard input until
  // search pattern was found
  std::cin >> std::noskipws; // don't skip whitespaces!
  char c;                    // next character to be read
  while (std::cin >> c) {
    std::cout << c;
    window = advance (window, c);
    if (window == pattern) break;
  }
  std::cout << "\n";

  return 0;
}
```

Anwendung: String Matching

```
int main ()
{
  // search pattern
  const std::string pattern = "bool";

  // "empty" text window of pattern length
  std::string window (pattern.length(), '\0');

  // read characterwise from standard input until
  // search pattern was found
  std::cin >> std::noskipws; // don't skip whitespaces!
  char c;                    // next character to be read
  while (std::cin >> c) {
    std::cout << c;
    window = advance (window, c);
    if (window == pattern) break;
  }
  std::cout << "\n";

  return 0;
}
```

Pattern als konstanter String

Gleitendes Fenster der gleichen Länge wie **`pattern`**

Anwendung: String Matching

```
int main ()
{
  // search pattern
  const std::string pattern = "bool";

  // "empty" text window of pattern length
  std::string window (pattern.length(), '\0');

  // read characterwise from standard input until
  // search pattern was found
  std::cin >> std::noskipws; // don't skip whitespaces!
  char c;                    // next character to be read
  while (std::cin >> c) {
    std::cout << c;
    window = advance (window, c);
    if (window == pattern) break;
  }
  std::cout << "\n";

  return 0;
}
```

Pattern als konstanter String

Gleitendes Fenster der gleichen Länge wie **pattern**

Eingabe wird Zeichen für Zeichen (incl. Whitespaces) verarbeitet.

Anwendung: String Matching

```
int main ()
{
  // search pattern
  const std::string pattern = "bool";

  // "empty" text window of pattern length
  std::string window (pattern.length(), '\0');

  // read characterwise from standard input until
  // search pattern was found
  std::cin >> std::noskipws; // don't skip whitespaces!
  char c;                    // next character to be read
  while (std::cin >> c) {
    std::cout << c;
    window = advance (window, c);
    if (window == pattern) break;
  }
  std::cout << "\n";

  return 0;
}
```

Pattern als konstanter String

Gleitendes Fenster der gleichen Länge wie **`pattern`**

Eingabe wird Zeichen für Zeichen (incl. Whitespaces) verarbeitet.

Variablen vom Typ **`std::string`** können verglichen werden. Rückgabe **`true`** dann wenn alle Zeichen übereinstimmen.

Anwendung: String Matching

- Aufruf des Programms mit

  ```
  ./string_matching < eratosthenes.cpp
  ```

- Variante mit Eingabe des Musters

  ```
  ./string_matching2 bool < eratosthenes.cpp
  ```

Anwendung: String Matching

- Aufruf des Programms mit

  `./string_matching < eratosthenes.cpp`

- Variante mit Eingabe des Musters

  `./string_matching2 bool < eratosthenes.cpp`

- Ausgabe

```
// Program: eratosthenes.cpp
// Calculate prime numbers in 2,...,n-1 using
// Eratosthenes' sieve.

#include <iostream>

int main()
{
  const unsigned int n = 1000;

  // definition and initialization: provides us with
  // Booleans crossed_out[0],..., crossed_out[n-1]
  bool
```

Anwendung: String Matching

- Aufruf des Programms mit

  Umleitung der Ausgabe in eine Datei ↓

  ```
  ./string_matching < eratosthenes.cpp > match.out
  ```

- Variante mit Eingabe des Musters

  ```
  ./string_matching2 bool < eratosthenes.cpp
  ```

Anwendung: String Matching

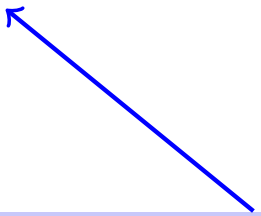
- Aufruf des Programms mit

  ```
  ./string_matching < eratosthenes.cpp > match.out
  ```

- Variante mit Eingabe des Musters

  ```
  ./string_matching2 bool < eratosthenes.cpp
  ```

- Der triviale Algorithmus ist meistens schnell, aber nicht immer (Übung)
- *Knuth-Morris-Pratt*-Algorithmus ist immer schnell



Nutzt Information aus den vorhergehenden Vergleichen

Mehrdimensionale Felder

- sind Felder von Feldern
- dienen zum Speichern von *Tabellen, Matrizen,...*

Mehrdimensionale Felder

- sind Felder von Feldern

```
int a[2][3]
```

`a` hat zwei Elemente, und jedes von ihnen ist ein Feld der Länge 3 mit zugrundeliegendem Typ `int`

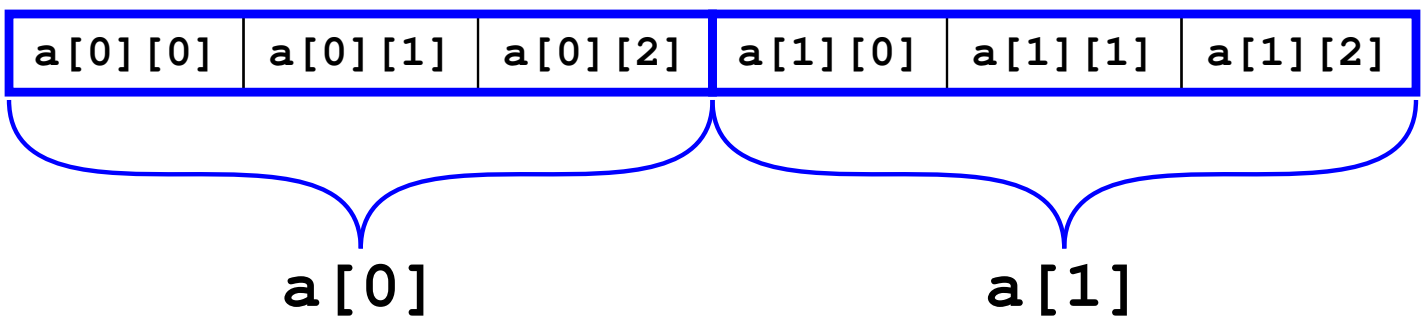
Mehrdimensionale Felder

- sind Felder von Feldern

```
int a[2][3]
```

`a` hat zwei Elemente, und jedes von ihnen ist ein Feld der Länge 3 mit zugrundeliegendem Typ `int`

Im Speicher: flach



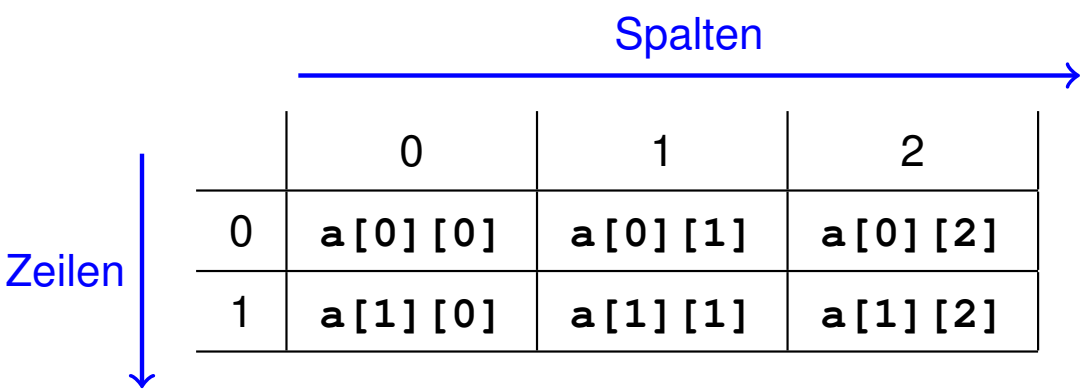
Mehrdimensionale Felder

- sind Felder von Feldern

```
int a[2][3]
```

`a` hat zwei Elemente, und jedes von ihnen ist ein Feld der Länge 3 mit zugrundeliegendem Typ `int`

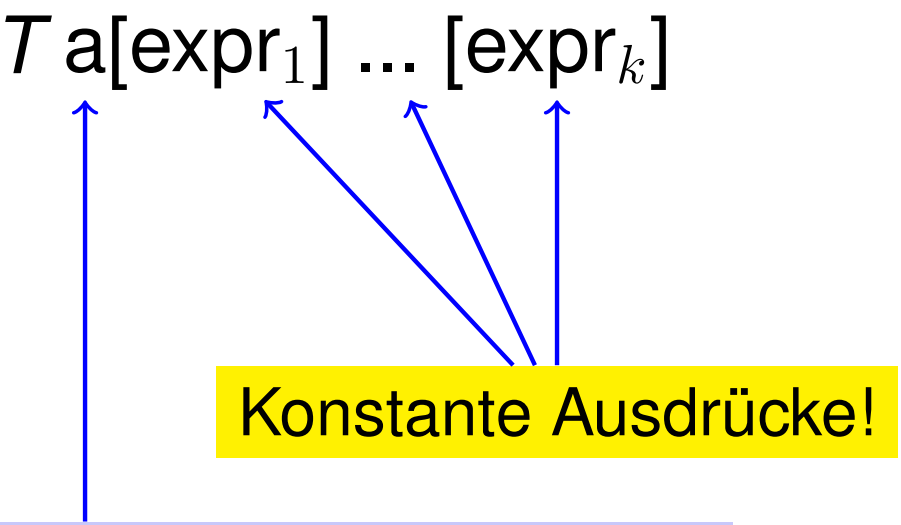
Im Kopf: Matrix, Tabelle, etc.



	0	1	2
0	`a[0][0]`	`a[0][1]`	`a[0][2]`
1	`a[1][0]`	`a[1][1]`	`a[1][2]`

Mehrdimensionale Felder

- sind Felder von Feldern von Feldern ....

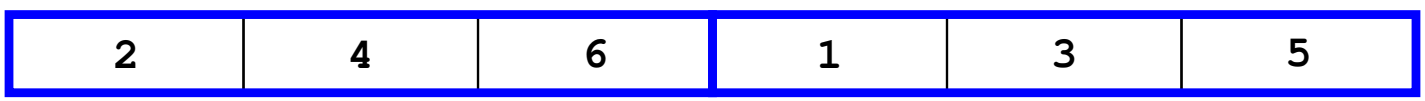


a hat $expr_1$ Elemente und jedes von ihnen ist ein Feld mit $expr_2$ Elementen, von denen jedes ein Feld mit $expr_3$ Elementen ist, ...

Mehrdimensionale Felder

Initialisierung:

```
int a[2][3] =
    {
        {2,4,6},{1,3,5}
    }
```

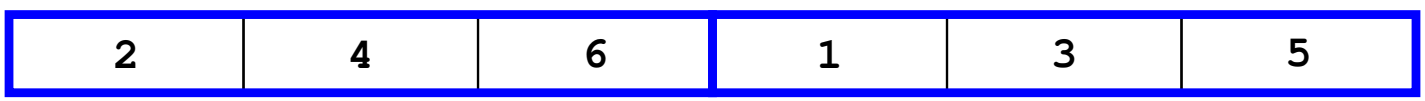


Mehrdimensionale Felder

Initialisierung:

```
int a[][3] =
    {
        {2,4,6},{1,3,5}
    }
```

Erste Dimension kann weggelassen werden

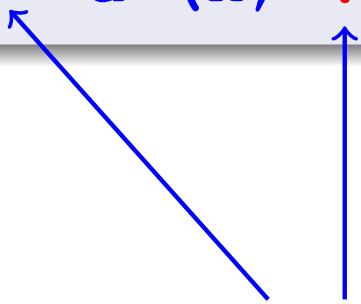


Vektoren von Vektoren

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?
- Lösung: Vektoren von Vektoren

Beispiel: Vektor der Länge n von Vektoren der Länge m:

```
std::vector<?> a (n, ?);
```



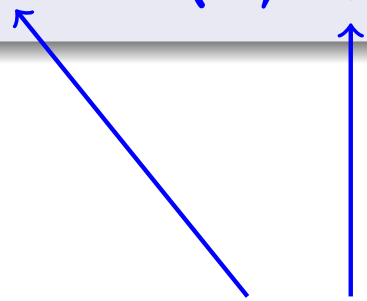
Zugrundeliegender Typ des ersten Vektors?

Vektoren von Vektoren

- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?
- Lösung: Vektoren von Vektoren

Beispiel: Vektor der Länge n von Vektoren der Länge m:

```
std::vector<?> a (n, ?);
```



Zugrundeliegender Typ des ersten Vektors?
`std::vector<int>`, initialisiert mit Länge m:
`std::vector<int>(m)`

Vektoren von Vektoren

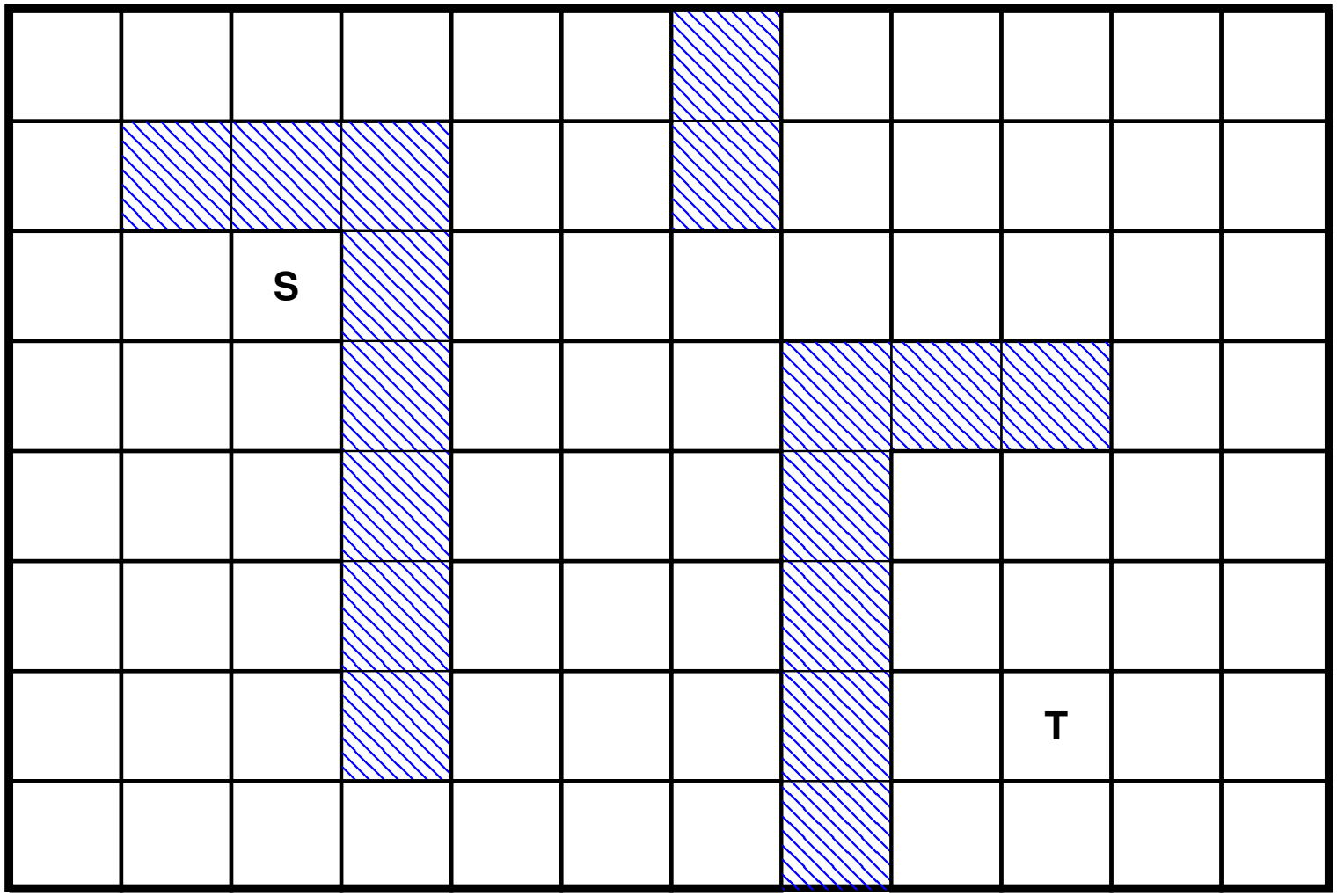
- Wie bekommen wir mehrdimensionale Felder mit variablen Dimensionen?
- Lösung: Vektoren von Vektoren

Beispiel: Vektor der Länge n von Vektoren der Länge m:

```
std::vector<std::vector<int> > a (n,
                                  std::vector<int>(m));
```

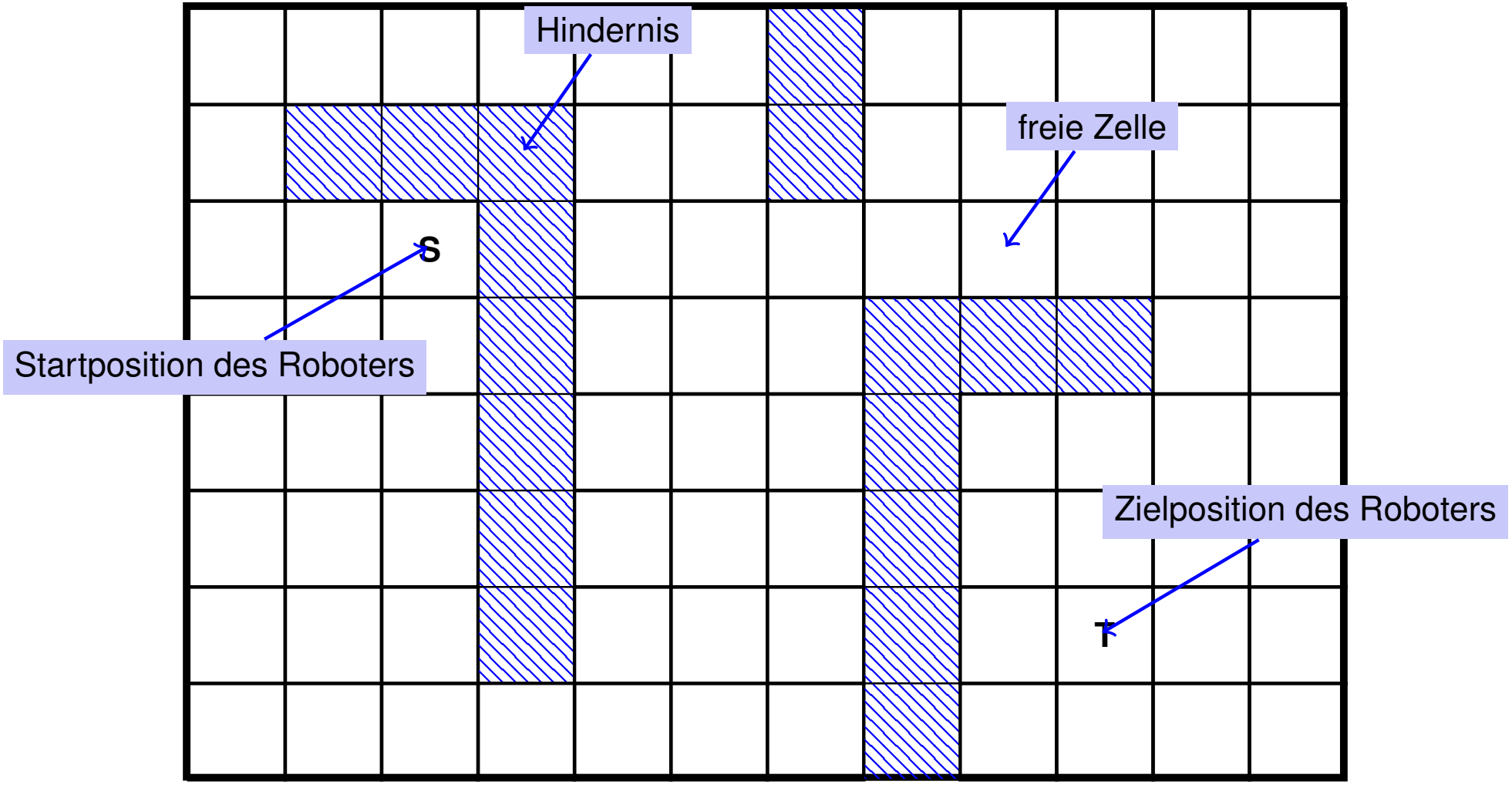
Anwendung: Kürzeste Wege

Fabrik-Halle ($n \times m$ quadratische Zellen)



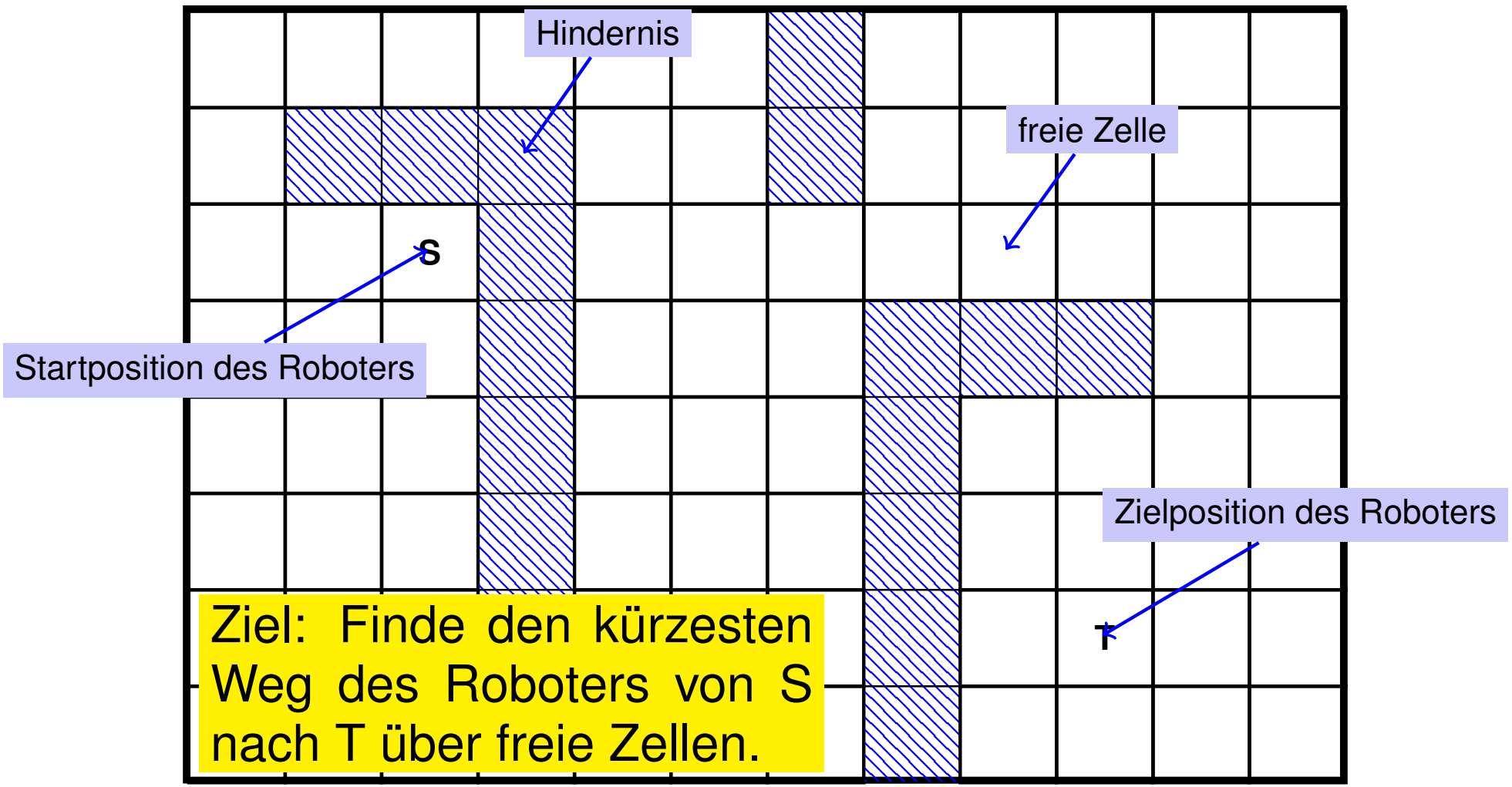
Anwendung: Kürzeste Wege

Fabrik-Halle ($n \times m$ quadratische Zellen)



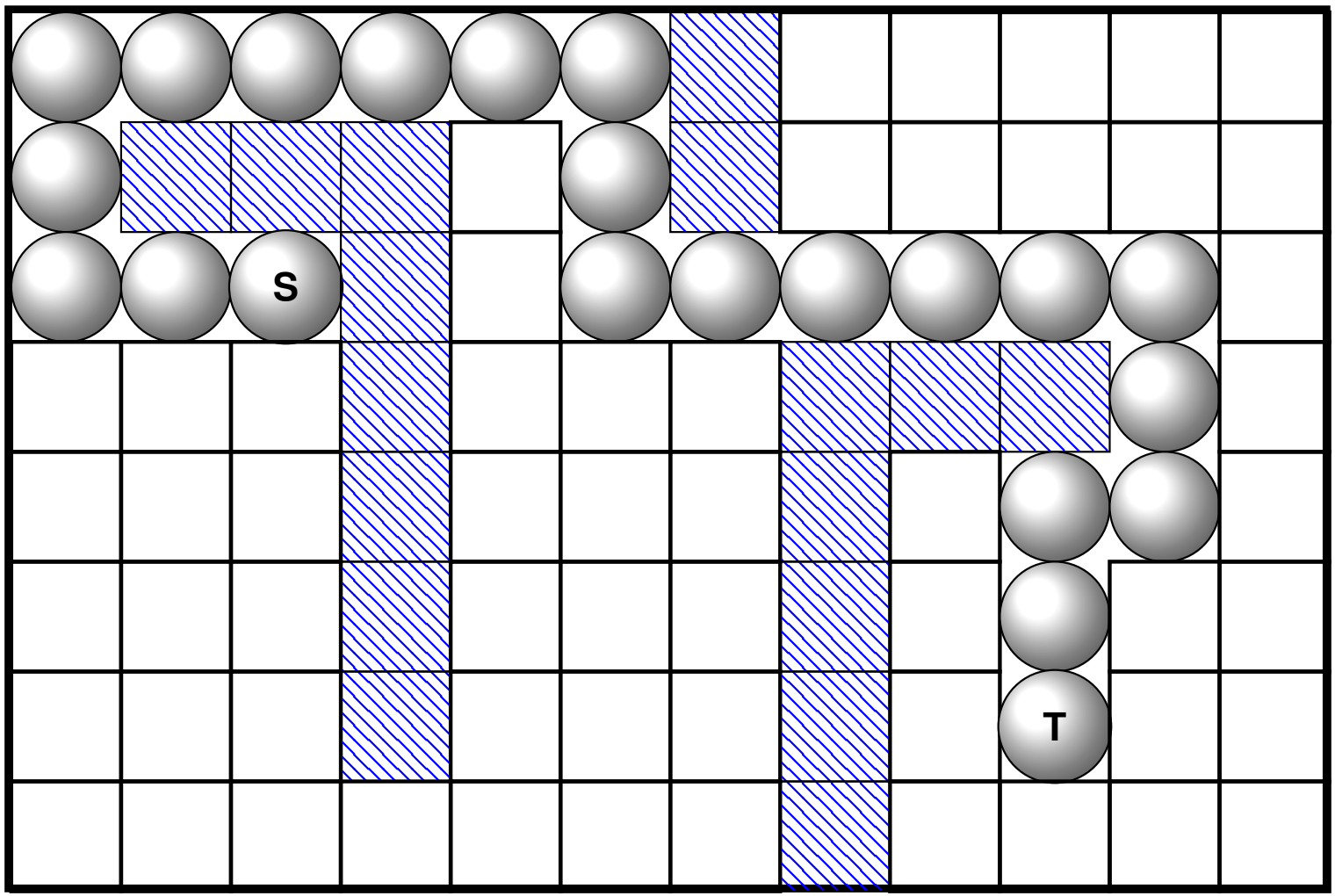
Anwendung: Kürzeste Wege

Fabrik-Halle ($n \times m$ quadratische Zellen)



Anwendung: Kürzeste Wege

Lösung



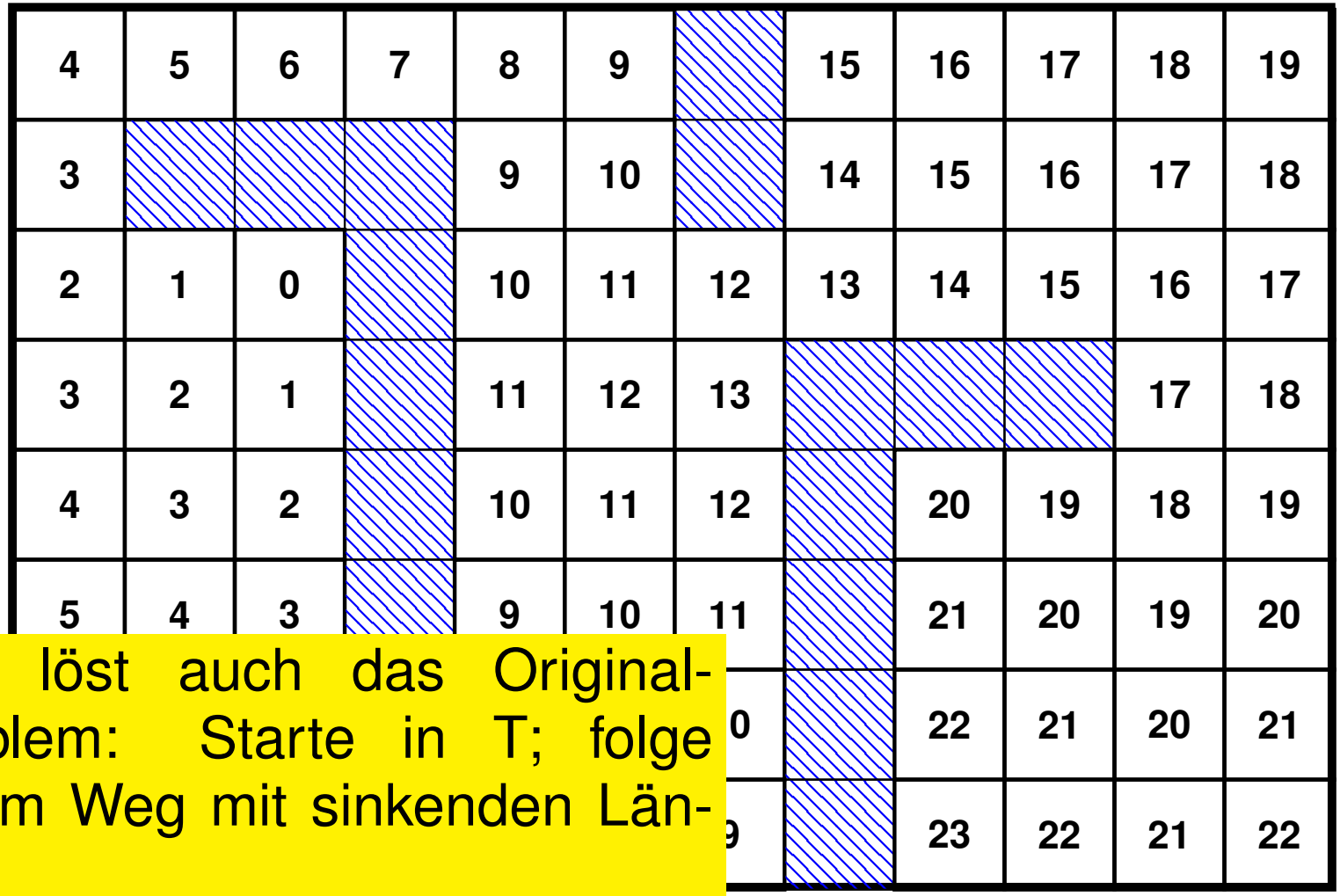
Ein (scheinbar) anderes Problem

Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen

4	5	6	7	8	9		15	16	17	18	19
3				9	10		14	15	16	17	18
2	1	0		10	11	12	13	14	15	16	17
3	2	1		11	12	13				17	18
4	3	2		10	11	12		20	19	18	19
5	4	3		9	10	11		21	20	19	20
6	5	4		8	9	10		22	21	20	21
7	6	5	6	7	8	9		23	22	21	22

Ein (scheinbar) anderes Problem

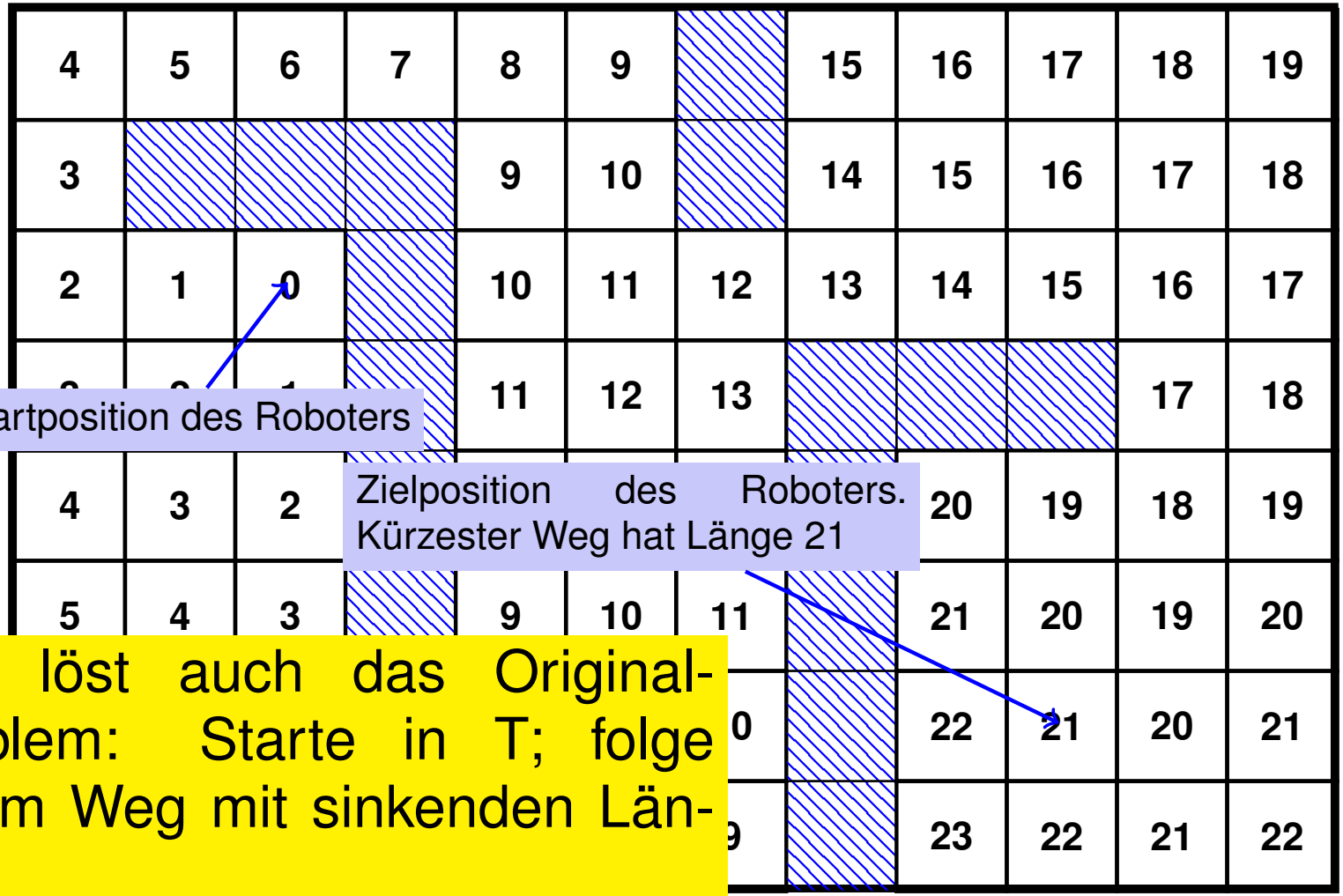
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

Ein (scheinbar) anderes Problem

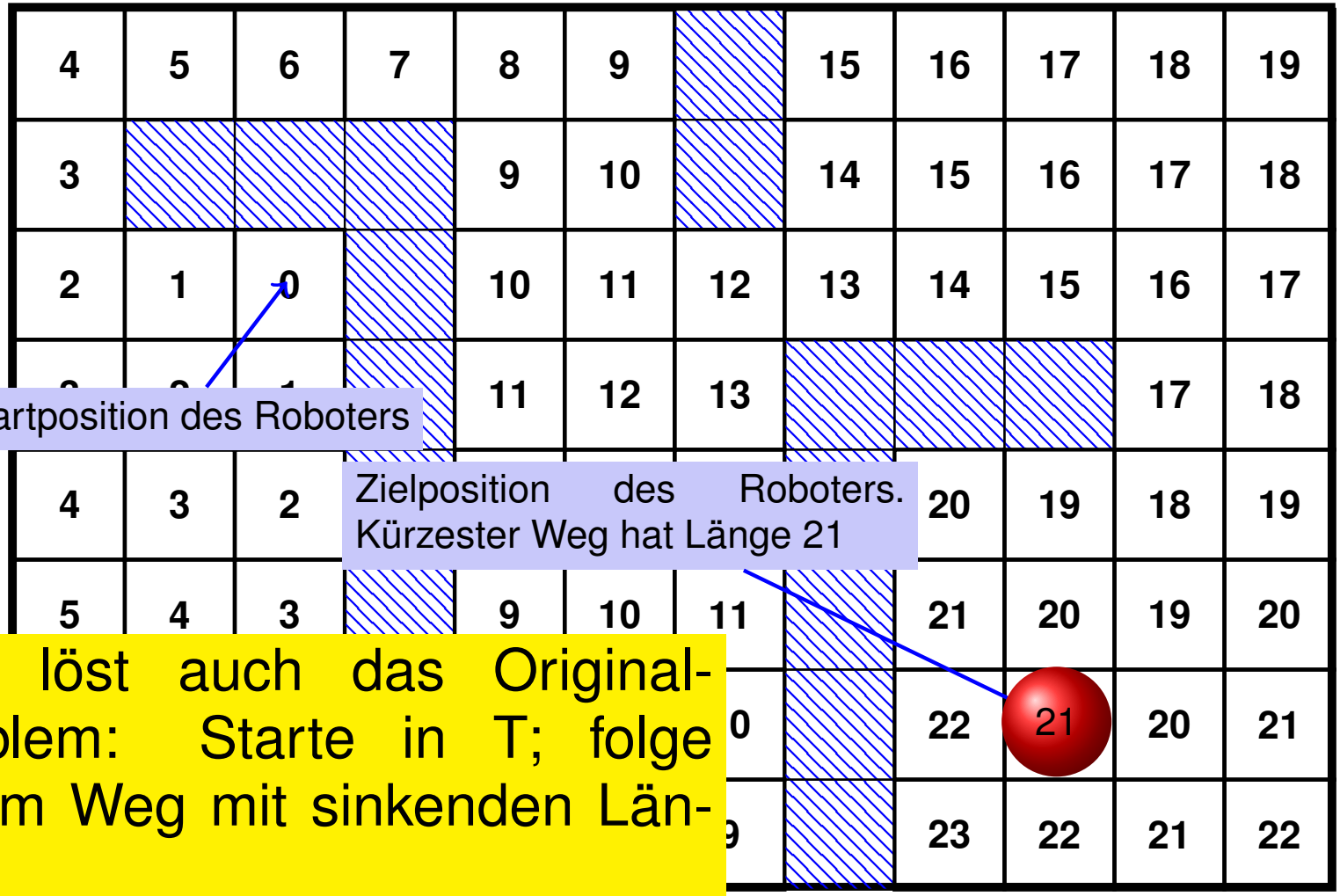
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

Ein (scheinbar) anderes Problem

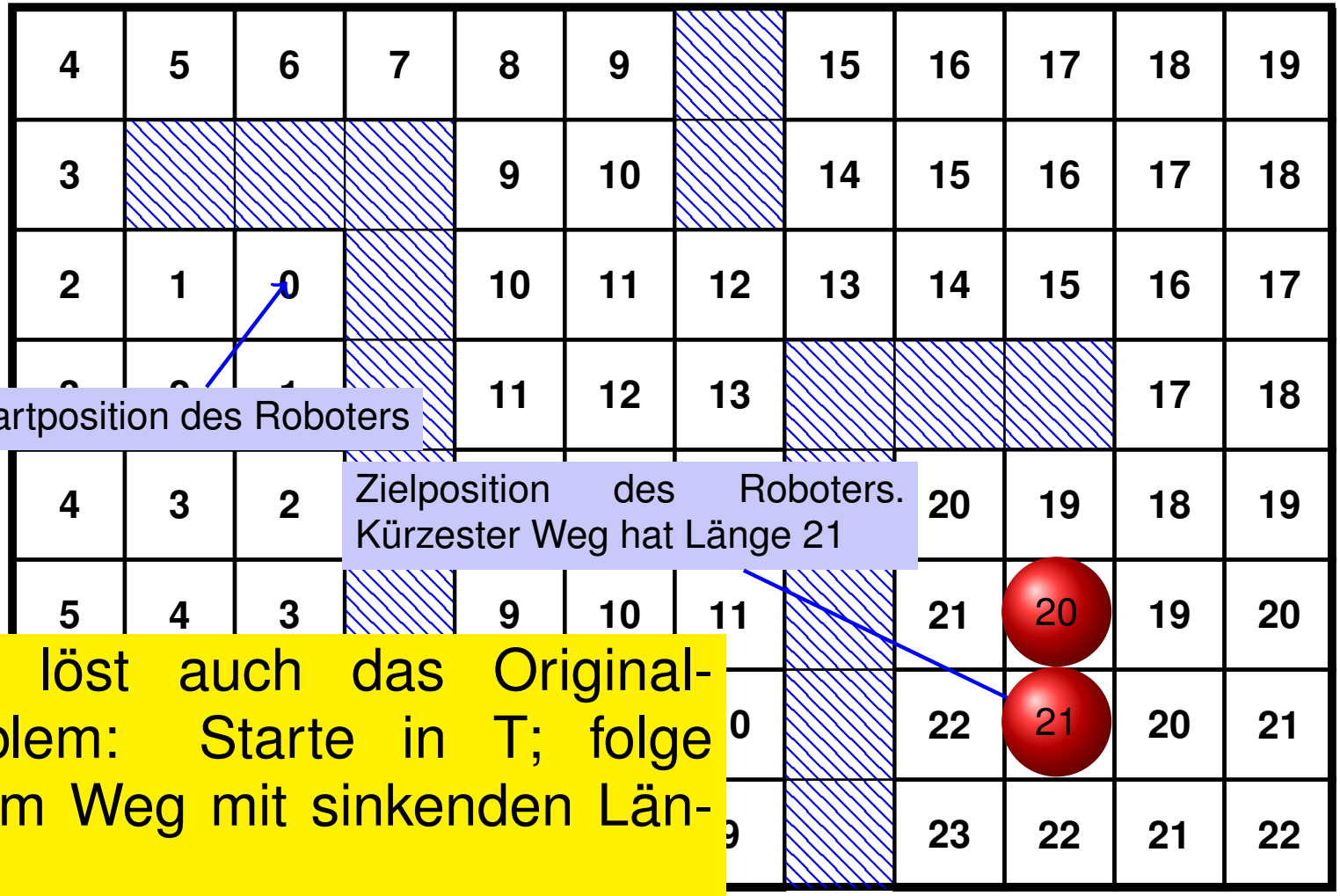
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

Ein (scheinbar) anderes Problem

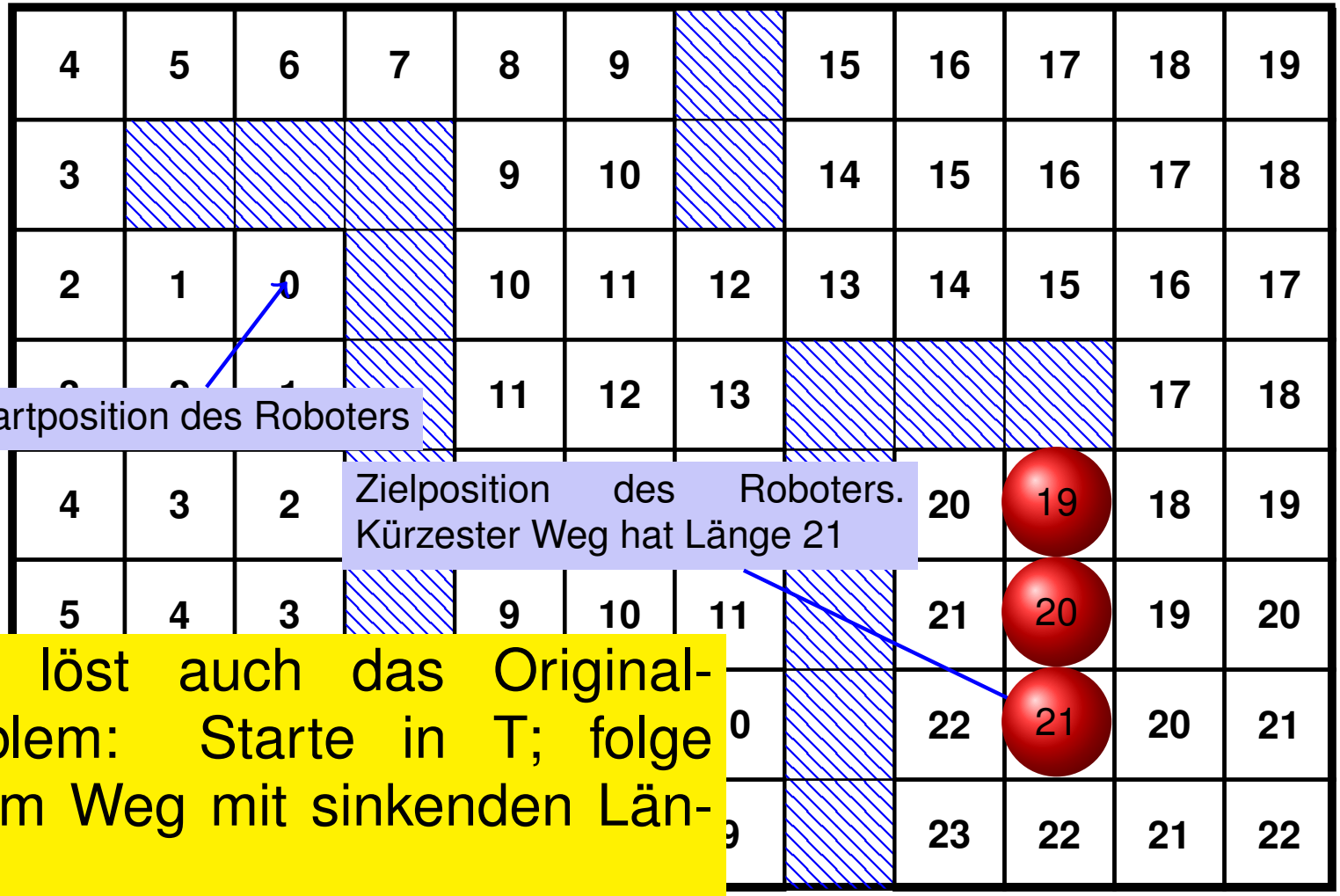
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

Ein (scheinbar) anderes Problem

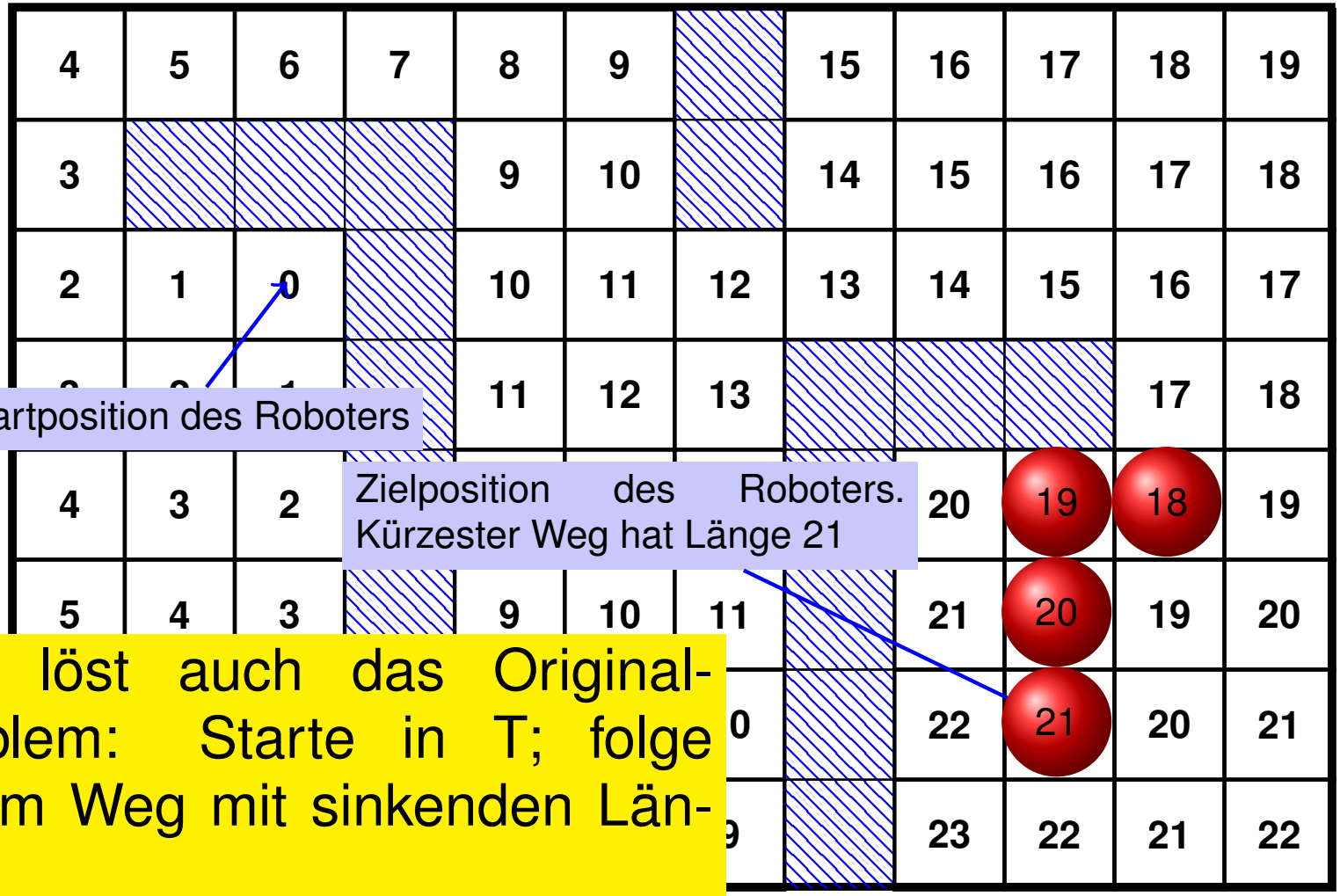
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

Ein (scheinbar) anderes Problem

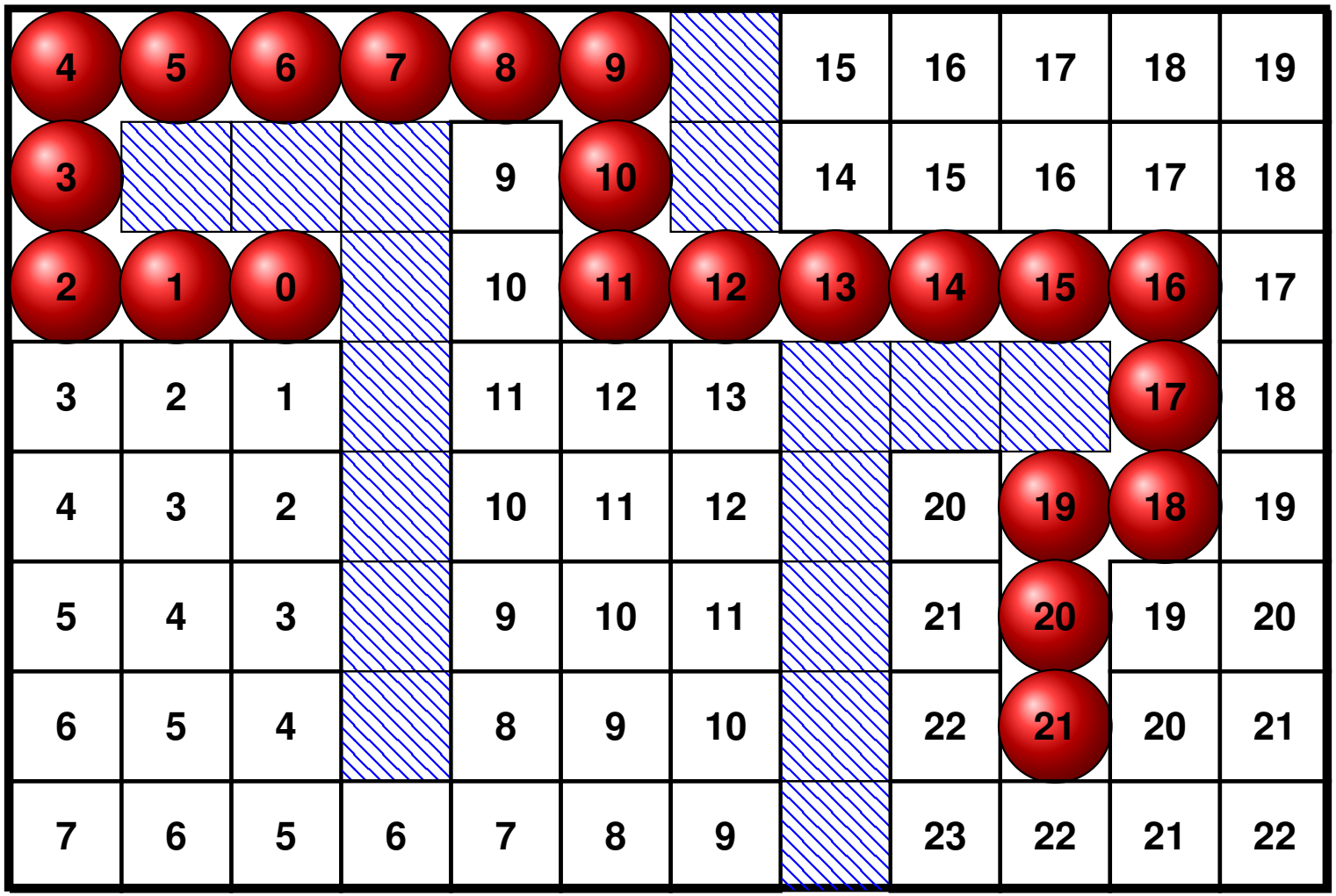
Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



Das löst auch das Original-Problem: Starte in T; folge einem Weg mit sinkenden Längen

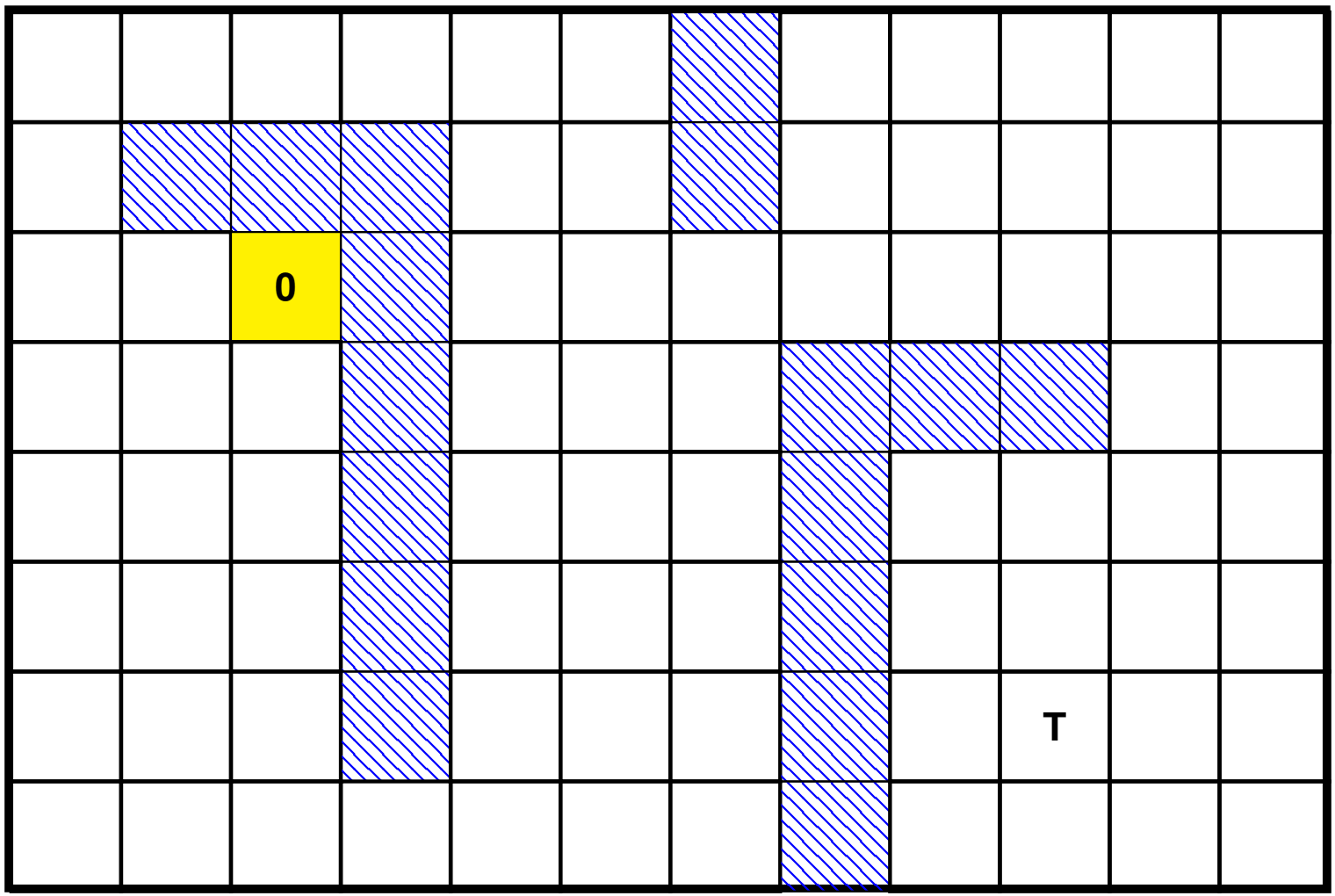
Ein (scheinbar) anderes Problem

Finde die *Längen* der kürzesten Wege zu *allen* möglichen Zielen



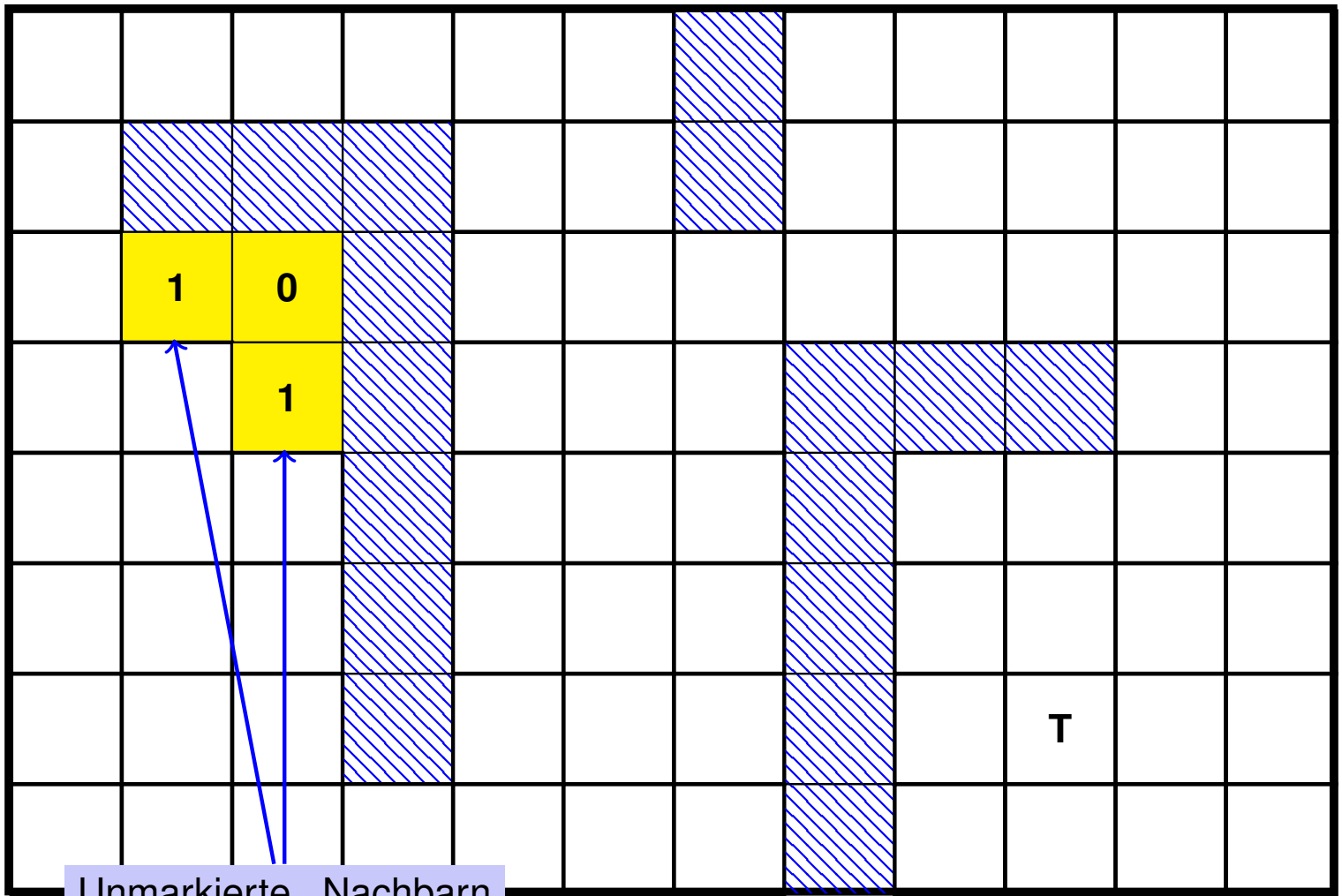
Markierung aller Zellen mit ihren Weglängen

Schritt 0: Alle Zellen mit Weglänge 0



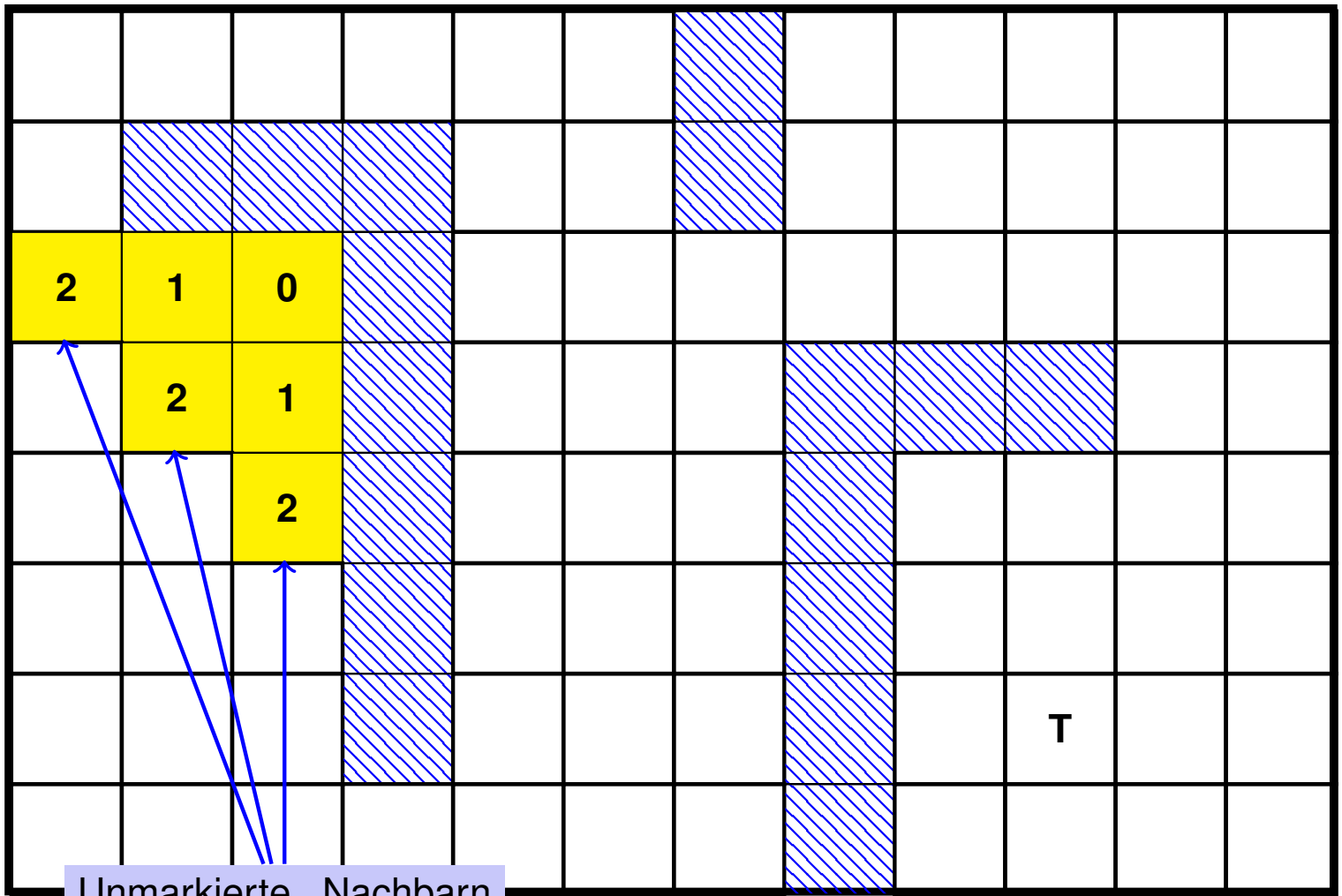
Markierung aller Zellen mit ihren Weglängen

Schritt 1: Alle Zellen mit Weglänge 1



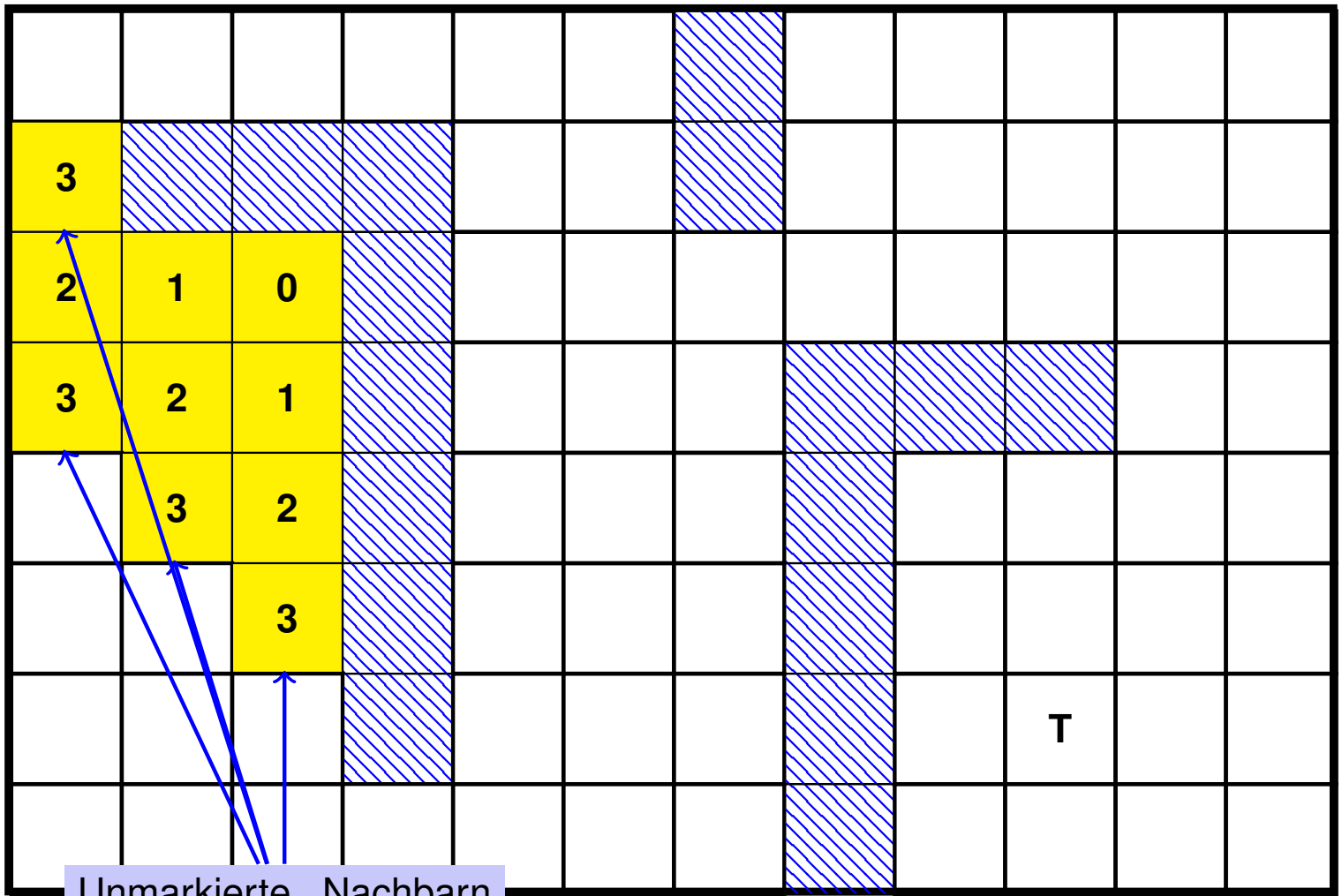
Markierung aller Zellen mit ihren Weglängen

Schritt 2: Alle Zellen mit Weglänge 2



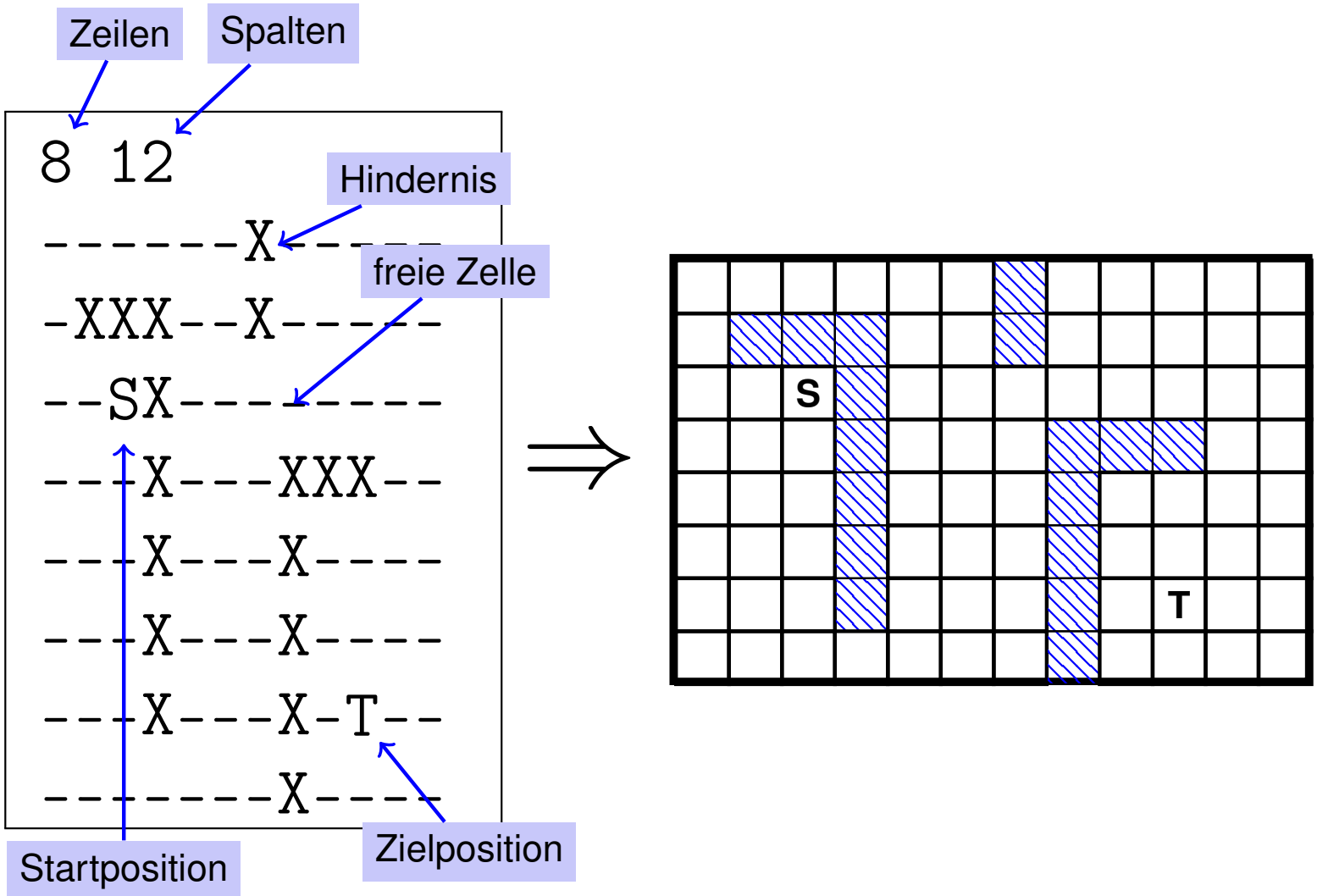
Markierung aller Zellen mit ihren Weglängen

Schritt 3: Alle Zellen mit Weglänge 3



Das Kürzeste-Wege-Programm

Eingabeformat:



Das Kürzeste-Wege-Programm

- Einlesen der Dimensionen und Bereitstellung eines zweidimensionalen Feldes für die Weglängen

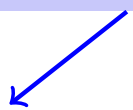
```
#include<iostream>
#include<vector>

int main()
{
  // read floor dimensions
  int n; std::cin >> n; // number of rows
  int m; std::cin >> m; // number of columns

  // define a two-dimensional
  // array of dimensions
  // (n+2) x (m+2) to hold the floor plus extra walls around
  std::vector<std::vector<int> > floor (n+2, std::vector<int>(m+2));
```

Das Kürzeste-Wege-Programm

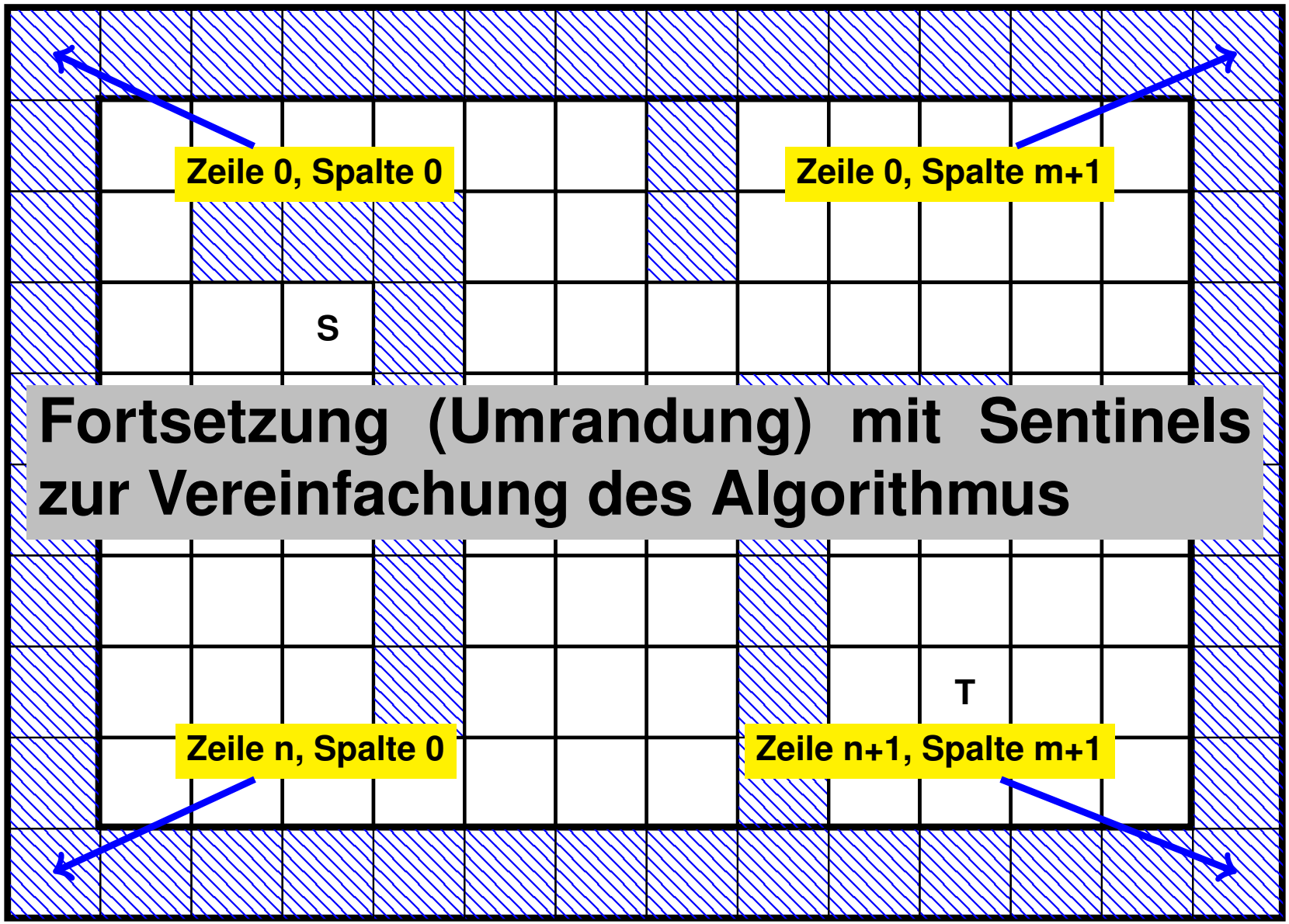
- Einlesen der Dimensionen und Bereitstellung eines zweidimensionalen Feldes für die Weglängen

```
#include<iostream>
#include<vector>

int main()
{
  // read floor dimensions
  int n; std::cin >> n; // number of rows
  int m; std::cin >> m; // number of columns

  // define a two-dimensional
  // array of dimensions
  // (n+2) x (m+2) to hold the floor plus extra walls around
  std::vector<std::vector<int> > floor (n+2, std::vector<int>(m+2));
```

Wächter (Sentinel)



Das Kürzeste-Wege-Programm



Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Zielkoordinaten: Zeilen-/Spaltenindex

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Zeilenweises Einlesen

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Eingabezeichen in Zeile `r`, Spalte `c`

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Länge bekannt (0)

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Länge unbekannt

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Länge nicht relevant (Hindernis)

Das Kürzeste-Wege-Programm

Einlesen der Hallenbelegung und Initialisierung der Längen

```
int tr = 0;
int tc = 0;
for (int r=1; r<n+1; ++r)
  for (int c=1; c<m+1; ++c) {
    char entry = '-';
    std::cin >> entry;
    if      (entry == 'S') floor[r][c] = 0;
    else if (entry == 'T') floor[tr = r][tc = c] = -1;
    else if (entry == 'X') floor[r][c] = -2;
    else if (entry == '-') floor[r][c] = -1;
  }
```

Länge unbekannt

Das Kürzeste-Wege-Programm

Hinzufügen der umschliessenden „Wände”

```
for (int r=0; r<n+2; ++r)
  floor[r][0] = floor[r][m+1] = -2;

for (int c=0; c<m+2; ++c)
  floor[0][c] = floor[n+1][c] = -2;
```

Das Kürzeste-Wege-Programm

Hinzufügen der umschliessenden „Wände”

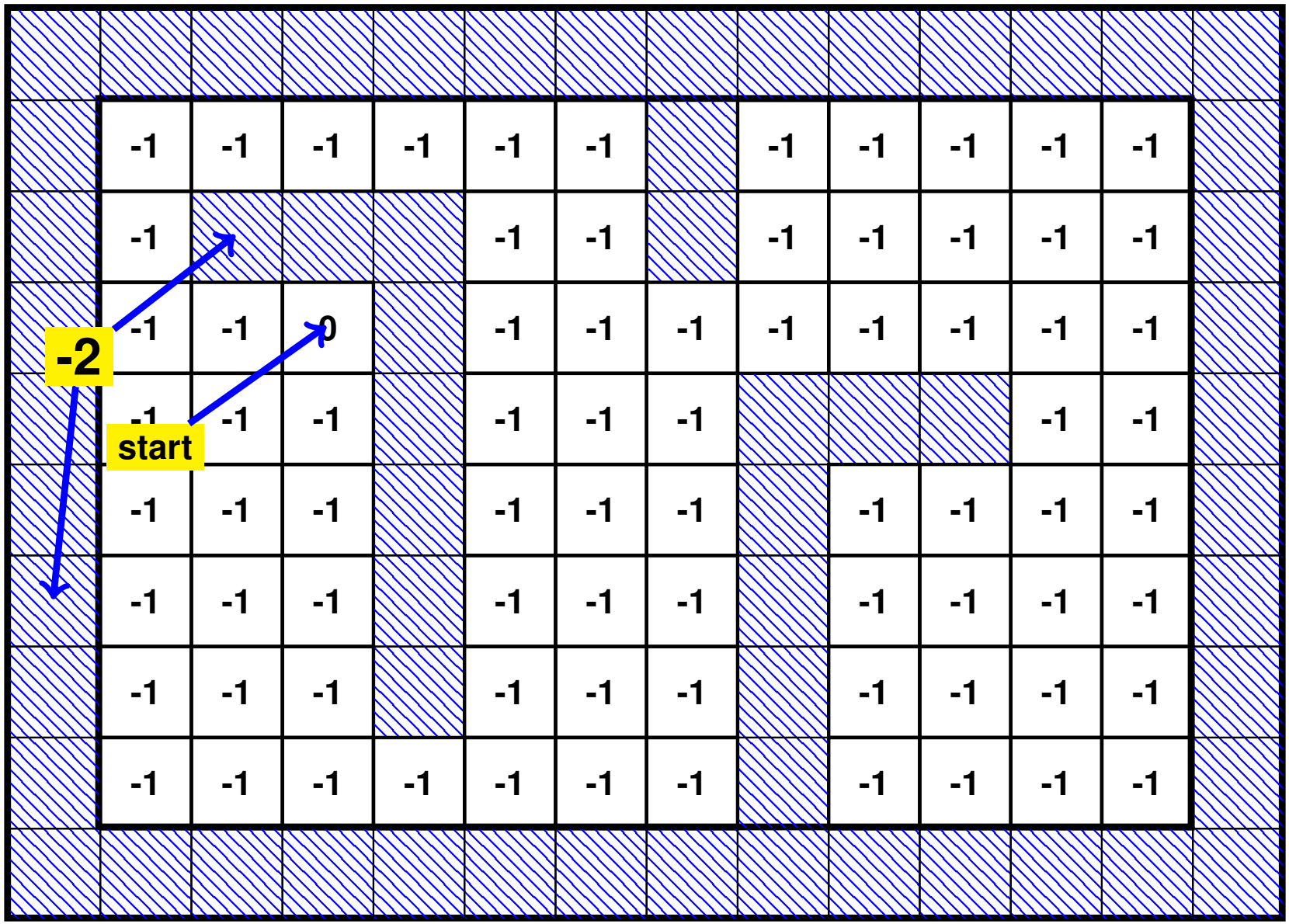
```
for (int r=0; r<n+2; ++r)
  floor[r][0] = floor[r][m+1] = -2;
```

Spalten 0 und m+1

```
for (int c=0; c<m+2; ++c)
  floor[0][c] = floor[n+1][c] = -2;
```

Zeilen 0 und n+1

Das Kürzeste-Wege-Programm



Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Variable progress zeigt an, ob in einem Durchlauf durch alle Zellen Fortschritt stattgefunden hat

Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Iteriere über alle Zellen

Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Zelle schon markiert oder Hindernis

Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Ein Nachbar hat Weglänge $i - 1$. Man darf alle Nachbarn testen, da durch die Wächter Nachbarn garantiert existieren.

Das Kürzeste-Wege-Programm

Hauptschleife: finde und markiere alle Zellen mit Weglängen $i = 1, 2, 3...$

```
for (int i=1;; ++i) {
  bool progress = false;
  for (int r=1; r<n+1; ++r)
     for (int c=1; c<m+1; ++c) {
      if (floor[r][c] != -1) continue;
      if (floor[r-1][c] == i-1 || floor[r+1][c] == i-1 ||
          floor[r][c-1] == i-1 || floor[r][c+1] == i-1 ) {
        floor[r][c] = i; // label cell with i
        progress = true;
      }
    }
  if (!progress) break;
}
```

Kein Fortschritt, alle erreichbaren Zellen markiert; fertig.

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```

Zielkoordinaten

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```

Solange Ziel nicht erreicht

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```

`d`: um eins kleinere Weglänge

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

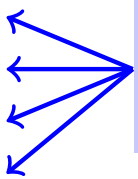
```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```

Markiere Zelle mit -3

Das Kürzeste-Wege-Programm

Markieren des kürzesten Weges durch „Rückwärtslaufen” vom Ziel zum Start

```
int r = tr; int c = tc;
while (floor[r][c] > 0) {
  const int d = floor[r][c] - 1;
  floor[r][c] = -3;
  if      (floor[r-1][c] == d) --r;
  else if (floor[r+1][c] == d) ++r;
  else if (floor[r][c-1] == d) --c;
  else ++c; // (floor[r][c+1] == d)
}
```



Gehe zu Nachbarzelle mit Weglänge d (existiert!) und wiederhole ...

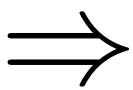
Das Kürzeste-Wege-Programm

-3	-3	-3	-3	-3	-3		15	16	17	18	19
-3				9	-3		14	15	16	17	18
-3	-3	0		10	-3	-3	-3	-3	-3	-3	17
3	2	1		11	12	13				-3	18
4	3	2		10	11	12		20	-3	-3	19
5	4	3		9	10	11		21	-3	19	20
6	5	4		8	9	10		22	-3	20	21
7	6	5	6	7	8	9		23	22	21	22

Das Kürzeste-Wege-Programm

Ausgabe

```
for (int r=1; r<n+1; ++r) {
  for (int c=1; c<m+1; ++c)
    if (floor[r][c] == 0)
        std::cout << 'S';
    else if (r == tr && c == tc)
        std::cout << 'T';
    else if (floor[r][c] == -3)
        std::cout << 'o';
    else if (floor[r][c] == -2)
        std::cout << 'X';
    else
        std::cout << '-';
  std::cout << "\n";
}
```



```
ooooooX-----
oXXX-oX-----
ooSX-oooooo-
---X---XXXo-
---X---X-oo-
---X---X-o--
---X---X-T--
--------X----
```

Das Kürzeste-Wege-Programm

- Das Programm kann recht langsam sein, weil für jedes i alle Zellen durchlaufen werden

Das Kürzeste-Wege-Programm

- Das Programm kann recht langsam sein, weil für jedes i alle Zellen durchlaufen werden
- Verbesserung: durchlaufe jeweils nur die Nachbarn der Zellen mit Markierung i-1