Introduction to Dynamic Programming

Andre Ryser

November 4, 2018

Schweizer Informatik Olympiade

Inhaltsverzeichnis

Einführung

Was ist dynamische Programmierung?

Was ist dynamische Programmierung?

Dynamische Programmierung ist...

Was ist dynamische Programmierung?

Dynamische Programmierung ist...

- **kein** Algorithmus

Was ist dynamische Programmierung?

Dynamische Programmierung ist...

- **kein** Algorithmus
- eine Technik um Probleme (insbesonders Optimierungsprobleme) effizienter zu lösen.

Was ist Dynamische Programmierung: Eine Richtlinie

Keep it simple, stupid!

Was ist Dynamische Programmierung: Eine Richtlinie

Keep it simple, stupid!

- Die selbe Arbeit zweimal machen ist schlecht.

Was ist Dynamische Programmierung: Eine Richtlinie

Keep it simple, stupid!

- Die selbe Arbeit zweimal machen ist schlecht.
- Wenn du den selben Code zweimal brauchst, mach eine Funktion.

Was ist Dynamische Programmierung: Eine Richtlinie

Keep it simple, stupid!

- Die selbe Arbeit zweimal machen ist schlecht.
- Wenn du den selben Code zweimal brauchst, mach eine Funktion.
- Das Programm sollte Dinge die es bereits berchnet hat, nicht noch einmal berechnen.

Was ist dynamische Programmierung

DP ist eine Technik die wir verwenden können wenn:

Was ist dynamische Programmierung

DP ist eine Technik die wir verwenden können wenn:

- Das Problem sich in Teilprobleme zerteilen lässt.

Was ist dynamische Programmierung

DP ist eine Technik die wir verwenden können wenn:

- Das Problem sich in Teilprobleme zerteilen lässt.
- Wir von diesen Teillösungen (Lösungen für die Teilprobleme) die Lösung für das ursprüchliche Problem konstruieren können.

Was ist dynamische Programmierung

DP ist eine Technik die wir verwenden können wenn:

- Das Problem sich in Teilprobleme zerteilen lässt.
- Wir von diesen Teillösungen (Lösungen für die Teilprobleme) die Lösung für das ursprüchliche Problem konstruieren können.
- Sich diese Teilprobleme überlappen und wir die selben Teilprobleme mehrmals lösen.

Was ist dynamische Programmierung

DP ist eine Technik die wir verwenden können wenn:

- Das Problem sich in Teilprobleme zerteilen lässt.
- Wir von diesen Teillösungen (Lösungen für die Teilprobleme) die Lösung für das ursprüchliche Problem konstruieren können.
- Sich diese Teilprobleme überlappen und wir die selben Teilprobleme mehrmals lösen.

Dynamische Programmierung vermeidet dieses Problem indem es sich **erinnert** was bereits berechnet wurde und es nicht erneut löst.

Ein simples Beispiel: Fibonacci Folge

Fibonacci Folge

Fibonacci Sequenz kann wie folgt definiert werden:

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ \forall n \in \mathbb{N} \setminus \{0,1\}, f_n = f_{n-1} + f_{n-2} \end{cases}$$

Fibonacci Folge

Fibonacci Sequenz kann wie folgt definiert werden:

$$\begin{cases} f_0 = 0 \\ f_1 = 1 \\ \forall n \in \mathbb{N} \setminus \{0, 1\}, f_n = f_{n-1} + f_{n-2} \end{cases}$$

Hier sind einige der ersten Werte der Sequenz:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Fibonacci Folge: Intuitiver Algorithmus

Ein intuitiver Ansatz um die Folge zu berechnen wäre:

```
int fib(int n) {
    if(n==0) return 0;
    if(n==1) return 1;
    return fib(n-1) + fib(n-2);
}
```

Fibonacci Folge: Das Problem

Wenn du diesen Code ausführst um f_{100} zu berechnen musst du **sehr** lange auf die Antwort warten.

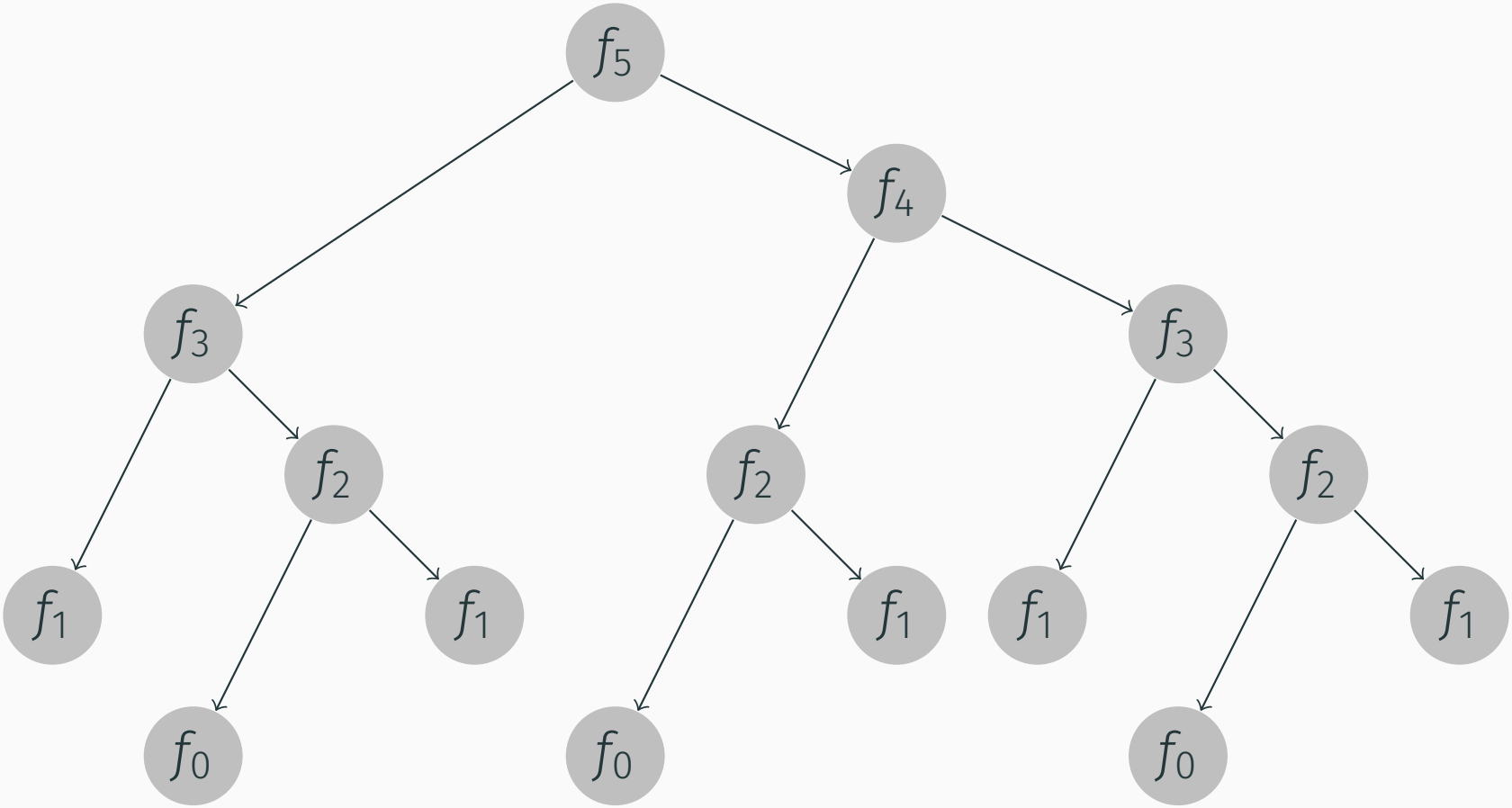
Fibonacci Folge: Das Problem

Wenn du diesen Code ausführst um f_{100} zu berechnen musst du **sehr** lange auf die Antwort warten.

Wieso?

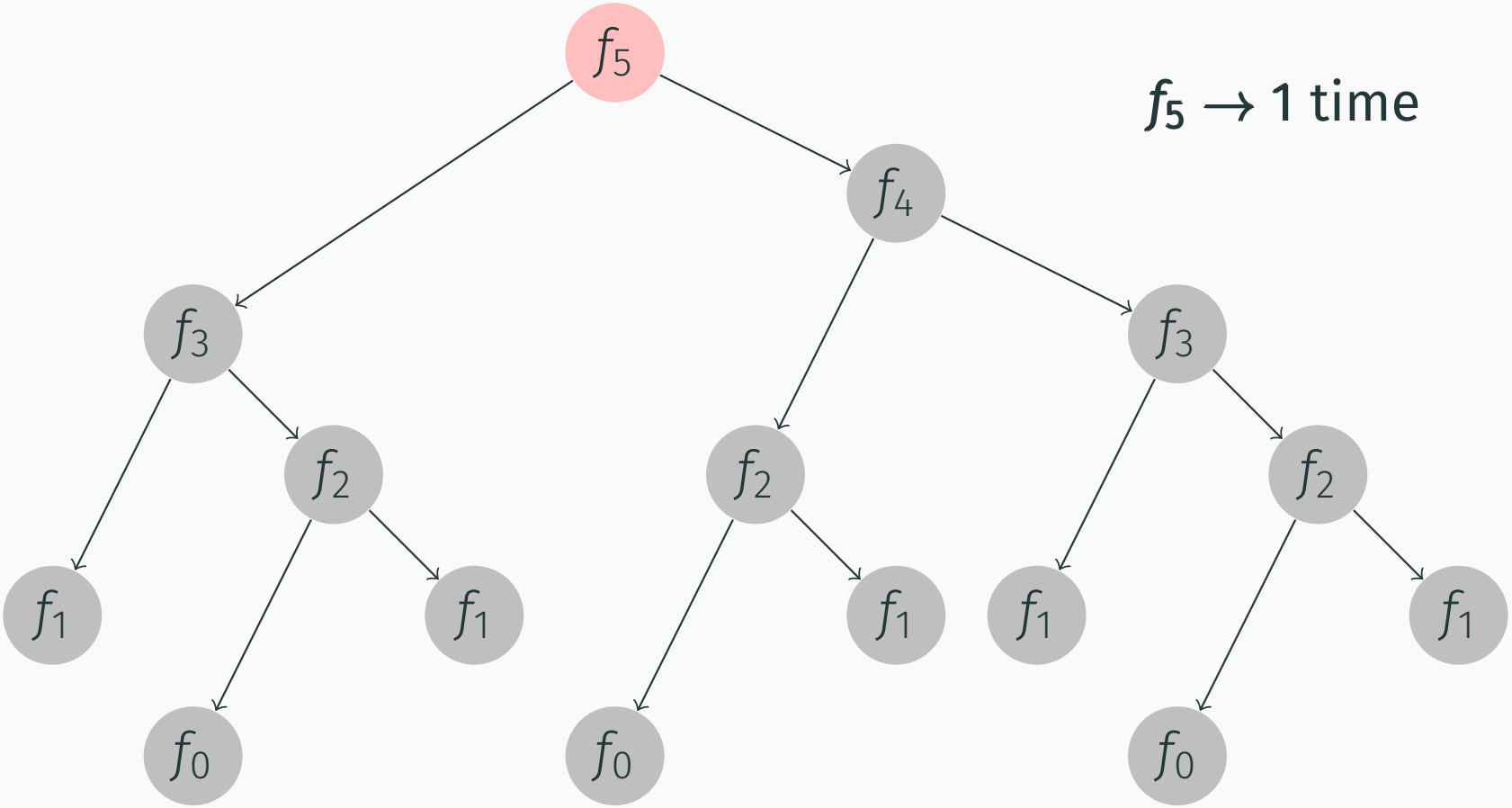
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



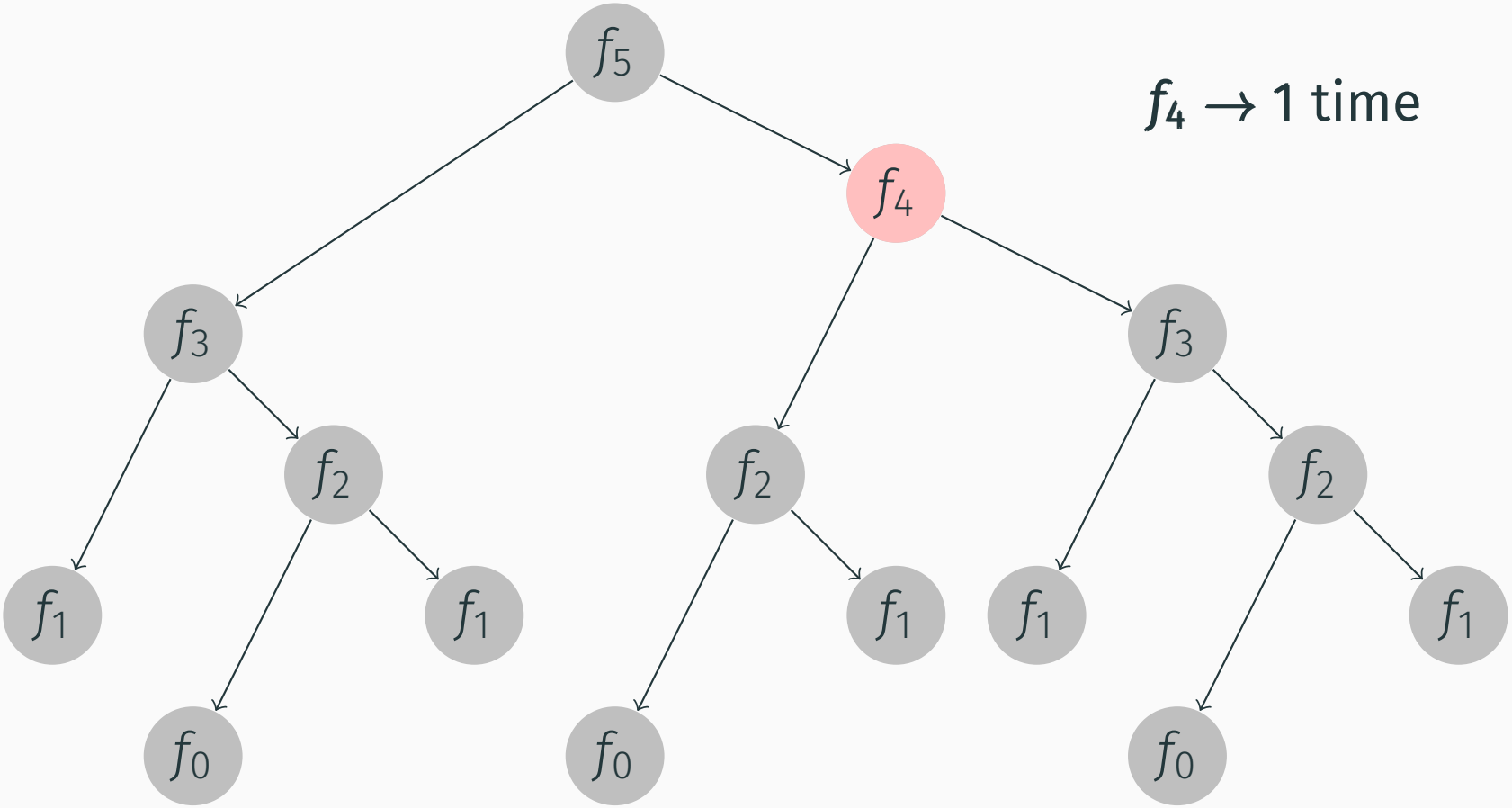
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



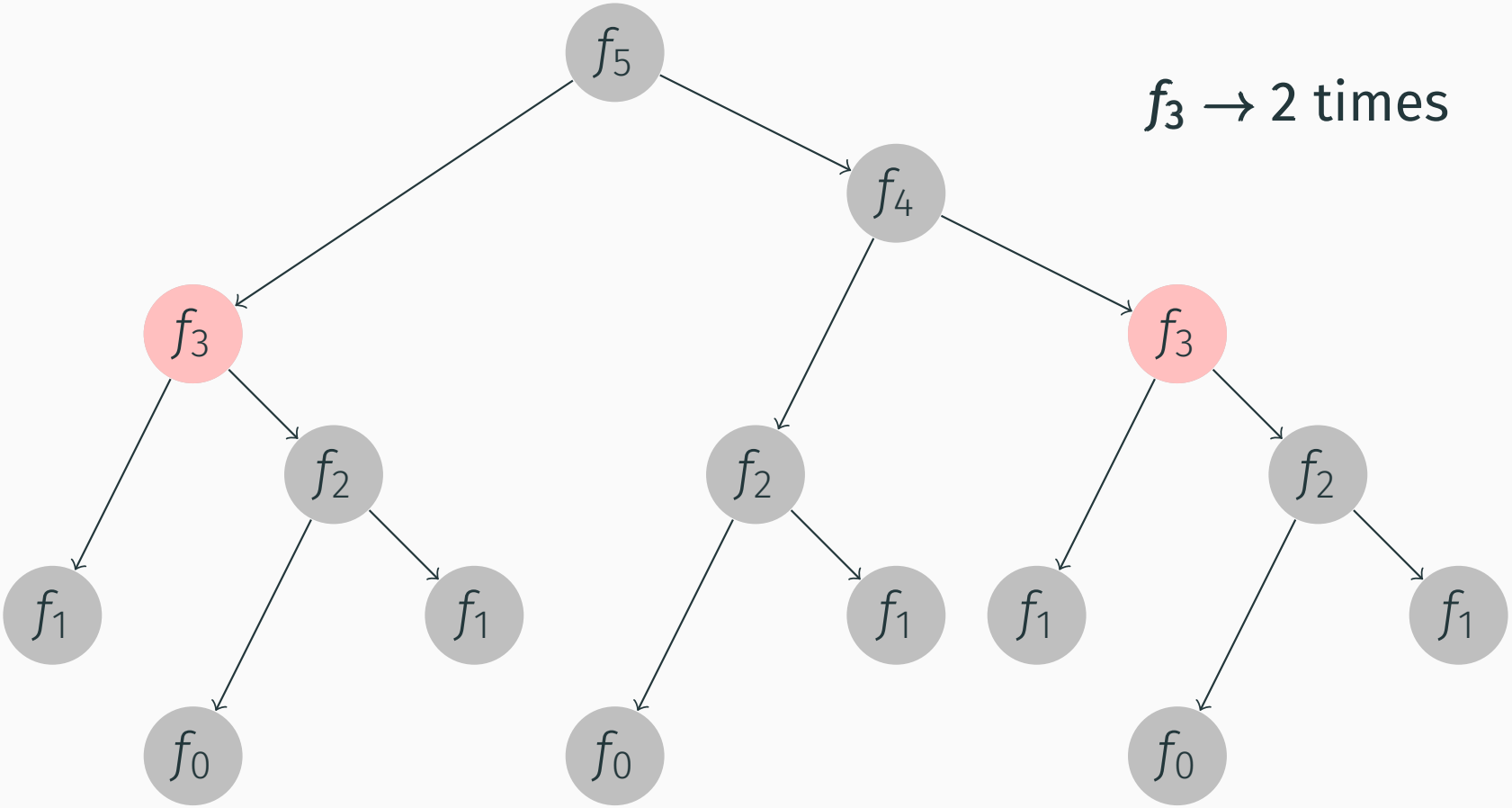
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



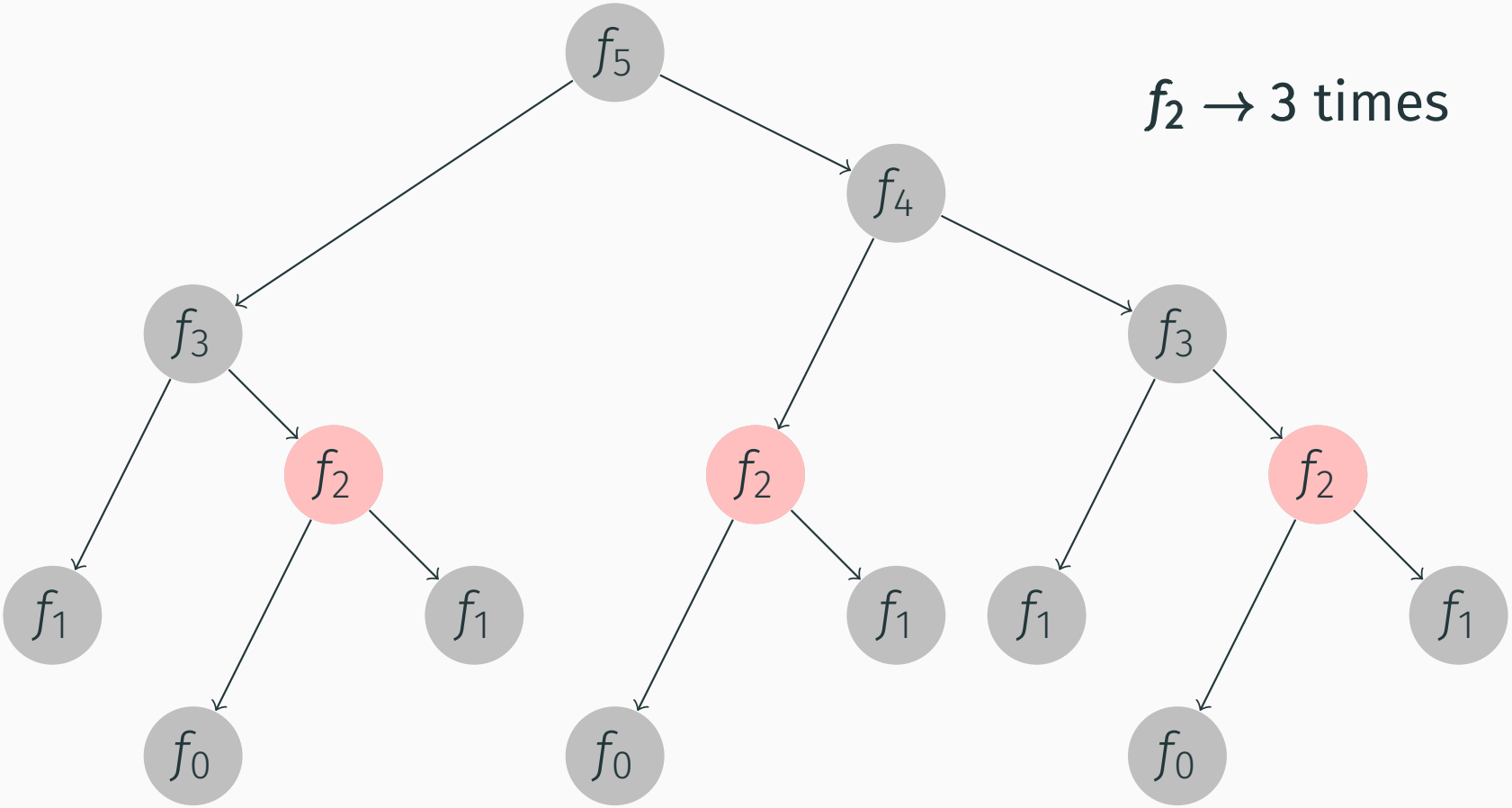
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



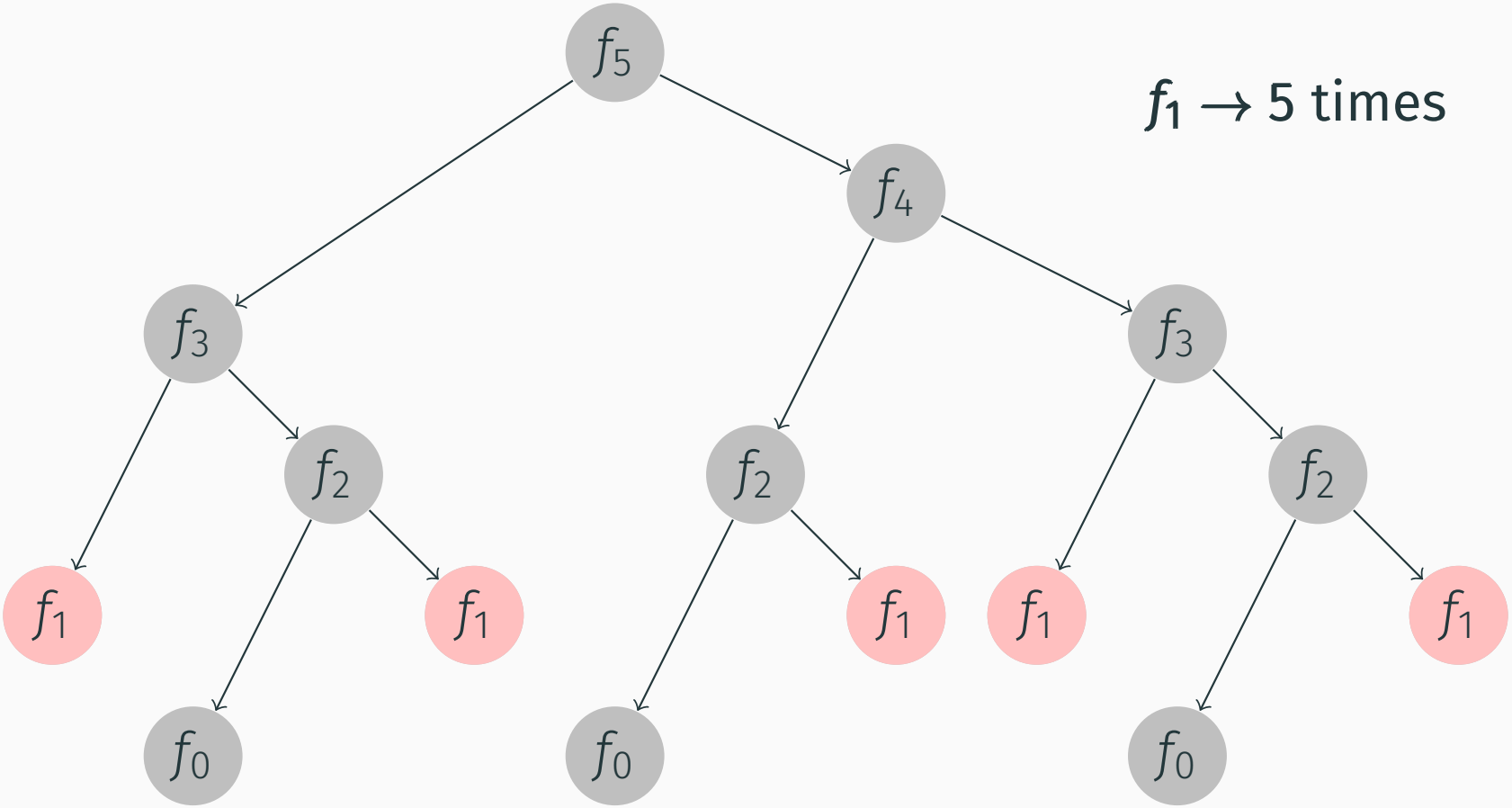
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



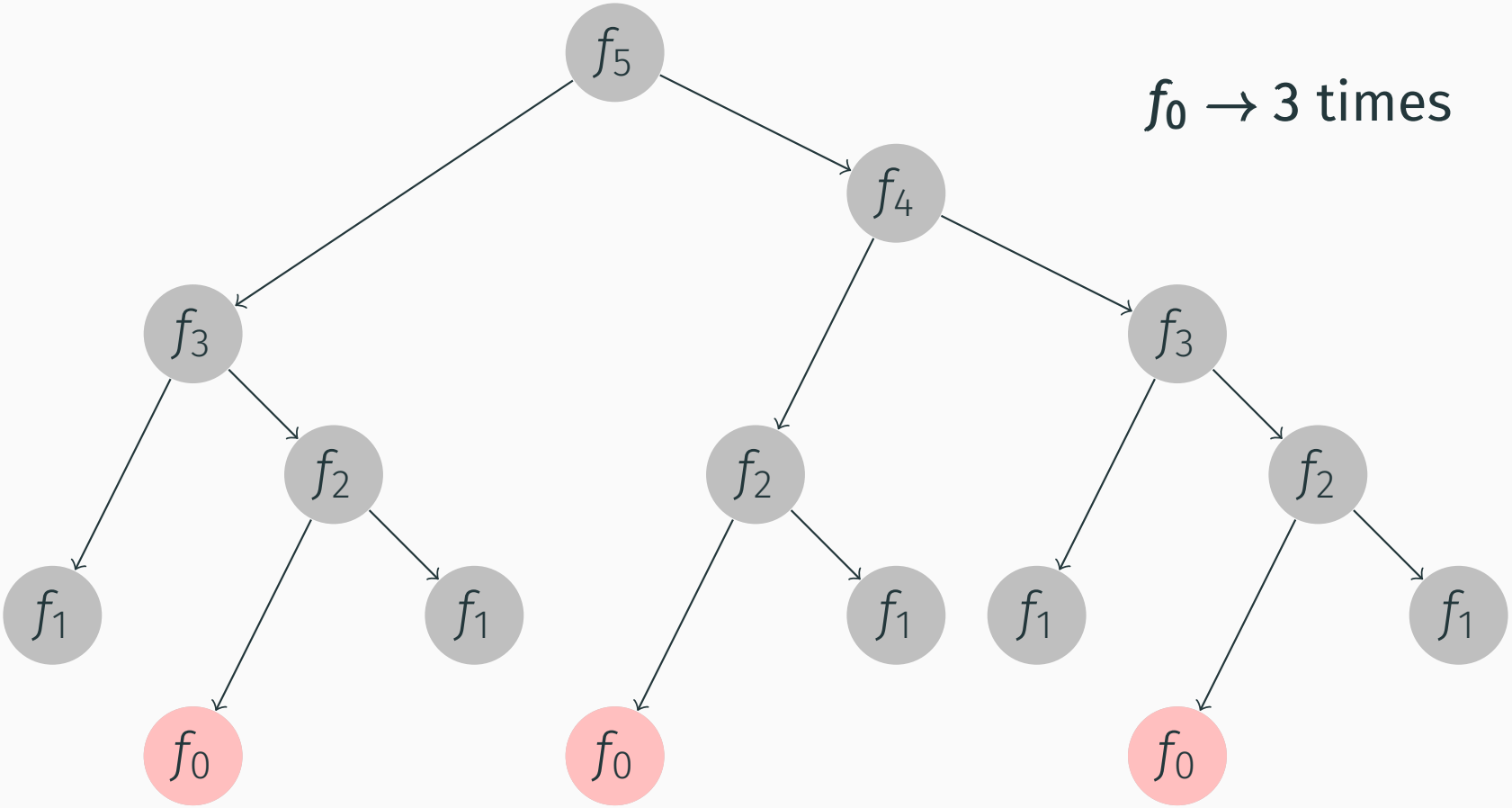
Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



Fibonacci Folge: Das Problem

Unser Programm berechnet die selben Werte mehrmals.
Schau wir uns diesen Baum an:



Fibonacci Folge: Das Problem

Die Anzahl an Operationen erhöht sich proportional mit dem Resultat (die Anzahl an Male die wir einen Wert berechnen ist Teile der Fibonacci Folge)

Fibonacci Folge: Das Problem

Die Anzahl an Operationen erhöht sich proportional mit dem Resultat (die Anzahl an Male die wir einen Wert berechnen ist Teile der Fibonacci Folge)
Die Fibonacci Folge wächst exponentiall, also haben wir exponentielle Laufzeit...

Fibonacci Folge: Die Lösung

Wir können eine effizientere Lösung finden.

Fibonacci Folge: Die Lösung

Wir können eine effizientere Lösung finden. Wir müssen Dinge nicht mehr als einmal berechnen:

Fibonacci Folge: Die Lösung

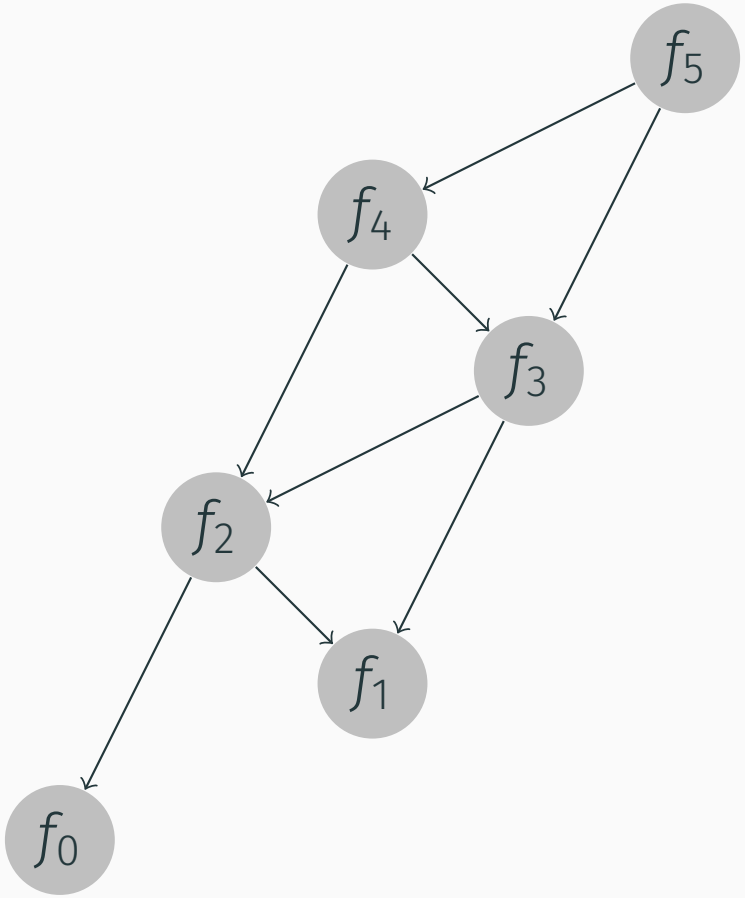
Wir können eine effizientere Lösung finden. Wir müssen Dinge nicht mehr als einmal berechnen:
Wir speichern einfach die Werte die wir bereits berechnet haben!

Fibonacci Folge: Die Lösung

Wir können eine effizientere Lösung finden. Wir müssen Dinge nicht mehr als einmal berechnen:
Wir speichern einfach die Werte die wir bereits berechnet haben!
Wir berechnen zuerst die tiefen Werte und kombinieren sie um den nächsten Wert zu berechnen.

Fibonacci Folge: Die Lösung

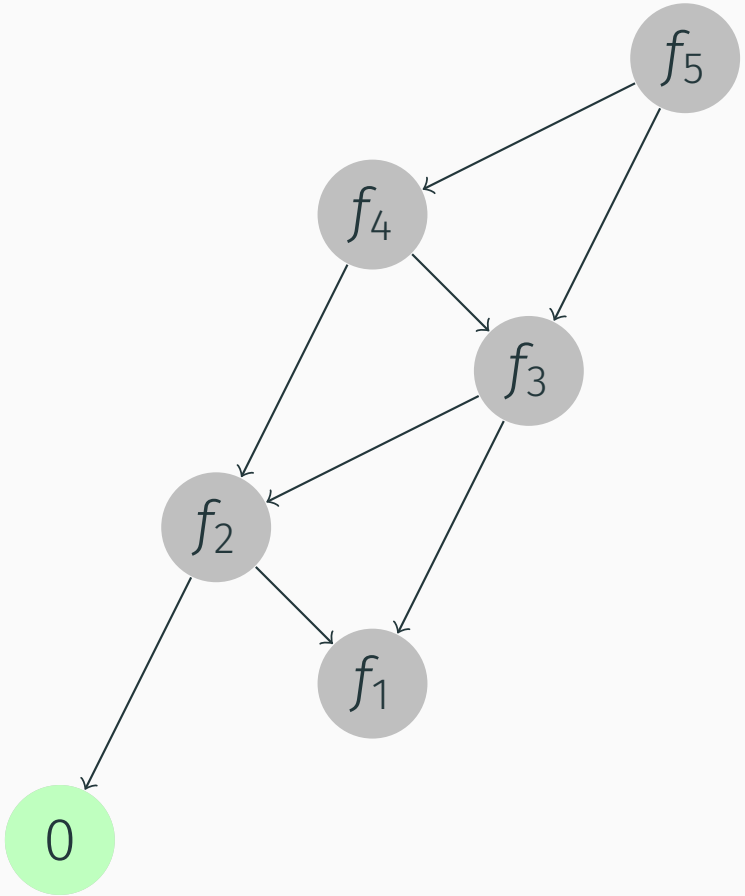
Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

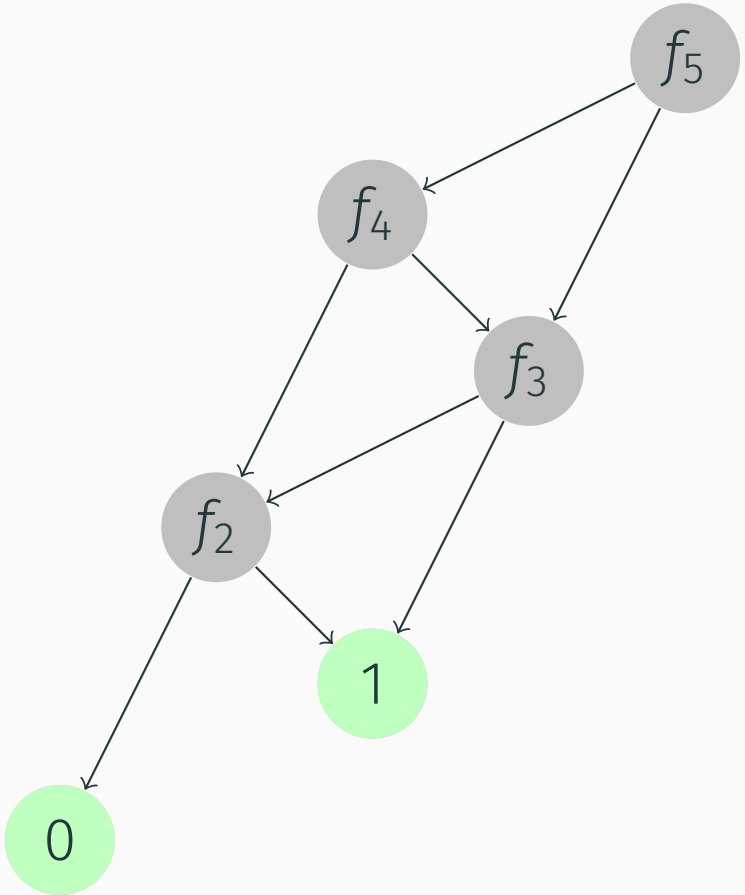
$$f_0 = 0$$



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

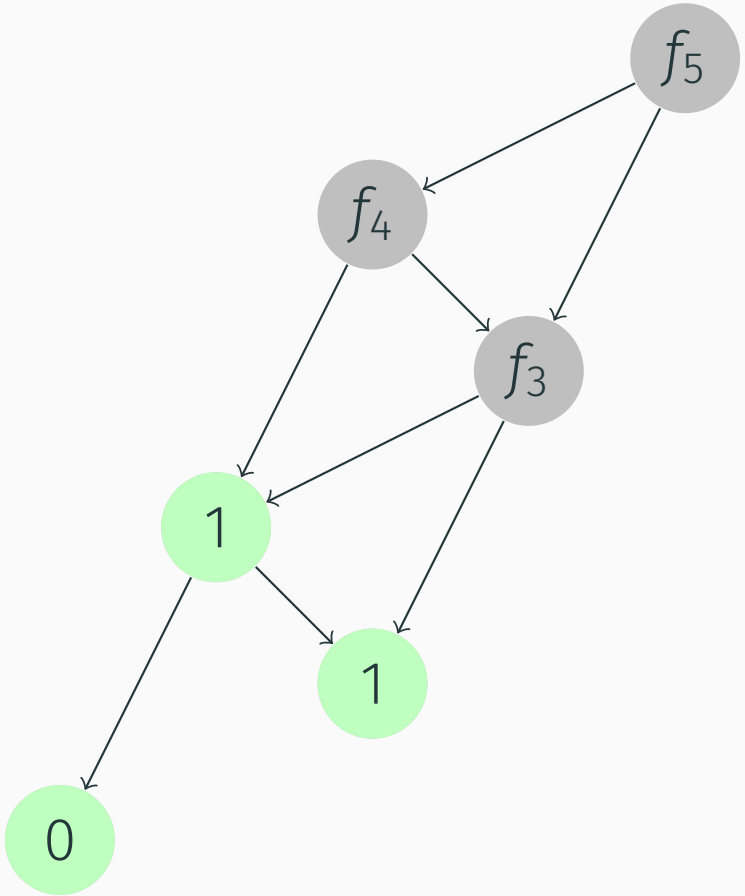
$f_1 = 1$



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

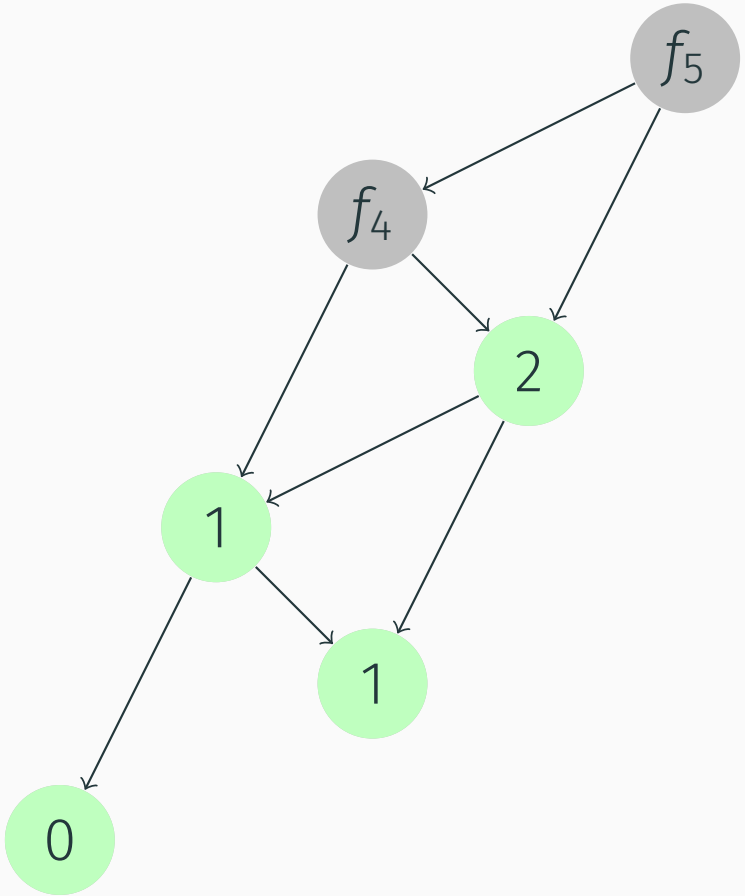
$$f_2 = f_1 + f_0$$



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

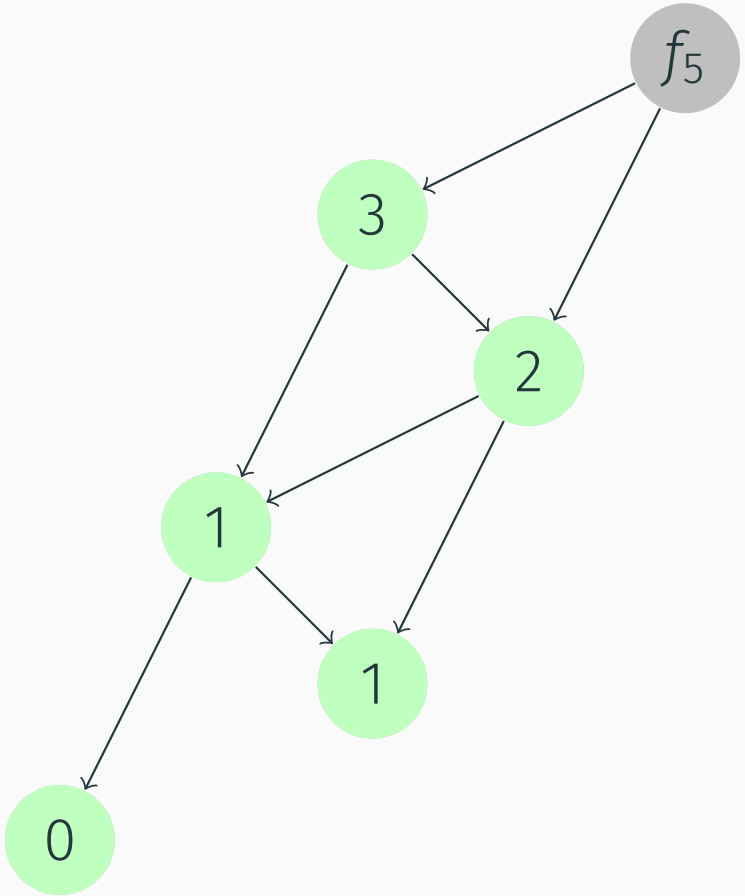
$$f_3 = f_2 + f_1$$



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

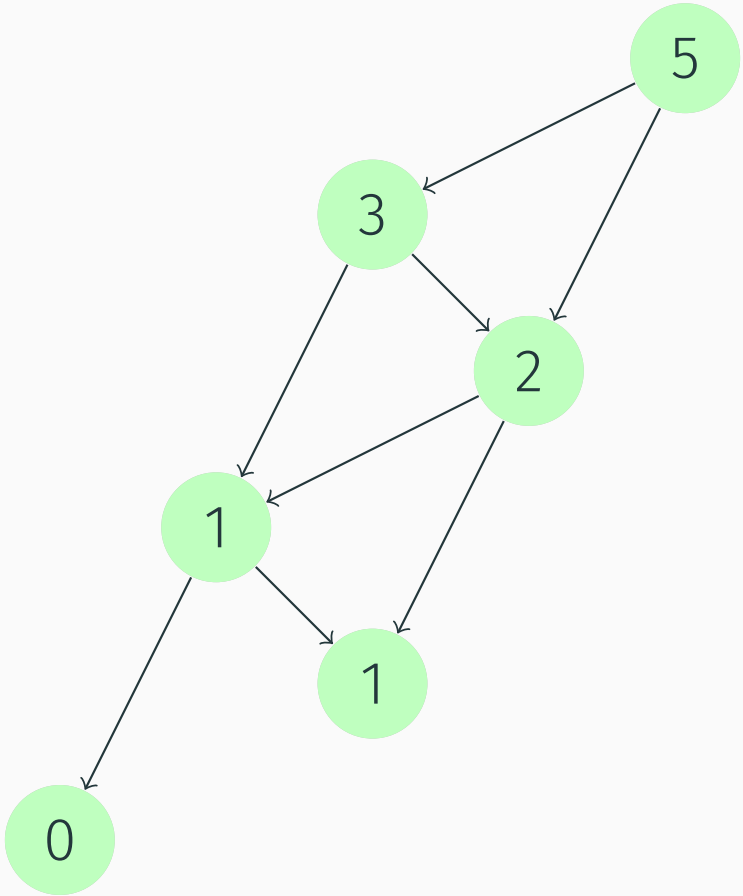
$$f_4 = f_3 + f_2$$



Fibonacci Folge: Die Lösung

Wir können bereits berechnete Werte verwenden um die höheren Werte zu berechnen:

$$f_5 = f_4 + f_3$$



Fibonacci Folge: Die Lösung

Wir berechnen alle Werte (einmal) von f_0 bis zu f_n:

```
int fib(int n) {
    vector<int> v;
    v.push_back(0);
    v.push_back(1);
    for(int i = 2; i <= n; i++) {
        v.push_back(v[v.size()-1] + v[v.size()-2]);
    }
    return v[n];
}
```

Fibonacci Folge: Die Lösung

Wir berechnen alle Werte (einmal) von f_0 bis zu f_n:

```
int fib(int n) {
    vector<int> v;
    v.push_back(0);
    v.push_back(1);
    for(int i = 2; i <= n; i++) {
        v.push_back(v[v.size()-1] + v[v.size()-2]);
    }
    return v[n];
}
```

Die Laufzeit ist jetzt nur noch...

Fibonacci Folge: Die Lösung

Wir berechnen alle Werte (einmal) von f_0 bis zu f_n:

```
int fib(int n) {
    vector<int> v;
    v.push_back(0);
    v.push_back(1);
    for(int i = 2; i <= n; i++) {
        v.push_back(v[v.size()-1] + v[v.size()-2]);
    }
    return v[n];
}
```

Die Laufzeit ist jetzt nur noch $\mathcal{O}(n)$.

Fibonacci Folge: Bonus Lösung

Bonus Lösung: Wir brauchen nicht $\mathcal{O}(n)$ Speicher.

```
int fib(int n) {
    int a = 0, b = 1;
    for(int i = 0; i < n; i++) {
        swap(a,b);
        b += a;
    }
    return a;
}
```

Fibonacci Folge: Bonus Lösung

Bonus Lösung: Wir brauchen nicht $\mathcal{O}(n)$ Speicher.

```
int fib(int n) {
    int a = 0, b = 1;
    for(int i = 0; i < n; i++) {
        swap(a,b);
        b += a;
    }
    return a;
}
```

Zur Vollständigkeit: Es gibt eine noch besser Lösung mit $\mathcal{O}(\log(n))$ Laufzeit

Das Rezept zum erstellen von guten DP Lösungen

Das Rezept zum erstellen von guten DP Lösungen

Das war eine einfache Aufgabe, aber das gleiche Schema ist auf viel schwierigere Probleme anwendbar. Wir werden nun dieses Schema generalisieren und es auf andere Probleme anwenden.

DP in vier Schritten

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

Zuerst denken, dann coden!

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

Zuerst denken, dann coden!

1. Teilprobleme finden.

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

Zuerst denken, dann coden!

1. Teilprobleme finden.
2. Eine Rekursionsformel finden die ein Teilproblem mithilfe von Lösungen von anderen Teilproblemen löst.

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

Zuerst denken, dann coden!

1. Teilprobleme finden.
2. Eine Rekursionsformel finden die ein Teilproblem mithilfe von Lösungen von anderen Teilproblemen löst.
3. Anfangsfälle, also Teilproblem die von keinen andere Teilproblem abhängen, finden.

DP in vier Schritten

Hier eine Variante um DP anzuwenden, in vier Schritten.

Zuerst denken, dann coden!

1. Teilprobleme finden.
2. Eine Rekursionsformel finden die ein Teilproblem mithilfe von Lösungen von anderen Teilproblemen löst.
3. Anfangsfälle, also Teilproblem die von keinen andere Teilproblem abhängen, finden.
4. Das relevante Teilproblem, das das Ursprungsproblem löst, finden.

DP in vier Schritten

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme:

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel:

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel: $f_i = f_{i-1} + f_{i-2}$.

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel: $f_i = f_{i-1} + f_{i-2}$.
3. Anfangsfälle:

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel: $f_i = f_{i-1} + f_{i-2}$.
3. Anfangsfälle: $f_0 = 0, f_1 = 1$.

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel: $f_i = f_{i-1} + f_{i-2}$.
3. Anfangsfälle: $f_0 = 0, f_1 = 1$.
4. Relevantes Teilproblem:

DP in vier Schritten für die Fibonacci Folge

Wie passt unsere Lösung für Fibonacci in die vier Schritte?

1. Teilprobleme: f_i.
2. Rekursionsformel: $f_i = f_{i-1} + f_{i-2}$.
3. Anfangsfälle: $f_0 = 0, f_1 = 1$.
4. Relevantes Teilproblem: f_n.

Implementierung einer DP Lösung

Teilproblem Reihenfolge

— Okay, wir sind den vier Schritten gefolgt. Aber wie verwende ich das Resultat nun um eine Lösunge zu programmieren?

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

1. Beginnen mit den Anfangsfällen, wir kennen die Lösung für diese bereits.

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

1. Beginnen mit den Anfangsfällen, wir kennen die Lösung für diese bereits.
2. Andere Teilprobleme lösen die nur von den Anfangsfällen abhängig sind.

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

1. Beginnen mit den Anfangsfällen, wir kennen die Lösung für diese bereits.
2. Andere Teilprobleme lösen die nur von den Anfangsfällen abhängig sind.
3. Weite Teilprobleme lösen die nun lösbar wurden.

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

1. Beginnen mit den Anfangsfällen, wir kennen die Lösung für diese bereits.
2. Andere Teilprobleme lösen die nur von den Anfangsfällen abhängig sind.
3. Weite Teilprobleme lösen die nun lösbar wurden.
4. Wenn wir auf das relevante Teilproblem stossen geben wir es zurück und sind fertig!

Teilproblem Reihenfolge

Die meisten Teilprobleme benötigen die Lösung anderer Teilproleme.
In welcher Reihenfolge lösen wir die Teilprobleme?

1. Beginnen mit den Anfangsfällen, wir kennen die Lösung für diese bereits.
2. Andere Teilprobleme lösen die nur von den Anfangsfällen abhängig sind.
3. Weite Teilprobleme lösen die nun lösbar wurden.
4. Wenn wir auf das relevante Teilproblem stossen geben wir es zurück und sind fertig!

Dies ist der schwierigste Teil beim Lösen von DP Problemen. Manchmal ist die Reihenfolge in der wir die Teilprobleme lösen offensichtlich, manchmal ist sie es nicht. Der beste Weg um sich daran zu gewöhnen ist viele solche Probleme zu lösen.

Anderes Beispiel: Hausrenovation

Hausrenovation: Aufgaben Beschreibung

Hausrenovation: Aufgaben Beschreibung

- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.

Hausrenovation: Aufgaben Beschreibung

- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.
- Deine Aufgabe ist es die Wand zu renovieren.

Hausrenovation: Aufgaben Beschreibung

- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.
- Deine Aufgabe ist es die Wand zu renovieren.
- Wenn die Wand an einer Stelle durch ein 1 beschrieben wird, ist dieses Stück Wand in einem guten Zustand, wir lassen es wie est ist.

Hausrenovation: Aufgaben Beschreibung

- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.
- Deine Aufgabe ist es die Wand zu renovieren.
- Wenn die Wand an einer Stelle durch ein 1 beschrieben wird, ist dieses Stück Wand in einem guten Zustand, wir lassen es wie est ist.
- Wenn es eine 0 hat, dann hat dieses Stück Wand Löcher und wir haben 2 Möglichkeiten: Das Stück renovieren oder ein Fenster einsetzen.

Hausrenovation: Aufgaben Beschreibung

- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.
- Deine Aufgabe ist es die Wand zu renovieren.
- Wenn die Wand an einer Stelle durch ein 1 beschrieben wird, ist dieses Stück Wand in einem guten Zustand, wir lassen es wie est ist.
- Wenn es eine 0 hat, dann hat dieses Stück Wand Löcher und wir haben 2 Möglichkeiten: Das Stück renovieren oder ein Fenster einsetzen.
- Damit ein Fenster an einer Position platziert werden kann muss die gesammte Spalte links und rechts davon ausschlieslich aus solider Wand bestehen (um die Stabilität des Hauses nicht zu gefährden).

Hausrenovation: Aufgaben Beschreibung

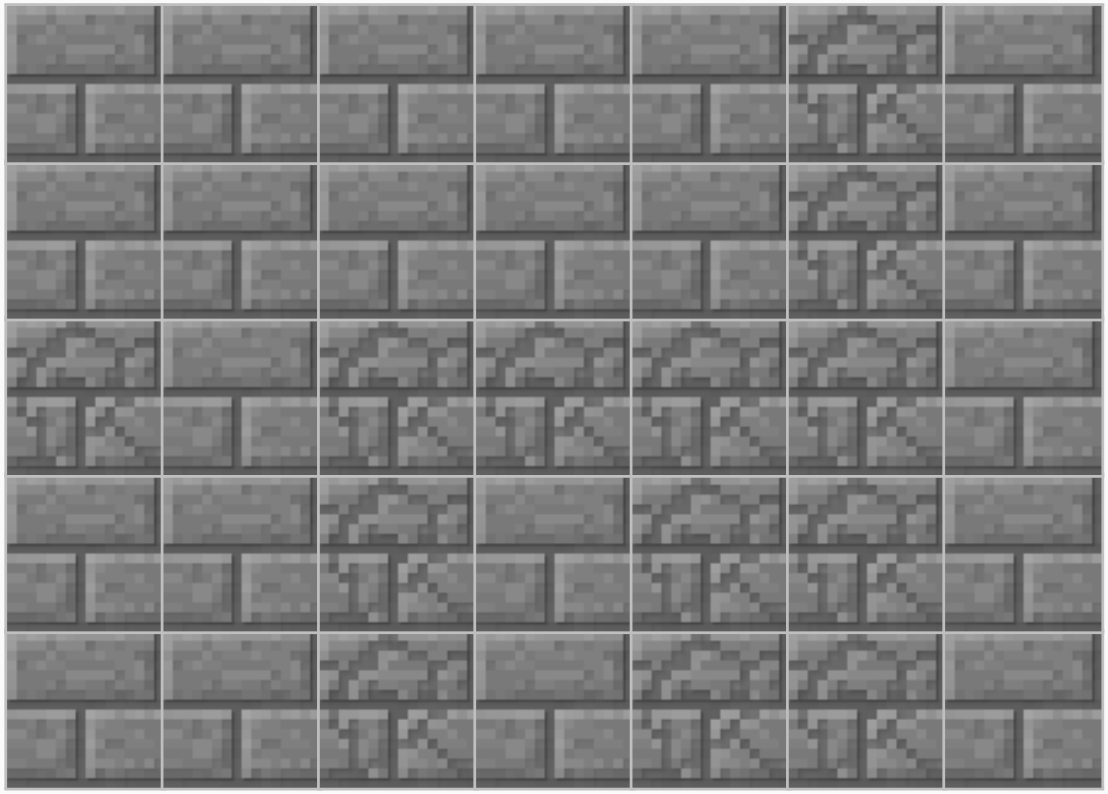
- Gegeben ist ein $n \times m$ Rechteck das eine Wand aus 1s und 0s representiert.
- Deine Aufgabe ist es die Wand zu renovieren.
- Wenn die Wand an einer Stelle durch ein 1 beschrieben wird, ist dieses Stück Wand in einem guten Zustand, wir lassen es wie est ist.
- Wenn es eine 0 hat, dann hat dieses Stück Wand Löcher und wir haben 2 Möglichkeiten: Das Stück renovieren oder ein Fenster einsetzen.
- Damit ein Fenster an einer Position platziert werden kann muss die gesammte Spalte links und rechts davon ausschlieslich aus solider Wand bestehen (um die Stabilität des Hauses nicht zu gefährden).
- Wir wollen die Anzahl an Fenster maximieren!

Hausrenovation: Beispiel

1	1	1	1	1	0	1
1	1	1	1	1	0	1
0	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	0	1	0	0	1

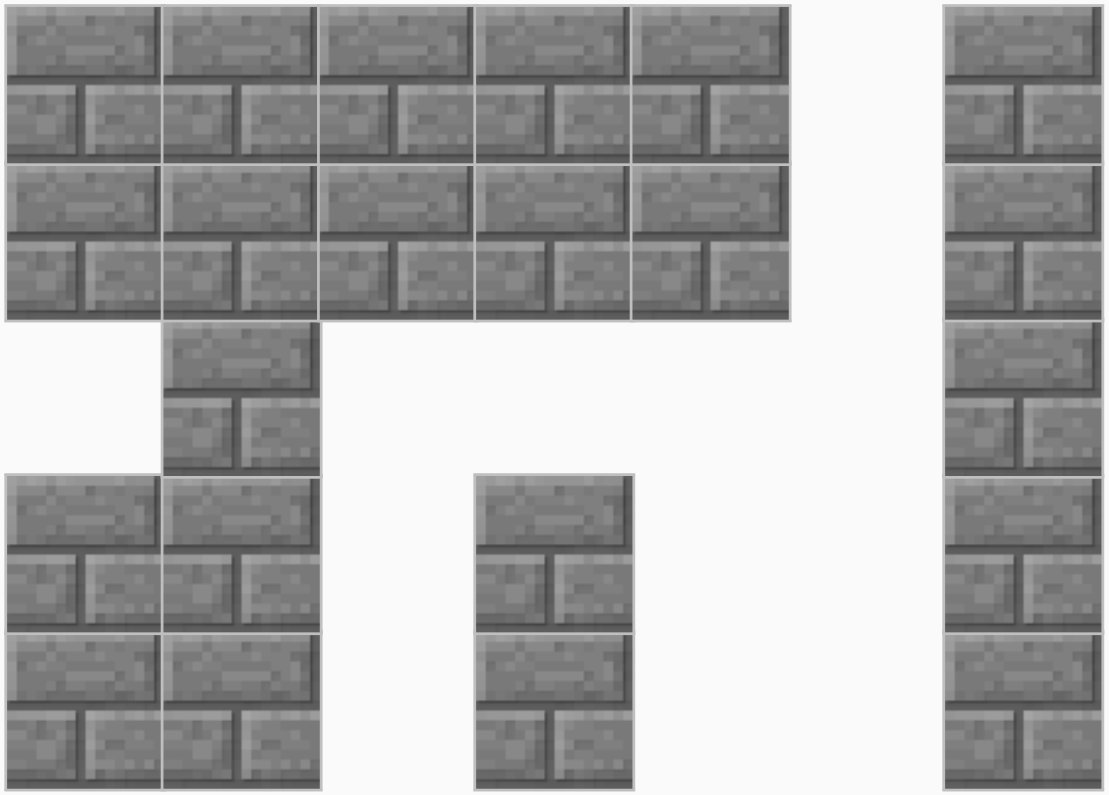
Die ist unser Input. Nullen sind kaputte Teile der Wand und Einsen sind solide Teile der Wand.

Hausrenovation: Beispiel



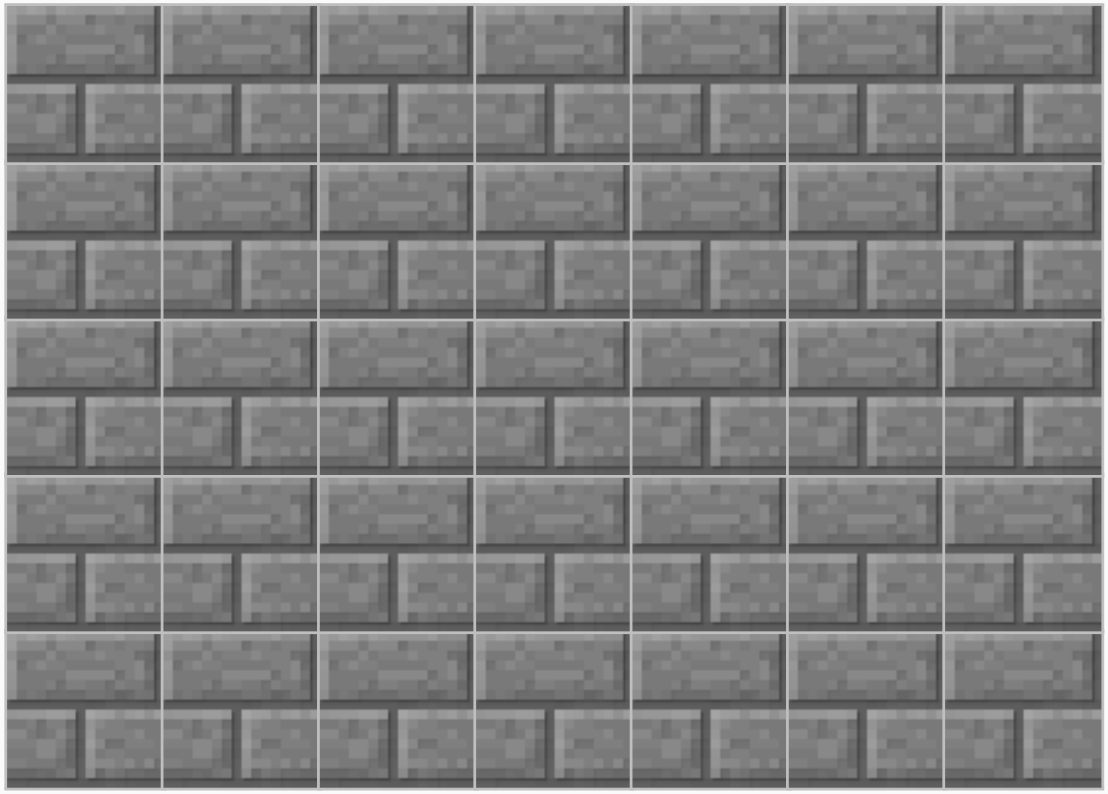
Hier ein Bild das das Beispiel etwas lesbarer macht.

Hausrenovation: Beispiel



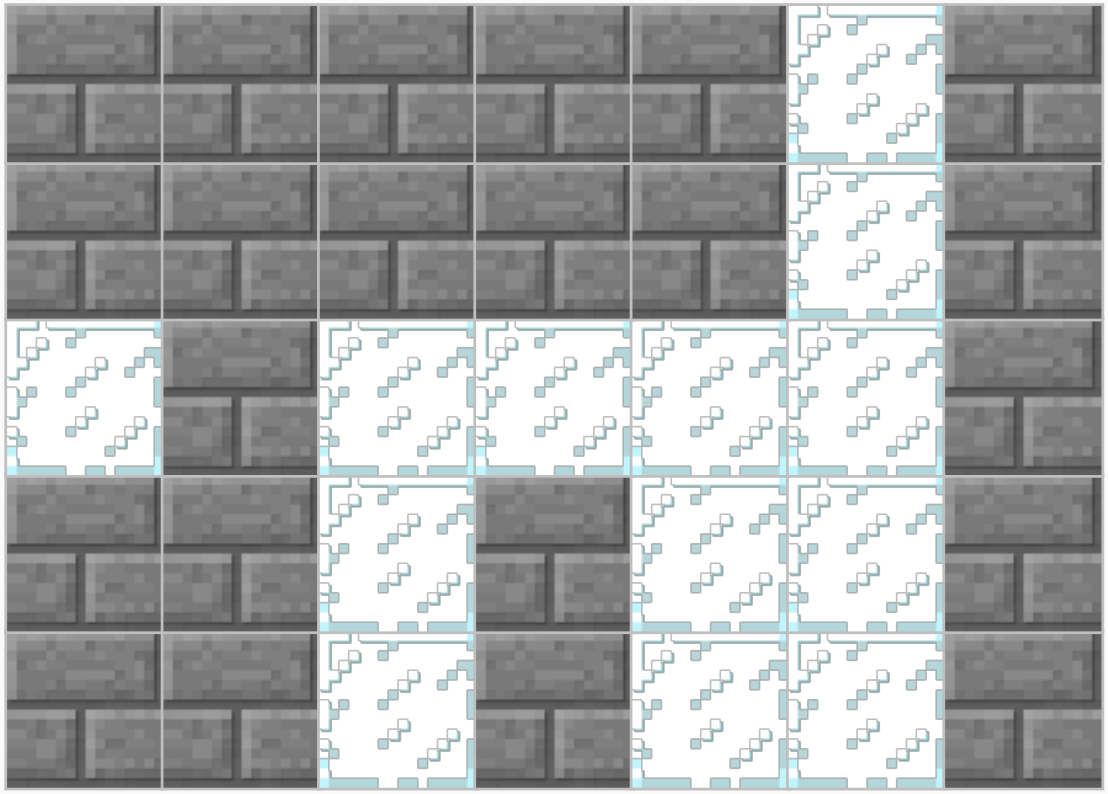
Für jedes Loch müssen wir entscheiden ob wir es mit Wand oder Fenster füllen.

Hausrenovation: Beispiel



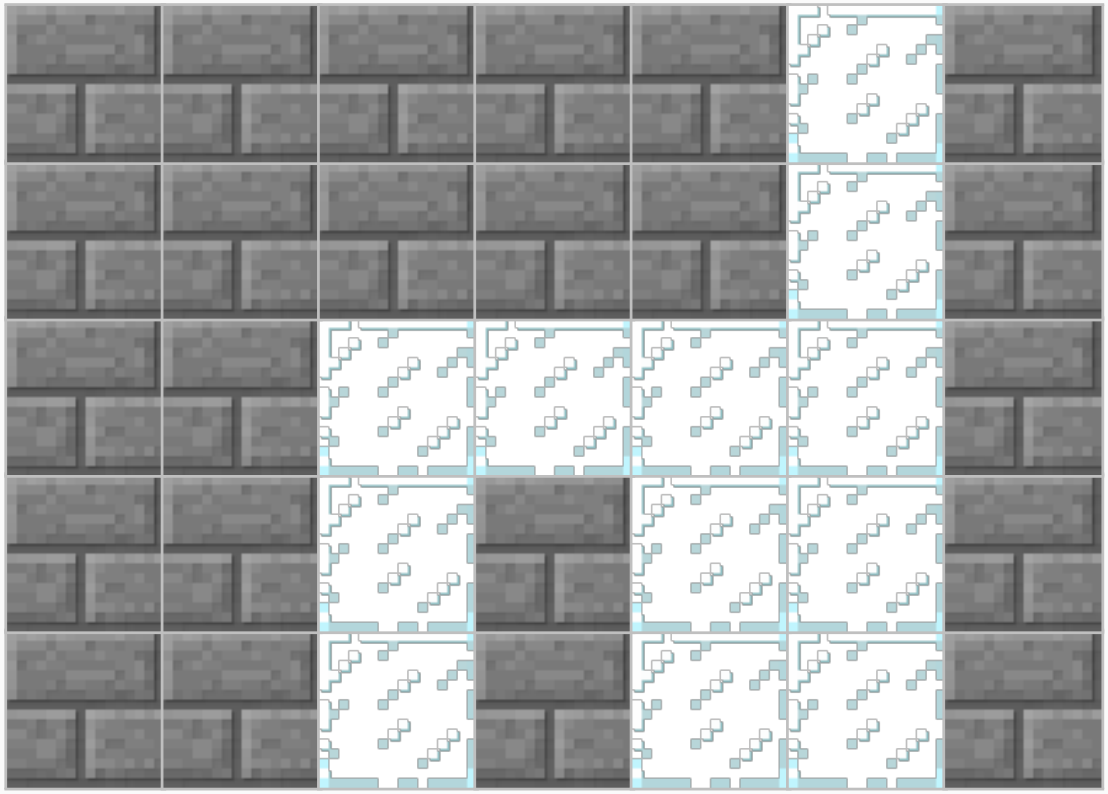
Es ist natürlich immer möglich einfach alle Löcher mit Wand zu füllen, aber dies ist wahrscheinlich nicht optimal (wir wollen möglichst viele Fenster).

Hausrenovation: Beispiel



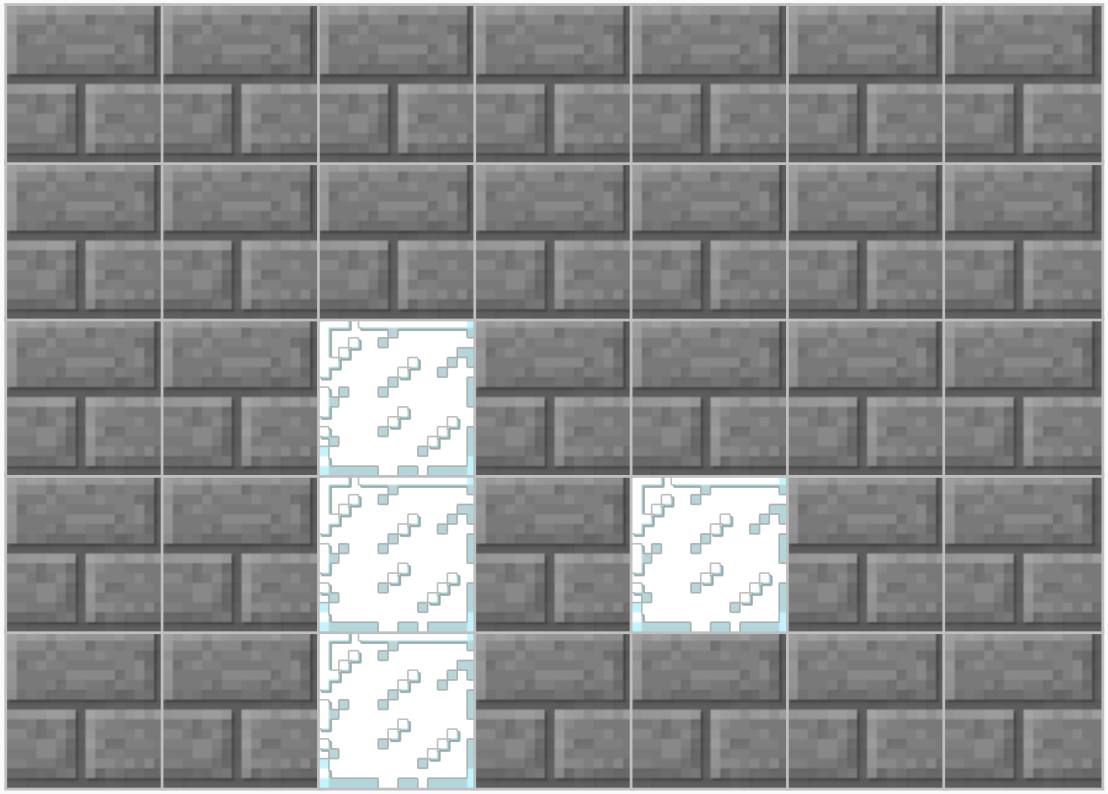
Einfach immer ein Fenster in jede Loch zu platzieren ist nicht immer möglich, da die Wand nicht mehr Stabil ist (gesammte Spalte links und rechts müssen aus Wand bestehen)

Hausrenovation: Beispiel



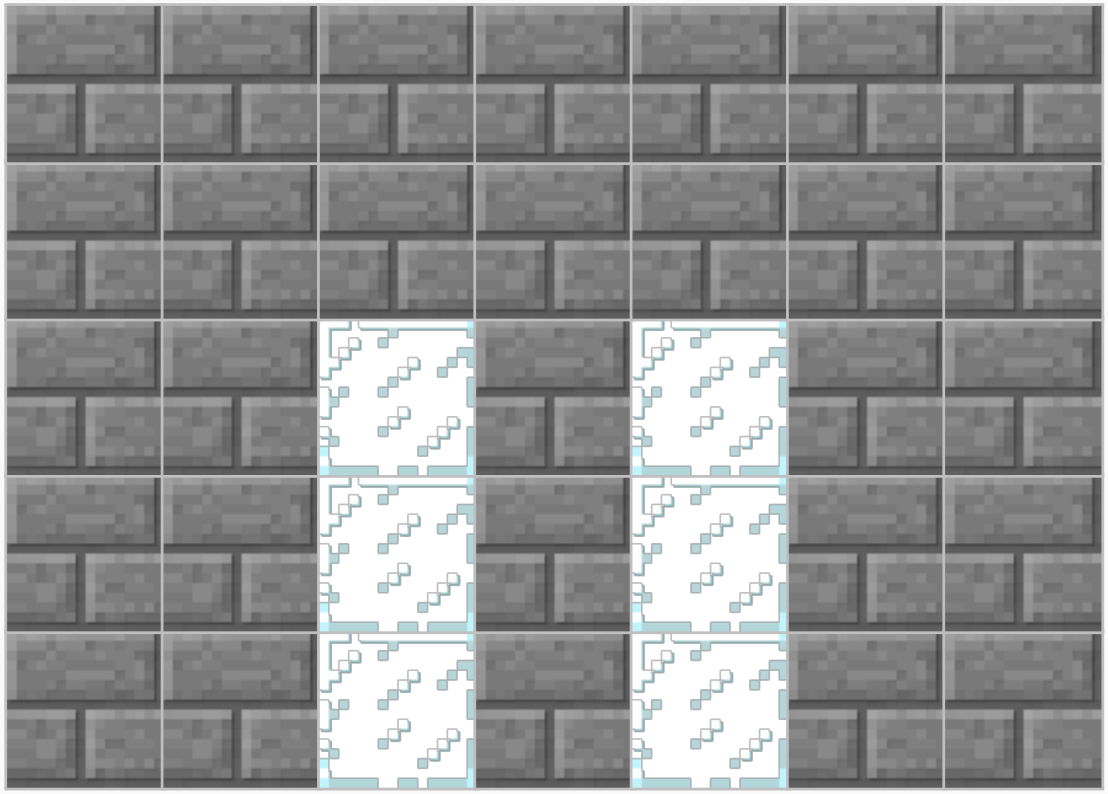
Wir können kein Fenster in der Spalte ganz links haben, weil links davon keine Spalte aus Wand ist. Es zu entfernen reicht nicht aus, die Wand ist immer noch instabil.

Hausrenovation: Beispiel



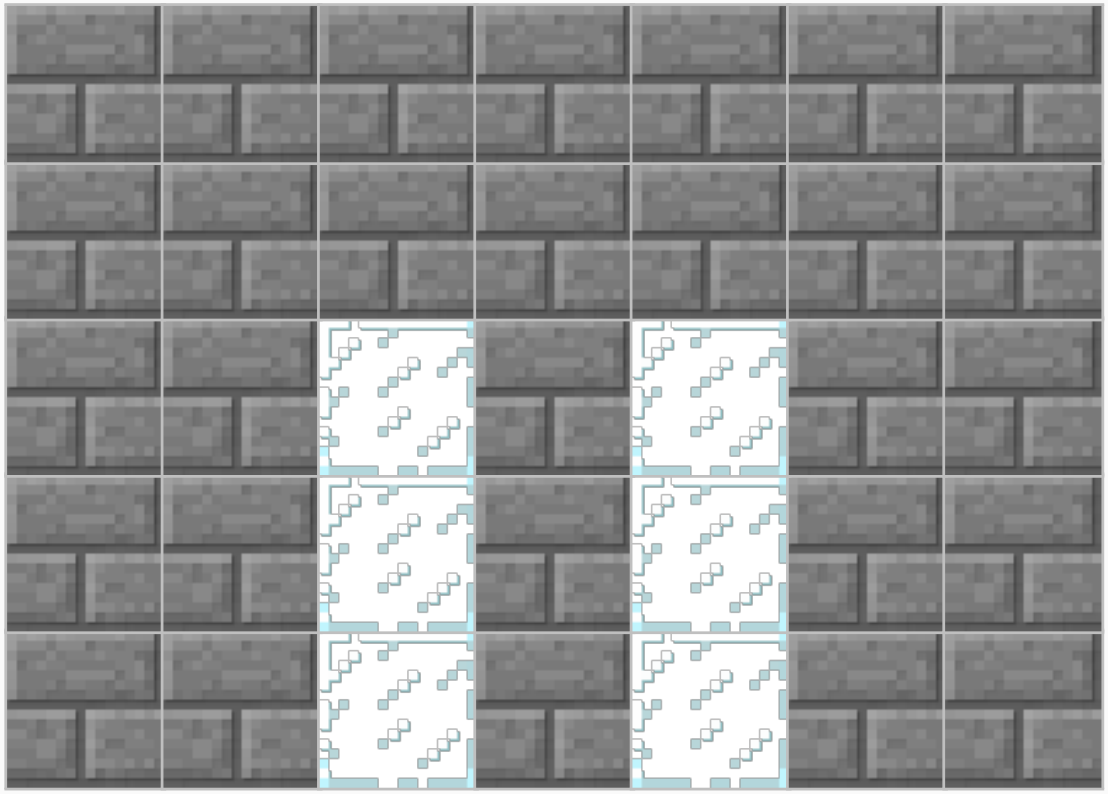
Wir müssen immer zwei vollständige Spalten um das Fenster herum bauen.

Hausrenovation: Beispiel



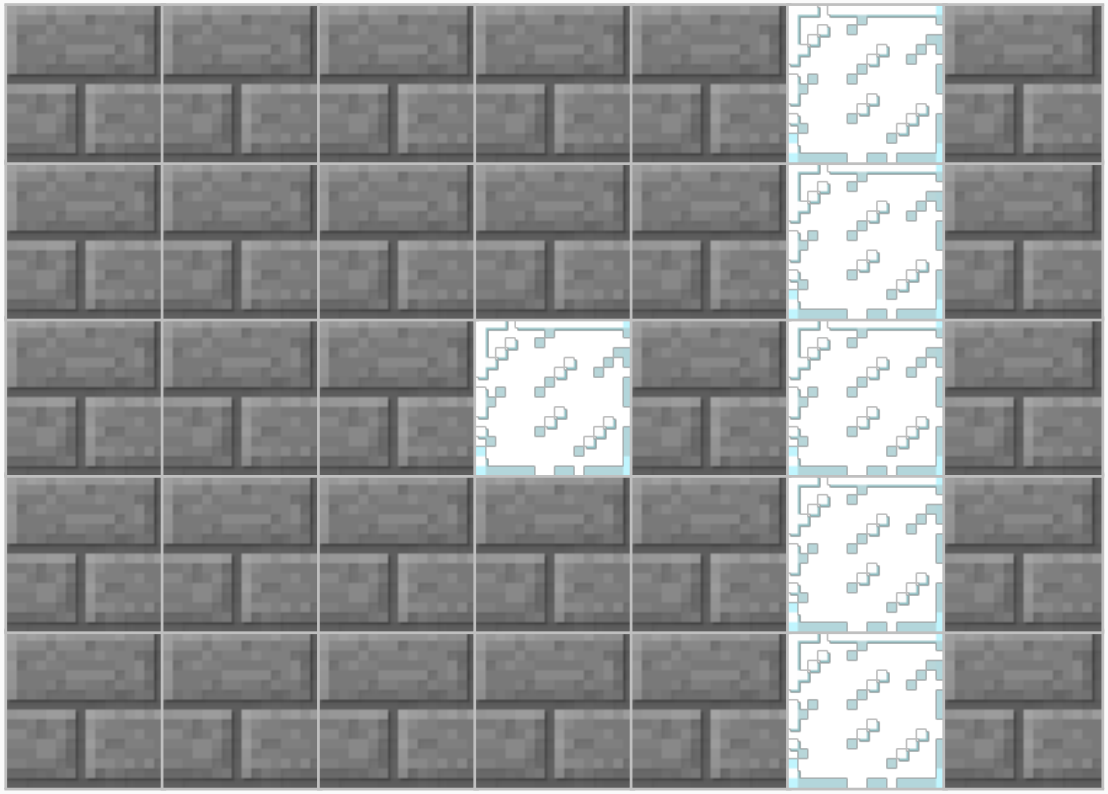
Wenn wir in einer Spalte ein Fenster bauen macht es keinen Sinn in der selben Spalte eine Wand zu bauen, weil wir keine Möglichkeit aufgeben ein Fenster zu bauen.

Hausrenovation: Beispiel



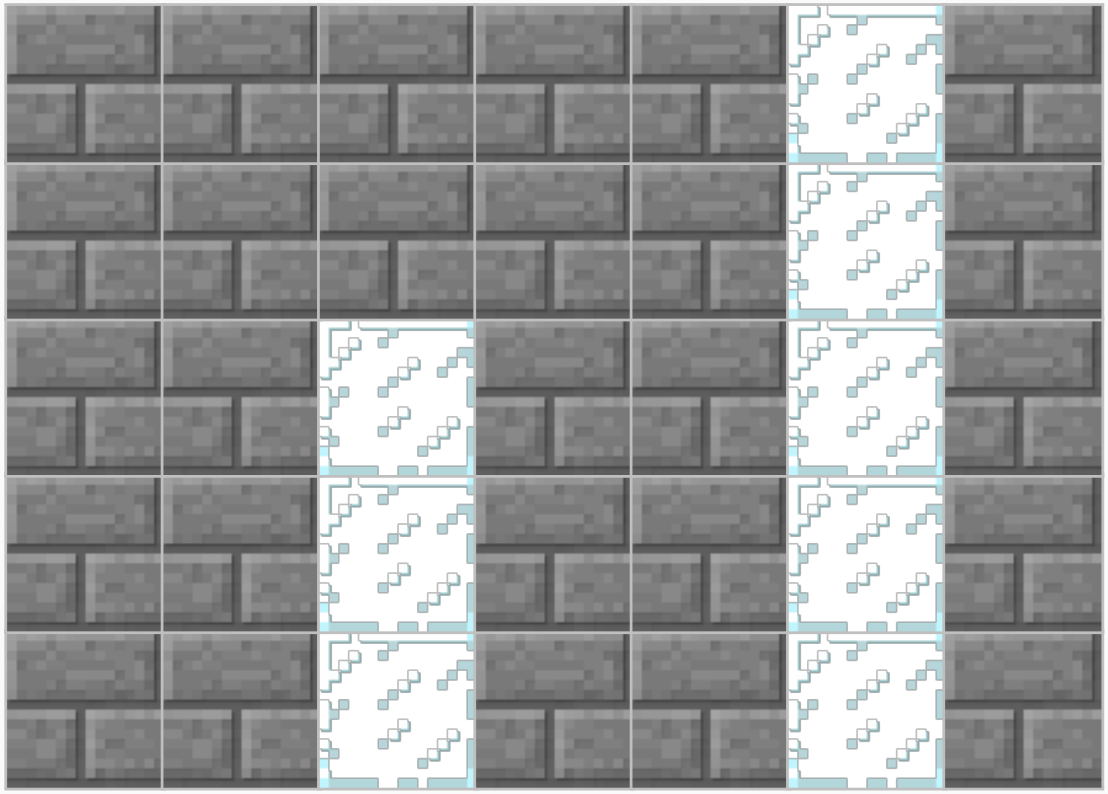
Das ist nicht optimal: 6 Fenster.

Hausrenovation: Beispiel



Das ist nicht optimal: 6 Fenster.

Hausrenovation: Beispiel



Das ist optimal: 8 Fenster.

Hausrenovation: Vorberechnen

Das Problem wird einfacher wenn wir diese Observationen verwenden bevor wir anfangen uns über DP gedanken zu machen.

Hausrenovation: Vorberechnen

Das Problem wird einfacher wenn wir diese Observationen verwenden bevor wir anfangen uns über DP gedanken zu machen.

- Wir reduzieren das Problem zu einem eindimensionales Problem: Berechne die Anzahl möglichen Fenster in jeder Spalte (wir werden immer alle oder keines davon bauen).

Hausrenovation: Vorberechnen

Das Problem wird einfacher wenn wir diese Observationen verwenden bevor wir anfangen uns über DP gedanken zu machen.

- Wir reduzieren das Problem zu einem eindimensionales Problem: Berechne die Anzahl möglichen Fenster in jeder Spalte (wir werden immer alle oder keines davon bauen).
- Wir setzen die Anzahl an möglichen Fenster in der Spalte ganz links und ganz rechts auf 0.

Hausrenovation: Vorberechnen

Das Problem wird einfacher wenn wir diese Observationen verwenden bevor wir anfangen uns über DP gedanken zu machen.

- Wir reduzieren das Problem zu einem eindimensionales Problem: Berechne die Anzahl möglichen Fenster in jeder Spalte (wir werden immer alle oder keines davon bauen).
- Wir setzen die Anzahl an möglichen Fenster in der Spalte ganz links und ganz rechts auf 0.

Wir berechnen ein Array $w[m]$. Für jedes $0 \leq i < m$, sei $w[i]$ die Anzahl möglichen Fenster in der i-ten Spalte.

Hausrenovation: Vorberechnen

```
int main(){
    int n, m; // n = height, m = width
    cin >> n >> m;
    // w[i] = number of holes in the i-th column
    vector<int> w(m, 0);
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            int c;
            cin >> c;
            w[j] += c == 0; // this section is hole
        }
    }
    w[0] = 0; w[m-1] = 0; // no window on the edges
    ... // actual computation
```

Hausrenovation: Vorberechnen

Wir haben das Problem das in 2D war und komplex aussah in ein 1D Problem, das einfacher scheint, transformiert.
Das neue Problem:

Hausrenovation: Vorberechnen

Wir haben das Problem das in 2D war und komplex aussah in ein 1D Problem, das einfacher scheint, transformiert.
Das neue Problem:

- Gegeben ist ein array von Ganzzahlen w der Länge n.

Hausrenovation: Vorberechnen

Wir haben das Problem das in 2D war und komplex aussah in ein 1D Problem, das einfacher scheint, transformiert.
Das neue Problem:

- Gegeben ist ein array von Ganzzahlen w der Länge n.
- Was ist die maximale Summe der Elemente wenn wir nie 2 Elemente addieren die nebeneinander sind.

Hausrenovation: DP in vier Schritten

Wie können wir das Problem mit den vier Schritten modelieren?

Hausrenovation: Teilprobleme

Unser Teilproblem wird sein

Hausrenovation: Teilprobleme

Unser Teilproblem wird sein s_i, die maximale Anzahl von Fenster wenn wir nur die ersten $i + 1$ Spalten vorwenden ($0 \leq i < m$).

Hausrenovation: Rekursionsformel

Wenn wir s_i, die Lösung für i, berechnen wollen haben wir 2 Möglichkeiten:

Hausrenovation: Rekursionsformel

Wenn wir s_i, die Lösung für i, berechnen wollen haben wir 2 Möglichkeiten:

1. Wir bauen keine Fenster in der *i*-ten Spalte.

Hausrenovation: Rekursionsformel

Wenn wir s_i, die Lösung für i, berechnen wollen haben wir 2 Möglichkeiten:

1. Wir bauen keine Fenster in der i-ten Spalte.
2. Wir baune alle möglichen Fester in der i-ten Spalte. In diesem Fall können wir keine Fenster in der $(i-1)$-ten Spalte haben.

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i =$

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i = s_{i-1}$.

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i = s_{i-1}$.
2. Wenn wir ein Fester in der i-ten Spalte bauen, können wir kein Fenster in der $(i-1)$-ten Spalte haben, aber wir können die Fenster in der i-ten Spalte addieren. Also $s_i =$

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i = s_{i-1}$.
2. Wenn wir ein Fester in der i-ten Spalte bauen, können wir kein Fenster in der $(i-1)$-ten Spalte haben, aber wir können die Fenster in der i-ten Spalte addieren. Also $s_i = s_{i-2} + w[i]$.

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i = s_{i-1}$.
2. Wenn wir ein Fester in der i-ten Spalte bauen, können wir kein Fenster in der $(i-1)$-ten Spalte haben, aber wir können die Fenster in der i-ten Spalte addieren. Also $s_i = s_{i-2} + w[i]$.

Da wir s_i maximieren wollen behalten wir den hösten der

beiten Werten.

Hausrenovation: Rekursionsformel

Wie wird das in unsere Formel übertragen?

1. Wenn wir kein Kenster in der i-ten Spalte bauen, $s_i = s_{i-1}$.
2. Wenn wir ein Fester in der i-ten Spalte bauen, können wir kein Fenster in der $(i-1)$-ten Spalte haben, aber wir können die Fenster in der i-ten Spalte addieren. Also $s_i = s_{i-2} + w[i]$.

Da wir s_i maximieren wollen behalten wir den hösten der

beiten Werten.

$$S_i = \max(S_{i-1}, S_{i-2} + W[i])$$

Hausrenovation: Anfangsfälle

Wir brauchen zwei Anfangsfälle weil unsere Rekursionsformel zwei Schritte zurück geht.

Hausrenovation: Anfangsfälle

Wir brauchen zwei Anfangsfälle weil unsere Rekursionsformel zwei Schritte zurück geht.

- $s_0 = w[0] = 0$ (wir können nie Fenster in der Spalte am linken Rand bauen).

Hausrenovation: Anfangsfälle

Wir brauchen zwei Anfangsfälle weil unsere Rekursionsformel zwei Schritte zurück geht.

- $s_0 = w[0] = 0$ (wir können nie Fenster in der Spalte am linken Rand bauen).
- $s_1 = w[1]$ (es hat in der ersten Spalte nie Fenster, deshalb ist es immer optimal in der zweiten Spalte Fenster zu bauen).

Hausrenovation: Relevantes Teilproblem

Das relevante Teilproblem ist dasjenige das alle Spalten der Wand beinhaltet also s...

Hausrenovation: Relevantes Teilproblem

Das relevante Teilproblem ist dasjenige das alle Spalten der Wand beinhaltet also s_{m-1}.

Hausrenovation: Relevantes Teilproblem

Das relevante Teilproblem ist dasjenige das alle Spalten der Wand beinhaltet also $s_{m-1} (= s_{m-2}$, wenn $m > 1)$.

Hausrenovation: Langsame Lösung

Wie bei der Fibonacci Folge könnten wir diese Rekursionsformel und Anfangsfälle benutzen um eine langsame, aber korrekte Lösung zu erhalten.

```
int s(int i, vector<int> &w) {
    if(i < 2) return w[i]; // base cases
    return max(s(i-1, w), s(i-2, w) + w[i]);
}
```

Hausrenovation: Berechnungen

Wir wollen aber Teilprobleme nicht mehrmals berechnen. Wir berechnen Sie einfach in der Reihenfolge in der sie gebraucht werden und speichern sie.

Hausrenovation: Berechnungen

Wir wollen aber Teilprobleme nicht mehrmals berechnen. Wir berechnen Sie einfach in der Reihenfolge in der sie gebraucht werden und speichern sie.
Die Reihenfolge in welcher sie gebraucht werde ist, wie bereits bei der Fibonacci Folge, ziemlich offensichtlich. Jedes Teilproblem hängt nur von vorherigen Teilproblemen ab, also können wir sie einfach in aufsteigender Reihenfolge lösen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	Optimale Anzahl an Fenstern für die erste 1 Spalte
1	0	Optimale Anzahl an Fenstern für die ersten 2 Spalten
2	3	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	w[0] (Anfangsfall)
1	0	Optimale Anzahl an Fenstern für die ersten 2 Spalten
2	3	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	Optimale Anzahl an Fenstern für die ersten 2 Spalten
2	3	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	w[1] (Anfangsfall)
2	3	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	$\max(v_1, v_0 + w[2])$
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	$\max(v_2, v_1 + w[3])$
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	Optimale Anzahl an Fenstern für die ersten 5 Spalten
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	$\max(v_3, v_2 + w[4])$
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	6
5	5	Optimale Anzahl an Fenstern für die ersten 6 Spalten
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	6
5	5	$\max(v_4, v_3 + w[4])$
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	6
5	5	8
6	0	Optimal number of windows for all Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	6
5	5	8
6	0	$\max(v_5, v_4 + w[4])$

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Schritt für Schritt: Ein Beispiel

i	$w[i]$	v_i
0	0	0
1	0	0
2	3	3
3	1	3
4	3	6
5	5	8
6	0	8

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	Optimale Anzahl an Fenstern für die erste 1 Spalte
1	Optimale Anzahl an Fenstern für die ersten 2 Spalten
2	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	Optimale Anzahl an Fenstern für die ersten 5 Spalten
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	Optimale Anzahl an Fenstern für die ersten 2 Spalten
2	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	Optimale Anzahl an Fenstern für die ersten 5 Spalten
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	$w[1]$ (Anfangsfall)
2	Optimale Anzahl an Fenstern für die ersten 3 Spalten
3	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	Optimale Anzahl an Fenstern für die ersten 5 Spalten
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	$w[1]$ (Anfangsfall)
2	$max(v_1, v_0 + w[2])$
3	Optimale Anzahl an Fenstern für die ersten 4 Spalten
4	Optimale Anzahl an Fenstern für die ersten 5 Spalten
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	$w[1]$ (Anfangsfall)
2	$max(v_1, v_0 + w[2])$
3	$max(v_2, v_1 + w[3])$
4	Optimale Anzahl an Fenstern für die ersten 5 Spalten
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	$w[1]$ (Anfangsfall)
2	$max(v_1, v_0 + w[2])$
3	$max(v_2, v_1 + w[3])$
4	$max(v_3, v_2 + w[4])$
...	...
`m-1`	Optimale Anzahl an Fenstern für alle Spalten

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Berechnung

Berechnung, Schrit für Schritt

i	v_i
0	$w[0]$ (Anfangsfall)
1	$w[1]$ (Anfangsfall)
2	$max(v_1, v_0 + w[2])$
3	$max(v_2, v_1 + w[3])$
4	$max(v_3, v_2 + w[4])$
...	...
`m-1`	$max(v_m - 2, v_m - 3 + w[m-1])$ (relevantes Teilproblem)

Das können wir mit einer **`for`** Schleife machen.

Hausrenovation: Implementation

```
    ... // precomputation
    vector<int> s(m);
    s[0] = 0; s[1] = w[1]; // base cases
    for(int i = 2; i < m; i++)
        s[i] = max(s[i-1], s[i-2] + w[i]);
    cout << s[m-1] << '\n'; // relevant subproblem
}
```

Laufzeit Analyse

Was ist die Laufzeit von dieser Lösung?

Laufzeit Analyse

Was ist die Laufzeit von dieser Lösung? $\mathcal{O}(nm)$ (wegen den Grösse der Eingabedaten; ohne die Vorberechnung brauchen wir nur $\mathcal{O}(m)$).

Fazit

Wann ist DP anwendbar?

- Wenn wir das Problem in Teilprobleme zerteilen können.

Wann ist DP anwendbar?

- Wenn wir das Problem in Teilprobleme zerteilen können.
- Wenn sich die Teilprobleme überlagern.

Wann ist DP anwendbar?

- Wenn wir das Problem in Teilprobleme zerteilen können.
- Wenn sich die Teilprobleme überlagern.
- Viele Optimierungsprobleme sind mit Dynamischer Programmierung lösbar.

Wann ist DP anwendbar?

- Wenn wir das Problem in Teilprobleme zerteilen können.
- Wenn sich die Teilprobleme überlagern.
- Viele Optimierungsprobleme sind mit Dynamischer Programmierung lösbar.
- DP wird oft bei Problemen mit mehr als einer Dimenson verwendet. In diesem Fall ist es etwas schwieriger die richtige Reihenfolge zum lösen der Teilprobleme zu finden.

Einige Anmerkungen zur Memoisation

Es ist möglich die langsame Lösung relativ einfach in eine DP Lösung umzuwandeln indem wir uns bereits bereits berechnete Werte merken.

```
vector<int> m(MAX_N);
int fib(int n) {
    if(n<2) return n;
    if(m[n]) return m[n];
    return m[n] = fib(n-1) + fib(n-2);
}
```

Einige Anmerkungen zur Memoisation

Es ist möglich die langsame Lösung relativ einfach in eine DP Lösung umzuwandeln indem wir uns bereits bereits berechnete Werte merken.

```
vector<int> m(MAX_N);
int fib(int n) {
    if(n<2) return n;
    if(m[n]) return m[n];
    return m[n] = fib(n-1) + fib(n-2);
}
```

Dieser Ansatz ”top-down” ist allerdings oft weniger gut da wir, wie bei der Fibonacci Folge, nicht alle vorherigen Werte speichern müssen und die Rekursion zu Problemen führen kann (weniger effizient, stack limit exceeded).

Bottom-up

Der Ansatz den wir verwendet habe heisst ”bottom-up” und es ist besser sich besser direkt mit dem vertraut zu machen.

Wie werde ich DP Champion?

- DP ist schwierig

Wie werde ich DP Champion?

- DP ist schwierig für die meisten.

Wie werde ich DP Champion?

- DP ist schwierig für die meisten.
- Das Konzept ist einfach, aber...

Wie werde ich DP Champion?

- DP ist schwierig für die meisten.
- Das Konzept ist einfach, aber es als Lösung für ein Probleme zu identifizieren und die Lösung zu implementieren ist schwierig.

Wie werde ich DP Champion?

- DP ist schwierig für die meisten.
- Das Konzept ist einfach, aber es als Lösung für ein Probleme zu identifizieren und die Lösung zu implementieren ist schwierig.
- Immer zuerst denken, dann programmieren!

Wie werde ich DP Champion?

- DP ist schwierig für die meisten.
- Das Konzept ist einfach, aber es als Lösung für ein Probleme zu identifizieren und die Lösung zu implementieren ist schwierig.
- Immer zuerst denken, dann programmieren!
- Am allerwichtigsten: Üben, üben, üben!

Was kommt als nächstes:
DP Aufgaben auf den Grader lösen.
Nächste Vorlesung: Subset Sum.