Aufgabe 1: Vermischtes ($2+2+2+4+3+1=14$ Punkte)

Geben Sie **kurze** präzise Antworten auf die Fragen.

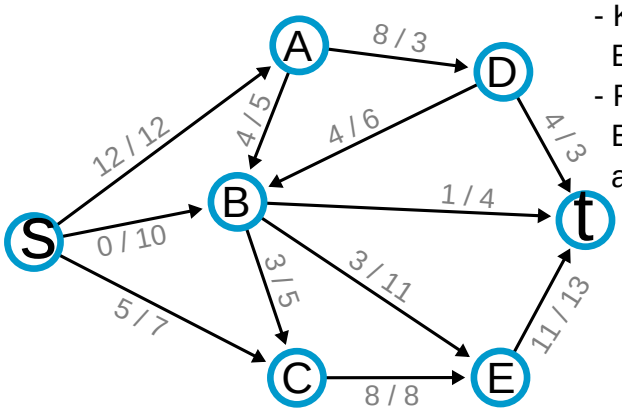
(a) Definieren Sie, was es heißt, dass ein Graph zusammenhängend ist.

Ein Graph G=(V,E) ist zusammenhängend, genau dann wenn zwischen zwei beliebigen Knoten v1, v2 in V ein Weg existiert

(b) Definieren Sie die Kapazität eines s von t trennenden Schnittes in einem Flussgraphen.

Die Kapazität des Schnitts S, T ist die Summe der Kapazitäten aller S verlassenden Kanten.

(c) Der 'Fluss', der in dem folgenden Flussgraphen eingezeichnet ist, enthält einen oder mehrere Verstöße gegen die Bedingungen der Flussdefinition. Listen Sie neben dem Graphen alle Verstöße auf. Geben Sie jeweils an, welche Bedingung nicht erfüllt ist.



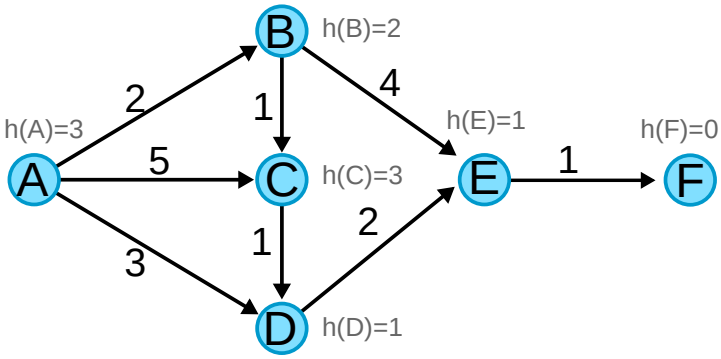
- Kapazitätserhaltung: A->D fließen 8 Einheiten, obwohl das Maximum 3 ist.
- Flusserhaltung: in B fließen 8 Einheiten (Zufluss), aus B heraus fließen aber nur 7 Einheite (Abfluss)

(d) Beschreiben Sie die Breitensuche in Pseudocode. Geben Sie dabei an, welche Datenstrukturen verwendet werden.
Hinweis: Der Code braucht nur die von einem Startknoten s erreichbaren Knoten zu bestimmen.

```
start_node   0          // Ganzzahl
visited      {false} * |V|   // Array von Booleans von Länge |V|
visited[start_node]          true
Enqueue(Q, start_node)    // Queue von Ganzzahlen
while Q        do
  v <- Dequeue(Q)
  for each w with w = (v, w)              E and ¬visited[w] do
    visited[w]            true
    Enqueue(Q, w)
  endfor
endwhile

return visited
```

(e) In dem folgenden gewichteten Graphen sind die Werte einer Heuristik h zur Suche der kürzesten Wege von A nach F angegeben. Ist die Heuristik zulässig? Ist sie konsistent? Begründen Sie.



Die Heuristik ist zulässig, da sie
die verbleibende Distanz nie überschätzt
Die Heuristik ist nicht konsistent.
Es gilt zwar h(F)=0, aber nicht immer
h(v) weight(v w)+h(w).
Betrachte v=C, w=D, dann gilt
3 1+1

(f) Welche Wachstumsordnung (in O Notation) beschreibt die Laufzeit der folgenden Methode f(int N, int x) am genausten? (ohne Begr ündung)

```
1 public static int f(int n, int x) {
2       for (int i = n; i > 0; i /= 2) {
3             for (int j = 0; j < i; j++) {
4                   x += j;
5             }
6       }
7       return x;
8 }
```

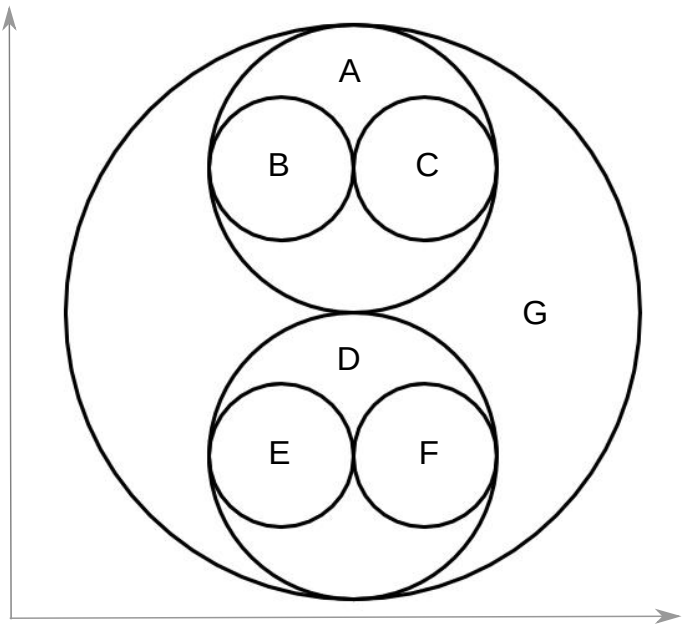
O(n^2)

Aufgabe 2: Java - Rekursion ($7 + 1 + 3 = 11$ Punkte)

Im Folgenden ist eine Klasse gegeben, welche die Kreise so zeichnet, wie sie in der Abbildung 1 zu sehen sind. Gehen Sie davon aus, dass StdDraw.circle(x,y,r) einen Kreis mit dem Radius r um den Punkt (x, y) zeichnet.

```
1  public class Circular {
2      int n;
3      public Circular(int n) {
4          this.n=n;
5      }
6      public void draw(double x, double y, double r) {
7          vert(n, x, y, r);
8      }
9      public static void horiz(int n, double x, double y, double r) {
10         if (n == 0) return;
11         vert(n - 1, x + r/2, y, r/2); // recur right
12         vert(n - 1, x - r/2, y, r/2); // recur left
13         StdDraw.circle(x, y, r);
14     }
15     public static void vert(int n, double x, double y, double r) {
16         if (n == 0) return;
17         horiz(n - 1, x, y + r/2, r/2); // recur up
18         StdDraw.circle(x, y, r);
19         horiz(n - 1, x, y - r/2, r/2); // recur down
20     }
21     public static void main(String[] args) {
22         Circular c= new Circular(3);
23         c.draw(0.5, 0.5, 0.5);
24     }
25 }
```

(a) Vervollständigen Sie die Abbildung 1, indem Sie die Buchstaben, die den Kreisen zugeordnet sind, in der Reihenfolge in die graue Tabelle eintragen, in der sie von der Klasse Circular gezeichnet werden.



1.	C
2.	B
3.	A
4.	G
5.	F
6.	E
7.	D

Abbildung 1

(b) Wie viele Kreise werden gezeichnet, wenn Sie `vert(4, .5, .5, .5);` aufrufen?

15 Kreise

(c) Angenommen es gäbe eine Klasse `Mandala`, die von der Klasse `Circular` erbt. Vervollst ändigen Sie die Lücken und schreiben Sie einen Konstruktor `Mandala(int n)` f ür diese unfertige Klasse. Die einzige Anforderung an den Konstruktor ist, dass damit die Klasse kompiliert.

```
public _____class_____ Mandala _____extends Circular_____{

  public Mandala(int n) {
    super(n);
  }


}
```


Aufgabe 3: Hashing ($2 + 5 + 3 = 11$ Punkte)

Betrachten Sie den Ausschnitt des Codes der Klasse Location. F ür diese Aufgabe wird die Hashfunktion

$$h(x) = abs(x) \mod 5$$

benutzt. Dabei ist $abs(x)$ der Absolutbetrag von x.

(a) Berechnen und notieren Sie die fehlenden Hashcodes und Hashadressen in der folgenden Tabelle. Die 'Namen' der Schlüssel in Spalte 1 dienen der einfacheren Notation der Schlüssel in Aufgabenteil (b).

```
1 public class Location {
2       public int x;
3       public int y;
4
5       // ...
6       public int hashCode() {
7             return 11    * x + y + 1;
8       }
9 }
```

Name	Schlüssel	Hashcode	Hashadresse
A	$(2, 7)$	30	0
B	$(3, 5)$	39	4
C	$(2, 6)$	-15	0
D	$(8, 1)$	88	3
E	$(6, 1)$	68	3
F	$(0, 5)$	6	1
G	$(1, 0)$	-10	0
H	$(5, 5)$	61	1
I	$(3, 0)$	34	4
J	$(3, 4)$	38	3
K	$(1, 5)$	-5	0

(b) Fügen Sie die ersten neun Schlüssel der Tabelle aus (a) der Reihe nach (von oben nach unten) in die Hashtabelle ein. Tragen Sie dazu die Namen der Schlüssel (A bis I) in die Tabelle ein. Verwenden Sie Kollisionsauflösung durch Verkettung (seperate chaining). Die Listen verlaufen dabei horizontal von links nach rechts.

0	A	C	G	K		
1	F	H				
2						
3	D	E	J			
4	B	I				

(c) Was ist ein wichtiges Kriterium für eine gute Hashfunktion in Bezug auf die Laufzeit für das Suchen und Einfügen von Elementen in Hashtabellen mit Kollisionsauflösung durch Verkettung?

Die Wahrscheinlichkeit, dass bei m Elementen, die auf n Slots abgebildet werden, deutlich mehr als m/n Elemente auf den gleichen Slot abgebildet werden (also den gleichen Hashwert haben) soll verschwindend gering sein.

Vorausgesetzt die Hashfunktion erfüllt das Kriterium (und Hashcodes sind sinnvoll definiert): Geben Sie die ungefähre Anzahl der Schlüsselvergleiche für Suchen und Einfügen bei einer Hashtabelle mit M verketteten Listen und N eingefügten Elementen an.

Um ein Element zu finden muss in dem durch die Hashfunktion festgelegten Slot das Element linear gesucht werden, was im Durchschnitt m/n Vergleiche benötigt.

Aufgabe 4: Laufzeit ($2 + 8 = 10$ Punkte)

Gegeben sei ein gewichteter Graph $G(V, E)$, dessen Kantengewichte alle entweder 1 oder 2 sind. Die Aufgabe ist es, kürzeste Wege von einem Startknoten s zu allen anderen Knoten zu finden.

(a) Welche Laufzeit hätte die Verwendung von Dijkstra in diesem Fall (in der Version, die in der Vorlesung als Pseudocode besprochen wurde)? Geben Sie auch die Datenstruktur an, mit der diese Laufzeit erreicht wird.

O(|E| log |V|) mit einer indizierten Prioritätswarteschlange

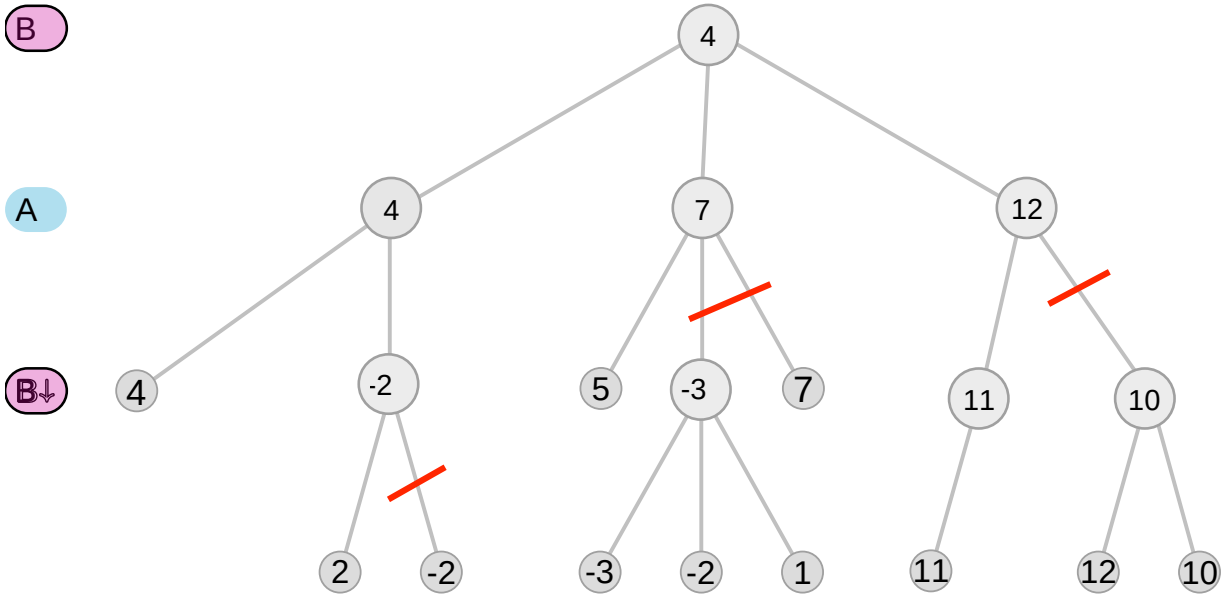
(b) Beschreiben Sie eine Methode, um eine Laufzeit in $O(\max(V, E))$ zu erreichen, und begründen Sie die Laufzeit.

Man kann die Kanten mit Gewicht 2 in 2 Kanten mit Gewicht 1 dekomponieren, die durch einen neuen Knoten verbunden sind.
Dieser Vorgang benötigt O(|E|) Zeit.
Daraufhin kann man im resultierenden ungewichteten Graph G' eine BFS in O(|V'|+|E'|) durchführen, um die kürzesten Wege zu allen Knoten in G zu finden.

O(|V'|+|E'|) = O(2·|V|+2·|E|) = O(2·(|V|+|E|)) = O(|V|+|E|)
Fall 1 (|V|>|E|):
O(|V|+|E|) O(2·|V|) = O(|V|) = O(max(|V|, |E|))
Fall 2: (|E| |V|):
O(|V|+|E|) O(2·|E|) = O(|E|) = O(max(|V|, |E|))

Somit hat der gezeigte Algorithmus eine Laufzeit in O(max(|V|, |E|))

Aufgabe 5: Minimax- und Alpha-Beta-Algorithmus (4 + 4 = 8 Punkte)

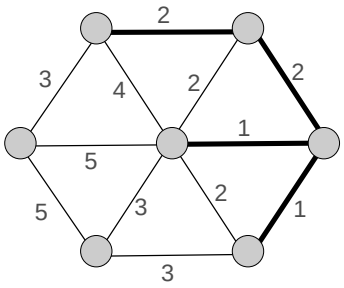


(a) Vervollständigen Sie den obigen Minimax Suchbaum.

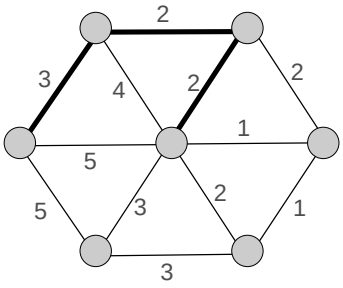
(b) Nehmen Sie an, Sie würden auf dem obigen Suchbaum eine Alpha-Beta-Suche ausführen, die von links nach rechts läuft. Welche Zweige würden nicht besucht? Tragen Sie α- und β- Cutoffs in den Baum ein. Kennzeichnen Sie, welcher Cut ein α oder ein β-Cutoff ist.

Aufgabe 6: Minimum Spanning Tree ($6 \times 2 = 12$ Punkte)

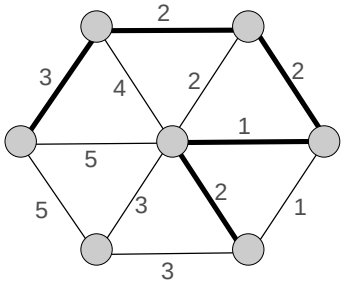
In der folgenden Abbildung sehen Sie sechs mal den gleichen Graphen. In diesen Graphen sind Kanten markiert, die durch den Prim (mit beliebigem Startknoten) oder den Kruskal Algorithmus oder durch keinen von beiden ausgewählt wurden. Dabei muss der jeweilige Algorithmus nicht bis zum Ende durchgelaufen sein, es können also auch partielle Lösungen markiert sein. Geben Sie unter jedem Graphen an, welcher der beiden Algorithmen die Kanten markiert hat, dazu schreiben Sie entweder **Kruskal** oder **Prim** unter den jeweiligen Graph in das graue Kästchen. Wenn es beide hätten sein können, schreiben Sie **beide**, wenn es keiner von beiden hätte sein können, schreiben sie **keiner** darunter.



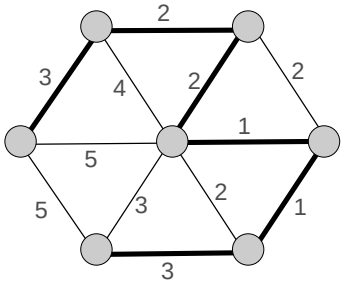
beide



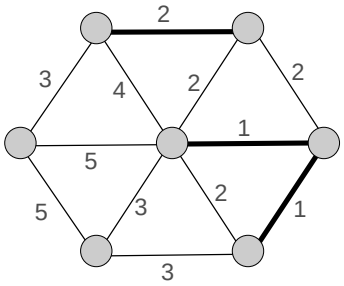
Prim



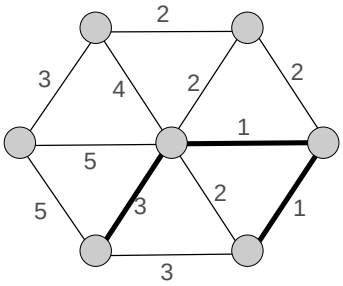
keiner



beide



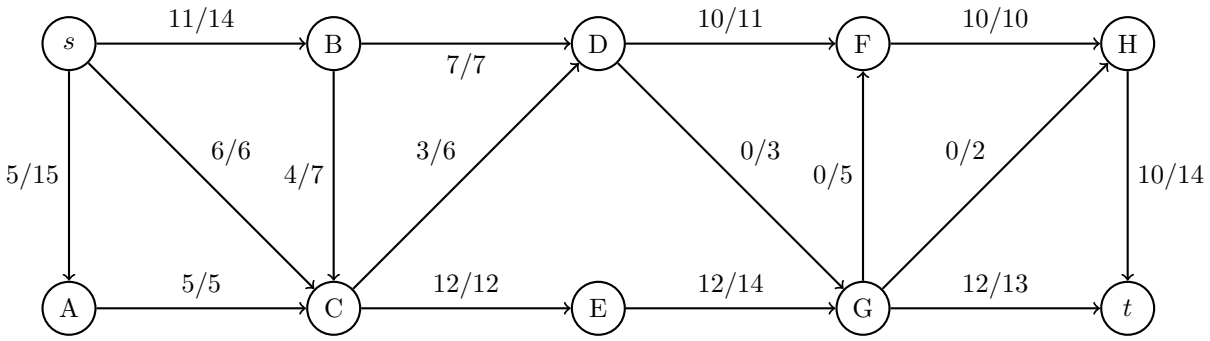
Kruskal



Prim

Aufgabe 7: Edmonds-Karp Algorithmus ($1+6+2+1=10$ Punkte)

Betrachten Sie den folgenden Flussgraphen mit Quelle s und Senke t und dem eingetragenen Fluss:



(a) Notieren Sie den Wert des Flusses im obigen Netzwerk.

f=22

(b) Führen Sie eine Iteration des Edmonds-Karp Algorithmus aus und notieren Sie die Knoten des vergrößernden Pfades beginnend in s und endend in t.

p={(s,B), (B,C), (C,D), (D,G), (G,t)}
cv=1

(c) Notieren Sie den Wert des maximalen Flusses im obigen Netzwerk.

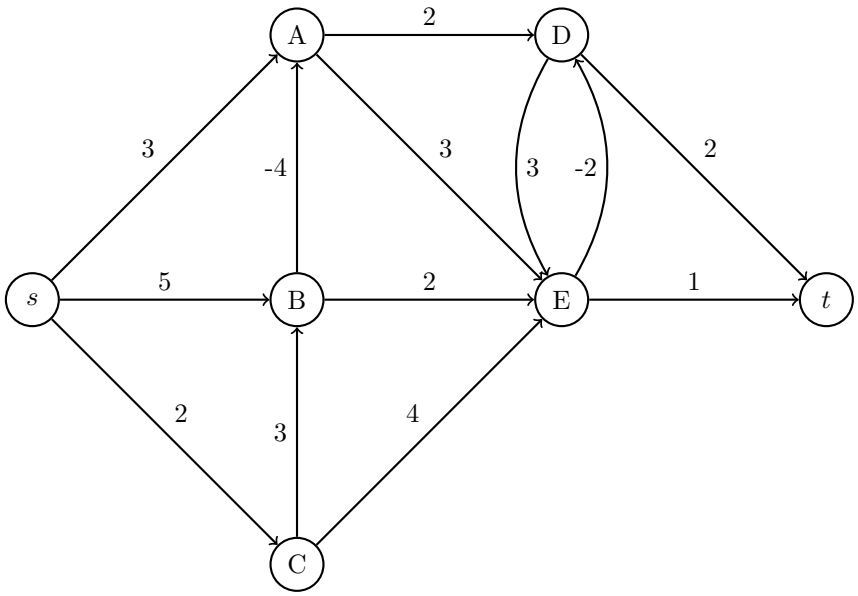
f'=25

(d) Notieren Sie die Kapazität des minimalen Schnittes im obigen Netzwerk.

Nach dem Max-Flow-Min-Cut-Theorem ist die Kapazität des maximalen Flusses gleich den Kantengewichten des minimalen Schnittes und somit ebenfalls 25.

Aufgabe 8: Kürzeste Wege ($2 + 9 + 2 = 13$ Punkte)

In dieser Aufgabe betrachten wir das Problem der kürzesten Pfade auf dem folgenden Graphen:



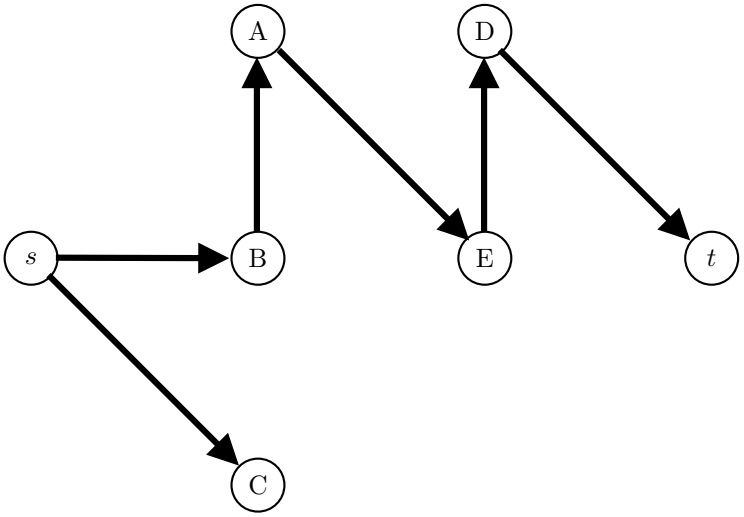
(a) Welcher Algorithmus sollte gemäß der VL zur Lösung des SSSP (Single-Source Shortest Paths) Problems für den gegebenen Graphen verwendet werden?

Da es Kanten mit negativen Gewichten gibt, muss der Bellman-Ford-Algorithmus verwendet werden.

(b) Führen Sie den Algorithmus auf dem obigen Graphen aus. Vervollständigen Sie dabei die gegebene Tabelle so, dass Sie für jeden Knoten, dessen ausgehende Kanten relaxiert werden, eine neue Zeile nutzen und die Distanz in der Tabelle aktualisieren. Tragen Sie dabei die Knoten, von denen die Relaxierung ausgeht, in die 2. Spalte ein.
Hinweis: Es müssen nicht alle Zeilen der Tabelle genutzt werden, aber achten Sie darauf, dass Sie die Tabelle konsistent ausfüllen.

Relaxierungsschritt	ausgehender Knoten der Relaxierung	Distanz zu s						
		s	A	B	C	D	E	t
Start	-	0						
1. Schritt	s	0	3	5	2			
2. Schritt	A	0	3	5	2	5	6	
3. Schritt	B	0	1	5	2	5	6	
4. Schritt	D	0	1	5	2	5	6	7
5. Schritt	E	0	1	5	2	4	6	7
6. Schritt	A	0	1	5	2	3	4	7
7. Schritt	D	0	1	5	2	3	4	5
8. Schritt	E	0	1	5	2	2	4	5
9. Schritt	D	0	1	5	2	2	4	4
10. Schritt								

(c) Zeichnen Sie den Baum aller kürzesten Pfade zu dem gegebenen Graphen.



Aufgabe 9: Dynamisches Programmieren ($2 + 7 + 3 = 12$ Punkte)

Es ist ein Array von ganzen Zahlen $a[]$ der Länge N gegeben und es werden die zusammenhängenden Teilfolgen (Subarrays) $a[i], \ldots, a[j]$ für alle $0 \quad i \quad j < N$ betrachtet. Mittels Dynamischer Programmierung soll die maximale Summe $a[i] + \cdots + a[j]$ aller Subarrays von a bestimmt werden. Für

$$a[] = 2, 2, \ 5, 4, 0, \ 2, 4, \ 7, 3$$

ist diese maximale Summe 6, und sie wird von der Teilfolge $4, 0, \ 2, 4$ erzielt.

(a) Ein brute-force Ansatz wird durch den folgenden Pseudocode beschrieben:

```
1 // gegeben numerisches Array a[] der Länge N
2 max =  inf
3 for all subarrays t of a
4   sum(t)  sum of subarray t
5   if sum(t) > max
6     max  sum(t)
7   end
8 end
9 return max   // maximale Summe aller Teilfogen von a[]
```

Welche Laufzeit hat dieser Algorithmus (minimale Größenordnung mit kurzer Begründung)?

Die Laufzeit ist in O(n^3)
Es gibt je n-k+1 Subarrays der Länge k gibt und die Summe benötigt k Operationen

(b) Geben Sie eine rekursive Definition der Funktion Opt(i) an, die die gr ößte Summe aller zusammenhängenden Teilfolgen von a[] bestimmt, die mit dem Element a[i] enden. Diese Funktion kann als Grundlage für dynamisches Programmieren verwendet werden.

Tipp: Entweder wird eine vorherige Teilfolge durch $a[i]$ verl ängert, oder $a[i]$ beginnt eine neue Teilfolge.

OPT(0) := a[0]
OPT(i) := max(OPT(i-1) + a[i], a[i])

(c) Nehmen Sie an, dass alle Werte $OPT(i)$ für $i = 0, \ldots, N \quad 1$ per bottom-up Verfahren durch dynamische Programmierung bestimmt wurden. Beschreiben Sie, wie daraus die maximale Summe aller zusammenhängenden Teilfolgen von a[] bestimmt werden kann.

Geben Sie die Laufzeit des gesamten Verfahrens an.

Bemerkung: Sie können diese Teilaufgabe unabhängig von (b) lösen. Es reicht die in (b) gegebene Beschreibung.

Die maximale Summe aller zusammenhängenden Teilfolgen ist durch max({OPT(i)| i in {0,…,N-1}}) gegeben.

Die Laufzeit des Gesamtverfahrens ist O(N), da sowohl die Ermittlung der OPT-Werte als auch die Maximumbestimmung in O(N) geschieht.