Fachhochschule Aachen Fachbereich Elektrotechnik und Informationstechnik	**Klausur** Algorithmen und Datenstrukturen Prof. Ingrid Scholl	SS2010 16.07.2010 Seite 1 / 24

Name (Blockschrift) Matrikelnummer

Hinweise:
Bearbeitungszeit: 180 Minuten
Hilfsmittel: Ein nichtprogrammierbarer Taschenrechner. Weitere Hilfsmittel sind nicht zugelassen!

- Bearbeiten Sie die Aufgaben direkt und nur auf den Aufgabenblättern oder Leerbögen
- Lösungen ohne erkennbaren und leserlichen Lösungsweg sind wertlos.
- Klausur zusammenlassen!!!
- Deckblatt mit Namen und Matrikelnummer beschriften und unterschreiben!
- Die letzte Seite der Klausur beinhaltet eine STL-Hilfe.

Hiermit bestätige ich, dass ich die Hinweise zur Kenntnis genommen habe:

(Unterschrift)

Aufgabe	Max. Pkt.	Punkte	
1	**16**		
2	**8**		
3	**21**		
4	**13**		
5	**8**		
6	**16**		
7	**10**		
8	**17**		
Gesamt	**109**		**Note:**

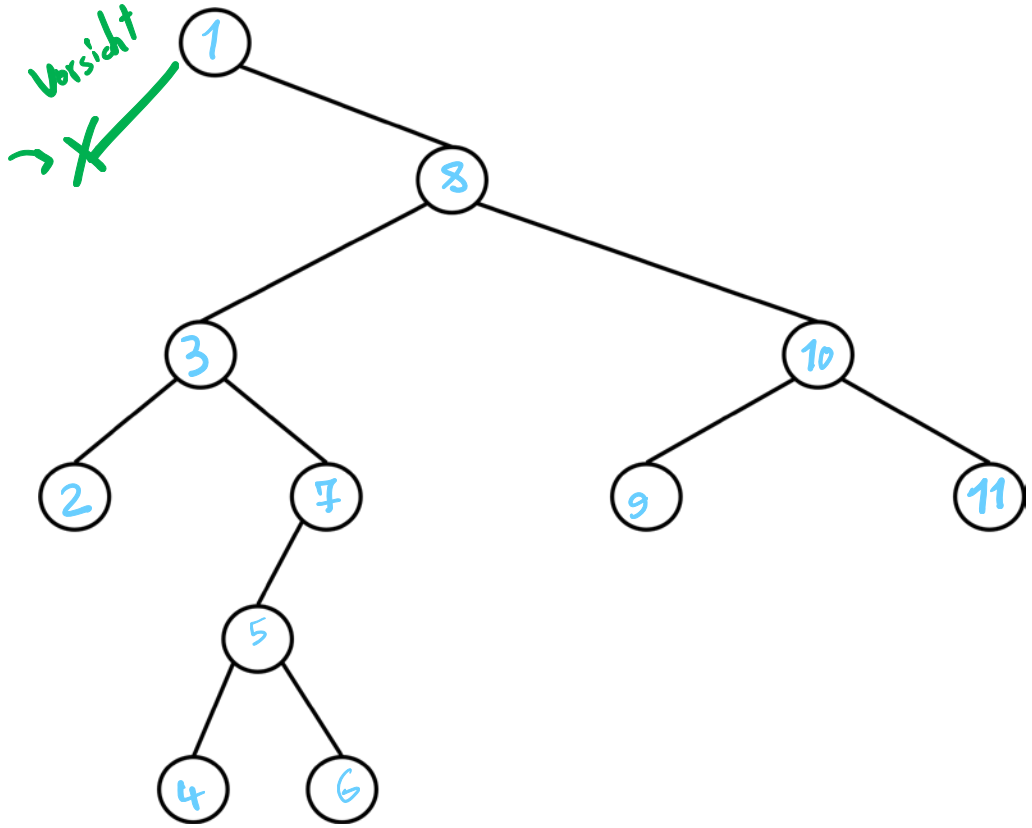
Aufgabe 1 (16 Punkte)

(a) Gegeben sei ein Binärer Suchbaum. Begründen Sie den möglichen Aufwand bzw. die möglichen Aufwände, ein Element im binären Suchbaum zu finden.

$O(\log n) <= O(g_n) <= O(n)$

Best Case — Worst Case

(b) Gegeben sei die folgende Struktur eines binären Suchbaums: Fügen Sie die Zahlen 1 bis 11 ein.



Wie sieht die die Traversierungsreihenfolge der Elemente aus, wenn der Baum mit den folgenden Traversierungsalgorithmen durchlaufen wird?

- Preorder: 1 8 3 2 7 5 4 6 10 9 11
- Inorder: 1 2 3 4 . — . — 11
- Postorder: 2 4 65 7 3 9 11 10 8 1
- Levelorder 1 8 3 10 2 7 9 11 5 4 6

(c) Mit welchem Aufwand kann in einer einfach verketteten Liste mit n Elementen ein Element eingefügt werden? Die Liste verfügt nur über einen Kopfzeiger auf das erste Element. Kreuzen Sie die richtigen Aussagen an. Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- ☒ O(1) wenn es am Kopf eingefügt wird.
- ☒ O(n) beim Einfügen am Ende.
- ☐ O(log n) beim Einfügen in der Mitte.
- ☐ O(n) beim Einfügen am Anfang.
- ☐ O(1) beim Einfügen am Ende.

(d) Gegeben sei eine verkettete Liste in Arraydarstellung. X
Das Array hat die Elemente: 0 1 2 3 4 5 6 7
Die Freispeicherliste liefert folgende verkettete Liste: 5 2 1 3
Wie ist der entsprechende Inhalt der Freispeicherliste?

(e) Gegeben ist der folgende Source Code eines Sortieralgorithmus:

```
for (int i=2; i <= N; i++)
{
  int tmp = a[i];

  int j = i;
  while (j > 0 && a[j-1] > tmp)
  {
     a[j] = a[j-1];
     j--;
  }

  a[j] = tmp;
}
```

Um welchen Sortieralgorithmus handelt es sich? Insertion Sort

(f) Eine dynamisch verkettete Liste hat folgende Eigenschaften – kreuzen Sie die wahren Aussagen an. Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- [x] Wenn ein Knoten der verketteten Liste als Nachfolger den Nullpointer hat, ist dieser Knoten der letzte Knoten in der verketteten Liste.
- [x] Die Liste besteht aus miteinander verbundenen Knoten.
- [ ] Eine verkettete Liste wird gelöscht, indem der Zeiger auf den ersten Knoten der verketteten Liste gelöscht wird.
- [x] Eine verkettete Liste ist ein gerichteter Graph.
- [ ] Wenn ein Knoten einen Vorgänger und einen Nachfolger hat, gehört dieser zu einer einfach verketteten Liste.
- [x] Die Knoten der Liste sind eigene Datenstrukturen.

(g) Welche Sortierverfahren werden **nicht** als elementare **(mit $O(n^2)$)** Sortierverfahren bezeichnet? Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- [x] HeapSort
- [ ] InsertionSort
- [x] ShellSort → weil $\lim_{n \to \infty} \frac{n^{i<2}}{n^2} = 0$
- [x] QuickSort
- [ ] BubbleSort
- [x] MergeSort
- [ ] SelectionSort

(h) Zu welchem Sortieralgorithmus gehört folgende Aussage:
Nehme das erste unsortierte Element und sortiere es in der Menge der sortierten Elemente ein. Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- [ ] Selection Sort
- [x] Insertion Sort
- [ ] Bubble Sort

(i) Ist der für AVL-Baum auch ein binärer Suchbaum? Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- (●) Ja
- ( ) Nein

(j) Wie lautet die Min.Heap-Eigenschaft? Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- ( ) Schlüssel jeder Knoten >= Schlüssel seiner Kinder
- (●) Schlüssel jeder Knoten <= Schlüssel seiner Kinder

(k) Bitte markieren Sie die richtigen Aussagen. Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- [x] MergeSort teilt die Folge solange in 2 gleich große Teilmengen auf, bis die Größe der Teilmengen nur noch aus 1 Element besteht.
- [x] QuickSort benötigt zum Sortieren keine zusätzliche Datenstruktur.
- [x] MergeSort mischt in der gleichen Reihenfolge die Teilmengen zusammen wie diese auch aufgeteilt wurden.
- [x] Wenn das Pivotelement immer das mittlere Element in der Folge ist, liefert QuickSort den gleichen Aufwand wie MergeSort.
- [x] MergeSort ist ein Divide & Conquer Algorithmus.
- [ ] HeapSort benötigt doppelten Speicherplatz beim Sortieren.
- [ ] HeapSort verwendet als Datenstruktur einen Binärbaum.
- [ ] Der Aufwand beim QuickSort ist unabhängig von der Wahl des Pivotelementes.
- [x] MergeSort benötigt zum Sortieren den doppelten Speicherplatz.

X (l) Was ist der Unterschied zwischen einem digitalen Trie und einem digitalen Suchbaum? Kreuzen Sie die richtigen Aussagen an. Achtung: Bei falschen Antworten gibt es Punktabzug, minimale Punktzahl sind 0 Punkte.

- [ ] Der digitale Suchbaum ist von der Eingabereihenfolge der Schlüssel unabhängig.
- [x] Die internen Knoten des digitalen Tries bestehen aus Teilen des Schlüssels.
- [x] Der digitale Trie ist von der Eingabereihenfolge der Schlüssel unabhängig.

Aufgabe 2 (8 Punkte)
Löschen eines Elementes in einer doppelt verketteten Liste (ohne STL)

Gegeben ist eine doppelt verkettete Liste, dynamisch programmiert, deren Elemente **aufsteigend sortiert** sind. Ein Knoten hat die folgende Schnittstelle:

```
class node {
public:
  int item;
  node* next;
  node* prev;
  node() { item = 0; node = 0; next = 0; }
};
```

Implementieren Sie eine Funktion, die in einer doppelt verketteten Liste einen Knoten mit dem Element x löscht. Der Kopfzeiger head referenziert das 1.te Listenelement, der Zeiger tail referenziert das letzte Listenelement. Beachten Sie, dass Sie durch das Löschen ggfls. den head- oder tail-Zeiger verändern müssen. Liefern Sie als Rückgabewert, ob das Löschen erfolgreich oder erfolglos war. Das Löschen kann erfolglos sein, wenn das zu löschende Element nicht in der Liste vorhanden ist. Head und tail seien private Variablen der Klasse zur doppelt verketteten Liste.

```
bool deleteItem(int x)
{
// Rückgabewert: erfolgreiches oder erfolgloses Löschen
// x: der Wert des zu löschenden Knotens
node * ptr = head;
while ( ptr != null && ptr->item != x)
    ptr = ptr->next;

if(!ptr)
    return false;
```

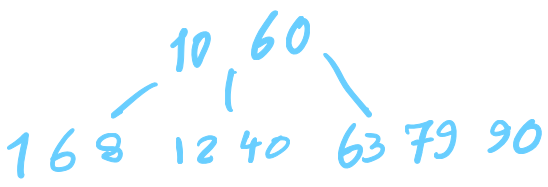
```
if (ptr->next)
    ptr->next->prev = ptr->prev;
else
    tail = ptr->prev;
if (ptr->prev)
    ptr->prev->next = ptr->next;
else
    head = ptr->next;
delete ptr;
return true;
```

Aufgabe 3 (6 + 15 = 21 Punkte)
234-Baum

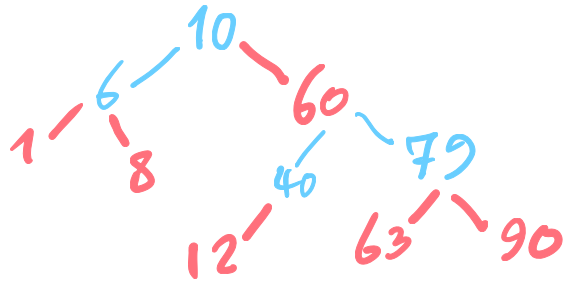
(a) Erzeugen Sie im TopDown Verfahren einen 234-Baum durch die Eingabe der Folge: 40, 6, 10, 1, 79, 60, 63, 8, 12, 90.
(b) Wie sieht der zugehörige Baum als Rot-Schwarz-Baum aus?
(c) Fügen Sie nun im Rot-Schwarz-Baum nacheinander die Schlüssel 5 und 80 ein und skizzieren und beschreiben Sie die notwendigen Schritte.
(d) Wie sieht das Ergebnis des Rot-Schwarz-Baumes als 234-Baum aus?
(e) Implementieren Sie eine Methode, die die Knoten des 234-Baumes in Levelorder niveauweise ausgibt. Das Niveau selbst soll immer dann ausgegeben werden, wenn ein Niveauwechsel stattfindet. Verwenden Sie dazu die Implementierung des Rot-Schwarz-Baumes. (15 Punkte)

```
// Klasse für einen Knoten
// im Rot-Schwarz-Baum
class node {
public:
   int item;
   node *left;
   node *right;
   int color; // 0 = schwarz,
              // 1 = rot
   node(){  left  = 0;
            right = 0;
            item  = 0;
            color = 1;
   }
};
```

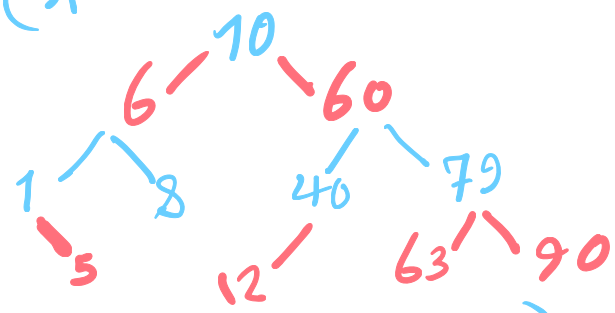
a)



b)

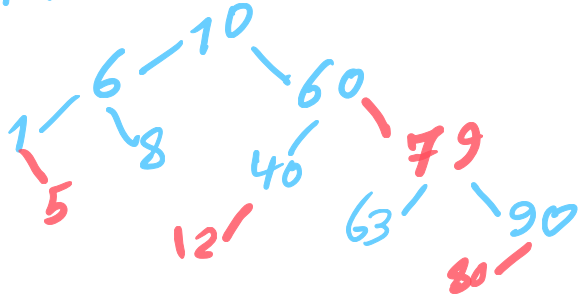


c) 5 → (split 6)

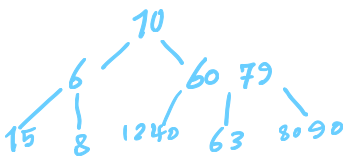


(split 10) && (split 79)

80 →



d)




```
void levelorder()
{  // sei anker der wurzel
   queue<node*> q;
   queue<int> levels;

   q.push(anker);
   levels.push(0);
   int maxLevel = -1;
   node* Knoten;
   int level;
   while(!q.empty())
   {  Knoten = q.front(); q.pop();
      level = levels.front(); levels.pop();
      if(!Knoten)
         continue;

      if(maxLevel < level){
         cout << endl;
         maxLevel = level;
         cout << "Niveau " << level << ":";
      }
      cout << "(";
      if(Knoten->left && Knoten->left->color){
         cout << Knoten->left->item << ", ";
         q.push(Knoten->left->left);
         levels.push(level+1);
         q.push(Knoten->left->right);
         levels.push(level+1);
      } else { q.push(Knoten->left); levels.push(level+1); }

      cout << Knoten->item << ", ";
      if(Knoten->right && Knoten->right->color){
         cout << Knoten->right->item;
         q.push(Knoten->right->left);
         levels.push(level+1);
         q.push(Knoten->right->right);
         levels.push(level+1);
      } else { q.push(Knoten->right); levels.push(level+1); }
      cout << ")";
   }
}
```

Aufgabe 4 (13 Punkte)
Programmieren des Shellsort-Algorithmus

Schreiben Sie eine C++-Methode, die **N** Zahlen in einen STL-Integer-Vektor **Vekt** einliest, die Zahlen nach dem **Shellsort-Algorithmus** sortiert, auf der Standardausgabe ausgibt und dem rufenden Programm zurückgibt. Zur Bestimmung der Abstandsfolge verwende man die rekursive Beziehung H := 2H + 1.

```
void shellSort(vector<int> Vekt, int N)
{
    Vekt.resize(N);
    for(int i=0; i<N; i++)
        cin >> Vekt[i];

    int H = 1;
    while( H <N)
        H = 2*H +1;
    for(int gap = (H-1)/2 ; gap>=1 ; gap = (gap-1)/2 )
    {   for(int i=gap; i<N; i++)
        {  int tmp = Vekt[i];
           int j = i;
           while(j-gap >=0 && Vekt[j-gap] > tmp)
           {  Vekt[j] = Vekt[j-gap];
              j-=gap;
           }
           Vekt[j] = tmp;
        }
    }

    for(int a : Vekt)
        cout << a << " ";

    cout << endl;

}
```

Aufgabe 5 (8 Punkte)
Templatefunktion

Gegeben ist der folgende Source-Code einer einfach verketteten Liste. Welche Änderungen müssen Sie vornehmen, um Elemente beliebigen Typs mit dieser Datenstruktur verarbeiten zu können. Tragen Sie die Änderungen direkt im Source-Code ein.

```
template <classname T>
class Node {
public:
T  int data;
  Node *next;
};   <T>

template <classname T>
class List {
private:   <T>
  Node *head; // Kopfzeiger der verketteten Liste
public:
  List();
  List(int item);   T
  bool insertItem(int item);
  void printContent();
};

template <classname T>
List::List()
{                  <T>
  head = new Node;  // Kopf erzeugt
  // Kopf initialisieren
  head->data = 0;
  head->next = 0; //head->next = 0;
}

template <classname T>
bool List::insertItem(int item)
{   <T>                 <T>
  Node *pnew = new Node();  // neuen Knoten erzeugen
  pnew->data = item;
  pnew->next = head->next;
  head->next = pnew;
  return true;
}
```

template<Classname T>

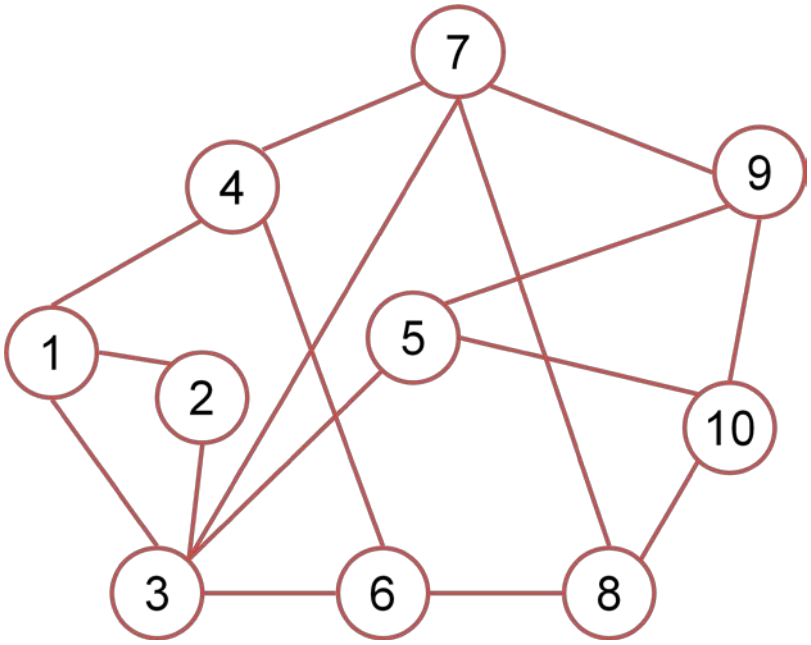
```
void List::printContent()
{
  // Inhalt der verketteten Liste ausgeben
  Node *temp = head;
  cout << "Inhalt der Liste: ";
  while(temp->next != 0)
  {
    cout << temp->next->data << " ";
    temp = temp->next;
  }
  cout << endl << endl;
}


int main(void)
{
  // Datenstruktur List instanzieren
  List myList;
  bool ok = myList.insertItem(5);
  myList.printContent();
}
```

<T>

<int>

Aufgabe 6 (10+6 = 16 Punkte)
Graphen

Gegeben ist der folgende Graph G mit V={1,2,..,10} Knoten:



(a) Um welche Art eines Graphen handelt es sich hier?
Ungerichtet ss ungewichtet

(b) Stellen Sie den Graphen in einer Knotenliste dar (bitte numerisch sortiert).
[10, 16, 3, 2, 3, 4, 2, 1, 3, 5, 1, 2, 5, 6, 7, 3, 1, 6, 7, 3, 3, 9, 10,
3, 3, 4, 8, 4, 3, 4, 8, 9, 3, 6, 7, 10, 3, 5, 7, 10, 3, 5, 8, 9]

(c) Geben Sie die Adjazenzliste zu dem Graphen an (bitte numerisch sortiert angeben).

1 → 2 3 4
2 → 1 3
3 → 1 2 5 6 7
4 → 1 6 7
5 → 3 9 10
6 → 3 4 8
7 → 3 4 8 9
8 → 6 7 10
9 → 5 7 10
10 → 5 8 9

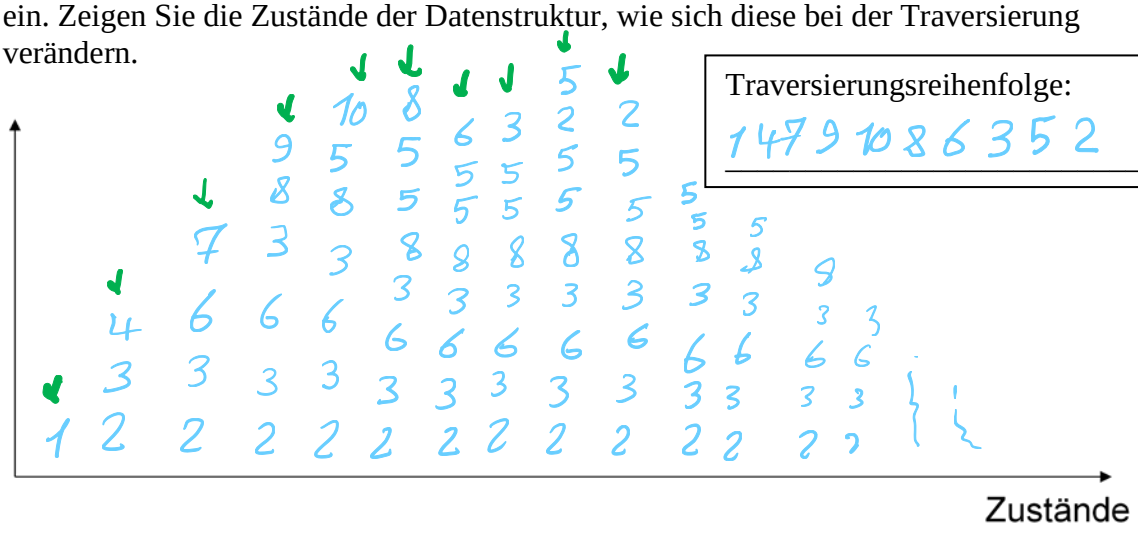
(d) Implementieren Sie unter Verwendung der STL eine Funktion **getAdjazenzList**, die aus einer Knotenliste eine Adjazenzliste berechnet und diese an das aufrufende Programm zurückgibt. (6 Punkte)

map < int, vector<int> > getAdjazenzList(vector<int> nodelist)

```
std::map<int, std::vector<int>> result;
int resultIndex = 1;

for (int i = 2; i < nodelist.size() - 1; i += nodelist[i] + 1)
{
    for (int k = i + 1; k <= i + nodelist[i]; k++)
    {
        result[resultIndex].push_back(nodelist[k]);
    }
    resultIndex++;
}

return result;
```

stack

(e) Führen Sie für den Graphen G eine **iterative Tiefensuche** beginnend vom Knoten 1 aus durch (Bearbeiten Sie die Nachfolgerknoten in der Reihenfolge der Adjazenzliste). Welche Datenstruktur verwenden Sie dazu? Tragen Sie diese als y-Achsenbeschriftung ein. Zeigen Sie die Zustände der Datenstruktur, wie sich diese bei der Traversierung verändern.



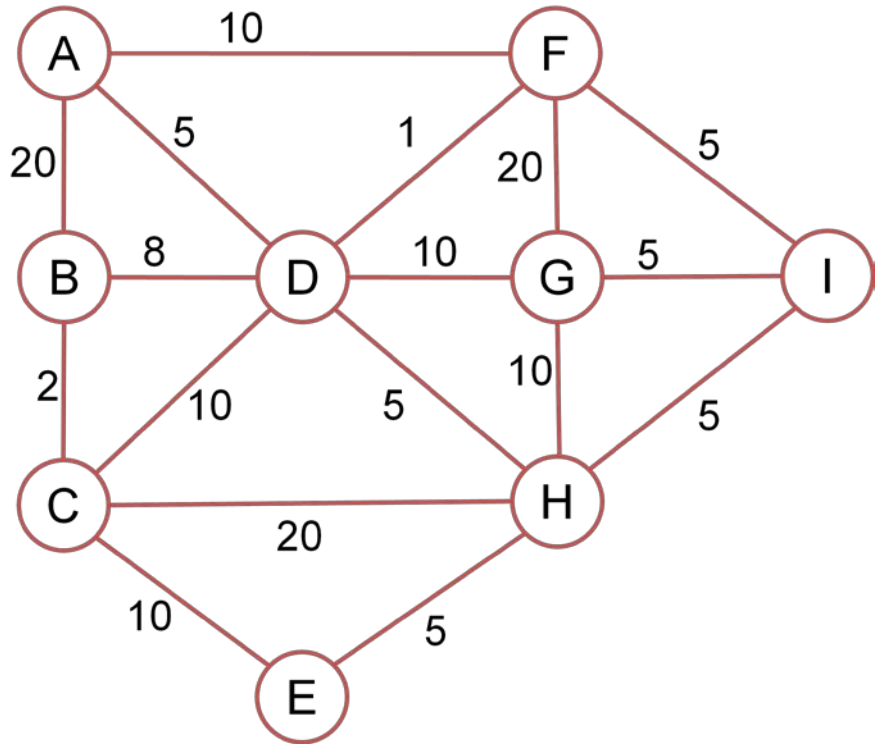
Traversierungsreihenfolge:
1 4 7 9 10 8 6 3 5 2

(f) Wie kann man feststellen, ob ein Knoten nicht traversiert wird?

mit vector<bool> marked;
man setzt marked[v] true wenn
stack.front() == v
v ist n.t. => marked[v] = false

Aufgabe 7 (10 Punkte)

Gegeben ist der folgende gewichtete Graph G = (V,E)



(a) Geben Sie die Adjazenzmatrix zu G an:

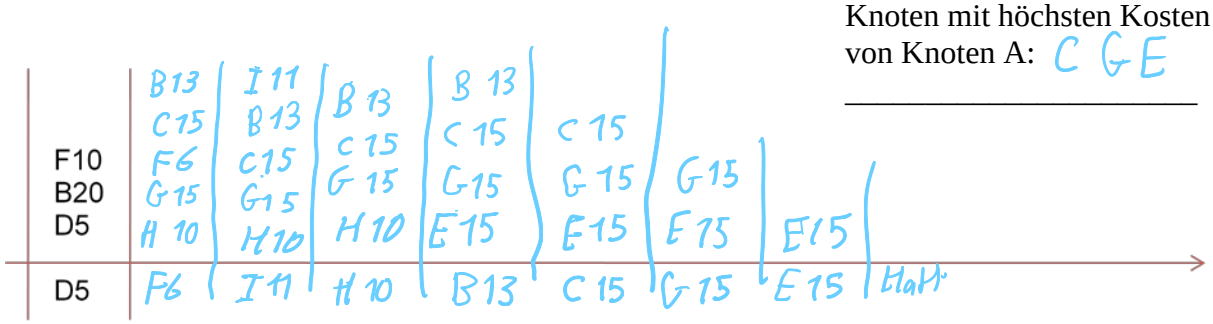
	A	B	C	D	E	F	G	H	I
	0	1	2	3	4	5	6	7	8
0	0	20	6	5	0	10	0	0	0
1	20	0	2	8	0	0	0	0	0
2	0	2	0	10	10	0	0	20	0
3	5	8	10	0	0	1	10	5	0
4	0	0	10	0	0	0	0	5	0
5	10	0	0	1	0	0	20	0	5
6	0	0	0	10	0	20	0	10	5
7	0	0	20	5	5	0	10	0	5
8	0	0	0	0	0	5	5	5	0

(b) Berechnen Sie mit dem **Dijkstra Algorithmus** die minimalen Kosten vom **Knoten A** zu jedem anderen Knoten. Welcher Knoten ist vom Knoten A mit den höchsten Kosten zu erreichen? Zeigen Sie die einzelnen Zustände der Priority Queue und beachten Sie, wie oft ein Update in der Priority Queue durchgeführt werden musste. (5 Punkte)

Kante	-	AD	DF	FI	DH	DB	DC	DG	HE				
Kosten von A	0	5	6	11	10	13	15	15	15				
Knoten	A	D	F	I	H	B	C	G	E				

11

Zustände der Priority Queue:



Knoten mit höchsten Kosten von Knoten A: C G E

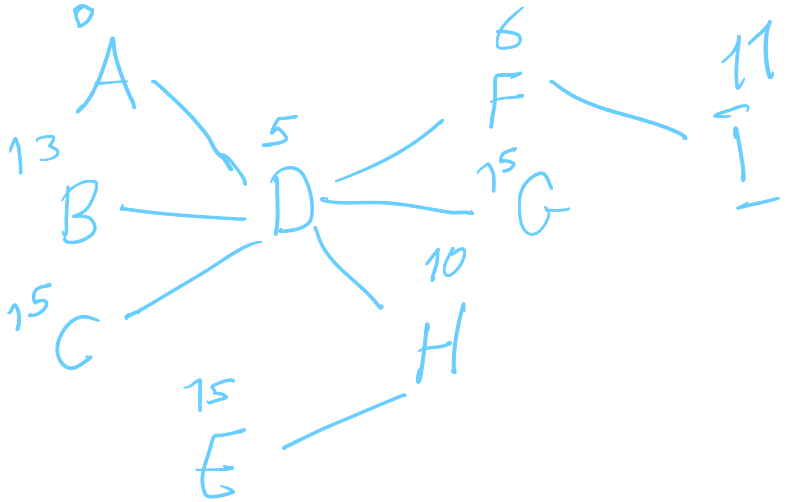
(c) Wie oft musste ein Update in der Priority Queue durchgeführt werden?

2

(d) Wann muss allgemein ein Update in der PQ durchgeführt werden?

Wenn ein Kürzer Weg zu einem Knoten gefunde wurde

(e) Zeichnen Sie den Graphen, der durch die Traversierung nach Dijkstra entsteht:



Aufgabe 8 (10 + 7 = 17 Punkte)
Hashing

(a) Die Zahlen **3, 18, 23, 21,14, 32** sollen in eine anfangs leere Hashtabelle der Größe M=9 nacheinander eingefügt werden. Zeigen Sie die Berechnungen der Kollisionsauflösung sowie das Ergebnis der Hashtabelle, wenn Sie die Kollisionen nach den folgenden Verfahren auflösen:

- Lineares Sondieren

i	0	1	2	3	4	5	6	7	8
h[i]	18	-1	-1	3	21	23	14	32	-1

- Quadratisches Sondieren (Tip: Durchhalten!)

i	0	1	2	3	4	5	6	7	8
h[i]	18			3	21	23	14		

- Doppeltes Hashing

i	0	1	2	3	4	5	6	7	8
h[i]	18	21	32	3	-1	23	-1	-1	14

- Welchen Wert hat der Belegungsfaktor nach dem Einfügen der Folge?

$\frac{6}{9} = \frac{2}{3}$

(b) Implementieren Sie eine Funktion, die ein Rehashing zu einer gegebenen Hashtabelle durchführt. Definieren Sie die Methode, sowie sämtliche Ein- und Ausgabeparameter. Eine Funktion getNextPrimeNumber kann genutzt werden. Sie liefert zu einer Zahl die nächst höhere Primzahl. Die Datenstruktur der Hashtabelle sei: `vector<int> a.` Lösen Sie mögliche Kollisionen mit dem linearen Sondierungsverfahren auf.
(7 Punkte)

int getNextPrimeNumber(int size)
{
// ist vorhanden und kann genutzt werden (bitte nicht implementieren)
// liefert zu einer Zahl size die nächst höhere Primzahl zurück
}

```
void Rehashing (vector<int> &a)
{   int newSize = getNextPrimeNumber (a.size()*2);
    vector<int> b = a;
    a.resize(newSize);
    for(int &i : a)
        i=-1;
    int hash, i;
    for(int x : b)
    {  i=0;
       while(true)
       {   hash = (x + i)%newSize;
           if( a[hash] == -1)
           {   a[hash]=x;
               break;
           } else {
              i++;
           }
       }
    }
}
```

STL-Hilfe:

Container, einige wichtige Funktionen und deren Verwendung:

vector<type> Feld

```
vector<int> numbers;
numbers.push_back(3);        // Einfügen am Ende des Feldes
numbers.pop_back();          // entfernt das letzte Element
int last = numbers.back();   // liefert das letzte Element
int first = numbers.front(); // liefert das erste Element
int item = numbers[i];       // liefert das i-te Element
int size = numbers.size();   // liefert die Größe des Feldes
bool empty = numbers.empty(); // true, wenn das Feld leer ist,
                              // andernfalls false
```

list<type> doppelt verkettete Liste (s.Funktionen wie bei vector)

```
list<int> numbers;
numbers.push_back(5); // fügt 5 ans Ende der Liste
numbers.remove(5);    // löscht das Element 5
```

queue<type> Queue

```
queue<int> q; // int – Warteschlange allokieren
q.push(5);    // Element 5 in die Warteschlange
bool leer = q.empty();   // Abfrage, ob Warteschlange leer ist
int element = q.front(); // liefert erstes Element
q.pop() ;     // löscht erstes Element
```

stack<type> Stapel

```
stack<int> nodes; // int - Stapel allokieren
bool ok = nodes.empty();   // überprüfe, ob Stapel leer ist
int element = nodes.top(); // hole Element vom Stapel
nodes.pop();   // entfernt Element vom Stapel
nodes.push(5); // 5 wird auf den Stapel gelegt
int i = (int) nodes.size(); // Anzahl Elemente im Stapel
```

map<keytype, valuetype> (Schlüssel, Referenz)-Container

```
map<int, vector<int> > adjlist;    // Adjazenzliste als Map
int size_adjlist = adjlist.size(); // Anzahl Knoten in der Adjazenzliste
vector<int> liste = adjlist[1]; // liefert den Vektor zum Schlüssel 1
adjlist[1].push_back(5); // Einfügen von Schlüssel 5 zur Liste von Schlüssel 1
```