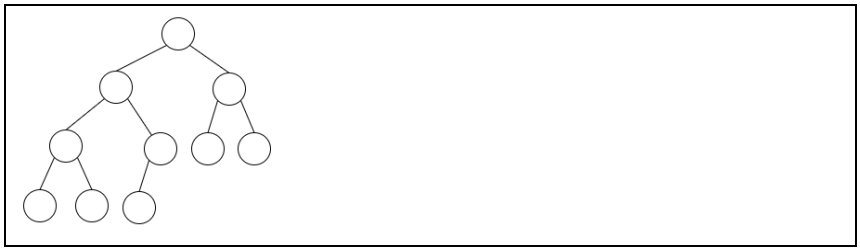
Bagian A

1. Manakah pernyataan berikut ini **YANG TEPAT** mengenai sembarang Binary Search Tree (BST) yang sudah berisi N buah node?
   a. **Operasi insert sebuah elemen berikutnya mungkin bersifat O (Log N)**
   b. Operasi insert sebuah elemen berikutnya tidak mungkin bersifat O (1)
   c. Operasi mencari elemen minimum pasti bersifat O (Log N)
   d. Operasi mencari elemen minimum pasti bersifat O (N)

2. Berapakah tinggi minimum dari sebuah Binary Search Tree (BST) dengan 10 buah node? Tulis dengan angka (misal: 100)
   Kunci Jawaban: 3

   Ilustrasi untuk kunci:
   Tinggi minimum tercapai ketika BST sebisa mungkin bersifat complete. Dengan 10 node:

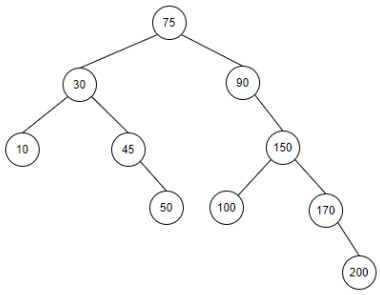


3. Sebuah BInary Search Tree (BST) dibentuk dengan urutan penyisipan node sebagai berikut:
   75, 30, 90, 10, 45, 150, 100, 170, 200, 50

   Lalu, node 75 dihapus dengan metode successor in-order. Node manakah yang menjadi root pada BST hasil akhirnya? Tulis dengan angka (misal: 110)

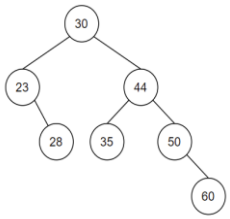
   Kunci Jawaban: 90

   Penjelasan:
   Berikut ini BST hasil insertion sesuai urutan di atas:

	Lalu 75 dihapus, maka 90 akan naik menjadi root.

19. [BST: IS CADANGAN] Manakah pernyataan **YANG TEPAT** mengenai sembarang Binary Search Tree (BST)?
    a. **Sebuah elemen baru yang di-insert ke dalam BST mungkin saja menjadi root.**
    b. Sebuah elemen baru yang di-insert ke dalam BST pasti menjadi leaf dan bukan root.
    c. Tidak mungkin ada node yang hanya memiliki 1 child.
    d. Tidak mungkin ada node yang tidak memiliki parent.

4. Dilakukan operasi delete 30 pada AVL Tree berikut ini.



Pilih **semua** pernyataan yang benar mengenai operasi deletion tersebut.
    a. **Jika kaidah predecessor in-order digunakan, node 30 digantikan oleh node 28**
    b. **Jika kaidah successor in-order digunakan, akan dilakukan single rotation pada node 44.**
    c. **Jika kaidah successor in-order digunakan, node 35 akan menggantikan 30**
    d. Jika kaidah predecessor in-order digunakan, node 30 digantikan oleh node 44
    e. Jika kaidah predecessor in-order digunakan, akan dilakukan single rotation pada node 23

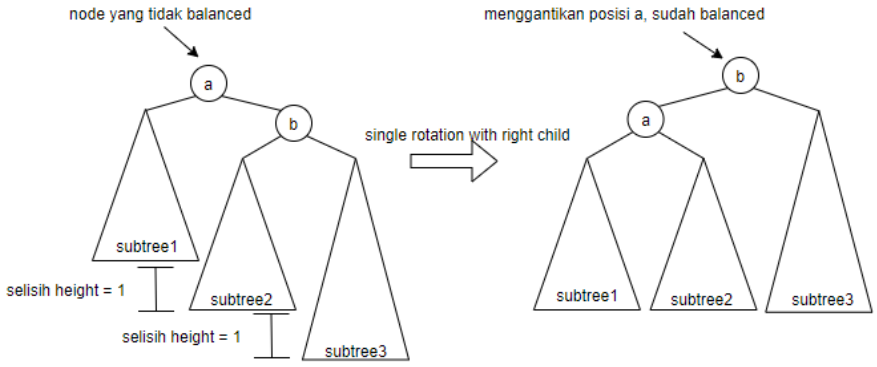
5. Berikut ini pernyataan yang benar mengenai AVL Tree dan BST
    a. **Operasi insertion dan deletion pada AVL Tree dengan N buah node tidak akan mencapai O(N) pada kasus terburuknya.**
    b. Operasi deletion pada BST dapat membuat tree tersebut tidak balanced, sehingga diperlukan single rotation maupun double rotation

c. Proses insertion pada BST dengan N buah node memiliki kompleksitas yang lebih baik (dalam big O) daripada insertion pada AVL Tree dengan jumlah node yang sama.
d. Setiap BST merupakan AVL Tree dan setiap AVL Tree juga merupakan sebuah BST

6. Pernyataan berikut yang benar mengenai AVL Tree adalah (N menunjukkan jumlah node pada AVL Tree):
   a. Kompleksitas single rotation pada AVL Tree tanpa memperhitungkan pencarian node yang perlu dirotasi adalah O(1)
   b. Kompleksitas method find elemen tidak jauh berbeda dengan method find pada BST
   c. Pencarian elemen membutuhkan waktu O (Log N) karena semua leaf terletak pada level yang sama
   d. Proses insertion bisa saja membutuhkan waktu O (N) jika elemen yang di-insert ditempatkan pada level terjauh dari root

7. Tinggi maksimum AVL Tree dengan 100 buah nodes adalah … (catatan: AVL Tree dengan 1 node memiliki ketinggian 0)
   Jawaban: 9
   Jumlah minimum node dengan ketinggian H
   1, 2, 4, 7, 12, 20, 33, 54, 88, 143
   Untuk node 100, maksimum tingginya di 8. Karena untuk tinggi 9, minimal butuh 143 nodes

8. Perhatikan ilustrasi berikut ini.



Catatan Gambar:
a dan b adalah nama reference ke MyAVLNode tersebut, bukan isi datanya
Pada gambar di atas, node a tidak balance, dan perlu dilakukan single rotation pada right child (anak kanan) dari a. Hasil setelah single rotation tersebut tampak di

sebelah kanan setelah menjalankan perintah singleRotateWithRightChild(a) yang mengembalikan root yang baru yaitu b.

Manakah code berikut ini yang tepat mengimplementasikan method singleRotateWithRightChild(MyAVLNode cr) sesuai dengan ilustrasi di atas?

a.	**MyAVLNode temp = cr.right;** **cr.right = temp.left;** **temp.left = cr;** **return temp;**

b.	MyAVLNode temp = cr.right;
	temp.left = cr;
	cr.right = temp;
	return temp;

c.	MyAVLNode temp = cr;
	cr.right = temp.left;
	temp.left = cr;
	return temp;

d.	MyAVLNode temp = cr.left;
	cr.right = temp.left;
	temp.left = cr;
	return temp;

9. MyAVLNode merepresentasikan sebuah node pada AVL Tree. MyAVLNode menyimpan data int, reference ke left dan right, juga atribut height. Height dari sebuah node yang tidak memiliki children adalah 0.
   MyAVLTree merepresentasikan sebuah AVL Tree, yang menggunakan MyAVLNode sebagai node-nya. Di dalam class MyAVLTree terdapat method insert(int x) dan method rekursif insert(int x, MyAVLNode cr) yang bertugas menyisipkan node berisi data x ke dalam subtree dengan root cr, dengan pseudo code sebagai berikut.
   (Catatan: asumsikan pengecekan tinggi subtree kiri dan kanan tidak mengakibatkan error. Misal: Jika cr.left adalah null, maka cr.left.height diset -1)

```
1. MyAVLNode insert(int x, MyAVLNode cr){
2.    if(cr == null )
3.          cr = new MyAVLNode(x, null, null);
4.    else if(x < cr.element)
5.    {
6.          cr.left = insert(x, cr.left );
7.          if(Math.abs(cr.left.height - cr.right.height == 2)
8.                if(x < cr.left.element)
9.                      cr = rotate1(cr);
10.               else
11.                     cr = rotate2(cr);
12.   }
13.   else if(x > cr.element)
14.   {
15.         cr.right = insert(x, cr.right);
16.         if(Math.abs(cr.left.height - cr.right.height == 2)
17.               if(x < cr.right.element)
18.                     cr = rotate3(cr);
19.               else
20.                     cr = rotate4(cr);
21.   }
22.   cr.height = Math.max(cr.left.height, cr.right.height) + 1;
23.   return cr;
24. }
25.
26. MyAVLNode insert(int x){
27.   return insert(x, root);
28. }

/* asumsikan masing-masing method rotate1, rotate2, rotate3, rotate4
sudah diimplementasikan */
```

   Diketahui myTree adalah sebuah objek bertipe MyAVLTree, dengan root yang masih null. Jika dilakukan operasi berikut ini:

```
int[] myArray = new int[]{30,20,10,15,12,40,50,60};
for (int i = 0; i < myArray.length; i++)
      myTree.insert(myArray[i]);
```

Tuliskan setiap data tersebut akan masuk case rotasi baris ke berapa. Contoh apabila insert 30 memerlukan rotasi1, pilih “9”. Jika tidak memerlukan rotasi, pilih “-”.

Jawab:

```
//rotateWithRightChild(cr); komen yang dikuning2in akan dihapus dari soal
```

(soal ini akan menggunakan drag n drop, mahasiswa tinggal pilih baris ke berapa)

30 : -

20 : -

10 : 9

15 : -

12 : 18

40 : -

50 : 20

60 : -

cat:

9. //rotateWithLeftChild(cr);

11. //doubleWithLeftChild(cr);

18. //doubleWithRightChild(cr);

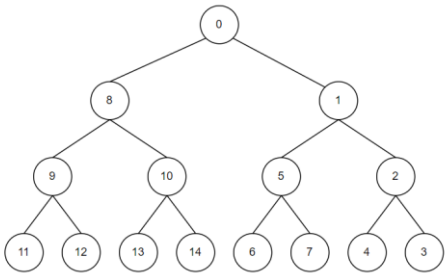
20. //rotateWithRightChild(cr);

10. Berikut adalah array yang berisi elemen-elemen sebuah complete binary tree yang ditulis secara level order. Di antara array berikut, manakah yang tidak mungkin merepresentasikan sebuah binary heap, baik min-heap maupun max-heap?
    a. **{ 29, 37, 60, 56, 45, 38, 49, 59 }**
    b. { 55, 19, 43, 9, 18, 40, 10, 6 }
    c. { 73, 60, 40, 54, 2, 7, 22, 15 }
    d. { 16, 43, 23, 45, 57, 80, 35, 50 }

11. Manakah pernyataan yang benar mengenai binary heap?
    a. Pada representasi array sebuah max-heap, nilai elemen di indeks 50 tidak lebih kecil dari nilai elemen di indeks 101.
    b. Binary heap merupakan struktur data yang direpresentasikan dalam sebuah binary search tree.
    c. Pada representasi array sebuah min-heap, elemen terkecil tersimpan di indeks 0 dan elemen terbesar tersimpan di indeks terakhir.
    d. Kompleksitas insert data adalah O(log N) sedangkan kompleksitas remove data adalah O(1), di mana N merepresentasikan jumlah data dalam binary heap.

12. Pada binary min-heap yang direpresentasikan dengan sebuah array, di mana setiap elemennya bernilai unik (tidak ada duplikasi), elemen terkecilnya ada di indeks 0 dan

elemen terkecil keduanya maksimal ada di indeks 2. Berapa indeks maksimal posisi dari elemen terkecil kelima? (Tulis angkanya saja)
Jawab: 30
Dalam min-heap, elemen terkecil ke-k paling jauh ada di level k-1.
Elemen terkecil ke-1 ada di level 0 (yg hanya ada 1 elemen)
Elemen terkecil ke-2 ada di level 1
Elemen terkecil ke-3 bisa ada di level 1 atau di level 2 (anak dari elemen terkecil ke-2 yg ada di level 1)
Binary tree dengan tinggi h paling banyak akan punya 2^(h+1) - 1 node, berarti kalau diasosiasikan dengan indeks array, akan ada array dengan indeks 0 sampai 2^(h+1)-2.
Berdasarkan soal, elemen terkecil kelima maksimal ada di level 4.
Binary tree dengan 4 level akan punya paling banyak 31 node.
Kalau diasosiasikan dengan array, berarti ada di indeks 0-30.
Ilustrasi:



Elemen terkecil keempat (elemen dengan nilai 3) ada di level 3.

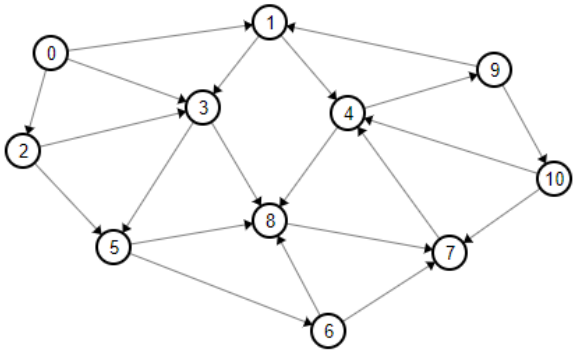
13. Diketahui sebuah array yang merepresentasikan sebuah min-heap sebagai berikut.
    18 - 40 - 20 - 51 - 43 - 30 - 22
    Jika dilakukan penambahan (insert) sebuah elemen yaitu 25, bagaimana urutan dari elemen pada array saat ini? (Urutan elemen array ditulis dari kiri ke kanan)
    Jawab: 18 - 25 - 20 - 40 - 43 - 30 - 22 - 51

14. Dalam algoritma fix heap (heapify) untuk membangun heap dari sebuah array tak urut, proses percolate down dilakukan terhadap setiap non-leaf node secara reverse level order. Jika diketahui sebuah array terdiri dari 275 elemen, berapa kali proses percolate down dipanggil? (Tulis angkanya saja)
    Jawab: 137

15. Manakah pernyataan yang benar mengenai algoritma pengurutan heap sort?
    a. Dalam pengurutan ascending dengan heap sort, dilakukan proses remove setiap elemen dalam max-heap dan dipindahkan ke indeks terakhir heap.
    b. Heap sort memiliki kompleksitas worst case yang sama dengan quick sort.
    c. Seperti merge sort, heap sort juga melakukan pembagian array menjadi dua bagian sama besar, namun direpresentasikan dalam suatu binary tree.

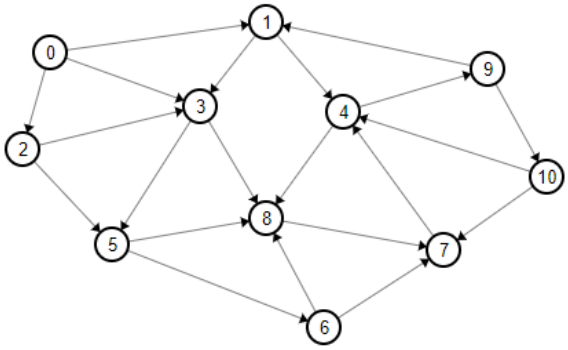
d. Jika heap sort digunakan untuk mengurutkan data yang terurut terbalik, proses heapify dapat dilakukan dalam kompleksitas O(log N), dengan N menyatakan jumlah data.

16. Traversal berdasarkan algoritma DFS dijalankan pada graph berikut.

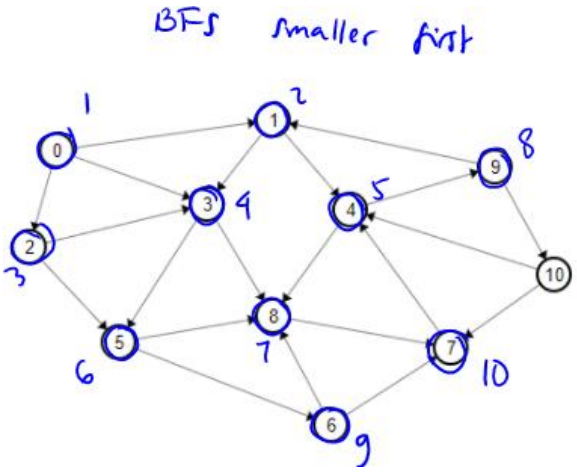


Algoritma akan melakukan pemilihan cabang yang akan diambil dilakukan **secara random**. Jika traversal dimulai dari 0, kemudian ke 3, lalu ke 8, dan seterusnya akan berakhir di verteks manakah traversal ini ?
Jawaban 6 (Penjelasan: urutannya 0 3 8 7 4 9 10 1 2 5 6, atau 0 3 8 7 4 9 1 10 2 5 6)

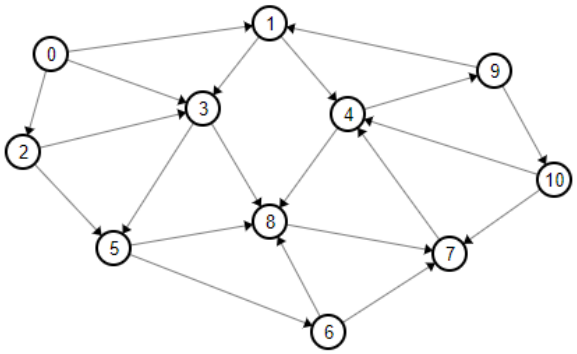
17. Traversal berdasarkan algoritma BFS dijalankan pada graph berikut. Ketika ada beberapa kemungkinan cabang yang dapat diambil, cabang dengan nomor verteks **lebih kecil** akan dipilih terlebih dahulu.



Jika traversal dimulai dari 0, yang kita sebut verteks 0 yang didatangi urutan pertama, verteks mana yang didatangi urutan ke 10?
Jawaban 7 (Penjelasan: urutannya 0 1 2 3 4 5 8 9 6 **7** 10)

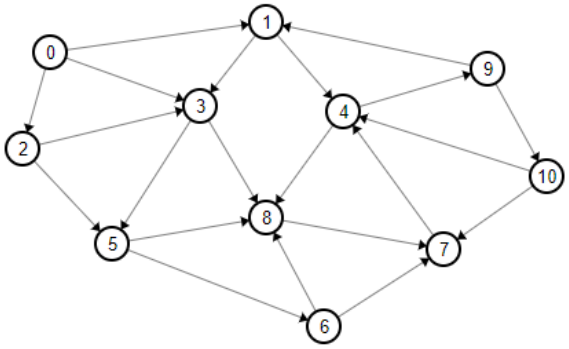


18. Traversal berdasarkan algoritma DFS dijalankan pada graph berikut. Ketika ada beberapa kemungkinan cabang yang dapat diambil, cabang dengan nomor verteks **lebih besar** akan dipilih terlebih dahulu.



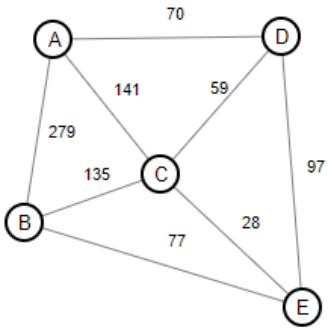
Jika traversal dimulai dari 0, yang kita sebut verteks 0 yang didatangi urutan pertama, verteks mana yang didatangi urutan ke 10?
Jawaban 6 (Penjelasan: urutannya 0 3 8 7 4 9 10 1 5 **6** 2)

19. Traversal berdasarkan algoritma BFS dijalankan pada graph berikut. Ketika ada beberapa kemungkinan cabang yang dapat diambil, cabang dengan nomor verteks **lebih kecil** akan dipilih terlebih dahulu.

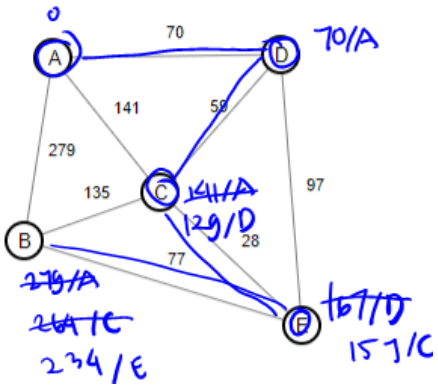


Jika traversal dimulai dari 0, yang kita sebut verteks 0 yang didatangi urutan pertama, verteks mana yang didatangi urutan ke 10?
Jawaban 7 (Penjelasan: urutannya 0 1 2 3 4 5 8 9 6 **7** 10)

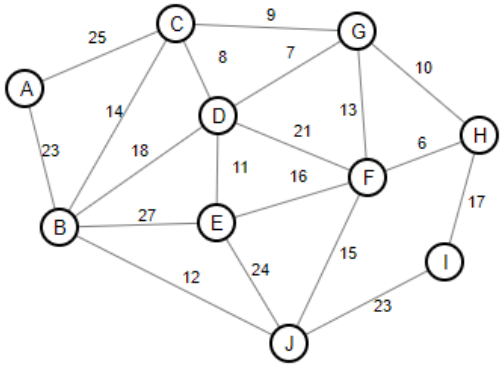
20. Pada graph berikut akan dijalankan algoritma mencari shortest path menurut algoritma Dijkstra untuk mencari jarak terpendek dari A ke setiap verteks lain. Algoritma Dijkstra menggunakan tabel Dist[v], dengan v adalah verteks, yang berisi jarak terpendek sementara dari A ke verteks tsb. Iterasi pertama adalah saat D masuk ke dalam “white-cloud” (shortest pathnya sudah ditemukan) lalu melakukan update pada Dist[C] dan Dist[E], sedangkan Dist[B] tetap pada harga sebelumnya yaitu 279.



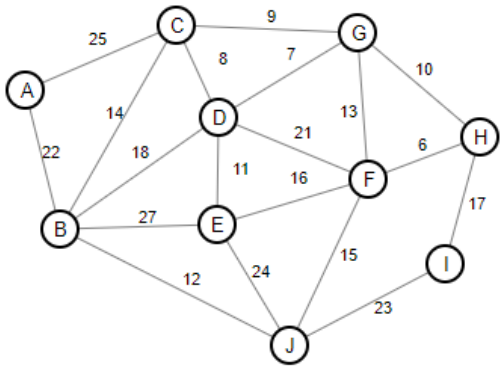
Pada saat E masuk ke dalam “white-cloud” dan melakukan update tabel Dist[v], berapakah nilai Dist[B]?
Jawab: 234 (penjelasan A-D-C-E-B)



21. [Graph: SS/Dikstra] Berapakah total jarak dari shortest path antara verteks B ke verteks H dari graph sebagai berikut?

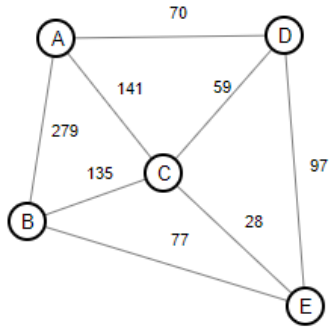


diralat menjadi



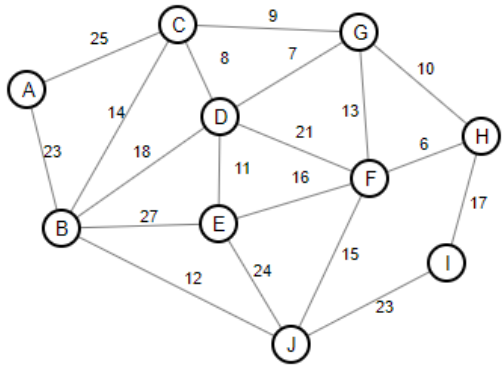
Jawab: 33 (penjelasan B-C-G-H)

22. Pada graph berikut akan dijalankan algoritma Prim untuk menemukan MST dan dimulai dari verteks B.

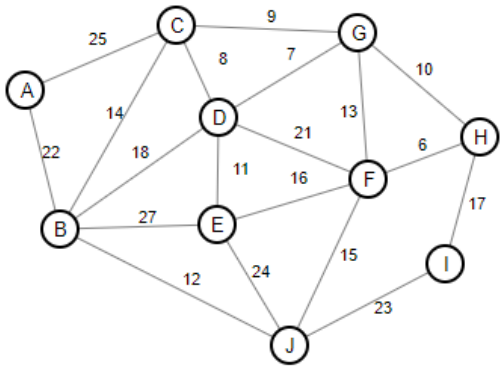


Jika sisi pertama yang dimasukkan ke dalam MST adalah BE, maka manakah sisi yang dimasukkan ke dalam MST pada urutan ke 3? Petunjuk: tulisakan nama sisi jawaban anda dengan dua huruf verteksnya dengan huruf pertama dan kedua sesuai urutan alfabet, misalnya AB adalah sisi menghubungkan A dan B.
Jawab: CD (Penjelasan BE, CE, CD, AD)

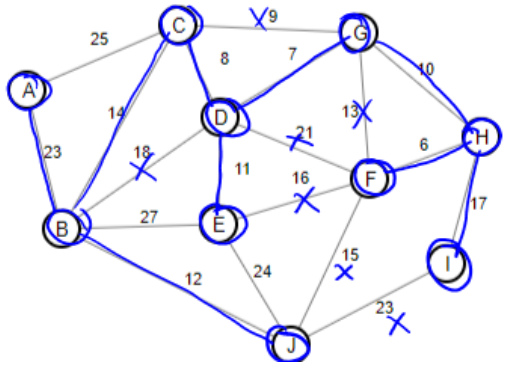
23. Pada graph berikut akan dilakukan pencarian MST menggunakan algoritma Kruskal.



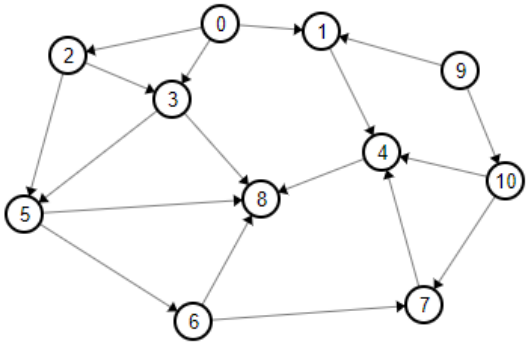
Diralat menjadi



Sebelum seluruh sisi yang tergabung dalam MST diperoleh, berapakah sisi yang direject karena dideteksi membentuk siklik?
Jawaban: 6



24. Pada graph berikut akan dilakukan Topological Sorting. Jika ada lebih dari satu verteks yang dapat dapat dipilih maka verteks dengan angka lebih kecil akan diambil lebih dulu.



Tuliskan deretan lengkap hasil pengurutannya sebagai nomor-nomor verteks dipisahkan 1 spasi. Misalnya: 0 1 2 3 4 5 6 7 8 9 10
Jawab: 0 2 3 5 6 9 1 10 7 4 8

25. Diketahui sebuah hash table berukuran 17 akan diisi dengan 6 key berturut-berturut sebagai berikut:

20 37 34 51 16 50

Jika H(key) = key % 17, berapa indeks untuk masing-masing key jika diberlakukan collision resolution sebagai berikut

1. Linear probing.
2. Quadratic probing.
3. Double hashing dengan H2(key) = 7 - (key % 7).

Di Scele akan dibuat close quiz dengan dropdown untuk tiap soal dan tiap key.

1. Linear probing
   20 % 17 = 3 (ok)
   37 % 17 = 3 (collision) - probing ke 4 (ok)
   34 % 17 = 0 (ok)
   51 % 17 = 0 (ok) - probing ke 1 (ok)
   16 % 17 = 16 (ok)
   50 % 17 = 16 (collision) - probing ke 0 (collision) - probing ke 1 (collision) probing ke 2 (ok)

2. Quadratic probing
   20 % 17 = 3 (ok)
   37 % 17 = 3 (collision) - probing ke 4 (ok)
   34 % 17 = 0 (ok)
   51 % 17 = 0 (collision) - probing ke 1 (ok)
   16 % 17 = 16 (ok)
   50 % 17 = 16 (collision) - probing ke 0 (collision) - probing ke 3 (collision) - probing ke 8 (ok)

3. Double hashing
   20 % 17 = 3 (ok)
   37 % 17 = 3 (collision)
   H2 = 7 - (37 % 7) = 7 - 2 = 5
   H = 3 + 5 = 8 (ok)
   34 = 0 (ok)
   51 = 0 (collision)
   H2 = 7 - (51 % 7) = 7 - 2 = 5
   H = 0 + 5 = 5
   ~~16 = 16 (ok)~~
   16 = 16 (ok)
   50 = 16 (collision)
   H2 = 7 - (50 % 7) = 7 - 1 = 6

   H = (16 + 6) % 17 = 5 (collision)
   H = (16 + 12) % 17 = 11

26. (gabung di atas)

27. (gabung di atas)

28. Diberikan sebuah hash table yang sudah berisi data. isActive merupakan penanda suatu cell telah diisi data atau belum. Jika suatu cell belum pernah diisi data, maka flag isActive false. Jika suatu cell berisi data dan isActive berstatus true, artinya data tersebut statusnya tidak dihapus. Namun jika suatu cell berisi data namun isActive berisi false, artinya data tersebut statusnya dihapus.

indeks	0	1	2	3	4	5	6	7	8	9
data	MN	KL		JK	RM			AB		DM
isActive	true	false	false	false	false	false	false	false	false	true

Dilakukan operasi penambahan data dengan nilai “RT”. Jika diketahui nilai hash code dari “RT” adalah 9, menggunakan quadratic probing, data “RT” akan ditempatkan pada cell dengan indeks …
(jawaban ditulis berupa angka indeks, misalnya 0)

Jawaban: 3

Commented [14]: Mei.. btw ini kayak ngasi clue ga ya, kan dihapus tu si JK, nah habis itu insert RT, ternyata posisinya di indeksnya JK yang tadi dihapus. Gimana kalo si JK nya ya udah statusnya false aja dari awal Mei, jadi ga ketauan ga nge-clue hehe

Commented [15]: siap

29. Diketahui 5 data dengan nilai key masing-masing sebagai berikut:
AB(key=15), DF(key=30),MN(key=41),JK(key=36),SJ(key=67)

Lima data tersebut akan dimasukkan secara berturut-turut pada hash table yang menerapkan open hashing dengan fungsi h(key) = key % 13. Jika setiap data yang memiliki nilai fungsi hash yang sama akan ditambahkan pada awal linked list, bagaimana urutan data pada linked list indeks ke 2 (urutan dari kiri ke kanan)?

a. SJ-MN-AB
b. AB-MN-SJ
c. SJ-DF-AB
d. DF-JK-SJ

Jawaban: A
penjelasan:
AB = 2
DF = 4
MN = 2
JK = 10
SJ = 2
urutan indeks 2: SJ-MN-AB

30. Pernyataan yang BENAR mengenai hash table adalah:
    a. Kompleksitas total operasi pencarian elemen di hash table adalah O(1) + O(collision resolution).
    b. Hash table efisien untuk mencari data terbesar atau terkecil.
    c. Kelebihan open addressing adalah dapat menyimpan data secara dinamis.
    d. Primary clustering merupakan kondisi di mana elemen-elemen yang menurut perhitungan fungsi hash ditempatkan pada lokasi yang sama dan diarahkan pada lokasi sel pengganti yang berbeda.

Jawaban: A

Commented [16]: ini salah satu aja mungkin ya, biar tidak dikira total dari ketiga operasi

atau bisa juga ditambah kata-kata "masing-masing"

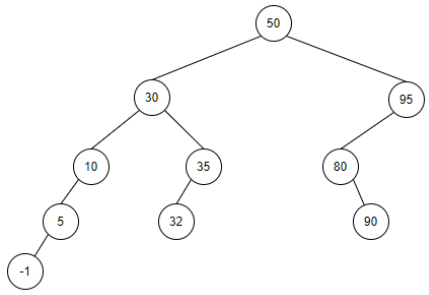
Commented [17]: ok

Bagian B

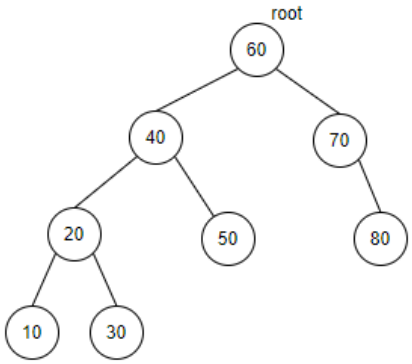
1. Hasil cetak post-order dari sebuah Binary Search Tree (BST) adalah sebagai berikut:
   -1, 5, 10, 32, 35, 30, 90, 80, 95, 50
   a. Node mana yang merupakan anak kiri dari root? Jelaskan dalam 3 - 4 kalimat.
   b. Berapakah tinggi tree tersebut?
   c. Tulislah hasil cetak pre-order-nya.

   Kunci Jawaban:
   a. 30. Karena post-order, maka yang dicetak terakhir yaitu 50 adalah root. Subtree kiri dari root adalah yang dicetak pertama kali yaitu semua elemen yang < 50: -1, 5, 10, 32, 35, 30. Berarti root dari subtree tersebut adalah 30, yang merupakan anak kiri dari 50.
   b. 4
   c. 50, 30, 10, 5, -1, 35, 32, 95, 80, 90

   Penjelasan: Dari hasil cetak post-order-nya, dapat dideteksi bentuk tree-nya:

Ket: ____: subtree kiri ____: subtree kanan R : root -1, 5, 10, 32, 35, 30, 90, 80, 95, 50 |_____| |_____| R -1, 5, 10, 32, 35, 30 |_____| |____| R dst..	Tree-nya: 

2. Class Node merepresentasikan sebuah node dalam Binary Search Tree (BST). Class Node memiliki Integer data, Node left, dan Node right. Lengkapilah method berikut ini untuk mencetak node-node yang berada di sisi kiri BST, mulai dari leaf sampai ke root (dari bawah ke atas).
   Contoh:
   Jika dilakukan pemanggilan cetakSisiKiri(root) untuk root pada tree berikut ini:



Maka hasil cetaknya adalah: 10, 20, 40, 60,

a. Lengkapi implementasi method cetakSisiKiri(Node n) berikut ini:

```
public void cetakSisiKiri(Node n){
        …
        …
        …
}
```

b. Berikan penjelasan mengenai algoritma dari method tersebut dalam 3 - 4 kalimat.

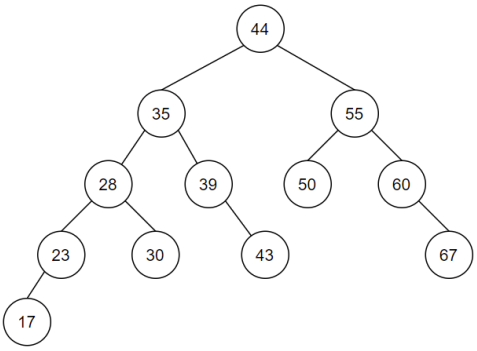
Kunci Jawaban:

a.

```
public void cetakSisiKiri(Node n){
        if(n.left != null)
                cetakSisiKiri(n.left);
        System.out.println(n.data);
}
```

b. Method tersebut memodifikasi method cetak post-order, hanya saja tidak melibatkan subtree kanan sama sekali. Sehingga pertama cetak subtree kiri terlebih dahulu. Jika sudah selesai, baru cetak root.

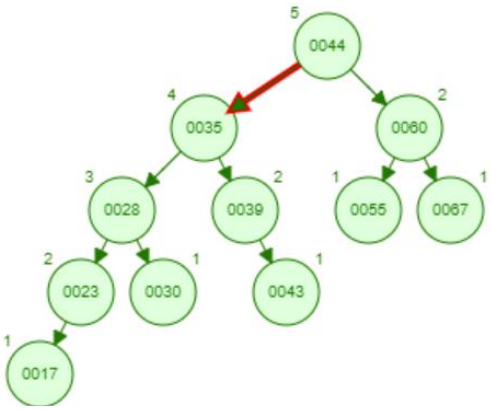
3. Perhatikan AVL Tree berikut ini. Operasi delete 50 dilakukan. Jelaskan langkah-langkah yang dilakukan untuk menghapus data 50 tersebut. Tuliskan step pengecekan node yang tidak balanced, rotasi yang dilakukan dan dilakukan pada node mana, serta bagaimana hasilnya. **(Jawaban gambar tree akhir saja tidak akan dinilai).**



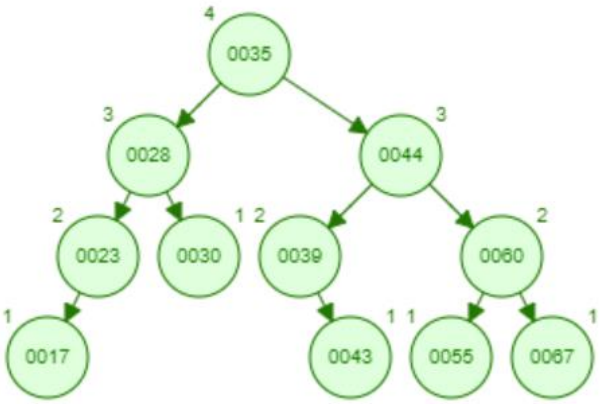
Catatan: Anda dapat mengunggah gambar langsung maupun menyertakan link menuju google drive (pastikan tanggal last modified adalah sebelum jam pelaksanaan UAS berakhir).

Jawaban:

- Node 50 langsung dihapus
- Periksa node 55. Tidak balanced, lakukan rotasi single pada 55. Hasilnya adalah:



- Periksa node 44 (root). Tidak balanced, lakukan single rotation pada 44. Hasilnya adalah:



4. Jelaskan permasalahan yang terjadi pada struktur data BST yang kemudian dapat diatasi dengan menggunakan struktur data AVL Tree.

5. Diberikan sebuah array tak urut berikut ini.
   21 - 17 - 43 - 65 - 14

   a. Simulasikan secara singkat algoritma heapify untuk mengubah array tersebut menjadi sebuah min-heap. Tuliskan proses yang terjadi serta hasil akhir yang diperoleh di setiap iterasi.
   b. Simulasikan secara singkat penambahan (insert) elemen 20 pada min-heap yang terbentuk dari soal (a). Tuliskan proses yang terjadi serta hasil akhir arraynya.
   c. Simulasikan secara singkat operasi remove min pada min-heap yang terbentuk dari soal (b). Tuliskan proses yang terjadi serta hasil akhir arraynya.

   Jawab:

   a. Simulasi min-heapify
      Awal: 21 - 17 - 43 - 65 - 14
      - Percolate down indeks 1; hasil: 21 - 14 - 43 - 65 - 17
      - Percolate down indeks 0; hasil: 14 - 17 - 43 - 65 - 21

      Hasil akhir: 14 - 17 - 43 - 65 - 21

   b. Simulasi insert 20
      - Tambah elemen 20 di indeks terakhir; hasil: 14 - 17 - 43 - 65 - 21 - 20
      - Percolate up indeks terakhir; hasil: 14 - 17 - 20 - 65 - 21 - 43

   c. Simulasi remove min
      - Pindahkan elemen di indeks terakhir ke indeks 0; hasil: 43 - 17 - 20 - 65 - 21
      - Percolate down indeks 0; hasil: 17 - 21 - 20 - 65 - 43

6. Perhatikan potongan kode berikut ini.

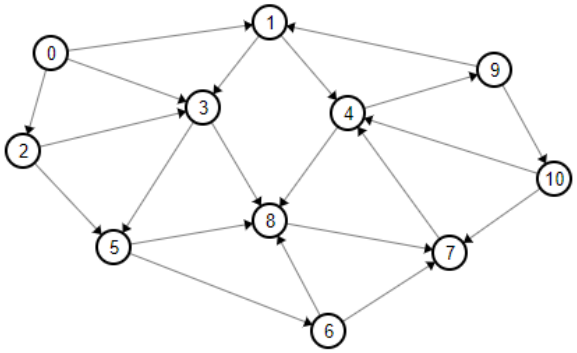
```
boolean mystery(int[] heap) {
    for (int i = 0; i < heap.length / 2; i++) {
        int j = (i * 2) + 1;
        if (j < heap.length && heap[i] < heap[j++])
            return false;
        if (j < heap.length && heap[i] < heap[j])
            return false;
    }
    return true;
}
```

a. Jika heap = {13, 20, 17, 51, 23, 30, 45}, apa luaran dari method mystery? Jelaskan alasannya secara singkat dan spesifik.
b. Jika heap = {73, 45, 59, 34, 17, 20, 63}, apa luaran dari method mystery? Jelaskan alasannya secara singkat dan spesifik.
c. Jika heap = {85, 20, 70, 18, 10, 12, 60}, apa luaran dari method mystery? Jelaskan alasannya secara singkat dan spesifik.
d. Secara umum, apa yang dilakukan oleh method mystery?

Jawab:

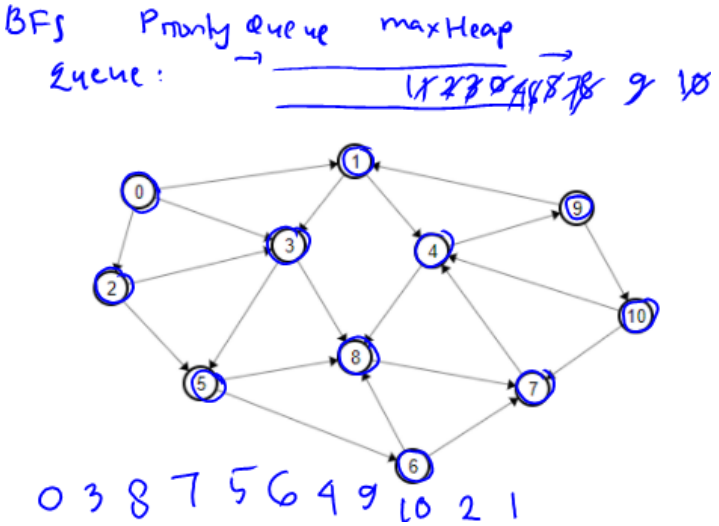
a. False karena indeks 0 (elemen 13) < indeks 1 (elemen 20).
b. False karena indeks 2 (elemen 59) < indeks 6 (elemen 63).
c. True karena semua perbandingan elemen yang dilakukan bernilai false.
d. Method mystery mengembalikan true jika array heap merupakan max-heap. Jika heap bukan max-heap, method mengembalikan false.

7. Graph berikut ini akan di-traverse dengan algoritma MaxPQ-BFS, yaitu modifikasi BFS dengan menggunakan priority-queue dengan kaidah maxheap. Setiap kali sebuah verteks di-visit (di-dequeue dari priority-queue), maka semua verteks yang adjacent dengannya dan belum di-visit akan di-enqueue ke dalam priority queue (perhatikan kaidah maxheap di sini). Dengan demikian, pada setiap iterasi algoritma MaxPQ-BFS, verteks bernomor paling besar dalam priority-queue yang akan dikunjungi lebih dulu.

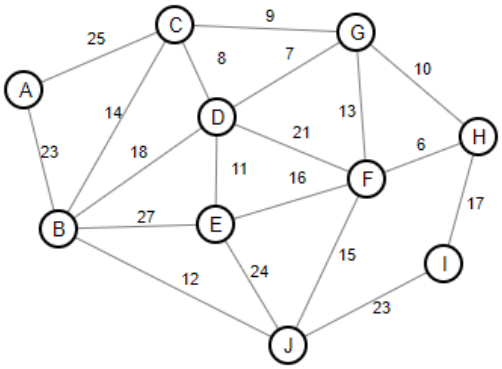


Dapatkan urutan verteks yang dikunjungi menggunakan modifikasi BFS ini dimulai dari verteks 0. Beri penjelasan singkat atas jawaban Anda.

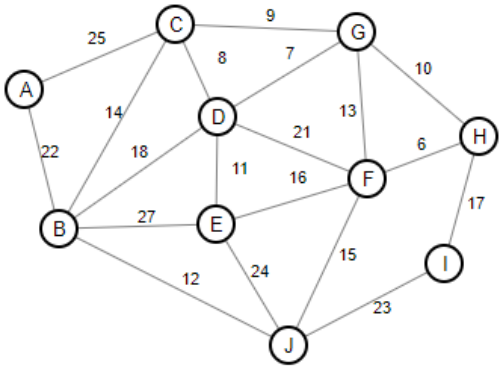
Jawaban: 0 3 8 7 5 6 4 9 10 2 1



8. Jalankan algoritma Shortest Path Dijkstra dalam menemukan setiap shortest path dari verteks H ke setiap verteks lainnya dari graph berikut ini.



Diralat menjadi



a. Lengkapi tabel Dist berikut ini sesuai yang diperoleh algoritma.
Petunjuk:
- Pada setiap iterasi, untuk setiap verteks yang berada di dalam queue tuliskan nilai C/P dengan C adalah cost saat ini untuk mencapai verteks tersebut, dan P adalah predecessor atau dari verteks mana C di-update
- Black area (di slide disebut area hitam, yaitu simpul yang sama sekali belum dihitung) dikosongkan saja
- White-cloud (di slide disebut area hijau) digarisbawahi atau dilingkari.

Ver-teks	Ini-siali-sasi	Iterasi ke									
		1	2	3	4	5	6	7	8	9	10
A											
B											
C											
D			27/F								
E			22/F								
F		6/H	6/H								
G		10/H	10/H								
H	0	0	0								
I		17/H	17/H								
J			21/F								

b. Tuliskan lintasan dari H ke B beserta total bobot dari shortest path dari H ke B tersebut.

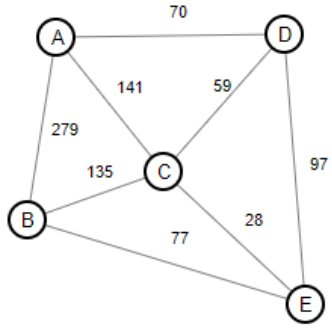
Jawaban:

a.

Ver-teks	Ini-siali-sasi	iterasi									
		1	2	3	4	5	6	7	8	9	10
A							44/C	44/C	44/C	44/C	44/C
B					35/D	35/D	33/C	33/C	33/C	33/C	33/C
C				19/G	19/G	19/G	19/G	19/G	19/G	19/G	19/G
D			27/F	17/G	17/G	17/G	17/G	17/G	17/G	17/G	17/G
E			22/F	22/F	22/F	22/F	22/F	22/F	22/F	22/F	22/F
F		6/H	6/H	6/H	6/H	6/H	6/H	6/H	6/H	6/H	6/H
G		10/H	10/H	10/H	10/H	10/H	10/H	10/H	10/H	10/H	10/H
H	0	0	0	0	0	0	0	0	0	0	0
I		17/H	17/H	17/H	17/H	17/H	17/H	17/H	17/H	17/H	17/H
J			21/F	21/F	21/F	21/F	21/F	21/F	21/F	21/F	21/F

b. H-G-C-B dengan total bobot 33.

9. Graph berikut menunjukkan peta biaya pengiriman logistik melalui jalan raya antar 5 kota di suatu provinsi. Dikabarkan telah terjadi bencana alam di TEPAT salah satu ruas jalan tersebut namun informasi ruas jalan manakah itu tidak diketahui karena gangguan komunikasi yang terjadi.



Tim penanggulangan bencana dari kota A akan mendistribusikan bantuan ke kota-kota lainnya. Untuk perencanaan perlu diketahui total biaya pengiriman terburuk yang bisa terjadi.

Commented [18]: soal ini memang bobot kesulitannya agak tinggi ya Pak :D

untuk soal ini sepertinya weight-nya terlalu besar Pak..lumayan waktunya jika menghitung manual penjumlahan bobot-bobotnya saat menjalankan dijkstra

lalu soal ini mungkin agak ambigu juga Pak, misalnya apakah ketika menyalurkan bantuan dari kota A ke 4 kota lain itu kendaraannya ada 4 yang jalan sendiri-sendiri mencari shortest path ke masing-masing kota, apakah bisa jadi hanya ada 1 kendaraan yang melewati semua kota secara berurutan jika memungkinkan (dan ini tentunya cost-nya lebih hemat)

a. Ruas jalan manakah yang jika bencana terjadi di situ maka akan menimbulkan total biaya pengiriman logistik paling besar?
b. Uraikan ide bagaimana cara Anda untuk mendapatkan solusinya secara efisien.

Jawaban:

a. BE
b. idenya dasarnya mencari shortest path terburuk dengan salah satu jalan dihilangkan tapi mencoba semua sisi dihilangkan demikian tidak efisien. Karena kemungkinan terburuk terjadi haruslah karena sisi yang termasuk dalam shortest path dari A ke B tsb yang dihilangkan karena kalau ruas lain maka shortest path tidak berubah. Jadi, dari setiap sisi XY dalam shortest path tsb coba hilangkan, lalu hitung kembali shotest pathnya, dan dari semua shortest path yang telah dihitung cari yang paling besar total biayanya.)

10. Diberikan sebuah hash table yang sudah berisi data. isActive merupakan penanda suatu cell telah berisi data atau belum. Jika suatu cell belum pernah diisi data, maka flag isActive berisi false. Jika suatu cell berisi data dan isActive berstatus true, artinya data tersebut statusnya tidak dihapus. Namun, jika suatu cell berisi data namun isActive berisi false, artinya data tersebut statusnya dihapus.

indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	26	14	28		39	57			17	52			25
isActive	true	true	false	false	true	true	false	false	true	false	false	false	true

Fungsi hash yang digunakan pada hash table adalah `H(x) = x % 13.`

a. Dari data yang sudah ada pada gambar hash table di atas, identifikasi data mana saja yang tadinya mengalami collision saat di-insert sehingga dilakukan probing sehingga mendapatkan posisi seperti pada gambar?

26 % 13 = 0
14 % 13 = 1
28 % 13 = 2
39 % 13 = 0 - collision, +1 collision, + 4 ok
57 % 13 = 5
17 % 13 = 4 (collision), 4+1 collision, 4+4 ok jadi 8
52 % 13 = 0 (collision), + 1, +4 collision, +9 ok
25 % 13 = 12

Jadi yang mengalami collision adalah data 39,17 dan 52

b. Collision resolution apa yang diterapkan pada hash table di atas? Jelaskan alasannya.

Collision resolution yang diterapkan adalah quadratic probing. Bisa dilihat pola probingnya pada penjelasan a yaitu +1, +4, +9 sehingga menunjukkan pola quadratic probing

c. Secara berturut-turut, dilakukan operasi sebagai berikut:
   - penambahan data 40
   - penghapusan data 17
   - penambahan data 30
   - penambahan data 64

   Data 40, 30 dan 64 akan ditempatkan pada indeks berapa? Berikan penjelasannya.

   40 % 13 = 1 (collision)
   1 + 1 = 2 isactive false sehingga 40 ditempatkan pada indeks 2.
   30 % 13 = 4 (collision)
   4 + 1 = 8 collision
   4 + 4 = 8 status inactive false setelah penghapusan 17, sehingga data 30 dapat diprobing ke indeks 8
   64 % 13 = 12 (collision)
   (12 + 1) % 13 = 0 (collision)
   (12 + 4) % 13 = 3, sehingga data 64 dapat diprobing ke indeks 3

Commented [19]: eh Mei sorry yang di atas aku ganti isActive semua, typo kan ya? karena kalo nama flag-nya inActive takutnya bingung karena di deskripsi dibilang kalau true artinya data aktif

Commented [20]: iya typo. makasih kak

11. Suatu hash table mengimplementasikan open hashing sebagai collision resolution. Struktur data hash table diimplementasikan dengan array dengan objek bertipe HashEntry. Objek HashEntry menyimpan atribut objek element yang berisikan data dan pointer ke objek HashEntry berikutnya.

```
public class MyHash{
     private static class HashEntry{
         public Object element; //elemen
         public HashEntry next; //linked list chaining

         //constructor
         public HashEntry(Object e){
            this(e, null);
         }

         public HashEntry{Object e, HashEntry n}{
            element = e;
            next = n;
         }
     }

    //mengembalikan jumlah items pada this collection
    public int size( )
    {
        return currentSize;
    }

    //method untuk menambahkan objek x ke array[currentPos]  di hash table
    public boolean add(Object x){
      if (getMatch(x))
            return false;
      int currentPos = x.hashCode();
```

```
      /**code untuk menambahkan objek x akan disimpan pada array[currentPos]
      sebagai objek HashEntry baru dan ditempatkan pada akhir linked list
      **/

     ------------------------------------------------------
     currentSize++;
     return true;
   }
}
//method getMatch(Object x) digunakan untuk mengecek apakan objek x sudah ada
pada array hash table atau belum.
//method hashCode() digunakan untuk menghitung posisi object x berdasarkan
fungsi hash
```

Tuliskan code pada bagian yang ditandai dengan tinta merah untuk menambahkan objek x pada array[currentPos] sebagai objek HashEntry baru yang ditempatkan pada akhir linked list.

Jawaban:

~~HashEntry entryBaru = new HashEntry(x, null);~~

~~//untuk mendapatkan elemen terakhir~~
~~while (array[currentPos].next != null){~~
~~array[currentPos] = array[currentPos].next;~~
~~}~~

~~array[currentPos].next = entryBaru;~~

```
HashEntry current = array[currentPos];
//jika masih kosong
if (current == null) current = new HashEntry(x,null);
else
{

        //jika linked-list sudah ada & perlu mendapatkan node terakhir
        while (current.next  != null){
                current = current.next;
        }

        current.next = new HashEntry(x,null);
}
```