E

E E

E E E E

E E E E E E E E

E E E E E E E E E E E E E E E E E E E E

Heavy Light Decomposition

Jonathan Irvin Gunawan
National University of Singapore

prerequisite

tau tree

tau segment tree/ BIT

ga buta warna

serius, bakal ada warna warni di slide ini

bisa ngitung

sebelum belajar,
kasih motivasi dulu

dikasih tree. tiap node ada value

ada Q query, dalam bentuk (a,b)

tentuin berapa jumlah value untuk semua node yang berada di path (a,b)

LCA doang

dikasih tree. tiap node ada value

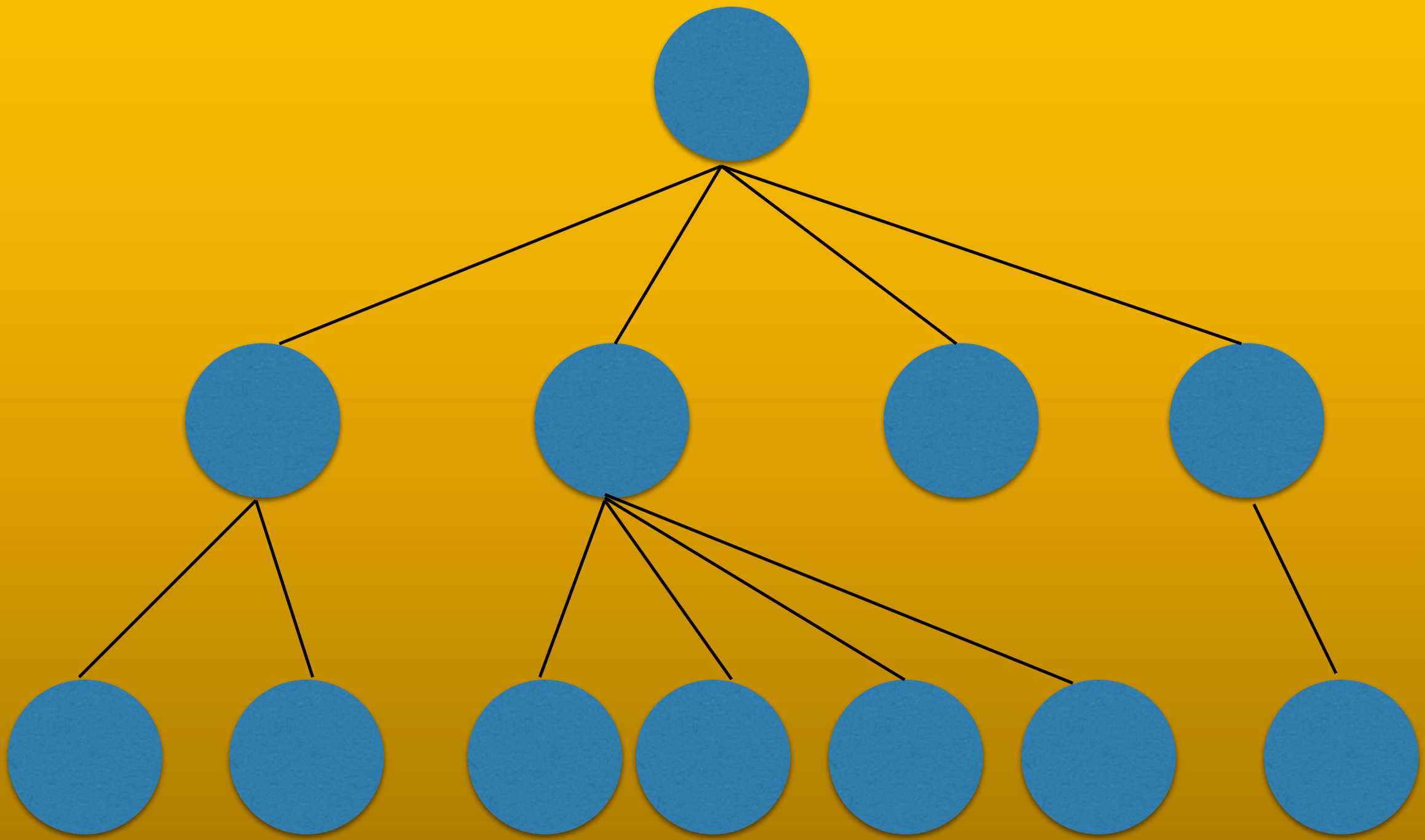
ada Q query, dalam bentuk (a,b)
tentuin berapa jumlah value untuk semua node yang berada di path (a,b)
tapi ditengah2 query bisa update value juga

HLD

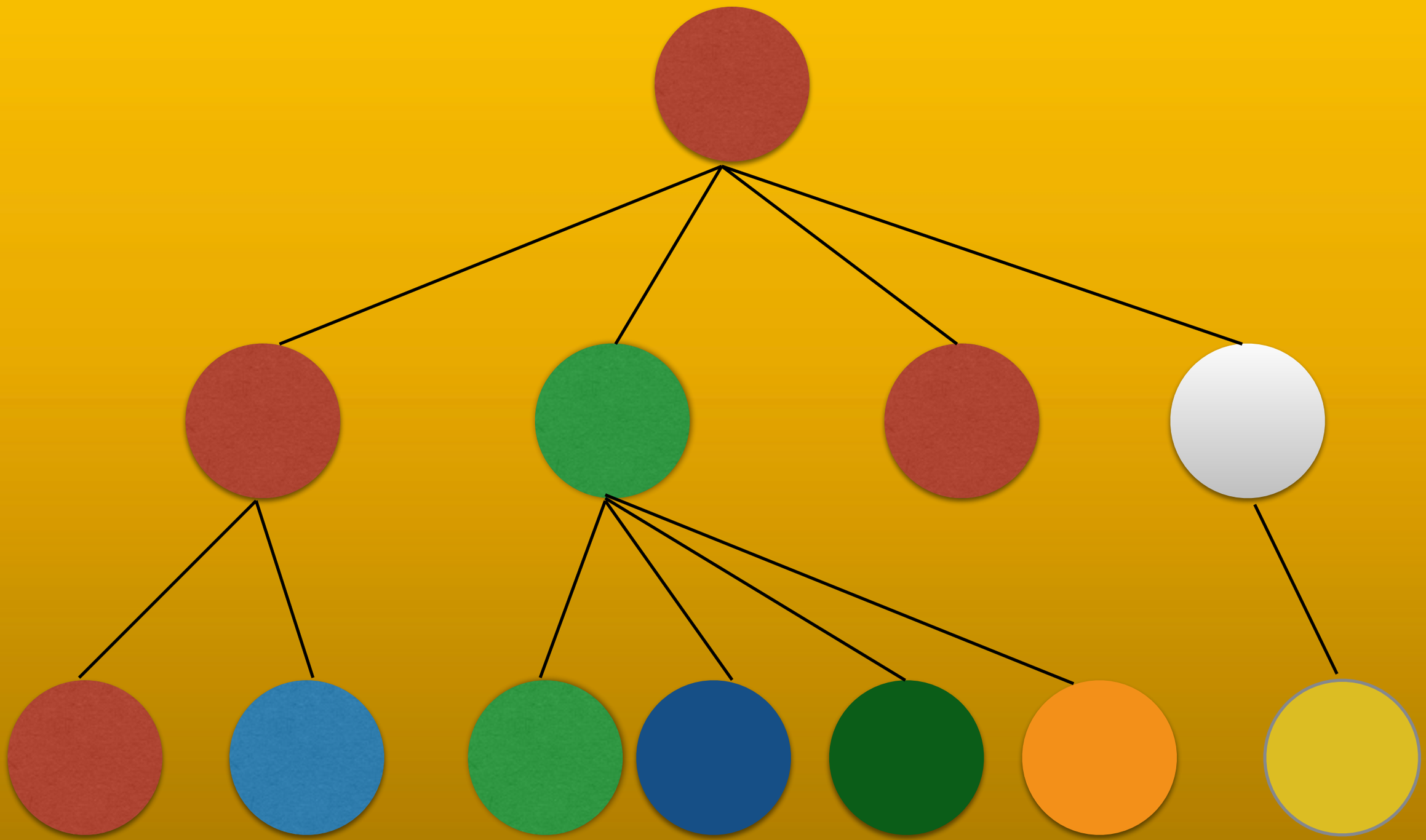
ngedecompose tree jadi beberapa path

setiap path dapat direpresentasikan
{v_1, v_2, v_3, v_4, ...}
dimana v_i = parent[v_i+1]

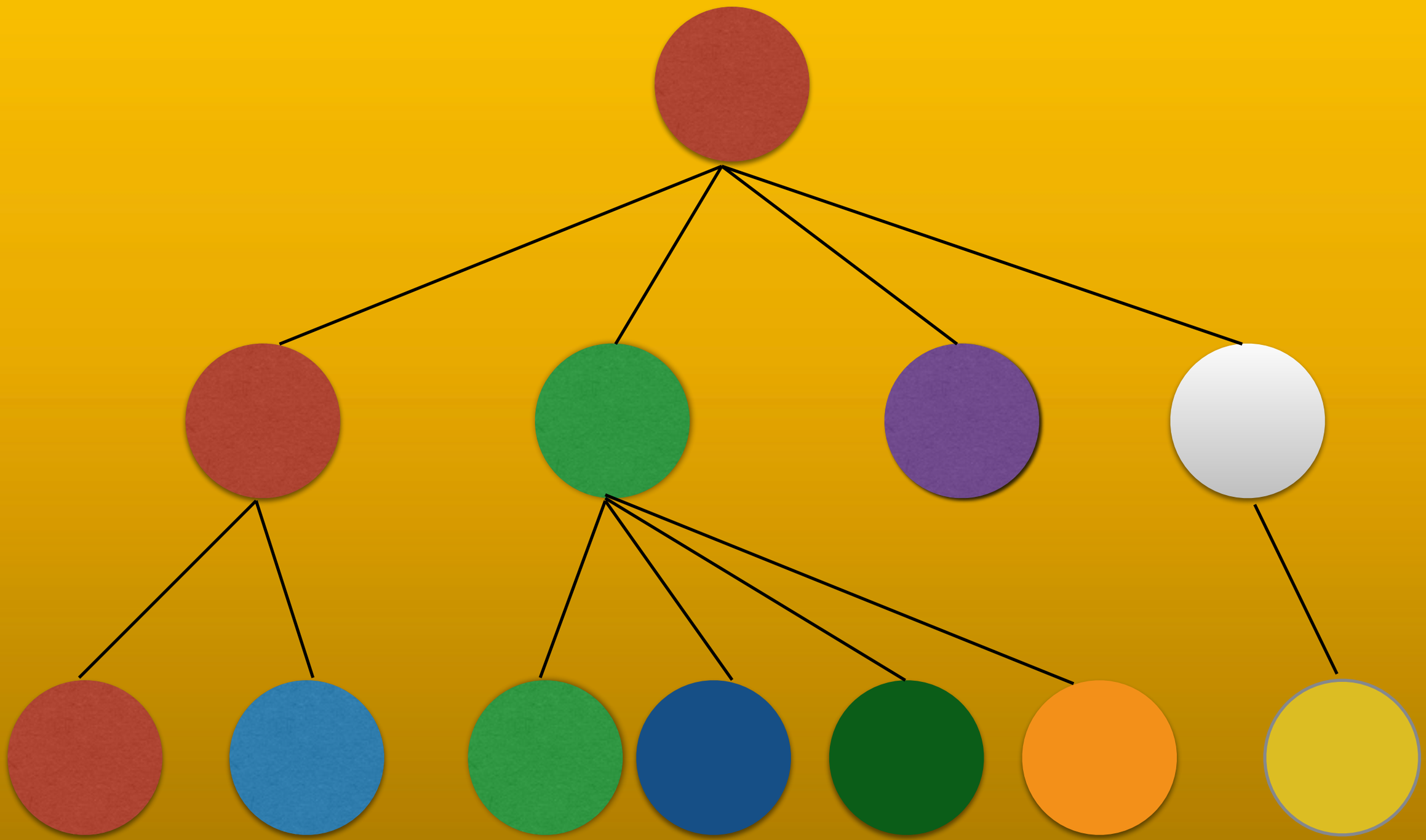
contoh



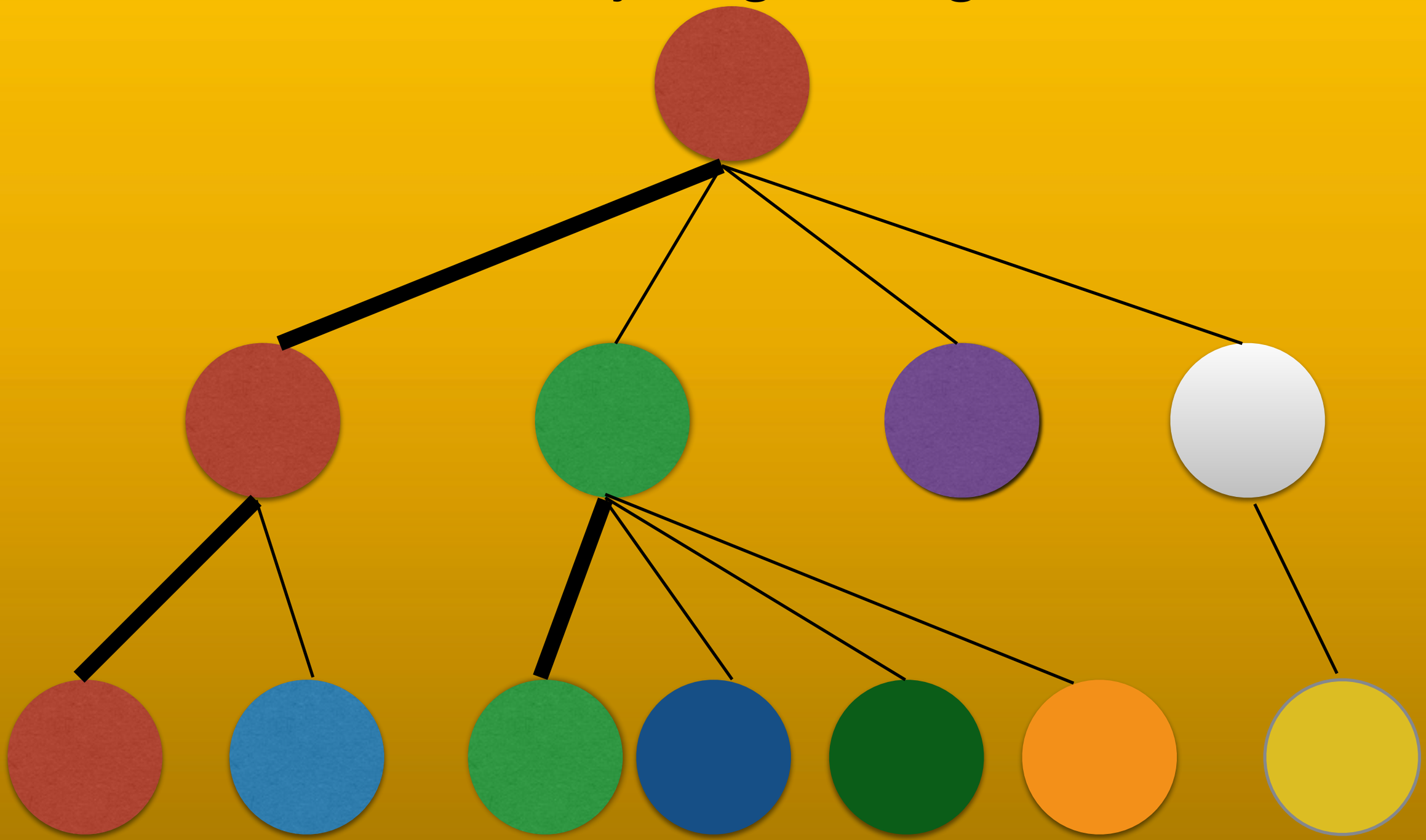
dekomposisi gak valid



dekomposisi valid



edge yang menghubungkan dua node dalam satu komponen kita sebut heavy edge sisanya light edge



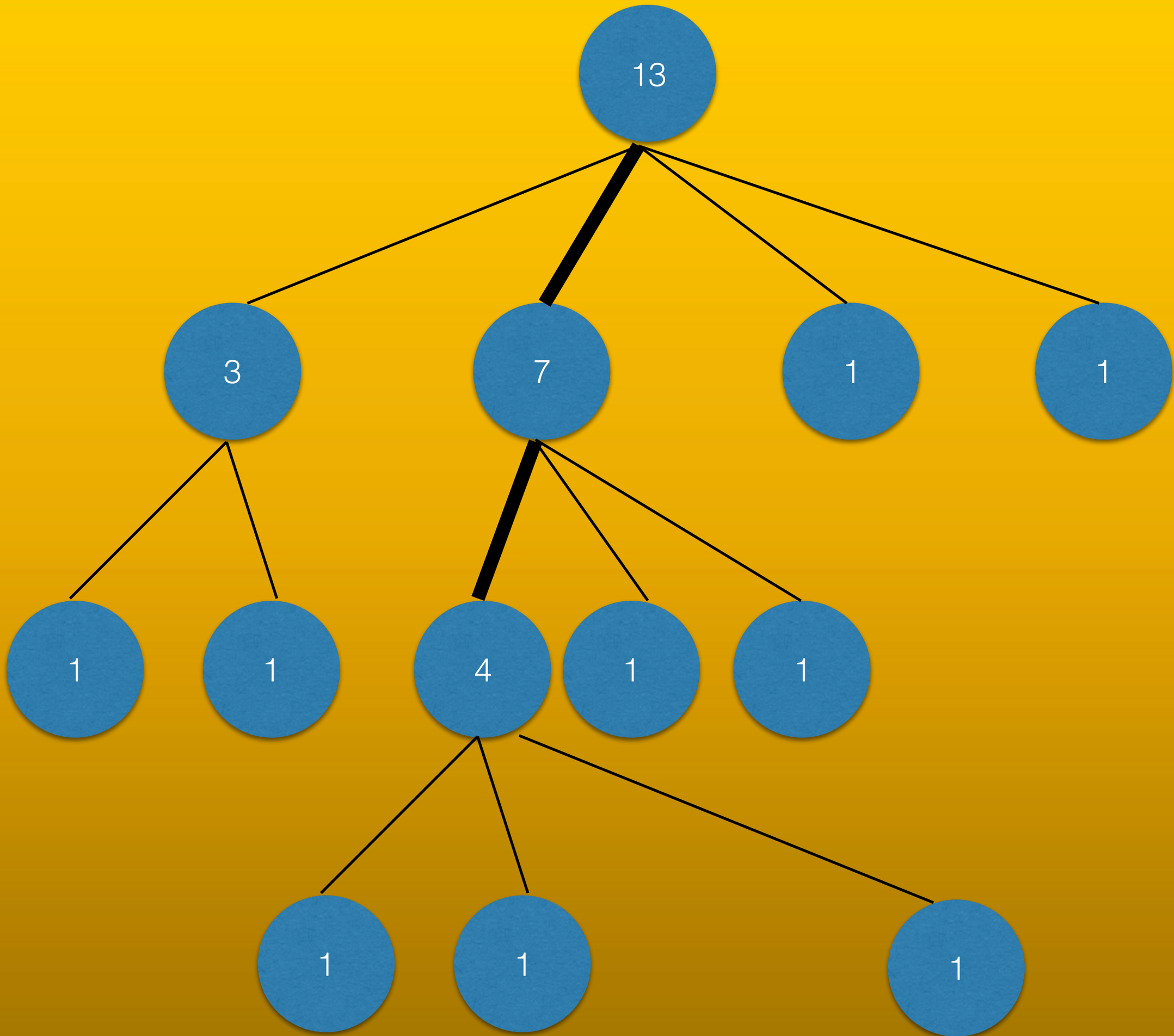
syarat dekomposisi yang kita mau

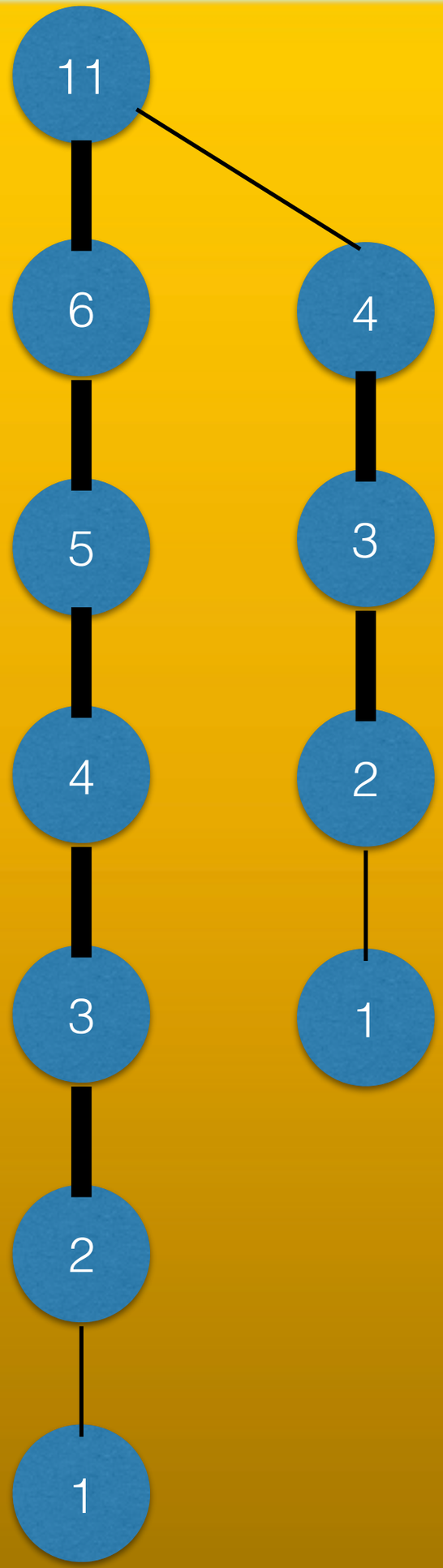
untuk setiap node v, banyaknya light edge dari v ke root ≤ lg(N)

cara dekomposisi

untuk setiap node u :

untuk setiap node v child dari u, jika dan hanya jika size(v) > 1/2 * size(u), maka (u,v) heavy edge


13
3
7
1
1
1
1
4
1
1
1
1
1


11
6
4
5
3
4
2
3
1
2
1

cek syarat

1. pasti setiap komponennya berupa path

iya dong. kan setiap node u pasti cuma at most dua heavy edge yang incident sama dia, ke parent sama **ke at most satu child**

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

pasti gak bisa ngeproofnya

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

assume enggak.
assume ada > lg(N) light edges dari root ke node u.

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

misalkan path dari root ke node u itu
V = {v_1, v_2, v_3, …, u}. V > lg(N)
w.l.o.g. assume semuanya dihubungin sama light edge

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

maka size(v_2) < 1/2 size(v_1)

size(v_3) < 1/2 size(v_1)

size(v_4) < 1/2 size(v_3)

...

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

maka size(v_2) < 1/2 size(v_1)

size(v_3) < 1/4 size(v_1)

size(v_4) < 1/8 size(v_1)

....

size(u) < 1/n size(v_1)

cek syarat

2. dari setiap node, jalan ke root cuma ada ≤ lg(N) light edges

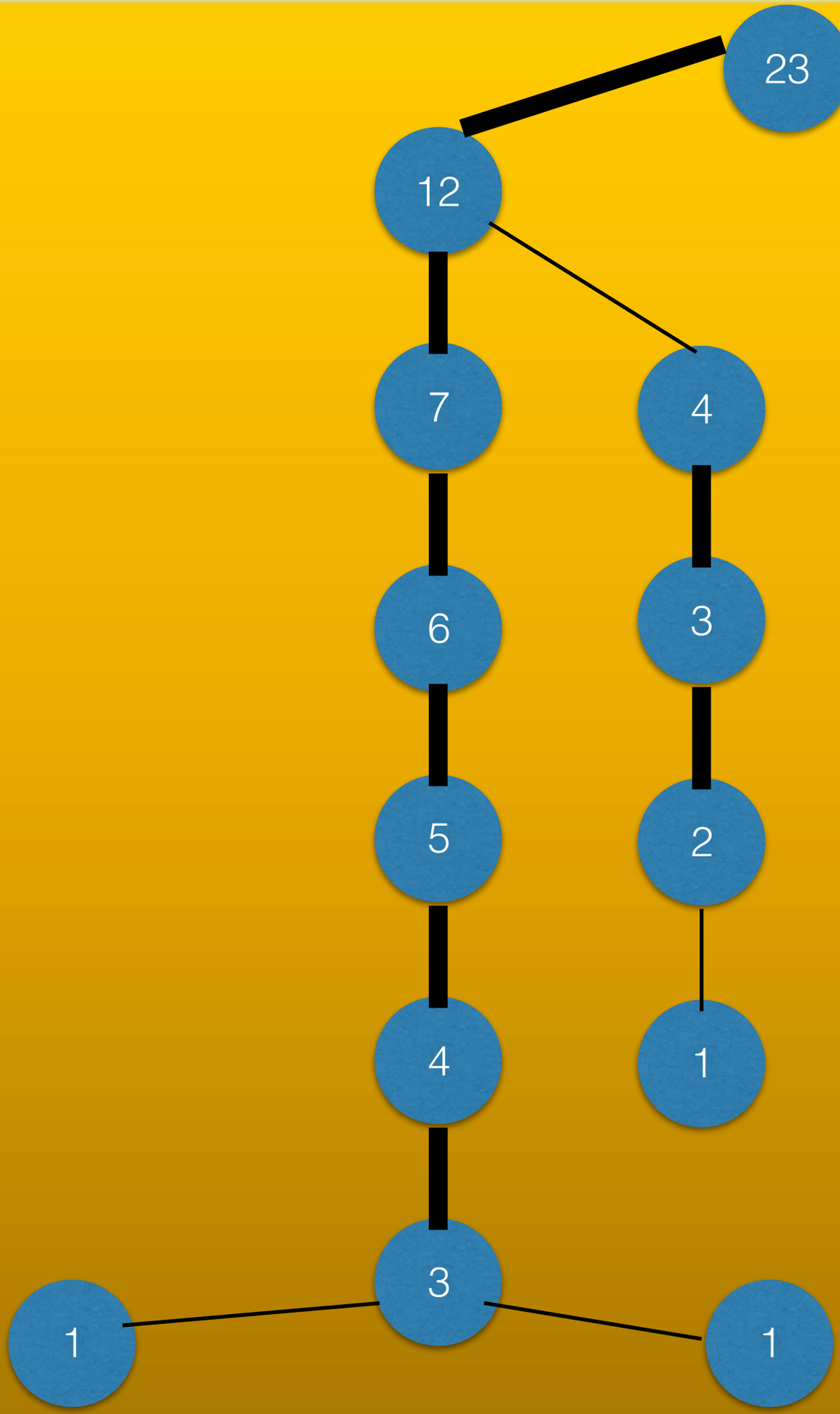
ga mungkin kan
size(u) < 1/n size(v_1)

kontradiksi

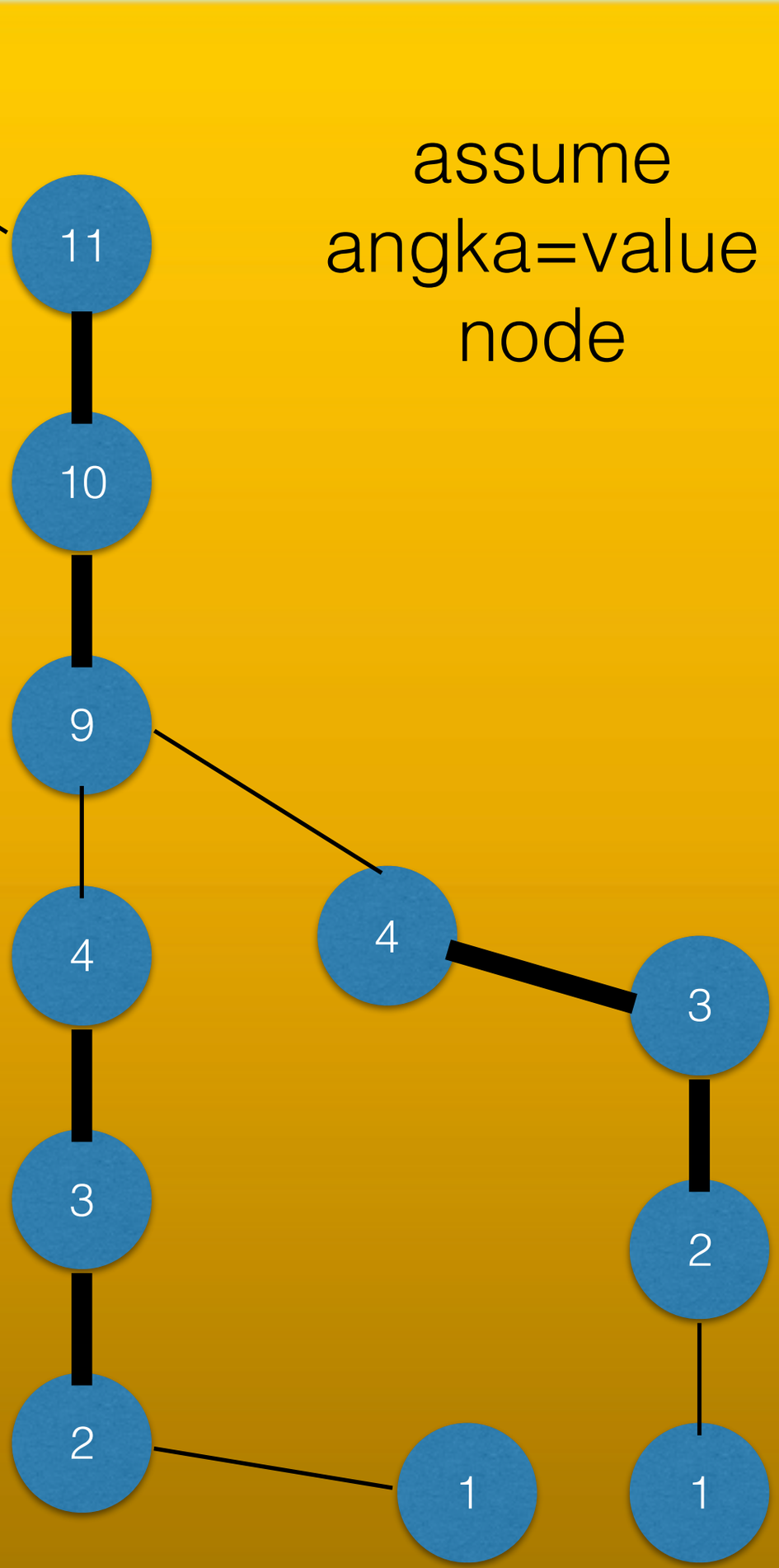
nah basic ideanya adalah

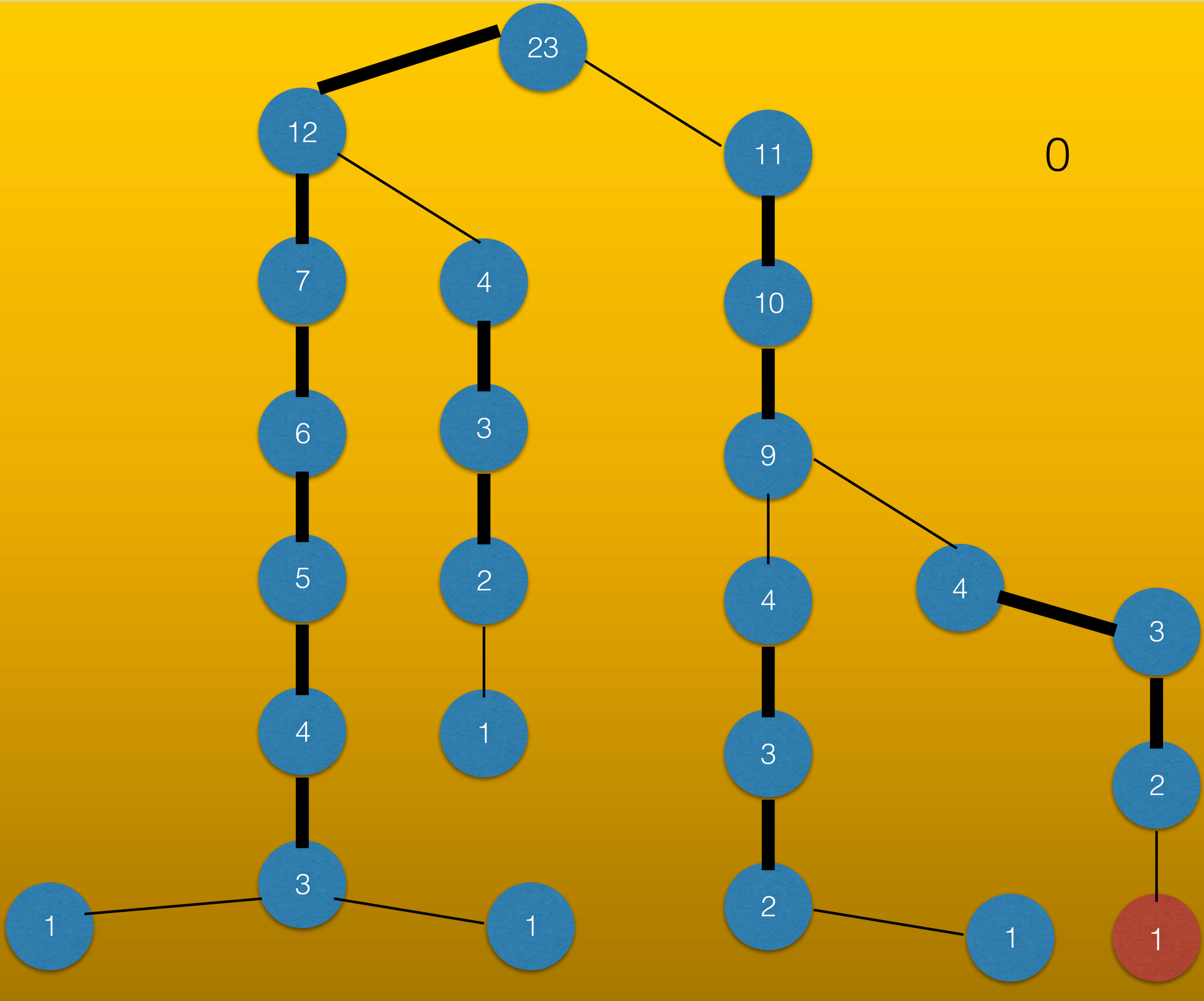
untuk tiap node, jalan ke rootnya
“ngeskip” heavy edge

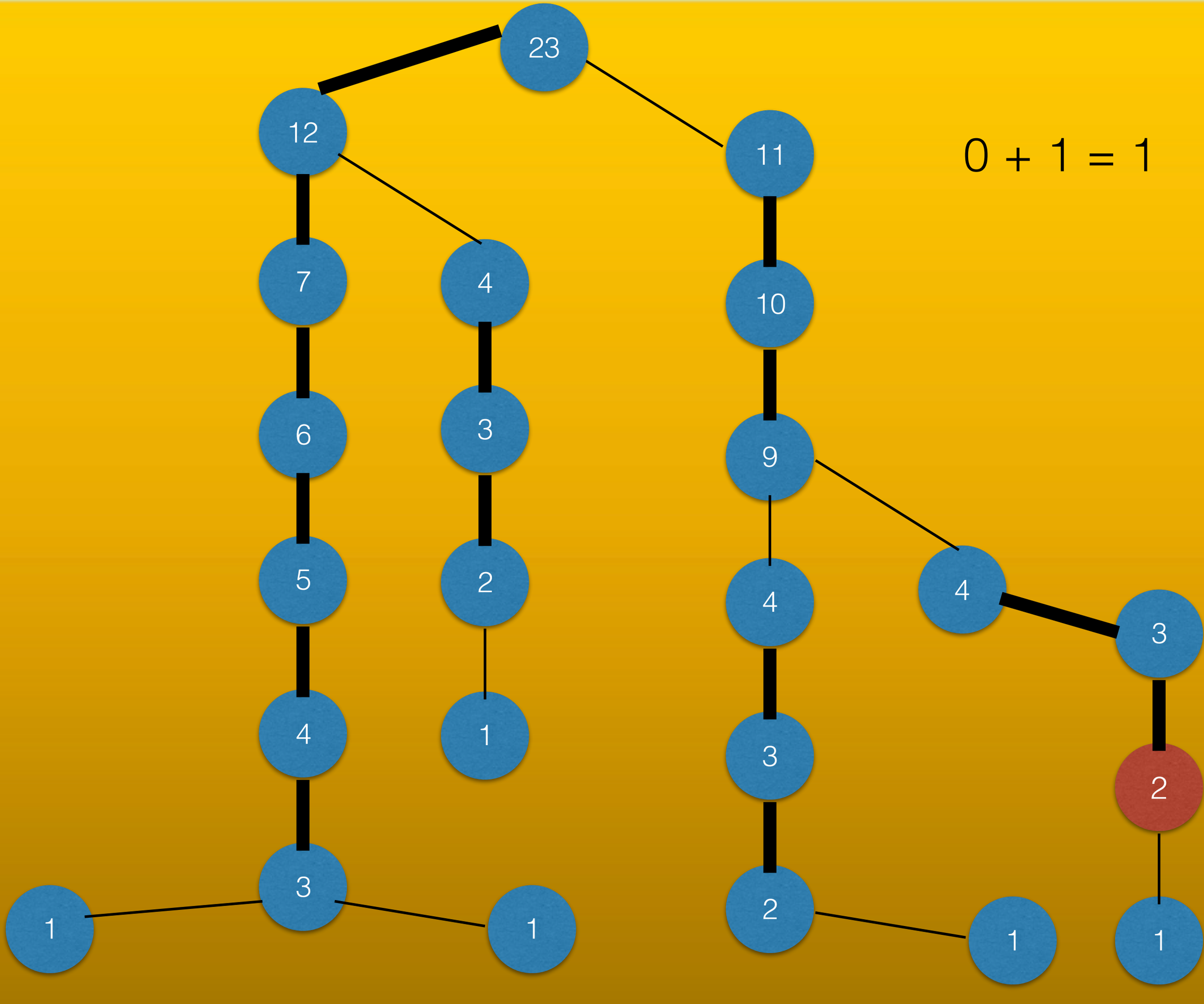
problem yang tadi bisa kita simplify seluruh querynya itu ke root kan?

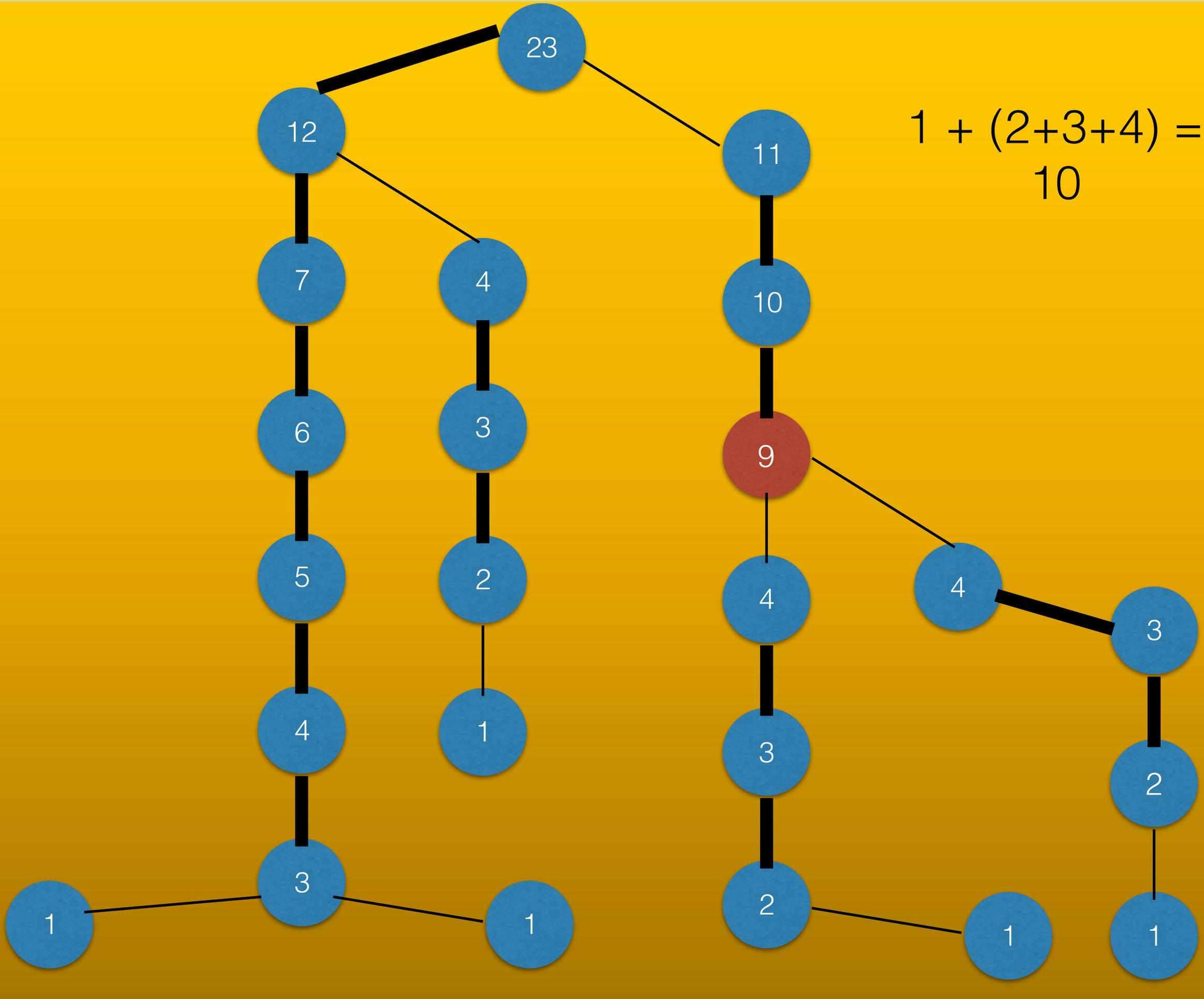


assume
angka=value
node

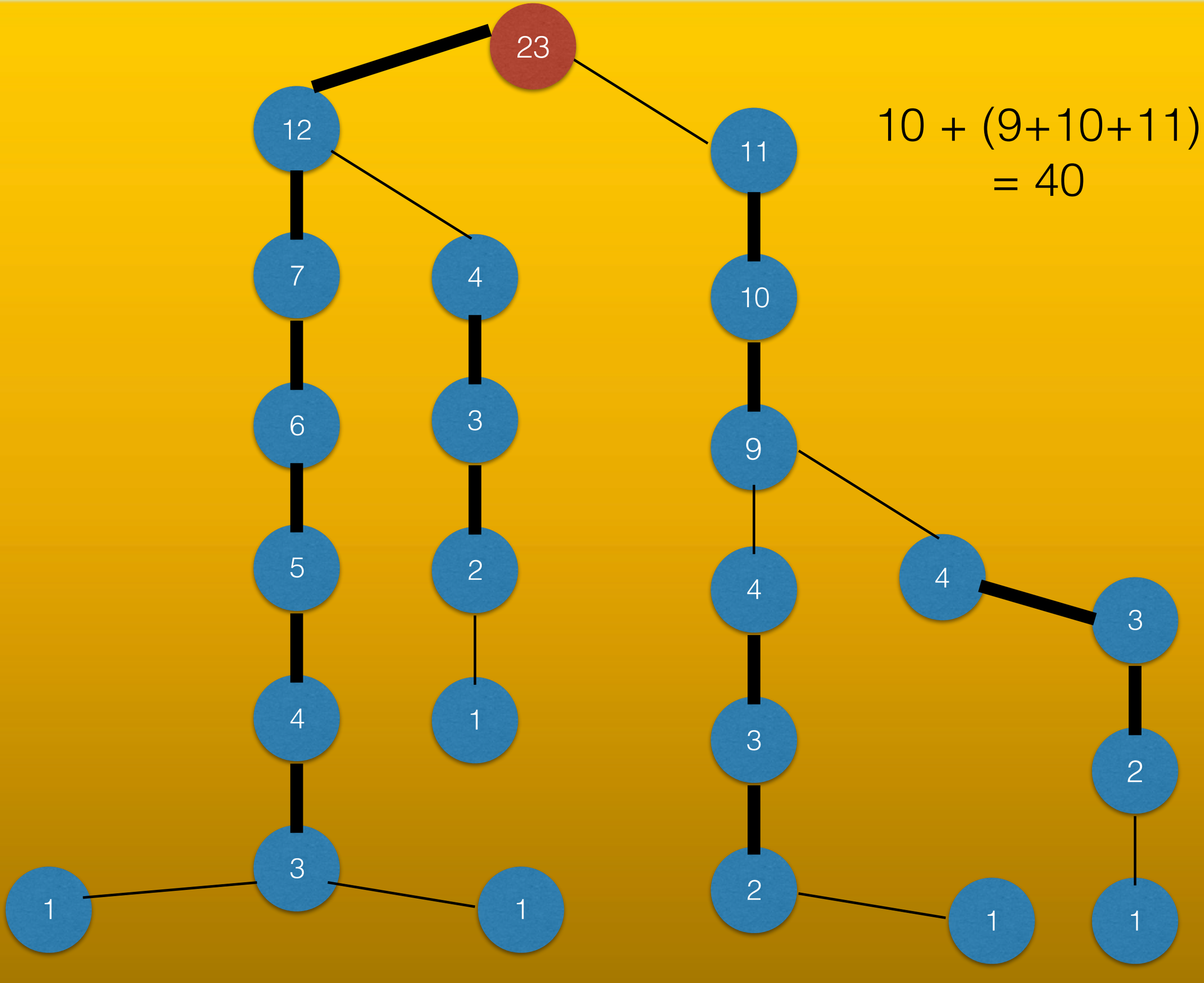



23
12
11
7
4
10
6
3
9
5
2
4
4
1
3
3
2
1
1
1
2
1
0

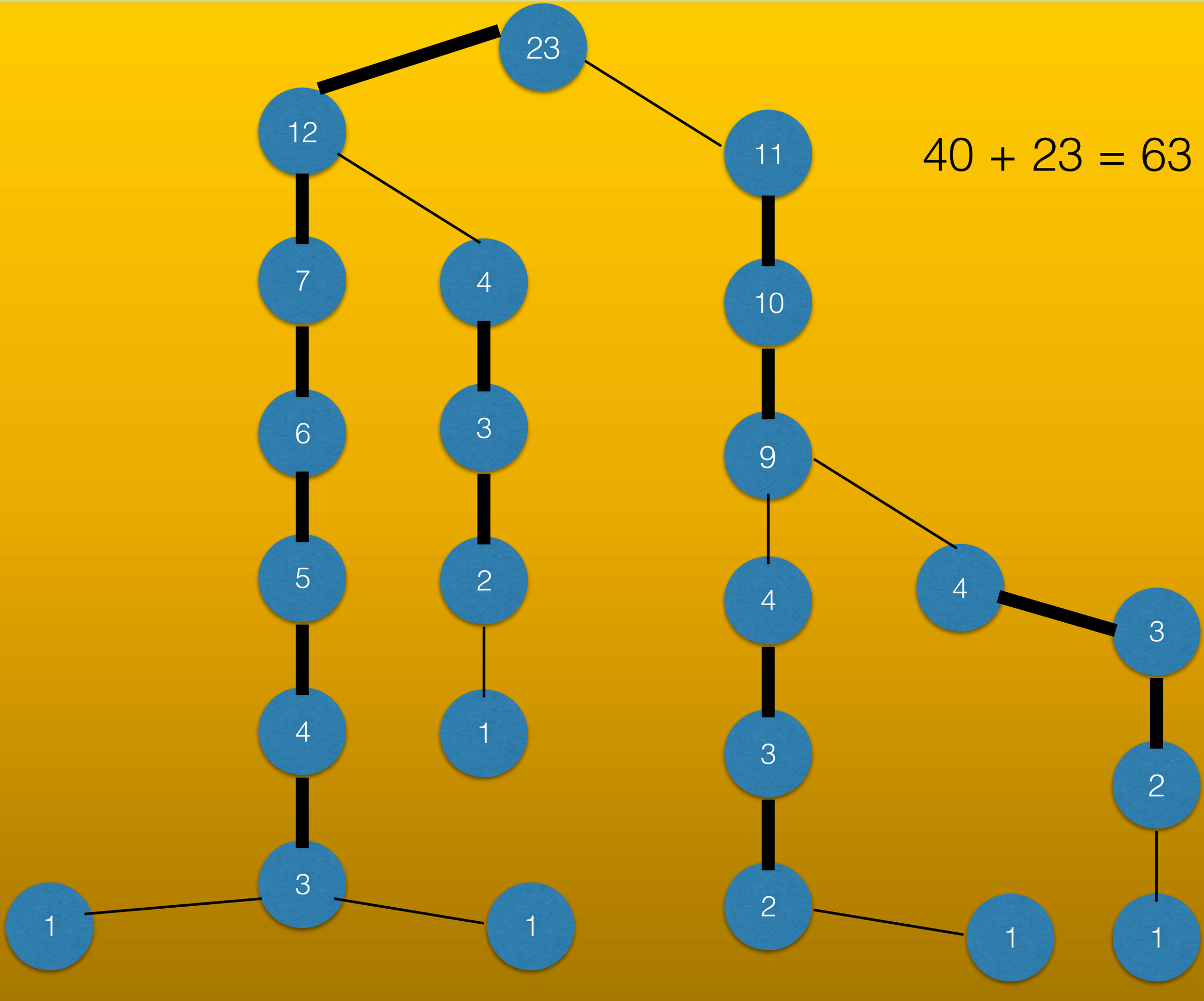

23
12
11
7
4
10
6
3
9
5
2
4
4
1
3
3
2
1
1
2
1
1
0 + 1 = 1



1 + (2+3+4) = 10

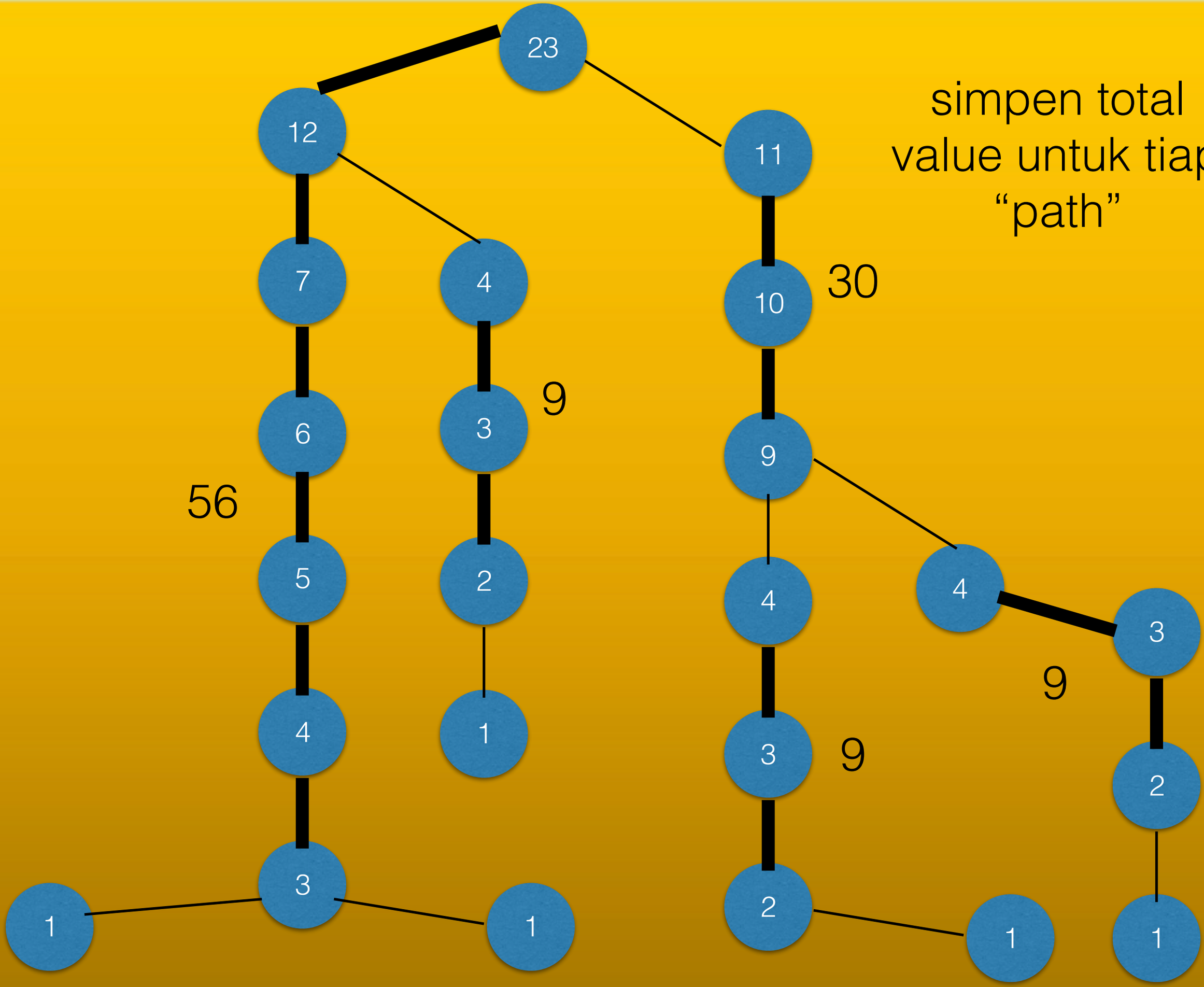


10 + (9+10+11)
= 40



40 + 23 = 63

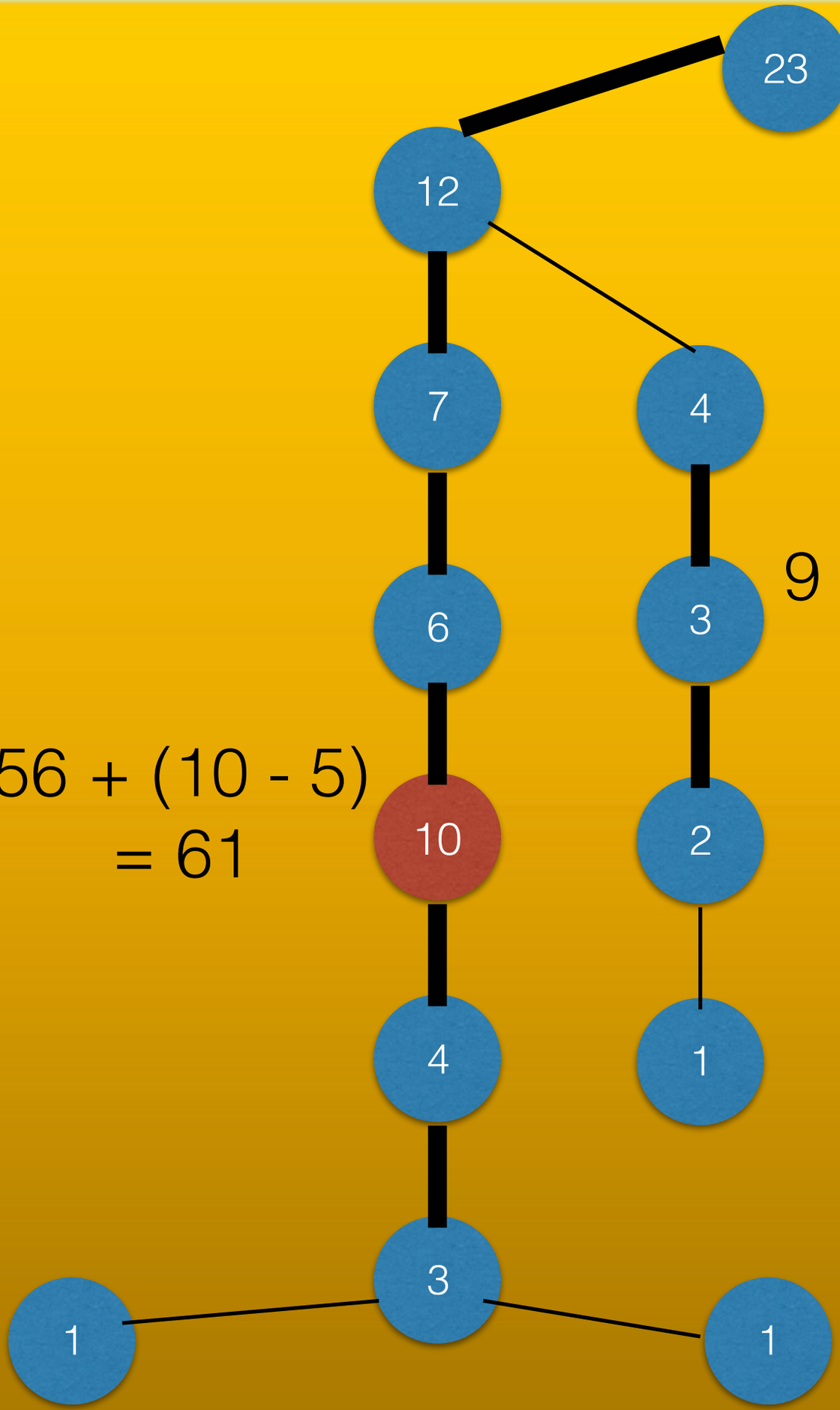
gimana cara tau total value yang di heavy edge yang di skip?



simpen total value untuk tiap “path”

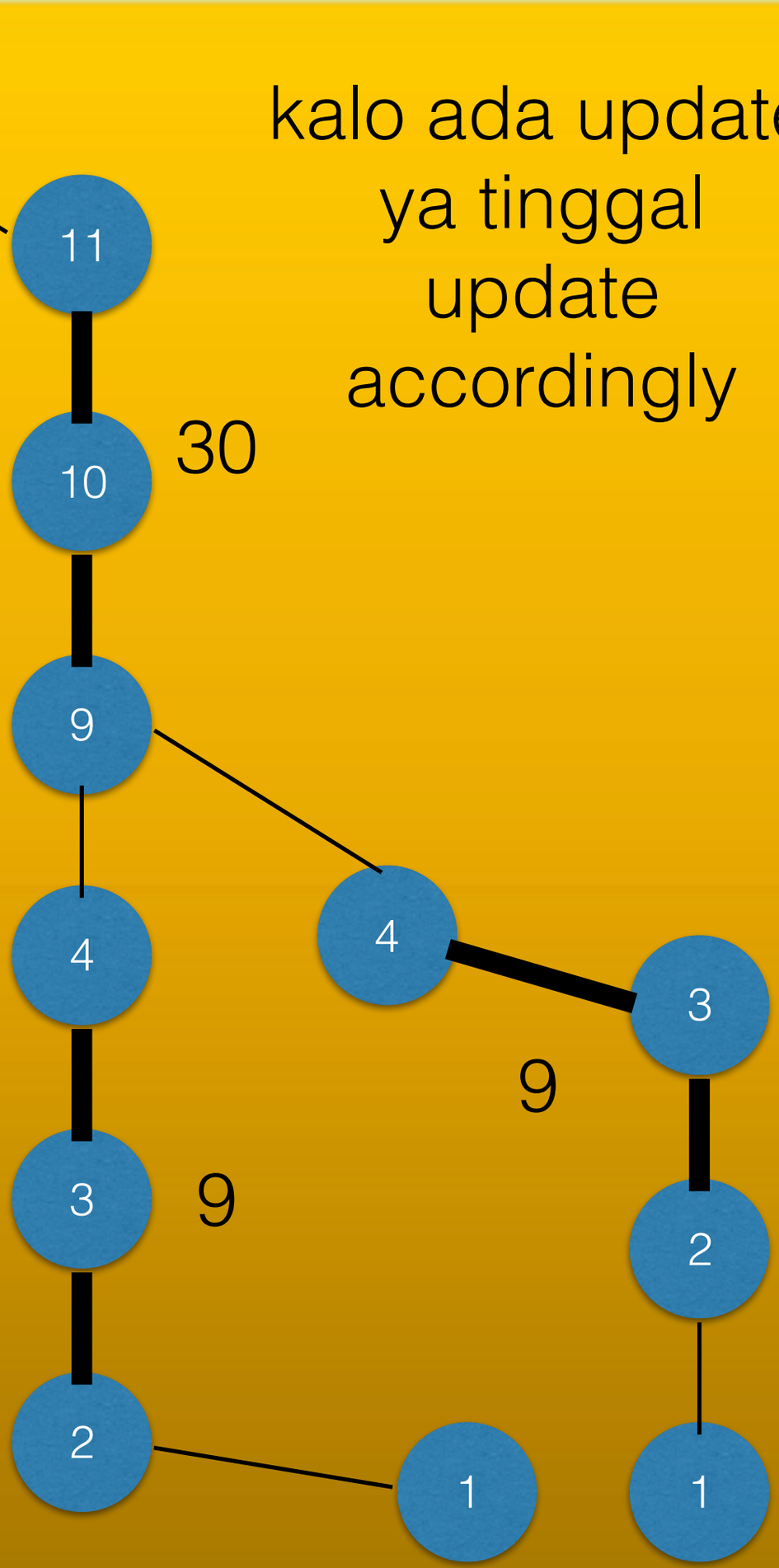
kalo ada update ya tinggal update accordingly

56 + (10 - 5)
= 61



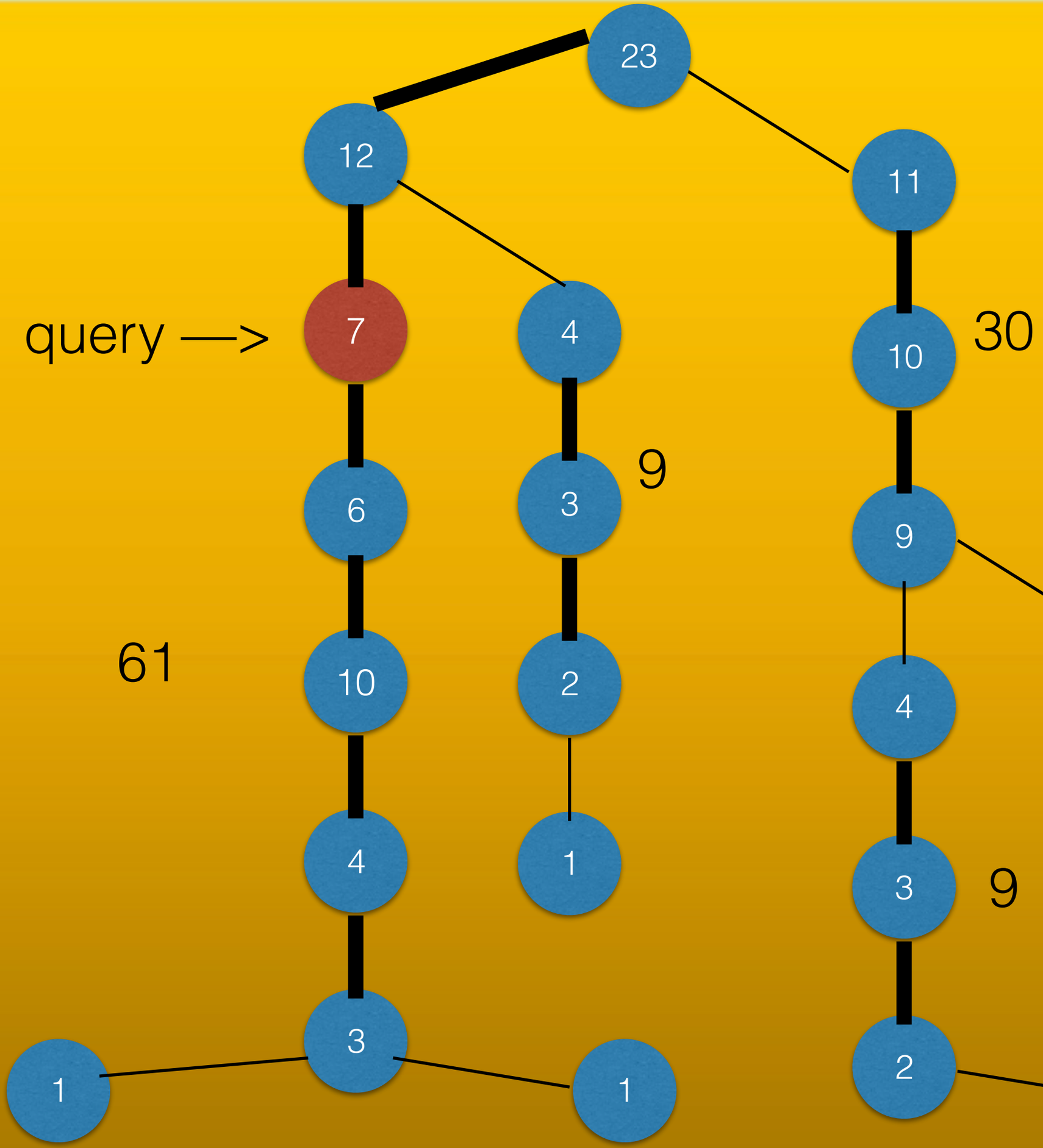
9

30

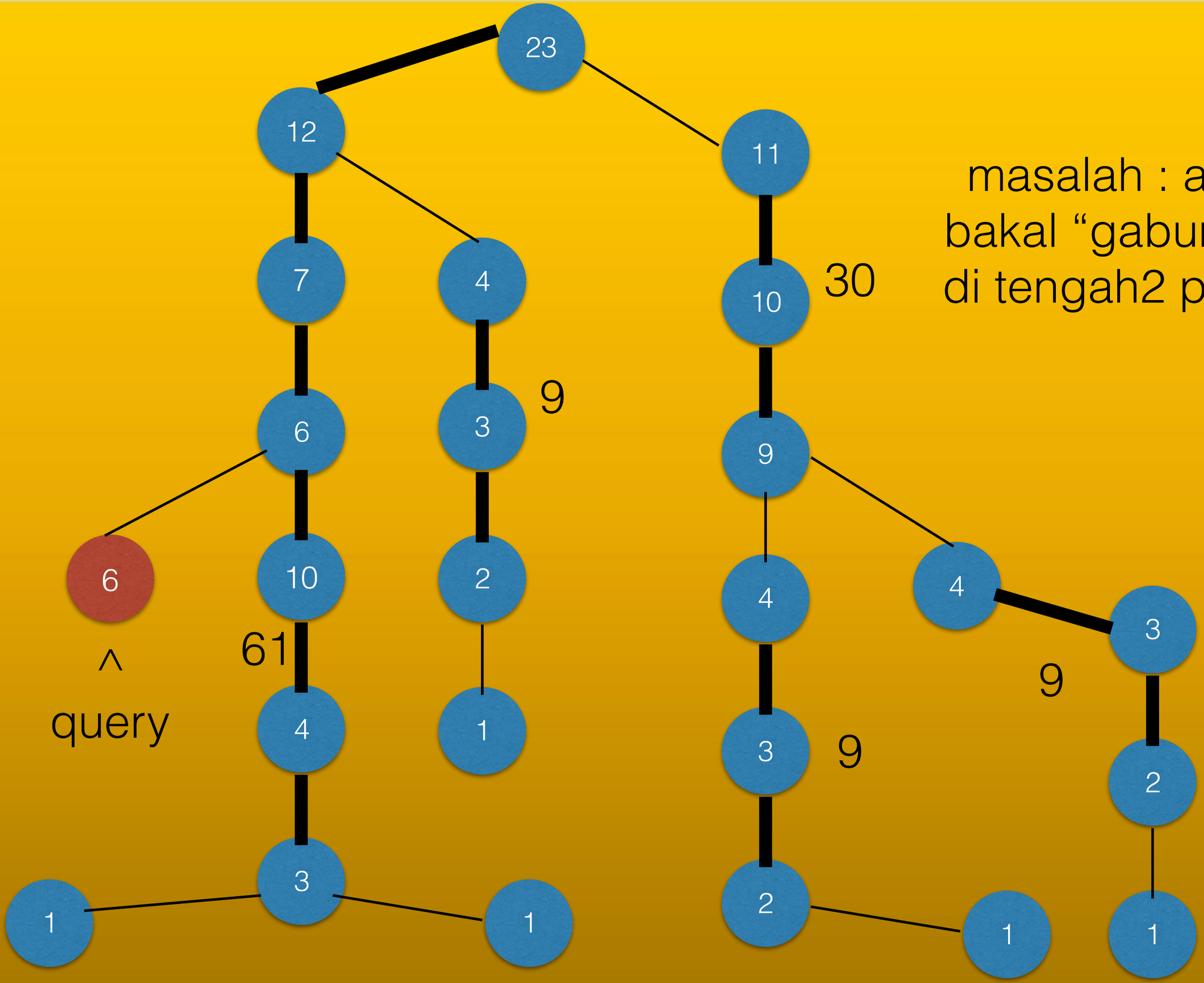


9

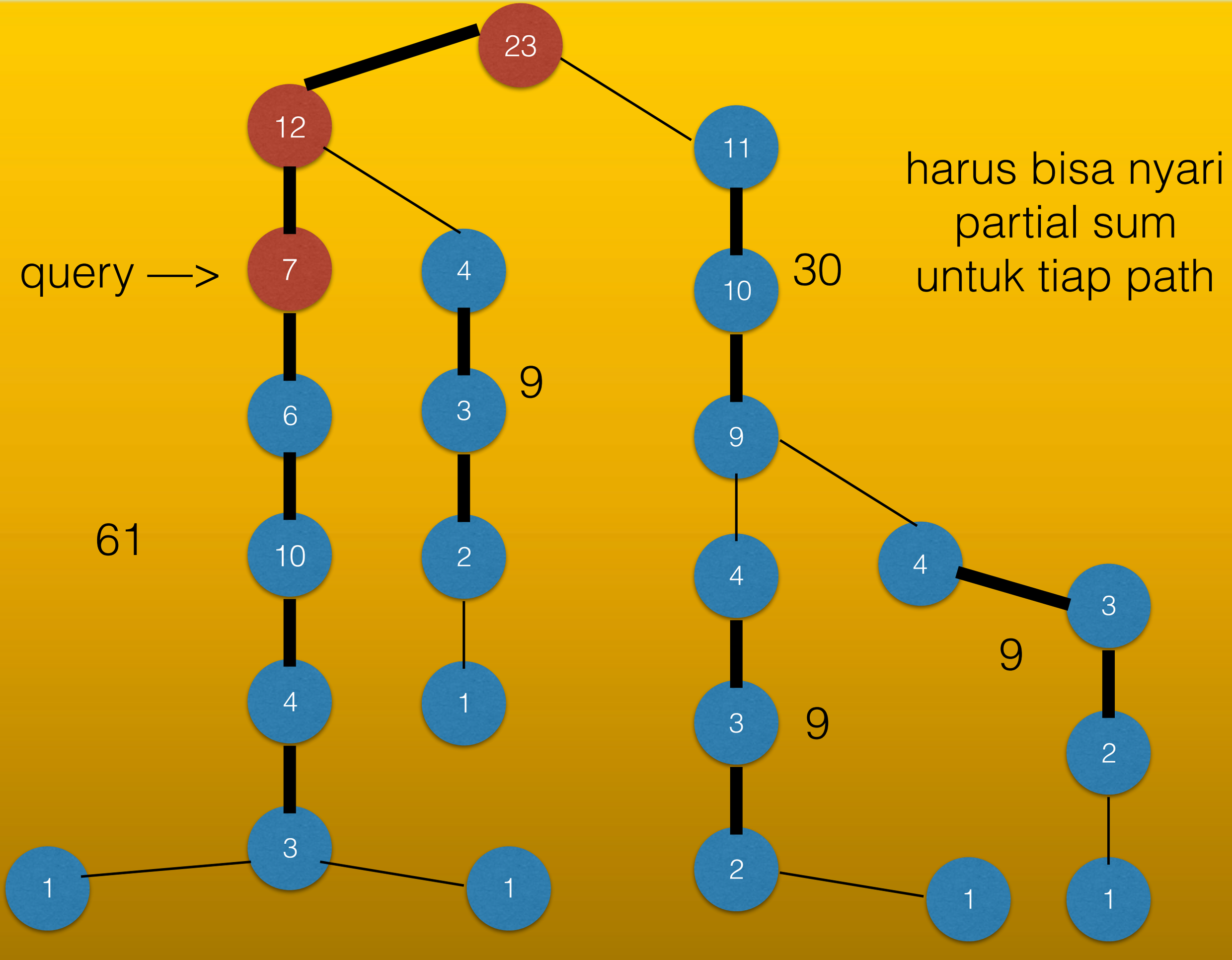
9



masalah :
gimana kalo querynya mulai dari tengah2 path?



masalah : ato bakal “gabung” di tengah2 path



harus bisa nyari
partial sum
untuk tiap path

BIT

masih inget kan apa kepanjangannya?

lu bakal butuh banyak BIT

mending dibikin OOP

```
class BIT {
public: // penting. bahaya parah, tapi yaudahlah ya
  vector<int> v;

  void init(int _N) {
    v.resize(_N);
  }

  void update(int x,int y) {
    for (int i = x; i < v.size(); i += (i & -i)) {
      v[i] += y;
    }
  }

  int query(int x) {
    int ans = 0;
    for (int i = x; i > 0; i -= (i & -i)) {
      ans += v[i];
    }
    return ans;
  }
};
```

kita jadi bisa bikin dua instance of BIT
contoh : cari variance

```
BIT sum, sumsq;

sum.init(N);
sumsq.init(N);
```

```
for (int i = 0; i < N; ++i) {
  sum.update(i, A[i]);
  sumsq.update(i, A[i] * A[i]);
}

// V = E(X^2) - (E(X))^2
V = (sumsq.query(N) / N) - (sum.query(N) / N) ^ 2
```

jadi HLD nya kira2nya gini

```
void dfs(int u) {
  size[u] = 1;
  for (int v : child[u]) {
    dfs(v);
    size[u] += size[v];
  }
}
```

```
void dfs(int u, int componentRoot) {
  componentRoot[u] = componentRoot;
  if (u == componentRoot) {
    bit[u] = new BIT();
  }
  for (int v : child[u]) {
    if (2 * size[v] > size[u]) {
      dfs(v, componentRoot);
    } else {
      dfs(v,v); // light edge, path baru
    }
  }
}
```

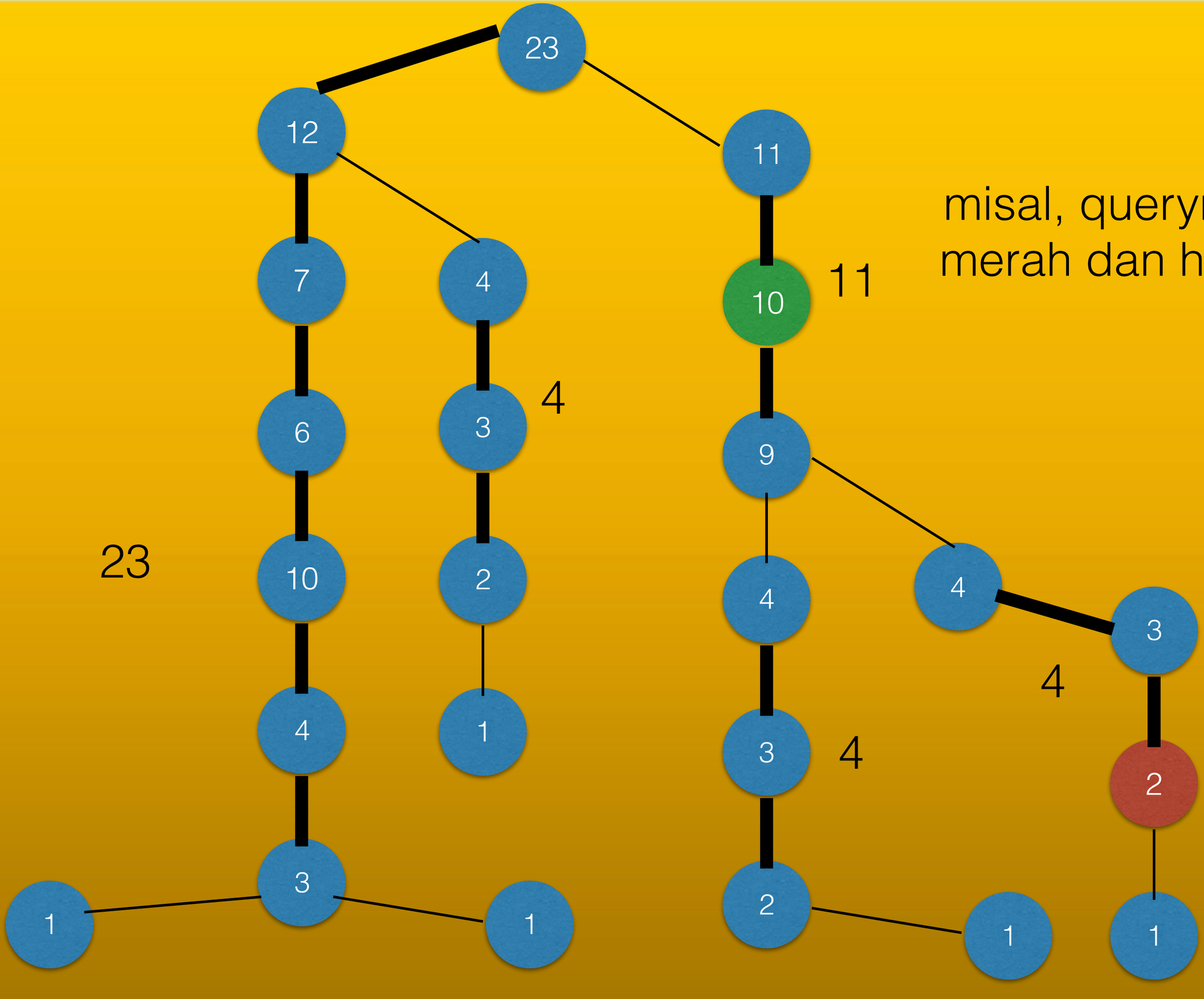
querynya

```
T query(int u) {
  T ans;
  while (u != null) {
    int cRoot = componentRoot[u];
    ans = merge(ans, bit[cRoot].query(cRoot, u));
    u = parent[cRoot];
  }
  return ans;
}
```

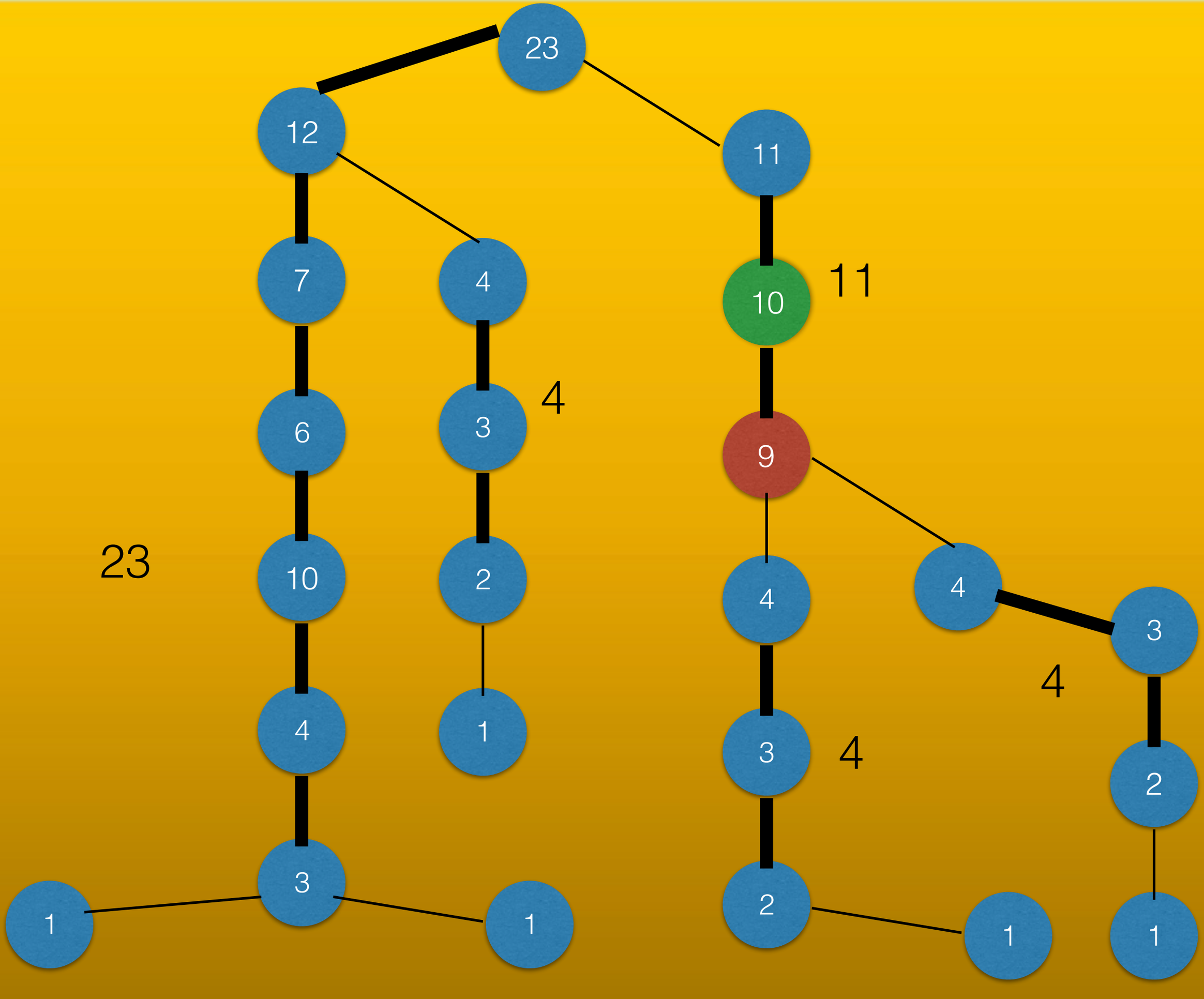
gimana kalo mau cari maks instead of sum?

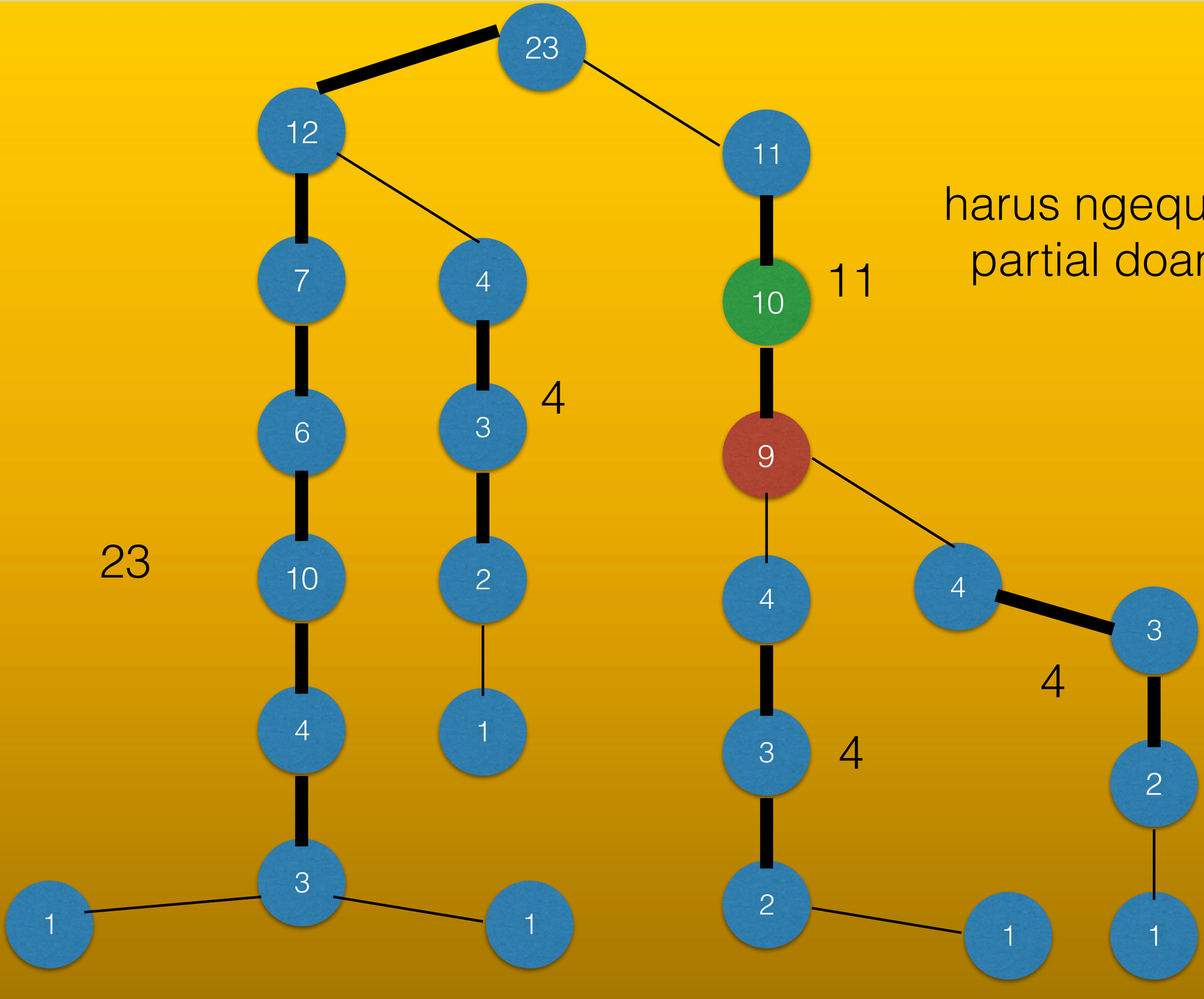
kalo cari maks kan gabisa

maks(u,root) - maks(v,root)



misal, querynya
merah dan hijau


23
12
11
7
4
10
11
6
3
4
9
10
2
4
4
4
3
3
4
2
1
2
1
3
1
1
1
1



harus ngequery
partial doang

```
T query(int u, int v) {
  // assume v itu ancestor u
  T ans;
  while (true) {
    int cRoot = componentRoot[u];
    if (h[cRoot] > h[v]) {
      // cRoot masih descendent v
      ans = merge(ans, bit[cRoot].query(cRoot, u));
      u = parent[cRoot];
    } else {
      ans = merge(ans, bit[cRoot].query(v, u));
      break;
    }
  }
  return ans;
}
```

gimana kalo updatenya bisa path?

segment tree with lazy update

semangat ya ngodingnya :)

nah coba latihan

contoh soal

SPOJ QTREE 3

dikasih tree N node, tiap node bisa antara putih ato hitam

bisa ada dua jenis query :

1. ganti warna sebuah node

2. dari path u->v, manakah node putih pertama yang dilalui

gimana?

gak terlalu susah, gausah pake jam lah ya

EOF

Q&A?