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

# DP Tree

Jonathan Irvin Gunawan
National University of Singapore

prerequisite

# tau DP

# tau tree

oke mulai dari yang gampang yah

# DP biasa tapi di tree

biasa = gak pake teknik khusus

# maximum independent set

(sama kayak yang kemaren yah)

on a tree

dikasih sebuah graph. tiap vertex ada weightnya. pilih beberapa vertex sedemikian sehingga total weightnya maksimum dan tidak ada vertex yang adjacent

kan kemaren max independent set O(2^N)

kalo di tree bisa
O(N) doang

state :

1. sekarang mau cobain node yang mana

2. parentnya dipilih ato kagak

base case :
kalo udah di leaf

transition :
cobain whether ambil node sekarang atau kagak. kalau node parent diambil, gak boleh ambil node sekarang

# dp(now,parent)

```
if (isLeaf(now)) {
  return max(0, parent == 0 ? value[now] : 0);
}
int &ret = dp[now][parent];
if (ret >= 0) return ret;

// not taking node now
int case1 = 0;
for (int x : child[now]) {
  case1 += dp(x, 0);
}
int case2 = 0;
if (parent == 0) {
  // taking node now, cuma bisa kalo parent gak taken
  case2 = value[now];
  for (int x : child[now]) {
    case2 += dp(x, 1);
  }
}
return ret = max(case1,case2)
```

# min vertex cover

# on a tree

dikasih sebuah graph. tiap vertex ada weightnya. pilih beberapa vertex sedemikian sehingga total weightnya minimum dan setiap edge at least endpointnya ada satu yang dipilih

statenya (sama transisinya)

sama kan ya

# dp(now,parent)

```
if (isLeaf(now)) {
   return min(weight[now], parent==1?0:weight[now]);
}
int &ret = dp[now][parent];
if (ret >= 0) return ret;

// taking node now
int case1 = weight[now];
for (int x : child[now]) {
  case1 += dp(x, 1);
}
int case2 = 0;
if (parent == 1) {
  //not taking node now, cuma bisa kalo parent taken
  for (int x : child[now]) {
   case2 += dp(x, 0);
 }
}
return ret = min(case1,case2)
```

coba sekarang yang
gak klasik

dikasih binary tree, di beberapa node ada nomernya (0 ≤ semua nomer ≤ 100). itung banyaknya cara buat lu nomerin node2 lainnya (≤ 100 juga)

s.t. untuk setiap internal node,
nomernya = sum dari nomer2 childnya.
nomer dari leaf bebas


6:00

state :

1. sekarang node mana
2. node ini harus dinomerin berapa

kalo node sekarang ada nomernya
dan nomernya beda sama yang ada
di state, return 0, otherwise

base case :
kalo leaf, return 1

rekurens :
coba bagi number sekarang ke dua child

banyaknya kemungkinan =
banyaknya kemungkinan kiri *
banyaknya kemungkinan kanan

karena dia independen
#multiplication-rule-ftw

# dp(now, number)

```
if (isleaf(now)) return 1
// if (SIZE(adj[now]) == 1) return 1; <— bener ga?

int &ret = dp[now][number];
if (ret >= 0) return ret;
ret = 0;
for (int i = 0; i <= number; ++i) {
 ret += dp(left(now),i) * dp(right(now),number-i)
}
return ret;
```

coba satu lagi yah sebelum kita masuk ke DP tree yang beneran

dikasih binary tree, tiap node ada weightnya. lu mau ambil K node sedemikian sehingga total weightnya maksimum

gw tau bisa di greedy tapi di DP aja yuk

statenya mirip kan :

1. sekarang node mana

2. boleh ngambil berapa angka lagi

terus lu cobain mau bagi berapa ke kiri sama berapa ke kanan

# dp(now, rem)

```
if (isleaf(now)) return (rem >= 1 ? w[now] : 0);

int &ret = dp[now][rem];
if (ret >= 0) return ret;
ret = 0;

for (int i = 0; i <= number; ++i) {
   ret = max(ret,dp(left(now),i)+dp(right(now),number-i))
}
for (int i = 0; i < number; ++i) {
   // kalo ambil node yang sekarang
   ret = max(ret,dp(left(now),i)+dp(right(now),number-i-1)+w[now]);
}
return ret;
```

state : O(NK)

transisi : O(K)

O(NK^2) kan ya

nah soal2 dari tadi

itu “gampang”

bisa disolve pake teknik2 DP standar, ga butuh teknik spesial DP yang bisa dipake on tree

sekarang soal yang barusan diganti

dikasih ~~binary~~ tree, tiap node ada weightnya. lu mau ambil K node sedemikian sehingga total weightnya maksimum

mampus loh kali ini gak dijamin binary

masalahnya

dimana?

terus lu cobain mau bagi
berapa ke kiri sama
berapa ke kanan

# ini jadi ga jalan kan?

#slide-dalam-slide #slide-ception

bisa coba2 untuk semua konfigurasi pembagian child

ntar transisi instead of O(K),
jadi O(K^(M-1)) dimana M itu banyaknya child

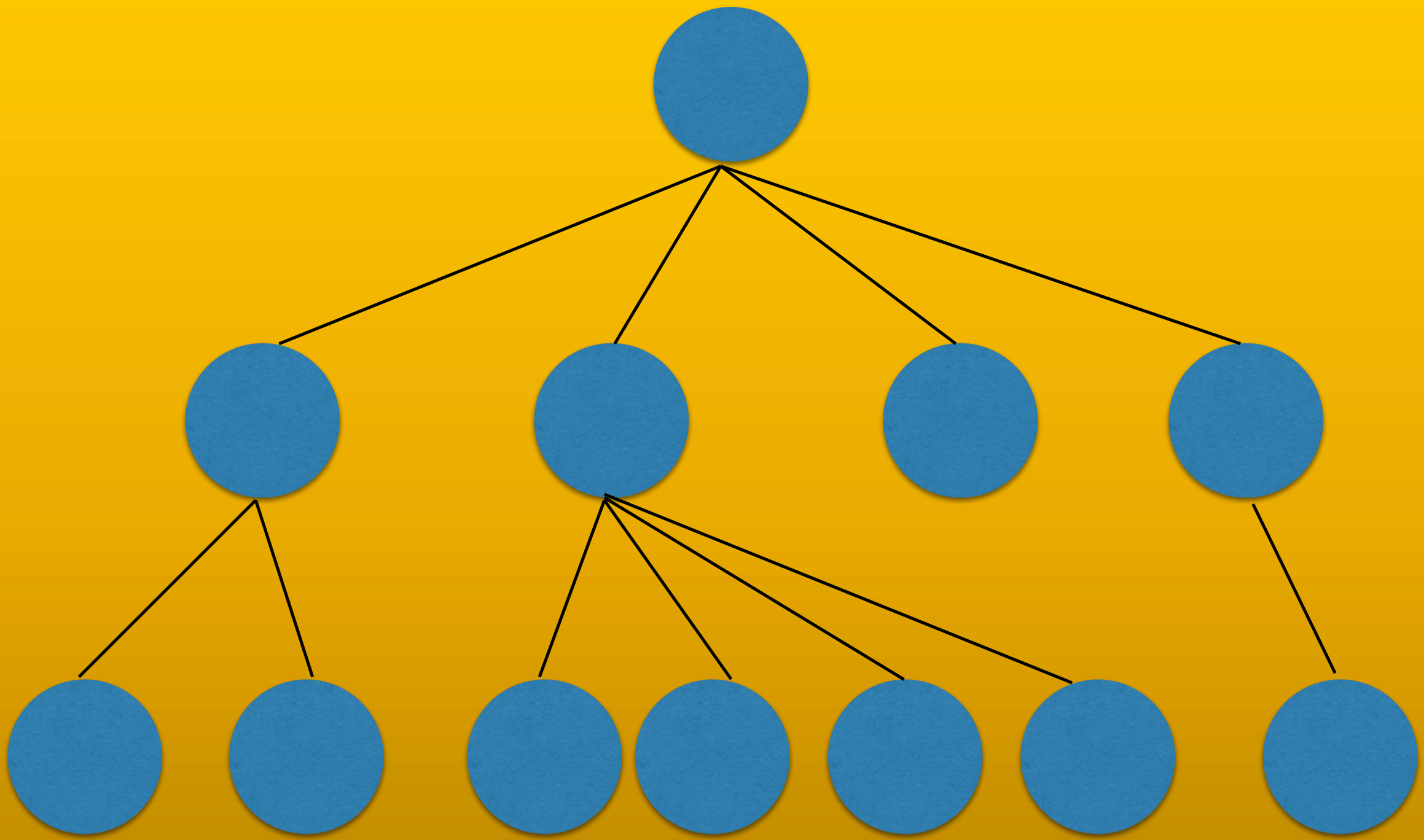
# EXPONENTIAL ALGORITHM

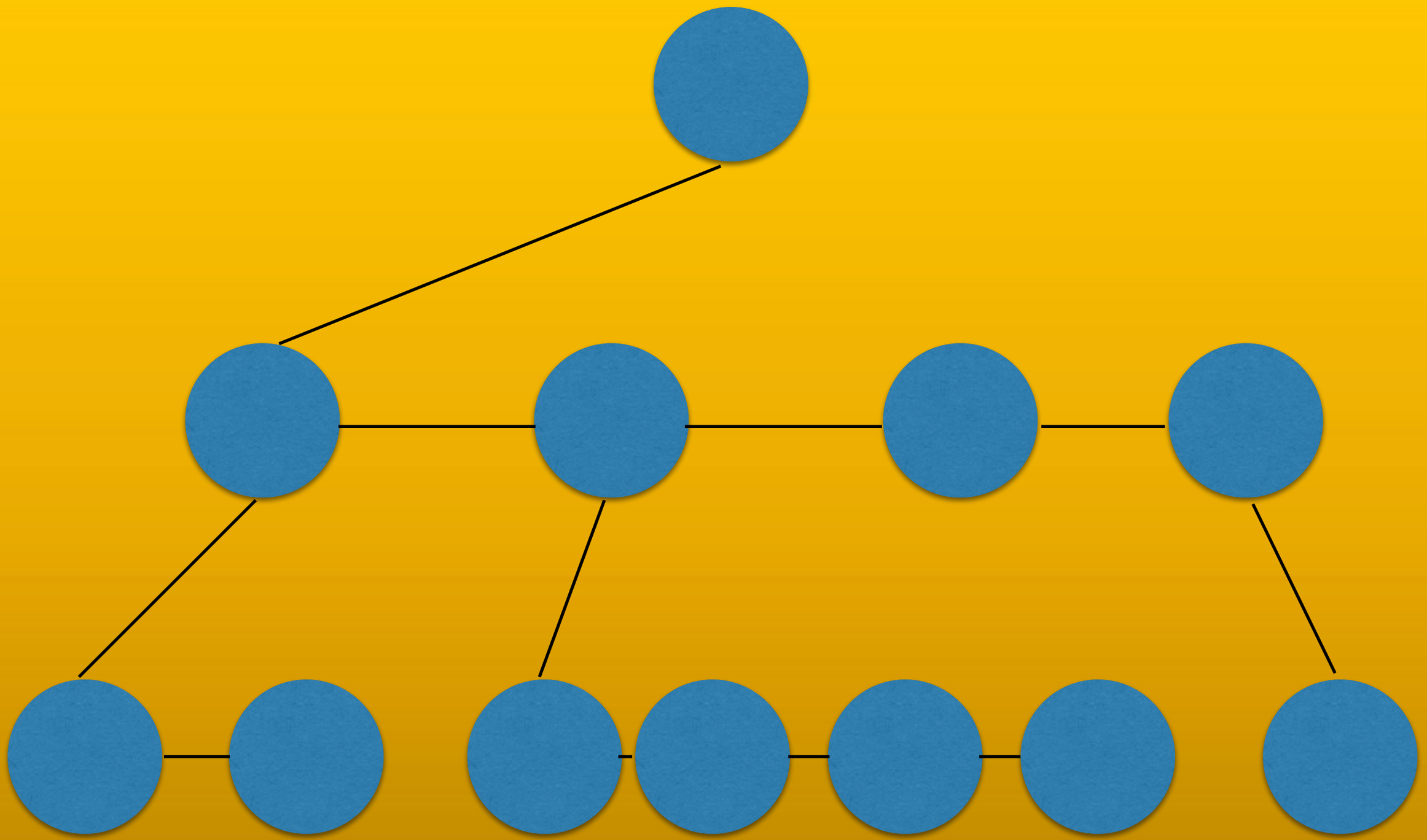
# WE HATE EXPONENTIAL ALGORITHM

# trik 1 : parent sibling

convert treenya supaya tiap node cuma punya satu child. sisa child2nya bakal jadi sibling dari satu child tersebut

pasti ngga ngerti



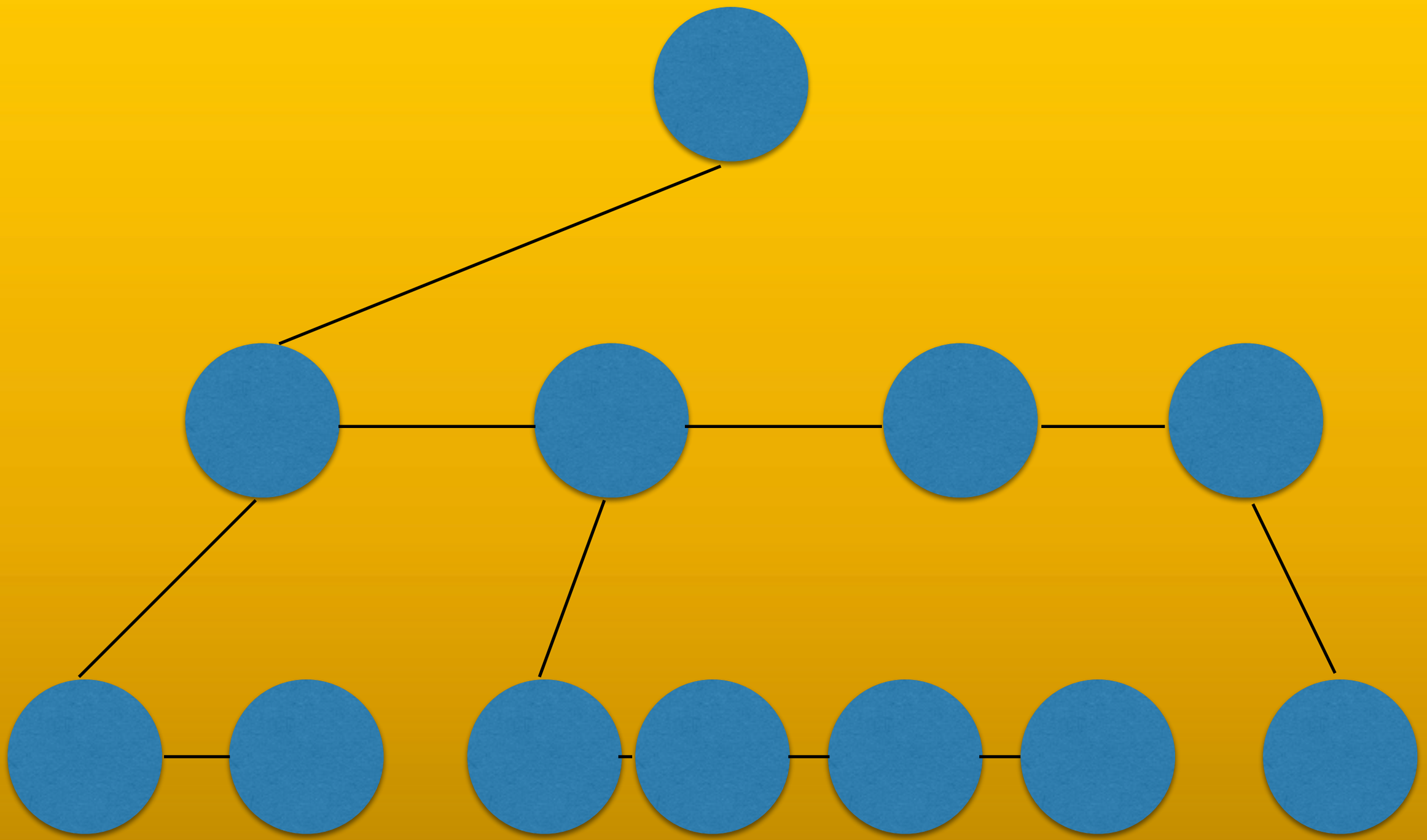


notice some

property here?

tiap node jadi cuman punya

dua neighbour selain parent



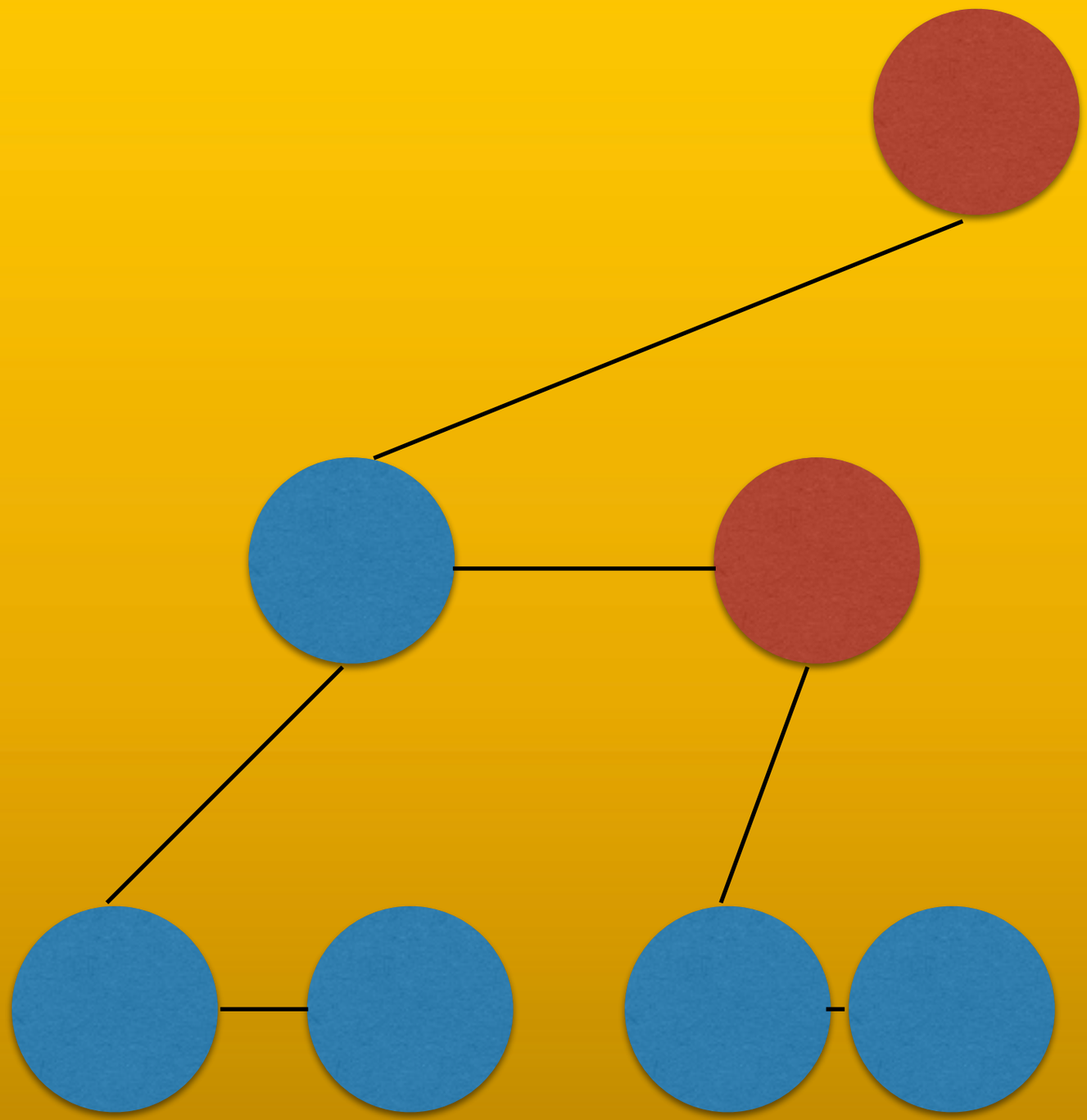
nah, buat soal yang tadi, structure treenya gak ngaruh kan

jadi simply ganti treenya jadi parent sibling, terus run DP tree yang persis sama

tapi gimana kalo struktur treenya ngaruh?

let’s say sekarang soalnya **lu ga boleh ngambil dua node yang adjacent**

since strukturnya ngaruh,
kalo lu ganti struktur treenya
jawabannya bakal beda



di parent sibling
dua node merah
ini boleh kita
ambil karena ga
adjacent

tapi di soal
aslinya tadi kan
mereka adjacent

jadi sekarang state DPnya tambah satu lagi :

1. sekarang dimana
2. sisa node yang harus diambil
3. **parent node diambil ato kagak**

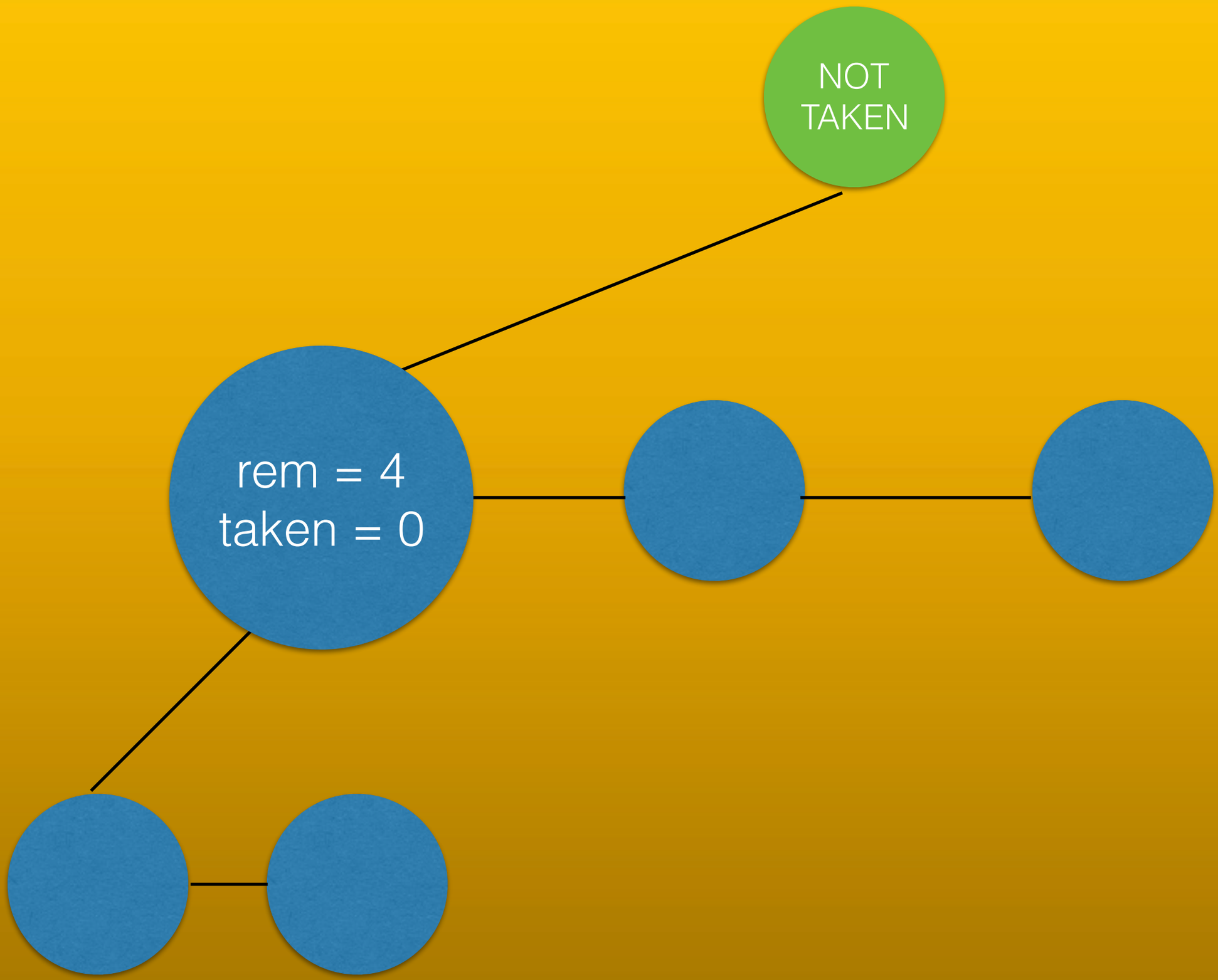
nah parameter yang ini :
**parent node diambil ato kagak**
itu kalo transisi ke kanan (sibling),
dia gak boleh berubah, no matters lu
ambil node sekarang ato kagak

karena

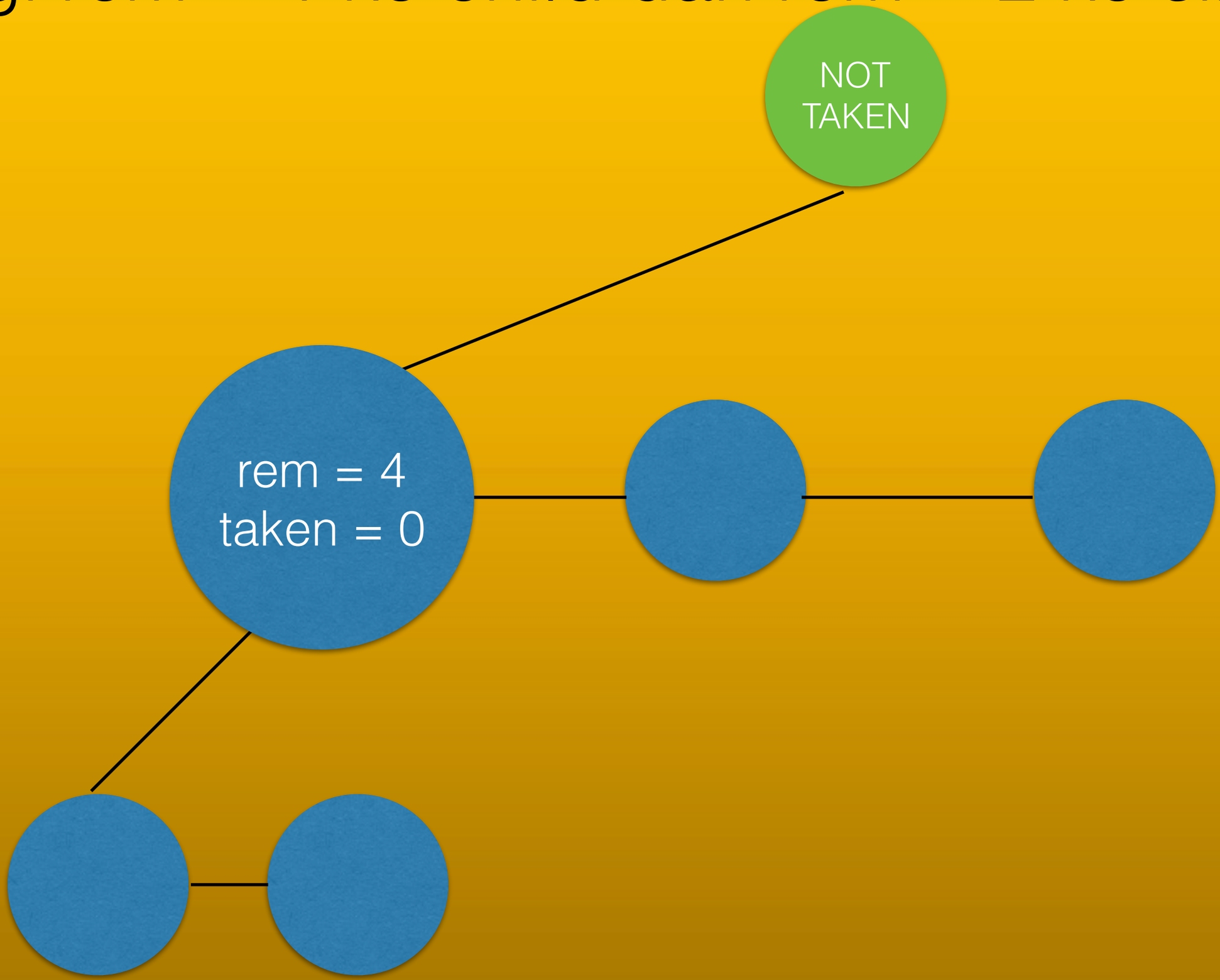
parent(sibling(now)) == parent(now)

kan

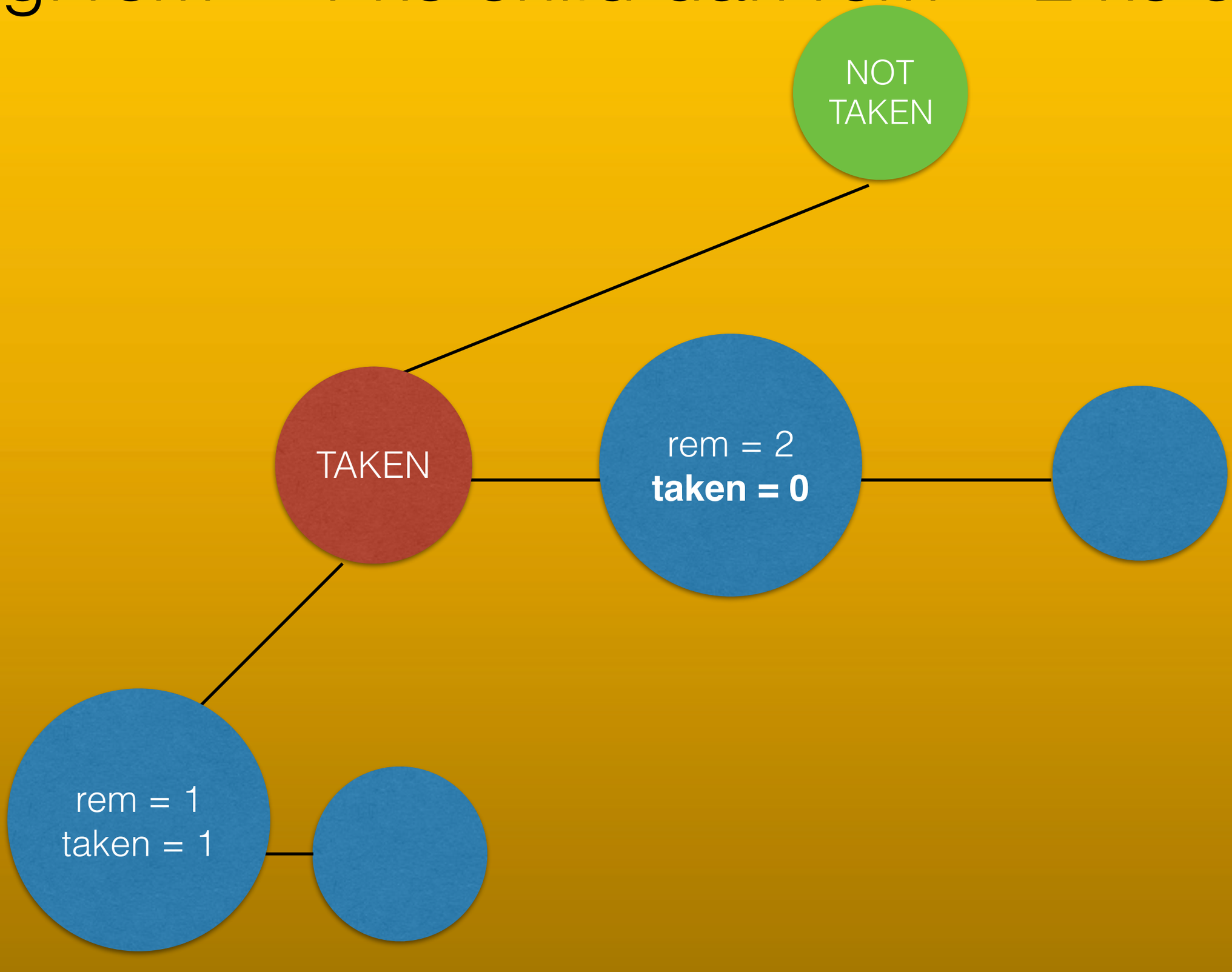
# contoh



let’s say kita ambil node sekarang (yang gede)
bagi rem = 1 ke child dan rem = 2 ke sibling



let’s say kita ambil node sekarang
bagi rem = 1 ke child dan rem = 2 ke sibling



# dp(now, rem, parent)

```
if (isleaf(now)) return (rem >= 1 && parent == 0 ? w[now] : 0);

int &ret = dp[now][rem][parent];
if (ret >= 0) return ret;
ret = 0;
```

```
for (int i = 0; i <= number; ++i) {
   ret = max(ret,dp(child(now),i,0)+dp(sibling(now),number-i,parent))
}
if (parent[now] == 1) return ret; // gak boleh ambil node sekarang
for (int i = 0; i < number; ++i) {
  // kalo ambil node yang sekarang
  ret = max(ret,
        dp(child(now),i,1)+dp(sibling(now),number-i-1,parent)+w[now]);
}
return ret;
```

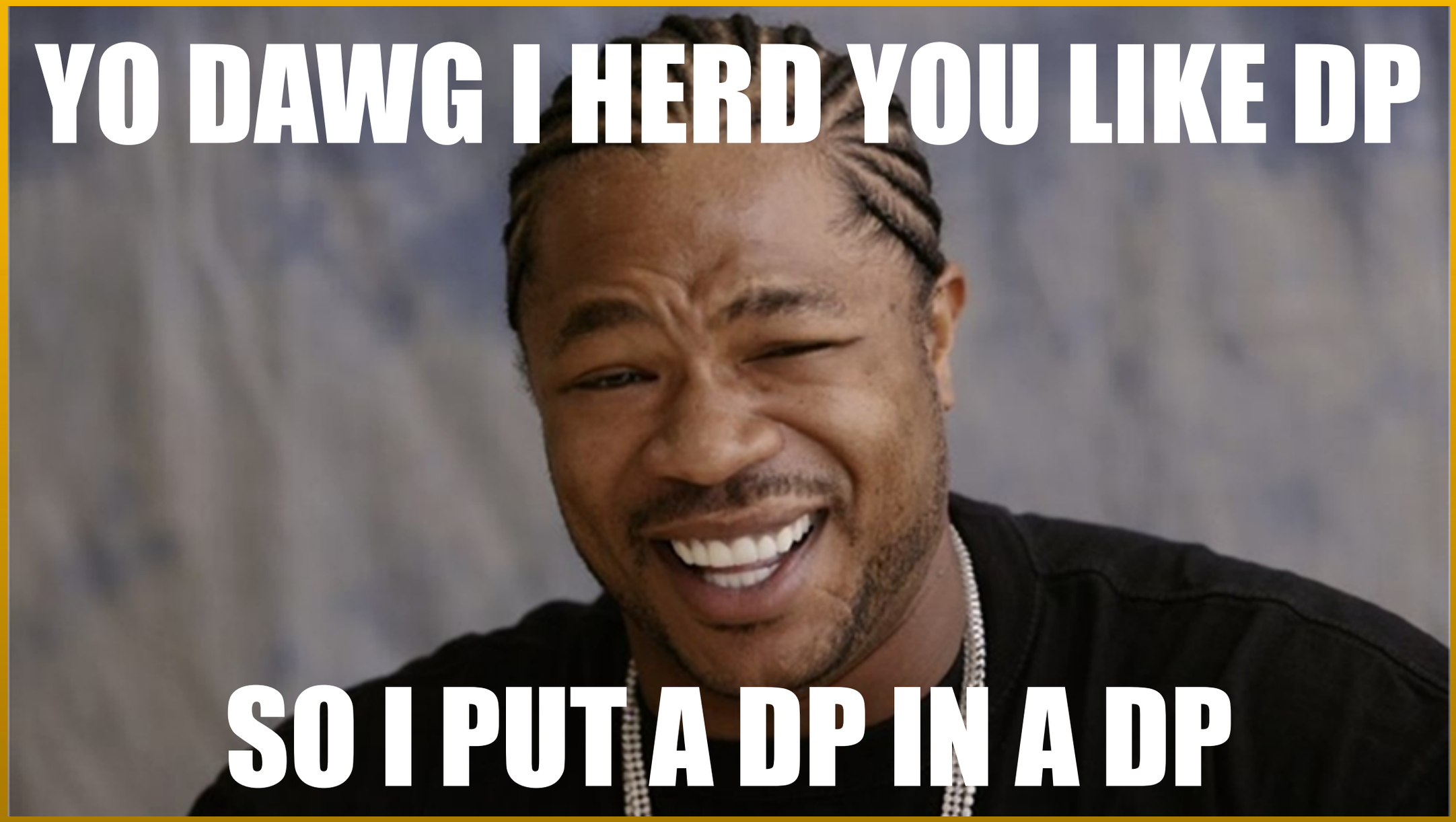
jadi kompleksitasnya balik ke kayak kalo soalnya binary tree yah
O(NK^2)

cara convert jadi parent sibling tree nya bisa lah ya

# dfs(now, parent)

```
child[now] = -1;
sibling[now] = -1;
int last_child = -1;
for (int x : adj[now]) {
  if (x == parent) {
    continue;
  }
  if (child[now] == -1) {
    child[now] = x;
  }
  if (last_child != -1) {
    sibling[last_child] = x;
  }
  dfs(x, now);
  last_child = x;
}
```

# trik 2 : DP knapsack buat transisinya



misal ada soal kayak gini :
ada N orang dan K dolar. Orang ke-i bakal menghasilkan tingkat produktivitas Pi,j kalo dikasih j-dolar.
Tentuin tingkat produktivitas maksimum

DP kan?

state :

1. sekarang lagi coba ngasih uang ke orang yang mana
2. sisa uang lu

# dp2(now,rem)

```
if (now == N) return (rem == 0 ? 0 : -INF);
int &ret = dp2[now][rem];
if (ret >= 0) return ret;

ret = 0;
for (int i = 0; i <= rem; ++i) {
  ret = max(ret, dp2(now + 1, rem - i) + P[now][i]);
}
return ret;
```

O(NK^2) kan ya?

jadi untuk tiap state dp(now, rem, parent), kita butuh O(MK^2) dimana M=adj(now)

totalnya jadi

O(NK^3)

bisa jadi O(NK^2) kalo manfaatin dp2 table dari dp(now,rem-1,parent)

detailnya pake papan tulis aja deh

# EOF

Q&A?