Skiplistor
Weiss, avsnitt 10.4.2

Peter Ljunglöf

DAT036, Datastrukturer

30 nov 2012

Länkade listor med flera nivåer

Med en länkad lista måste vi gå igenom N noder i värsta fall:

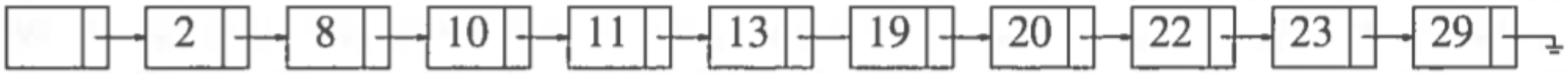


Figure 10.56 Simple linked list

I en lista med en extra nivå räcker det med (N/2 + 1) noder:

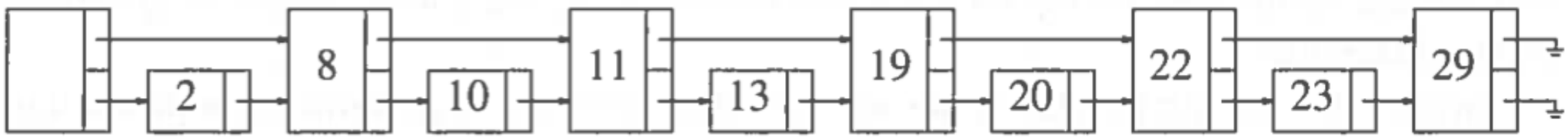


Figure 10.57 Linked list with links to two cells ahead

I en lista med två extra nivåer räcker det med (N/4 + 2):

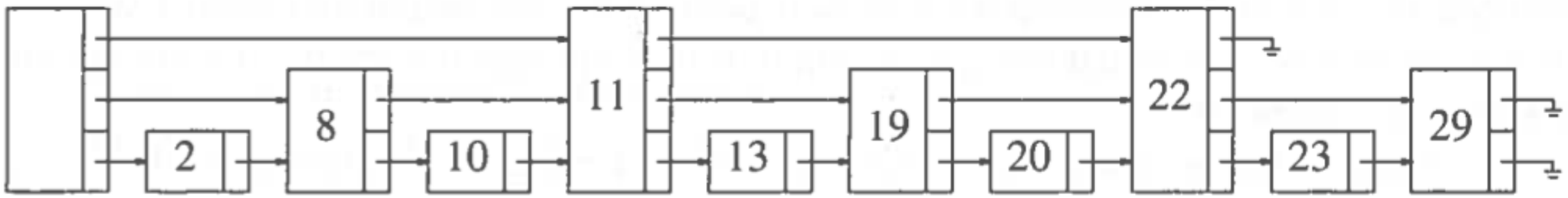


Figure 10.58 Linked list with links to four cells ahead

Länkade listor med flera nivåer

Om vi antar att listan har $k = \log_2(N)$ extra nivåer, hur många noder måste vi gå igenom i värsta fall?

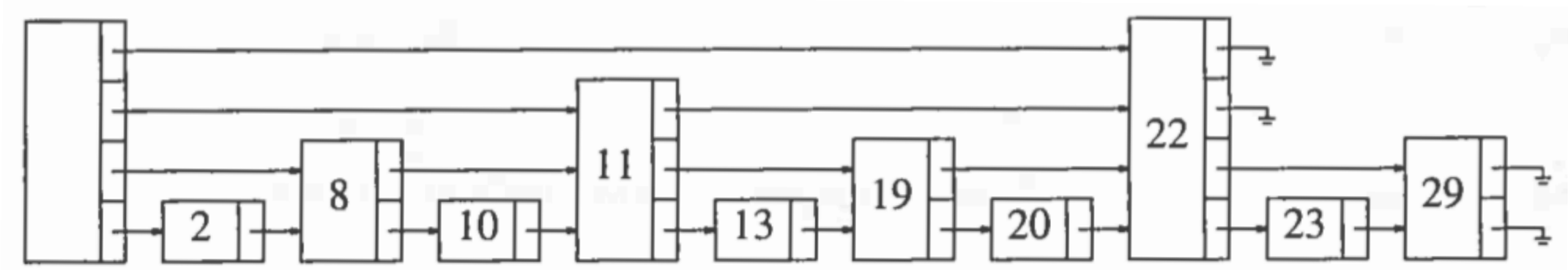


Figure 10.59 Linked list with links to 2^i cells ahead

($k = 3$, i detta fall)

Länkade listor med flera nivåer

Om vi antar att listan har $k = \log_2(N)$ extra nivåer, hur många noder måste vi gå igenom i värsta fall?

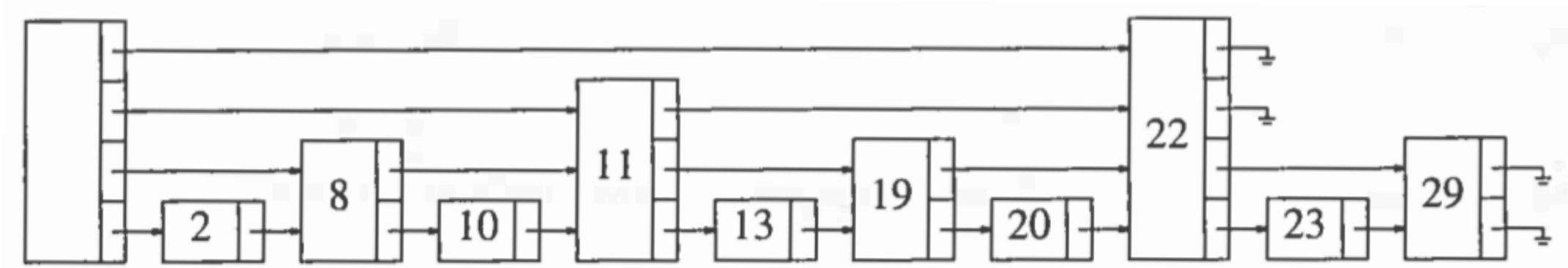


Figure 10.59 Linked list with links to 2^i cells ahead ($k = 3$, i detta fall)

Jo, i värsta fall söker vi igenom $N/2^k + k$ noder

- men eftersom $2^k = 2^{\log(N)} = N$
- så blir värsta fallet $N/N + \log_2(N) = O(\log N)$

Insättning i flernivålistor

Alltså, i en flernivålista är varannan nod nivå 0, var fjärde nivå 1, var åttonde nivå 2, etc.

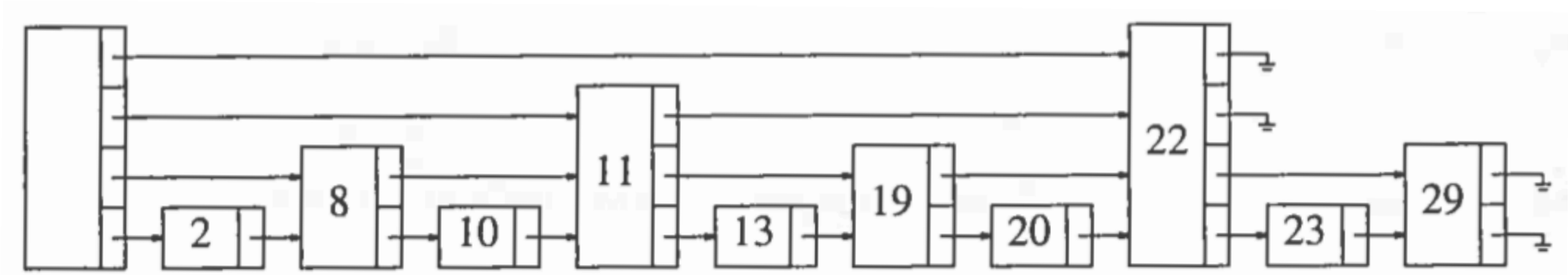


Figure 10.59 Linked list with links to 2^i cells ahead

Men, vilket nivå ska då ett nytt element ha?

- t.ex. hur stoppar vi in 15 i listan ovan?

Svar: Vi gör nivåerna slumpmässiga!

Skiplistor

En skiplista är alltså en flernivålista

- Nivå 0 innehåller alla noder
- Nivå k innehåller (ungefär) $1/2^k$ av noderna

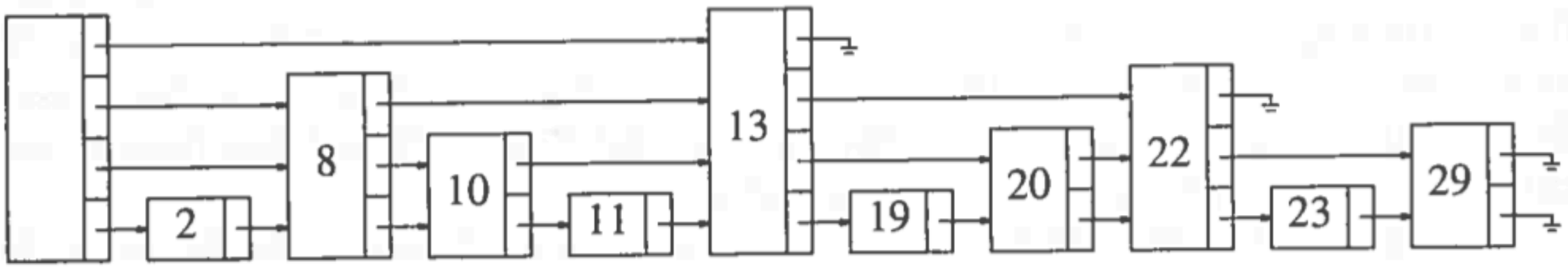


Figure 10.60 A skip list

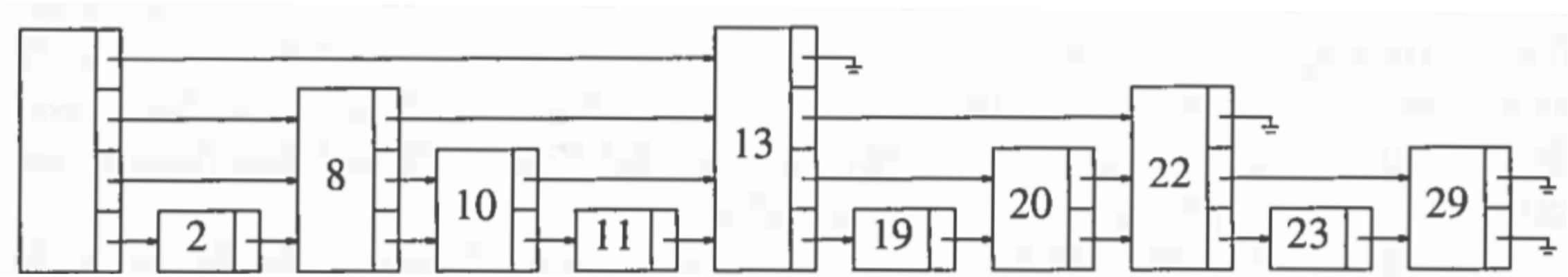
Varje nod i en skiplista har en nivå

- nivån bestäms slumpmässigt: 50% chans för nivå 0, 25% för nivå 1, 12,5% för nivå 2, etc.
- skiplistans nivå är den högsta nivån i listan

Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

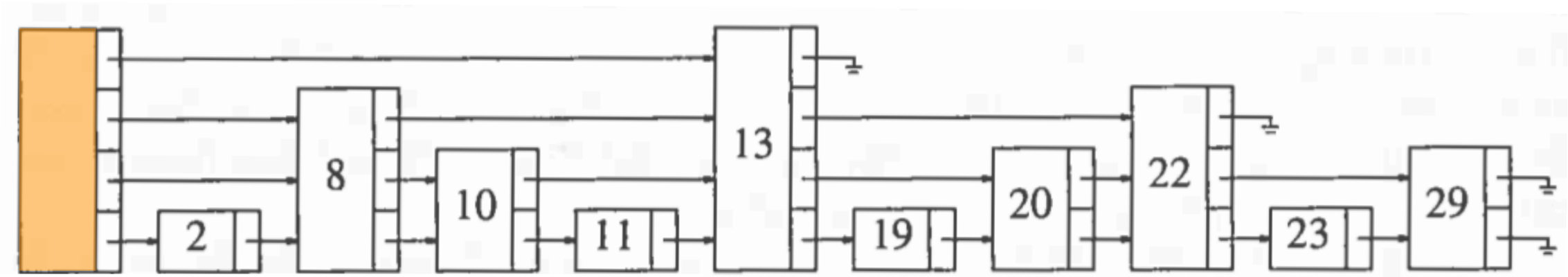
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

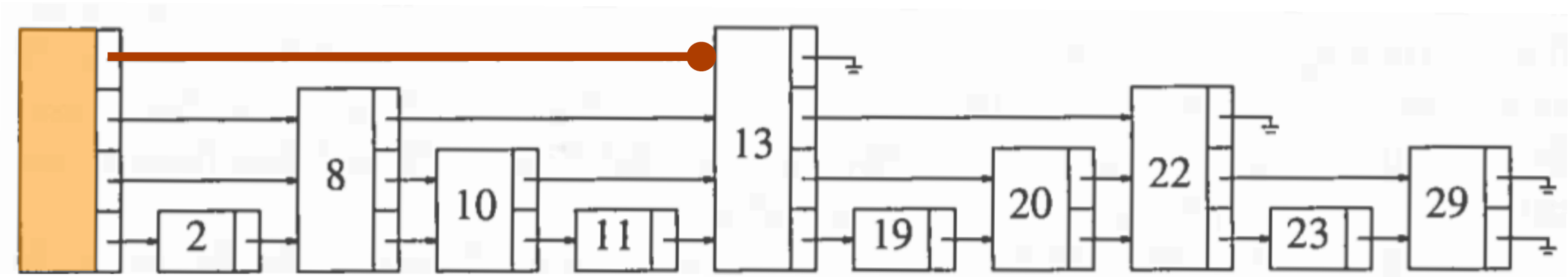
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

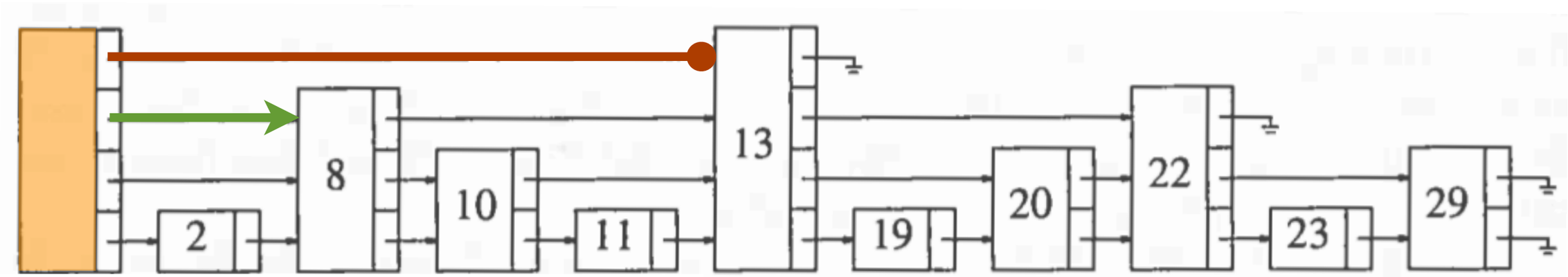
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

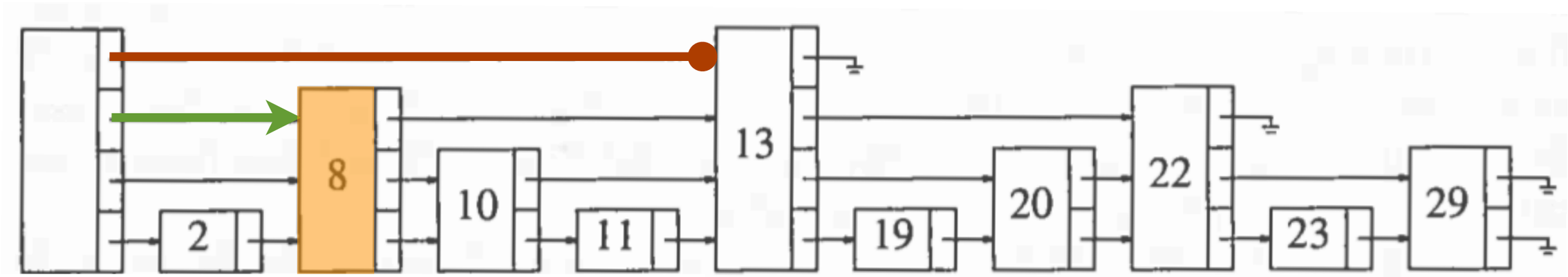
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

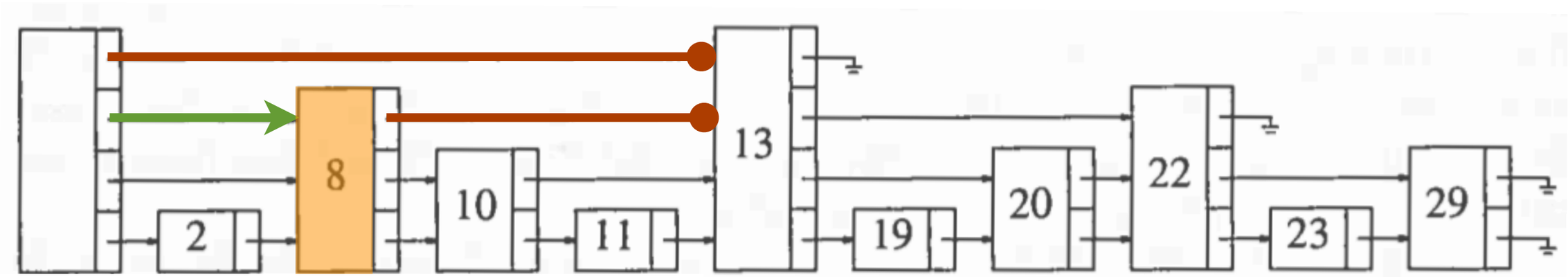
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

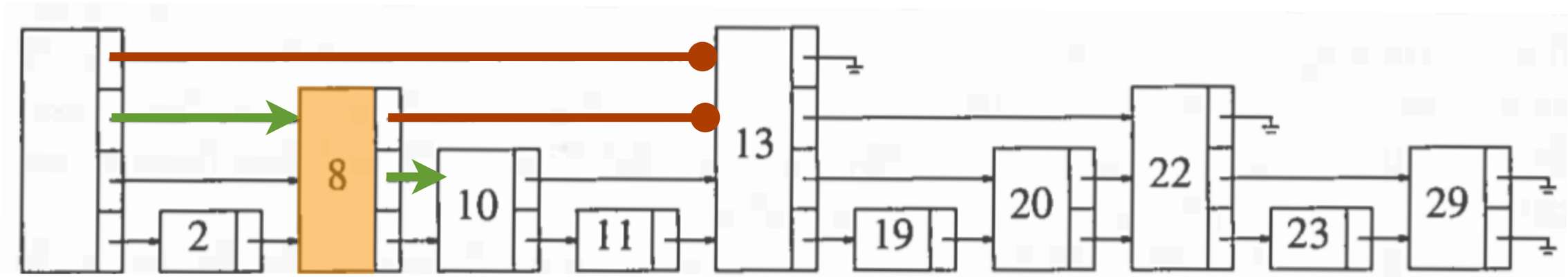
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

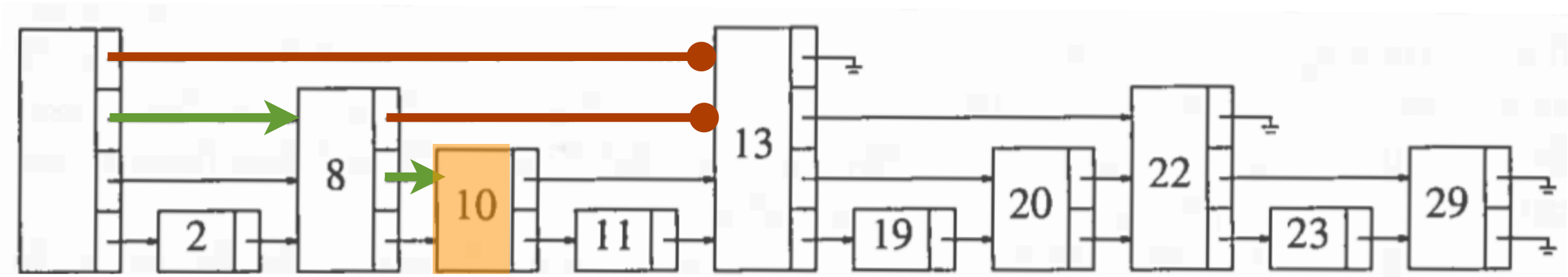
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

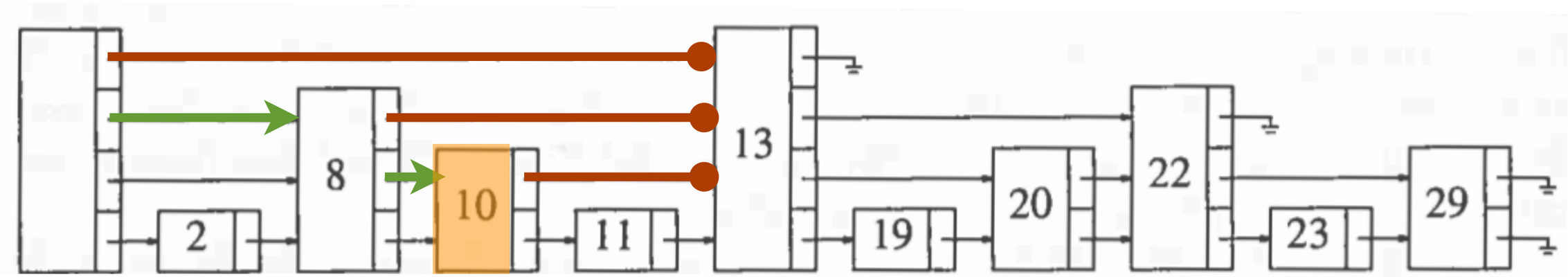
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

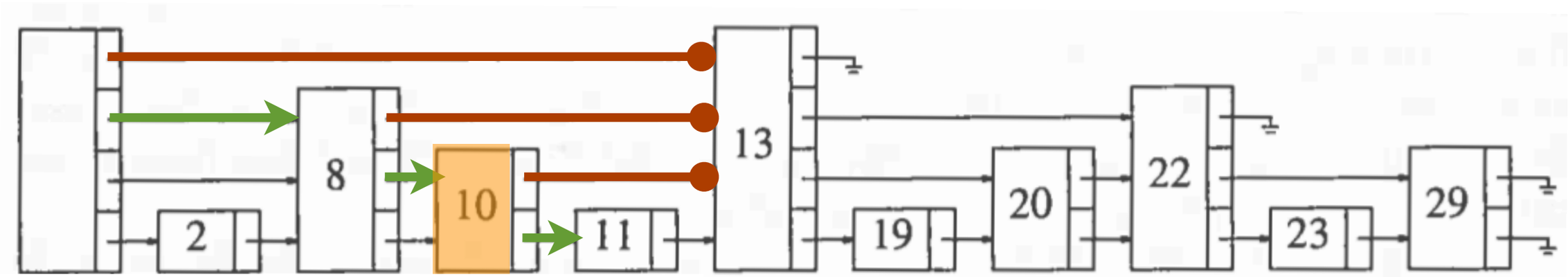
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

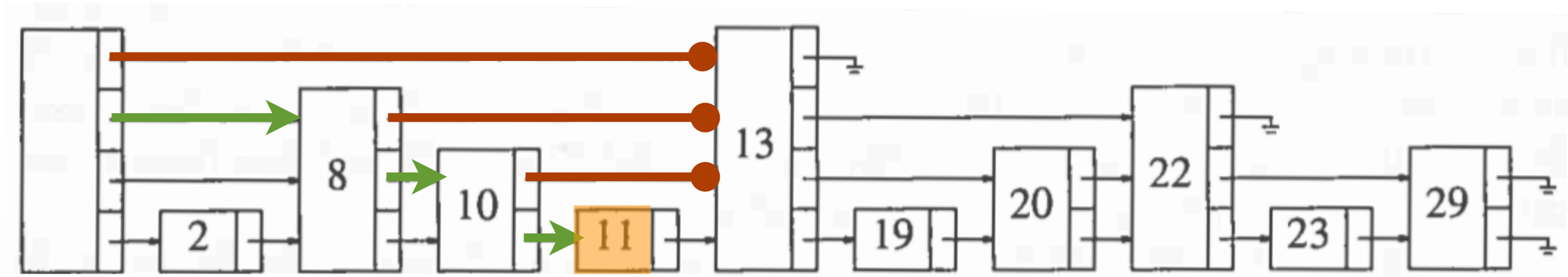
- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Sökning i en skiplista

Vi börjar på högsta nivån, och letar upp den sista noden som är ≤ x (det sökta värdet)

- sedan går vi ner en nivå, och gör om proceduren: letar upp den sista noden som är ≤ x
- vi fortsätter så tills vi antingen har hittat x, eller är på lägsta nivån och inte kan gå längre



Att lägga till i en skiplista

Om sökningen inte hittar målet, så hittar den iallafall dess föregångare i nivå-0-listan:

- det är efter den vi ska stoppa in det nya noden

Vi måste också veta vilken nivå den nya noden ska ha:

- nivån väljs slumpmässigt enligt en logaritmisk distribution: 50% chans för nivå 1, 25% för nivå 2, 12,5% för nivå 3, etc.

När vi sökte efter målet så gick vi igenom föregångaren i nivå-1-listan, och i nivå-2-listan, etc.

- då är det bara att peka om alla dessa föregångare till den nya noden

Effektivitet för en skiplista

Den första listan har ett konstant antal element
(k, som kanske är ca 2–3)

- den andra listan har dubbelt så många element, men vi har ju avgränsat sökrymden till halva antalet noder: alltså $k \cdot 2/2 = k$
- den tredje listan har i sin tur dubbel så många element, men nu har vi ju halverat sökrymden en gång till: alltså $k \cdot 4/4 = k$
- och så vidare…
- varje listsökning blir alltså konstant, O(1)

Den totala komplexiteten beror då på antalet listor,
dvs antalet nivåer:

- antalet nivåer är logaritmen av antalet noder n
- alltså är sökkomplexiteten logaritmisk, O(log n)
- (om slumptalsgenereringen är korrekt, förstås)