شاخص‌های چندسطحی و درختهای B

سید مهدی وحیدی پور

ارایه دوم: شاخص‌های چندسطحی

نکته

- مشکل اصلی نگه داشتن شاخص در حافظه جانبی این است که دسترسی به حافظه جانبی کند است
- دو مشکل اصلی:
  - **مشکل اول**: جستجو بر حسب شاخص باید سریع تراز جستجوی دودویی باشد.
  - **مشکل دوم**: درج و حذف باید با سرعت جستجو کردن انجام شود.

راهکار مشکل اول

- استفاده از درخت های دودویی صفحه ای

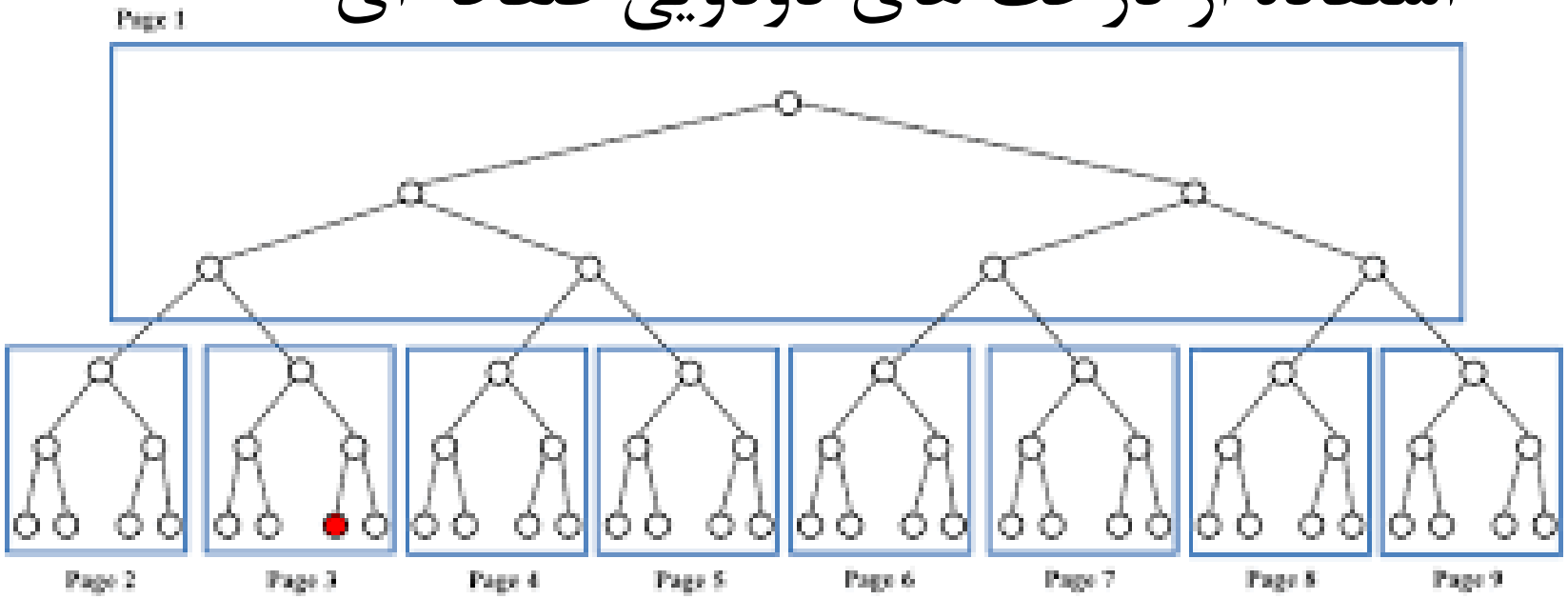
درخت دودویی صفحه‌ای سعی می‌کند با قرار دادن چندین گره دودویی در یک صفحه دیسک مشکل را حل کند. به این ترتیب با یک دستیابی به دیسک بخش بزرگی از درخت را می‌توان به دست آورد.

در بین تعداد زیادی کلید، با تعداد کمی دستیابی به دیسک، کلیدی را می‌توان جستجو کرد.

مشکل اول: جستجو بر حسب شاخص باید سریع تراز جستجوی دودویی باشد.

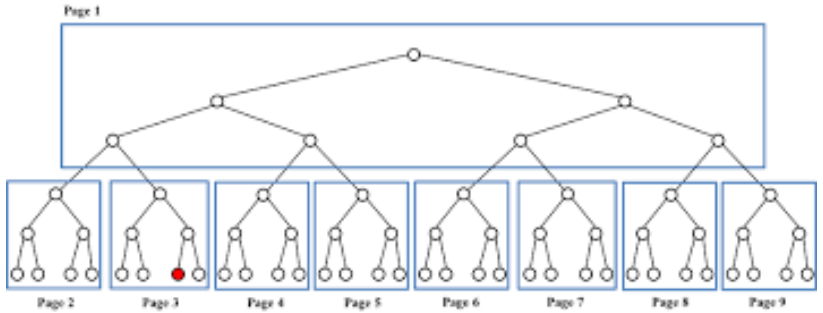
راهکار مشکل اول

- استفاده از درخت های دودویی صفحه ای



نمونه‌ای از درخت دودویی صفحه‌ای

راهکار مشکل اول



- تعداد پیگردهای لازم برای یک درخت جستجوی کاملا پر و موازنه شده، در بدترین حالت $\lg (N+1)$
- تعداد پیگردهای لازم برای نسخه صفحه‌ای یک درخت دودویی کاملا پر و موازنه شده، $\log_{K+1}(N+1)$
- k تعداد کلید در هر صفحه
- N تعداد کلید

راهکار مشکل دوم

- استفاده از درخت موازنه شده مانند درخت AVL

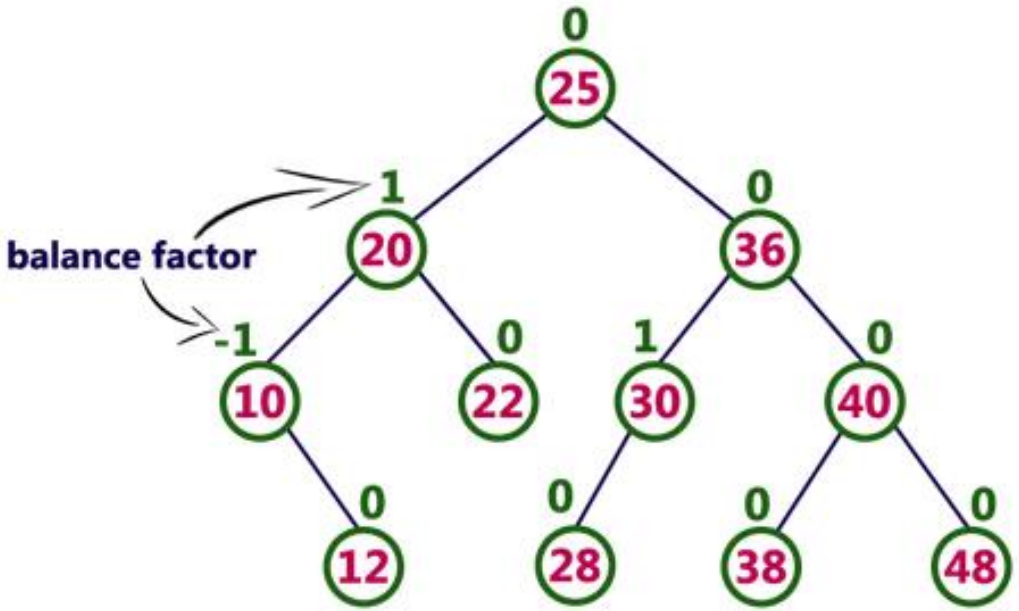
- با تعیین کردن حداکثر تفاوت مجاز در ارتفاع هر دو زیر درخت این درخت‌ها حداقل کارایی را در جستجو تضمین می‌کنند.

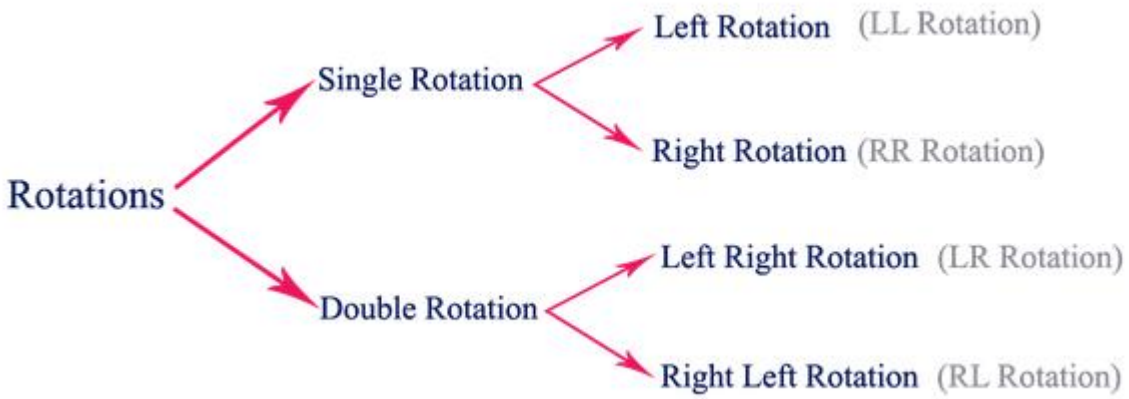
ارتفاع زیردرخت راست-ارتفاع زیردرخت چپ=ضریب تعادل

- برای اینکه هنگام درج در درخت AVL ویژگی خود را حفظ کند مستلزم چهار نوع چرخش است.

مشکل دوم: درج و حذف باید با سرعت جستجو کردن انجام شود.

درخت موازنه شده AVL



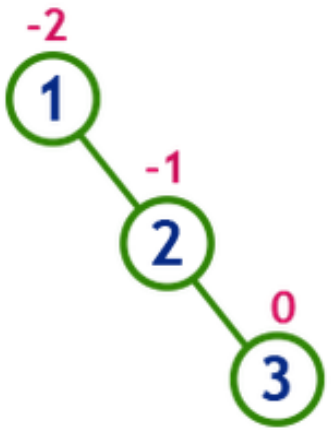


درخت موازنه شده AVL–ادامه

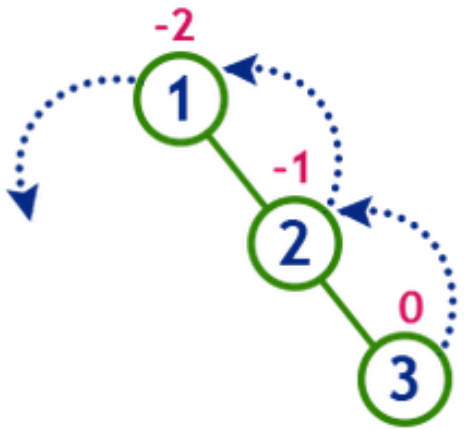
Single Left Rotation (LL Rotation)

In LL Rotation every node moves one position to left from the current position. To understand LL Rotation, let us consider following insertion operations into an AVL Tree...

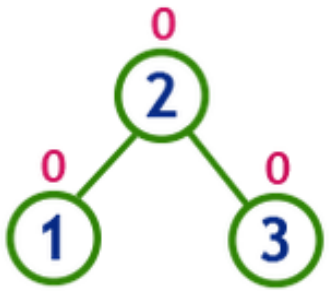
insert 1, 2 and 3



Tree is imbalanced



To make balanced we use LL Rotation which moves nodes one position to left



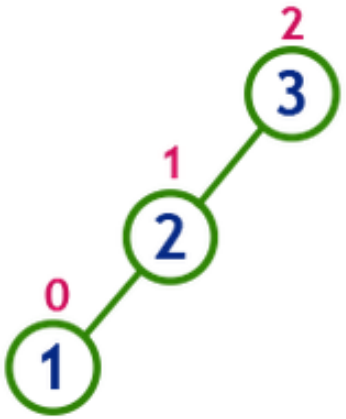
After LL Rotation Tree is Balanced

درخت موازنه شده AVL–ادامه

Single Right Rotation (RR Rotation)

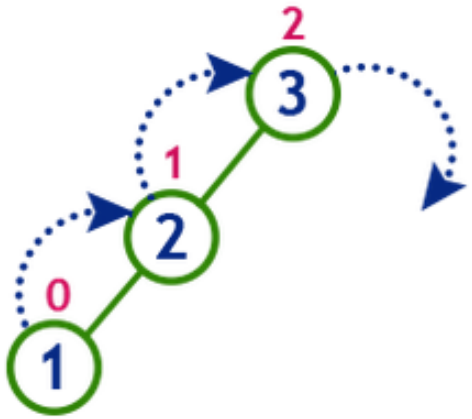
In RR Rotation every node moves one position to right from the current position. To understand RR Rotation, let us consider following insertion operations into an AVL Tree...

insert 3, 2 and 1

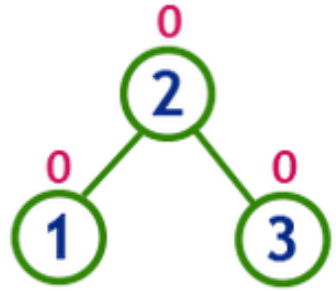


Tree is imbalanced

because node 3 has balance factor 2



To make balanced we use RR Rotation which moves nodes one position to right



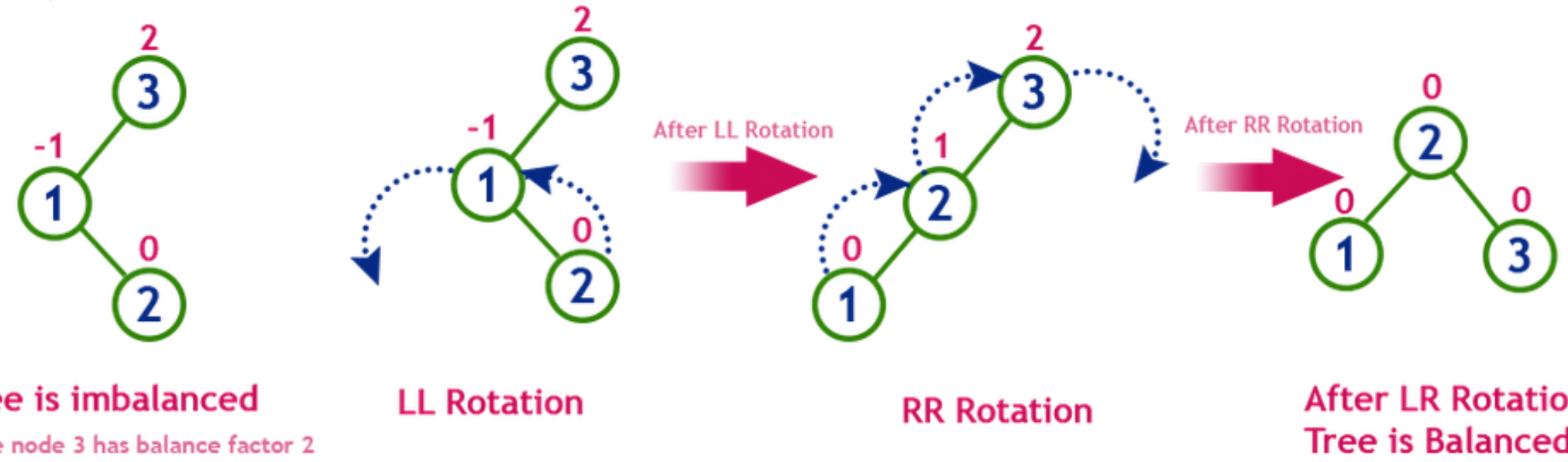
After RR Rotation Tree is Balanced

درخت موازنه شده AVL–ادامه

Left Right Rotation (LR Rotation)

The LR Rotation is combination of single left rotation followed by single right rotation. In LR Roration, first every node moves one position to left then one position to right from the current position. To understand LR Rotation, let us consider following insertion operations into an AVL Tree...

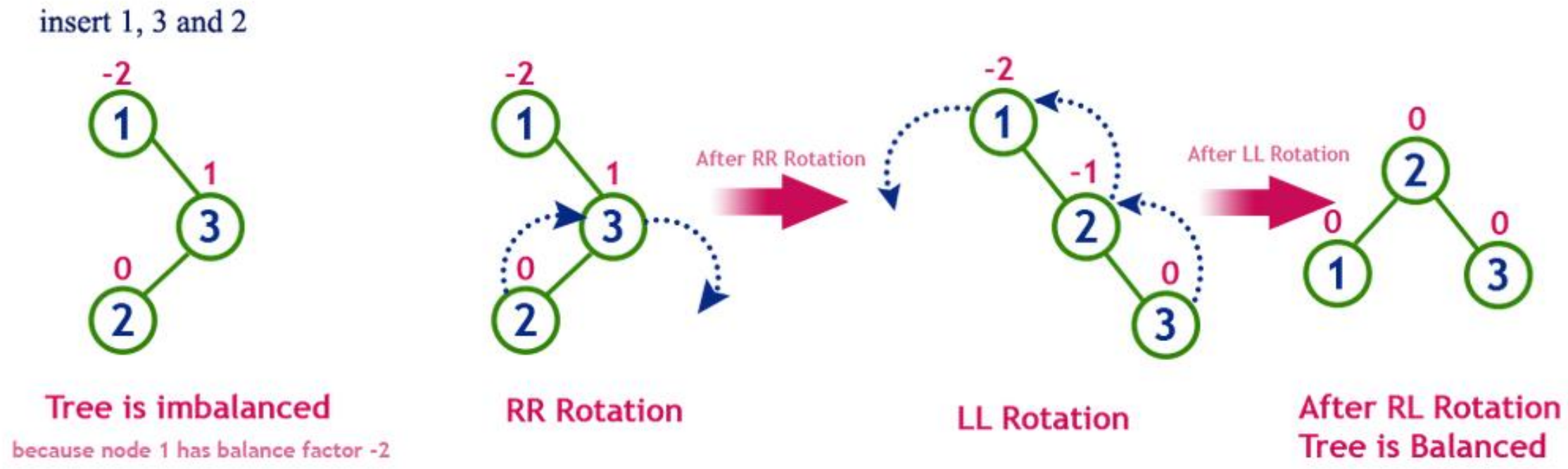
insert 3, 1 and 2



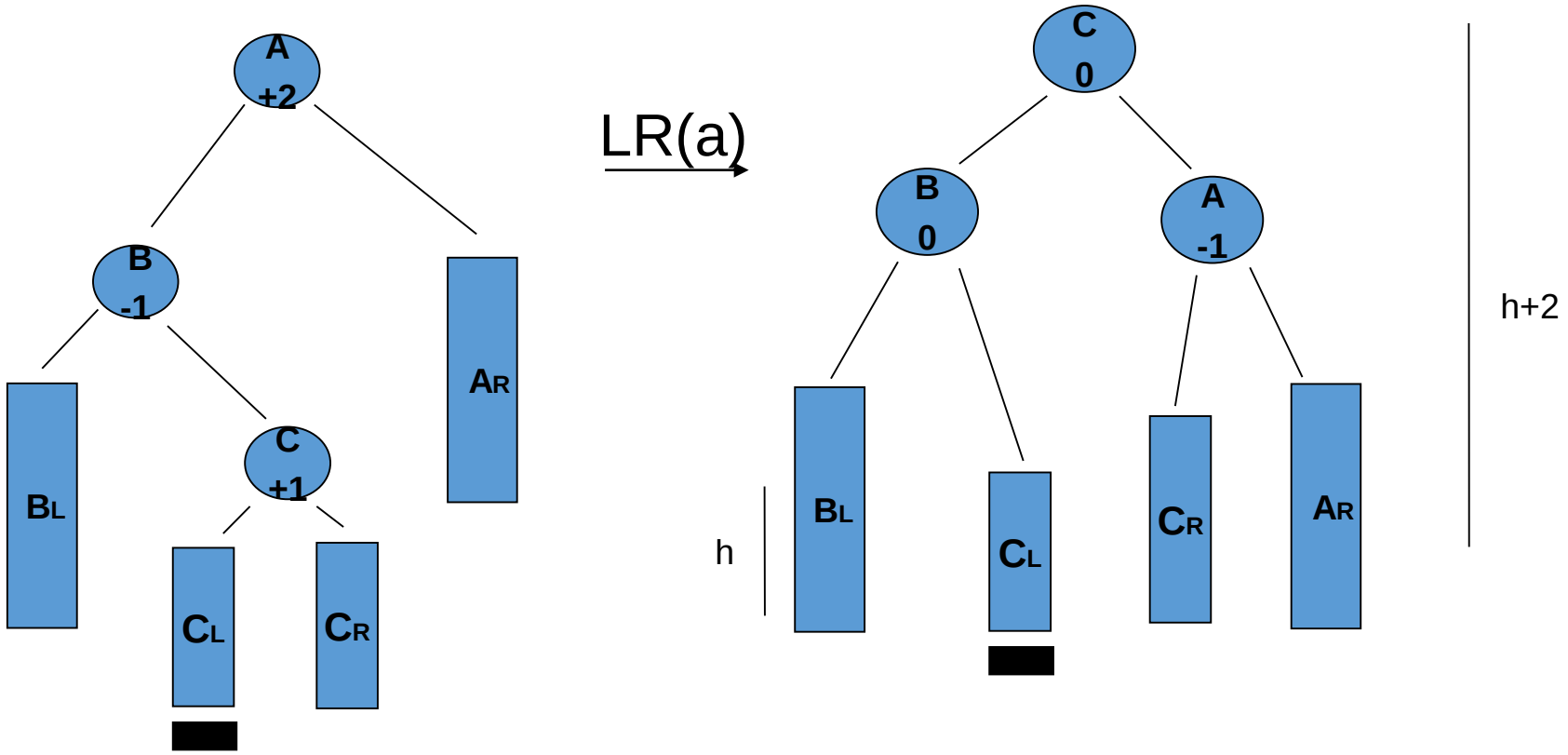
درخت موازنه شده AVL–ادامه

Right Left Rotation (RL Rotation)

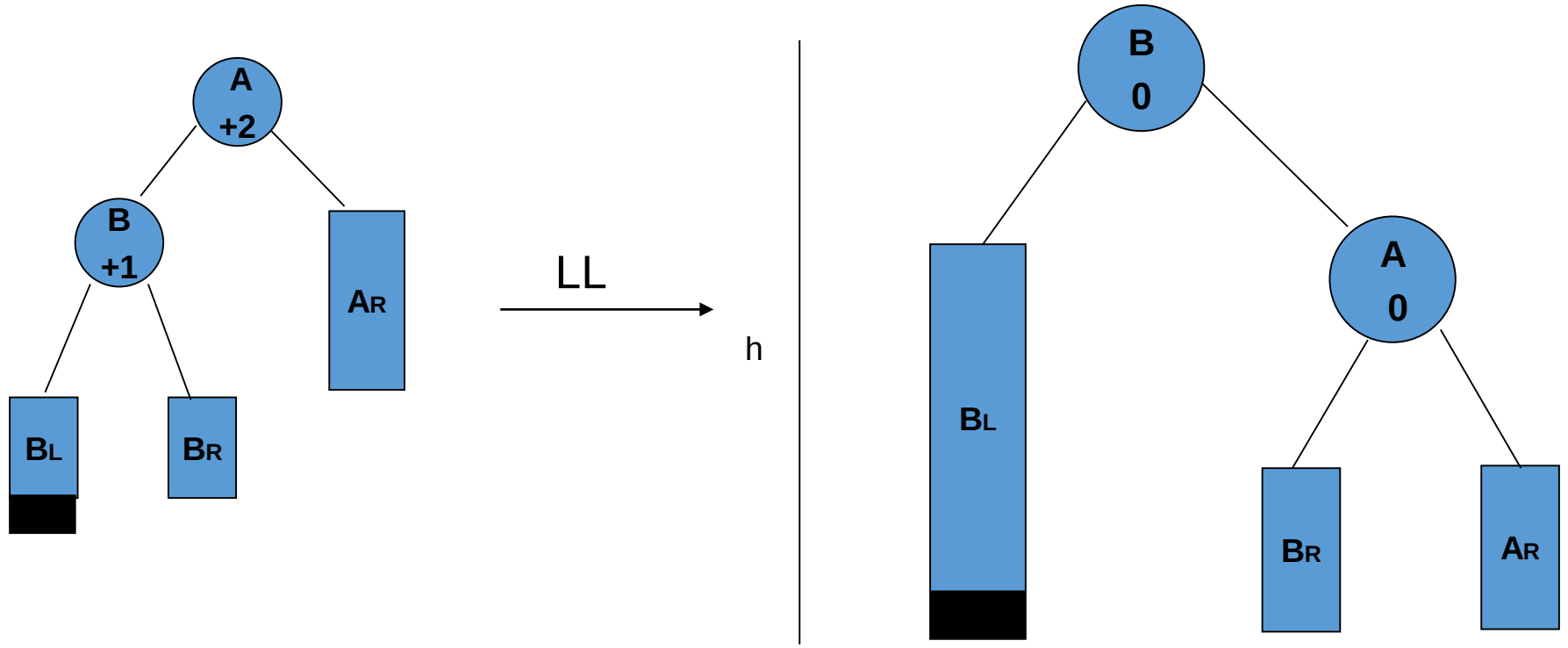
The RL Rotation is combination of single right rotation followed by single left rotation. In RL Roration, first every node moves one position to right then one position to left from the current position. To understand RL Rotation, let us consider following insertion operations into an AVL Tree...



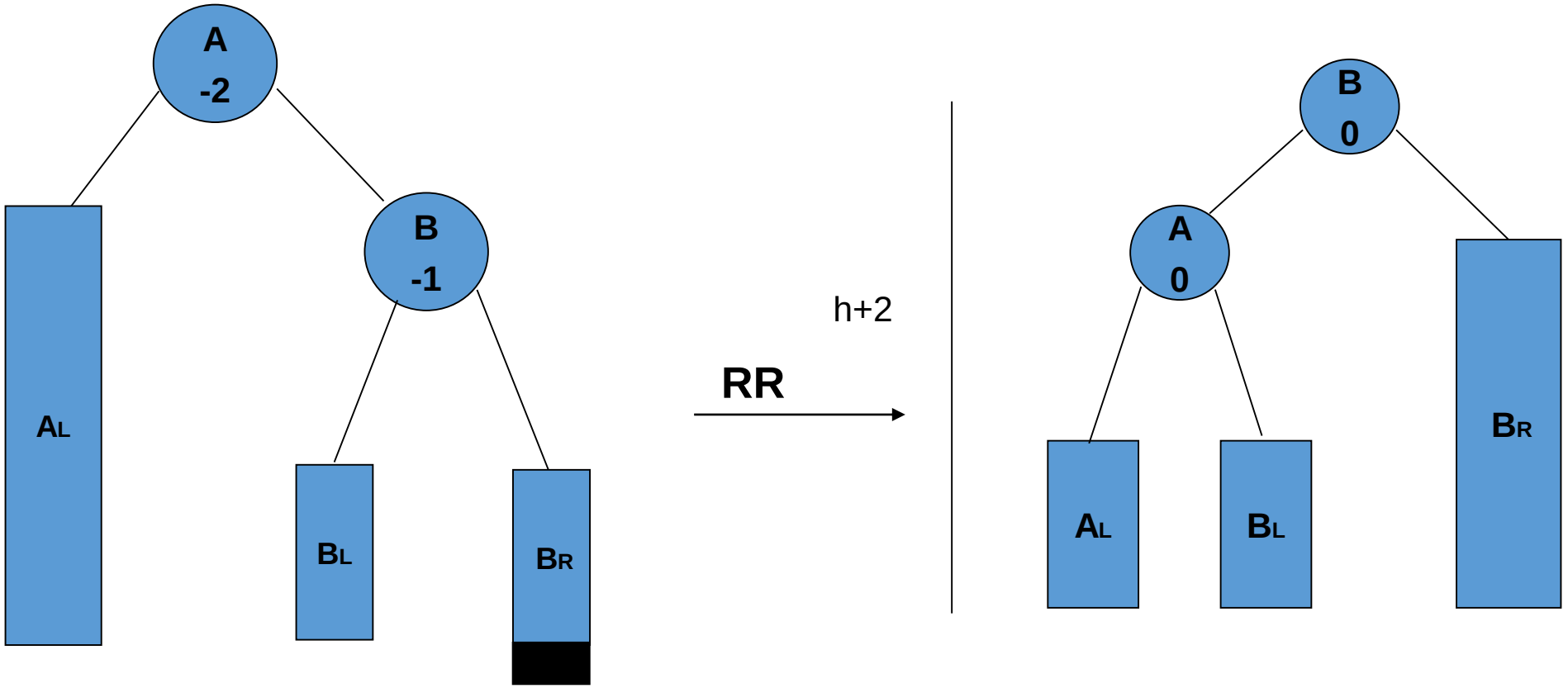
درخت موازنه شده AVL–ادامه



درخت موازنه شده AVL–ادامه



درخت موازنه شده AVL–ادامه



درخت موازنه شده AVL–ادامه

- درخت کاملا موازنه شده: جستجوی کلیدی در بین N کلید، در بدترین حالت باید lg(N+1) سطح را نگاه کند ولی در درخت AVL، باید 1.44 lg (N+2) سطح
- برای یک میلیون کلید: درخت موازنه شده ۲۰ تا ۲۱ سطح
- برای یک میلیون کلید: درخت AVL ۲۹ سطح
- درخت AVL تضمین می‌کند برای سازماندهی دوباره آن به بازسازی بیش از ۵ اشاره‌گر نیاز نیست.
- مطالعات تجربی وان‌دوری و گری (1974) نشان می‌دهد که این سازماندهی برای تقریبا نیمی از عملیات جمع و تقریبا یک چهارم از عمیات حذف لازم است.

تاریخچه درخت B

- داگلاس کومر، در مقاله با عنوان "درخت B درهمه جا" در اواخر دهه ی ۱۹۶۰
  - پیدا کردن روشی کلی برای ذخیره و بازیابی داده‌ها در فایل بزرگ بود، که امکان دسترسی سریع با حداقل زمان را فراهم کند.
- ر. بایر و پ. مک کرایت در سال ۱۹۷۲ مقاله ای به نام «سازمان دهی و نگهداری شاخص‌های مرتب شده‌ی بزرگ " درخت B را به جهان معرفی کردند.

Rudolf Bayer

Born	May 7, 1939 (age 78)
Nationality	German
Alma mater	University of Illinois at Urbana–Champaign
Known for	B-tree UB-tree red-black tree
Awards	Cross of Merit, First class (1999), SIGMOD Edgar F. Codd Innovations Award (2001)
	Scientific career
Institutions	Technical University Munich
Thesis	*Automorphism Groups and Quotients of Strongly Connected Automata and Monadic Algebras* (1966)
Doctoral advisor	Franz Edward Hohn[1]

درخت B

هر گره درخت B یک رکورد شاخص است .هر کدام از این رکورد ها تعداد یکسانی از جفتهای کلید – آدرس دارند که مرتبه درخت نام دارد.

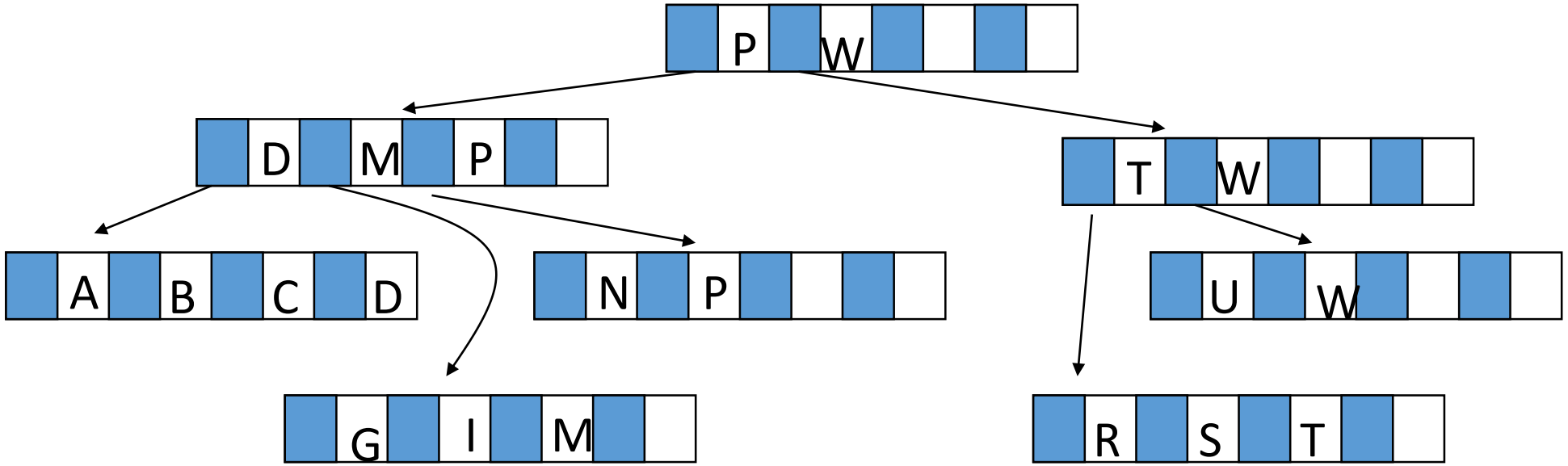
ایده اصلی

- درخت یک رکورد اندیس است
- تعدادی کلید-آدرس در داخل نودها نگهداری می شود.
- تعداد حداکثر آنها به مرتبه درخت بی شناخته می شود.
- هر نود می تواند به تعداد حداقل کلید در خود نگهداری کند .
- معمولا نصف مرتبه درخت

خواص یک درخت B از مرتبه m

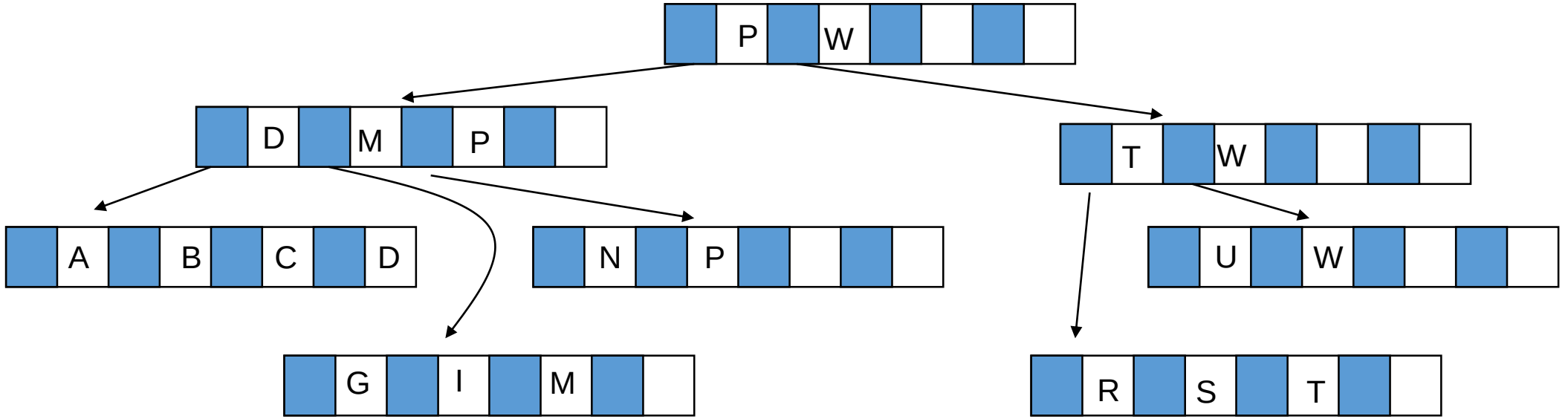
- هر صفحه حد اکثر m فرزند دارد.
- هر صفحه بجز ریشه‌ها و برگها حداقل [m/2] فرزند دارد.
- ریشه حداقل دو فرزند دارد.
- تمام برگها در یک سطح قرار دارند.
- سطح برگها, یک شاخص کامل و مرتب شده از داده‌های مربوط به درخت را ایجاد می‌کند.

مثالی از درخت B



Note: references to actual record only occur in the leaf nodes. The interior nodes are only higher level indexes (this is why there are duplications in the tree)

جستجو در درخت B



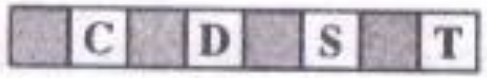
- ***Problem 1***: Look for L
- ***Problem 2***: Look for S

مراحل درج کردن

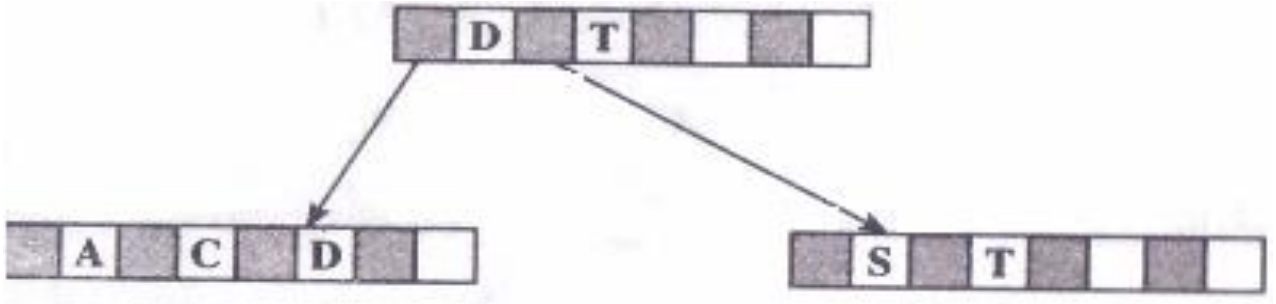
- جستجو تا سطح برگ با استفاده از متد FINDLEAF قبل از تکرار.
- درج تشخیص سرریز و تقسیم کردن در مسیر رو به بالا.
- ایجاد یک ریشه جدید، در صورتی که ریشه فعلی تقسیم شده است.

مثال

a) Insert of *C, S, D, T* into the initial node.



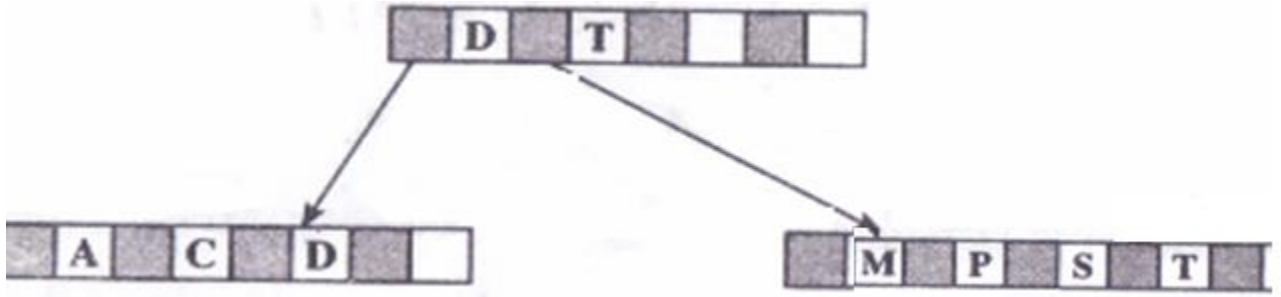
b) Insertion of *A*



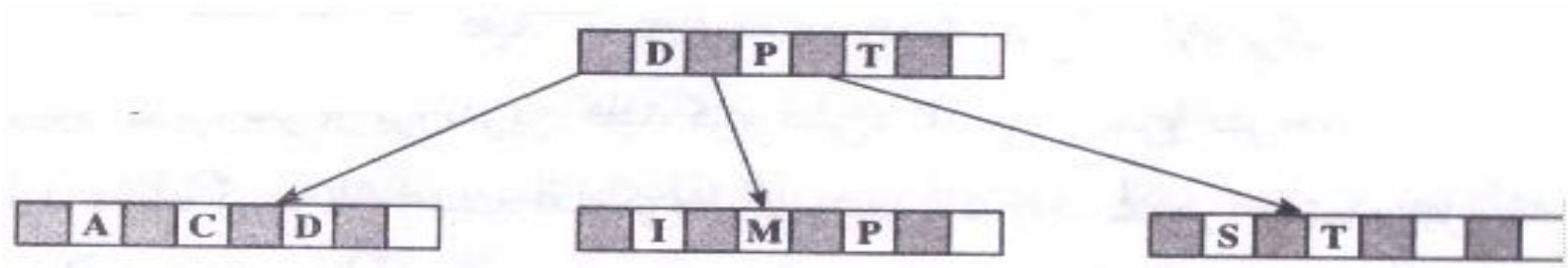
c) Insertions of *M* and *P ?*

مثال –ادامه

c) Insertions of *M* and *P* ?

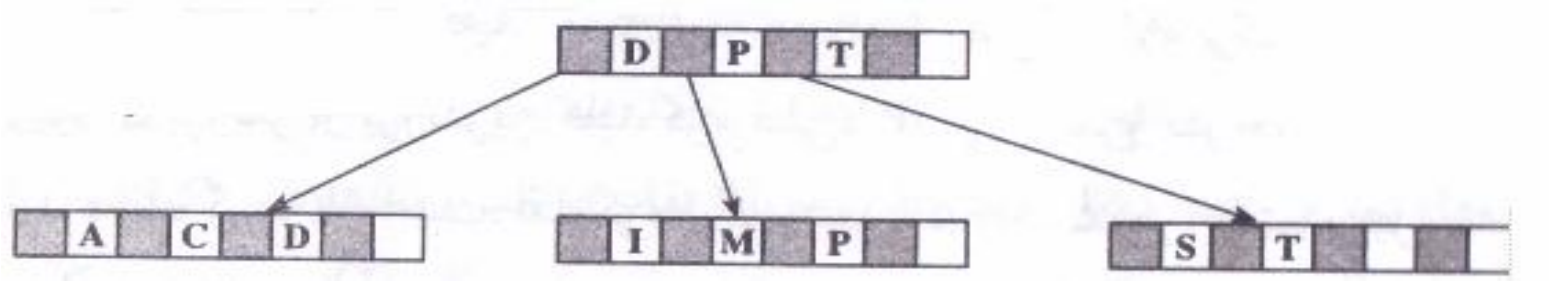


d) Insertion of *I*

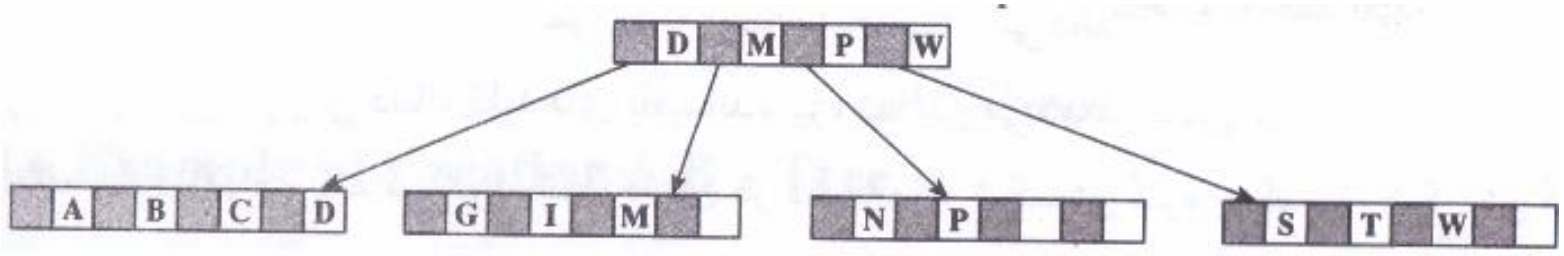


e) Insertions of *B, W, N* and *G*

مثال –ادامه

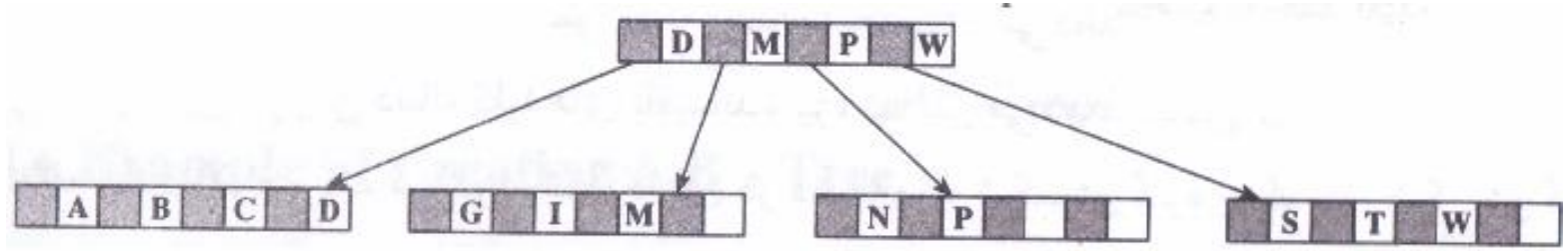


e) Insertions of *B, W, N* and *G*

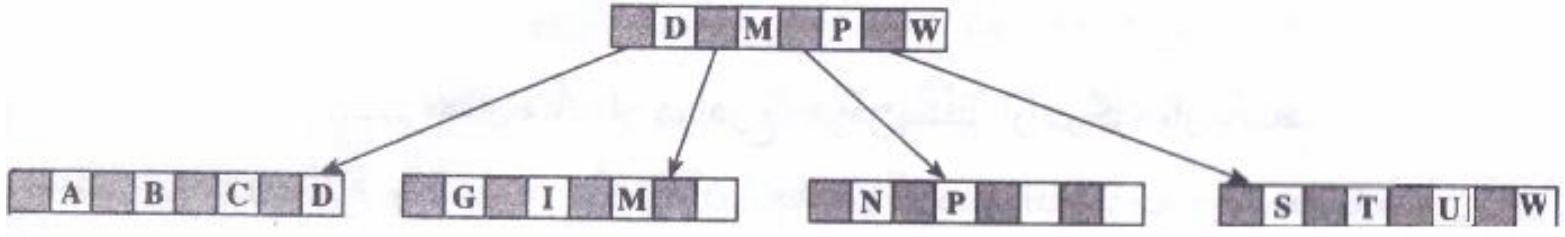


f) Insertion of *U?*

مثال –ادامه

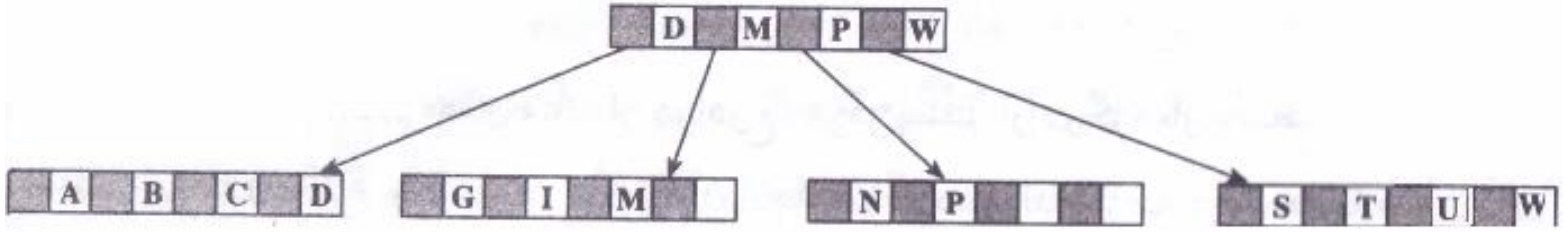


f) Insertion of *U?*

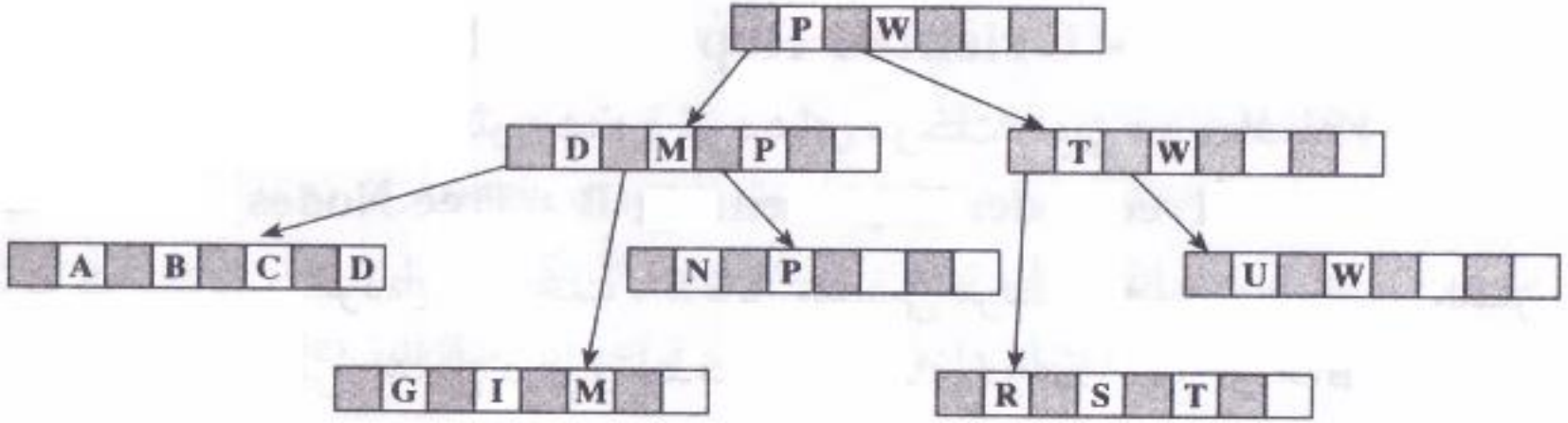


r) Insertion of *R?*

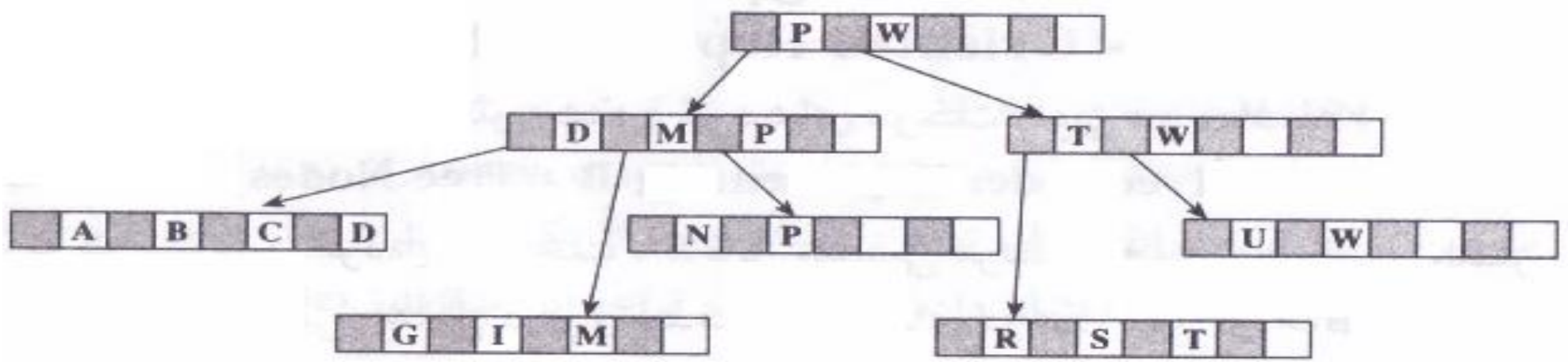
مثال –ادامه



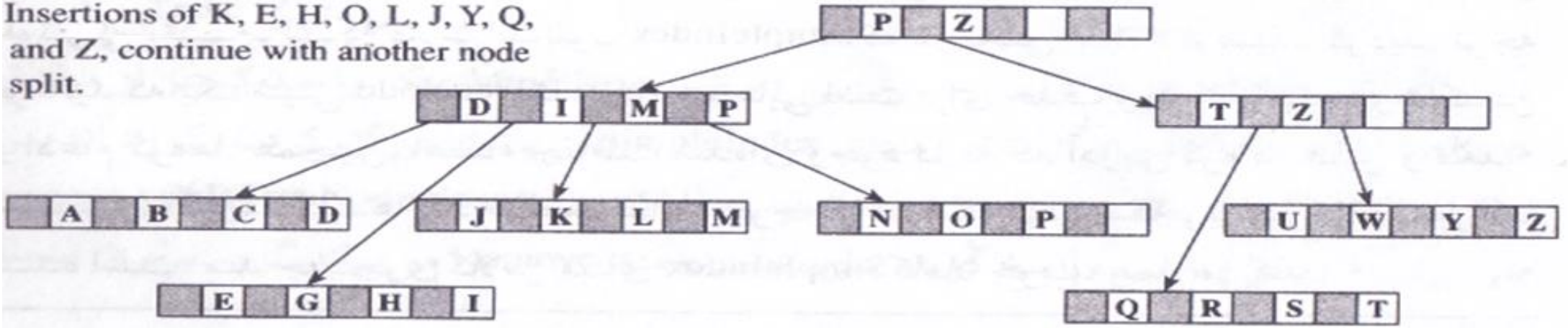
r) Insertion of *R?*



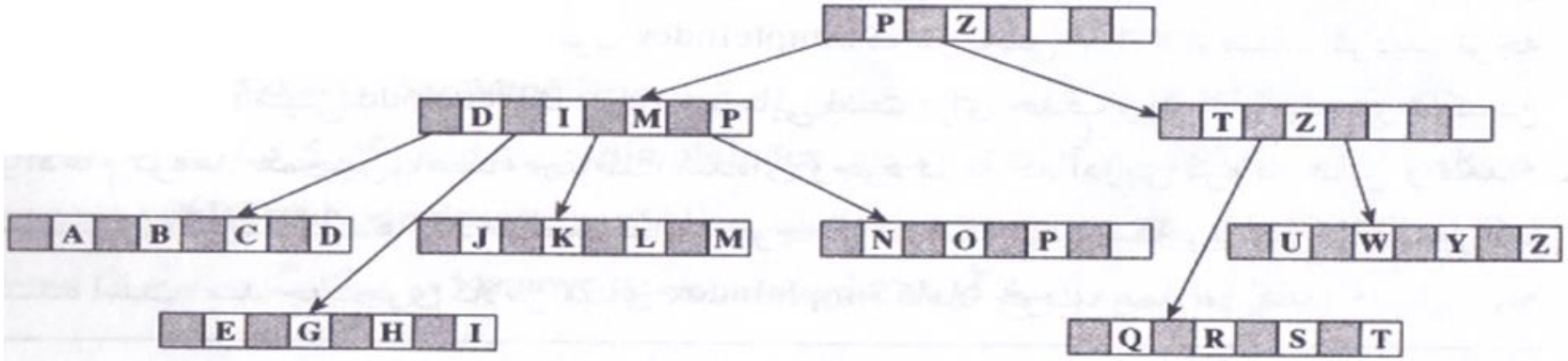
مثال –ادامه



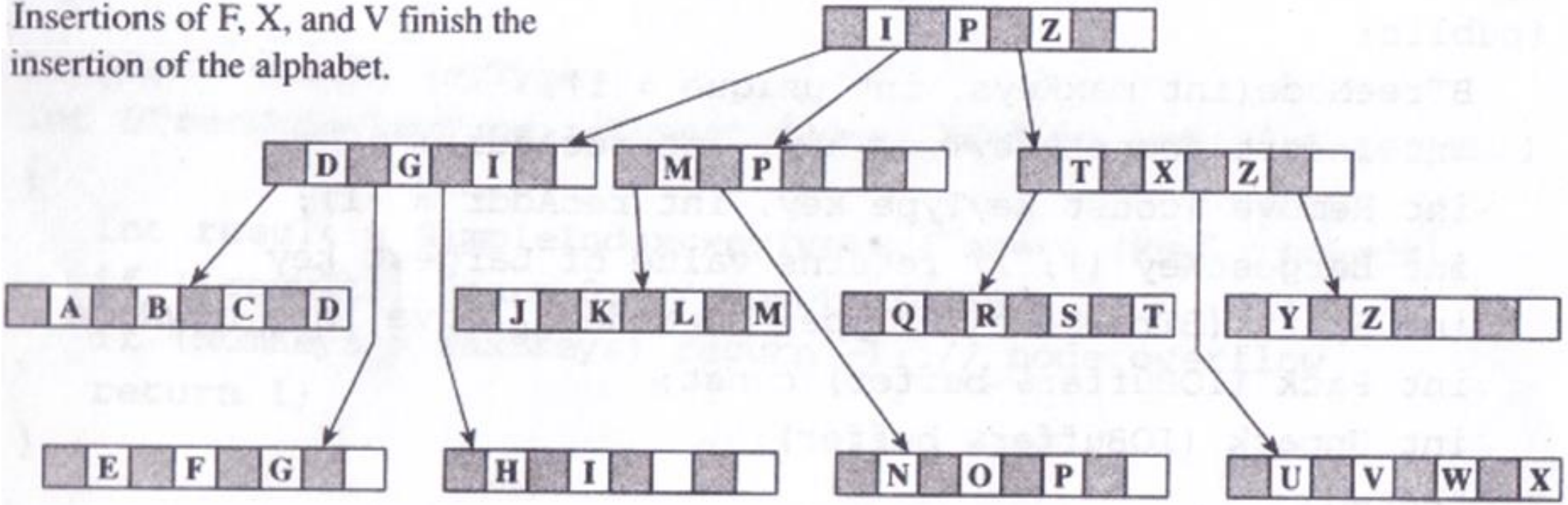
Insertions of K, E, H, O, L, J, Y, Q, and Z, continue with another node split.



مثال –ادامه

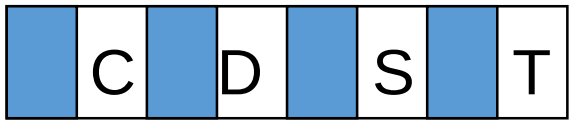


Insertions of F, X, and V finish the insertion of the alphabet.

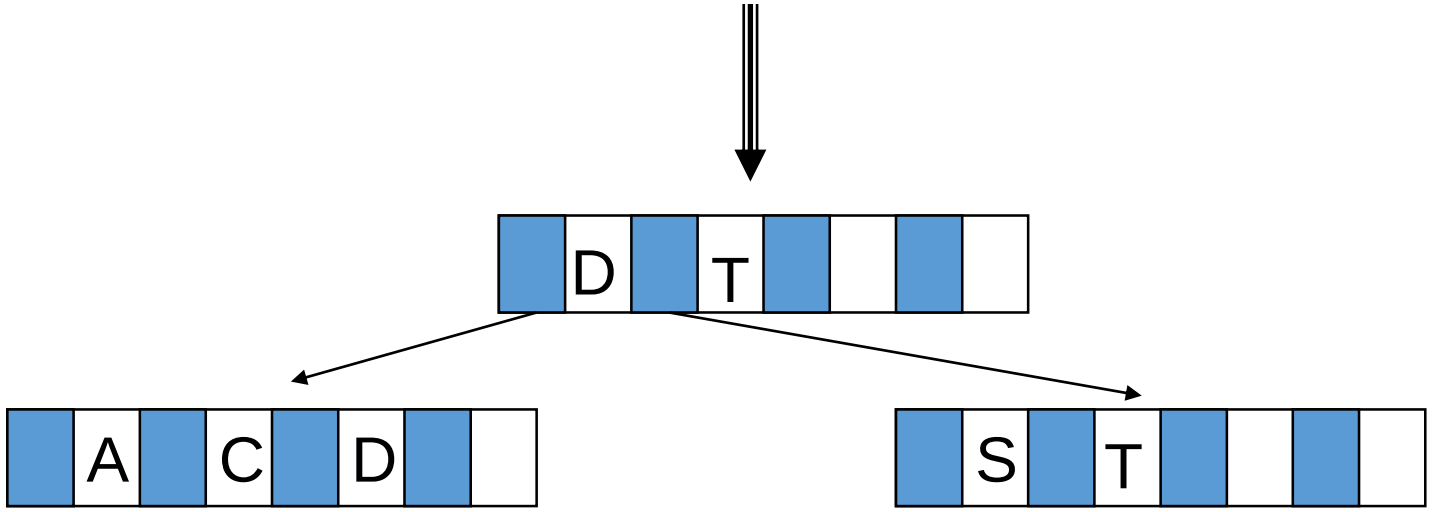


Split **اضافه سازی بدون نیاز به**

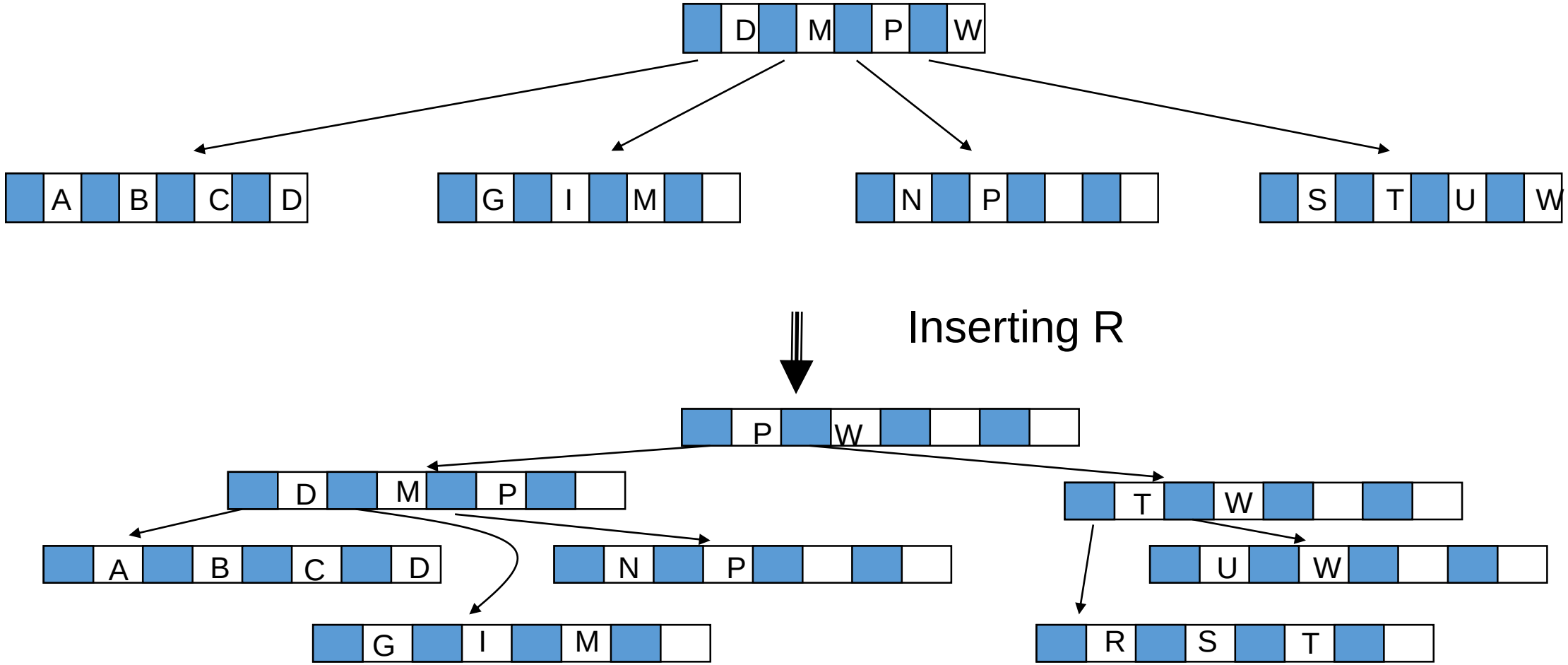
After inserting C, S, D, T:



Inserting A



اضافه سازی همراه با شکست چند مرحله ای



بررسی بدترین حالت

- تعداد یک میلیون کلید، درخت بی از مرتبه ۵۱۲. تعداد حداکثر دسترسی؟
- به عبارت دیگر: حداکثر عمق درخت
  - هر کلید در برگ درخت است: پس حداکثر عمق درخت بی با یک میلیون کلید در برگ؟
- ماکزیمم زمانی رخ می‌دهد که تمام نودها حداقل تعداد فرزند را داشته باشد.
  - حداقل برای ریشه ۲ و برای سایر گره‌ها $\lceil m/2 \rceil$ است

بررسی بدترین حالت – ادامه

the minimum number of descendants	Level
2	1
$2 * \lceil m/2 \rceil$	2
$2 * \lceil m/2 \rceil * \lceil m/2 \rceil$	3
$2 * \lceil m/2 \rceil^3$	4
….	…
$2 * \lceil m/2 \rceil^{d-1}$	d

بررسی بدترین حالت – ادامه

- برای سطح d از درخت بی، حداقل تعداد فرزندانی که از آن سطح به دست می‌آید

$$2\lceil m/2 \rceil^{d-1}$$

- برای سطح برگ با N کلید ..

$$N \geq 2\lceil m/2 \rceil^{d-1}$$

$$\Leftrightarrow d \leq 1 + \log_{\lceil m/2 \rceil}(N/2)$$

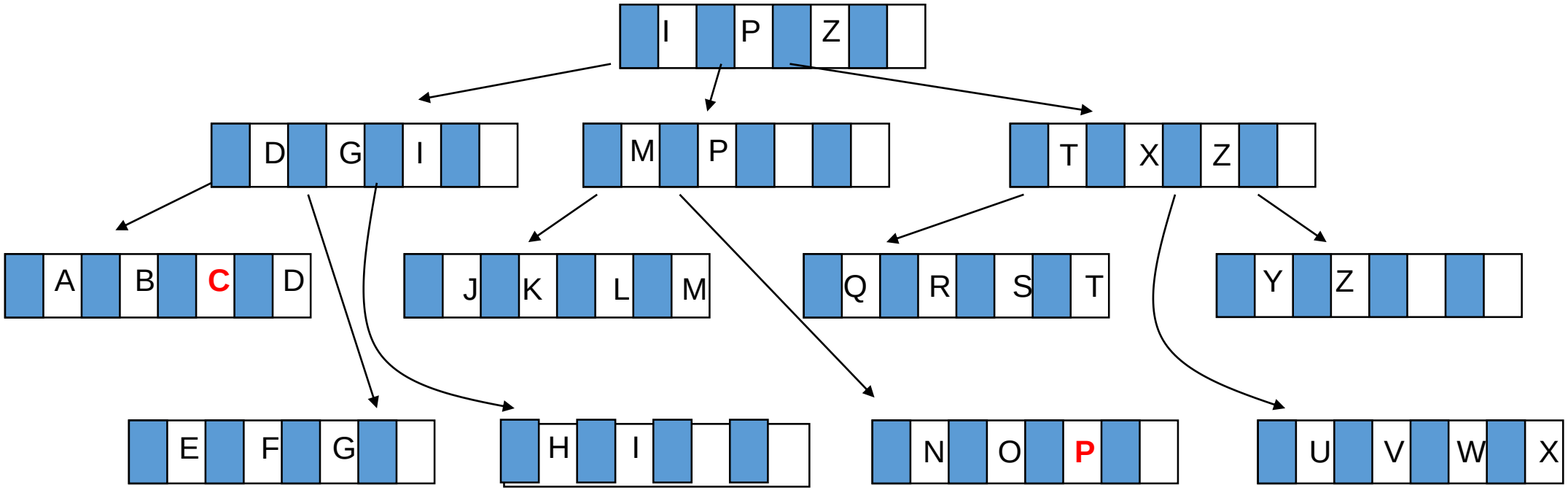
- برای m=512 و N=1000000

$$d \leq 3.37$$

حذف از درخت بی

- قوانین حذف کلید k از نود n
  - اگر به اندازه کافی کلید دارد و k بزرگترین کلید نیست، براحتی حذف می‌شود.
  - اگر به اندازه کافی کلید دارد و k بزرگترین است، بعد از حذف k، مقدار بزرگترین جدید در یک سطح بالاتر با مقدار k جابجا شود.
  - اگر n با حذف k، به اندازه‌ی کافی کلید ندارد (n دقیقا به اندازه مینیمم کلید دارد) و یکی از همسایه‌هایش (sibling) به اندازه کافی کم کلید (few enough) دارد آنگاه این دو را با هم ترکیب (**merge**) می‌کنیم. کلید را از والد حذف می‌کنیم.
  - اگر n با حذف k، به اندازه‌ی کافی کلید ندارد (n دقیقا به اندازه مینیمم کلید دارد) و یکی از همسایه‌هایش (sibling) زیاد کلید دارد، آنگاه کلیدها را بازتوزیع (**redistribute**) می‌کنیم. تعدادی کلید از همسایه گرفته و متناسب با آن در والد تغییرات ایجاد می‌کنیم.

مثال: حذف از درخت بی



- ***Problem 1***: Delete C
- ***Problem 2***: Delete P
- ***Problem 3***: Delete H

بازتوزیع در هنگام درج

- بازتوزیع در هنگام درج سعی می‌کند ایجاد صفحه جدید در درخت اندیس را به تعویق بیاندازد؛ ایجاد صفحه جدید آخرین راه‌حل است.
- به جای ایجاد صفحه جدید و تقسیم کلیدها بین صفحات، سعی می‌کند کلیدها را به همسایه‌ها منتقل کند
- درخت بی استار این ایده را پیاده کرده است.

خواص درختهای B*

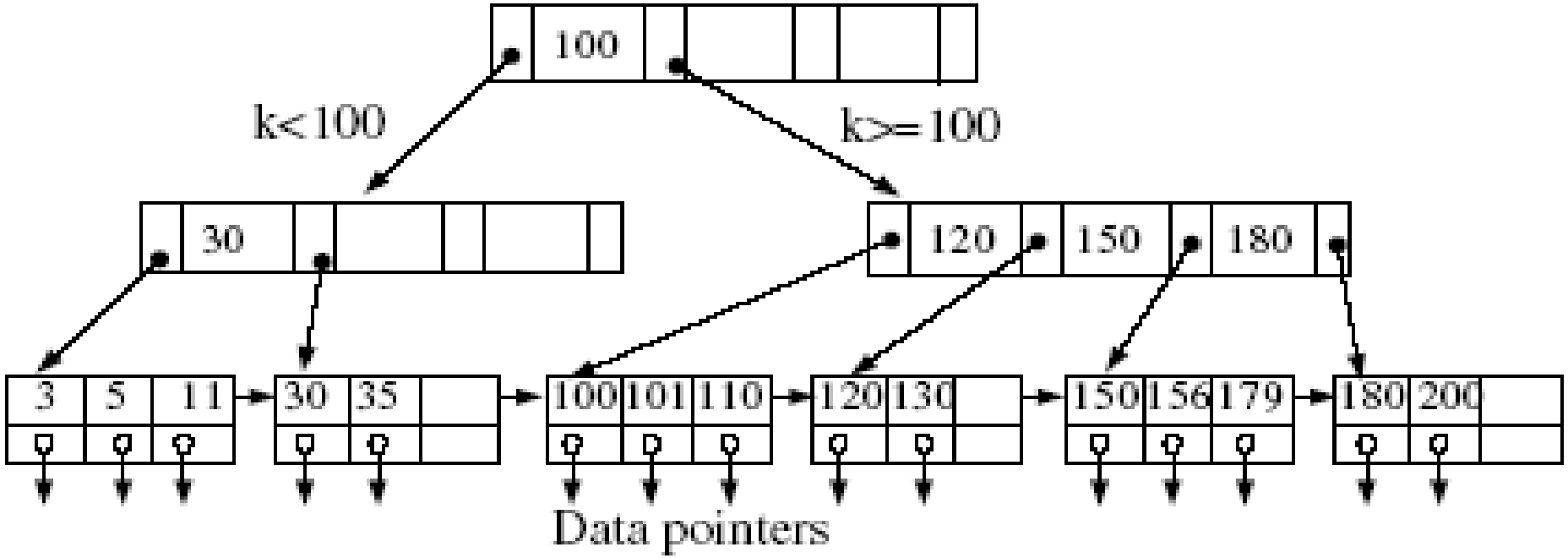
- هر صفحه حد اکثر m فرزند دارد.
- هر صفحه بجز ریشه حداقل[(2m-1)/3] فرزند دارد.
- ریشه حداقل دو فرزند دارد.
- تمام برگها در یک سطح قرار دارند.

درخت B^+

- B tree
  - اشاره‌گرهای داده ممکن است در نودهای داخلی ذخیره شوند.
  - هر مقدار از فیلدهای جستجو فقط یک بار ظاهر می‌شوند.

- B+ tree
  - اشاره‌گرهای داده فقط در برگ‌ها ذخیره می‌شوند.
  - نودهای داخلی فقط شامل کلیدها و اشاره‌گرهای درخت هستند.
    - بنابراین نودهای داخلی قابلیت ذخیره تعداد بیشتری از اشاره‌گرها را دارند.
    - زمان جستجو به خاطر کاهش تعداد سطوح کاهش می‌یابد.
    - به خاطر نداشتن اشاره‌گرهای تهی در برگها فضای تلف شده نداریم.

درخت B^+ – ادامه



- Internal node of order p=4, leaf nodes of order 3
  - i.e. Internal/leaf nodes must have between 2 and 4 pointers
- Leaf nodes are chained using sequence pointers

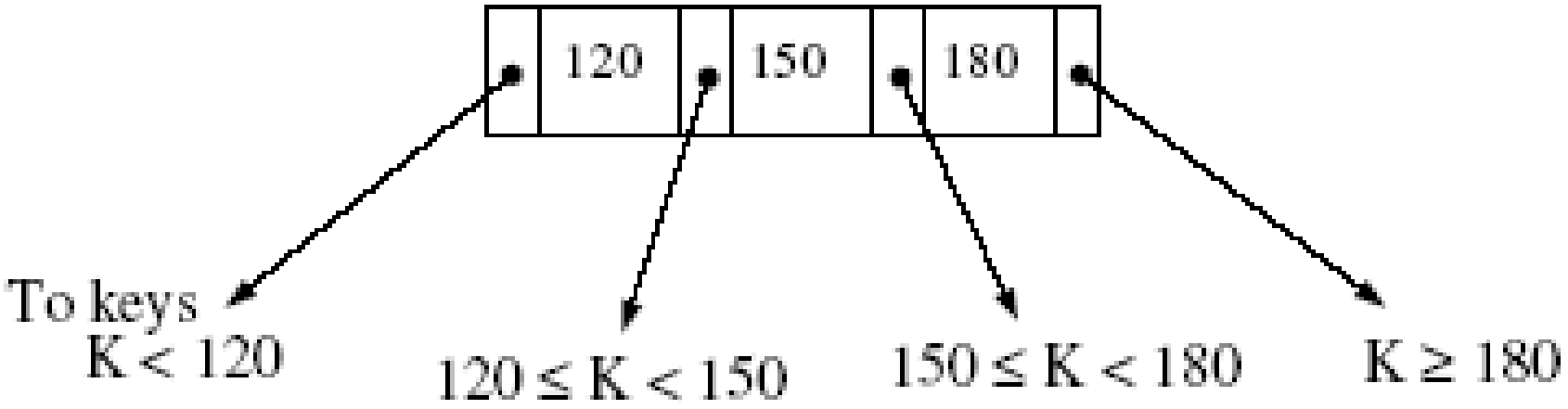
درخت B^+–نودهای داخلی

- هر نود داخلی در b+ tree به شکل زیر است:

$$<P_1, K_1, P_2, K_2,\ldots,P_{q-1},K_{q-1}, P_q>$$

- P_iیک اشاره گر درخت است که به نودهای دیگر در b+tree اشاره می‌کند.
- برای هر نود داخلی: $K_1 < K_2 < .. < K_{q-1}$
  - کلید کوچکتر از K_i == رفتن به سمت چپ K_i (اشاره گر چپ)
  - کلید بزرگتر یا مساویK_i == رفتن به سمت راست K_i (اشاره‌گر راست)

درخت B^+–نودهای داخلی

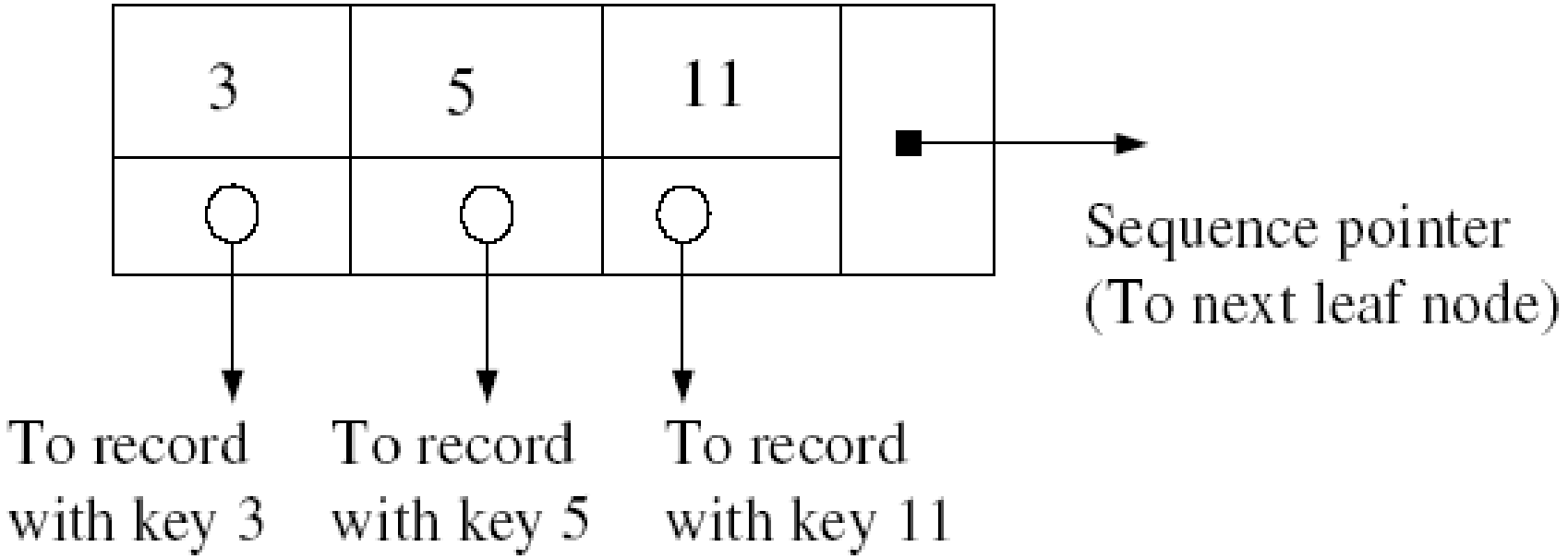


درخت B^+–برگها

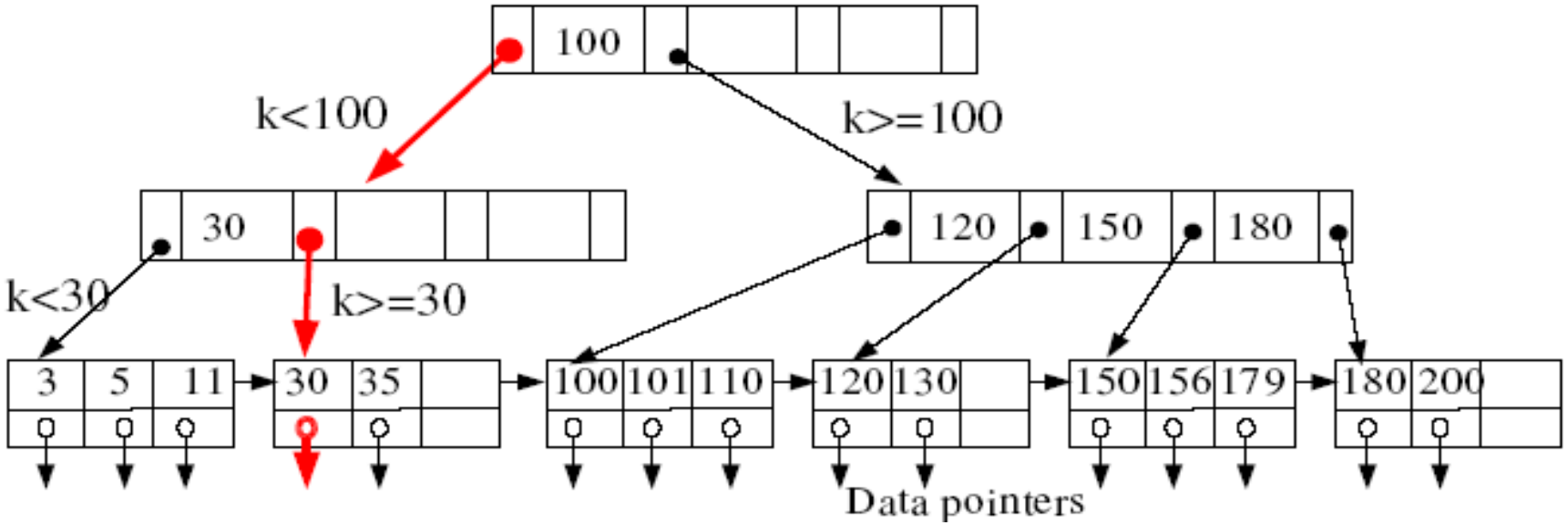
- هر نود برگ به شکل زیر است:
  - $<(K_1,Pr_1), (K_2,Pr_2), ..,(K_{q-1}, Pr_{q-1}), P_{next}>$
  - P_{next}اشاره گر ترتیبی است که به برگ بعدی در b^+tree اشاره می کند.
  - Pr_i یک اشاره گر داده است که به رکوردی که حاوی کلید K_iاست اشاره می کند.
    - یا به بلاکی از اشاره گر های داده اشاره می کند.

- در هر برگ:$K_1 < K_2 < ... < K_{q-1}$
- هر نود برگ حد اقل $p/2$اشاره گر دارد. (p درجه درخت)
- همه نودهای برگ در یک سطح قرار دارند.

درخت B^+–برگها


3
5
11
Sequence pointer
(To next leaf node)
To record
with key 3
To record
with key 5
To record
with key 11

درخت B^+–جستجو



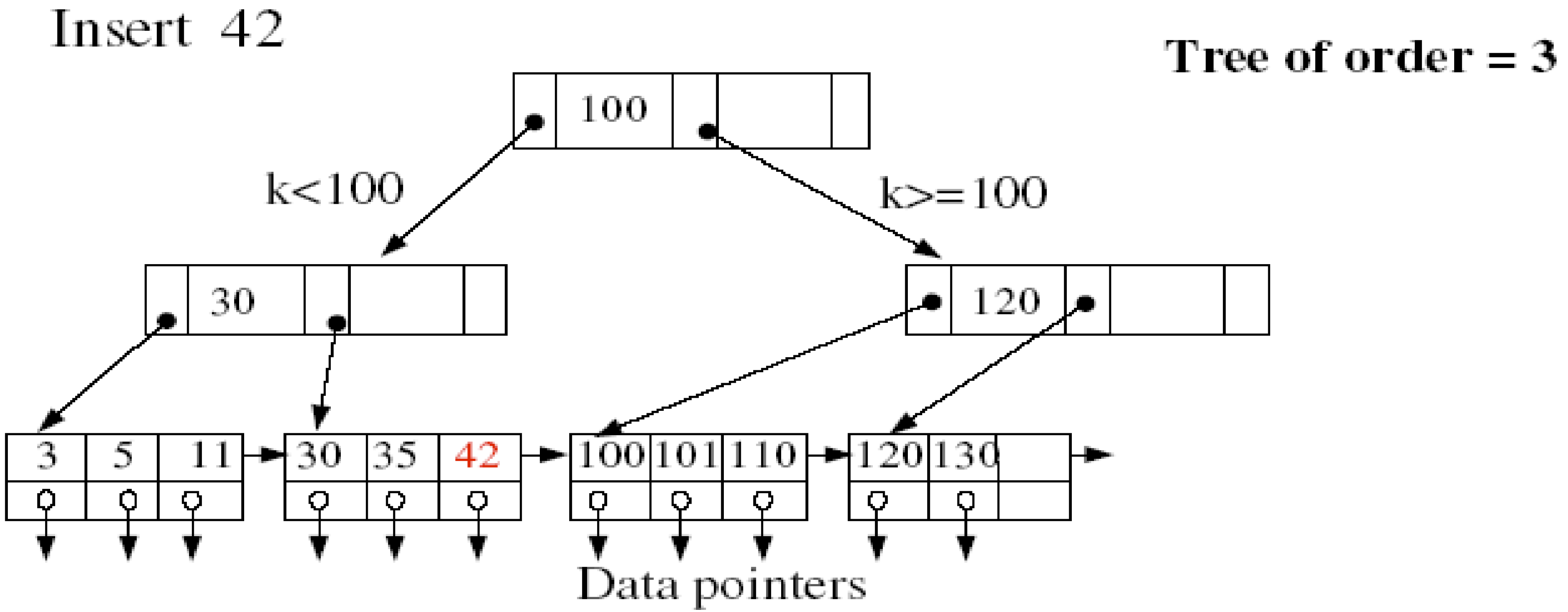
- در هر سطح کوچکترین کلید K_i که بزرگتر از فیلد جستجوی ما است پیدا می کنیم.
  - اشاره گرهای مربوط به ان را دنبال می کنیم.
  - اگر چنین کلیدی پیدا نشد اخرین اشاره گر در نود را دنبال می کنیم... تا وقتی که به یک نود برگ برسیم.

درخت B^+–درج

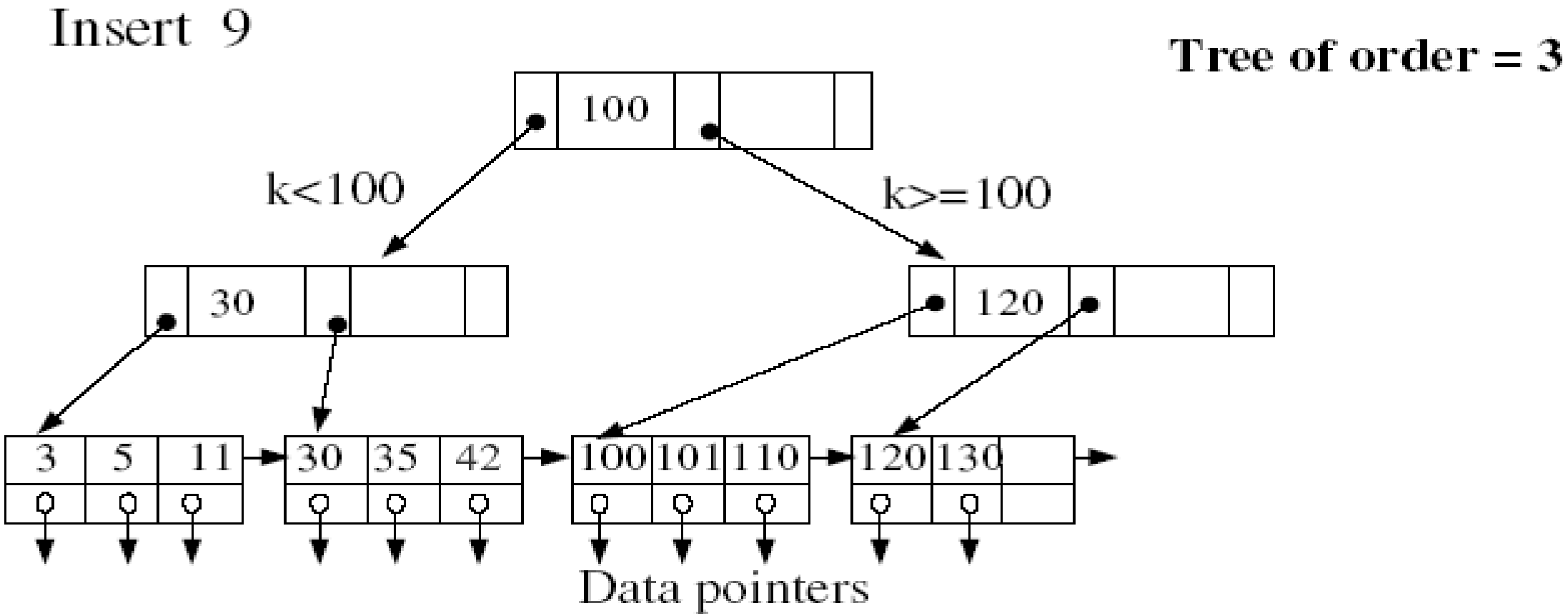
۴ نوع :

1) نوع ساده: فضای کافی برای درج در نود برگ وجود داشته باشد.
2) باعث ایجاد overflow در نود برگ شود.
3) باعث ایجاد overflow در نود داخلی شود.
4) باعث ایجاد ریشه جدید شود.

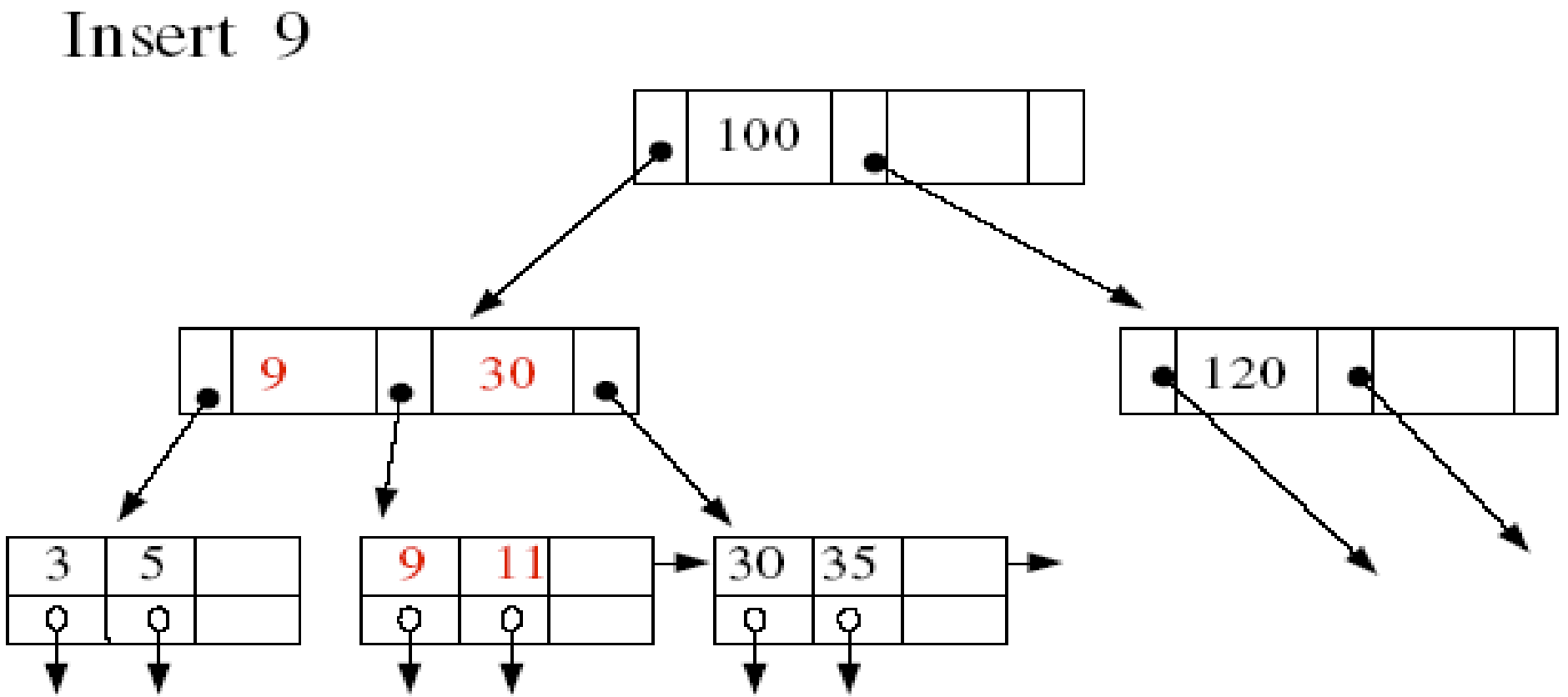
درخت B^+–درج نوع ۱



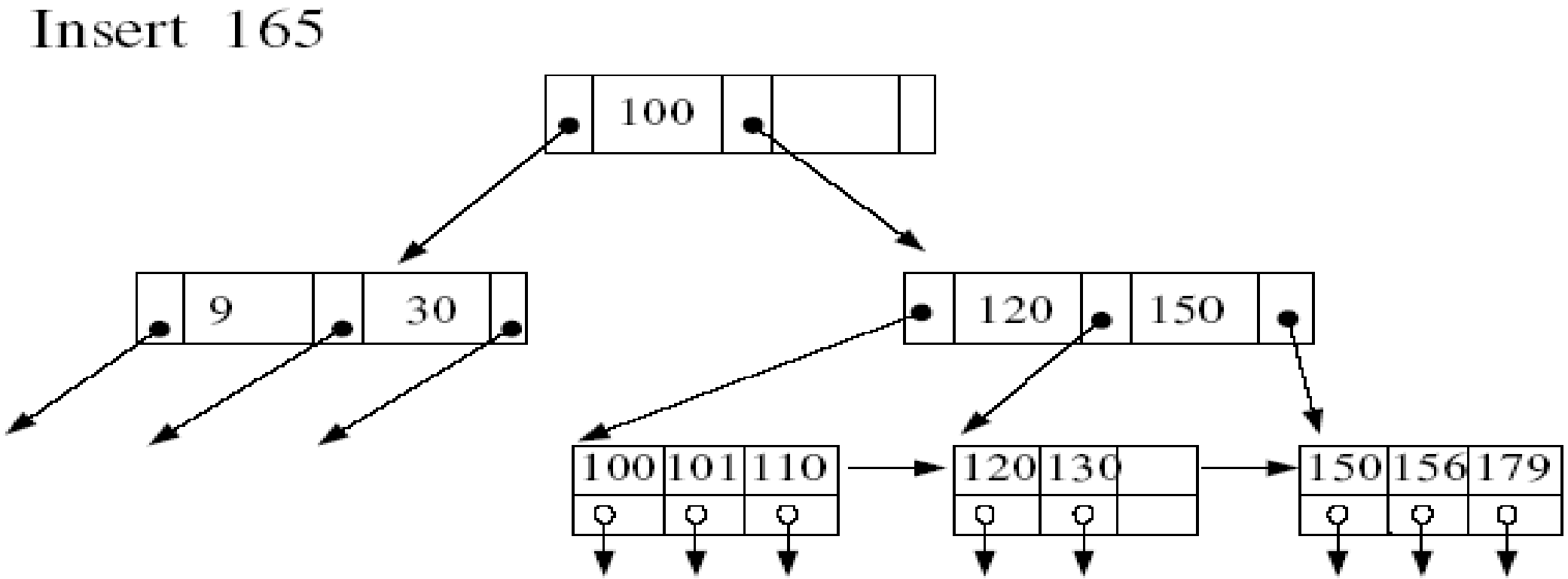
درخت B^+–درج نوع ۲



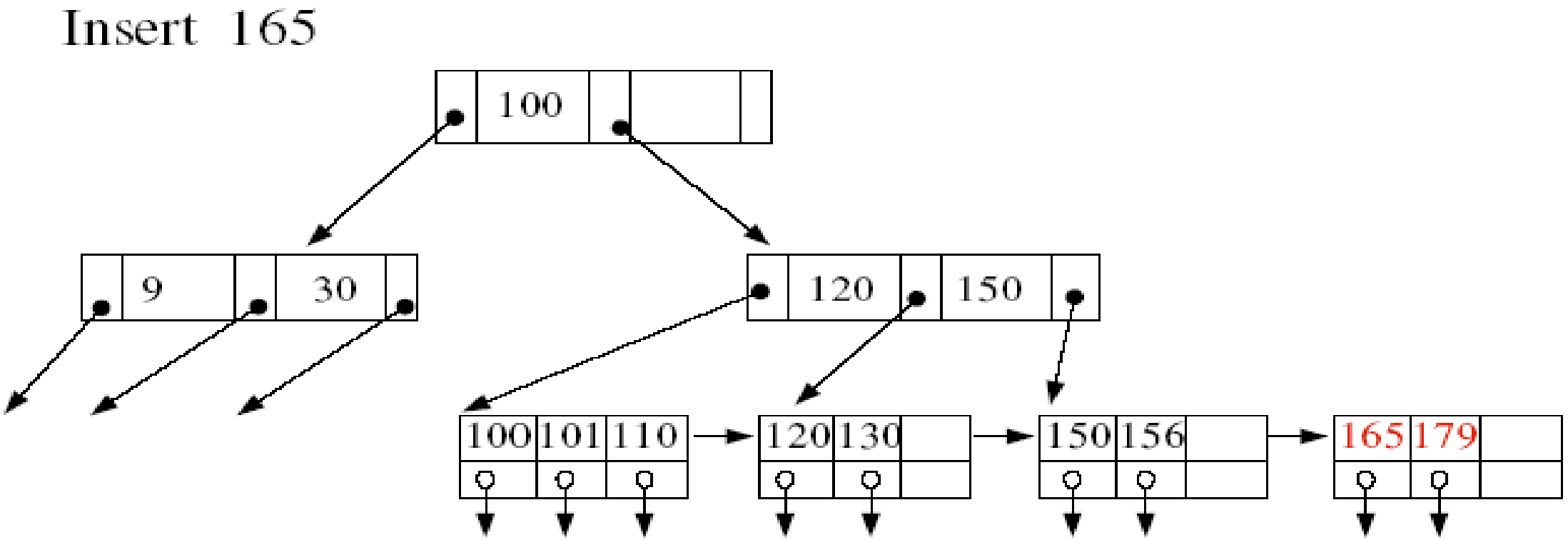
درخت B^+–درج نوع ۲



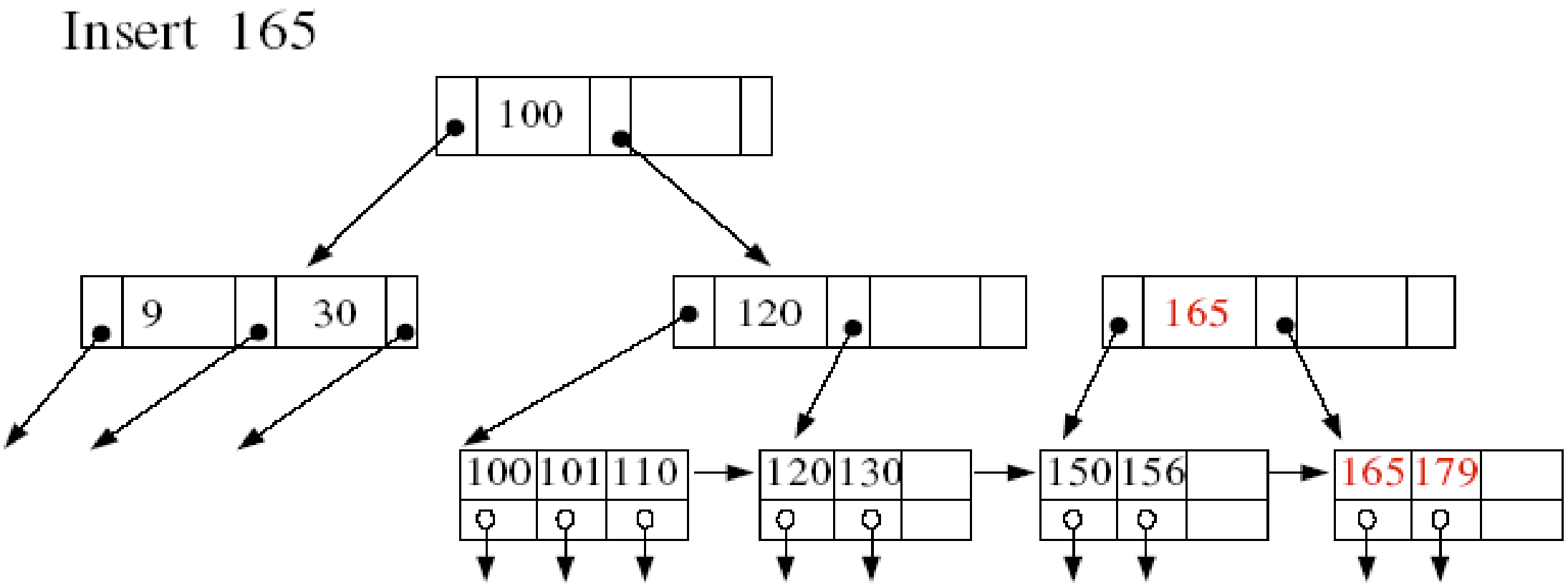
درخت B^+–درج نوع ۳



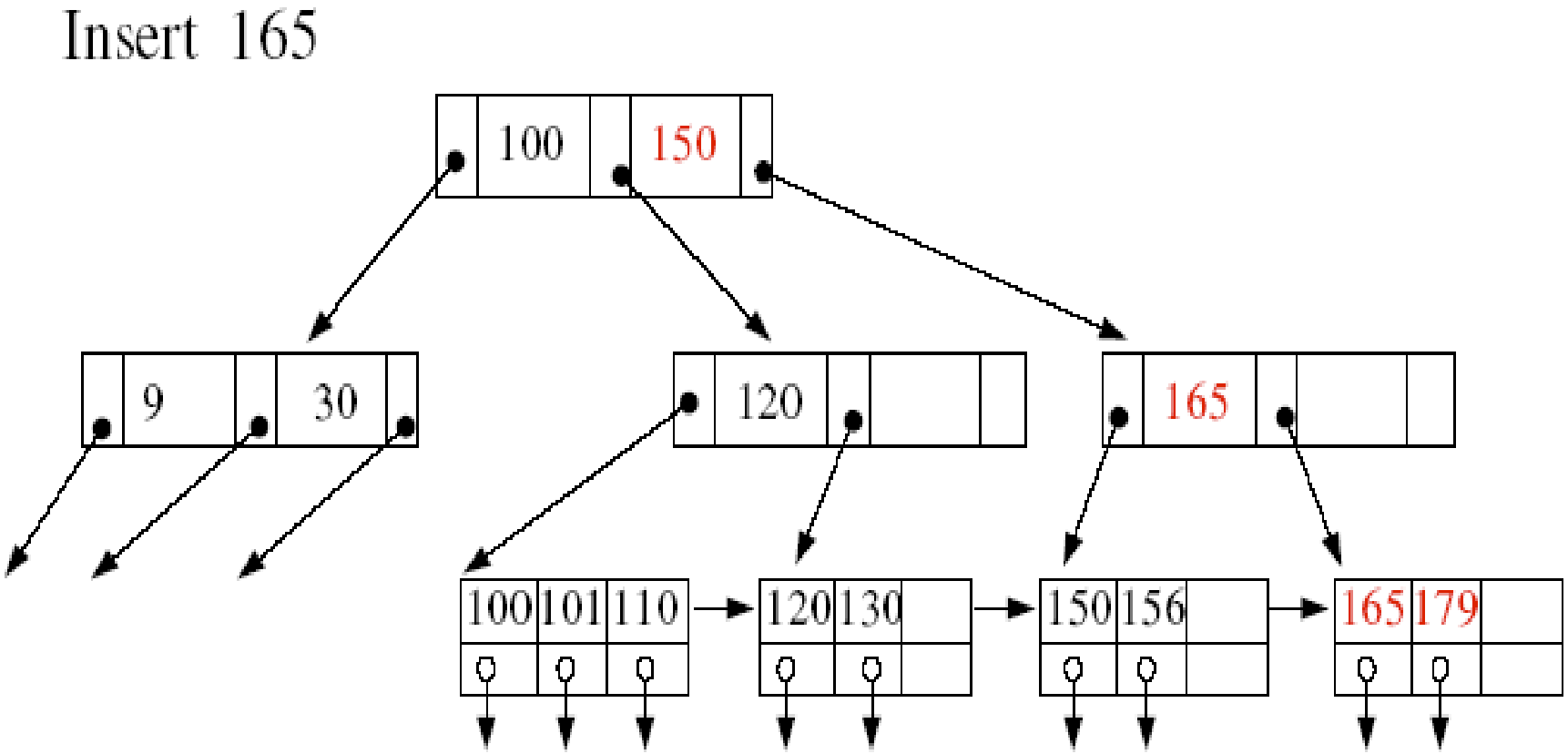
درخت B^+–درج نوع ۳



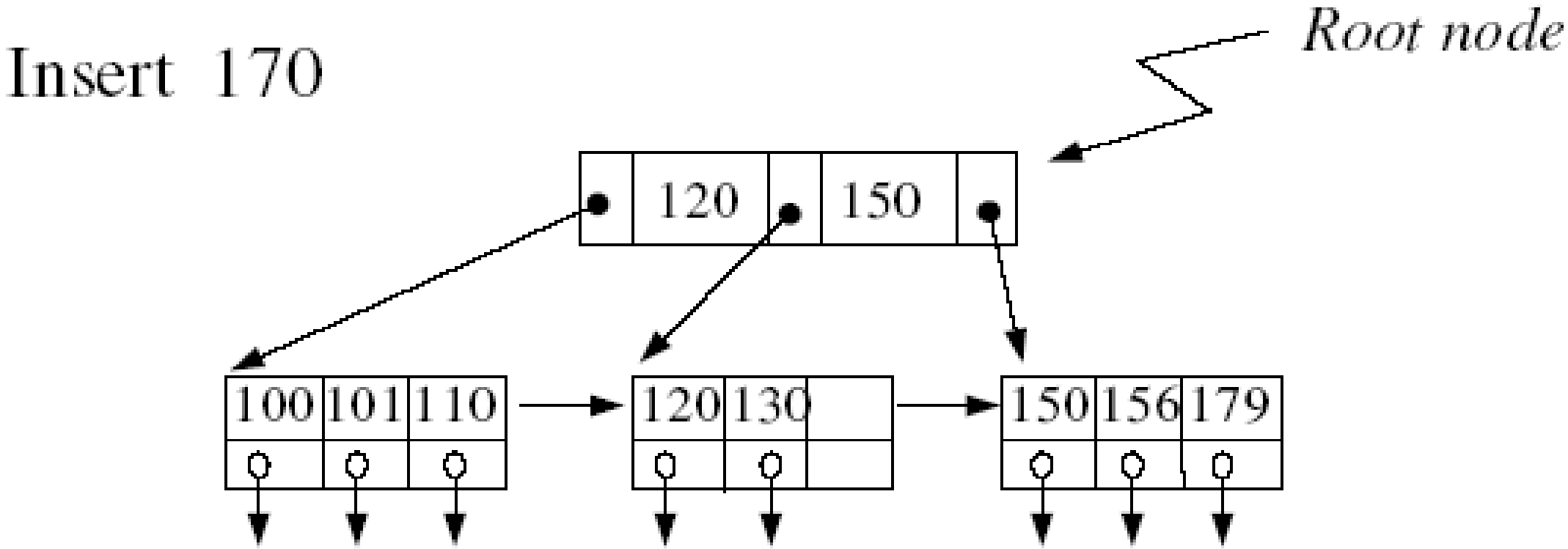
درخت B^+–درج نوع ۳



درخت B^+–درج نوع ۳

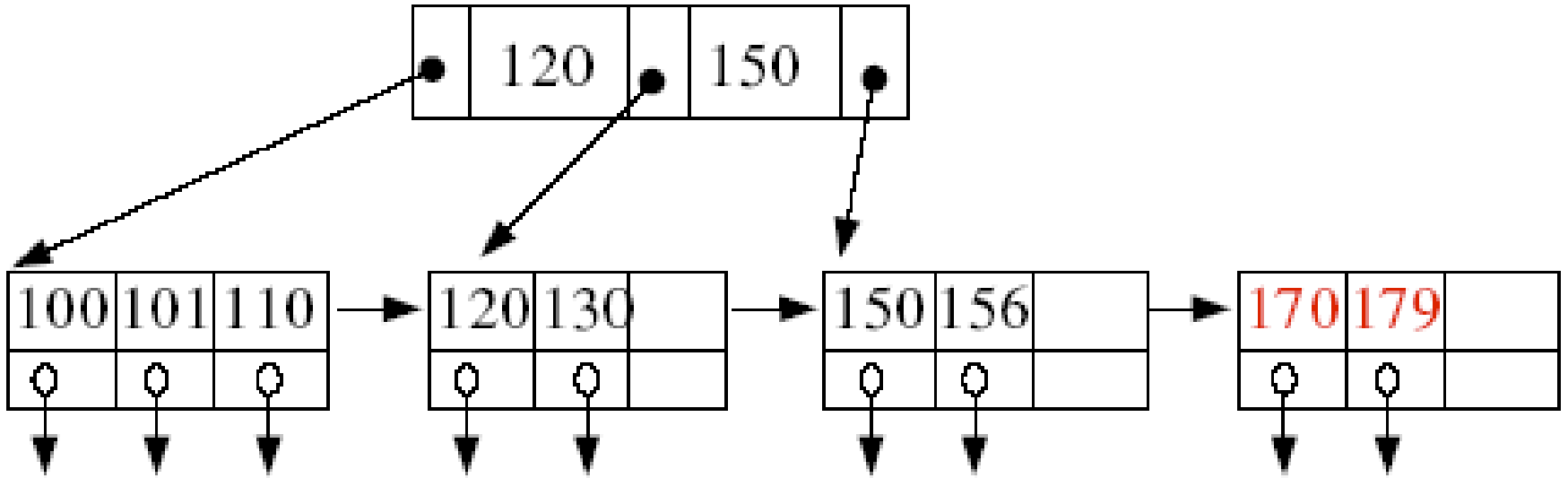


درخت B^+–درج نوع ۴



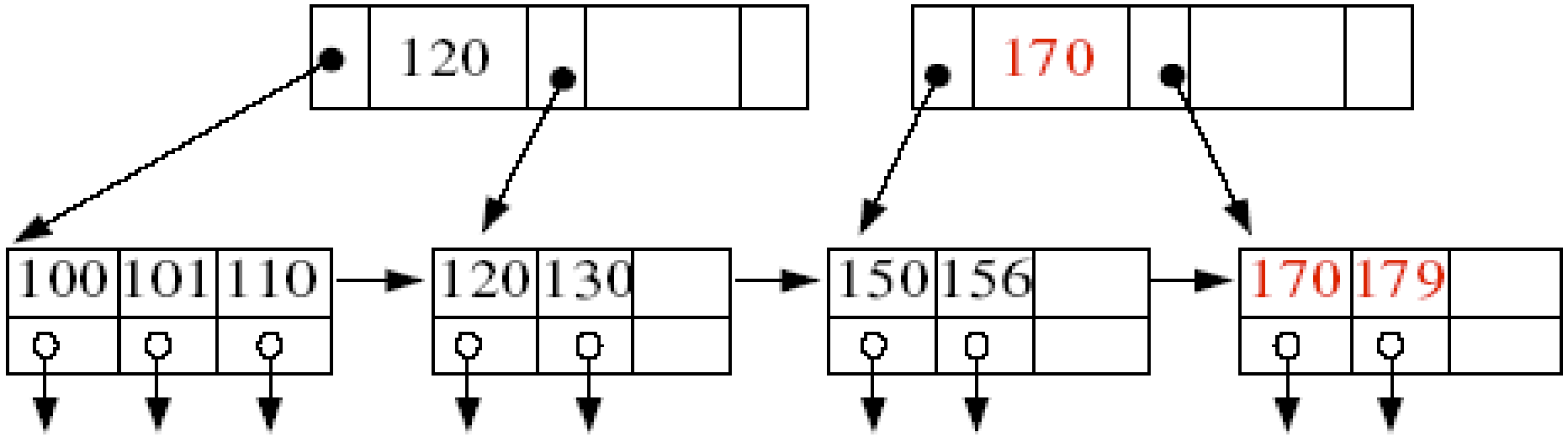
درخت B^+–درج نوع ۴

Insert 170

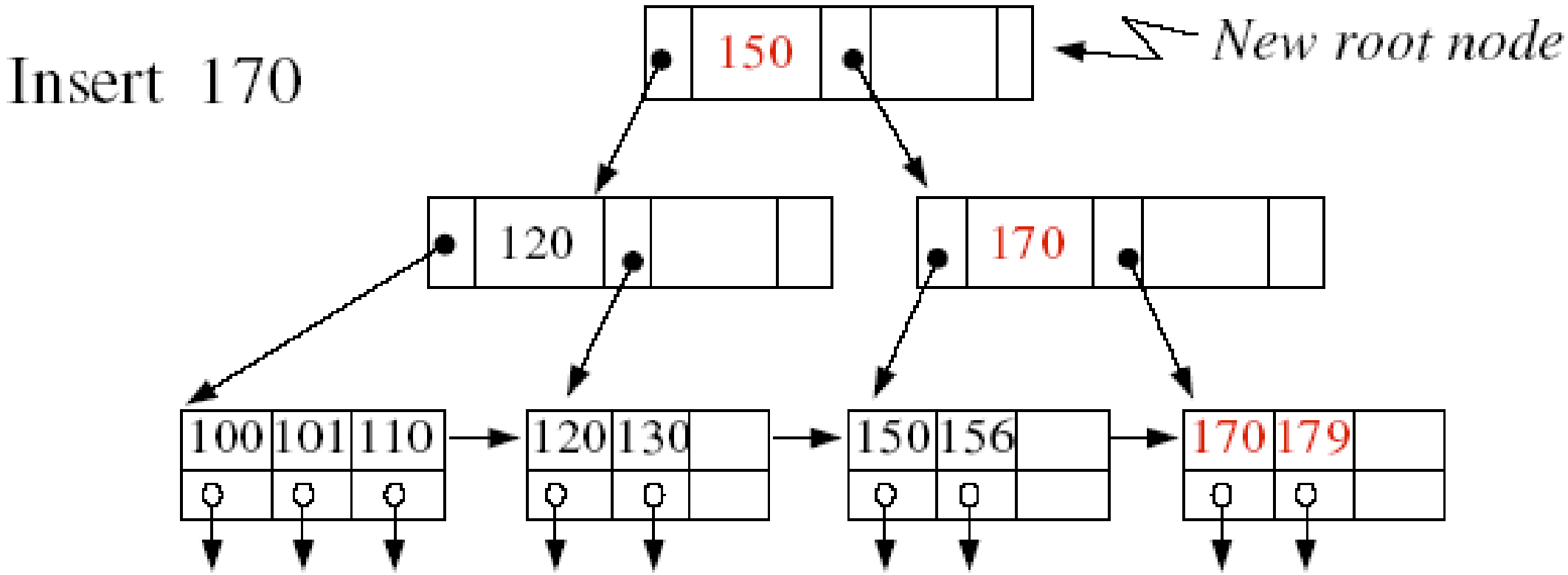


درخت B^+–درج نوع ۴

Insert 170



درخت B^+–درج نوع ۴



تاریخچه ساختارهای چندسطحی