

توری جستجوی دودویی کامل:

یک ساختمان داده موازی برای کامپیوترهای موازی با حافظه توزیع شده

حمید بیگی، محمدرضا میبیدی

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی امیرکبیر

چکیده

یک ماشین دیکشنری از پایگاه داده ای از زوج (کلید، رکورد) و مجموعه ای از عملگرها تشکیل میشود. عملگرهای درج، حذف، حذف رکورد دارای کوچکترین کلید، حذف رکورد دارای بزرگترین کلید، جستجو، رکورد دارای نزدیکترین کلید، رکورد بعد، رکورد قبل، رکورد دارای کوچکترین کلید، رکورد دارای بزرگترین کلید، تعداد کلیدهای کوچکتر و تعداد رکوردهای موجود در پایگاه داده نمونه هایی از عملگرهای ماشین های دیکشنری هستند. پیاده سازی های مختلفی از ماشین های دیکشنری برای کامپیوترهای موازی با حافظه توزیع شده ارائه شده است که زنجیره مرتب [۲۷]، مکعب متوازن [۱۰] و هیپ باین [۲۳، ۲۶] از آن جمله اند. توری جستجوی دودویی نمونه ای از یک ساختمان داده موازی برای ماشین های دیکشنری بمنظور پیاده سازی در کامپیوترهای موازی با حافظه توزیع شده با ساختار توری یا ابر مکعب میباشد. پیاده سازی توری جستجوی دودویی منظور که در [۲۴] ارائه شده است دارای اشکالاتی میباشد که میتوان به ایجاد حفره در هنگام عمل درج، از بین رفتن خاصیت توری جستجوی دودویی پس از عمل حذف، عدم توازن توری و پایین بودن توان عملیاتی اشاره نمود. اشکالاتی مانند عدم توازن در توری، تشکیل حفره در هنگام درج و از بین رفتن خاصیت توری جستجوی دودویی پس از عمل حذف، توسط ساختمان داده توری جستجوی دودویی متوازن برطرف شده است [۴]. در این مقاله توری جستجوی دودویی کامل معرفی میگردد. این ساختمان داده بمنظور حل مشکلاتی که در توری جستجوی دودویی و توری جستجوی دودویی متوازن وجود دارد ارائه گردیده است. در این ساختمان داده، سطوح مختلف توری بترتیب بر میگردند و این خود باعث افزایش کارایی میگردد. در هنگام پیاده سازی این ساختمان داده روی توری یا ابر مکعب، ماشین از نظر منطقی از دو شبکه غیر مجزا پخش عملگرها و ارسال جواب ها تشکیل میشود. پخش عملگرها و ارسال جواب ها از طریق دو شبکه مختلف سبب کم شدن ترافیک روی شبکه میگردد. مقایسه توری جستجوی دودویی کامل با طرح های دیگر گزارش شده برای کامپیوترهای موازی با حافظه توزیع شده و ساختار توری [۱۱، ۱۲، ۳۱، ۳۴] نشان میدهد که توری جستجوی دودویی کامل دارای کارایی بمراتب بالاتری میباشد.

کلمات کلیدی: ساختمان داده موازی، ماشین دیکشنری، توری جستجوی دودویی، کامپیوترهای موازی

۱- مقدمه

پایگاه داده موجود باشد در اینصورت گفته میشود که کلید k در دیکشنری ذخیره شده است. فرض کنید $F(k) = \{(k, r) \mid (k, r) \in F\}$. بنابراین در صورت وجود کلید k در دیکشنری، $F(k)$ یک مجموعه تک عضوی و در غیر اینصورت یک مجموعه تهی میباشد. شرح تعدادی از عملگرهای دیکشنری بصورت زیر است.

Insert (k, r) :

$$F \leftarrow (F - F(k)) \cup \{(k, r)\}$$

Delete (k) :

$$F \leftarrow F - F(k)$$

Min (k) :

$$\text{return } F(k_{\min})$$

یک ماشین دیکشنری از پایگاه داده ای از زوج (کلید، رکورد) و مجموعه ای از عملگرها تشکیل میشود. عملگرهای درج (Insert)، حذف (Delete)، حذف رکورد دارای کوچکترین کلید (XMin)، حذف رکورد دارای بزرگترین کلید (XMax)، جستجو (Search)، نزدیکترین کلید (Near)، رکورد بعد (Next)، رکورد قبل (Prev)، رکورد دارای کوچکترین کلید (Min)، رکورد دارای بزرگترین کلید (Max)، تعداد کلیدهای کوچکتر (CountLess) و تعداد رکوردهای موجود (Count) در پایگاه داده نمونه هایی از عملگرهای ماشینهای دیکشنری هستند. فرض کنید F نشاندهنده مجموعه زوج های (k, r) با کلید k و رکورد r موجود در پایگاه داده باشد. اگر بتوان برای کلید k ، رکورد r را پیدا نمود که زوج (k, r) در

می‌باشند. مکعب متوازن^۸ [۱۰]، زنجیره مرتب^۹ [۲۷] و هیپ بانین^{۱۰} [۲۶، ۲۳] نمونه‌هایی از ساختمان داده موازی برای کامپیوترهای موازی با حافظه توزیع شده و ساختار ابر مکعب می‌باشند. کارایی یک ماشین دیکشنری با توجه به معیارهای زیر مورد بررسی قرار می‌گیرد.

فاصله شروع دو عملگر^{۱۱}: حداقل فاصله زمانی که بین شروع اجرای دو عملگر متفاوت نیاز می‌باشد را فاصله شروع دو عملگر می‌نامند.

زمان پاسخ^{۱۲}: زمان بین شروع و اتمام یک عملگر را زمان پاسخ عملگر می‌نامند.

توان عملیاتی^{۱۳}: تعداد عملیات موازی انجام شده در واحد زمان را توان عملیاتی ماشین دیکشنری می‌نامند.

در ادامه سه ماشین دیکشنری که روی کامپیوترهای موازی با حافظه توزیع شده با ساختار توری^{۱۴} پیاده‌سازی شده اند شرح داده می‌شود. در پایان این مقاله کارایی توری جستجوی دودویی کامل که در این مقاله پیشنهاد شده است با این سه ماشین که در زیر بطور مختصر شرح داده شده اند مقایسه خواهد شد.

ماشین Schmeck و Schroder: ساختار این ماشین بصورت یک توری $n \times n$ است که در آن هر پردازنده به چهار پردازنده همسایه‌اش متصل شده است. گمانه‌های این ماشین به دو دسته تقسیم می‌شوند. یک دسته از گمانه‌ها برای انتشار^{۱۵} عملگرها و جمع‌آوری نتایج و دسته دیگر برای ارتباط بین پردازنده‌های همسایه بکار می‌روند. در این ماشین عملگرها بطور همزمان به تمام پردازنده‌های سطر اول ارسال می‌شوند بطوریکه پس از n واحد زمان، عملگرها به سطر آخر می‌رسند و از ترکیب نتایج تولید شده در سطر آخر نتیجه عملگر حاصل می‌شود. این ماشین دارای فاصله شروع دو عملگر $O(1)$ و زمان پاسخ $O(n)$ می‌باشد [۳۱].

ماشین Santoro و Dehne: ساختار این ماشین بصورت یک توری $n \times n$ است که در آن هر پردازنده به چهار یا هشت پردازنده همسایه‌اش متصل شده است. از نظر منطقی این ماشین از دو شبکه مارپیچی^{۱۶} و شبکه انتشار تشکیل شده است که هر دو در توری جا داده می‌شوند و بطور همزمان عمل می‌کنند. شبکه مارپیچی بصورتی در توری جا داده می‌شود که هر پردازنده فقط یک بار در آن ظاهر می‌شود. رکوردها بصورت نزولی (از جهت پردازنده ورودی-خروجی)^{۱۷} در این شبکه ذخیره می‌شوند. بطوریکه رکورد یا بزرگترین کلید در پردازنده ورودی-خروجی قرار دارد. شبکه انتشار برای عملگر جستجو بکار می‌رود. این شبکه‌ها بصورت مجزا یا غیر مجزا روی توری جا داده می‌شوند و یک گراف بدون حلقه را تولید می‌کنند. این ماشین دارای فاصله شروع دو عملگر $O(1)$ و زمان پاسخ $O(n)$ می‌باشد [۱۱، ۱۲].

ماشین Young و Youn: ساختار این ماشین بصورت یک توری $n \times n$ است که در آن هر پردازنده به شش پردازنده همسایه‌اش متصل می‌باشد. از نظر منطقی این ماشین از دو شبکه خطی^{۱۸} و شبکه پوشا^{۱۹} تشکیل شده است که هر دو در توری جا داده می‌شوند. شبکه خطی برای عملگر درج و شبکه پوشا برای عملگر جستجو بکار می‌رود. این ماشین دارای فاصله شروع دو عملگر $O(1)$ و زمان پاسخ $O(n)$ می‌باشد [۳۴].

توری جستجوی دودویی^{۲۰}، یک نمونه از ساختمان داده‌های موازی روی کامپیوترهای موازی با حافظه توزیع شده و ساختار توری یا ابر مکعب می‌باشد که برای اولین بار در [۲۴] پیشنهاد شده است. پیاده‌سازی این ساختمان داده روی توری یا ابر مکعب آنطور که در [۲۴] ارائه شده است دارای اشکالاتی می‌باشد که می‌توان به ایجاد حفره در هنگام عمل درج، از بین رفتن خاصیت توری جستجوی دودویی پس از عمل حذف، عدم توازن توری و پایین بودن توان عملیاتی آن اشاره نمود. عدم توازن توری سبب افزایش زمان پاسخ می‌گردد. برای کاهش زمان پاسخ، ساختمان داده توری جستجوی دودویی متوازن^{۲۱} پیشنهاد شده است [۴] که در آن تعداد رکوردهای ذخیره شده در تمام پردازنده‌های نیم توری تقریباً برابر

Max (k) :
return $F(k_{max})$

XMin (k) :
 $F \leftarrow F - F(k_{min})$
return $F(k_{min})$

XMax (k) :
 $F \leftarrow F - F(k_{max})$
return $F(k_{max})$

Search (k) :
if $k \in F$ then return $F(k)$
else return

Next (k) :
if $Succ(k) \in F$ then return $F(Succ(k))$
else return

Prev (k) :
if $Pred(k) \in F$ then return $F(Pred(k))$
else return

Near (k) :
return $F(X_{near})$ where k_{near} is the stored key closest to k

CountLess (k) :
return number of keys x where $x < k$

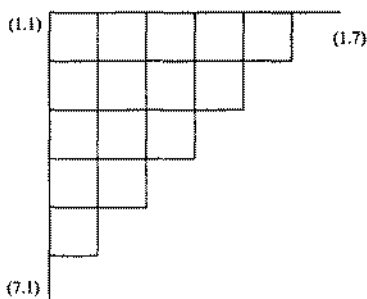
Count () :
return number of keys k where $k \in F$

اگر کلید k قبل از عمل درج در دیکشنری موجود باشد، عمل درج را زائد^{۲۲} و اگر قبل از عمل حذف در دیکشنری موجود نباشد، عمل حذف را زائد می‌گویند. برای سهولت ارائه در این مقاله، یک زوج توسط کلید آن نمایش داده می‌شود همچنین فرض می‌شود که عملگرهای درج و حذف زائد نباشند.

عملگرهای ماشین دیکشنری را می‌توان به دو دسته عملگرهای اصلاحی^{۲۳} و عملگرهای پرس وجو^{۲۴} تقسیم نمود. عملگرهای اصلاحی محتوای ماشین دیکشنری را تغییر می‌دهند در حالی که عملگرهای پرس وجو محتوای ماشین را تغییر نمی‌دهند. عملگرهای درج، حذف و حذف رکورد دارای کوچکترین کلید نمونه‌هایی از عملگرهای اصلاحی و عملگرهای جستجو، رکورد بعد و رکورد دارای کوچکترین کلید نمونه‌هایی از عملگرهای پرس وجو می‌باشند.

برای پیاده‌سازی ماشین‌های دیکشنری سه روش گزارش شده است که عبارتند از: الگوریتم‌های ترتیبی برای پیاده‌سازی روی کامپیوترهای تک‌پردازنده [۱، ۲۶، ۳۱]، الگوریتم‌های موازی برای پیاده‌سازی روی سخت‌افزار [۳، ۶، ۱۲، ۱۵، ۲۱، ۲۵، ۳۱، ۳۴]، و الگوریتم‌های موازی برای پیاده‌سازی روی کامپیوترهای موازی [۲، ۷، ۱۲، ۱۴، ۱۶، ۱۷، ۲۵]. الگوریتم‌های موجود برای پیاده‌سازی ماشین‌های دیکشنری روی کامپیوترهای موازی به دو دسته، الگوریتم‌های موازی برای پیاده‌سازی روی کامپیوترهای موازی با حافظه مشترک و الگوریتم‌های موازی برای پیاده‌سازی روی کامپیوترهای موازی با حافظه توزیع شده تقسیم می‌شوند. در طراحی الگوریتم‌های موازی برای پیاده‌سازی روی کامپیوترهای موازی با حافظه مشترک، توجه به تداخل فرایند‌های موازی در دسترسی به ساختمان داده مشترک می‌باشد در حالی که در طراحی الگوریتم‌های موازی برای پیاده‌سازی روی کامپیوترهای موازی با حافظه توزیع شده، توجه به استخراج توازی در ساختمان داده است. عملگرهای موازی برای درخت‌های دودویی [۱۷]، درخت‌های متوازن [۱۹]، عملگرهای درج و جستجو در درخت‌های AVL و درخت‌های ۲-۳ [۱۳، ۱۴]، عملگرهای موازی درج و حذف در هیپ [۲۹] نمونه‌هایی از الگوریتم‌های موازی ارائه شده برای کامپیوترهای موازی با حافظه مشترک

$(I, J+1)$ و فرزند سمت چپ $(I+1, J)$ می باشد. شماره سطح یک گره، عمق آن گره و بیشترین عمق در بین گره های غیر تهی، عمق توری نامیده میشود.



شکل ۱- نیم توری دو بعدی 7×7

توری جستجوی دودوئی: توری جستجوی دودوئی یک نیم توری دو بعدی است که دارای شرایط زیر میباشد.

- ۱- هر گره غیر برگ میتواند دارای صفر یا یک رکورد باشد.
- ۲- هر گره برگ میتواند دارای صفر یا بیشتر رکورد باشد.
- ۳- کلید رکورد ذخیره شده در گره پدر از کلید رکورد ذخیره شده در گره فرزند سمت چپ بزرگتر و از کلید رکورد ذخیره شده در گره فرزند سمت راست کوچکتر است.

پیاده سازی توری جستجوی دودوئی برای کامپیوترهای موازی با حافظه توزیع شده و ساختار اسر مکعب در [۲۴] آمده است. الگوریتم های ارائه شده برای این ساختمان داده دارای اشکالاتی بوده است. یکی از این اشکالات بوجود آمدن حفره (یک گره میانی خالی که دارای دو پدر و فرزند سمت راست پر باشد) در عملگر درج میباشد که سبب از بین رفتن خواص توری جستجوی دودوئی میگردد. اشکال دیگر عدم برقراری خواص توری جستجوی دودوئی و تشکیل حفره پس از عمل حذف میباشد. اشکالات فوق بصورت زیر برطرف شده اند [۴]. برای از بین بردن حفره دوروش پیشنهاد شده است. در روش اول هنگام درج رکورد x ، تشکیل یا عدم تشکیل حفره بررسی میشود. در صورت تشکیل حفره با جایگزینی رکورد پدر سمت راست حفره توسط رکورد x و درج رکورد پدر سمت راست حفره در توری جستجوی دودوئی از تشکیل حفره جلوگیری میشود. روش دوم برای از بین رفتن حفره استفاده از توری جستجوی دودوئی متوازن میباشد. توازن در توری جستجوی دودوئی سبب عدم تشکیل حفره میگردد.

عدم برقراری خواص توری جستجوی دودوئی و تشکیل حفره در هنگام حذف رکورد x بصورت زیر برطرف شده است. برای حذف رکورد x ابتدا گره حاوی x پیدا میشود. اگر گره پیدا شده برگ یا گره میانی بدون فرزند یا گره مرزی بدون فرزند باشد، رکورد x حذف میشود. اگر گره پیدا شده یک گره مرزی باشد از برگ تا گره پیدا شده شیفت داده میشوند تا حفره ایجاد شده پر گردد. اگر گره پیدا شده دارای یک فرزند باشد و رکورد فرزند موجود شرایط انتقال به گره پدر را داشته باشد، رکورد فرزند جایگزین رکورد پدر میشود و در غیر اینصورت رکورد یکی از والدین که شرایط جایگزینی را داشته باشد به این گره منتقل می شود و عملگر حذف به گرهی منتقل می گردد که رکورد آن انتقال یافته است. در صورتیکه گره پیدا شده دارای دو فرزند باشد در صورت برقرار بودن شرایط جایگزینی، حفره ایجاد شده توسط رکورد یکی از فرزندان و در غیر اینصورت توسط رکورد یکی از والدین پر می شود و عملگر حذف به گره واجد شرایط منتقل می شود. جایگزینی و انتقال تا زمانی ادامه پیدا می کند که به گره برگ، گره مرزی یا گره میانی بدون فرزند برسیم.

در توری جستجوی دودوئی میتوان حداکثر یک رکورد در گره های غیر برگ و چندین رکورد را در گره های برگ ذخیره نمود. بهمین دلیل بار روی گره های برگ

میباشد. در توری جستجوی دودوئی $n \times n$ و توری جستجوی دودوئی متوازن $n \times n$ فاصله زمانی شروع دو عملگر مختلف $O(n)$ میباشد و بهمین دلیل دارای توان عملیاتی بالایی نیستند. در این مقاله یک ساختمان داده موازی جدید بنام توری جستجوی دودوئی کامل^{۲۲} معرفی میگردد. این ساختمان داده که مشابه توری جستجوی دودوئی است دارای توان عملیاتی بالاتری میباشد. در هنگام پیاده سازی این ساختمان داده روی توری یا اسر مکعب، ماشین از نظر منطقی از دو شبکه غیر مجزا پخش عملگرها و ارسال جوابها تشکیل میشود. شبکه پخش عملگرها مسیری یکتا بین ریشه و گره های دیگر توری جستجوی دودوئی کامل ایجاد میکند که باعث میگردد که عملگرهای ماشین دیکشنری بصورت متوالی وارد پردازنده ها گردند. این باعث میشود تا سازگاری در توری جستجوی دودوئی کامل بسادگی قابل پیاده سازی باشد. در توری جستجوی دودوئی کامل، عملگرها دارای زمان پاسخ از مرتبه $O(n)$ و فاصله زمانی بین شروع دو عملگر از مرتبه $O(1)$ میباشد.

بخش های بعدی مقاله بصورت زیر سازماندهی شده است. ساختمان داده توری جستجوی دودوئی کامل در بخش ۲ معرفی شده است. پیاده سازی عملگرهای این ساختمان داده و سازگاری آن در بخش های ۳ و ۴ بررسی گردیده است. مقایسه ای بین توری جستجوی دودوئی کامل و چند ماشین دیکشنری گزارش شده [۱۱، ۱۲، ۲۱، ۲۴] برای کامپیوترهای موازی با ساختار توری در نتیجه گیری آمده است.

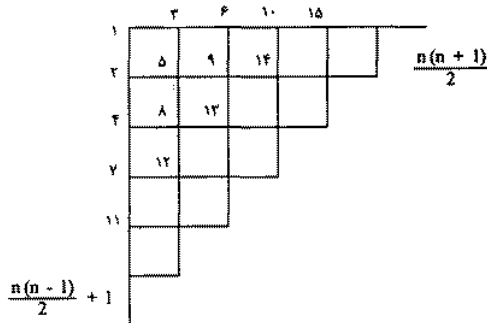
۲- توری جستجوی دودوئی کامل

در این قسمت در ابتدا تعاریف توری d - بعدی و نیم توری دو بعدی داده شده است و سپس توری جستجوی دودوئی و توری جستجوی دودوئی متوازن تعریف و اشکالات موجود در آنها بررسی گردیده است و نهایتاً تعریف توری جستجوی دودوئی کامل که پیاده سازی آن در این مقاله آمده است داده شده است.

توری d - بعدی: یک توری d - بعدی از اجتماع گره هایی تشکیل میشود که در طول نقاط فضای d - بعدی قرار گرفته اند و کمائی بین هر گره و نزدیکترین همسایه هایش وجود دارد. گره های یک توری d - بعدی با n_i نقطه در طول بعد i ام بوسیله d - تایی (x_1, x_2, \dots, x_d) نمایش داده میشوند بطوریکه هر یک از مختصات x_i (برای $i = 1, 2, \dots, d$) میتواند مقادیر صحیح ۱ تا n_i را اختیار نمایند. دو گره $x = (x_1, x_2, \dots, x_d)$ و $y = (y_1, y_2, \dots, y_d)$ همسایه یکدیگر گفته میشود اگر یک i ($1 \leq i \leq d$) وجود داشته باشد بطوریکه برای تمام i های مخالف j داشته باشیم $|x_i - y_j| = 1$.

نیم توری دو بعدی: یک نیم توری دو بعدی $n \times n$ (که به اختصار نیم توری $n \times n$ نامیده میشود) از تقسیم توری دو بعدی $n \times n$ در راستای قطر $\{(1,n), (2,n-1), \dots, (n-1,2), (n,1)\}$ بدست میاید. شکل ۱ یک نیم توری دو بعدی 7×7 را نشان میدهد. در این نیم توری، گره $(1,1)$ ریشه، گره های روی قطر $N_L = \{(i,j) | i+j=n+1\}$ برگ، گره های روی اضلاع $N_{LB} = \{(k,1) | 1 < k < n\}$ و $N_{TB} = \{(1,k) | 1 \leq k < n\}$ (بترتیب مرز بالا و مرز چپ) و بقیه گره ها، گره های میانی نامیده میشوند. نیم توری $n \times n$ دارای n سطح میباشد که بترتیب از سطح ۱ الی n شماره گذاری میشوند. ریشه نیم توری در سطح ۱ قرار دارد. در سطح k ام نیم توری k گره $L_k = \{(i,j) | i+j=k+1\}$ وجود دارند که برادر یکدیگر نامیده میشوند. گره $(k,1)$ را برادر کوچکتر (گره مرز چپ) و گره $(1,k)$ برادر بزرگتر (گره مرز بالا) نامیده میشوند. برگ های نیم توری در سطح n ام نیم توری قرار دارند. گره میانی $(1,1)$ دارای دو پدر راست $(1-1, J)$ و پدر چپ $(I, J-1)$ ، گره مرز بالا $(1, J)$ دارای پدر سمت چپ $(I, J-1)$ ، گره مرز چپ $(I, 1)$ دارای پدر سمت راست $(I-1, 1)$ و گره ریشه بدون پدر می باشد. گره غیر برگ (I, J) دارای دو فرزند سمت راست

فرض کنید که گره های نیم توری دوبعدی $n \times n$ با عمق m که در آن N رکورد ذخیره شده باشد را بصورت شکل ۲ شماره گذاری نماییم در اینصورت برای یک توری جستجوی کامل قضایای زیر صادق می باشد.



شکل ۲- شماره گذاری نیم توری دو بعدی

قضیه ۱: اگر $N \leq n(n-1)$ و $m^2 \geq N$ باشد گره های با شماره $\frac{m(m-1)}{2} + 1, \dots, N, \frac{m(m-1)}{2}$ پر و گره های با شماره $\frac{m(m-1)}{2} + 1, \dots, N, \frac{m(m-1)}{2}$ نیمه پر هستند.

قضیه ۲: اگر $n(n-1) \leq N \leq m^2$ باشد گره های با شماره $1, \dots, N, \frac{m(m+1)}{2}$ پر و گره های با شماره $\frac{m(m+1)}{2} + 1, \dots, N, \frac{m(m+1)}{2}$ نیمه پر هستند.

قضیه ۳: اگر $N > n(n-1)$ باشد در این صورت گره های برگ با شماره $1, \dots, \frac{n(n-1)}{2} + K$ پر، برگ های با شماره $\frac{n(n-1)}{2} + 1, \dots, \frac{n(n-1)}{2} + K$ دارای $\left\lfloor \frac{N-n(n-1)}{n} \right\rfloor$ رکورد و برگ های با شماره $\frac{n(n-1)}{2} + K + 1, \dots, \frac{n(n+1)}{2}$ دارای $\left\lfloor \frac{N-n(n-1)}{n} \right\rfloor$ رکورد هستند که K برابر است با $\text{mod}(N-n(n-1), n)$. اثبات قضایای فوق در [۵] آورده شده است.

فرض کنید که برای $1 \leq k \leq n-k$ مجموعه های $R_k(I) = \{(I, J) \mid k < J \leq n\}$ و $L_k = \{(I, k) \mid k < I \leq n\}$ بصورت شکل ۳ تعریف شده باشند.

گره مرز بالا (I, k) را کامل میگوئیم اگر

۱- گره (I, k) خالی یا نیمه پر باشد یا

۲- تفاوت تعداد رکوردهای موجود در مجموعه های $R_k(I) = \{(I, J) \mid k < J \leq n\}$ و $L_k = \{(I, k) \mid k < I \leq n\}$ برابر صفر یا ۱- باشد و گره مرز بالا $(I, k+1)$ کامل باشد.

نیم توری دو بعدی $n \times n$ با ریشه q را کامل میگوئیم اگر

۱- ریشه کامل باشد و

۲- نیم توری دو بعدی $(n-1) \times (n-1)$ باریشه فرزند سمت راست q کامل باشد.

قضیه ۴: اگر در یک توری جستجوی دودویی کامل N رکورد ذخیره شده باشد

$$\text{عمق توری} = \min \left(n, \left\lfloor \frac{\sqrt{1+4N}-1}{2} \right\rfloor \right)$$

اثبات: اثبات در دو قسمت انجام میگیرد

بیشتر می باشد که منجر به پایین آمدن توان عملیاتی ساختمان داده می گردد. مشکل دیگری که در توری جستجوی دودویی وجود دارد رشد یک بعدی توری (مانند آنچه در درخت جستجوی دودویی وجود دارد) می باشد. اگر داده های درج شونده بترتیب صعودی (نزولی) وارد توری شوند تنها در گره های مرز بالا (مرز چپ) توری قرار می گیرند و بقیه گره ها خالی می ماند. رشد توری در یک بعد و عدم توازن تعداد رکوردها در گره های توری نیز سبب کاهش توان عملیاتی این ساختمان داده می گردد. برای رفع این اشکالات، توری جستجوی دودویی متوازن که تعریف آن در زیر آمده است پیشنهاد شده است [۴]. در این ساختمان داده رکوردها بین گره های توری بصورت متوازن توزیع می شوند تا از این طریق توان عملیاتی ساختمان داده افزایش یابد.

تسوری جستجوی دودویی متوازن: توری جستجوی دودویی متوازن یک نیم توری دوبعدی است که دارای شرایط زیر باشد.

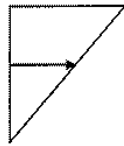
- ۱- هر گره می تواند دارای صفر یا بیشتر رکورد باشد.
- ۲- کلید رکوردهای ذخیره شده در گره پدر از کلید رکوردهای ذخیره شده در گره فرزند سمت چپ بزرگتر و از کلید رکوردهای ذخیره شده در گره فرزند سمت راست کوچکتر است.
- ۳- قدر مطلق تفاوت تعداد رکوردهای ذخیره شده در هر دو گره موجود در توری حداکثر یک است.

- ۴- یک گره میانی خالی نمی تواند فرزند پر داشته باشد.
- ۵- یک گره مرز چپ خالی می تواند فرزند سمت راست پر داشته باشد ولی نمی تواند فرزند سمت چپ پر داشته باشد.
- ۶- یک گره مرز بالا خالی می تواند فرزند سمت چپ پر داشته باشد ولی فرزند سمت راست پر داشته باشد.

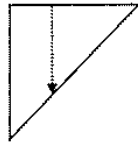
در تسوری جستجوی دودویی و توری جستجوی دودویی متوازن $n \times n$ فاصله زمانی شروع دو عملگر مختلف از مرتبه $O(n)$ می باشد [۴] که این خود باعث پایین آمدن توان عملیاتی این دو ساختمان داده می گردد. هدف از معرفی توری جستجوی دودویی کامل حل مشکل پایین بودن توان عملیاتی در توری جستجوی دودویی و توری جستجوی دودویی متوازن می باشد.

توری جستجوی دودویی کامل: توری جستجوی دودویی کامل $n \times n$ (که به اختصار توری جستجوی دودویی کامل نامیده می شود) یک نیم توری دوبعدی است که شرایط زیر را دارا می باشد.

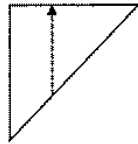
- ۱- هر گره غیر برگ می تواند دارای صفر (گره خالی)، یک (گره نیمه پر) یا دو (گره پر) رکورد باشد.
- ۲- هر گره برگ می تواند دارای صفر یا بیشتر رکورد باشد.
- ۳- بزرگترین رکورد در هر گره از رکوردهای ذخیره شده در فرزند سمت چپ بزرگتر و از رکورد های ذخیره شده در فرزند راست کوچکتر است.
- ۴- کوچکترین رکورد در هر گره از رکوردهای ذخیره شده در فرزندان سمت چپ و سمت راست آن گره کوچکتر است.
- ۵- همه گره های نیمه پر در یک سطح قرار دارند یا عبارتی دیگر در نیم توری تنها یک سطح وجود دارد که دارای گره های نیمه پر است.
- ۶- هر گره نیمه پر و یا پر دارای والدین پر می باشد.
- ۷- یک سطح بطور همزمان نمی تواند دارای گره های خالی و پر باشد.
- ۸- یک گره نیمه پر نمی تواند دارای برادر بزرگتر پر باشد.
- ۹- یک گره خالی نمی تواند دارای برادر بزرگتر نیمه پر باشد.
- ۱۰- برای دو گره برگ a و b بطوریکه گره a برادر کوچکتر گره b باشد تعداد رکوردهای ذخیره شده در گره a مساوی یا بزرگتر از تعداد رکوردهای ذخیره شده در گره b می باشد.



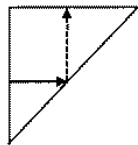
نتیجه ۲: رشته $\{S(1,J), S(2,J), \dots, S(n-J+1, J)\}$ یک رشته صعودی است.



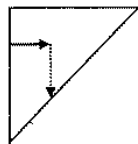
نتیجه ۳: رشته $\{L(n-J+1, J), \dots, L(2,J), L(1,J)\}$ یک رشته صعودی است.



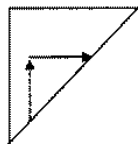
نتیجه ۴: رشته $\{I(k,1), \dots, I(k,J-1), S(k,J), L(k,J), \dots, L(1,J)\}$ یک رشته صعودی است.



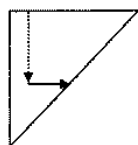
نتیجه ۵: رشته $\{I(k,1), \dots, I(k,J-1), S(k,J), \dots, S(J,n-J+1)\}$ یک رشته صعودی است.



نتیجه ۶: رشته $\{L(n-J+1,J), \dots, L(k,J), I(k,J), \dots, I(k,n-I+1)\}$ یک رشته صعودی است.

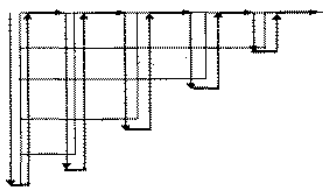


نتیجه ۷: رشته $\{S(1,J), \dots, S(k,J), I(k, J), \dots, I(k, n-k+1)\}$ یک رشته صعودی است.

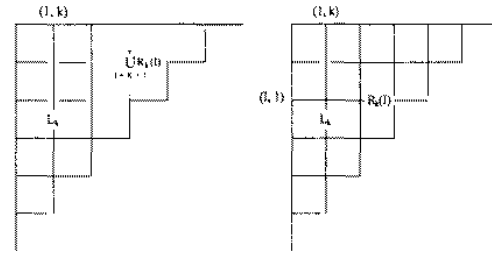


قضیه ۶: زنجیره زیر یک رشته صعودی است

$\{S(1,1), \dots, S(n,1), L(n,1), \dots, L(1,1), S(1,2), \dots, S(n,2), \dots, L(1,2), \dots, S(1,n-1), S(2,n-), L(2,n-1), L(1,n-), S(1,n), L(1,n)\}$



اثبات: با استفاده از نتایج ۱، ۲ و ۳.



شکل ۲- مجموعه های L_k و R_k

۱- تعداد رکوردهای ذخیره شده در برگها کمتر و یا مساوی دو باشد:

فرض میکنیم عمق توری جستجوی دودوئی کامل k باشد. با توجه به تعریف توری جستجوی دودوئی کامل، اگر تعداد رکوردهای ذخیره شده در توری بزرگتر از $k(k-1)$ و کوچکتر یا مساوی $k(k+1)$ باشد. پس استفاده از نامساویهای $k(k-1) < N$ و $k(k+1) \geq N$ حل آنها جوابهای مثبت زیر بدست می آیند.

$$\frac{\sqrt{1+4N}-1}{2} \leq k < \frac{\sqrt{1+4N}+1}{2}$$

بدلیل اینکه عمق توری عددی صحیح است، در نتیجه عمق برابر است با

$$k = \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil$$

۲- تعداد رکوردهای ذخیره شده در حداقل یک برگ بیش از دو رکورد باشد. در این حالت عمق توری برابر n است. در نتیجه عمق توری دارای N رکورد، برابر است با

$$\min \left(n, \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil \right)$$

و بدین ترتیب قضیه اثبات می گردد. ■

قضیه ۵: زمان پاسخ عملگر جستجو در توری جستجوی دودوئی کامل که در آن N رکورد ذخیره شده باشد برابر است با:

$$\min \left(n-1, \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil \right)$$

اثبات: با توجه به قضیه ۴، در توری جستجوی دودوئی کامل با N رکورد، تعداد مقایسه های لازم برای پیدا کردن یک رکورد برابر است با

$$= \min \left(n, \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil \right) - 1$$

$$= \min \left(n-1, \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil - 1 \right)$$

$$= \min \left(n-1, \left\lceil \frac{\sqrt{1+4N}-1}{2} \right\rceil \right)$$

و قضیه اثبات می گردد. ■

در توری جستجوی دودوئی کامل. کوچکترین و بزرگترین رکورد هر گره بترتیب توسط دو متغیر SmallItem و LargeItem ذخیره میشوند. برای گرههای دارای یک رکورد مقدار LargeItem و SmallItem یکسان میباشد.

با توجه به تعریف توری جستجوی دودوئی کامل میتوان نتایج زیر را گرفت. برای سهولت نمایش، رکورد های SmallItem (S) ، LargeItem (L) و زوج (SmallItem, LargeItem) (I) بترتیب توسط خطوط نقطه چین، خط چین و پر نشان داده میشوند.

نتیجه ۱: رشته زیر یک رشته صعودی است.

$\{S(k, 1), L(k, 1), S(k, 2), L(k, 2), \dots, S(k, n-k+1), \{L(k, n-I+1)\}$

۳- عملگرها

در این بخش چگونگی کارکرد عملگرهای درج، حذف، حذف رکورد دارای کوچکترین کلید، جستجو، رکورد دارای کلید بعد، رکورد دارای کوچکترین کلید و رکورد دارای بزرگترین کلید شرح داده میشود. برای سادگی ارائه، در ادامه بحث رکوردهای موجود در یک گره غیربرگ توسط زوج (SmallItem, LargeItem) نشان داده می شود.

توازن در گره مرز بالا (l,k) توسط متغیر Balance نشان داده می شود. اگر مقدار این متغیر مساوی صفر باشد در اینصورت این گره متوازن است و نیم توری که ریشه آن گره مرز بالا (l, k) است یک توری جستجوی دودویی کاملاً پر می باشد. در صورتی که مقدار این متغیر بزرگتر از صفر باشد تعداد رکوردهای موجود در مجموعه R_{k(l)} بیشتر از تعداد رکوردهای موجود در مجموعه L_k و در غیر اینصورت تعداد رکوردهای موجود در مجموعه L_k بیشتر از تعداد رکوردهای موجود در مجموعه R_{k(l)} میباشد. بدلیل اینکه رکوردهایی که از طریق فرزند سمت راست گره مرز بالا (l,k) درج میشوند مشخص نیستند که در چه گره ای درج میشوند بنابراین مقدار متغیر Balance قابل محاسبه نیست. قضیه زیر بر اساس تعداد رکوردهایی که از طریق فرزند سمت راست و فرزند سمت چپ درج میشوند چگونگی محاسبه مقدار متغیر Balance را بیان می کند.

قضیه ۷: فرض کنید R و L بترتیب نشان دهنده تعداد رکورد های موجود در مجموعه های $\sum_{l=k+1}^n R_k(l)$ و L_k باشد. اگر گره مرز بالا، (l,k) کامل باشد در اینصورت رابطه زیر برقرار است.

$$R = \begin{cases} \left\lceil \frac{L}{2} \right\rceil \left(\left\lfloor \frac{L}{2} \right\rfloor + 1 \right) & \text{if } L \leq 2(n-k) \\ (n-k)[L - (n-k+1)] & \text{if } L > 2(n-k) \end{cases}$$

اثبات: اثبات قضیه در دو قسمت انجام میگردد.

۱- تعداد رکورد های ذخیره شده در برگ ها کمتر یا مساوی دو باشد:

با توجه به تعریف توازن سطحی و اینکه حداکثر دو رکورد در هر گره غیر برگ میتواند ذخیره شود تعداد گره های نیمه پر و پر در مجموعه های R_{k(l)} و L_k برابر است با $n_L = \left\lceil \frac{L}{2} \right\rceil$. با توجه به تعریف توری جستجوی کامل، R برابر است با

$$R = \sum_{m=0}^{n_L-1} (L - 2m) = \left\lceil \frac{L}{2} \right\rceil \left(\left\lfloor \frac{L}{2} \right\rfloor + 1 \right)$$

۲- تعداد رکورد های ذخیره شده در برگ ها بیش از دو باشد:

با توجه به تعریف نیم توری دو بعدی n_xn_x، (n-k) گره برگ در مجموعه $\sum_{l=k+1}^n R_k(l)$ و (n-k-1) گره غیر برگ در مجموعه L_k وجود دارند. با توجه به اینکه (n-k-1) گره غیر برگ موجود در مجموعه L_k پر (دارای دو رکورد) هستند با استفاده از قسمت ۱ همین قضیه، تعداد رکوردهای موجود در گره های غیر برگ مجموعه R برابر است با

$$R_1 = (n-k-1)(n-k)$$

و L-2(n-k-1) رکورد در برگ (n-k+1, k) ذخیره شده است. با توجه به تعریف

توازن سطحی، تعداد رکورد های موجود در گره های برگ مجموعه $\sum_{l=k+1}^n R_k(l)$ (n-k) برابر تعداد رکوردهای موجود در برگ (n-k+1, k) میباشد. در نتیجه تعداد رکوردهای موجود در برگهای مجموعه برابر است با

$$R_2 = [L-2(n-k-1)](n-k)$$

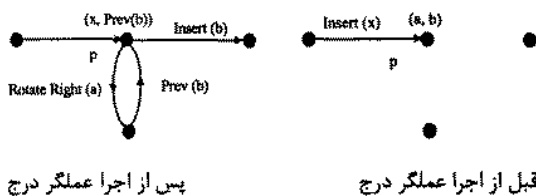
بنابراین تعداد رکورد های موجود در مجموعه $\sum_{l=k+1}^n R_k(l)$ برابر است با $R = R_1 + R_2 = [L - (n-k-1)](n-k)$ و بدین ترتیب قضیه اثبات میگردد. ■

عملگر درج: این عملگر علاوه بر درج یک رکورد وظیفه حفظ خاصیت توری جستجوی دودویی کامل را بر عهده دارد. این عملگر در حین درج یک رکورد در توری با ایجاد چرخش های لازم توازن در توری جستجوی دودویی را برقرار مینماید. عملکرد این عملگر برای گره های مرز بالا و گره های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع گره بصورت زیر می باشد.

الف- اگر گره q یک گره مرز بالا باشد، عملگر درج رکورد x را از پدر سمت چپ خود دریافت میکند. در صورتیکه گره q خالی یا نیمه پر باشد رکورد x در این گره قرار می گیرد. در صورتیکه گره q دارای دو رکورد باشد متغیر Balance مشخص می کند که رکورد x، باید از طریق کدام فرزند درج شود. اگر مقدار متغیر Balance کوچکتر از صفر باشد رکورد x از طریق فرزند سمت راست درج میشود. در فرایند درج رکورد x، از طریق فرزند سمت راست امکان برهم خوردن خاصیت توری جستجوی دودویی کامل وجود دارد. برای برقرار نمودن خاصیت توری جستجوی دودویی کامل یکی از روشهای زیر می تواند مورد استفاده قرار گیرد.

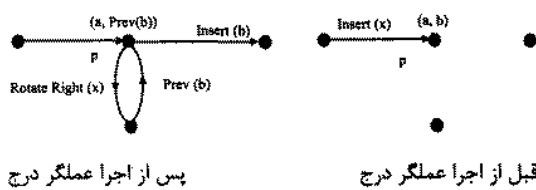
۱- اگر رکورد x، از رکورد LargeItem بزرگتر باشد، رکورد x، از طریق فرزند سمت راست درج میشود.

۲- اگر رکورد x، از رکورد SmallItem کوچکتر باشد، رکورد LargeItem از طریق فرزند سمت راست درج میشود و رکورد SmallItem برای چرخش از طریق فرزند سمت چپ استفاده میگردد و سپس رکوردهای x و Prev(LargeItem) (رکورد LargeItem در فرزند سمت چپ این گره) بترتیب جای خالی رکوردهای SmallItem و LargeItem را پر میکنند. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



شکل ۴- نحوه اجرای عملگر درج برای یک گره مرز بالا (مورد الف-۲)

۳- اگر رکورد x از رکورد SmallItem بزرگتر و از رکورد LargeItem کوچکتر باشد، رکوردهای x و LargeItem بترتیب از طریق فرزندان سمت چپ و سمت راست درج میشوند و سپس رکورد Prev(LargeItem) جای خالی LargeItem را پر میکنند. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



شکل ۵- نحوه اجرای عملگر درج برای یک گره مرز بالا (مورد الف-۳)

اگر مقدار Balance بزرگتر یا مساوی صفر باشد، رکورد x از طریق فرزند سمت چپ درج میشود که یکی از حالت های زیر اتفاق می افتد.

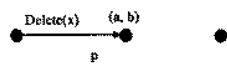
۲- اگر رکورد x از رکورد $SmallItem$ بزرگتر و از رکورد $LargeItem$ کوچکتر باشد رکورد x از طریق فرزند سمت چپ درج میشود.

عملگر حذف: نحوه اجرای این عملگر مشابه عملگر حذف در درخت جستجوی دودویی است یعنی با حذف رکورد x ، رکورد $Next(x)$ (کوچکترین رکورد بزرگتر از x) یا رکورد $Prev(x)$ (بزرگترین رکورد کوچکتر از x) جای خالی x را پر می کنند. در این عملگر، در حین حذف یک رکورد با ایجاد چرخش های لازم خواص توری جستجوی دودویی کامل برقرار می گردد. عملکرد این عملگر برای گره های مرز بالا و گره های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع گره بصورت زیر می باشد.

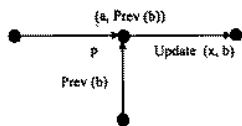
الف - اگر گره q یک گره مرز بالا باشد، عملگر حذف رکورد x را از پدر سمت چپ خود دریافت میکند. اگر گره q فرزندی نداشته باشد رکورد x از این گره حذف میگردد. در غیر اینصورت یکی از شش حالت زیر پیش آید.

۱- اگر رکورد x از رکورد $LargeItem$ و متغیر $Balance$ از صفر بزرگتر باشند رکورد x از طریق فرزند سمت راست حذف میشود.

۲- اگر رکورد x از رکورد $LargeItem$ بزرگتر و متغیر $Balance$ کوچکتر یا مساوی صفر باشند، از طریق فرزند سمت راست رکورد های x و $LargeItem$ بترتیب حذف و درج میشوند و سپس رکورد $Prev(LargeItem)$ جای خالی $LargeItem$ را پر می کند. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



قبل از اجرا عملگر درج

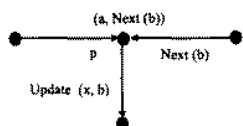


پس از اجرا عملگر درج

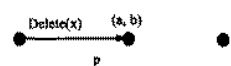
شکل ۸- نحوه اجرای عملگر حذف برای یک گره مرز بالا

۳- اگر رکورد x بزرگتر از رکورد $SmallItem$ و کوچکتر از رکورد $LargeItem$ و متغیر $Balance$ کوچکتر یا مساوی صفر باشد، رکورد x از طریق فرزند سمت چپ حذف میشود.

۴- اگر رکورد x بزرگتر از رکورد $SmallItem$ و کوچکتر از رکورد $LargeItem$ و متغیر $Balance$ بزرگتر از صفر باشد، رکورد x از طریق فرزند سمت چپ حذف و رکورد $LargeItem$ از طریق فرزند سمت چپ درج میگردد. جای خالی $LargeItem$ توسط رکورد $Next(LargeItem)$ پر میشود. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



پس از اجرا عملگر درج

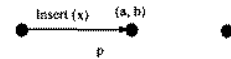


قبل از اجرا عملگر درج

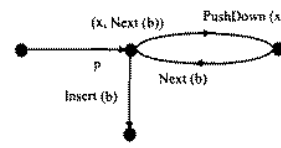
شکل ۹- نحوه اجرای عملگر حذف برای یک گره مرز بالا

۱- اگر رکورد x از رکورد $SmallItem$ بزرگتر و از رکورد $LargeItem$ کوچکتر باشد رکورد x از طریق فرزند سمت چپ درج می شود.

۲- اگر رکورد x از رکورد $LargeItem$ بزرگتر باشد، رکورد $LargeItem$ از طریق فرزند سمت چپ درج می شود و رکورد x برای چرخش از طریق فرزند سمت راست استفاده می گردد و سپس جای خالی رکورد $LargeItem$ توسط رکورد $Next(LargeItem)$ (رکورد $SmallItem$ در فرزند سمت راست این گره) پر میگردد. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



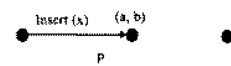
قبل از اجرا عملگر درج



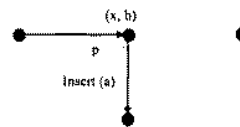
پس از اجرا عملگر درج

شکل ۶- نحوه اجرای عملگر درج برای یک گره مرز بالا

۳- اگر رکورد x از رکورد $SmallItem$ کوچکتر باشد، رکورد $SmallItem$ از طریق فرزند سمت چپ درج میشود و سپس جای خالی آن توسط رکورد x پر می گردد. نحوه اجرای این عمل در شکل زیر نشان داده شده است.



قبل از اجرا عملگر درج



پس از اجرا عملگر درج

شکل ۷- نحوه اجرای عملگر درج برای یک گره مرز بالا

ب- اگر گره q یک گره میانی یا گره مرز چپ یا گره برگ باشد، عملگر درج رکورد x را از پدر سمت راست خود دریافت میکند. در صورتیکه گره q ، گره میانی خالی یا نیمه پر یا گره برگ باشد رکورد x در گره q قرار میگیرد. در غیر اینصورت یکی از سه حالت زیر پیش میاید.

۱- اگر رکورد x از رکورد $LargeItem$ بزرگتر باشد رکورد $LargeItem$ از طریق فرزند سمت چپ درج میشود و سپس جای خالی آن توسط رکورد x پر میگردد.

۲- اگر رکورد x از رکورد $SmallItem$ کوچکتر باشد، رکورد $SmallItem$ از طریق فرزند سمت چپ درج میشود و سپس رکورد x جای خالی آنرا پر میکند.

۳- اگر کلید x کوچکتر از کلید رکورد $SmallItem$ یا بزرگتر از کلید رکورد $LargeItem$ یا گره q خالی باشد، کلید x در توری موجود نمی‌باشد و جستجو پایان می‌پذیرد.

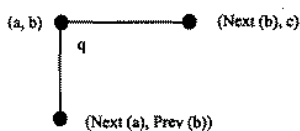
عملگر جستجوی رکورد دارای کوچکترین کلید: وظیفه این عملگر پیدا نمودن رکورد دارای کوچکترین کلید در توری می‌باشد. رکورد دارای کوچکترین کلید در متغیر $SmallItem$ گره ریشه قرار دارد.

عملگر جستجوی رکورد دارای بزرگترین کلید: وظیفه این عملگر پیدا نمودن رکورد دارای بزرگترین کلید در توری می‌باشد. رکورد دارای بزرگترین کلید در سمت راست ترین گره مرز بالای غیر تهی قرار دارد و با حرکت از ریشه به فرزند سمت راست، این رکورد پیدا می‌شود.

عملگر حذف رکورد دارای کوچکترین کلید: وظیفه این عملگر پیدا نمودن رکورد دارای کوچکترین کلید و حذف آن از توری می‌باشد. رکورد دارای کوچکترین کلید در متغیر $SmallItem$ گره ریشه قرار دارد. پس از حذف این رکورد، اگر ریشه دارای فرزندی باشد از چرخش های مشابه چرخش های عملگر حذف (شکل های ۸ الی ۱۰) برای پر نمودن جای خالی رکورد $SmallItem$ استفاده می‌شود.

عملگر جستجوی رکورد بعد: وظیفه این عملگر در صورتیکه کلید x بزرگترین کلید توری نباشد پیدا نمودن کوچکترین رکورد بزرگتر از رکورد x (رکورد $(Next(x))$ می‌باشد و در غیر اینصورت جواب $LastKey$ را تولید می‌کند. عملکرد این عملگر برای گره های مرز بالا و گره های دیگر متفاوت است. این عملگر ابتدا گره ای را پیدا می‌کند که رکورد با کلید x در آن ذخیره شده باشد و سپس براساس نوع گره رکورد بعد را تعیین می‌کند. نحوه اجرای این عملگر با توجه به نوع گره بصورت زیر می‌باشد.

الف - اگر گره q یک گره مرز بالا باشد، عملگر رکورد بعد را از پدر سمت چپ خود دریافت می‌کند. ترتیب رکوردهای ذخیره شده در این گره و فرزندانش بصورت زیر می‌باشد.



شکل ۱۱- نحوه اجرای عملگر جستجوی رکورد بعد

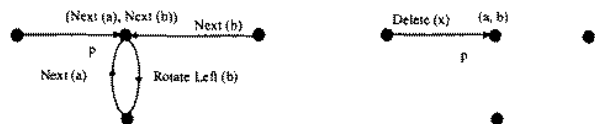
برای اجرای این عملگر یکی از چهار حالت زیر پیش می‌آید.

- ۱- اگر گره q یک گره نیمه پر باشد، این رکورد دارای بزرگترین کلید می‌باشد.
- ۲- اگر گره q یک گره پر بدون فرزند و کلید x مساوی کلید رکورد $SmallItem$ باشد، رکورد بعد، رکورد $LargeItem$ گره q است.
- ۳- اگر گره q یک گره پر بدون فرزند سمت راست و کلید x مساوی کلید رکورد $LargeItem$ گره q باشد این رکورد دارای بزرگترین کلید می‌باشد.
- ۴- اگر گره q یک گره پر دارای فرزند سمت راست و کلید x مساوی کلید رکورد $LargeItem$ باشد، رکورد بعد، رکورد $SmallItem$ فرزند سمت راست این گره است.

۵- اگر گره q یک گره پر دارای فرزند سمت چپ و کلید x مساوی کلید رکورد $SmallItem$ باشد، رکورد بعد، رکورد $SmallItem$ فرزند سمت چپ این گره است. ب- اگر گره q یک گره مرز میانی یا مرز چپ باشد، عملگر رکورد بعد را از پدر سمت راست خود دریافت می‌کند. ترتیب رکوردهای ذخیره شده در این گره و فرزند سمت چپ و پدر سمت راستش در شکل ۱۲ نشان داده شده است. برای اجرای این عملگر یکی از سه حالت زیر پیش می‌آید.

- ۱- اگر گره q یک گره نیمه پر باشد، رکورد بعد، رکورد $LargeItem$ پدر سمت

۵- اگر رکورد x مساوی رکورد $SmallItem$ باشد، رکورد $SmallItem$ حذف و سپس جای خالی آن توسط رکورد $Next(SmallItem)$ پر میشود. اگر متغیر $Balance$ بزرگتر از صفر باشد رکورد $LargeItem$ برای چرخش از طریق فرزند سمت چپ استفاده میشود و سپس جای خالی رکورد $LargeItem$ توسط رکورد $Next(LargeItem)$ پر میشود. نحوه اجرای این چرخش در شکل زیر نشان داده شده است.



پس از اجرا عملگر درج

قبل از اجرا عملگر درج

شکل ۱۰- نحوه اجرای عملگر و حذف برای یک گره مرز بالا

۶- اگر رکورد x مساوی رکورد $LargeItem$ باشد، رکورد $LargeItem$ حذف میشود. اگر متغیر $Balance$ بزرگتر از صفر باشد جای خالی $LargeItem$ توسط رکورد $Next(LargeItem)$ و اگر متغیر $Balance$ مساوی یا کوچکتر از صفر باشد جای خالی $LargeItem$ توسط رکورد $Prev(LargeItem)$ پر میشود.

ب- اگر گره q یک گره میانی، یا مرز چپ و یا برگ باشد، عملگر حذف رکورد x را از پدر سمت راست خود دریافت میکند. اگر گره q یک گره برگ، یک گره میانی نیمه پر و یا پر بدون فرزند باشد رکورد x از این گره حذف می‌شود. اگر گره q یک گره میانی دارای فرزند باشد یکی از سه حالت زیر پیش می‌آید.

- ۱- اگر رکورد x از رکورد $SmallItem$ بزرگتر و از رکورد $LargeItem$ کوچکتر باشد، رکورد x از طریق فرزند سمت چپ حذف می‌شود.
- ۲- اگر رکورد x مساوی رکورد $LargeItem$ باشد، رکورد $LargeItem$ حذف و سپس جای خالی آن توسط رکورد $Prev(LargeItem)$ پر میگردد.
- ۳- اگر رکورد x مساوی $SmallItem$ باشد رکورد $SmallItem$ حذف می‌شود و جای خالی آنرا رکورد $Next(SmallItem)$ پر میکند.

عملگر جستجو: اگر رکورد با کلید x ، در توری ذخیره شده باشد عملگر جستجو رکورد دارای کلید x و در غیر اینصورت $search-fail$ را بعنوان جواب تولید میکند. عملکرد این عملگر برای گره های مرز بالا و گره های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع گره بصورت زیر می‌باشد.

الف - اگر گره q یک گره مرز بالا باشد، عملگر جستجو را از پدر سمت چپ خود دریافت می‌کند، و یکی از چهار حالت زیر پیش می‌آید.

- ۱- اگر کلید x مساوی کلید یکی از رکوردهای ذخیره شده در گره q باشد، کلید x در توری وجود دارد و جستجو پایان می‌پذیرد.
- ۲- اگر کلید x بزرگتر از کلید رکورد $LargeItem$ باشد، جستجو از طریق فرزند سمت راست انجام می‌شود.

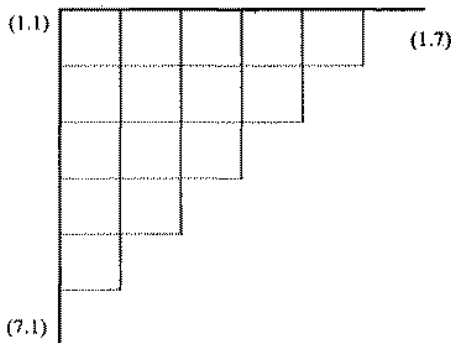
۳- اگر کلید x بزرگتر از کلید رکورد $SmallItem$ و کوچکتر از کلید رکورد $LargeItem$ باشد، جستجو از طریق فرزند سمت چپ انجام می‌شود.

۴- اگر کلید x کوچکتر از کلید رکورد $SmallItem$ یا گره q خالی باشد، کلید x در توری موجود نمی‌باشد و جستجو پایان می‌پذیرد.

ب- اگر گره q یک گره میانی یا مرز چپ یا برگ باشد، عملگر جستجو را از پدر سمت راست خود دریافت می‌کند و یکی از سه حالت زیر پیش می‌آید.

- ۱- اگر کلید x مساوی کلید یکی از رکوردهای ذخیره شده در گره q باشد، کلید x در توری وجود دارد و جستجو پایان می‌پذیرد.
- ۲- اگر کلید x بزرگتر از کلید رکورد $SmallItem$ و کوچکتر از کلید رکورد $LargeItem$ باشد، جستجو از طریق فرزند سمت چپ انجام می‌شود.

شده توسط برگ ها استفاده نمود. شکل زیر یک توری جستجوی دودوئی کامل $Y \times Y$ را نشان می دهد.



شکل ۱۴- توری جستجوی دودوئی کامل $Y \times Y$

مطابق شکل فوق کمانهای نیم توری به دو دسته زیر تقسیم می شوند. **کمانهای ارسال جواب:** کمان متصل کننده هر گره به پدر سمت چپ خود مخصوص ارسال جواب می باشد. در شکل فوق این کمانها توسط خطوط نقطه چین نشان داده شده اند. **کمانهای دریافت عملگر:** برای گره های مرزی بالا کمان متصل کننده هر گره با پدر سمت چپ خود و برای گره های دیگر کمان متصل کننده هر گره با پدر سمت راست خود مخصوص دریافت عملگرهای دیکشنری می باشد. در شکل فوق این کمانها توسط خطوط پر نشان داده شده است. هر گره در توری جستجوی دودوئی کامل دارای اطلاعات زیر می باشد.

Small Item	Left Count	Right Count	Large Item
------------	------------	-------------	------------

الف) اطلاعات موجود در گره های مرز بالا

Small Item	Large Item
------------	------------

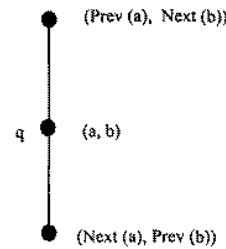
ب) اطلاعات موجود در گره های میانی و مرز چپ

شکل ۱۵- اطلاعات موجود در گره های توری جستجوی دودوئی کامل

SmallItem کوچکترین رکورد در هر گره و **LargeItem** بزرگترین رکورد در هر گره است. در صورتیکه پردازنده برگ باشد بدلیل اینکه بیشتر از یک رکورد در برگها ذخیره می شوند رکوردهای موجود در گره های برگ بصورت مرتب شده ذخیره می شوند. **LeftCount** و **RightCount** بترتیب تعداد رکوردهای درج شده از طریق فرزند سمت راست و فرزند سمت چپ می باشد. طبق قضیه ۸ مقدار متغیر **Balance** در پردازنده $(1, k)$ بصورت زیر می باشد.

$$Balance = \begin{cases} RightCount - \left\lfloor \frac{LeftCount}{2} \right\rfloor \left(\left\lfloor \frac{LeftCount}{2} \right\rfloor + 1 \right) & LeftCount \leq 2(n-k) \\ RightCount - (n-k) \{ LeftCount - (n-k+1) \} & LeftCount > 2(n-k) \end{cases}$$

فرایندی که در هر یک از گره های ابر مکعب اجرا میشود از دو قسمت مجزا تشکیل شده است. قسمت اول فرایند متعلق به برنامه کاربردی است که روی همه گره های ابر مکعب اجرا می شود. قسمت دوم فرایند که اجرا کننده نامیده می شود عملگرهای ماشین دیکشنری را اجرا می کند. برای سادگی فرض می شود یک اجرا کننده می تواند دریافت و پردازش تعدادی پیغام را بطور همزمان انجام دهد. پیغام



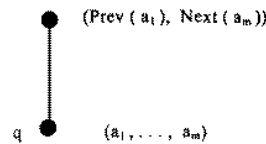
شکل ۱۲- نحوه اجرای عملگر جستجوی رکورد بعد

راست این گره می باشد.

۲- اگر کلید x مساوی کلید رکورد **LargeItem** باشد، رکورد بعد، رکورد **LargeItem** پدر سمت راست این گره است.

۳- اگر گره q یک گره دارای فرزند سمت چپ و کلید x مساوی کلید رکورد **SmallItem** باشد، رکورد بعد، رکورد **SmallItem** فرزند سمت چپ این گره می باشد.

ج - اگر گره q یک برگ باشد، عملگر رکورد بعد را از پدر سمت راست خود دریافت می کند. ترتیب رکوردهای ذخیره شده در این گره و پدر سمت راستش بصورت زیر می باشد.



شکل ۱۳- نحوه اجرای عملگر جستجوی رکورد بعد

فرض کنید که m رکورد (a_1, a_2, \dots, a_m) در برگ q ذخیره شده باشد بطوریکه برای $k = 1, \dots, m-1$ شرط $a_k < a_{k+1}$ برقرار باشد. برای اجرای این عملگر یکی از دو حالت زیر پیش می آید.

۱- اگر کلید x مساوی کلید رکورد a_m باشد، رکورد بعد، رکورد **LargeItem** پدر سمت راست گره q است.

۲- اگر کلید x مساوی کلید رکورد a_k ($k < m$) باشد، رکورد بعد، رکورد a_{k+1} گره q است.

۴- پیاده سازی

در این قسمت چگونگی پیاده سازی توری جستجوی دودوئی کامل روی یک کامپیوتر موازی با حافظه توزیع شده و ساختار ابر مکعب یا توری شرح داده می شود.

نیم توری بگونه ای در یک ابر مکعب نگاشت می شود که خاصیت همسایگی گره ها در آن حفظ شده باشد. یک روش شناخته شده برای برقراری خاصیت همسایگی در این نگاشت استفاده از کد گری انعکاسی است. این روش را میتوان بصورت زیر شرح داد: گره (x_1, x_2) در یک توری دو بعدی به گره $S_1 S_2$ در ابر مکعب نگاشت داده می شود بطوریکه S_i, x_i آمین کد گری i بیتی می باشد [۳۰، ۹، ۸].

گره هایی از ابر مکعب که عضو نیم توری نگاشت شده نیستند تشکیل یک نیم توری $(n-1) \times (n-1)$ را می دهند که نیم توری آینه ای نامیده می شود. نیم توری آینه ای را میتوان برای ذخیره نمودن ساختمان داده دیگری و یا ترکیب نتایج تولید

```

else
    SEND (LC (q), RotateRight (SmallItem (q)))
    SmallItem (q) = x
end if
end if
inc (RightCount (q))
else
    if x < SmallItem (q) then
        SEND (LC (q), Insert (SmallItem (q)))
        SmallItem (q) = x
    else
        if x < LargeItem (q) then
            SEND (LC (q), Insert (x))
        else
            SEND (RC (q), PushDown (x))
            SEND (LC (q), Insert (LargeItem (q)))
        end if
    end if
    inc (LeftCount (q))
end if
end if
end if

```

ب- اگر پردازنده q یک پردازنده میانی، سرز چپ و یا برگ باشد عملگر Insert (x) را از پدر سمت راست خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if inst = Insert (x) then
    if leaf (q) or empty (q) or q has one record then
        insert x into node q
    else {q has two records}
        if x < SmallItem (q) then
            SEND (LC (q), Insert (SmallItem (q)))
            SmallItem (q) = x
        else
            if x < LargeItem (q) then
                SEND (LC (q), Insert (x))
            else
                SEND (LC (q), Insert (LargeItem (q)))
                LargeItem (q) = x
            end if
        end if
    end if
end if
end if

```

عملگر حذف: عملکرد این عملگر برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد. الف- اگر پردازنده q یک پردازنده مرز بالا باشد عملگر Delete(x) را از پدر سمت چپ خود دریافت می‌کند. در اجرای این عملگر، اگر رکورد x از طریق فرزند سمت راست حذف شود مقدار متغیر RightCount و در صورتیکه از طریق فرزند سمت چپ حذف شود مقدار متغیر LeftCount کاهش پیدا می‌کند. برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = Delete (x) then
    if LC (q) = null then
        delete x
    else
        if Balance (q) ≥ 0 then
            if x > LargeItem (q) then
                SEND (RC (q), Delete (x))
            else
                SEND (RC (q), PopUpSmall)
            if SmallItem < x < LargeItem (q) then
                SEND (RC (q), Update (x, LargeItem (q)))
            else

```

برای این عملگرها بوسیله فرآیند کاربردی که روی گره‌های ابر مکعب اجرا می‌شوند راه اندازی می‌گردد. این پیغام‌ها پس از دریافت به اجرا کننده ارسال می‌شود سپس اجرا کننده این پیغام‌ها را برای اجرا به گره ریشه ارسال می‌کند. عملگرهایی که بوسیله ریشه دریافت می‌شوند بوسیله پردازش اولیه ای اجرا می‌گردند. در صورت نیاز نتیجه هر عمل به پردازنده درخواست کننده ارسال می‌شود. هر اجرا کننده می‌تواند یک عملگر را از یکی از همسایگان خود یا قسمت برنامه کاربردی فرآیند خود دریافت نماید.

ما از گرامر و معنای زیر برای دستورات SEND و RECEIVE استفاده می‌کنیم. دستور SEND (<processor>, <instruction>) دستورالعمل <instruction> را برای اجرا به پردازنده <processor> ارسال می‌کند. پردازنده‌ای که دستورالعمل (<processor>, <information>) را اجرا می‌کند سبب دریافت اطلاعات <information> از پردازنده <processor> و ارسال آن به فرآیند درخواست کننده می‌گردد. در این مقاله فرض می‌شود که دستور RECEIVE دستور Blocking باشد (روبه‌ای که دستور RECEIVE را اجرا می‌کند منتظر می‌ماند تا اطلاعات مورد نیاز دریافت شود). اگر مقدار <processor> برابر anyprocessor باشد در اینصورت دستورالعمل RECEIVE زمانی کامل می‌گردد که اطلاعات از یکی از پردازنده‌ها دریافت شود. برنامه کاربردی که روی پردازنده p اجرا می‌شود عملگر دیکشنری را توسط دستور SEND (root, <instruction>) به ریشه ارسال می‌کند که <instruction> یکی از عملگرهای دیکشنری است. سپس اگر دستورالعمل صادر شده نیاز به جواب داشته باشد دستورالعمل RECEIVE (anyprocessor, <instruction-done>) را اجرا می‌کند و منتظر اتمام دستور می‌گردد. مقدار <instruction-done> می‌تواند برابر یکی از مقادیر search-fail, search-success, max-found, anyprocessor باشد و NextKey, min-deleted, LastKey باشد و شماره یک پردازنده در ابر مکعب یا توری می‌باشد.

برای اجرای عملگرهای دیکشنری از دستورات زیر استفاده شده است. برای درج رکورد با کلید x از دستور Insert (x). برای جستجوی رکورد با کلید x از دستور Search (p,x). برای حذف رکورد با کلید x از دستور Delete(x). برای پیدا کردن رکورد دارای کوچکترین کلید از دستور Min (p). برای پیدا کردن رکورد دارای بزرگترین کلید از دستور Max(p). برای حذف رکورد دارای کوچکترین کلید از دستور XMin(p) و برای رکورد بعد از دستور Next(p,x) استفاده می‌شود. p پردازنده‌ای است که عملگر را شروع می‌کند. حال به چگونگی پیاده‌سازی عملگرهای ماشین دیکشنری می‌پردازیم.

عملگر درج: عملکرد این عملگر برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد. الف- اگر q یک پردازنده مرز بالا باشد عملگر Insert(x) را از پدر سمت چپ خود دریافت می‌کند. در اجرای این عملگر، اگر رکورد x از طریق فرزند سمت راست درج شود مقدار متغیر RightCount و در صورتیکه از طریق فرزند سمت چپ درج شود مقدار متغیر LeftCount افزایش پیدا می‌کند. برای اجرای این عملگر پردازنده مرز بالا q قطعه کد زیر را اجرا می‌کند.

```

if inst = Insert (x) then
    if empty (q) or q has one record then
        insert x into node q
    else {q has two records}
        if Balance (q) < 0 then
            if x > LargeItem (q) then
                SEND (RC (q), Insert (x))
            else
                SEND (RC (q), Insert (LargeItem (q)))
            if x > SmallItem (q) then
                SEND (LC (q), RotateRight (x))

```

```

if x is in q then
  SEND (p, search-success)
else
  SEND (p, search-fails)
end if
else
  if empty (q) or x < SmallItem (q) or x > LargeItem (q) then
    SEND (p, search-fails)
  else
    if SmallItem (q) < x < LargeItem (q) then
      SEND (LC (q), Search (p, x))
    else
      SEND (p, search-success)
    end if
  end if
end if
end if

```

عملگر جستجوی رکورد دارای کوچکترین کلید: بدلیل اینکه رکورد دارای کوچکترین کلید در پردازنده ریشه قرار دارد عملگر Min(p) تنها در پردازنده ریشه اجرا می‌شود. برای اجرای این عملگر پردازنده ریشه قطعه کد زیر را اجرا می‌کند.

```

if inst = Min (p) then
  SEND (p, min-found (SmallItem (root)))
end if

```

عملگر جستجوی رکورد بعد: عملکرد این عملگر برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد.

الف- اگر q پردازنده مرز بالا باشد عملگر Next (p, x) را از پدر سمت چپ خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if inst = Next (p, x) then
  if q has one record then
    SEND (p, lastKey)
  else
    if q has no childs then
      if x = SmallItem (q) then
        SEND (p, NextKey (LargeItem (q)))
      else
        SEND (p, lastKey)
      end if
    else
      if x = SmallItem (q) then
        SEND (LC (q), SendSmall (p))
      else
        if x < LargeItem (q) then
          SEND (LC (q), Next (p, x))
        else
          if x = LargeItem (q) then
            if RC (q) ≠ null then
              SEND (RC (q), SendSmall (p))
            else
              SEND (p, last-record)
            end if
          else
            SEND (RC (q), Next (p, x))
          end if
        end if
      end if
    end if
  end if
end if
end if
end if
end if

```

ب- اگر یک پردازنده میانی، مرز چپ و یا برگ باشد، عملگر Next(p,x) را از پدر سمت راست خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if x = SmallItem (q) then
  SEND (LC (q), RotateLeft (LargeItem (q)))
end if
end if
end if
dec (RightCount (q))
else { Balance (p) ≤ 0 }
  if x ≥ LargeItem (q) then
    SEND (LC (q), PopupLarge)
  if x > LargeItem (q) then
    SEND (RC (q), Update (x, LargeItem (q)))
  end if
else
  if x > SmallItem then
    SEND (LC (q), Delete (x))
  else
    SEND (LC (q), PopupSmall)
  end if
end if
dec (LeftCount (q))
end if
end if
end if

```

ب- اگر پردازنده q یک پردازنده میانی، مرز چپ و یا برگ باشد عملگر Delete(x) را از پدر سمت راست خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if inst = Delete (x) then
  if leaf (q) or LC (q) = null then
    delete x
  else
    if x = LargeItem (q) then
      SEND (LC (q), PopupLarge)
    else
      if x = SmallItem (p) then
        SEND (LC (p), PopupSmall)
      else
        SEND (LC (p), Delete (x))
      end if
    end if
  end if
end if

```

عملگر جستجو: عملکرد این عملگر برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد.

الف- اگر q پردازنده مرز بالا باشد عملگر Search(p, x) را از پدر سمت چپ خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if inst = Search (p, x) then
  if empty (q) or x < SmallItem (q) then
    SEND (p, search-fails)
  else
    if x > LargeItem (q) then
      SEND (RC (q), Search (p, x))
    else
      if x > SmallItem (q) then
        SEND (LC (q), Search (p, x))
      else
        SEND (p, search-success)
      end if
    end if
  end if
end if

```

اگر q یک پردازنده میانی یا مرز چپ یا برگ باشد عملگر Search(p,x) را از پدر سمت راست خود دریافت و قطعه کد زیر را اجرا می‌کند.

```

if inst = Search (p, x) then
  if leaf (q) then

```

برای حفظ توازن در توری، رکورد LargestItem برای چرخش به چپ به فرزند سمت چپ ارسال می‌گردد و سپس جای خالی رکورد LargestItem توسط رکورد Next (LargestItem) (که توسط عملگر PopUpSmall مشخص می‌گردد) پر می‌شود. برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = PopUpSmall then
  SEND (LP (q), SetLarge (SmallItem (q)))
  if LC (q) = null then
    delete SmallItem (q)
  else
    if Balance (q) ≤ 0 null then
      SEND (LC (q), PopUpSmall)
      dec (LeftCount (q))
    else
      SEND (RC (q), PopUpSmall)
      SEND (LC (q), RotateLeft (LargestItem (q)))
      dec (RightCount (q))
    end if
  end if
end if

```

ب- اگر q یک پردازنده میانی، مرز چپ و یا برگ باشد عملگر PopUpSmall را از پدر سمت راست خود دریافت می‌کند و باعث می‌شود تا رکورد SmallItem از این گره حذف شود و برای جایگزینی در رکورد SmallItem پردازنده پدر سمت راست q به آن پردازنده ارسال گردد. اگر پردازنده q، یک پردازنده میانی یا مرز چپ دارای فرزند باشند عملگر PopUpSmall برای پر نمودن جای خالی رکورد حذف شده SmallItem به فرزند سمت چپ ارسال می‌شود. برای اجرای این عملگر پردازنده میانی، مرز چپ و یا برگ q قطعه کد زیر را اجرا می‌کند.

```

if inst = PopUpSmall then
  if leaf (q) then
    find the smallest value y in q and delete it
    SEND (RP (q), SetSmall (y))
  else
    SEND (RP (q), SetSmall (SmallItem (q)))
    if LC (q) = null then
      delete SmallItem
    else
      SEND (LC (q), PopUpSmall)
    end if
  end if
end if

```

عملگر PopUpLarge: این عملگر توسط عملگر Delete(x) استفاده می‌شود و تنها در پردازنده‌های میانی، مرز چپ و یا برگ اجرا و باعث می‌شود تا رکورد LargestItem از این گره حذف و برای جایگزینی در رکورد LargestItem پدر سمت راست ارسال شود. اگر پردازنده q، یک پردازنده میانی و یا مرز چپ دارای فرزند باشند عملگر PopUpLarge برای پر نمودن جای خالی رکورد حذف شده LargestItem به فرزند سمت چپ ارسال می‌شود. برای اجرای این عملگر پردازنده میانی و یا مرز چپ یا برگ q قطعه کد زیر را اجرا می‌کند.

```

if inst = PopUpLarge then
  if leaf (q) then
    find the largest value y in q and delete it
    SEND (RP (q), SetLarge (y))
  else
    SEND (RP (q), SetLarge (LargestItem (q)))
    if LC (q) = null then
      delete LargestItem (q)
    else
      SEND (LC (q), PopUpLarge)
    end if
  end if
end if

```

```

if inst = Next (p, x) then
  if leaf (q) then
    if x is largest record of node q then
      SEND (RP (q), SendLarge (p))
    else
      SEND (p, NextKey (next record of x))
    end if
  else
    if x = LargestItem (q) then
      SEND (RP (q), SendLarge (p))
    else { x = SmallItem (q) }
    if LC (q) = null then
      if q has one record then
        SEND (RP (q), SendLarge (p))
      else
        SEND (p, NextKey (LargestItem (q)))
      end if
    else
      SEND (LC (q), SendSmall (p))
    end if
  end if
end if
end if

```

عملگر جستجوی رکورد دارای بزرگترین کلید: بدلیل اینکه رکورد دارای بزرگترین کلید در پردازنده های مرز بالا ذخیره میشود، عملگر Max(p) تنها در این پردازنده ها اجرا می‌شود. برای اجرای این عملگر پردازنده های مرز بالا q قطعه کد زیر را اجرا می‌کنند.

```

if inst = Max (p) then
  if empty (RC (q)) then
    SEND (p, min-found (LargestItem (q)))
  else
    SEND (RC (q), Max (p))
  end if
end if

```

عملگر حذف رکورد دارای کوچکترین کلید: بدلیل اینکه رکورد دارای کوچکترین کلید در پردازنده ریشه قرار دارد، عملگر XMin(p) تنها در پردازنده ریشه اجرا می‌شود. برای اجرای این عملگر پردازنده ریشه قطعه کد زیر را اجرا می‌کند.

```

if inst = XMin (p) then
  if LC (root) ≠ null or RC (root) ≠ null then
    if Balance (p) ≥ 0 then
      SEND (RC (root), PopUpSmall)
      SEND (LC (root), RotateLeft (LargestItem (root)))
      dec (RightCount (root))
    else
      SEND (LC (root), PopUpSmall)
      dec (LeftCount (root))
    end if
  end if
  SEND (p, min-deleted (SmallItem (root)))
end if

```

عملگر PopUpSmall: این عملگر توسط عملگر Delete(x) استفاده می‌شود و نحوه عملکرد آن برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد.

الف- اگر q یک پردازنده مرز بالا باشد، عملگر PopUpSmall را از پدر سمت چپ خود دریافت می‌کند و سبب می‌گردد تا رکورد SmallItem از پردازنده q حذف و برای پر کردن جای خالی، رکورد LargestItem مربوط به پدر سمت چپ پردازنده q به آن پردازنده ارسال شود و سپس جای خالی رکورد SmallItem توسط رکورد Next (SmallItem) پر می‌گردد. اگر مقدار متغیر Balance بزرگتر از صفر باشد،

۲- اگر رکورد x مساوی رکورد $LargeItem$ باشد رکورد $SmallItem$ برای چرخش به راست به فرزند سمت چپ ارسال می‌شود و سپس رکوردهای y و $Prev(LargeItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargeItem$ را پر می‌کنند.

۳- اگر رکورد x بزرگتر از رکورد $SmallItem$ و کوچکتر از رکورد $LargeItem$ باشد رکوردهای x و $SmallItem$ توسط عملگر $Update(x, SmallItem)$ برای حذف رکورد x و درج رکورد $SmallItem$ به فرزند سمت چپ ارسال می‌شوند و سپس رکورد y جای خالی رکورد $SmallItem$ را پر می‌کنند. برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = Update (x, y) then
  if SmallItem (q) < x < LargeItem (q) then
    SEND (LC (q), Update (x, SmallItem (q)))
  else
    if x ≥ LargeItem (q) then
      SEND (LC (q), RotateRight (SmallItem (q)))
    if x > LargeItem (q) then
      SEND (RC (q), Update (x, LargeItem (q)))
    end if
  end if
end if
SmallItem (q) = y
end if
    
```

ب- اگر پردازنده q یک پردازنده میانی، مرز چپ و یا برگ باشد عملگر $Update(x, y)$ را از پدر سمت راست خود دریافت می‌کند. با توجه به نوع استفاده عملگر $Update(x, y)$ ، رکورد y همواره کوچکتر از رکورد $SmallItem$ پردازنده q است. بنابراین محل درج رکورد y در متغیر $SmallItem$ پردازنده q میباشد. با توجه به مقدار رکورد x یکی از حالات زیر اتفاق می‌افتد.

۱- اگر رکورد x مساوی رکورد $SmallItem$ باشد، رکورد y جایگزین رکورد $SmallItem$ می‌شود.

۲- اگر x رکورد مساوی رکورد $LargeItem$ باشد، رکورد $SmallItem$ برای چرخش به راست به فرزند سمت چپ ارسال می‌شود و سپس رکوردهای y و $Prev(LargeItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargeItem$ را پر می‌کنند.

۳- اگر رکورد x بزرگتر از رکورد $LargeItem$ باشد، رکورد $SmallItem$ برای چرخش به راست به فرزند سمت چپ و رکوردهای x و $LargeItem$ توسط عملگر $Update(x, LargeItem)$ برای حذف رکورد x و درج رکورد $LargeItem$ به فرزند سمت راست ارسال می‌شوند و سپس رکوردهای y و $Prev(LargeItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargeItem$ را پر می‌کنند.

۴- اگر رکورد x بزرگتر از رکورد $SmallItem$ و کوچکتر از رکورد $LargeItem$ باشد، رکوردهای x و $SmallItem$ توسط عملگر $Update(x, SmallItem)$ برای حذف رکورد x و درج رکورد $SmallItem$ به فرزند سمت چپ ارسال می‌شوند و سپس رکورد y جای خالی رکورد $SmallItem$ را پر می‌کنند. برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = Update (x, y) then
  if leaf (q) or LC (q) = null then
    delete x from q and insert y into node q
  else
    if x = LargeItem (q) then
      SEND (LC (q), RotateRight (SmallItem (q)))
    else
      if x > SmallItem (q) then
        SEND (LC (q), Update (x, SmallItem (q)))
      end if
    end if
  end if
end if
    
```

```

end if
end if
end if
    
```

عملگر $PushDown(x)$: عملگر $PushDown(x)$ توسط عملگر $Insert(x)$ راه اندازی و تنها در پردازنده مرز بالا q اجرا می‌شود و سبب می‌گردد تا رکورد x از طریق این پردازنده درج و سپس کوچکترین رکورد در این پردازنده و فرزند های آن حذف و به پدر سمت چپ ارسال گردد. با توجه به مقدار رکورد x یکی از حالات زیر اتفاق می‌افتد.

۱- اگر پردازنده q خالی یا رکورد x از رکورد $SmallItem$ کوچکتر باشد، رکورد x به پدر سمت چپ ارسال می‌شود.

۲- اگر پردازنده q پر یا نیمه پر و فرزندی نداشته باشد، رکورد $SmallItem$ حذف و به پدر سمت چپ ارسال می‌شود و سپس رکورد x در این پردازنده درج می‌شود.

۳- اگر پردازنده q دارای فرزند سمت چپ و رکورد x بزرگتر از رکورد $LargeItem$ باشد رکورد $SmallItem$ حذف و به پدر سمت چپ ارسال می‌گردد و رکورد $LargeItem$ برای چرخش به چپ به فرزند سمت چپ و رکورد x توسط عملگر $PushDown(x)$ به فرزند سمت راست ارسال می‌گردند و سپس جای خالی رکوردهای $SmallItem$ و $LargeItem$ بترتیب توسط رکوردهای $Next(SmallItem)$ و $Next(LargeItem)$ پر می‌شوند.

۴- اگر پردازنده q دارای فرزند سمت چپ و رکورد x کوچکتر از رکورد $LargeItem$ و بزرگتر از رکورد $SmallItem$ باشد رکورد $SmallItem$ حذف و به پدر سمت چپ ارسال می‌گردد و رکورد x برای چرخش به چپ به فرزند سمت چپ ارسال می‌گردد. جای خالی رکورد $SmallItem$ توسط رکورد $Next(SmallItem)$ پر می‌شود.

برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = PushDown (x) then
  if empty (q) or x < SmallItem (q) then
    SEND (LP (q), SetLarge (x))
  else
    SEND (LP (q), SetLarge (SmallItem (q)))
    if LC (q) = null then
      insert x into q
    else
      if x > LargeItem (q) then
        SEND (LC (q), RotateLeft (LargeItem (q)))
        SEND (RC (q), PushDown (x))
      else
        SEND (LC (q), RotateLeft (x))
      end if
    end if
  end if
end if
end if
    
```

عملگر $Update(x, y)$: این عملگر توسط عملگر $Delete(x)$ استفاده می‌شود و باعث می‌شود تا رکورد x از طریق این پردازنده حذف و رکورد y در جای مناسب خود درج شود. نحوه عملکرد آن برای پردازنده‌های مرز بالا و پردازنده‌های دیگر متفاوت است. نحوه اجرای این عملگر با توجه به نوع پردازنده بصورت زیر می‌باشد.

الف- اگر q یک پردازنده مرز بالا باشد، عملگر $Update(x, y)$ را از پدر سمت چپ خود دریافت می‌کند. با توجه به طریقه استفاده عملگر $Update(x, y)$ ، رکورد y همواره کوچکتر از رکورد $SmallItem$ پردازنده q است. بنابراین محل درج رکورد y در متغیر $SmallItem$ پردازنده q می‌باشد. با توجه به مقدار رکورد x یکی از حالات زیر اتفاق می‌افتد.

۱- اگر رکورد x مساوی رکورد $SmallItem$ باشد، رکورد y جایگزین رکورد $SmallItem$ می‌شود.

۳- اگر رکورد x کوچکتر از رکورد $LargeItem$ و بزرگتر از رکورد $SmallItem$ باشد، رکورد $SmallItem$ حذف و به پدر سمت راست و رکورد x برای چرخش به راست به فرزند سمت چپ ارسال می‌گردند و سپس جای خالی رکورد $SmallItem$ توسط رکورد $Next (SmallItem)$ پر می‌شوند.

برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = RotateLeft (x) then
  if leaf (q) or LC (q) = null then
    insert x into node q
    find the largest value y and delete it from node q
    SEND (RP (q), SetSmall (y))
  else
    if x < SmallItem (q) then
      SEND (RP (q), SetSmall (x))
    else
      SEND (RP (q), SetSmall (SmallItem (q)))
      if x > LargeItem (q) then
        SEND (LC (q), RotateLeft (LargeItem (q)))
        LargeItem (q) = x
      else
        SEND (LC (q), RotateLeft (x))
      end if
    end if
  end if
end if

```

عملگرهای $SetSmall$ ، $SetLarge$ ، $SendSmall$ که در زیر آمده‌اند توسط همه پردازنده‌ها اجرا می‌شوند.

```

if inst = SetSmall (x) then
  SmallItem = x
end if

if inst = SetLarge (x) then
  LargeItem = x
end if

if inst = SendSmall (p) then
  SEND (p, NextKey (SmallItem (q)))
end if

if inst = SendLarge (p) then
  SEND (p, NextKey (LargeItem (q)))
end if

```

قضیه ۸: عملگر درج (درج رکورد x) خواص توری جستجوی دودویی را حفظ می‌کند.

اثبات: اثبات توسط استقرا روی تعداد رکوردهای نیم توری (N) انجام می‌گیرد.

پایه استقرا: اگر $N=0$ یا $N=1$ باشد درج رکورد x در نیم توری خصوصیات توری جستجوی دودویی را حفظ می‌کند زیرا ریشه نیم توری دارای صفر یا یک رکورد است و رکورد x در ریشه قرار می‌گیرد.

فرض استقرا: اگر $N = K (K > 1)$ باشد فرض می‌کنیم که خصوصیات توری جستجوی دودویی کامل برقرار باشد.

حکم استقرا: اگر نیم توری دارای $N = K (K > 1)$ رکورد و خصوصیات توری جستجوی دودویی کامل برقرار باشد در اینصورت درج رکورد x خصوصیات توری جستجوی دودویی کامل را حفظ می‌کند.

با توجه به تعریف توری جستجوی دودویی کامل برای درج رکورد x یکی از دو حالت زیر پیش می‌آید.

الف- اگر $Balance < 0$ باشد یکی از چهار حالت زیر پیش می‌آید.

```

end if
end if
SmallItem (q) = y
end if
end if

```

عملگر چرخش به راست: این عملگر فقط در پردازنده میانی، مرز چپ و یا برگ q اجرا می‌شود و سبب می‌شود تا رکورد x از طریق این پردازنده درج شود و سپس رکورد $LargeItem$ از پردازنده q حذف و به پدر سمت راست ارسال گردد. اگر پردازنده q ، یک پردازنده برگ یا یک پردازنده غیر برگ بدون فرزند باشد پس از درج رکورد x ، بزرگترین رکورد این پردازنده به پدر سمت راست ارسال می‌شود. با توجه به مقدار رکورد x یکی از حالات زیر اتفاق می‌افتد.

۱- اگر پردازنده q خالی یا رکورد x از رکورد $LargeItem$ بزرگتر باشد، رکورد x به پدر سمت راست ارسال می‌شود.

۲- اگر رکورد x کوچکتر از رکورد $SmallItem$ باشد رکورد $LargeItem$ حذف و به پدر سمت راست و رکورد $SmallItem$ برای چرخش به راست به فرزند سمت چپ ارسال می‌شوند و سپس جای خالی رکوردهای $SmallItem$ و $LargeItem$ بترتیب توسط رکوردهای x و $Prev (LargeItem)$ پر می‌شوند.

۳- اگر رکورد x کوچکتر از رکورد $LargeItem$ و بزرگتر از رکورد $SmallItem$ باشد رکورد $LargeItem$ حذف و به پدر سمت راست و رکورد x برای چرخش به چپ به فرزند سمت چپ ارسال می‌گردند و سپس جای خالی رکورد $LargeItem$ توسط رکورد $Prev (LargeItem)$ پر می‌شود.

برای اجرای این عملگر پردازنده q قطعه کد زیر را اجرا می‌کند.

```

if inst = RotateRight (x) then
  if leaf (q) or LC (q) = null then
    insert x into node q
    find the largest value y and delete it from node q
    SEND (RP (q), SetLarge (y))
  else
    if x > LargeItem (q) then
      SEND (RP (q), SetLarge (x))
    else
      SEND (RP (q), SetLarge (LargeItem (q)))
      if x < SmallItem (q) then
        SEND (LC (q), RotateRight (SmallItem (q)))
        SmallItem (q) = x
      else
        SEND (LC (q), RotateRight (x))
      end if
    end if
  end if
end if

```

عملگر چرخش به چپ: این عملگر فقط در پردازنده میانی، مرز چپ و یا برگ q اجرا می‌شود و سبب می‌شود تا رکورد x از طریق پردازنده درج شود و سپس رکورد $SmallItem$ پردازنده q حذف و به پدر سمت راست ارسال گردد. اگر پردازنده q ، یک پردازنده برگ و یا یک پردازنده غیر برگ بدون فرزند باشد پس از درج رکورد x ، بزرگترین رکورد این پردازنده به پدر سمت راست ارسال می‌شود. با توجه به مقدار رکورد x یکی از حالات زیر اتفاق می‌افتد.

۱- اگر پردازنده q خالی و یا رکورد x کوچکتر از رکورد $SmallItem$ باشد، رکورد x به پدر سمت راست ارسال می‌شود.

۲- اگر رکورد x بزرگتر از رکورد $LargeItem$ باشد، رکورد $SmallItem$ حذف و به پدر سمت راست و رکورد $LargeItem$ برای چرخش به چپ به فرزند سمت چپ ارسال می‌شوند و سپس جای خالی رکوردهای $SmallItem$ و $LargeItem$ بترتیب توسط رکوردهای x و $Prev (LargeItem)$ پر می‌شوند.

۴- اگر نیم توری سمت چپ، یک توری جستجوی دودوئی کامل با عمق $m-1$ (که گره‌های سطح آخر آن نیمه پر هستند) و نیم توری سمت راست، توری جستجوی دودوئی کامل با عمق $m-1$ و گره‌های سطح آخر نیمه پر باشد، برای حفظ خواص توری جستجوی دودوئی کامل رکورد باید در نیم توری سمت چپ درج گردد. با توجه به رویه‌های پوشانی دو نیم توری و با توجه به اینکه گره‌های مرز بالا عملگر را تنها از پدر سمت چپ خود دریافت می‌کنند یک رکورد باید در نیم توری سمت راست درج شود که یکی از سه حالت زیر پیش می‌آید.

۱-۴ اگر $LargerItem > x$ باشد، رکورد x ، باید در نیم توری سمت راست درج شود.

۲-۴ اگر $SmallItem < x$ باشد، رکورد $LargerItem$ ، باید نیم توری سمت راست و رکورد $SmallItem$ در نیم توری سمت چپ درج شود و سپس رکورد های x و $Prev(LargerItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargerItem$ را پر کنند.

۳-۴ اگر $SmallItem < x < LargerItem$ باشد، رکوردهای x و $LargerItem$ باید بترتیب در نیم توری های سمت چپ و سمت راست درج شوند و سپس رکورد $Prev(LargerItem)$ جای خالی $LargerItem$ را پر کند.

ب- اگر $Balance=0$ باشد نیم توری های سمت چپ و راست، توری جستجوی دودوئی کامل پر هستند و رکورد جدید باید در نیم توری سمت چپ درج شود که یکی از سه حالت زیر پیش می‌آید.

۱- اگر $LargerItem > x$ باشد، باید رکورد $LargerItem$ در نیم توری سمت چپ و رکورد x در نیم توری سمت راست درج گردد و سپس جای خالی رکورد $LargerItem$ توسط رکورد $Next(LargerItem)$ پر گردد.

۲- اگر $SmallItem < x$ باشد، باید رکورد $SmallItem$ در نیم توری سمت چپ درج و سپس جای خالی آن توسط رکورد x پر گردد.

۳- اگر $SmallItem < x < LargerItem$ باشد، رکورد x ، باید در نیم توری سمت چپ درج شود.

با توجه به اینکه رویه فوق توازن در ریشه و همچنین مرتب بودن رکوردها را حفظ می‌کند بنابراین عملگر درج خواص توری جستجوی دودوئی کامل را در ریشه حفظ می‌کند. حال باید ثابت نماییم که درج در نیم توری های سمت چپ و سمت راست خواص توری جستجوی دودوئی کامل را حفظ می‌کنند. درج در نیم توری سمت راست طبق رویه فوق انجام می‌پذیرد بنابراین خواص حفظ می‌گردد. بدلیل اینکه دو نیم توری سمت چپ و سمت راست در یک ستون نیم توری رویه پوشانی ندارند و کامل بودن توری تنها در گره‌های مرز بالا بررسی می‌گردد بنابراین کافی است اثبات نماییم که درج در این ستون خواص توری (ترتیب رکوردها) را حفظ می‌کند. در صورتیکه ریشه نیم توری گره خالی یا نیمه پر یا برگ باشد رکورد x باید در ریشه قرار گیرد. در غیر اینصورت یکی از سه حالت زیر پیش می‌آید.

۱- اگر $LargerItem > x$ باشد، باید رکورد $LargerItem$ در نیم توری سمت چپ درج و سپس جای خالی آن توسط رکورد x پر گردد.

۲- اگر $SmallItem < x$ باشد، باید رکورد $SmallItem$ در نیم توری سمت چپ درج و سپس جای خالی آن توسط رکورد x پر گردد.

رکورد $SmallItem$ در نیم توری سمت چپ درج و سپس جای خالی آن توسط رکورد x پر گردد.

۳- اگر $SmallItem < x < LargerItem$ باشد، رکورد x ، باید در نیم توری سمت چپ درج شود. ■

قضیه ۹: عملگر حذف خواص توری جستجوی دودوئی را حفظ می‌کند.

اثبات قضیه فوق در [۵] آمده است.

۱- اگر نیم توری سمت چپ یک توری جستجوی دودوئی کامل پر با گره‌های سطح آخر پر و همچنین نیم توری سمت راست توری جستجوی دودوئی کامل باشد، با توجه به روی هم پوشانی دو نیم توری سمت چپ و سمت راست، تنها یک گره نیمه پر در نیم توری سمت راست وجود دارد و یک رکورد باید در آن (در نیم توری سمت راست درج گردد) قرار گیرد تا نیم توری سمت راست به یک توری جستجوی دودوئی کامل تبدیل گردد. با توجه به مقدار رکورد x یکی از سه حالت زیر پیش می‌آید.

۱-۱ اگر $LargerItem > x$ باشد، رکورد x ، باید در نیم توری سمت راست درج شود.

۲-۱ اگر $SmallItem < x$ باشد، رکورد $LargerItem$ ، باید نیم توری سمت راست و رکورد $SmallItem$ در نیم توری سمت چپ درج شود و سپس رکوردهای x و $Prev(LargerItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargerItem$ را پر کنند.

۳-۱ اگر $SmallItem < x < LargerItem$ باشد، رکوردهای x و $LargerItem$ باید بترتیب در نیم توری های سمت چپ و سمت راست درج شوند و سپس رکورد $Prev(LargerItem)$ جای خالی $LargerItem$ را پر کند.

۲- اگر نیم توری سمت چپ، یک توری جستجوی دودوئی کامل پر با گره‌های سطح آخر نیمه پر و نیم توری سمت راست، توری جستجوی دودوئی کامل باشد، با توجه به روی هم پوشانی دو نیم توری سمت چپ و سمت راست، تنها یک گره خالی در نیم توری سمت راست وجود دارد و یک رکورد باید در آن (در نیم توری سمت راست درج گردد) قرار گیرد تا نیم توری سمت راست به یک توری جستجوی دودوئی کامل تبدیل گردد. با توجه به مقدار رکورد x یکی از سه حالت زیر پیش می‌آید.

۱-۲ اگر $LargerItem > x$ باشد، رکورد x ، باید در نیم توری سمت راست درج شود.

۲-۲ اگر $SmallItem < x$ باشد، رکورد $LargerItem$ ، باید نیم توری سمت راست و رکورد $SmallItem$ در نیم توری سمت چپ درج شود و سپس رکوردهای x و $Prev(LargerItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargerItem$ را پر کنند.

۳-۲ اگر $SmallItem < x < LargerItem$ باشد، رکوردهای x و $LargerItem$ باید بترتیب در نیم توری های سمت چپ و سمت راست درج شوند و سپس رکورد $Prev(LargerItem)$ جای خالی $LargerItem$ را پر کند.

۳- اگر نیم توری سمت چپ، یک توری جستجوی دودوئی کامل با عمق $m-1$ و نیم توری سمت راست، توری جستجوی دودوئی کامل پر با عمق $m-2$ و گره‌های سطح آخر پر باشد. برای حفظ خواص توری جستجوی دودوئی کامل، رکورد باید در نیم توری سمت چپ درج گردد. با توجه به روی هم پوشانی دو نیم توری و با توجه به اینکه گره‌های مرز بالا، عملگر را تنها از پدر سمت چپ خود دریافت می‌کنند یک رکورد باید در نیم توری سمت راست درج شود که یکی از سه حالت زیر پیش می‌آید.

۱-۳ اگر $LargerItem > x$ باشد، رکورد x ، باید در نیم توری سمت راست درج شود.

۲-۳ اگر $SmallItem < x$ باشد، رکورد $LargerItem$ ، باید نیم توری سمت راست و رکورد $SmallItem$ در نیم توری سمت چپ درج شود و سپس رکوردهای x و $Prev(LargerItem)$ بترتیب جای خالی رکوردهای $SmallItem$ و $LargerItem$ را پر کنند.

۳-۳ اگر $SmallItem < x < LargerItem$ باشد، رکوردهای x و $LargerItem$ باید بترتیب در نیم توری های سمت چپ و سمت راست درج شوند و سپس رکورد $Prev(LargerItem)$ جای خالی $LargerItem$ را پر کند.

۴- عملگر بعد از عملگر رکورد بعد باید با دو واحد تأخیر آغاز شود. یک واحد از این تأخیر مربوط به حالت زیر می باشد. اگر کلید x مساوی کلید رکورد LargestItem یک پردازنده میانی یا مرز چپ (با توجه به شکل ۱۲) یا مساوی کلید بزرگترین رکورد یک پردازنده برگ (با توجه به شکل ۱۳) باشد رکورد بعد در رکورد LargestItem پدر سمت راست این پردازنده قرار دارد و بهمین دلیل نیاز به یک ارتباط در جهت عکس حرکت عملگرهای ساختمان داده دارد در نتیجه یک تأخیر اضافه مورد نیاز می باشد.

۵- بخاطر اینکه عملگرهای جستجو، کوچکترین کلید و بزرگترین کلید نیاز به عمل چرخش و ارتباط اضافی ندارند به بیش از یک واحد تأخیر بین این عملگرها و عملگر بعدی نیاز نمی باشد.

قضیه ۱۴: توان عملیاتی توری جستجوی دودویی کامل از مرتبه $O(1)$ است. اثبات: بدلیل اینکه تمامی عملگرها برای اجرا به ریشه ارسال می گردند و ریشه در هر زمان قادر به راه اندازی بیش از یک عملگر نمی باشد، توان عملیاتی این ماشین دیکشنری $O(1)$ است.

۶- نتیجه گیری

در این مقاله یک ساختمان داده موازی بنام توری جستجوی دودویی کامل برای یک ماشین دیکشنری بمنظور پیاده سازی در کامپیوترهای موازی با حافظه توزیع شده پیشنهاد شده است. در این ساختمان داده زمان پاسخ عملگرهای جستجو، بزرگترین کلید و کلید بعد از مرتبه $O(n)$ و زمان پاسخ عملگرهای کوچکترین کلید و حذف کوچکترین کلید از مرتبه $O(1)$ می باشد. پس از شروع عملگرهای درج، حذف، حذف کوچکترین کلید و رکورد بعد نیاز به دو واحد تأخیر برای شروع عملگر بعدی می باشد در حالیکه عملگرهای کوچکترین کلید، بزرگترین کلید و جستجو نیاز به یک واحد تأخیر دارند.

توری جستجوی دودویی کامل با سه ماشین Young و Youn، ماشین Schmeck و Schroder و ماشین Dehne و Santoro مقایسه گردیده است که نتایج آن در جدول ۱ آمده است. با توجه به این جدول و مباحث انجام گرفته مشخص می گردد که ماشین پیشنهادی در این مقاله بهتر از سه ماشین دیگر عمل می کند. زیرا:

۱- زمان پاسخ توری جستجوی دودویی کامل حدود نصف زمان پاسخ توری هشت و شش ضلعی است.

۲- با توجه به اینکه شبکه پخش عملگرها یک مسیر یکتا بین ریشه و گره های دیگر توری جستجوی دودویی کامل ایجاد می کند، عملگرهای ماشین دیکشنری بصورت متوالی وارد گره ها می گردند و هیچ حالتی اتفاق نمی افتد که یک عملگر از دو مسیر مختلف و همزمان به یک پردازنده برسد و این باعث می گردد تا سازگاری در توری جستجوی دودویی کامل بسادگی قابل پیاده سازی باشد.

۳- دریافت عملگرها و ارسال جوابها در توری جستجوی دودویی کامل از طریق دو شبکه غیر مجزا انجام می شود که باعث کم شدن ترافیک و برخورد پیغامها می گردد.

۴- توری جستجوی دودویی کامل قابل جا دادن در ابر مکعب می باشد. روشی برای جای دادن توری هشت و شش ضلعی در ابر مکعب تا آنجایی که نویسندگان این مقاله اطلاع دارند هنوز گزارش نشده است.

۵- تعداد اتصالات موجود در توری جستجوی دودویی کامل کمتر از توری هشت و شش ضلعی می باشد که پیاده سازی VLSI آنرا ساده تر می نماید. معیارهایی که توسط x نشان داده شده اند گزارش نشده اند.

قضیه ۱۰: عملگر حذف رکورد دارای کوچکترین کلید خواص توری جستجوی دودویی را حفظ می کند.

اثبات: فرض کنید که K_{min} کوچکترین کلید موجود در توری جستجوی دودویی کامل باشد. با توجه به توضیح و قطعه کد داده شده برای این عملگر، روشن است که این عملگر فقط از قسمت کوچکی از عملگر حذف که مربوط به حالت Delete (K_{min}) می باشد استفاده میکند و با توجه به قضیه ۷ این عملگر خصوصیات توری جستجوی دودویی کامل را حفظ می کند.

قضیه ۱۱: زمان پاسخ عملگرهای رکورد بعد و رکورد دارای بزرگترین کلید $O(n)$ است.

اثبات: عملگرهای فوق از دو مرحله جستجو و تولید جواب تشکیل می شوند و زمان پاسخ آنها از مجموع زمان اجرای این دو مرحله تشکیل می گردد. با توجه به قضیه ۵ زمان پاسخ عملگر جستجو $O(n)$ است و بدلیل اینکه تولید جواب نیاز به یک ارتباط دارد بنابراین زمان لازم برای تولید جواب $O(1)$ است در نتیجه زمان پاسخ این عملگرها $O(n)$ است.

قضیه ۱۲: زمان پاسخ عملگرهای رکورد دارای کوچکترین کلید و حذف رکورد دارای کوچکترین رکورد $O(1)$ می باشد.

اثبات: با توجه به اینکه رکورد دارای کوچکترین کلید در پردازنده ریشه قرار دارد زمان پاسخ این دو عملگر هر کدام $O(1)$ است.

۵- سازگاری در ساختمان داده موازی

در ادامه بحث سازگاری در ساختمان داده های موازی تعریف می شود و سپس سازگاری در توری جستجوی کامل مورد بررسی قرار خواهد گرفت.

ساختمان داده موازی سازگار: یک ساختمان داده موازی را سازگار می گویند اگر نتیجه تولید شده توسط اجرای موازی هر ترتیب از عملگرها روی ساختمان داده با نتیجه تولید شده توسط اجرای متوالی آن ترتیب روی ساختمان داده یکسان باشد.

قضیه ۱۳: پیاده سازی توری جستجوی دودویی کامل روی توری یا ابر مکعب سازگار نیست.

اثبات: فرض کنید عملگر درج و سپس عملگر جستجو بترتیب وارد توری شوند. اگر عملگر درج سبب چرخش در توری شود نتیجه عملگر جستجو قبل و بعد از اتمام عملگر درج می تواند متفاوت باشد که منجر به ناسازگاری ساختمان داده خواهد شد.

در ادامه بحث رویه ای بیان می گردد که با اجرای آن سازگاری پیاده سازی توری جستجوی دودویی کامل روی توری یا ابر مکعب تضمین می گردد. این رویه بصورت زیر می باشد. عملگرهای دیکشنری برای اجرا به ریشه توری فرستاده می شوند و در پردازنده ریشه در یک صف قرار می گیرند. پردازنده ریشه علاوه بر قراردادن عملگر در انتهای صف، تأخیر مورد نیاز برای شروع عملگر بعدی را محاسبه و با این عملگر در صف حفظ میکند. تأخیر مورد نیاز برای عملگر بعد بصورت زیر خواهد بود.

۱- در صورتیکه عملگر بعد از عملگر (p, x) Next یکی از عملگرهای پرس و جو باشد در اینصورت این عملگر می تواند بدون تأخیر اجرا شود.

۲- عملگر بعد از عملگر درج بایستی با دو تأخیر آغاز شود. یک واحد از این تأخیر بخاطر عمل چرخشی است که در بعضی موارد در موقع انجام عملگر درج به آن نیاز می باشد.

۳- عملگر بعد از عملگرهای حذف و حذف رکورد دارای کوچکترین رکورد بایستی با دو واحد تأخیر آغاز شود. یک واحد از این تأخیر بخاطر عمل چرخشی است که در بعضی موارد در موقع انجام عملگر درج به آن نیاز می باشد.

- [12] F. Dehne and N. Santoro, "Optimal VLSI Dictionary Machines on Meshes", *Proc. of Int. Conf. on Parallel Processing*, pp. 832-840, August 1987.
- [13] C. S. Ellis, "Concurrent Search and Insertion in 2-3 Trees", *Acta Informatica*, Vol. 14, pp. 63-86, 1980.
- [14] C. S. Ellis, "Concurrent Search and Insertion in AVL Trees", *IEEE Trans. on Computers*, Vol. 29, No. 9, pp. 811-817, 1980.
- [15] A. L. Fisher, "Dictionary Machine With Small Number of Processors", *Proc. of Int. Symp. on Computer Architecture*, University of Michigan, Ann arbor, Michigan, pp. 151-156, 1984.
- [16] J. Ghosh, S. K. Das, and A. John, "Concurrent Processing of Lineary Ordered Data Structures on Hypercube Multicomputers", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 9, pp. 898-911, 1994.
- [17] H. T. Kung and P. L. Lehman, "Concurrent Manipulation of Binary Search Trees", *ACM Trans. on Database Systems*, Vol. 5, No. 3, pp. 354-382, 1980.
- [18] S. K. Lee and H. Choi, "Embedding of Complete Binary Trees into Meshs with Row-Column Routing", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 5, pp. 493-497, 1996.
- [19] P. L. Lehman and S. B. Yao, "Efficient Locking for Concurrent Operations on B-Trees", *ACM Trans. on Database System*, Vol. 6, No. 4, pp. 650-670, 1981.
- [20] H. O. Li and D. K. Probst, "Optimal VLSI Dictionary Machines Without Compress Instructions", *IEEE Trans. on Computers*, Vol. 39, No. 5, pp. 676-, 1990.
- [21] M. R. Meybodi, "Implementating Priority Queue on Hypercube Machine", *Proc. of Fourth Parallel Processing Symp.*, University of Callifornia, Irvine, pp. 85-111, April 1990.
- [22] M. R. Meybodi, "New Designs for Priority Queue Machine", *Proc. of PARABASE-90*, Miami, Florida, pp. 123-121, May 1990.
- [23] M. R. Meybodi, "Concurrent Data Structures for Hypercube Machine", *Lecture Notes in Computer Science: Parallel Architecture and Languages*, Springer Verlage, pp. 559-577, 1992.
- [24] M. R. Meybodi, "Binary Search Mesh: A Concurrent Data Structure for Hypercube Computer", *Proc. of ICEE-93*, Amirkabir University, Tehran, Iran, pp. 601-607, May 1993.
- [25] M. R. Meybodi, "Tree Structured Dictionary Machines for VLSI", *Proc. of 23th Annual Modeling and Simulation Conf.*, School of Eng., University of Pittsburg, Pittsburg, Pennsylvania, pp. 703-724, May 1992.
- [26] M. R. Meybodi, "Banyan Heap Machine", *Proc. of IEEE 6th Int. Parallel Processing Sympos.*, Los Almitos, CA, pp. 224-231, 1992.
- [27] A. R. Omondi and J. D. Brock, "Implementing a Dictionary on Hypercube Machine", *Proc. of Int. Conf. on Parallel Processing*, pp. 707-709, 1987.

عملگر	Insert	Max	XMin	Next	Min	Delete	Search
توری جستجوی دودویی کامل	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)
مانسین Young و Youn	O(1)	*	*	*	O(1)	O(1)	O(1)
مانسین Schmeck و Schroder	O(1)	*	*	*	O(1)	O(1)	O(1)
مانسین Santoro و Dehne	O(1)	*	*	*	O(1)	O(1)	O(1)

جدول ۱- مقایسه بین فاصله شروع دو عملگر در توری جستجوی دودویی کامل و مانسینهای دیگر

عملگر	Search	Max	XMin	Next	Min
توری جستجوی دودویی کامل	O(n)	O(n)	O(1)	O(n)	O(1)
مانسین Young و Youn	O(n)	*	*	*	O(1)
مانسین Schmeck و Schroder	O(n)	*	*	*	O(1)
مانسین Santoro و Dehne	O(n)	*	*	*	O(1)

جدول ۲- مقایسه بین زمان پاسخ عملگرها در توری جستجوی دودویی کامل و مانسین های دیگر

مراجع

- [1] A. V. Aho, J. E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithm*, Addison Wesley, 1974.
- [2] M. J. Atallah, F. Dehne, R. Millers, and A. Rau-Chaplin, "Multisearch Techniques: Parallel Data Structures on Mesh-Connected Computers", *Journal of Parallel and Distributed Computing*, Vol. 20, pp. 1-13, 1994.
- [3] M. J. Atallah and S. R. Kosaraju, "A Generalized Dictionary Machine for VLSI", *IEEE Trans. on Computers*, Vol. 34, No. 2, pp. 151-155, 1985.
- [4] H. Beigy and M. R. Meybodi, "Notes on Binary Search Mesh: A Concurrent Data Structure for Hypercube Computer", Technical Report 3CE98, Computer Eng. Dept., Amirkabir University, Tehran, Iran, 1998.
- [5] H. Beigy and M. R. Meybodi, "Complete Binary Search Mesh: A Concurrent Data Structure for Distributed Memory Parallel Computer", Technical Report 4CE98, Computer Eng. Dept., Amirkabir University; Tehran, Iran, 1998.
- [6] J. L. Bentley and H. T. Kung, "A Tree Machine for Searching Problems", *Proc. of Int. Conf. on Parallel Processing*, August 1979.
- [7] J. Biswas and J. C. Browne, "Simultaneous Update of Priority Structures", *Proc. of Int. Conf. on Parallel Processing*, pp. 124-131, 1987.
- [8] J. E. Brandenburg and D. S. Scott, "Embedding of Communication Tree and Grid into Hypercube", IPSC Technical Reports, 1986.
- [9] D. P. Brestekas and J. N. Tsitsiklis, *Parallel and Distributed Computation*, Printice-Hall, 1989.
- [10] W. J. Dally, *A VLSI Architecture for Concurrent Data Structures*, Kluwer Academic Publishers, 1987.
- [11] F. Dehne and N. Santoro, "An Improved New Embedding for VLSI Dictionary Machines on Meshes", *Proc. of Int. Sympos. Computer Applications in Design, Simulation, and Analysis*, pp. 113-116, 1989.



حمید بیگی تحصیلات خود را در مقاطع کارشناسی و کارشناسی ارشد مهندسی کامپیوتر بترتیب در سالهای ۱۳۷۱ و ۱۳۷۴ از دانشگاه شیراز به پایان رساند و هم‌اکنون در مقطع دکترای مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر مشغول به تحصیل است. زمینه‌های تحقیقاتی مورد علاقه او عبارتند از: الگوریتم‌های موازی،

پردازش موازی، محاسبات نرم و کاربردهای آن و شبکه‌های کامپیوتری. نامبرده بیش از ۵۰ مقاله در مجلات و کنفرانس‌های بین‌المللی در زمینه‌های فوق منتشر کرده است. آدرس پست الکترونیکی ایشان عبارتست از:

beigy@ce.aku.ac.ir



محمد رضا میسبدی تحصیلات خود را در مقاطع کارشناسی و کارشناسی ارشد اقتصاد بترتیب در سالهای ۱۳۵۲ و ۱۳۵۶ از دانشگاه شهید بهشتی و در مقاطع کارشناسی ارشد و دکتری علوم کامپیوتر بترتیب در سالهای ۱۳۵۹ و ۱۳۶۲ از دانشگاه اوکلاهامای آمریکا به پایان رسانده است و هم‌اکنون استاد دانشکده مهندسی

کامپیوتر دانشگاه صنعتی امیرکبیر می‌باشد. نامبرده قبل از پیوستنش به دانشگاه صنعتی امیرکبیر در سالهای ۱۳۶۲ الی ۱۳۶۴ استادیار دانشگاه میشیگان غربی و در سالهای ۱۳۶۴ الی ۱۳۷۰ دانشیار دانشگاه اوهایو در ایالات متحده آمریکا بوده است. زمینه‌های تحقیقاتی مورد علاقه ایشان عبارتند از: الگوریتم‌های موازی، پردازش موازی، محاسبات نرم و کاربردهای آن، شبکه‌های کامپیوتری و مهندسی نرم‌افزار. آدرس پست الکترونیکی نامبرده

به قرار زیر است: meybodi@ce.aku.ac.ir

- [28] T. A. Ottman, A. L. Rosenberge, and L. J. Stockmeyer, "A Dictionary Machine (for VLSI)", *IEEE Trans. on Computers*, Vol. 31, No. 9, pp. 892-897, 1982.
- [29] V. N. Rao and V. Kumar, "Concurrent Access to Priority Queue", *IEEE Trans. on Computers*, Vol. 37, No. 12, pp. 1657-1665, 1988.
- [30] Y. Saad and M. Schultz, "Topological Properties of Hypercubes", *IEEE Trans. on Computers*, Vol. 37, No. 7, pp. 867-872, 1988.
- [31] H. Schemeck and H. Schroder, "Dictionary Machines for Different Models of VLSI", *IEEE Trans. on Computers*, Vol. 34, No. 5, pp. 472-475, 1985.
- [32] A. K. Somani and V. K. Agarwal, "An Unsorted VLSI Dictionary Machine", *Proc. of 1983 Canadian VLSI Conf.*, University of Waterloo, Waterloo, 1983.
- [33] A. K. Somani and V. K. Agarwal, "An Efficient VLSI Dictionary Machine", *Proc. of 11th Annual Int. Symp. on Computer Architecture*, University of Michigan, Ann arbor, Michigan, pp. 142-150.
- [34] H. Y. Youn and J. Y. Lee, "An Efficient Dictionary Machine Using Hexagonal Processor Arrays", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 3, pp. 226-273, 1996.
- [35] W. Zhang and R. E. Korf, "Parallel Heap Operations on EREW PRAM", *Journal of Parallel and Distributed Computing*, Vol. 20, pp. 248-255, 1994.
- [36] D. E. Knuth, *The Art of Computer Programming*, Volume 3, Addison Wesley, 1975.

- ¹ Insert
- ² Delete
- ³ Extract minimum
- ⁴ Extract maximum
- ⁵ Redundant
- ⁶ Modify operators
- ⁷ Query operations
- ⁸ Balance cube
- ⁹ Sorted chain
- ¹⁰ Banyan heap
- ¹¹ Initiation interval
- ¹² Response time
- ¹³ Throughput
- ¹⁴ Mesh
- ¹⁵ Broadcast net
- ¹⁶ Snak net
- ¹⁷ IO processor
- ¹⁸ Linear net
- ¹⁹ Spanning net
- ²⁰ Binary search mesh (BSM)
- ²¹ Balanced binary search mesh (BBSM)
- ²² Complete binary search mesh (CBSM)