ساختار فایل ها

(ذخیره و بازیابی اطلاعات)

ترجمه:

مهندس عین ا... جعفرنژاد قمي

ابراهیم محرابي

تعداد واحد: 3

فهرست جلسات

جلسه اول: آشنايي با طراحي و مشخصات ساختار فايلها، عمليات مهم پردازش فايل، حافظه جانبي و نرم افزار سيستم

جلسه دوم: ادامه مبحث حافظه جانبي و نرم افزار سيستم

جلسه سوم: ادامه مبحث حافظه جانبي و نرم افزار سيستم

جلسه چهارم: مفاهيم اساسي ساختار فايل، مديريت فايلهايي از ركوردها

جلسه پنجم: ادامه مبحث مديريت فايلهايي از ركوردها

جلسه ششم: ادامه مبحث مديريت فايلهايي از ركوردها، سازماندهي فايلها براي كارايي

فهرست جلسات

جلسه هفتم: ادامه مبحث سازماندهي فايلها براي كارايي، شاخص گذاري

جلسه هشتم: ادامه مبحث شاخص گذاري

جلسه نهم: ادامه مبحث شاخص گذاري، پردازش كمک ترتيبي و مرتب سازي فايل هاي بزرگ

جلسه دهم: ادامه مبحث پردازش كمک ترتيبي و مرتب سازي فايل هاي بزرگ

جلسه يازدهم: ادامه مبحث پردازش كمک ترتيبي و مرتب سازي فايلهاي بزرگ، شاخص بندي چند سطحي و درختهاي B

جلسه دوازدهم: ادامه مبحث شاخص بندي چند سطحي و درختهاي B

فهرست جلسات

جلسه سيزدهم: دستيابي به فايل هاي ترتيبي شاخص دار و درخت هاي +B

جلسه چهاردهم: ادامه مبحث دستيابي به فايل هاي ترتيبي شاخص دار و درخت هاي +B ، درهم سازي

جلسه پانزدهم: ادامه مبحث درهم سازي

جلسه شانزدهم: ادامه مبحث درهم سازي، درهم سازي قابل توسعه

جلسه اول

- آشنايي با طراحي و مشخصات ساختار فايلها
- عمليات مهم پردازش فايل
- حافظه جانبي و نرم‌افزار سيستم

آشنايي با طراحي و مشخصات ساختار فايلها

ساختار فايل تركيبي از نحوه نمايش داده ها در فايل ها و عمليات لازم براي دستيابي به داده ها است. ساختار فايل به برنامه كاربردي اين امكان را مي دهد كه داده ها را بخواند ،بنويسد و اصلاح كند.

طي سه دهه اخير با بررسي تكامل ساختارهاي فايل مشاهده مي كنيم كه طراحي ساختار فايل ابتدا از ترتيبي شروع شد ،سپس به ساختارهاي درختي رسيد و سرانجام دستيابي مستقيم مطرح شد. در همه اين موارد مشكلات و ابزارهاي طراحي مشابهي مشاهده شده است. اين ابزارها را ابزارهاي مفهومي مي نامند كه روش هايي براي تنظيم و حل يک مسئله طراحي اند.

يک مشکل اصلي در توصيف کلاس هايي که بتوان براي طراحي ساختار فايل آنها را به کار برد ، آن است که اين کلاس ها پيچيده و در حال رشد هستند. کلاس هاي جديد غالباً شکل اصلاح شده يا توسعه يافته اي از کلاس ها ديگر بوده ،جزئيات ارائه داده ها و عمليات باز هم پيچيده تر مي شود.

در يک سيستم اطلاعاتي شيء گرا محتوا و رفتار داده ها ، در يک طراحي منسجم مي شود. اشياي سيستم به کلاس هاي اشيايي با ويژگي هاي مشترک تقسيم مي شوند. هر کلاس توسط اعضاي (members) خود توصيف مي شود که يا صفات داده ها (عضوهاي داده اي) يا توابع (توابع عضو يا متدها) هستند.

مشكل اصلي در طراحي ساختار فايل زمان نسبتاً زيادي است كه براي گرفتن تطلاعات از ديسک مورد نياز است. در همه طراحي هاي ساختار فايل آنچه مورد توجه است به حد اقل رساندن دفعات دستيابي به ديسک و به حد اكثر رساندن احتمال وجود اطلاعات مورد نظر برنامه كاربردي در حافظه است.

عملیات مهم پردازش فایل

هنگامي كه درباره فايلي روي يک ديسک يا نوار صحبت مي كنيم ،منظور ما مجموعه اي از بايت ها است كه در آنجا ذخيره شده اند. فايل در اين معنا داراي موجوديت فيزيكي است. يک ديسک ممكن است حاوي صدها و حتي هزاران فايل فيزيكي باشد.

برنامه غالباً نمي داند بايت ها از كجا مي آيند يا به كجا مي روند ، اين را مي داند كه كدام خط را مورد استفاده قرار داده است. اين خطوط را معمولاً فايل منطقي مي نامند تا از فايل فيزيكي ،كه روي ديسک يا نوار قرار دارد متمايز گردد.

هنگامي كه شناسه (identifier) فايل منطقي با دستگاه يا فايل فيزيكي ارتباط پيدا كرد ،بايد اعلام كنيم كه مي خواهيم با فايل چه كنيم :

۱) باز كردن يک فايل موجود
۲) فيلي خا فيزيكي فايل جديد و حذف محتويات موجود در

هنگامي كه برنامه اي به صورت عادي پايان مي يابد فايل ها معمو
لاً به طور خودكار بسته مي شوند. در نتيجه اجراي يک دستور

خواندن و نوشتن در پردازش فايل اهميت بنيادي دارند ،اينها اعمالي هستند كه پردازش فايل را به يک عمل ورودي/خروجي تبديل مي كنند.

براي دستيابي آسان به تعداد زياد از فايل ها كامپيوتر روشي براي سازماندهي فايل ها دارد. در يونيكس اين روش سيستم فايل ناميده مي شود. چون هر نام فايل در سيستم يونيكس بخشي از سيستم فايلي است كه با ريشه آغاز مي شود ،هر فايل را مي توان انحصاراً با دادن نام مسير آن شناسايي كرد.

يكي از پر قدرت ترين ايده ها در يونيكس تعريفي است كه از فايل مي شود. در يونيكس فايل مجموعه اي از بايت ها است و چگونگي و محل ذخيره آنها هم مهم نيست. همچنين مهم نيست كه اين بايت ها از كجا مي آيند. اين نگرش معمولي به فايل موجب مي شو كاري را كه در سيستم عامل هاي ديگر به زحمت انجام مي شوند ، در اين سيستم عامل به راحتي انجام پذير باشد.

يونيكس فرمان هاي بسياري براي دستكاري فايل ها دارد
كه عبارتند از :

cat, tail, cp, mv, rm, chmod, ls, mkdir, rmdir

حافظه جانبي و نرم افزار سيستم

دستگاه هاي حافظه جانبي ،با حافظه تفاوت بسيار دارند. همان طور كه پيش از اين نيز متذكر شديم يک اختلاف از آنجا ناشي مي شود كه در دستگاه هاي حافظه جانبي زمان بيشتري براي دستيابي مورد نياز است. اختلاف ديگر آن است كه همه دستيابي ها يكسان نيستند.

ديسک ها انواع مختلفي دارند :

۱) ديسک هاي سخت (hard disks)

۲) ديسک هاي فلاپي (floppy disks)

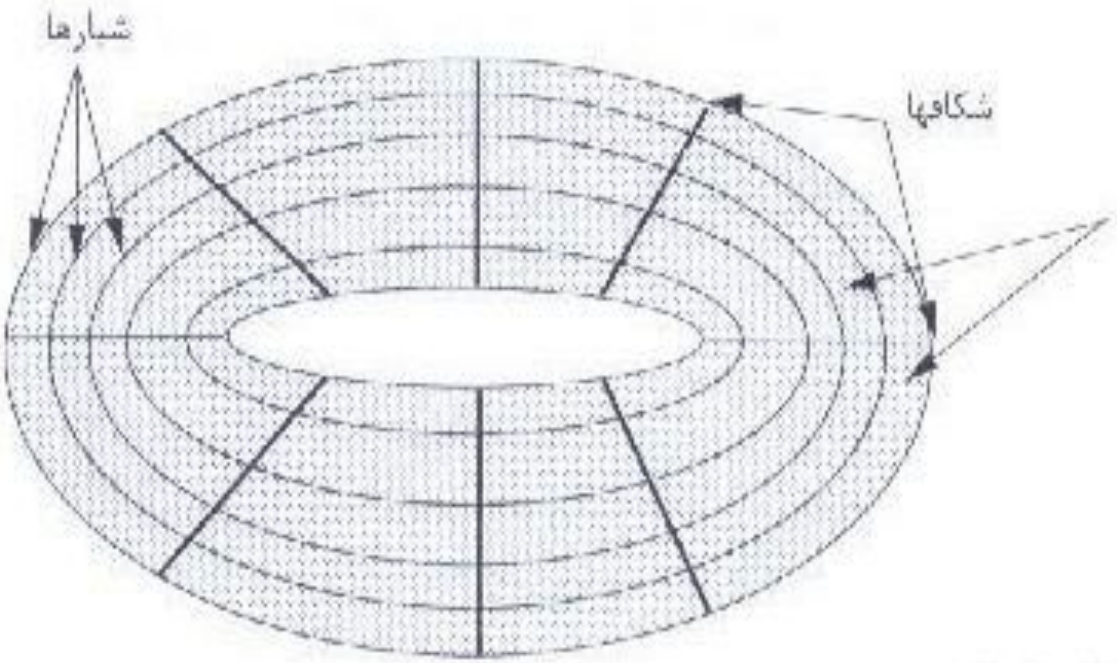
۳) کارتريج ديسک

۴) ديسک هاي نوري

جلسه دوم

- ادامه مبحث حافظه جانبي و نرم افزار سيستم

اطلاعات ذخيره شده روي ديسک ،در سطح يک يا چند صفحه نگهداري مي شود. ترتيب کار به صورتي است که اطلاعات به صورت شيار هايي (tracks) روي سطح ديسک نگهداري مي شوند. هر شيار غالباً به چند سکتور (sector) تقسيم مي شود. سکتور کوچکترين بخشي از ديسک است که قابل آدرس دهي است.

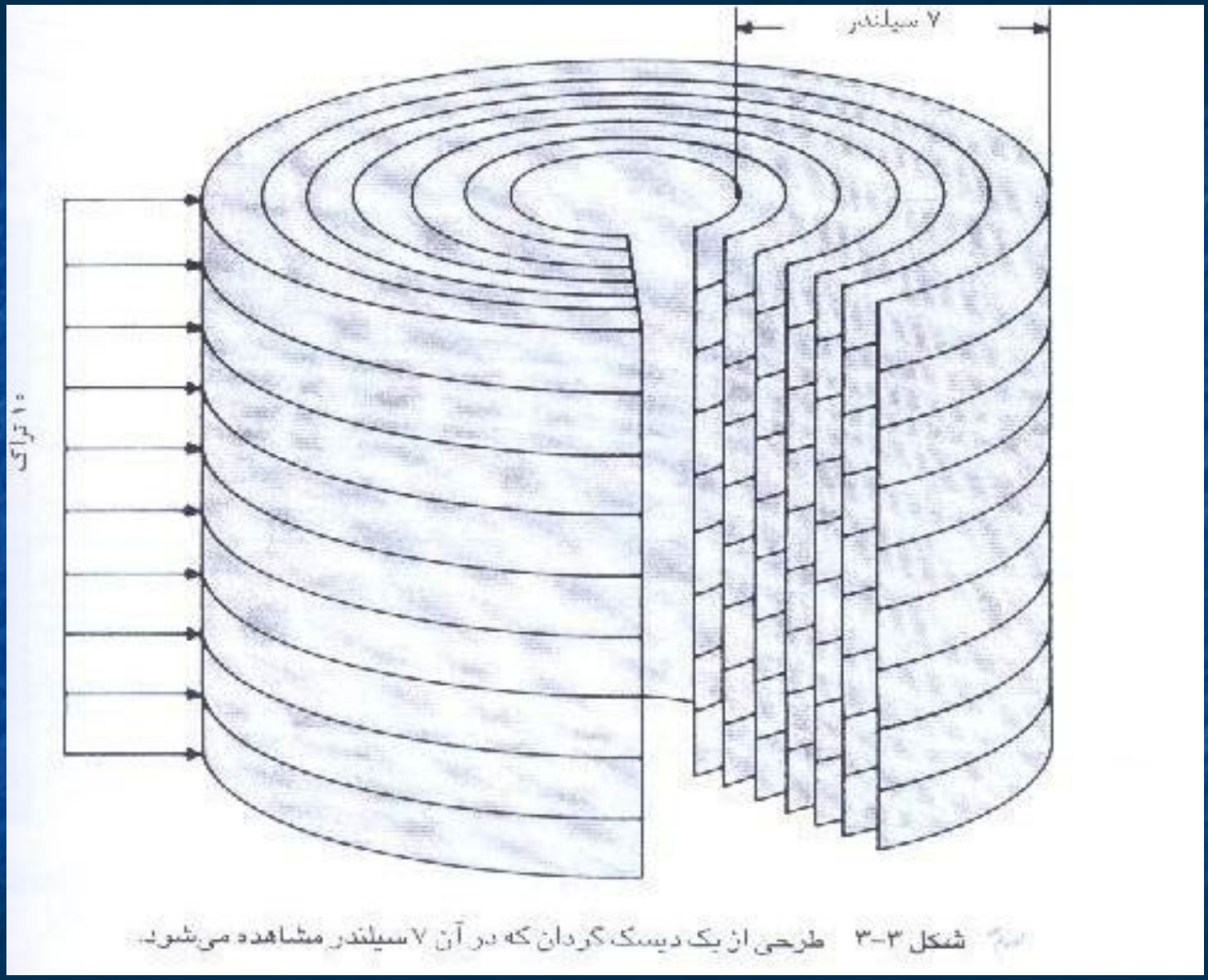


شکل ۲-۳

سطح دیسک با شیارها و قطاعهای آن.

دیسک گردان ها

معمو
[illegible] سیلندر [illegible]
به همه اطلاعات روي یک سیلندر مي توان بدون حرکت دادن بازوي نگهدارنده هد (head) خواندن/نوشتن دستیابي داشت. حرکت این بازو پیگرد (seeking) نام دارد.



شکل ۳-۳ طرحی از یک دیسک گردان که در آن ۷ سیلندر مشاهده می‌شود.

ظرفيت ديسک تابعي از تعداد سيلندرها ،تعدا شيارها به ازاي هر سيلندر و ظرفيت هر شيار است.

دو روش براي سازماندهي داده ها بر روي ديسک وجود دارد :

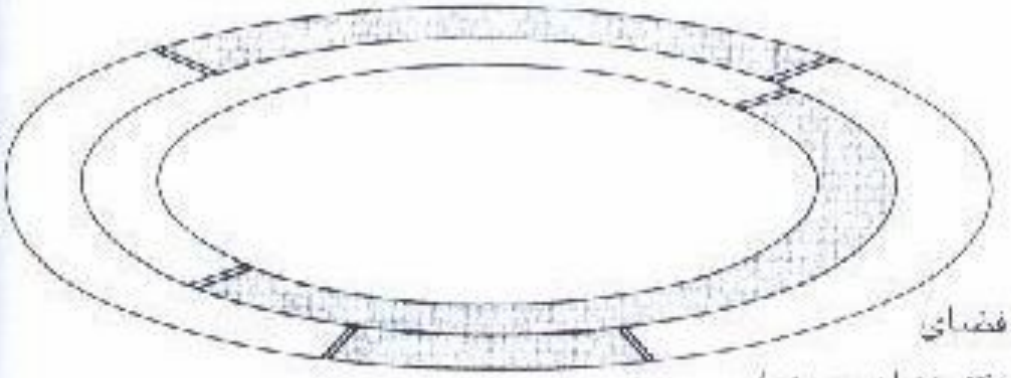
۱) بر اساس سکتور
۲) بر اساس بلوک هاي تعريف شده توسط کاربر

کلاستر عبارت از تعداد ثابتي از سکتورهاي پيوسته است.

مديريت فايل براي در نظر گرفتن فايل به عنوان مجموعه اي از کلاسترها و در عين حال حفظ حالت سکتوري ،سکتورهاي منطقي را به کلاسترهاي فيزيکي که به آنها تعلق دارد ،به وسيله جدول تخصيص فايل (FAT) ارتباط مي دهد.

اگر فضاي زيادي روي ديسک باشد ، ممکن است بتوان کاري کرد که فايل به طور کامل از کلاسترهاي پيوسته تشکيل شود. در چنين موردي گفته مي شود که فايل حاوي يک حد (extent) است.

(الف)



(ب)

شکل ۶-۳

بلوک‌های فایل (ناحیه سایه زده شده، فضای روی دیسک را که یک فایل اشغال می‌کند نشان می‌دهد).

اتلاف فضا در داخل يک سکتور را پراکندگي داخلي مي نامند. سازماندهي بلوک ها مشکلات پوشايي سکتورها و پراکندگي را ندارد ،زيرا اندازه بلاک ها مي تواند تغيير کند تا سازماندهي منطقي داده ها امکان پذير شود.

بلوک

معمو

لاً طوري سازماندهي مي شود که تعداد مناسبي از رکوردهاي م

در الگوهاي آدرس دهي بلاكي هر بلوک از داده ها معمولاً با يک يا چند زير بلوک (subblock) همراه است که حاوي اطلاعات اضافي راجع به بلوک داده ها است. معمولاً يک زيربلوک شمارشي وجود دارد که تعداد بايت هاي موجود در بلوک مربوط را نگه مي دارد. همچنين ممکن است که يک زيربلوک کليد ،حاوي کليد مربوط به آخرين رکورد در بلوک داده ها باشد.

هم بلاک ها و هم سکتورها نیازمند آنند که مقدار معيني از فضاي ديسک را به شکل سربار غير داده اي اشغال کنند. بخشي از اين سربار از اطلاعاتي تشکيل مي شود که طي فرمت کردن ،بر روي ديسک نگهداري مي شود. فرمت کردن ،پيش از آنکه ديسک بتواند مورد استفاده قرار گيرد ،صورت مي پذيرد.

فرمت كردن موجب مي شود تا شكاف (gap) و علامت هاي هم زمان سازي ،بين فيلدهاي اطلاعاتي قرار داده شود تا مكانيزم خواندن/نوشتن ، بين آنها تمايز قائل شود.

دستيابي به ديسک را مي توان به سه عمل فيزيکي متمايز تقسيم کرد که هر يک هزينه خود را دارد :

۱) زمان پيگرد (seek time)
۲) تأخير چرخشي (rotational delay)
۳) زمان انتقال (transfer time)

جلسه سوم

- ادامه مبحث حافظه جانبي و نرم افزار سيستم

زمان پیگرد عبارت است از زمان لازم انتقال بازوي دستيابي ،به سيلندر مناسب.

تأخیر در چرخش عبارت است از زمان لازم براي چرخش دیسک ،تا سکتور مورد نظر زیر هد خواندن/نوشتن قرار گیرد.

علي رغم افزايش كارايي ديسک ها ،سرعت شبكه ها به حدي بالا رفته است كه دستيابي به ديسک غالباً تنگناي مهمي در كل سيستم I/O به شمار مي رود.

چند تكنيک براي مقابله با اين مشكل وجود دارد :

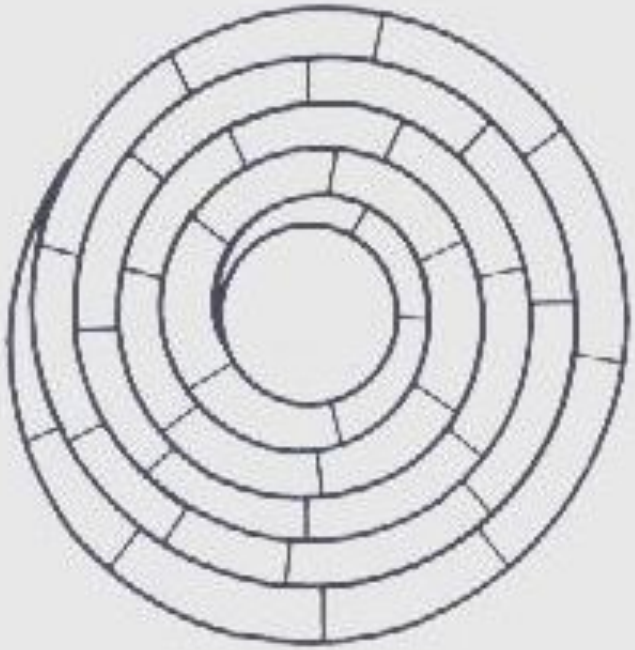
۱) نواربندي
۲) استفاده از ديسک RAM
۳) حافظه نهان

نوارهاي مغناطيسي به دستگاه هايي تعلق دارند كه دستيابي مستقيم به داده ها را فراهم نمي آورند ولي دستيابي ترتيبي به سرعت انجام مي شود. نوارها فشرده اند ،شرايط محيطي متفاوت را خوب تحمل مي كنند،نگهداري و حمل آنها آسان است و ارزانتر از ديسک ها هستند.

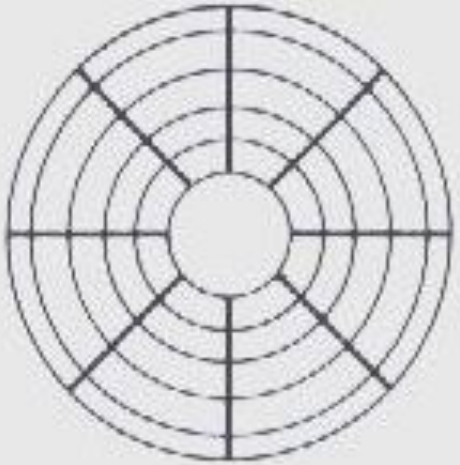
نقاط قوت CD-ROM شامل ظرفيت ذخيره سازي بالا،بهاي كم و دوام آن است. نقطه ضعف اصلي آن اين است كه جستجو در CD-ROM بسيار كند است،يعني غالباً هر جستجو نيم تا يک ثانيه طول مي كشد.

در قالب سرعت خطي ثابت (CLV) ،سرعت چرخش ديسک هنگام خواندن لبه هاي بيروني ،کندتر از هنگام خواندن لبه هاي داخلي است.

در سرعت زاويه اي ثابت (CAV) ،ديسک با شيارهاي متحدالمرکز و سکتورهاي مدور خود ،داده ها را با تراکم کمتري در شيارهاي خارجي نسبت به شيارهاي داخلي مي نويسد.



سرعت خطی ثابت



سرعت زاویه‌ای ثابت

شکل ۳ ۱۳ ضبط CAV و CLV.

بافر I/O سيستم ،به مديريت فايل اين امكان را مي دهد تا داده ها را در واحدهايي به اندازه سكتور يا بلوک بخواند يا بنويسد.

عمل كنترل عمليات ديسک توسط دستگاهي انجام مي شود كه كنترلگر ديسک ناميده مي شود.

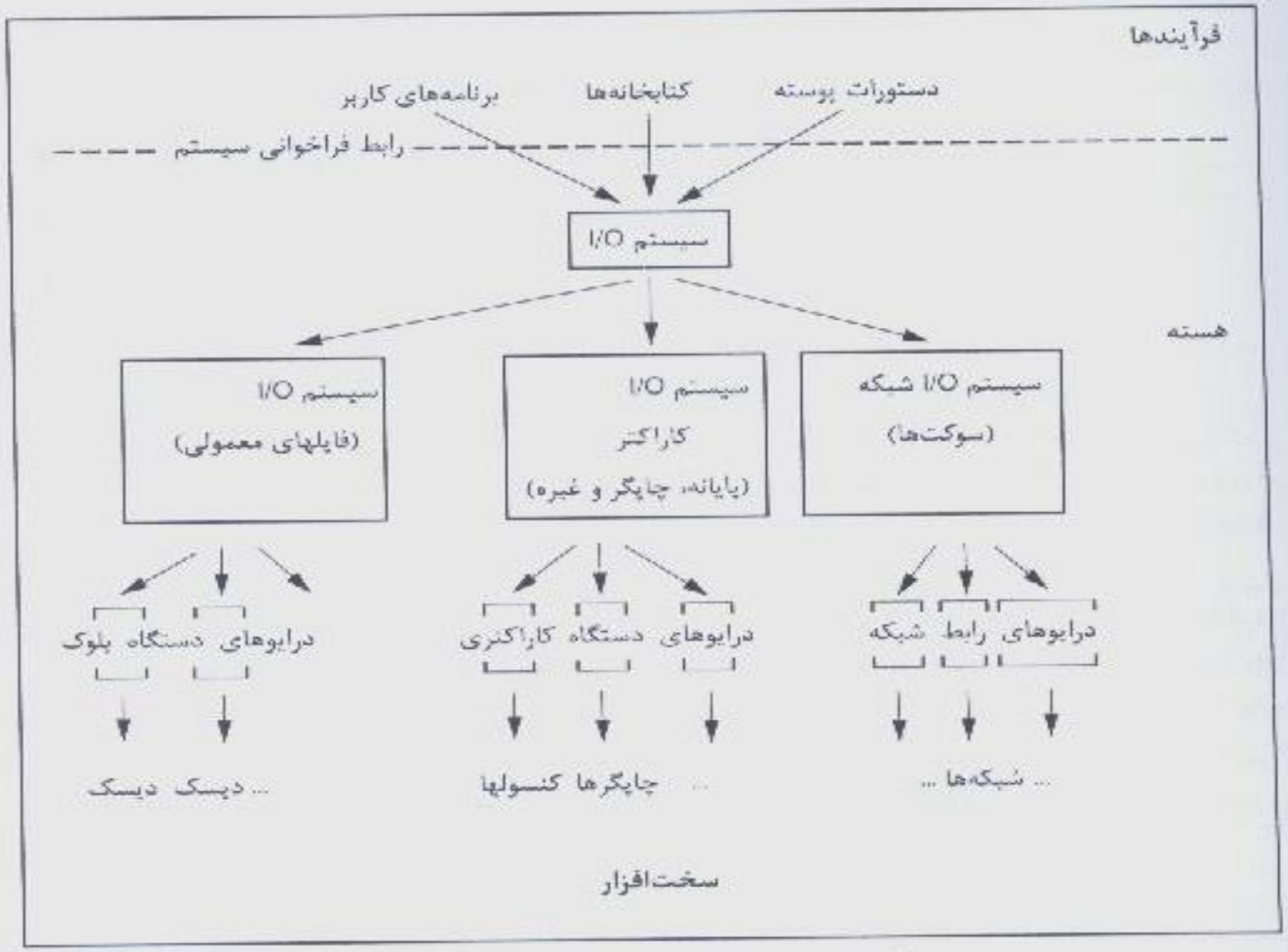
سيستم هاي I/O تقريباً هميشه حداقل دو بافر دارند. يكي براي ورودي و ديگري براي خروجي. برخي سيستم هاي فايل از يک طرح بافردهي موسوم به استخر بافري (buffer spooling) استفاده مي كنند.

با پراکنش ورودي ،با يک بار خواندن ،نه يک بار بافر بلکه مجموعه اي از بافرهايي که داده هاي يک بلوک بايد در آن ها پخش شود شناسايي مي شود.

با تمرکز خروجي ،چند بافر را مي توان گردهم آورد و يکباره بر روي همه آنها نوشت ،بدين ترتيب لازم نيست آنها را در يک بافر خروجي کپي کرد.

هسته به نوبت به اين چهار جدول زير مراجعه مي كند تا اطلاعاتي را كه براي نوشتن در فايل موجود در ديسک نياز دارد به دست آورد :

۱) جدول توصيف گر فايل
۲) جدول فايل هاي باز
۳) جدول تخصيص فايل
۴) جدول گره هاي انديسي



شکل ۲۳-۳ ساختار I/O هسته.

اشاره گري از يک فهرست به اينود فايل را اتصال سخت (hard link) مي نامند و اتصال نرم (soft link) يا اتصال سمبوليک ، نام فايل را به جاي فايل واقعي ،به يک نام فايل ديگر مرتبط مي سازد.

سه نوع سیستم I/O متفاوت داریم :

۱) سیستم I/O بلوکي
۲) سیستم I/O کاراکتري
۳) سیستم I/O شبکه اي

براي هر دستگاه جانبي مجموعه اي از روال هاي جداگانه وجود دارد كه راه انداز دستگاه ناميده مي شود.

جلسه چهارم

- مفاهيم اساسي ساختار فايل
- مديريت فايلهايي از ركوردها

مفاهيم اساسي ساختار فايل

واحد اصلي داده ها،فيلد است كه حاوي يک مقدار داده است.

فيلدها به صورت مجموعه اي از داده ها يا به صورت كپي هاي متعددي از يک فيلد (آرايه) يا ليستي از فيلدهاي متفاوت (ركورد) سازماندهي مي شوند.

هنگامي كه ركوردي در حافظه نگهداري شد آن را يک شيء و فيلدهاي آن را اعضاي آن مي نامند.

راههاي فراواني براي افزودن ساختار به فايل وجود دارد تا هويت فيلد حفظ شود.

چهار روش متداول عبارتند از :

۱) ثابت كردن طول فيلدها
۲) قرار دادن نشانگر طول فيلد در ابتداي هر فيلد
۳) جدا كردن فيلدها با فاصل
۴) استفاده از عبارت كليدي براي شناسايي فيلدها

ركورد مجموعه اي از فيلد ها است و مجموعه اي از ركوردها فايل را نشان مي دهند.

بعضي از روش هاي سازماندهي رکوردهاي فايل عبارتند از :

۱) قابل پيش بيني کردن طول رکوردها بر حسب بايت
۲) قابل پيش بيني کردن طول رکوردها بر حسب فيلدها
۳) شروع هر رکورد با نشانگر طول
۴) استفاده از انديس براي نگهداري آدرس ها
۵) قرار دادن فاصل در انتهاي هر رکورد

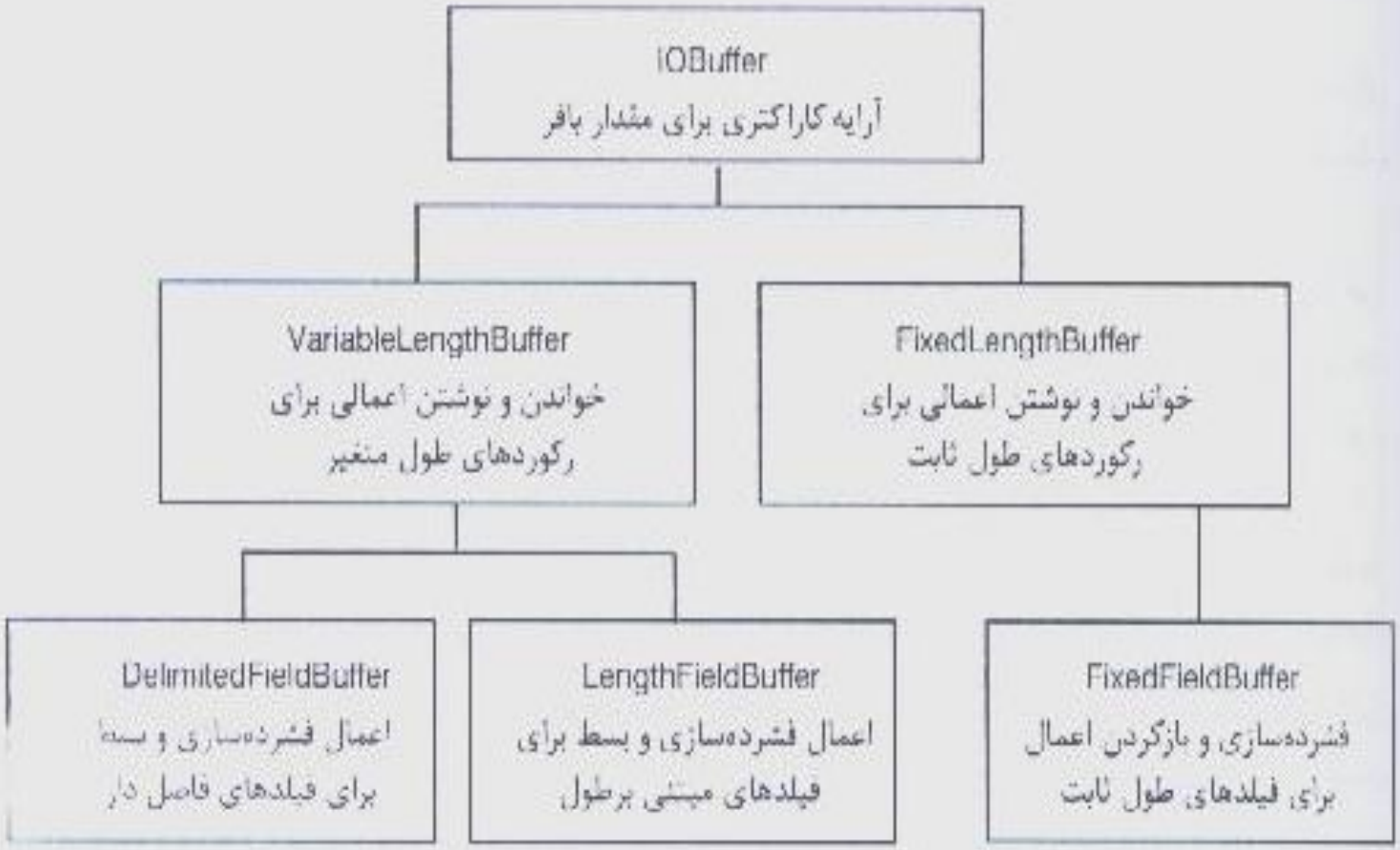
دو روش براي نمايش طول رکورد وجود دارد :

۱) طول رکورد به صورت يک عدد صحيح دو بايتي ، قبل از هر فيلد ديگر رکورد نوشته شود.

۲) تبديل طول به يک کاراکتر رشته اي با استفاده از خروجي فرمت بندي شده

با روبرداري فايل مي توان بايت هاي واقعي نگهداري شده در آن را مشاهده كرد.

C++ وراثت را در اختيار قرار مي دهد تا چندين كلاس مي توانند از اعضا و متدهاي مشترک استفاده کنند.



شکل ۱۴ ۴ سلسله مراتب کلاس بافر

مديريت فايلهايي از ركوردها

هنگامي كه كارايي جستجوهاي انجام شده در حافظه
الكترونيكي را توصيف مي كنيم
معمو
لاً از تعداد مقايسه هاي مورد نياز براي جستجو ،به عنوان واحد

به طورکلي ،کار مورد نياز براي جستجوي ترتيبي ، در فايلي با n رکورد با n متناسب است :
حداکثر n مقايسه و به طور ميانگين n/2 مقايسه مورد نياز است.

در تحليل و بحث درباره بلوک بندي رکورد چند چيز را بايد مورد توجه قرار داد :

۱) گرچه بلوک بندي مي تواند منجر به بهبود چشمگير کارايي شود ،مرتبه عملکرد جستجوي ترتيبي را تغيير نمي دهد.

۲) بلوک ساري ،اختلاف ميان سرعت دستيابي در حافظه و زمان دستيابي در حافظه ثانويه را نشان مي دهد.

۳) بلوک سازي تعداد مقايسه هايي را که بايد در حافظه انجام شوند ،تغيير نمي دهد و احتما
لاً مقدار داده هاي انتقال يافته ميان حافظه و ديسک را افزايش مي دهد.

۴) با بلوک سازي ،در زمان صرفه جويي مي شود زيرا مقدار جستجو کاهش مي يابد.

جستجوي ترتيبي براي اكثر شرايط بازيابي زمان بسيار مي برد.

اين كه آيا جستجوي ترتيبي توصيه مي شود يا خير تا حد زيادي به چگونگي استفاده از فايل ،سرعت سيستم عامل در انجام جستجو ،و چگونگي ساختار فايل بستگي دارد.

متداول ترین ساختار فایل که در یونیکس وجود دارد ، یک فایل اسکي با کاراکتر خط جدید به عنوان فاصل رکوردها و در صورت امکان ،فضاي خالي به عنوان فاصل فیلدها است.

روش ديگري كه با جستجوي ترتيبي تفاوت بنيادي دارد ، دستيابي مستقيم است.

هنگامي كه بتوانيم مستقيماً به ابتداي يک ركورد برويم و آن را به حافظه وارد كنيم ،به آن ركورد دستيابي مستقيم داريم.

جلسه پنجم

- ادامه مبحث مديريت فايلهايي از ركوردها

غالباً لازم است از برخي اطلاعات عمومي مربوط به فايل آگاه باشيم تا در آينده به استفاده از فايل كمک شود ، رکورد سرآيند غالباً در آغاز فايل قرار داده مي شود تا اين نوع اطلاعات را نگهداري كند.

هر فايل داراي يک رکورد سرآيند (header) است که حاوي سه مقدار در پايين است :

۱) اندازه سرآيند
۲) تعداد رکوردها
۳) اندازه هر رکورد

دو ساختار مختلف از رکوردها که حاوي فيلدهاي طول متغير در يک رکورد با طول ثابت است :

۱) فايل حاوي سرآيند ۳۲ بيتي و دو رکورد طول ثابت است که شامل فيلدهاي طول متغيري است که به NULL ختم مي شوند.

۲) فايل حاوي سرآيند ۶۶ بيتي و رکوردهايي با طول ۶۸ بايت است که با فيلدي دو بايتي شروع مي شوند.

يک طراحي شيءگراي خوب ،براي ماندگار شدن اشياء بايد عملياتي براي خواندن و نوشتن مستقيم اشياء فراهم آورد.

تا کنون عمل نوشتن نیازمند به دو عمل جداگانه بود :

۲) فشردن بطری دی یفکیل بافر

در اين بخش ،كلاس recordfile را معرفي مي كنيم كه نوعي عمل خواندن را پشتيباني مي كند كه شيئي از يک كلاس را گرفته، آن را در يک فايل مي نويسد. كاربرد بافرها در داخل كلاس پنهان مي شود.

در بحث هايي كه طي اين فصل و فصل قبل داشتيم به موارد زير پرداختيم :

۱)ركوردهاي طول متغير
۲) ركوردهاي طول ثابت
۳) دستيابي ترتيبي
۴) دستيابي مستقيم

دو مورد اول به سازماندهي فايل و دو مورد آخر به دستيابي به فايل مربوط مي شود.

داده هايي مثل صوت ،تصاوير ،و اسناد به صورت فيلدها و ركوردها ذخيره نمي شوند.

اگر در زبان برنامه نويسي امكان داشته باشد، مي توانيم اطلاعات كاربردي بيشتري درباره ساختار فايل در سرآيند قرار دهيم.

هنگامي كه سرآيند فايل حاوي اين نوع اطلاعات باشد، گفته مي شود اين فايل ،خود- توصيفگر است.

شبه داده ها را مي توان با هر فايلي همراه ساخت ،كه داده هاي اصلي آن نياز به اطلاعات پشتيبان دارد .

تصوير راستر رنگي ،آرايه اي راست گوشه از نقاط رنگي يا پيكسل ها است ،كه روي صفحه به نمايش در مي آيند.

انواع متفاوت فراواني از شبه داده ها وجود دارند كه مي توانند با يک تصوير همراه شوند ،از جمله :

۱) تعداد بيت هاي به كار رفته در توصيف هرپيكسل

۲) ابعاد تصوير- تعداد پيكسل ها به ازاي هر سطر و تعداد سطرها

۳) يک جدول جستجوي رنگ يا جعبه رنگ (pallet) كه نشان مي دهد كدام رنگ بايد به هر مقدار پيكسل در تصوير نسبت داده شود.

متدهايي براي كار با تصاوير به عنوان اشياي خاصي وجود دارد :

۱) نمايش يک تصوير پنجره اي در صفحه نمايش كنسول

۲) همراه كردن يک تصوير با يک جدول جستجوي رنگ خاص

۳) قرار دادن تصويري بر روي يک تصوير ديگر و ايجاد يک تصوير تركيبي

۴) به نمايش در آوردن پياپي چند تصوير براي انيميشن (animation)

ايده دنبال كردن فايل ها براي قرار دادن دامنه وسيعي از اشياي گوناگون ،اجتناب ناپذير است ،بويژه براي كاربردهايي كه نياز به مقدار زيادي از شبه داده ها يا تركيب غير قابل پيش بيني از انواع متفاوت داده ها دارند ،زيرا به اين ترتيب ديگر لازم نيست ركوردها حتماً از يک نوع باشند.

براي توصيف ديدي كه يک برنامه كاربردي از اشياي داده اي دارد ،از اصطلاح مدل داد هاي انتزاعي استفاده نموديم.

اين كار اساساً ديدي كاربردگرا و درون- حافظه اي از اشياء است و در آن از فرمت فيزيكي اشياء به آن صورت كه در فايل ها نگهداري مي شود چشم پوشي مي گردد.

جلسه ششم

- ادامه مبحث مديريت فايلهايي از ركوردها
- سازماندهي فايلها براي كارايي

يكي از مزاياي استفاده از برچسب ها براي شناسايي اشياي موجود در فايل ها آن است كه نيازي نيست كه از پيش بدانيم همه اشيايي كه نرم افزار با آنها سرو كار خواهد داشت به چه صورت خواهد بود.

اختلاف ميان زبان ها ،سيستم هاي عامل ،و معماري ماشين ،سه مشكل اصلي هستند كه هنگام توليد فايل هاي قابل حمل با آن ها مواجهيم.

چند راه حل براي دستيابي به قابليت حمل :

۱) توافق بر سر يک فرمت فيزيکي استاندارد براي رکورد و وفاداري به آن

۲) توافق بر سر رمزگذاري دودويي استاندارد براي عناصر داده اي

۳) تبديل اعداد و متون

۴) تبديل ساختارهاي فايل

سازماندهي فايلها براي كارايي

فشرده سازي يک فرايند دسته اي (batch) است که براي حذف حفره هاي خالي فايلي به کار مي رود که بارها و بارها مورد حذف و بهنگام سازي قرار گرفته است.

دلايل زيادي براي كوچک كردن فايلها وجود دارد :

۱) فايل هاي كوچكتر نياز به حافظه ي كمتري دارند كه باعث صرفه جويي مي شود.

۲) سريع تر انتقال داده مي شوند كه زمان دسترسي را كوتاهتر مي كند يا به جاي آن مي توان با همان زمان دسترسي از پهناي باند كمتر و ارزان تر استفاده كرد.

۳) به صورت ترتيبي ،سريع تر قابل پردازش هستند.

فشرده سازي داده ها عبارت است از رمزگذاري اطلاعات در فايل ،به صورتي كه جاي كمتري بگيرد.

تکنیک فشرده سازي که در آن تعداد بیت ها با استفاده از یک نمادگذاري فشرده تر کاهش مي یابد یکي از روش هاي فشرده سازي است که به عنوان کاهش زوايد شناخته مي شوند.

آرايه هاي اسپارس براي نوعي فشرده سازي به نام رمزگذاري طول رانش مناسب اند. ابتدا مقدار خاصي را براي شروع رمزگذاري طول اجرا در يک بايت ذخيره کرده سپس الگوريتم آن را اجرا مي کنيم.

الگوريتم آ رايه هاي اسپارس را به صورت زير اجرا مي كنيم :

۱) پيكسل هاي تشكيل دهنده ي شكل را خوانده آنها به ترتيب در فايل ذخيره كن.

۲) جايي كه يک مقدار پيكسل بيش از يک بار پشت سر هم تكرار شود اين سه بايت را به ترتيب جايگزين كن :

الف) نشان دهنده كد طول اجرا
ب) مقدار پيكسلي كه تكرار شده
ج) تعداد دفعاتي كه اين مقدار تكرار شده است.

نوع ديگري از فشرده سازي كدهاي با طول متغير را بسته به تعداد دفعات ظاهر شدن مقادير ، به آن مقادير نسبت مي دهد. به مقاديري كه بيشتر تكرار مي شوند كدهاي كوتاهتري نسبت داده مي شود بنابر اين جاي كمتري مي گيرند. كدهاي هافمن مثالي از كدهاي با طول متغير هستند.

راهي ديگر براي صرفه جويي فضا در يک فايل، بازيابي فضا در آن فايل پس از تغيير يافتن فايل است.

تغييرات فايل به سه شكل انجام مي شود :

۱) اضافه كردن ركورد

۲) بهنگام سازي ركورد

۳) حذف ركورد

متراکم کردن فایل از طریق پیدا کردن مکان هایي در فایل که حاوي هیچ داده اي نیستند و از بین بردن این مکان هاي خالي فایل ها را کوچکتر مي کند. و مکان هاي خالي هم وقتي در فایل ایجاد مي شود که رکوردي را حذف مي کنیم.

متراکم کردن فایل آسان ترین و رایج ترین روش هاي بازیابي فضا است.

به طور كلي براي فراهم كردن مكانيسمي براي حذف ركوردها و به دنبال آن استفاده دوباره از فضاي آزاد شده بايد بتوانيم دو مسئله را تضمين كنيم :

۱) ركوردهاي حذف شده به طور خاصي علامت گذاري شوند.

۲) بتوانيم محلي را كه توسط ركوردهاي حذف شده اشغال شده بود پيدا كنيم تا بتوانيم براي اضافه كردن ركوردهاي جديد از اين محل ها استفاده كنيم.

جلسه هفتم

- ادامه مبحث سازماندهي فايلها براي كارايي
- شاخص گذاري

براي بازيابي سريعتر فضا به وارد زير نيازمنديم :

۱) راهي كه بلافاصله بدانيم كه حفره هاي خالي در فايل وجود دارد يا نه

۲) راهي كه اگر چنين حفره اي وجود دارد مستقيماً به آن پرش كنيم.

استفاده از ليست هاي پيوندي براي پيوند دادن تمام ركوردها هر دو نياز فوق را برآورده مي كند.

آسان ترين راه براي كار كردن با ليست استفاده از آن به صورت پشته است.

پشته ليستي است كه در آن اضافه و حذف گره ها از يک انتهاي ليست انجام مي شود.

براي بازيابي ركوردها از طريق ليست پيوندي به موارد زير نياز داريم :

۱) راهي براي پيوند دادن ركوردهاي حذف شده و تبديل آنها به يک ليست

۲) الگوريتمي براي اضافه كردن ركوردهاي حذف شده به ليست

۳) الگوريتمي براي پيدا كردن و خارج كردن يک ركورد از ليست هنگامي كه مي خواهيم از آن ركورد استفاده كنيم.

براي مقابله با پراكندگي خارجي يک روش متراكم كردن فايل است و دو راه ديگر به قرار زير است:

۱) اگر دو حفره ركورد در ليست به صورت فيزيكي كنار هم قرار گيرند آنها را با هم يكي مي كنيم تا يک حفره ركورد بزرگتر ايجاد شود. به اين كار ادغام حفره ها در فايل ميگوييم.

۲) سعي مي كنيم پراكندگي را به حداقل برسانيم. به اين ترتيب كه يک راهبرد انتخاب جا را در نظر مي گيريم كه برنامه با استفاده از آن يک حفره ركورد را از ليست انتخاب كند.

هنگامي كه نياز داريم يک حفره ركورد را از ليست خارج كنيم با شروع از ابتداي فايل عمل جستجو را انجام مي دهيم تا ركوردي به اندازه كافي بزرگ پيدا كنيم يا به انتهاي ليست برسيم. اين راهبرد انتخاب جا به عنوان راهبرد اولين جاي مناسب(first fit) ناميده مي شود.

سه مشكل اساسي مربوط به مرتب سازي و جستجوي دودويي عبارتند از :

۱) جستجوي دودويي نياز به بيش از يک يا دو دسترسي به ديسک دارد.

۲) نگهداري يک فايل به صورت مرتب شده خيلي گران تمام مي شود.

۳) مرتب سازي داخلي تنها در مورد فايل هاي کوچک عملي است.

مرتب سازي كليدي كه گاهي به آن مرتب سازي با برچسب مي گويند بر اين ايده استوار است كه وقتي فايلي را در حافظه مرتب مي كنيم تنها چيزيكه واقعاً به آن نياز داريم كليد ركوردها است.

عيب مرتب سازي كليدي اين است كه مرتب كردن فايلي با n ركورد نياز به n دستيابي تصادفي به فايل اصلي دارد كه مي تواند بسيار بيشتر از خواندن ترتيبي همان تعداد ركورد وقت بگيرد.

شاخص گذاري

همه شاخص ها بر اساس يک مفهوم اصلي واحد عمل مي كنند: كليدها و آدرس فيلدها.

انواع شاخص هايي كه در اين فصل بررسي مي كنيم شاخص ساده ناميده مي شوند زيرا با استفاده از آرايه هاي ساده اي از ساختمان ها نشان داده مي شوند ،كه حاوي كليدها و آدرس فيلدها هستند.

چون شاخص ها به طور غير مستقيم عمل مي كنند ،
بدون دستكاري محتويات فايل ،به فايل نظم و ترتيب مي بخشند.

کاتالوگ کارتي در واقع مجموعه اي از سه شاخص است که هر کدام از یک فیلد کلید متفاوت استفاده مي کنند و همه انها از یک شماره کاتالوگ یکسان به عنوان فیلد آدرس بهره مي گیرند.

بنابراين كاربرد ديگر شاخص بندي اين است كه مي توان از طريق مسير هاي گوناگوني به فايل دست يافت.

در جستجوي دودويي لازم است امكان پرش به وسط فايل را داشته باشيم.

جلسه هشتم

- ادامه مبحث شاخص گذاري

راه ديگر براي مرتب سازي ، ايجاد شاخص براي فايل است.

ساختار شيء شاخص بسيار ساده است.

اين ساختار ليستي است كه هر عنصر آن دو فيلد دارد:

يک فيلد كليد و يک فيلد براي آفست بايت.

عملياتي كه براي يافتن داده هاي مورد نظر ،از طريق شاخص لازمند عبارتند از :

۱) ايجاد فايل داده ها و شاخص خالي اوليه

۲) باز كزدن فايل شاخص در حافظه ،قبل از به كارگيري آن

۳) نوشتن فايل شاخص بر روي ديسک ،پس از به كارگيري آن

۴) افزودن ركوردهايي به فايل و داده ها

۵) حذف ركوردها از فايل داده ها

۶) بهنگام كردن ركوردها در فايل داده ها

۷) بهنگام كردن شاخص براي انعكاس تغييرات به عمل آمده در فايل داده ها.

مزيت بزرگي كه روش شيء گرا دارد آن است كه براي اجراي اين عمليات به هرچه نياز داشته باشيم مي توانيم در متدهاي كلاس خود بيابيم.

در ايجاد فايل ها بايد دو فايل ايجاد شوند :

۱) فايل داده ها براي نگهداري اشياي داده اي

۲) فايل شاخص براي نگهداري شاخص كليد اوليه

بهنگام سازي ركوردها به دو صورت انجام مي شود :

۱) بهنگام سازي ،تعداد فيلد و كليد را تغيير مي دهد.

۲) بهنگام سازي ،در فيلد و كليد تأثير نمي گذارد.

آشکارترین بهینه سازي ،استفاده از جستجوي دودويي در متد find است که توسط :

search , *insert* و *remove* به کار گرفته مي شود.

منبع ديگر بهينه سازي ،چنانچه رکورد شاخص تغيير نکرده باشد ، نوشتن درباره رکورد شاخص در فايل شاخص است.

دستيابي به شاخص روي ديسک داراي معايب زير است :

۱) جستجوي دودويي شاخص به جاي آنکه با سرعت حافظه صورت پذيرد ،نياز به چندين پيگرد دارد.

۲) ترتيب مجدد شاخص که از حذف يا افزودن رکورد ناشي مي شود نياز به جابه جا کردن يا مرتب سازي رکوردها در حافظه ثانويه دارد که اين کار ميليونها بار گران تر از اجراي اين عمليات در حافظه است.

هرگاه یک شاخص ساده در حافظه جا نشود باید از موارد زیر استفاده کرد :

۱) در صورتي که سرعت دستیابي در اولویت قرار داشته باشد ،از سازماندهي درهمسازي استفاده شود.

۲) در صورتي که به هر دو نوع دستیابي کلیدي و ترتیبي نیاز داشته باشید ،از یک شاخص چند سطحي با ساختار درختي نظیر درخت B استفاده شود.

شاخص هاي ساده نسبت به استفاده از فايل داده اي كه بر حسب كليد مرتب شده اند مزاياي چشمگيري دارد :

۱) شاخص ساده استفاده از جستجوي دودويي را براي دستيابي كليدي به يک ركورد در فايلي كه طول ركوردهاي آن متغير است امكان پذير مي سازد.

۲) اگر ورودي هاي شاخص بسيار كوچكتر از ركوردهاي فايل داده ها باشد ،مرتب سازي و نگهداري شاخص نسبت به مرتب سازي و نگهداري فايل داده ها زمان كمتري مي برد.

۳) اگر در فايل داده ها ركوردهايي وجود دارند كه در جاي خود مستقر هستند ،با استفاده از شاخص مي توان ترتيب كليدها را بدون جابجايي ركوردهاي داده ها عوض كرد.

هنگاميكه شاخص ثانويه اي موجود باشد ،افزودن يک ركورد به فايل به معناي افزودن يک ورودي شاخص ثانويه است. زمان لازم برا انجام اين كار بسيار مشابه زمان لازم براي افزودن ورود يي به شاخص اوليه است.

یک اختلاف مهم شاخص ثانویه و شاخص اولیه آن است که شاخص ثانویه مي تواند حاوي کليدهاي دوگانه باشد.

حذف يک رکورد

معمو

لاً به معناي حذف تمامي آدرس هاي آن رکورد در سيستم فايل اس

بنابراين حذف رکوردي از فايل داده ها نه تنها به معناي حذف ورودي مربوط در شاخص اوليه بلکه به معناي حذف همه ورودي هاي موجود در همه شاخص هاي ثانويه اي است که به اين ورودي از شاخص اوليه رجوع مي کنند.

مشكل اين است كه شاخص هاي ثانويه همانند شاخص اوليه به ترتيب كليدها نگهداري مي شوند. در نتيجه حذف يک ورودي شامل ترتيب مجدد ورودي هاي موجود ،به منظور بستن فضاي باقيمانده از حذف است.

جلسه نهم

- ادامه مبحث شاخص گذاري
- پردازش کمک ترتيبي و مرتب سازي فايل هاي بزرگ

بهنگام سا زي فايل داده ها فقط هنگامي شاخص ثانويه را تحت تأثير قرار مي دهد كه كليد اوليه يا ثانويه تغيير يابند. كه سه وضعيت ممكن است پيش بيايد :

۱) بهنگام سازي باعث تغيير كليد ثانويه مي شود.

۲) بهنگام سازي باعث تغيير كليد اوليه مي شود.

۳) بهنگام سازي محدود به فيلدهاي ديگر

ساختارهاي شاخص ثانويه اي كه تا كنون ارائه كرديم دو مشكل دارند :

۱) هرباركه ركورد جديدي به فايل افزوده مي شود ،بايد فايل شاخص را دوباره مرتب كنيم ،حتي اگر ركورد جديد به يک كليد ثانويه موجود مربوط باشد.

۲) اگر كليدهاي ثانويه وجود داشته باشد ،فيلد كليد ثانويه براي هر ورودي تكرار مي شود. اين كار باعث هدر رفتن فضا مي شود.

درسيستم فايلي كه طي اين فصل طراحي كرديم ، انقياد كليدهاي اوليه به آدرس در زمان ايجاد شدن فايل ها رخ مي دهد ولي كليدهاي ثانويه در زمان استفاده ،به آدرس خود پيوند مي يابند.

پردازش کمک ترتيبي و مرتب سازي فايل هاي بزرگ

عمليات كمک ترتيبي شامل پردازش هماهنگ دو يا چند ليست ترتيبي براي ايجاد يک ليست خروجي است.

گاهي پردازش منجر به ادغام يا اتحاد اقلام موجود در ليست هاي خروجي مي شود ،گاهي هدف ،تطابق يا جايگذاري اقلام در ليست ها است ،گاهي نيز عمليات شامل تركيبي از تطابق و ادغام مي شود.

اين نوع عمليات روي ليست هاي ترتيبي ،مبناي بسياري از پردازش هاي فايل ها را تشكيل مي دهند.

گرچه روال همخواني بسيار ساده به نظر مي رسد ،براي آن که اين روال بهتر عمل کند به چند نکته بايد توجه داشت :

۱) آماده سازي

۲) دستيابي به عضو بعدي ليست

۳) همزمان سازي

۴) کنترل شرايط پايان فايل

۵) تشخيص خطاها

اگر قرار باشد تعداد زيادي از ليست ها با هم ادغام شوند ،مي توان به جاي حلقه مقايسه ها از درخت انتخاب استفاده كرد.

مرتب سازي در حافظه شامل سه مرحله است :

۱) خواندن کل فايل از روي ديسک به حافظه

۲) مرتب سازي رکوردها با استفاده از يک روال مرتب سازي استاندارد ،مثل مرتب سازي shell

۳) نوشتن دوباره فايل روي ديسک

آيا يک الگوريتم مرتب سازي داخلي وجود دارد که به قدر کافي سريع باشد و بتواند مرتب سازي اعداد را بلافاصله پس از خوانده شدن آنها آغاز کند و منتظر قرار گرفتن کل فايل در حافظه نشود؟

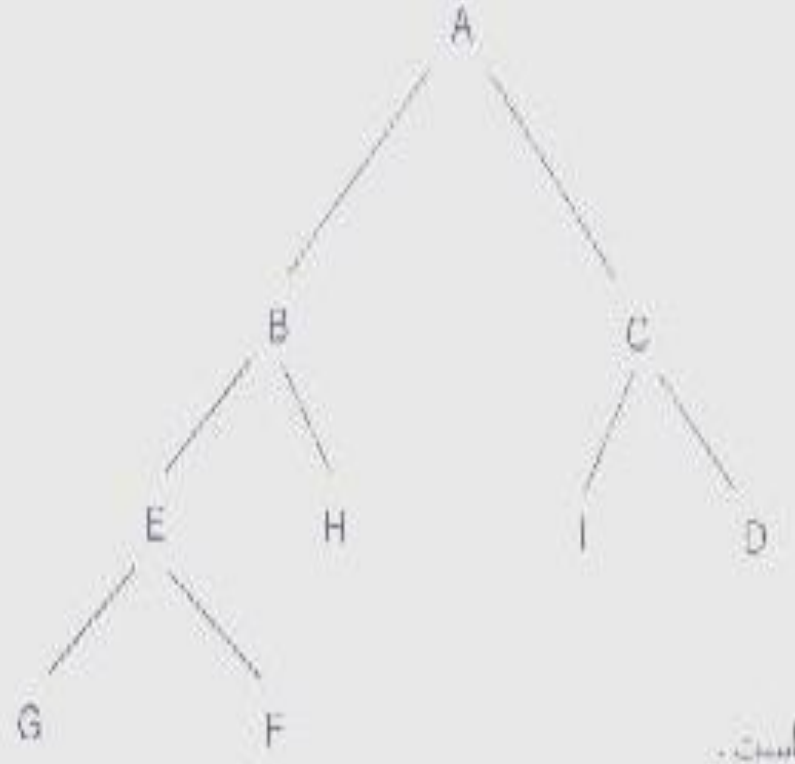
بله ،نام آن مرتب سازي هرمي (heapsort) است و مبتني بر همان اصل درخت انتخاب است.

هرم درختي دودويي با ويژگي هاي زير است :

۱) هر گره داراي کليدي است که آن کليد بزگتر يا مساوي کليد واقع در گره پدرش است.

۲) يک درخت دودويي کامل است.

۳) به خاطر ويژگيهاي ۱ و ۲ ،در نگهداري درخت مي توان آرايه اي اختصاص داد که در آن ،گره ريشه ،انديس ۱ و انديسهاي فرزندان چپ و راست گره i ،به ترتيب برابر با
i۲ و ۱ + i۲ باشند.



1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F

شکل ۸-۱۶

یک هرم که به شکل آرایه و درخت نمایش داده شده است.

الگوريتم مرتب سازي هرمي دو بخش دارد:

ابتدا هرم را ايجاد مي كنيم سپس كليدها را به صورت مرتب شده در خروجي قرار مي دهيم.

بازيابي ترتيبي كليدها به صورت زير انجام مي شود :

۱) تعيين مقدار كليد موجود در اولين موقعيت هرم . اين مقدار كوچكترين مقدار هرم است.

۲) انتقال بزرگترين مقدار هرم به اولين محل آن و كم كردن يک واحد از تعداد عناصر.

۳) ترتيب دوباره هرم. با اينكار بزرگترين عنصر با فرزند كوچكش جابجا مي شود.

هر بار كه اين سه مرحله اجرا مي شود ،كوچكتري مقدار بازيابي شده از هرم حذف مي گردد.

مرتب سازي كليدي دو نارسايي دارد :

۱) هنگاميكه كليدها مرتب سازيمي شوند ،بايد زمان زيادي صرف اين موارد شود. پيگرد هر ركورد در ركوردهاي مرتب شده ،خواندن هر ركورد به حافظه و نوشتن آن روي فايل مرتب شده جديد شود.

۲) در مرتب سازي كليدي ،اندازه فايلي كه قابل مرتب سازي است به تعداد جفت كليد/اشاره گري كه در حافظه جا شود ،محدود مي شود. در نتيجه هنوز نمي توانيم فايل هاي واقعاً بزرگ را مرتب سازي كنيم.

رانش داراي ويژگي هاي زير است :

۱) واقعاً قادر به مرتب سازي فايل هاي بزرگ هست و به فايل هايي به هر اندازه قابل بسط است.

۲) خواندن فايل ورودي در مرحله ايجاد رانش ،ترتيبي است و لذا بسيار سرعتر از ورودي است ،زيرا ورودي به ازاي هر ركورد نياز به پيگرد دارد.

۳) خواندن هر رانش طي مرحله ادغام و نوشتن ركوردهاي مرتب شده نيز ترتيبي است.

۴) اگر براي بخشي از ادغام كه در حافظه انجام مي شود از مرتب سازي هرمي استفاده شود مي توانيم اين عمليات را با I/O همپوشاني كنيم تا زمان ادغام افزايش پيدا نكند.

۵) چون I/O تا حد زيادي ترتيبي است ،در صورت نياز مي توان براي هر دو عمليات ورودي و خروجي از نوار نيز استفاده كرد.

I/O چهار بار اجرا مي گردد.

در مرحله مرتب سازي :

۱) خواندن همه ركوردها به حافظه براي مرتب سازي و تشكيل رانش ها
۲) نوشتن رانش هاي مرتب شده روي ديسک.

در مرحله ادغام :

۱) خواندن رانش هاي مرتب شده به حافظه براي ادغام
۲) نوشتن فايل مرتب شده روي ديسک

جلسه دهم

- ادامه مبحث پردازش کمک ترتيبي و مرتب سازي فايل هاي بزرگ

یک اختلاف عمده میان مرحله مرتب سازي و مرحله ادغام ،در تعداد دستیابي ترتیبي (در مقابل دستیابي مستقیم) است.

با استفاده از مرتب سازي هرمي براي ایجاد رانش هایي در مرحله مرتب سازي ،مي توان تضمین نمود که همه I/O ها از یک لحاظ ترتیبي است.

به طور خلاصه چون مرحله ادغام تنها مرحله اي است كه در آن مي توان بازدهي را با بهبود بخشيدن به روش كار ،افزايش داد بنابر اين بر آن بيشتر تأكيد داريم.

با بزرگ شدن فايل ها زمان لازم براي مرتب سازي ادغامي به سرعت افزايش مي يابد. براي كاهش اين زمان چند راه وجود دارد :

۱) تخصيص سخت افزار بيشتر نظير ديسک گردان ،حافظه و كانال هاي I/O

۲) اجراي ادغام در بيش از يک مرحله ،كاهش دادن مرتبه هر ادغام و افزايش دادن اندازه بافر براي هر رانش

۳) افزايش طول رانش هاي مرتب شده از لحاظ الگوريتمي

۴) يافتن راههايي براي همپوشاني عمليات I/O

در اين بخش سه تغيير ممكن در پيكربندي سيستم را در نظر مي گيريم كه مي تواند زمان مرتب سازي را به طور چشمگيري كاهش دهد :

۱) افزايش مقدار حافظه

۲) افزايش تعداد ديسک گردان ها

۳) افزايش تعداد كانال هاي I/O

ادغام چند مرحله ای دارای ویژگی های مطلوب زیر است :

۱) هر رکورد فقط یک بار خوانده می شود.

۲) اگر برای مقایسه های انجام شده در عملیات ادغام از یک درخت انتخاب استفاده شود ،در آن صورت تعداد مقایسه های مورد نیاز برای ادغام K جانبهٔ N رکورد ،تابعی از $N \times \log_2 k$ می شود.

۳) چونK با N تناسب مستیم دارد این عمل از مرتبهٔ $O(N \times \log_2 N)$(بر حسب تعداد مقایسه ها) است.

هدف اصلي اين مرتب سازي ادغامي آن است كه قادر باشيم فايلهايي را مرتب سازي كنيم كه در حافظه جا نمي شوند.

در ادغام چند مرحله اي، سعي نمي كنيم همه رانش ها را به يكباره ادغام كنيم. بلكه رانش هاي اوليه را به گروههاي كوچک تقسيم كرده ،رانش هاي موجود در اين گروه ها را جداگانه ادغام مي كنيم.

اساس انتخاب جايگزيني ،اين ايده است :

انتخاب هميشگي كليدي از حافظه كه داراي كمترين مقدار باشد ،قرار دادن آن كليد در خروجي ،و سپس تعويض آن با يک كليد جديد از ليست ورودي.

الگوريتم انتخاب جايگزيني را مي توان به طريق زير پياده سازي نمود :

۱) خواندن مجموعه اي از رکوردها و مرتب سازي آنها با استفاده از مرتب سازي هرمي

۲) به جاي نوشتن کل هرم اوليه به شکل مرتب شده فقط رکوردي را مي نويسيم که کليد آن داراي کمترين مقدار است.

۳) آوردن یک رکورد جدید و مقایسه مقدار کلید آن با مقدار کلیدي که به تازگي در خروجي قرار گفته است.

الف) اگر مقدار کلید جدید بزرگتر باشد رکورد جدید را در محل مناسب آن در هرم اولیه ،به همراه رکوردهایي قرار مي دهیم که از خروجي گزینش مي شوند.

ب) اگر مقدار کلید رکورد جدید کمتر باشد ،رکورد را در یک هرم ثانویه از رکوردها با مقادیر کلیدي کمتر از آنهایي که پیش از این نوشته شده اند قرار مي دهیم.

۴) تا هنگامیکه رکوردي در هرم اولیه باقي باشد و رکوردهایي براي خواندن وجود داشته باشد مرحله ۳ را تکرار مي کنیم.

گزينش جايگزيني ابزاري سودمند براي فايل هاي ورودي است كه قدري مرتب هستند و اين گزينش فرصتي براي صرفه جويي در زمانهاي انتقال و پيگرد فراهم مي آورد كه روشهاي مرتب سازي حافظه اي فاقد آن است.

در گزينش جايگزيني اگر دو ديسک در اختيار داشته باشيم ،بايد حافظه را نيز طوري پيکربندي کنيم که از آنها بهره برداري شود. حافظه را چنين پيکربندي مي کنيم :

يک بافر ورودي و يک بافر خروجي اختصاص مي دهيم تا بافردهي دوگانه امکان پذير گردد و بقيه حافظه را به تشکيل درخت انتخاب اختصاص دهيم.

اختصاص دادن بيش از يک پردازنده به يک کار امري است متداول که به حالات زير امکان پذير است :

۱) کامپيوترهاي بزرگ که بسياري از آنها مقدار زيادي از وقت خود را صرف مرتب سازي مي کنند
،معمو
لاً داراي دو يا چند پردازنده هستند که همزمان روي بخش هاي متفاوت يک مسئله کار م

۲) پردازنده هاي برداري و آرايه اي را مي توان طوري برنامه ريزي کرد که انواع گوناگون الگوريتم مرتب سازي را سريعتر از پردازنده هاي اسکالر اجرا کند.

۳) ماشين هاي موازي انبوه ،هزاران و شايد ميليونها پردازنده دارند که مي توانند مستقل از هم کار کنند و در عين حال به شيوه هاي پيچيده با هم ارتباط برقرار کنند.

۴) شبکه هاي محلي بسيار سريع و نرم افزارهاي ارتباطي ،ارسال بخش هاي متفاوتي از يک فرايند به چند ماشين متفاوت را امکان پذير مي سازد.

اگر در سيستمي برنامه نويسي چندگانه امكان پذير باشد زمان كل براي I/O ممكن است طولاني تر باشد ،زيرا كارما بايد منتظر بماند تا كارهاي ديگر نيز I/O خود را انجام دهند.

يكي از دلايل برنامه ريزي چندگانه آن است كه به سيستم عامل امكان دهيم تا راههايي براي افزايش بازدهي كل سيستم ،با هم پوشاني پردازش و I/O در ميان امور متفاوت بيابد.

جلسه یازدهم

- ادامه مبحث پردازش کمک ترتیبي و مرتب سازي فایل هاي بزرگ
- شاخص بندي چند سطحي و درختهاي B

ليست كاملي از مجموعه جديد ابزاهايي كه كارايي مرتب سازي خارجي را بهبود مي بخشند شامل موارد زيراست :

۱) براي مرتب سازي درون حافظه اي ،از مرتب سازي هرمي براي تشكيل ليست اوليه عناصر مرتب شده در يک رانش استفاده مي كنيم.

۲) استفاده از حداكثر حافظه ممكن

۳) اگر تعداد رانش هاي اوليه چنان بزرگ باشد كه زمان كل پيگرد و چرخش ،بسيار بزگتر از زمان انتقال كل باشد از ادغام چندمرحله اي استفاده مي كنيم.

۴) استفاده از گزينش جايگزيني براي تشكيل رانش هاي اوليه را در نظر بگيريم.

۵) از بيش از يک ديسک گردان و كانال I/O استفاده مي كنيم.

۶) عناصر بنيادي مرتب سازي خارجي و هزينه هاي نسبي آنها را به خاطر مي سپاريم و به دنبال راه هايي براي بهره بردن از معماري ها و سيستمهاي جديد ،نظير پردازش موازي مي گرديم.

درادغام موازنه شده دوجانبه ،توزيع اوليه بايد روي دو نوارگردان صورت پذيرد و در هر مرحله از ادغام ،به جز آخري خروجي بايد روي دو نوارگردان صورت گيرد.

اين ادغام ساده ترين الگوريتم ادغام نواري است.

ايده هاي استفاده از الگوريتم هاي ادغام مرتبه بالاتر و ادغام رانش ها از روي يک نوار در چند مرحله مبناي دو روش معروف براي ادغام ،موسوم به ادغام چند مرحله اي يا ادغام آبشاري است.

به طور كلي اين ادغام ها در ويژگي هاي زير با هم مشتركند :

۱) توزيع اوليه رانش ها چنان است حداقل ادغام اوليه يک ادغام j -1 جانبه است كه در آن j تعداد نوارگردان ها است.

۲) توزيع رانش ها در ميان نوارها چنان است كه نوارها غالباً حاوي تعداد متفاوتي از رانش ها هستند.

چون دیسک ها دستگاههاي دستيابي مستقيم هستند ادغام هاي با مرتبه بسيار بزرگ را مي توان انجام داد ،حتي اگر فقط يک ديسک گردان وجود داشته باشد.

چون نوارها دستگاه هاي دستيابي مستقيم نيستند براي هر رانش اضافي که بخواهيم ادغام کنيم به يک نوارگردان اضافي نياز داريم.

بنابر اين ديسک ها بهترند.

شاخص بندي چند سطحي و درختهاي B

مشكل اصلي نگهداشتن شاخص در حافظه جانبي اين است كه دستيابي به حافظه جانبي كند است.

اين مشكل مي تواند به دو مشكل ويژه تقسيم شود :

۱) جستجو بر حسب شاخص بايد سريعتر از جستجوي دودويي باشد.
۲) درج وحذف بايد با سرعت جستجو كردن انجام شود.

درخت جستجوي دودويي چه اشكالي دارد؟

۱) براي شاخص بندي روي ديسک سرعت لازم را ندارد.

۲) يک راهبرد مؤثر براي موازنه کردن درخت وجود ندارد.

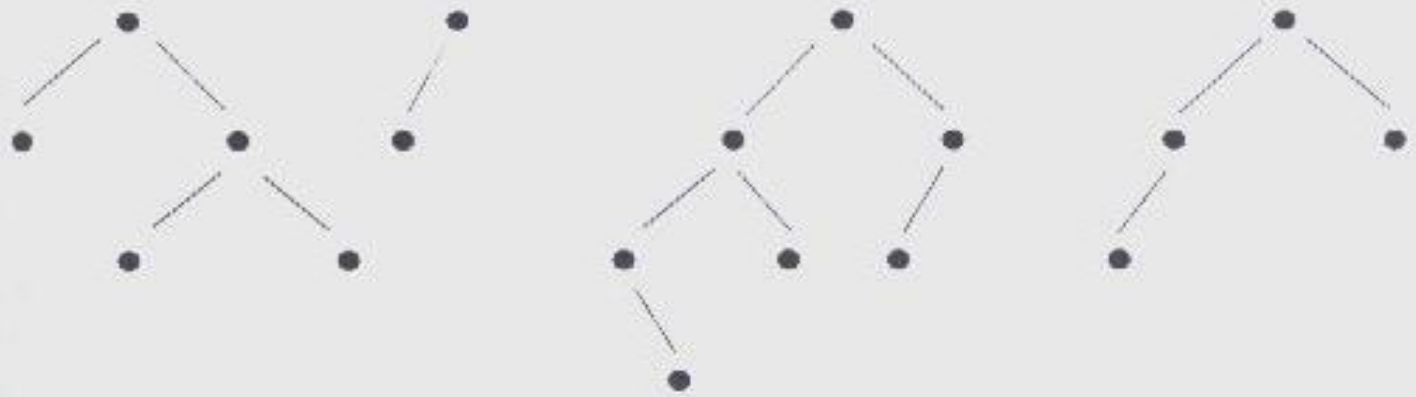
تلاشهايي براي حل مشكلات درخت جستجوي دودويي انجام گرفت كه دو تا از آنها عبارتند از :

۱) درخت هاي AVL

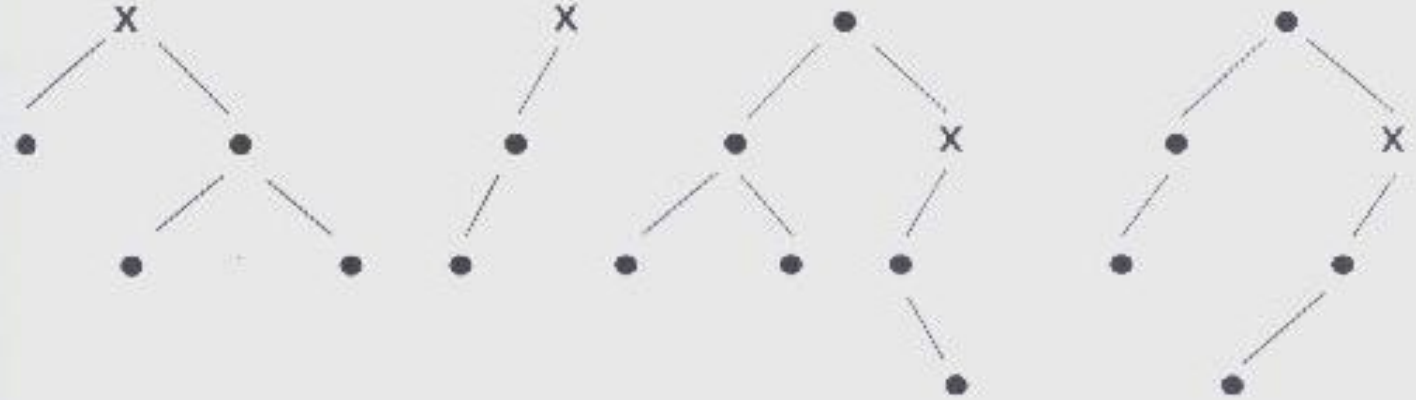
۲) درخت هاي دودويي صفحه صفحه

درخت *AVL* درختي با ارتفاع موازنه شده است.

يعني اينكه ،اختلاف مجاز ميان هر دو زيردرخت كه ريشه مشتركي دارند محدوديت دارد و حداكثر تفاوت مجاز ۱ است.



شکل ۸-۹ درختهای AVL



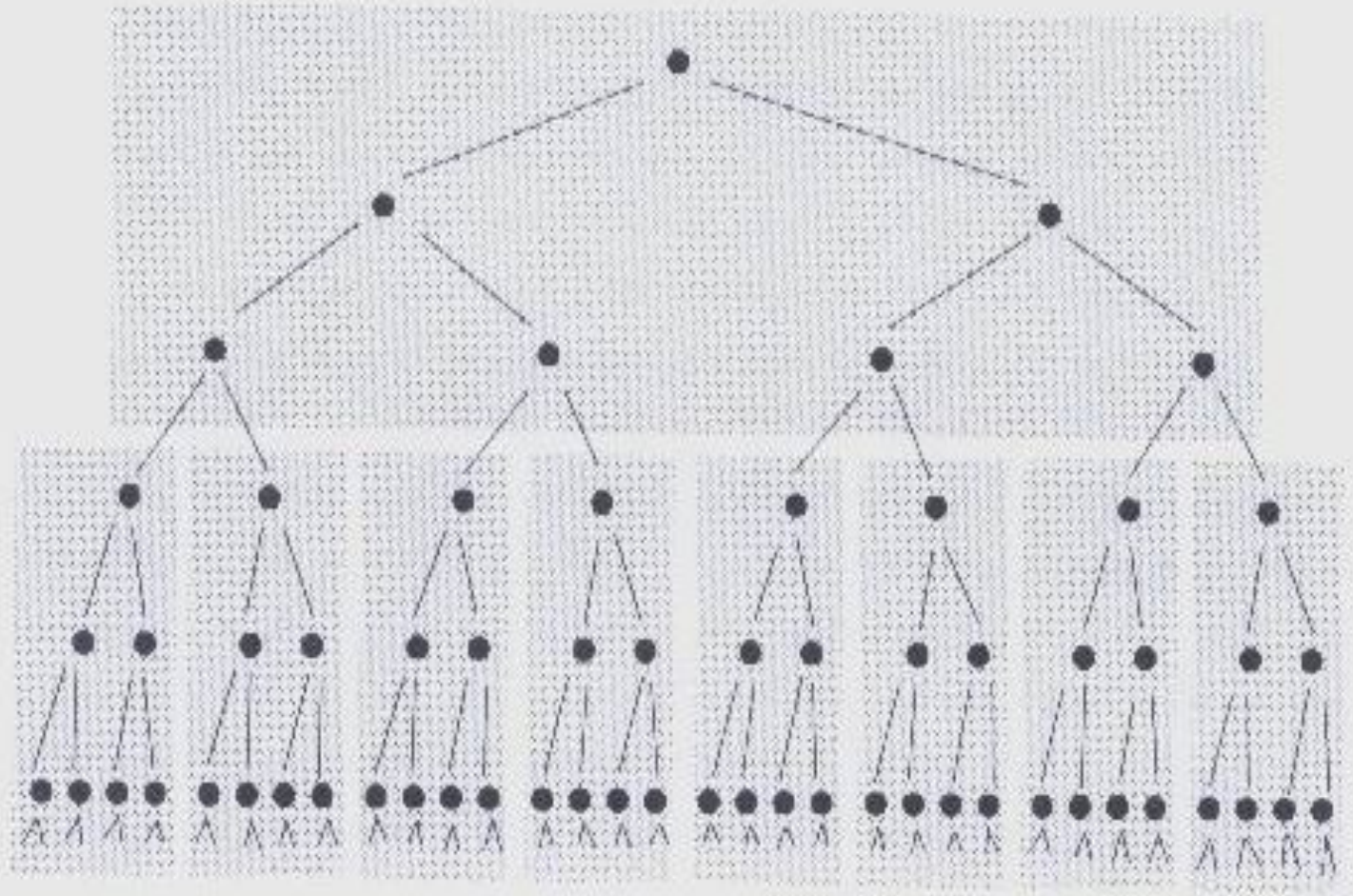
شکل ۹-۹ درختهایی که AVL نیستند

دو مزيت كه درخت هاي AVL را با اهميت مي كنند عبارتند از :

۱) با تعيين كردن حداكثر تفاوت مجاز در ارتفاع هر دو زيردرخت ،درخت هاي AVL حداقل كارايي را در جستجو تضمين مي كنند.

۲) براي اينكه هنگام درج در درخت AVL ،ويژگي خود را حفظ كند ،مستلزم چهار نوع چرخش است.

درخت دودويي صفحه اي ،سعي مي كند با قرار دادن چندين گره دودويي در يک صفحه ديسک ،مشکل را حل کند.



شکل ۱۲-۹ درخت دودویی صفحه‌ای.

واضح است که تقسیم کردن درخت به چندین صفحه امکان جستجوي سريعتر در حافظه جانبي را فراهم مي کند وبه دست آوردن اطلاعات را سريعتر از هر روش ديگر دستيابي با استفاده از کليد امکان پذير مي کند.

مشکل اصلي درخت هاي صفحه اي هنوز هم استفاده از ديسک است.

درخت هاي *B* شاخص هاي چند سطحي هستندكه مشكل هزينه خطي درج و حذف كردن را حل مي كنند.

اين ويژگي باعث جذابيت درخت B مي شود ،زيرا اكنون درخت هاي B روش استاندارد شاخص سازي هستند و از پايين به بالا ساخته مي شوند و عملياتي نظي درج و حذف ،در حافظه روي گره هاي درخت B اعمال مي شود.

جلسه دوازدهم

- ادامه مبحث شاخص بندي چند سطحي و درختهاي B

جستجو در درخت B :

۱) به صورت تكراري عمل مي كنند.

۲) در دو مرحله عمل مي كنند :

الف) به صورت يک درميان روي كل صفحات

ب) در داخل صفحات

در مورد فرايند درج كردن ،تقسيم كردن و ارتقاء ملاحظات زير را در نظر مي گيريم :

۱) با جستجويي كه تا سطح برگ پيش مي رود شروع مي شود.

۲) بعد از پيدا كردن محل درج در سطح برگ ،كار درج ،تشخيص سر ريز و تقسيم كردن از پايين به سمت بالا پيش مي رود.

از مرتبه درخت B به عنوان حداقل تعداد کليدهايي که مي تواند در يک صفحه درخت وجود داشته باشد تعريف مي شود.

پايين ترين سطح كليدها در درخت B را برگ مي نامند.

با استفاده از تعاريف ارائه شده از مرتبه و برگ ،مي توانيم خواص يک درخت B از مرتبه m را دقيقاً بيان کنيم :

۱) هر صفحه حد اکثر m فرزند دارد.

۲) هر صفحه ،به جز ريشه و برگ ها ،حداقل $\frac{m}{2}$ [فرزند دارد.

۳) ريشه حداقل دو فرزند دارد.

۴) تمام برگ ها در يک سطح قرار دارد.

۵) سطح برگ ها ،يک شاخص کامل و مرتب شده از فايل داده هاي مربوط به درخت را ايجاد مي کند.

تضمین این که درخت پهن و کم عمق باشد ،نه باریک و عمیق ،مربوط به این قوانین است :

۱) هر صفحه به جز ریشه و برگ ها حداقل [$\frac{m}{2}$] فرزند دارد.

۲) هر صفحه حاوي حداقل [$\frac{m}{2}$] و حداکثر m کلید است.

قوانين حذف كليد k از گره n در يک درخت B به اين ترتيبند :

۱) اگر تعداد كليدهاي n بيشتر از حد اقل كليدهاي مجاز است و k بزرگترين كليد در nنيست ،كافي است k را از n حذف كنيد.

۲) اگر تعداد كليدهاي n بيشتر از حد اقل كليدهاي مجاز است و k بزرگترين كليد در nاست ،k را حذف كنيد و شاخص هاس سطح بالاتر را متناسب با بزرگترين كليد جديد در n تغيير دهيد.

۳) اگر تعداد كليدهاي n دقيقاً برابر حداقل كليدهاي مجاز است و يكي از برادرهاي n به اندازه كافي خالي است n را در برابرش ادغام كنيد و يک كليد را از گره مادر حذف كنيد.

۴) اگر تعداد كليدهاي n دقيقاً برابر حداقل كليدهاي مجاز است و يكي از برادرهاي n كليدهاي زيادي دارد ،با انتقال دادن بعضي از كليدها از يک برادر به n ،كليدها را دوباره توزيع كنيد و شاخص هاي سطح بالاتر را متناسب با بزرگترين كليدهاي جديد گره هاي دستكاري شده تغيير دهيد.

توزيع دوباره در هنگام درج ،راهي براي جلوگيري يا حداقل به تعويق انداختن ايجاد صفحات جديد است.

به جاي تقسيم كردن يک صفحه پر و ايجاد دو صفحه نيمه پر، توزيع دوباره اين امكان را به ما مي دهد كه تعدادي از كليدهاي سرريز شده را به صفحه ديگري انتقال دهيم.

بنابراين استفاده از توزيع دوباره به جاي تقسيم كردن باعث مي شود كه درخت B از فضاي حافظه جانبي به طور مؤثرتر استفاده كند.

يک نوع جديد از درخت B را به عنوان درخت B^* تعريف مي کنيم که خواص آن به اين ترتيب است :

۱) هر صفحه حداکثر m فرزند دارد.

۲) هر صفحه به جزريشه حداقل $[\frac{2m-1}{3}]$ فرزند دارد.

۳) ريشه حداقل دو فرزند دارد.

۴) تمام برگ ها در يک سطح قرار دارند.

فرايند دستيابي به ديسک براي خواندن صفحه اي که در بافر وجود ندارد، نقص صفحه اي (page fault) ناميده مي شود.

دو علت براي نقص صفحه وجود دارد :

۱) هيچگاه تا كنون از آن صفحه استفاده نكرده ايم.

۲) آن صفحه قبلاً در بافر بوده است اما صفحه جديدي جايگزين آن شده است.

يک راه براي مورد دوم صفحه قبل اين است که صفحه اي را که زودتر از همه مورد استفاده قرار گرفته است جايگزين کنيم ،اين روش جايگزيني ،(List ***LRU*** Recently Used) ناميده مي شود.

استفاده از رکوردهاي با طول متغير باعث صرفه جويي در فضا و در نتيجه کاهش ارتفاع درخت B مي شود.

جلسه سيزدهم

- دستيابي به فايل‌هاي ترتيبي شاخص‌دار و درخت‌هاي +B

دستيابي به فايل هاي ترتيبي شاخص دار و درخت هاي B^+

ساختارهاي فايل ترتيبي انديس دار ،امكان انتخاب از ميان دو ديدگاه متفاوت نسبت به فايل را فراهم مي آورند :

۱) شاخص دار : فايل را مي توان به عنوان مجموعه اي از كليدها در نظر گرفت كه توسط كليد ،شاخص بندي شده اند.

۲) ترتيبي : به فايل مي توان دستيابي ترتيبي داشت و ركوردها را به ترتيب توسط كليد بازگرداند.

مجموعه اي از ركوردها كه به طور فيزيكي توسط كليدها مرتب شده اند و ركوردهايي به آن اضافه و يا از آن حذف مي شوند. چنين مجموعه اي از ركوردها را مجموعه ترتيبي مي ناميم.

هنگاميكه ركوردها را بلوک بندي مي كنيم ، بلوک واحد اصلي ورودي و خروجي مي شود.

همانند درخت هاي B ،درج ركوردهاي جديد در يک بلوک مي تواند باعث سرريز شدن بلوک شود.

مشكل سرريز شدن را مي توان توسط يک فرايند شکستن بلوک ها کنترل کرد که شبيه فرايند شکستن بلوک ها در درخت هاي B است ،ولي نه دقيقاً همان فرايند.

ته ريز شدن در درخت B مي تواند منجر به يكي از دو راه حل زير شود :

۱) اگر يک گره مجاور نيز نيمه پر باشد ،مي توان دو گره را در هم ادغام كرد و يكي از آنها را براي استفاده دوباره آزاد ساخت.

۲) اگر گره هاي مجاور بيش از نيمه پر باشند ،مي توان ركوردها را دوباره ميان گره ها توزيع كرد تا توزيع تقريباً متعادل گردد.

مسئله اندازه بلوک به تعیین حدود (limits) اندازه بلوک تبدیل مي شود.

دو شرطي كه در رابطه با حد بالايي اندازه بلوک در نظر مي گيريم عبارت است از :

۱) اندازه بلوک بايد چنان باشد كه بتوانيم چندين ركورد را به يكباره در حافظه نگه داريم.

۲) خواندن يا نوشتن يک بلوک نبايد زمان زيادي را صرف كند.

ساختار مختلط یک شاخص B بعلاوه یک مجموعه ترتیبي که رکوردها را نگه مي دارد درخت B^+ را تشکیل مي دهد.

هدف از ساختن شاخص آن است که هنگام جستجوي رکوردي با يک کليد مشخص ،به ما کمک نمايد.

شاخص بايد ما را به بلوکي از مجموعه ترتيبي هدايت کند که حاوي اين رکورد است.

شاخص به عنوان نقشه راهنمايي براي مجموعه ترتيبي عمل مي کند.

محتويات شاخص فقط تا آن حد مورد علاقه ما هستند كه بتوانند ما را در رسيدن به بلوک درست در مجموعه ترتيبي رهنمون باشند.

با در نظر گرفتن مجموعه شاخص به عنوان یک نقشه راهنما مي توانيم اين گام بسيار مهم را برداريم كه :

لازم نيست كليدها در شاخص نگهداري شوند. آنچه كه واقعاً نياز داريم ،جداكننده ها هستند.

شاخص درخت B را مجموعه شاخص مي نامند.

اين شاخص به همراه مجموعه ترتيبي ،ساختار فايلي را تشكيل مي دهد كه درخت B^+ پيشوندي ساده نام دارد.

عبارت پيشوندي ساده (simple prefix) نشانگر آن است که مجموعه شاخص حاوي کوتاهترين جداکننده ها يا پيشوندهاي کليدها است نه يک کپي از روي خود کليدهاي واقعي.

اگر حذف رکوردها از مجموعه ترتيبي و اضافه کردن رکوردها به آن ،تعداد رکوردها را در مجموعه ترتيبي تغيير دهد ،چه پيش مي آيد؟

واضح است که اگر تعداد بلوک هاي بيشتري داشته باشيم ، به تعداد جداکننده هاي کمتري نياز خواهيم داشت. تغيير دادن تعداد جداکننده ها قطعاً بر مجموعه شاخص تأثير خواهد داشت ، زيرا جداکننده ها در آنجا نگهداري مي شوند.

جلسه چهاردهم

- ادامه مبحث دستيابي به فايل هاي ترتيبي شاخص دار و درخت هايB+
- درهم سازي

درج و حذف رکوردها همواره در مجموعه ترتيبي رخ مي دهد ،زيرا رکوردها در آنجا قرار دارند.

اگر شكافتگي ،ادغام يا توزيع دوباره مورد نياز باشد ، عمليات را درست طوري انجام مي دهيد كه در صورت نبود مجموعه شاخص انجام مي داديد.

پس از کامل شدن عملیات مربوط به رکورد در مجموعه ترتیبي تغییرات مورد نیاز در مجموعه شاخص را اعمال کنید :

۱) اگر بلوک ها در مجموعه ترتیبي شکافته شوند ،یک خط جداکننده جدید باید در مجموعه شاخص درج گردد.

۲) اگر بلوک ها در مجموعه ترتیبي ادغام شوند ،یک جداکننده باید از مجموعه شاخص حذف گردند.

۳) اگر رکوردها بین بلوک ها در مجموعه ترتیبي دوباره توزیع شوند ،مقدار یک جداکننده در مجموعه شاخص باید تغییر یابد.

چند دليل براي استفاده از ا ندازه بلوک مشترک ميان مجموعه هاي ترتيبي و انديسي وجود دارد :

۱) معمولا از آن رو براي مجموعه شاخص از اندازه بلوک مجموعه ترتيبي استفاده مي شود که تطابق خوبي ميان اندازه بلوک ،ويژگيهاي ديسک گردان ،ومقدار حافظه در دسترس وجود دارد.

۲) با اندازه بلوک مشترک پياده سازي يک الگوي بافردهي براي ايجاد درخت B^+ پيشوندي ساده مجازي ،مشابه درختهاي B مجازي آسان تر مي گردد.

۳) بلوک هاي مجموعه ترتيبي و بلوک هاي مجموعه شاخص غالباً در يک فايل قرار داده مي شوند تا از جستجو ميان دو فايل جداگانه در هنگام دستيابي به درخت B^+ پيشوندي ساده پرهيز شود.

علاوه بر بردار جداکننده ها ،شاخص این جداکننده ها و ليست شماره هاي بلوک مربوط ،ساختار بلوک مجموعه شاخص شامل موارد زير مي شود :

۱) تعداد جداکننده ها

۲) طول کل جداکننده ها

فرايند بارگذاري بسيار سريع پيش مي رود زيرا :

۱) خروجي را مي توان به طور ترتيبي نوشت.

۲) بجاي چندين بار گذر مربوط به عمليات درج ،فقط يک بار از داده ها گذر مي کنيم.

۳) با پيشرفت کار نيازي به سازماندهي دوباره بلوک ها نيست.

تفاوت ميان درخت B^+ پيشوندي ساده و درخت B^+ آن است كه B^+ از پيشوندهايي به عنوان جداكننده استفاده نمي كند. در عوض جداكننده ها در مجموعه انديس ،صرفاً يک کپي از كليدهاي واقعي اند.

هر دو نوع درخت B^+ پيشوندي ساده و درخت B^+ ، از يک مجموعه رکورد تشکيل مي شود که در يک مجموعه ترتيبي بر حسب کليد مرتب شده ان و با يک مجموعه انديس جفت شده اند که دستيابي سريع به بلوک حاوي هرگونه ترکيب رکوردي خاص را فراهم مي آورد. تنها اختلاف در آن است که درخت B^+ پيشوندي ساده اي که ساخته ايم ، با استفاده از پيشوندهاي کليد مجموعه انديسي از کوتاهترين جداکننده ها ،تشکيل مي شود.

دو عامل وجود دارد که ممکن است وضعیت را به نفع درخت B^+ که در آن ،کپي کاملي از کليدها به عنوان جدا کننده بکار گرفته مي شود ،تغيير دهد :

۱) دليل استفاده از کوتاهترين جداکننده ها ،فشرده سازي هرچه بيشتر آنها در يک بلوک از مجموعه شاخص است.

۲) برخي مجموعه هاي کليدي ،هنگام استفاده از روش پيشوندي ساده براي ايجاد جداکننده ها ،فشرده سازي چنداني از خود نشان نمي دهند.

درخت B ، درخت B^+ پيشوندي ساده و درخت B^+ در خصوصيات زير مشتركند :

۱) همگي ساختارهاي شاخص صفحه اي هستند ،يعني كل اطلاعات موجود در بلوک را يكباره به حافظه منتقل مي كنند.

۲) در هر سه روش ، درختهايي نگهداري مي شود كه ارتفاع آنها وازنه است.

۳) در همه موارد ،درختها از پايين به بالا رشد مي كنند و موازنه از طريق شكستن بلوک ،ادغام و توزيع دوباره حفظ مي شود.

۴) با هر سه ساختار مي توان از طريق استفاده از شكافتگي دو به سه و در صورت امكان ،توزيع دوباره به جاي شكستن بلوک ،بازدهي را بالا برد.

۵) هر سه روش را مي توان به عنوان ساختارهاي درختي مجازي پياده سازي كرد كه در آن ،آخرين بلوک هاي استفاده شده ،در حافظه نگهداري مي شوند.

۶) هر يک از اين سه روش را مي توان با استفاده از ساختارهاي موجود در بلوک با ركوردهاي طول متغير به كار برد.

ساختار درخت B^+ سه مزيت مهم بر درخت B دارد:

۱) مجموعه ترتيبي را مي توان به شيوه اي واقعاً خطي و ترتيبي پردازش كرد و در نتيجه به ترتيب كليدها به ركوردها دستيابي مؤثري داشت.

۲) شاخص با يک كليد منفرد يا جداكننده به ازاي هر بلوک از ركوردهاي داده ها به جاي يک كليد (به ازاي هر ركورد از داده ها) ساخته مي شود.

درهم سازي

دستيابي O(1) به فايل به اين معنا است كه مهم نيست فايل تا چه اندازه بزرگ مي شود ،بلكه دستيابي به يک ركورد هميشه به تعداد كم و ثابتي پيگرد نياز دارد.

در مقابل اين نوع دستيابي ، دستيابي O(N) قرار دارد كه از جستجوهاي ترتيبي حاصل مي شود و هرچه اندازه فايل بزرگتر شود تعداد پيگردها نيز بيشتر مي شود.

تابع درهم سازي مانند يک جعبه سياه است که هرگاه کليدي در داخل آن انداخته مي شود ،يک آدرس ارئه مي دهد. به بيان رسمي تر ،درهم سازي ،تابع h(K) است که کليد k را به يک آدرس انتقال مي دهد.

درهم سازي را تصادفي كردن نيز مي گويند.

درهم سازي ،از اين نظر كه يک كليد به يک آدرس وابسته مي شود ،شبيه انديس سازي است.

درهم سازي و انديس سازي از دو جهت با هم تفاوت دارند :

۱) آدرس هايي كه از درهم سازي به دست مي آيند به صورت تصادفي اند.

۲) با درهم سازي ،دو كليد مختلف ممكن است به يک آدرس انتقال داده شوند.

جلسه پانزدهم

- ادامه مبحث درهم سازي

اگر دو رکورد به یک مکان در فایل انتقال یابند به آن برخورد مي گويند.

روش ايده آل مقابله با برخوردها اين است كه بتوان الگوريتم تبديلي پيدا كرد كه به طور كلي از برخوردها جلوگيري كند. به چنين الگوريتمي الگوريتم درهم سازي كامل گفته مي شود.

چندين راه مختلف براي كاهش تعداد برخوردها وجود دارد كه بعضي از آنها عبارتند از :

۱) پراكنده كردن ركوردها

۲) استفاده از حافظه اضافي

۳) قرار دادن بيش از يک ركورد در يک آدرس

به آدرس هايي كه مي توانند چندين ركورد را نگهداري كنند باكت مي گويند.

در اين فصل يک الگوريتم درهم سازي ساده نوشته تا انواع اعمالي را که در الگوريتم درهم سازي انجام مي شوند نشان دهد که اين الگوريم سه مرحله دارد که عبارتند از :

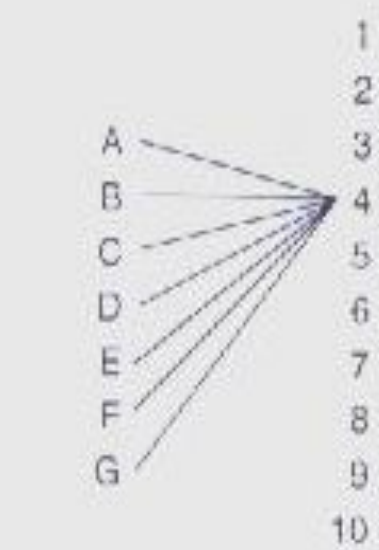
۱) نمايش کليد به شکل عددي

۲) تا کردن و اضافه کردن

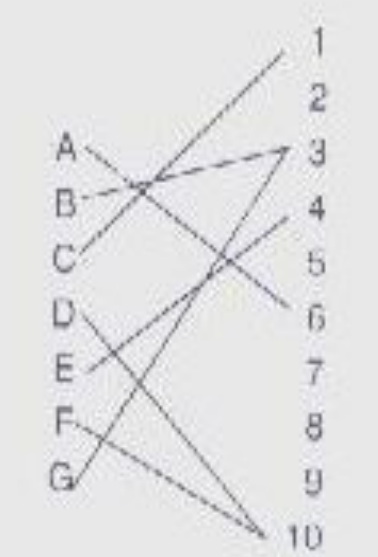
۳) تقسيم کردن بر يک عدد اول و استفاده از باقيمانده به عنوان آدرس

در حالت ايده آل تابع درهم سازي ،رکوردها را به گونه اي توزيع مي کند که هيچ برخوردي وجود نداشته باشد. چنين توزيعي را توزيع يکنواخت گويند زيرا رکوردها را به صورت يکنواخت بين آدرس ها توزيع کرده است.

(الف)



(ب)



(ج)

شکل ۳-۱۱ توزیع‌های مختلف. الف) بدون مترادفها (یکنواخت) ب) تمام مترادف‌اند (بدترین حالت) ج) مترادفهای کم

بعضي از روشهايي كه به صورت بالقوه بهتر از درهم سازي هستند نام مي بريم:

۱) جستجو در كليدها براي يافتن يک الگو

۲) تا كردن قسمتهايي از كليد

۳) تقسيم يک كليد بر يک عدد

۴) مجذور كردن كليد و گرفتن عدد مياني

۵) تبديل مبنا

توزيع پوآسون ابزاري رياضي را براي بررسي اثرات توزيع تصادفي ارائه مي كند.

از توابع پوآسون مي توان براي پيش بيني تعداد آدرس هايي كه ممكن است به ركوردهاي 0 و1 و2 وغيره نسبت داده شوند استفاده كرد.

گرچه مي توانيم تعداد برخوردها را كم كنيم ،بايد ابزارهايي داشته باشيم كه در صورت وقوع برخورد ،با آنها مبارزه كنيم. روش سرريز فزاينده را براي اين كار انتخاب مي كنيم.

اگر جستجو براي يافتن ركورد آغاز شود اما ركورد در فايل ذخيره نشده باشد چه اتفاقي مي افتد؟

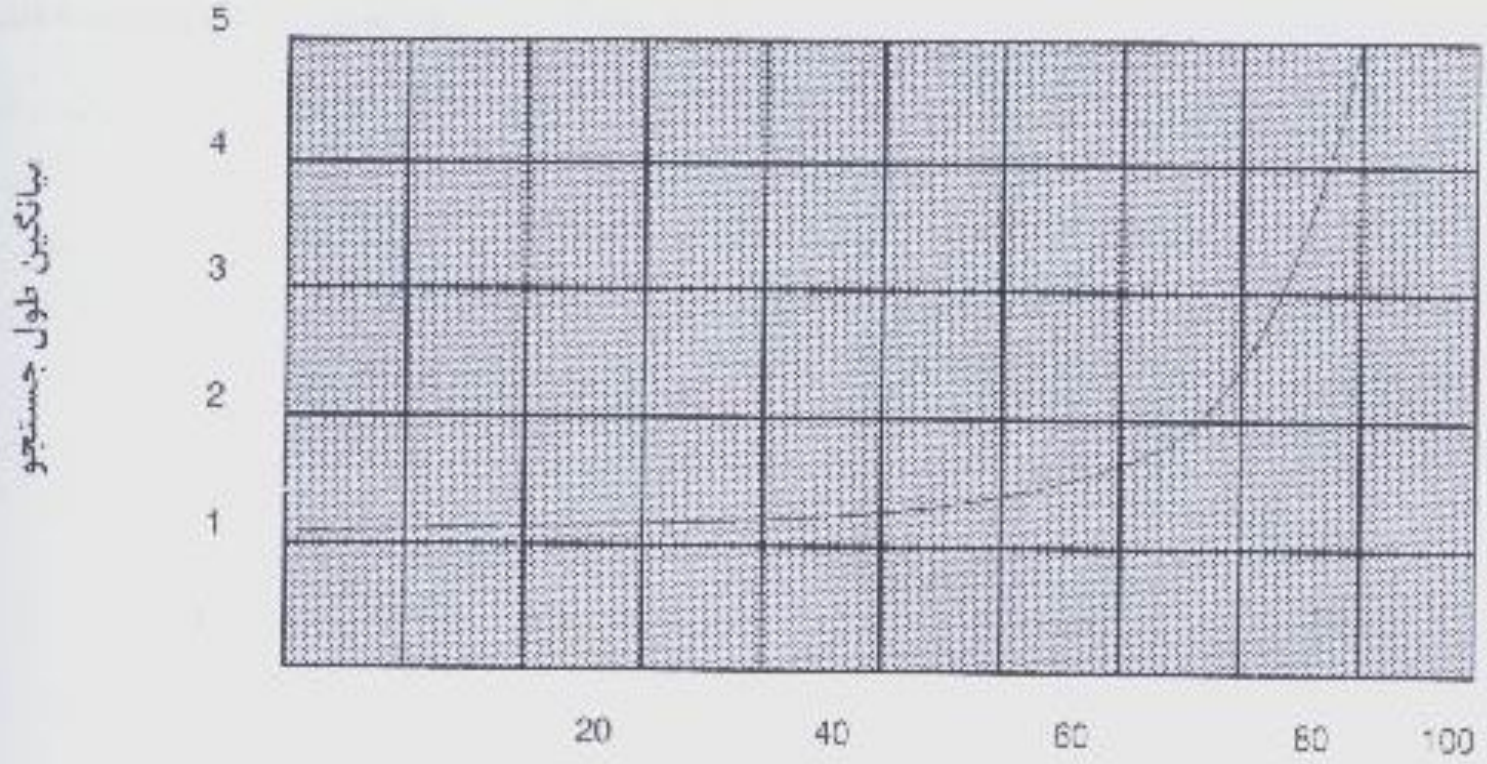
جستجو از آدرس خانگي ركورد آغاز مي شود و اين جستجو در آدرس هاي بعدي نيز ادامه پيدا مي كند.

دو اتفاق ممكن است رخ دهد :

۱) اگر با يک فضاي خالي مواجه شود ،جستجوگر ممكن است فرض كند كه فضاي خالي به اين معنا است كه ركورد در فايل موجود نيست.

۲) اگر فايل پر باشد ،جستجو ادامه پيدا مي كند تا به جايي مي رسد كه جستجو ،از آنجا شروع شده بود و مشخص مي شود كه ركورد در فايل موجود نيست.

منظور از طول جستجو ،تعداد دستيابي هاي لازم براي بازيابي يک رکورد از حافظه ثانويه است.



شکل ۷-۱۱ میانگین طول جستجو در مقایسه با دانسیتهٔ فشردگی، دو فایل درهم‌سازی که در آن، در هر آدرس، یک رکورد می‌تواند ذخیره شود. برای رفع برخوردها از سرریز فزاینده استفاده می‌شود و فایل به حافظه بار شده است.

هنگاميكه قرار است كه يک ركورد ذخيره يا بازيابي شود ،آدرس باكت خانگي ،توسط درهم سازي مشخص مي شود.

كل باكت در حافظه قرار مي گيرد.

براي پيدا كردن ركورد مورد نظر ،ركوردهاي موجود در باكت جستجو مي شوند.

براي محاسبه دانسيته فشردگي فايل بايد هم به تعداد آدرس ها (باكت ها) و هم به تعداد ركوردهايي كه مي توان در هر آدرس قرار داد توجه كرد(اندازه باكت).

جلسه شانزدهم

- ادامه مبحث درهم سازي
- درهم سازي قابل توسعه

به عنوان يک اصل معمولاً استفاده از باکت هاي بزرگتر از شيار ،ايده خوبي نيست(مگر در مواردي که رکوردها خيلي بزرگ باشند).

فايلهاي درهم سازي به دو روش با فايلهايي كه تا كنون مورد بررسي قرار گرفته ان متفاوتند :

۱) چوت تابع درهم سازي ،بر اساس تعداد ثابتي از آدرس هاي موجود بنا نهاده شده است ،اندازه منطقي فايل درهم سازي شده بايد از قبل مشخص باشد و همچنين ،وقتي اين تابع بر روي فايل عمل مي كند ،طول فايل ثابت باقي بماند.

۲) چون شماره ركورد نسبي (RRN) خانگي هر ركورد در فايل درهم سازي شده ،نسبت به كليدش منحصر به فرد است ،هر روالي كه يک ركورد را اضافه يا حذف كند و يا تغيير دهد ،بايد طوري عمل كند كه پيوند بين ركورد و آدرس خانگي آن را از بين نبرد.

تنها تفاوت بين فايل هايي كه باكت دارند و فايل هايي كه تنها مي توانند يک ركورد را در خود جاي دهند اين است كه در فايل هاي باكت دار ، هر آدرس ،فضاي كافي براي ذخيره سازي بيش از يک ركورد منطقي را دارد.

به دو دليل حذف كردن يک ركورد از فايل درهم سازي شده پيچيده تر از اضافه كردن ركورد است :

۱) محلي از فايل كه در اثر حذف كردن آزاد شده است ،نبايد مانع جستجوهاي بعدي شود.

۲) بايد امكان استفاده مجدد از فضاهاي آزاد شده ،در اضافه كردن هاي بعدي وجود داشته باشد.

خوبي علائم ويژه اين است كه ،دو مشكلي را كه درقبل آن اشاره شد حل مي كند :

۱) فضاي آزاد شده مانع جستجوي متوالي ركورد نمي شود.

۲) مسلماً مي توان از اين فضاي آزاد شده براي ذخيره ركوردهاي بعدي استفاده كرد.

چون رکوردهاي سرريز ،تأثير بسزايي در کارايي دارند تکنيک هاي زيادي براي جلوگيري از برخوردها پيشنهاد شده اند:

۱) درهم سازي دوگانه

۲) سرريز فزاينده زنجيره اي

۳) پيوند با ناحيه سرريز ديگر

۴) جدول هاي پراکندگي

درهم سازي قابل توسعه

ويژگي مهم در درخت AVL و درخت B آن است كه اين ساختارها خود تنظيم هستند و شامل مكانيسم هايي اند كه خود را نگهداري مي كنند.

ايده كليدي در فرايند درهم سازي قابل توسعه ،تركيب كردن درهم سازي معمولي با يک روش بازيابي ديگر موسوم به تراي trie)) است.

تراي ها را جستجوي مبنا نيز مي نامند زيرا ضريب انشعاب درخت جستجو برابر با تعداد نماهاي مختلف است كه ممكن است در هر موقعيت از كليد ظاهر شوند.

اگر ركوردي اضافه كنيم وجايي براي آن در باكت وجود نداشته باشد ،باكت را مي شكافيم.

هدف از سيستم درهم سازي قابل توسعه ،يافتن راهي براي افزايش فضاي آدرس ،در پاسخ به سرريز شدن است نه پاسخ دهي با ايجاد رشته اي بلند از ركوردهاي سرريز و باكت هايي كه بايد به طور خطي جستجو شوند.

عمليات اصلي روي باكت ها دقيقاً همانند عمليات ركوردهاي شاخص است :

افزودن يک جفت کليد- آدرس به باکت ،جستجو به دنبال يک کليد و بازگرداندن آدرس آن ،و حذف يک کليد.

حذف ،عکس فرایند اضافه کردن است ،با ذکر این نکته که فقط در صورتي مي توان رکوردهاي دو باکت را با هم ترکیب کرد که دو باکت با هم دوست باشند یعني این دو باکت از شکافتن یک باکت نتیجه شده باشند.

عمل تنزل شامل تخصيص فضا به آرايه جديدي از آدرس هاي باكت است كه اندازه آن نصف اندازه اوليه است و سپس آدرس هاي باكت مشترک در هر جفت سلول را به يک سلول واحد در فهرست راهنماي جديد كپي مي كند.

درهم سازي پويا و درهم سازي قابل توسعه از نظر عملكرد شباهت بسيار دارند.

هر دو از يک فهرست راهنما براي فقط آدرس باكت ها استفاده مي كنند و هر دو فهرست راهنما را از طريق استفاده از تراي ها توسعه مي دهند و هر دو تابع درهم سازي را به طور محلي به صورت يک تراي جستجوي دودويي توسعه مي دهند تا سرريز شدن قابل كنترل باشد.

اختلاف اصلي ميان دو روش درهم سازي پويا ودر هم سازي قابل توسعه آن است كه در هم سازي پويا ،رشد تدريجي و آهسته فهرست راهنما را امكان پذير مي سازد حال آنكه درهم سازي قابل توسعه فهرست راهنما را با دو برابر كردن آن بزرگ مي كند.

درهم سازي پويا فهرست راهنماي توسعه يافته را به عنوان يک ساختار متصل بيان مي كند كه به نوبه خود به عنوان يک آرايه قابل بيان است.