« به نام راستگوی بی همتا »

مبانی کامپیوتر و برنامه‌سازی

پروژه نهایی فاز ۱

مهلت تحویل : شنبه ۹۳/۹/۱۵

پاییز ۹۳

دکتر هاشمی و دکتر مرادی

مقدمه:

پیچیدگی و حجم بالای اطلاعات به همراه نیاز به دسترسی بهنگام به آنها، همگی بیانگر لزوم ساختار بخشیدن به داده‌ها در برنامه‌های کامپیوتری هستند. به طوری که در بسیاری از برنامه‌ها، عدم وجود ساختار مناسب برای مجموعه اطلاعات به شدت بر کارایی تاثیر گذاشته، سرعت اجرای سیستم را کاهش داده و یا حتی کلا سیستم را دچار اختلال می‌کند. لذا در غالب مسائلی که در آن با حجم قابل توجهی داده سروکار داریم، راه‌حل‌هایی مناسب تشخیص داده می‌شوند که نه تنها پاسخ درست داده، بلکه این پاسخ صحیح را در محدودیت زمانی مورد نظر بدهند.

مثال‌هایی از این دست در کاربرد روزمره با کامپیوترها (و تقریبا همه‌ی برنامه‌هایی که شما با آن سروکار دارید) یافت می‌شود. مثلا ورود به حساب کاربری گوگل را در نظر بگیرید، هنگام ورود شما انتظار دسترسی سریع به حساب خود را دارید و از سوی دیگر گوگل نه تنها باید اطلاعات شما را از میان میلیون‌ها داده (رکورد) مربوط به حساب کاربران مختلف استخراج کند، بلکه باید این کار را در محدوده‌ی زمانی‌ی که برای شما قابل تحمل است (کسری از ثانیه) انجام دهد. کالاهای مختلف و اطلاعات مربوط به آن‌ها در سایت ها و برنامه‌هایی چون ebay، steam و ... نمونه‌ای دیگری است که در آن کاربران به جستجو در حجم عظیمی از داده‌ها می‌پردازند و این اطلاعات باید در کسری از ثانیه، بر اساس ویژگی‌های متفاوت استخراج و برای ارائه به کاربر مرتب شوند.

از پیش‌نیازهای میسر کردن این دسترسی به موقع، بخشیدن ساختارهای نظام‌مند به مجموعه داده‌هاست. به بیانی دیگر، مساله این است که چگونه و به چه نحوی داده‌ها را در حافظه‌ی موقت نگه‌داری کنیم که عملیاتی که می‌خواهیم روی آن‌ها انجام بدهیم، در کمترین زمان ممکن صورت گیرند.

جستجو و پیدا کردن بخش خاصی از اطلاعات در میان تعداد کم، هزینه‌ی چندانی ندارد اما با افزایش حجم و پیچیدگی داده‌ها، نیاز به دسترسی سریع به آنها، ما را ملزم می کند که داده‌ها را به نحوی ساختار یافته ذخیره کرده تا بتوانیم با سرعت به آن‌ها دسترسی پیدا کنیم. جهت ساختار دادن به داده‌ها و افزایش سرعت جستجو، روش‌های مختلفی وجود دارد که ما در این تمرین به معرفی و بررسی ابتدایی دو مورد مرتب‌کردن[1] و درهم‌سازی[2] می پردازیم.

[1] sorting

دقت کنید، مفاهیمی که در این تمرین یاد می‌گیرید و پیاده‌سازی می‌کنید در پروژه‌ی نهایی به کارتان خواهند آمد، لذا انجام هرچه کامل‌تر این تمرین و تسلط هرچه بیشتر و بهتر بر مفاهیم مطرح شده و مرتبط به آنها، برای انجام درست پروژه‌ی پایانی ضروری خواهد بود.

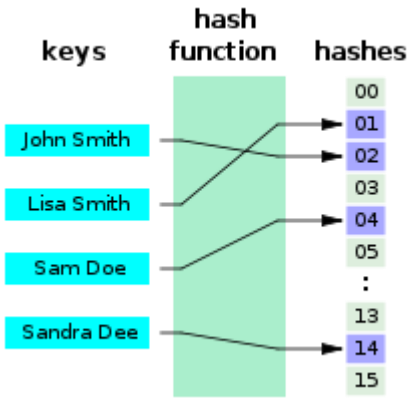
جدول درهم‌سازی[۳]:

شما تا کنون با ساختار آرایه برای نگه‌داری داده‌ها آشنا شده‌اید، در این بخش از این تمرین کامپیوتری با نگاهی خاص به آرایه‌ها، این مفهوم را گسترش می‌دهیم تا به ساختارداده‌ای پیچیده‌تر به نام جدول درهم‌سازی برسیم. آرایه‌ها را با نگاهی متفاوت می‌توان به نگاشتی تعبیر کرد که دامنه‌ی آن زیر مجموعه‌ای از اعداد صحیح (آدرس مقادیر ذخیره شده در مموری و یا آفستشان[۴] از پایه آرایه) بوده و بردش مجموعه‌ای است که هر عنصر آن یک رشته بیت (همان داده‌ی ذخیره شده) هستند. آرایه‌ها هرچند در دید سطح پایین، یک نمایش برای بلوک‌های متوالی اطلاعات در حافظه محسوب می‌شوند اما در مفهومی که در اینجا مد نظر داریم، از قراردادن محدودیت‌های توالی و صحیح بودن بر روی مجموعه دامنه‌ی یک نگاشت بدست می آیند. این نگاشت دو ویژگی خاص را با خود به همراه دارد : اول اینکه امکان اعمال یک ترتیب و دسترسی مبتنی بر آن به دیتا ها را به ما می‌دهد و دوم، با داشتن ایندکس[۵] هر داده(یا به زبان دیگر کلید مربوط به داده) بدون نیاز به محاسبات سنگین و با تعداد محدودی دستور حجم محاسباتی کوچک و مستقل از حجم کل داده‌ها، (مثال ورود به حساب گوگل را به یاد آورید) دسترسی به آن را برای ما ممکن می‌کند، که دلیل تحقق این دسترسی سریع، انطباق طبیعی مجموعه دامنه‌ی نگاشت (ایندکس‌ها) با آدرس داده‌ها در حافظه است. ویژگی ترتیبی بودن، به طور ذاتی در کاربرد آرایه مشاهده می‌شود و مبحث مرتب‌سازی نیز به نوعی به آن بستگی دارد که در بخش خود توضیح داده می‌شود (ویژگی توالی و انطباق با آدرس حافظه مزایای دیگری هم دارد که در مباحث پیچیده‌تری چون cache کردن از آن استفاده می‌شود)؛ ویژگی دوم،یعنی دسترسی سریع، نکته‌ای است که در این بخش مورد توجه است و گسترش مورد نظر ما بر آرایه‌ها و درنتیجه ساختن جدول‌درهم‌سازی را بر مبنای آن انجام می‌دهیم.

---

[2] hashing
[3] hash table
[4] offset
[5] index

در ساختن جداول درهم‌سازی مانند آرایه‌ها، کار اصلی ارتباط دادن مجموعه‌ای از کلیدها (دامنه‌ی نگاشت) به داده‌های اصلی (برد) است؛ به شیوه‌ای که بتوان بدون اعمال محدودیت‌های صحیح و متوالی بودن بر مجموعه کلیدها، همچنان دسترسی سریع به داده‌ها محقق شود. با برداشتن محدودیت‌های فوق مجموعه کلیدهادیگر الزاما با آدرس داده‌های ما در حافظه منطبق نیستند. کلیدها که درواقع ایندکس‌هایی پیچیده هستندحالا می‌توانند ساختارهای متفاوت و متنوعی داشته باشند از گونه داده‌های دیگری همچون string بوده و یا حاصل ترکیب مقادیر مختلف باشند، برای مثال ورود به حساب‌کاربری گوگل خود را که قبلا اشاره شد در نظر بگیرید دراینجا نام کاربری مجموعه کلید بوده و از جنس string است.



تصویر ۱

در ساده‌ترین سطح برای ساختن ساختارداده‌ی درهم‌ساز، کار اصلی ما معرفی یک نگاشت است که کلیدهای پیچیده را به ایندکس‌های آرایه منطبق کند.

موسسه‌ای را درنظر بگیرید که ده‌هزار دانشجو داشته با شماره دانشجویی از 810100000 تا 810100000 و به برنامه‌ای برای مدیریت رکورد دانشجوهای خود نیاز دارد، یکی از پیش‌نیازهای کار این برنامه، دسترسی سریع به رکورد دانشجوهاست. ساده‌ترین و بهترین کاری که می‌توان کرد این است که آدرس رکورد دانشجو با شماره دانشجویی i را در خانه‌ی i-810100000 قرار دهیم. در این صورت بعدا که خواستیم پرونده‌ی این دانشجو را بررسی کنیم نیز صرفا مقدار خانه‌ی i-810100000 را دریافت می‌کنیم. به عمل ایجاد این نگاشت که که شماره دانشجویی i را به مقداری دیگر که معرف آدرس داده‌ی مورد نظر است، تبدیل کردیم درهم‌سازی؛ به خود نگاشت i-810100000، تابع درهم‌ساز و ساختار داده‌ی حاصل از این نوع ذخیره‌سازی(در اینجا آرایه نهایی)، جدول‌درهم‌سازی می‌گویند؛

یکی از مشکلاتی که در جداول‌درهم‌ساز با آن مواجه می‌شویم احتمال انطباق مقادیر متفاوت، به کلیدهای یکسان است، که به آن تصادم[6] می‌گویند. در مثال قبل اصلا تصادم نداشتیم اما در عمل تعریف توابع درهم‌سازی که بتوانند کلیدها را به صورت یک‌به‌یک به ایندکس‌ها منطبق کند به سادگی امکان پذیر نیست. توضیح بیشتر این مساله را در فازهای آینده خواهید دید.

[6] collision

برای مثال تابع زیر را برای درهم‌کردن رشته‌ها در نظر بگیرید. نحوه‌ی عملکرد این تابع به این شکل است که به ازای هر رشته، از چپ به راست، چهار کاراکتر به چهار کاراکتر از رشته می‌خواند و هر کاراکتر را معادل ۸ بیتی آن‌ها را در نظر می‌گیرد، کنار هم میگذارد که می‌شود یک عدد ۳۲ بیتی و سپس معادل decimal آن‌را در نظر گرفته، این معادل‌ها را با یکدیگر جمع کرده و باقیمانده‌ی جمع را بر N که طول آرایه خواهد بود، برمی‌گرداند.

به عنوان مثال اگر رشته aaaa باشد، با توجه به اینکه معادل decimal کاراکتر a عدد ۹۷ است که به صورت باینری به شکل ۰۱۱۰۰۰۰۱ است، عدد حاصل از این کاراکتر ۰۱۱۰۰۰۰۱۰۱۱۰۰۰۰۱۰۱۱۰۰۰۰۱۰۱۱۰۰۰۰۱ خواهد بود که معادل عدد ۱۶۳۳۷۷۱۸۷۳ است، که در یک آرایه به طول ۲۰، در خانه‌ی ۱۳ قرار می‌گیرد.

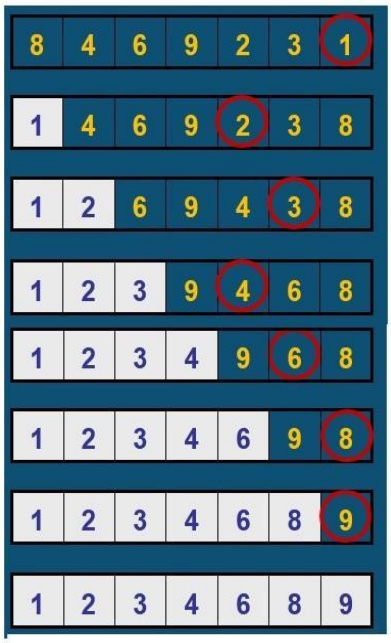
مرتب‌سازی[۷]:

از آغاز علوم کامپیوتر تاکنون، مسائل مرتب‌سازی داده‌های مختلف، همواره مورد توجه بوده‌است. مرتب‌سازی داده‌ها به معنی تغییر ترتیب آن‌ها به گونه‌ای است که بر اساس معیار مورد نظر از یک سیر صعودی یا نزولی پیروی کنند. به عنوان مثال مرتب‌سازی نزولی رکوردهای تعدادی دانشجو بر اساس معدل، به این معنی است که این رکوردها به ترتیبی قرار بگیرند که معدل‌هایشان یک توالی نزولی داشته باشند. بسیاری از داده‌هایی که با آن‌ها سروکار داریم از یک ترتیب طبیعی پیروی می‌کنند که تعاملات کاربران نهایی[۸] و متعاقبا خود برنامه، با این داده‌ها مبتنی بر این ترتیب خواهد بود. به عنوان مثال گونه داده‌های اصلی[۹] در زبان C را درنظر بگیرید : اعداد، حروف و واژه‌ها یک ترتیب طبیعی دارند و ما به طور روزمره با هزاران داده مبتنی بر این ترتیب‌ها سروکار داریم. زمانی که ایمیل‌ها یا پست‌های یک صفحه وب را به ترتیب زمانی می‌خواهیم، هنگامی که فایل های موجود در یک فولدر را به ترتیب اسم، آخرین زمان دسترسی یا ... می‌بینیم، هنگامی که موتور یک بازی باید اجسام را بر اساس فاصله با نقطه‌ی دید مرتب کند تا بتوند تشخیص دهد کدام یک از اشیا در پشت دیگر اجسام پنهان شده، کدام یک دیده می‌شوند و به چه ترتیب، تا در نهایت بتواند صحنه را نمایش دهد و ... .

[7] sorting
[8] End-users
[9] primary

مرتب‌سازی مبتنی بر مقایسه

روش‌های مرتب‌سازی به طور کلی شامل پیمایشی خاص بر اطلاعات هستند که هر قدم از این پیمایش با تکرار رویه‌ای مشخص بر روی این داده‌ها همراه است، تاجایی که لیستی مرتب از آنها به دست آید. به روش‌های مرتب‌سازی‌ی که در آن‌ها قدم‌های پیمایش برپایه‌ی مقایسه‌ی دوبه‌دوی عناصر با یکدیگر است، مرتب سازی مبتنی بر مقایسه می‌گویند. که این دسته مقایسه‌ها در بدترین حالت می‌تواند به معنی مقایسه‌ی هر عنصر با تمام عناصر دیگر باشد. در واقع الگوریتم‌های مرتب‌سازی متفاوت، روش‌های زیرکانه‌ای هستند برای کاهش تعداد این مقایسه‌ها.

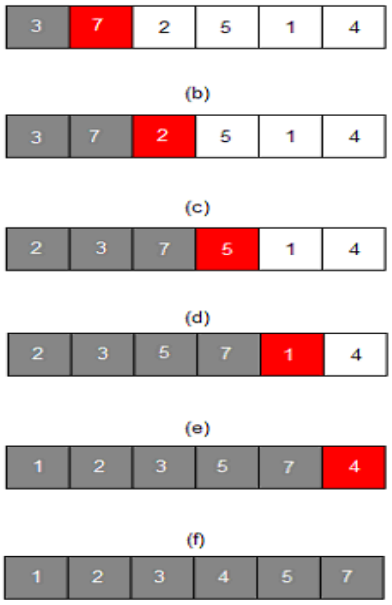


تصویر ۲

در دو روش مطرح شده در این بخش، در هر مرحله آرایه را به طور فرضی به دوبخش مرتب‌شده و نامرتب تقسیم می‌کنیم. آنگاه هر قدم الگوریتم مرتب سازی شامل اضافه کردن یک المان از بخش نامرتب به بخش مرتب خواهد بود تا جایی که کل آرایه مرتب شود.

در مرتب‌سازی انتخابی(تصویر ۲) در هر مرحله کوچکترین عنصر از بخش نامرتب را انتخاب کرده و به پایان قسمت مرتب اضافه می‌کنیم. به این ترتیب آرایه‌ی مرتب فرضی کم‌کم رشدمی‌کندتا جایی که همه‌ی عناصر مرتب شوند.

در مرتب‌سازی درجی(تصویر۳) نیز هر قدم شامل اضافه کردن یک عنصر از قسمت نامرتب به بخش مرتب است ولی در این روش طی هر مرحله، اولین عنصر بخش نامرتب آرایه را انتخاب و در محل مناسب درقسمت مرتب درج می‌کنیم.



توجه کنید که همه‌ی تغییرات در آرایه‌ی اصلی انجام می‌شود و تقسیم آرایه به دو قسمت مرتب و نامرتب با نگه‌داری یک متغیر به عنوان شاخص انفصال، که مرز این دو قسمت را مشخص می‌کند ممکن می‌شود. (در شکل‌های زیر قسمت‌های مرتب و نامرتب با رنگ متمایز شده اند و در هر مرحله اضافه کردن عنصر قرمز رنگ از قسمت نامرتب به قسمت مرتب به گسترش این قسمت و افزایش مقدار شاخص انفصال می‌انجامد)

تصویر ۳

مرتب‌سازی شمارشی

در کنار روش‌های مبتنی بر مقایسه نوعی دیگر از الگوریتم‌های مرتب‌سازی وجود دارند، که بر خلاف دسته‌ی قبل، مرتب‌سازی را بدون مقایسه عناصر با یکدیگر و با در اختیار داشتن اطلاعات اضافی راجع به داده‌ها (اطلاعاتی که لزوم این مقایسه‌ی دو به دو میان داده‌ها را از بین می‌برد) انجام می‌دهند.

برای نمونه روش مرتب‌سازی شمارشی، برای مرتب کردن عناصری به کار می‌رود که اعداد صحیح هستند؛ لذا می‌دانیم مجموعه‌ی مورد بررسی، شمارش‌پذیر و کراندار است پس می‌تواند تحت یک رابطه‌ی یک به یک به زیرمجموعه‌ای از اعداد طبیعی منطبق شود. با توجه به خوش‌ترتیب[10]ی مجموعه‌ی اعداد طبیعی تحت رابطه‌ی کوچکتر بودن، هر عنصر از مجموعه‌ی مورد بررسی دارای مکانی مشخص (نسبت به کوچکترین عضو مجموعه) است (برای مثال عدد ۱۰۰ در مجموعه‌ی اعداد طبیعی صدمین عدد است). در این حالت برای مرتب کردن آنها می‌توان به جای مقایسه‌ی عناصر با یکدیگر، از جایگاه مطلق این عناصر در مجموعه (نسبت به عضو نخست) استفاده کرد و هر عنصر را در جای خودش قرار داد. که ایده اصلی روش مرتب‌سازی شمارشی همین است. برای مثال در ساده‌ترین حالت برای مرتب‌سازی ۱۰۰ عدد در بازه‌ی ۱ تا ۱۰۰۰ می‌توانیم یک آرایه به طول هزار در نظر گرفته و هر عدد را در جای خودش (در خانه با ایندکس خودش) قرار دهیم. و به این ترتیب لیست مرتبی از این عناصر را بدست می‌آوریم.دقت کنید که قدم‌های پیمایش در این روش مرتب‌سازی با قرار دادن عناصر در جای مناسب خود در یک ترتیب مطلق و بدون انجام مقایسه‌ی عناصر با یکدیگر صورت پذیرفته است.این روش تا اینجای کار با دو مشکل روبروست، یکی تکرار و دیگری وجود عناصر بی‌هوده در آرایه‌ی اصلی(۹۰۰ خانه‌ی آرایه خالی است). لذا برای حل این مشکل روش مرتب‌سازی شمارشی را به صورت مدون زیر تعریف می‌کنیم:

ابتدا تعداد تکرار هر عنصر را شمرده و در ایندکس مربوطش در یک آرایه موقتی قرار می‌دهیم (آرایه‌ی B در تصویر ۴ پیش از add کردن)،پس از آن با تغییر این آرایه آن را به یک جدول تبدیل می‌کنیم که برای هر عنصر، مکان آن را در آرایه‌ی نهایی نشان دهد و سپس این عناصر را به ترتیب در آرایه‌ی نهایی (آرایه‌ی C) قرار می‌دهیم. شرح مراحل(تصویر ۴):

۱. آرایهC به طول آرایه‌ی اصلی و آرایه‌ای B به طول حدود اعداد (درمثال برابر ۵، چون اعدادی که باید مرتب شوند در بازه‌ی ۱ تا ۵ قرار دارند) با مقادیر اولیه‌ی صفر ایجاد می‌کنیم.

[10] well-ordered

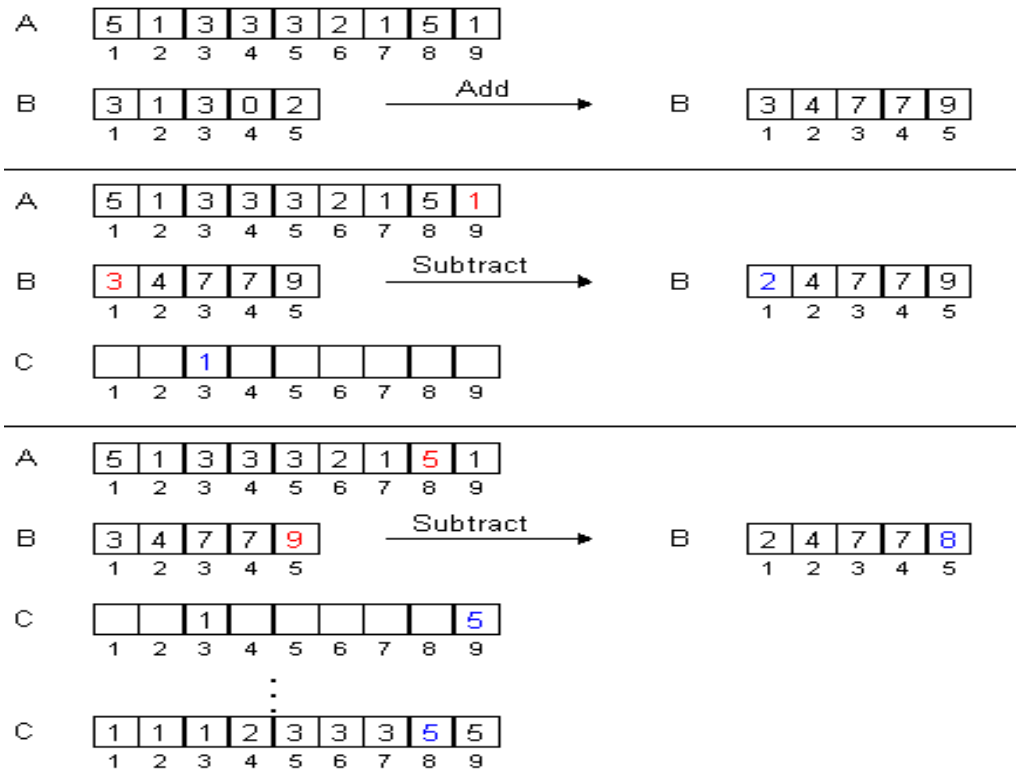
۲. به ازای هر عنصر با مقدار k، مقدار خانه‌ی kام آرایه‌ی B را یکی اضافه می‌کنیم.

۳. در پایان ارایه‌ی B نشان دهنده‌ی تعداد تکرار هر عنصر است. برای اینکه محل نهایی یک عنصر را پیدا کنیم، لازم است بدانیم که چند عنصر قبل از آن وجود دارند. لذا مرحله‌ی چهار را اعمال می‌کنیم.

۴. در آرایه‌ی B، هر خانه را برابر مجموع مقادیر خانه‌های قبلی قرار می‌دهیم.

۵. به ازای هر عنصر با مقدار k، مکان آن در خانه‌ی مقصد C[B[k]] خواهد بود. پس از قرار دادن عنصر با مقدار k در مکانش، از B[k] یک واحد می‌کاهیم. این کار را تا جایی ادامه میدهیم تا تمام عناصر در آرایه‌ی نهایی قرارگیرند.

برای افزایش شفافیت نحوه‌ی کار این مرتب‌سازی به تصویر زیر توجه کنید:



تصویر ٤

(به این نکته توجه کنید که آرایه‌ی B، درواقع یک جدول درهم‌سازی است که مشکل تصادم را هم با روشی زیرکانه حل کرده‌است و وظیفه‌ی آن در این راه‌حل افزایش سرعت دسترسی ما به داده‌های مورد نیاز است)

آن چه شما پیاده‌سازی می‌کنید:

در این فاز شما با پیاده‌سازی توابع موجود در هدرفایلی که همراه این سند در اختیارتان قرار گرفته، با سه روش مرتب‌سازی توضیح داده‌شده و همچنین درهم‌سازی آشنا می‌شوید. مقادیر نگه‌داری شده در جدول‌درهم‌سازی برای این فاز خود کلیدها (رشته‌ی ورودی) هستند. توضیح کارکرد و اطلاعات لازم برای پیاده‌سازی هریک از توابع در هدر فایل آمده‌است. کد شما و توابع پیاده‌سازی شده باید کاملا مطابق با این هدرفایل باشند.

تحویل برنامه:

شما باید یک فایل c. آپلود کنید که نام فایل شامل نام، نام خانوادگی و شماره دانشجویی شما باشد (مثلا name_family_810193123.c). از آپلود کردن فایل‌های اضافی جدا خودداری فرمایید. کسانی که این فرمت را رعایت نکنند به مشکل برخواهند خورد.

سیستم نمره‌دهی:

نمره‌دهی این فاز به صورت غیر حضوری است و سورس‌فایل شما به همراه هدرفایلی که در اختیارتان گذاشته‌ایم، در برنامه‌های تست ما استفاده می‌شوند (مشابه کتابخانه هایی که شما استفاده می‌کنید) و پاسخ به صورت اتوماتیک چک شده و نمره شما از ۱۰۰ اعلام می‌شود.

نکات پایانی:

- **این پروژه یک کار تک‌نفره است!**
- **Comment نویسی *درست* در کد الزامی است.**
- **تمیز بودن کد شما اهمیت ویژه‌ای دارد. عدم رعایت فاصله از سر خط[11] در کدنویسی و نام‌گذاری‌های نامناسب تا ۱۵٪ از نمره‌ی نهایی شما خواهد کاست.**
- **در صورت مشاهده‌ی هر گونه تشابه بین برنامه‌ی دو یا چند نفر، نمره‌ی تمامی افراد شرکت‌کننده در تقلب صفر خواهدشد.**
- **پروژه‌ی شما حتماً باید به زبان C (و نه C++) باشد. یعنی حق استفاده از هیچ کدام از کتابخانه‌های استاندارد C++ (مانند iostream، vector و ...) را ندارید. در صورت رعایت نکردن این مسئله نمره‌ی صفر برای شما لحاظ می‌گردد.**
- **برنامه‌ی شما در یک محیط استاندارد تحویل گرفته می‌شود. پس باید از توابع استاندارد C استفاده کنید. شرط استاندارد بودن، وجود آن در یکی از کتابخانه‌های بخش C Library در این آدرس است.**
- **در صورت عدم تسلط به برنامه‌ی خود در زمان تحویل، نمره‌ی صفر خواهید گرفت.**
- **در صورت استفاده از دستور goto، متغیرهای گلوبال و دستور system() نمره‌ی شما صفر خواهد شد.**
- **دقت کنید در هنگام نمره‌دهی غیر حضوری، در صورت برخورد با runtime error یا در لوپ افتادن برنامه‌ی شما در هرکدام از تست‌ها، ۴۰ درصد از نمره‌ی کل را از دست خواهید داد.**

[11] indentation

خوراک بیشتر:

در راستای درک بهتر الگوریتم‌های مطرح شده، کسب اطلاعات بیشتر و اغنا(!) اطلاعات خود لینک‌های زیر وجود دارند. البته منابع بسیار خوب دیگری هم در وب وجود دارد که با جستجو می‌توانید پاسخ سوالات خود را در آن‌ها بیابید.

http://en.wikipedia.org/wiki/Sorting_algorithm

https://en.wikipedia.org/wiki/Insertion_sort

https://en.wikipedia.org/wiki/Selection_sort

https://en.wikipedia.org/wiki/Counting_sort

http://www.csc.twu.ca/rsbook/Ch13/Ch13.2.html

http://www3.cs.stonybrook.edu/~skiena/214/lectures/lect16/lect16.html

http://en.wikipedia.org/wiki/Hash_function

http://www.cse.yorku.ca/~oz/hash.html

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-7-hashing-hash-functions/ (video)

http://burtleburtle.net/bob/hash/integer.html (it's cool :))

شاد باشید و موفق