به نام خدا

درس سیستم‌های عامل

نیم‌سال دوم ۹۹-۹۸

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس **مهدی خرازی**

تمرین **سه**

موضوع **مدیریت حافظه**

موعد تحویل **ساعت ۲۳:۵۹ دوشنبه ۲۲ اردیبهشت ۱۳۹۹**

با سپاس از دستیار آموزشی **مریم ابراهیم زاده**

اقتباس شده از **CS162 در بهار ۲۰۲۰ در دانشگاه کالیفرنیا، برکلی**

۱ مقدمه

هدف این تمرین **پیاده‌سازی دستورات مدیریت حافظه** در کتابخانه‌ی استاندارد C است. در انجام این تمرین شما با واسط POSIX و ساختار حافظه مجازی پردازه‌ها[1] آشنا شده و با چالش‌های الگوریتمی جذابی روبه‌رو خواهید شد.

راهنما: صفحات راهنمای رسمی malloc و sbrk مراجع خوبی برای انجام این تمرین هستند.

توجه: بدیهی است استفاده از دستورهای استاندارد مدیریت حافظه در C مانند malloc free، و realloc در این تمرین مجاز نیست و با هدف آن در تناقض خواهد بود.

۲ راه اندازی

ابتدا می‌بایست قالب انجام تمرین را از مخزن تمرین‌های درس دریافت کنید:

```
$ cd /home/vagrant/code/ce424-982-handouts
$ git pull origin
$ cd hw3
```

پس از دریافت فایل‌ها در مسیرِ hw۳ پرونده‌ای با نام mm_alloc.c خواهید یافت که قالبی ساده برای انجام پروژه است. در این پرونده سه دستور mm_malloc، mm_free و mm_realloc تعریف شده‌اند که شما می‌بایست آن‌ها را پیاده‌سازی کنید. از تغییر نام این توابع خودداری کنید!

همچنین در این پوشه، پرونده‌ی دیگری با نام mm_test.c وجود دارد که می‌توانید آن را برای تست کردن کدهای خود استفاده کنید. از آنجا که این پرونده در نمره‌دهی تاثیر ندارد، شما می‌توانید آن را به طور دلخواه تغییر دهید.

۳ پیش‌زمینه: گرفتن حافظه از سیستم‌عامل

۱.۳ حافظهٔ پردازه

می‌دانیم هر پردازه در سیستم‌عامل دارای فضای آدرس‌دهی مجازی [2] مخصوص به خود است. بخش‌هایی از این فضای آدرس‌دهی به هنگام تبدیل آدرس [3] توسط MMU [4] و هسته‌ی سیستم‌عامل به حافظه فیزیکی [5] نگاشته می‌شوند. برای ساختن یک تخصیص‌دهنده حافظه [6]، می‌بایست ابتدا ساختار حافظه heap را به درستی درک کرد.

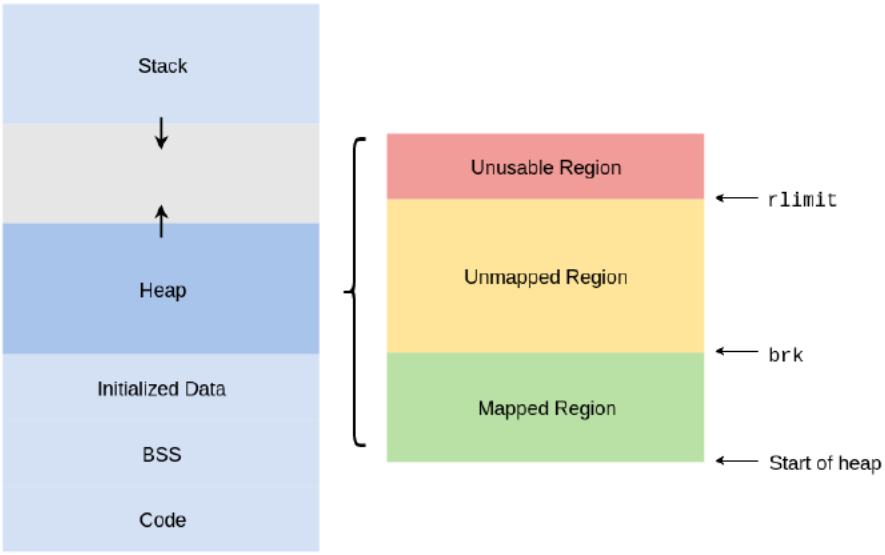
حافظه heap فضایی پیوسته از آدرس‌های مجازی است که رشد رو به بالا دارد و برای آن ۳ مرز تعریف می‌شود:

- پایین یا شروع heap
- بالای heap که به آن break (وقفه) گفته می‌شود. break پایان قسمتی از حافظه را مشخص می‌کند که به حافظه فیزیکی نگاشته شده و به کمک فراخوانی‌های سیستمی [7] brk و sbrk تغییر داده می‌شود. آدرس‌های مجازی بالاتر از break توسط سیستم‌عامل به حافظه فیزیکی نگاشته نشده‌اند.
- مرز سخت حافظه heap که break نمی‌تواند از آن بگذرد و باید پایین‌تر از آن باشد. فضای بالاتر از این آدرس قابل اختصاص به heap نیست و دسترسی به آن موجب خطا می‌شود.

  این مرز توسط تابع‌های getrlimit و setrlimit تعریف شده در فایل sys/resource.h مدیریت می‌شود.

[1] Process
[2] Virtual Address Space
[3] Address Translation
[4] Memory Management Unit
[5] Physical Address
[6] Memory Allocator
[7] system call

شکل ۱: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با linked list



در انجام این تمرین شما باید قطعه‌های **نگاشته شدهٔ** حافظه را به هنگام فراخوانی دستور allocate به فراخواننده تخصیص دهید. همچنین هنگامی که لازم شد ناحیهٔ نگاشته‌شده را گسترش دهید و محل break را به کمک دستور sbrk به میزان مناسب تغییر دهید.

۲.۳ sbrk

اندازه‌ی قسمت نگاشته‌شده‌ی حافظه heap در ابتدا صفر است. برای گسترش قسمت نگاشته شده لازم است محل break تغییر داده شود. همانطور که گفته شد فراخوانی سیستمی که برای این کار پیشنهاد می‌شود، sbrk است:

```
void *sbrk(int increment);
```

sbrk محل فعلی break را به اندازه ورودی آن (increment) افزایش می‌دهد و آدرس محل قبلی break را برمی‌گرداند. بنابراین برای گرفتن محل break کافیست به آن صفر را پاس دهید. (در واقع می‌توانید مقداری که sbrk برمی‌گرداند را به صورت محل شروع حافظه‌ای که بعد از فراخوانی sbrk به ناحیهٔ نگاشته‌شده افزوده می‌شود ببینید.)

برای اطلاعات بیشتر می‌توانید صفحات راهنمای (manual) مربوطه را مطالعه کنید.

۳.۳ داده‌ساختار Heap

برای مدیریت حافظه، لازم است مشخصات قطعه‌های آزاد (و یا اشغال شده‌ی) حافظه را در داده‌ساختار مناسبی نگهداری کنیم تا هنگام درخواست حافظه بدانیم با توجه به مقدار درخواست شده، کدام قطعه حافظه قابل اختصاص به درخواست‌کننده است. یک داده ساختار مناسب لیست پیوندی [۸] است که عناصر آن قطعه‌های حافظه هستند که ممکن است آزاد یا مورداستفاده باشند. به این منظور، درست قبل از هر قطعه‌ی حافظه تعداد مشخصی بایت برای نگهداری فراداده [۹] آن کنارگذشته می‌شود که به منزله سرآیند [۱۰] آن است.

در این سرآیند، مقادیر زیر نگهداری می‌شود:

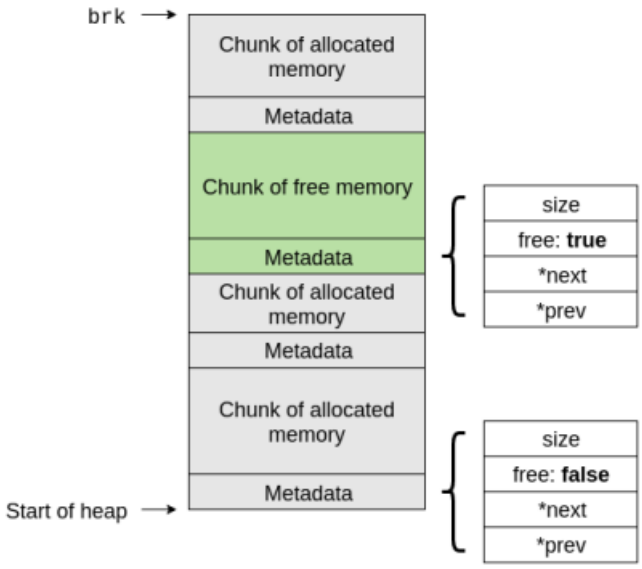
- prev و next: اشاره‌گرهایی به عناصر قبلی و بعدی linked list (که همان metadata یا سرآیند قطعه‌های قبلی و بعدی حافظه هستند)

[8] linked list
[9] metadata
[10] header

- free: مقداری دودویی که بیانگر آزاد یا مورداستفاده بودن قطعه حافظه است.
- size: اندازه قطعه حافظه

همچنین ممکن است استفاده از یک array zero-length را برای نشان دادن بلوک حافظه در نظر بگیرید.

شکل ۲: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با linked list



۴ پیاده‌سازی

روش های متفاوتی برای پیاده‌سازی اختصاص‌دهنده حافظه وجود دارد. همانطور که در قسمت قبل توضیح داده شد، در این تمرین قصد داریم از یک linked list برای پیاده‌سازی اختصاص دهنده حافظه استفاده کنیم. در این بخش، روش اختصاص، [۱۱] بازپس‌گیری [۱۲] و اختصاص مجدد [۱۳] قطعه‌های حافظه وجود را توضیح می‌دهیم. برای پیاده‌سازی موارد موردنظر، شما باید فایل mm_alloc.c را تغییر دهید.

۱.۴ اختصاص حافظه

```
void *mm_malloc(size_t size);
```

کاربر مقدار حافظه مورد درخواست خود به بایت را به صورت ورودی *size* پاس می‌دهد. تابع mm_malloc یک قطعه حافظه با اندازه خواسته شده را به کاربر اختصاص داده و اشاره‌گر به آن را برمی‌گرداند.

توجه داشته‌باشید مقداری که برمی‌گردانید باید نقطه شروع حافظه قابل استفاده باشد، نه سرآیند قطعه حافظه اختصاص داده‌شده.

یکی از الگوریتم‌های ساده برای اختصاص حافظه، **first-fit** نام دارد. در این روش هنگامی که تابع اختصاص فراخوانی می‌شود، قطعه‌های حافظه را به ترتیب مرور می‌کند تا قطعه‌ای که به اندازه کافی بزرگ باشد را پیدا کند:

- اگر چنین قطعه‌ای پیدا نشود، sbrk را صدا می‌کنیم تا فضای heap را گسترش دهیم.

[11] Allocation
[12] Deallocation
[13] Reallocation

- اولین قطعه حافظه‌ای که به اندازه کافی بزرگ باشد به کاربر اختصاص داده می‌شود. در صورتی که این قطعه آنقدر بزرگ باشد که علاوه بر مقدار مورد درخواست کاربر بتواند قطعه دیگری را نیز در خود جای دهد، قطعه به دو قسمت تقسیم می‌شود که یکی دقیقاً به اندازه مورد درخواست کاربر است و دیگری شامل قسمت اضافی از قطعه اولیه است.

  در انجام محاسبات یاد شده به وجود سرآیند metadata توجه کنید. برای مثال ممکن است قطعه‌ای پیدا کنید که از مقدار درخواست‌شده توسط کاربر بزرگتر باشد ولی نتواند علاوه بر آن سرآیند یک قطعه جدید را در خود جای دهد. در این صورت در این روش از مقدار اضافه صرف‌نظر می‌کنیم و آن را بدون استفاده خواهیم گذاشت.

- اگر نمی‌توانیم قطعه‌ای با اندازه خواسته شده را به کاربر اختصاص دهیم، مقدار NULL را برمی‌گردانیم.
- اگر مقدار درخواست شده صفر باشد، NULL را برمی‌گردانیم.
- برای ساده‌تر کردن نمره‌دهی، مقدار بایت‌های اختصاص داده شده را قبل از تحویل به کاربر صفر کنید. (عملاً دستور calloc را پیاده‌سازی می‌کنید! می‌توانید از دستور memset کتابخانه استاندارد C استفاده کنید)

۲.۴ بازپس‌گیری حافظه

```
void mm_free(void *ptr);
```

کاربر زمانی که دیگر به یک قطعه از حافظه نیازی نداشته باشد، اختصاص‌دهنده حافظه را فراخوانی می‌کند تا آن قطعه را آزاد کند. به این منظور کاربر همان آدرسی که از mm_malloc دریافت کرده (شروع قطعه موردنظر) را به mm_free پاس می‌دهد.

دقت داشته‌باشید عمل deallocate به این معنا نیست که لازم است حافظه بازپس‌گرفته‌شده را به سیستم‌عامل برگردانیم، بلکه فقط باید بتوان آن را مجدداً برای درخواست دیگری اختصاص داد. به این ترتیب شما هیچ گاه break را پایین‌تر نخواهید برد.

- از آنجا که هنگام اختصاص گاهی قطعات حافظه را تقسیم می‌کنیم، پس از مدتی با مسئله fragmentation روبرو می‌شویم: هنگامی که بلوک های شما برای درخواست تخصیص حافظه بزرگ، بسیار کوچک می‌شوند و حتی با وجود یک بخش آزاد که به اندازه کافی بزرگ است امکان تخصیص آن را ندارند. به طور مثال،

  ممکن است قطعه‌ای آزاد از حافظه به مقدار N بایت موجود باشد اما چون در چند قطعه مجاور شکسته شده نمی‌توانیم آن را به یک درخواست $N >$ بایتی اختصاص دهیم.

  برای جلوگیری از این موضوع، هنگام فراخوانی دستور free قطعه آزاد شده را در صورت وجود به قطعات آزاد مجاور ملحق می‌کنیم. یعنی اگر قطعه‌ای که آزاد شده در همسایگی قطعه آزاد دیگری باشد، آن دو قطعه را با یکدیگر merge کرده و یک قطعه آزاد بزرگتر ایجاد می‌کنیم.

  در اینجا نیز باید به سرآیند قطعات حافظه توجه داشته باشید و عمل حذف سرآیند میانی از linked list را به درستی انجام دهید.

- اگر اشاره‌گر NULL به deallocator شما پاس داده شود، نباید هیچ کاری انجام دهید.

۳.۴ اختصاص مجدد حافظه

```
void *mm_realloc(void* ptr, size_t size);
```

دستور Reallocation باید اندازه قطعه حافظه واقع در آدرس *ptr* را به *size* تغییر دهد.

ptr همان مقدار دریافت شده از mm_malloc هنگام اختصاص حافظه است و size می‌تواند بزرگتر یا کوچکتر از اندازه قطعه داده شده باشد.

برای سادگی، realloc را با آزاد کردن قطعه داده‌شده به کمک mm_free و اختصاص یک قطعه جدید با اندازه درخواست شده به کمک mm_malloc و در نهایت کپی کردن داده‌های قدیمی به محل جدید با کمک دستور memcpy کتابخانه استاندارد C پیاده‌سازی کنید.

اگر اندازه درخواست شده از اندازه قطعه اولیه بزرگتر باشد، بایت های جدید در انتهای قطعه باید همگی صفر شوند.

- اگر نمی‌توانید قطعه‌ای با اندازه درخواست شده را به کاربر اختصاص دهید، مقدار NULL را برگردانید.
- realloc(ptr،۰) هم ارز دستور mm_free(ptr) است و مقدار NULL را برمی‌گرداند.
- realloc(NULL،n) هم ارز دستور mm_malloc(n) است.
- realloc(NULL،۰) هم ارز دستور mm_malloc(۰) است و باید NULL را برگرداند.
- اطمینان حاصل کنید که حالتی که size کوچکتر از مقدار اصلی است را انجام داده‌اید.

۴.۴ تحویل‌دادنی‌ها

تحویل‌دادنی شما در این تمرین صرفاً **پیاده‌سازی شما از دستورات ،realloc malloc و free** به شیوه توصیف شده در بالا است. از شما انتظار می‌رود در پیاده‌سازی خود از داده‌ساختار و الگوریتم توصیف شده در صورت تمرین پیروی کنید اما در طراحی ساختار کد خود آزاد هستید.

برای مثال دستورات دیگر تعریف شده در فایل header مانند split_block و fusion پیشنهادی هستند و پیاده‌سازی آنها تا جایی که الگوریتم توصیف شده رعایت شود جزو تحویل‌دادنی‌های شما محسوب نمی‌شود.

این تمرین تحویل‌دادنی غیر کد (مانند مستند و یا گزارش) **ندارد**.

۵ ارسال پاسخ

برای ارسال پاسخ تمرین، کافیست تغییرات خود را push کنید تا نمره‌دهی خودکار انجام شود.

پس از حداکثر نیم ساعت، شما می‌توانید با به‌روز کردن مخزن خود، نمره‌ی خود را در پرونده‌ی grade.txt مشاهده نمایید.

دقت کنید که شما باید تمام کد خود را در همان دو فایل داده‌شده یعنی mm_alloc.h و mm_alloc.c بزنید و قالب توابع اصلی گفته‌شده در بخش قبل را هم به هیچ وجه تغییر ندهید! در حین اجرای تست ابتدا تمام فایل‌های دیگر حذف می‌شوند، سپس فایل Makefile سمت سرور و mm_test.c (برای هر تست بخصوص) جایگزین فایل شما می‌شود و پروژه از ابتدا build می‌شود، بنابراین اگر فرمت داده‌شده را رعایت نکنید دچار خطای کامپایل خواهید شد که البته در فایل نمرات می‌توانید جزییات این خطا را ببینید.

هم‌چنین دقت کنید که در نسخه‌ی ارسالی تمامی استفاده‌های توابع printf یا توابع مشابه را باید حذف کنید (حتی کامنت کردن هم قابل قبول نیست و باید آن‌ها را پاک کنید) وگرنه کد شما تست نمی‌شود.

در فایل نمرات شما می‌توانید اطلاعات سودمندی درمورد تست‌هایی که برنامه‌ی شما احیانا در آن‌ها جواب غلط یا خطای اجرا داده‌است ببینید.

۶ اطلاعات اضافه

۱.۶ قسمت نگاشته‌نشده و فضای بدون مالک

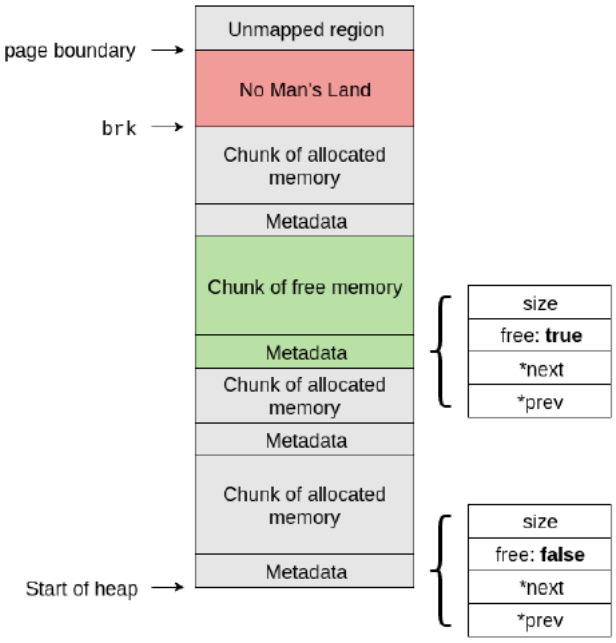
همانطور که گفته شد break انتهای قسمت نگاشته شده فضای آدرس مجازی به فضای آدرس فیزیکی را مشخص می‌کند. با این فرض، دسترسی به آدرس‌های بالاتر از break می‌بایست منجر به خطا شود. (معمولاً "bus error" یا "segmentation fault")

اما این قاعده همیشه درست نیست. می‌دانیم فضای آدرس مجازی دارای پیمانه هایی به نام page است که معمولاً اندازه آنها مضربی از ۴۰۹۶ بایت می‌باشد. هنگامی که sbrk صدا شود، سیستم عامل باید حافظه بیشتری را به heap اختصاص دهد. به این منظور، سیستم عامل یک page کامل از حافظه فیزیکی را به heap اختصاص داده و قسمت نگاشته شده‌ی heap را گسترش می‌دهد.

بنابراین همواره این احتمال وجود دارد که break دقیقاً در انتهای یک page قرار نگیرد. در این حالت، وضعیت فضای بین break و انتهای page حافظه چه خواهدبود؟

این فضا، فضای بدون مالک (***No man's land***) نامیده می‌شود و به لحاظ منطقی به heap اختصاص ندارد چرا که بالاتر از break قرار دارد ولی دسترسی به آن منجر به خطا نیز نمی‌شود چرا که در page ای از حافظه فیزیکی قرار خواهد داشت که به حافظه heap پردازه اختصاص داده شده‌است.

شکل ۳: ساختار قسمت نگاشته‌شده حافظه heap هنگام پیاده‌سازی اختصاص‌دهنده با linked list



این موضوع می‌تواند به باگ‌های عجیبی در نرم‌افزار منجر شود. دسترسی به فضایی بیرون از heap (مثلاً به علت شماره درایه نادرست در استفاده از آرایه‌ها) منجر به بروز خطانمی‌شود و برنامه به عملکرد نادرست خود ادامه می‌دهد. یافتن منشأ چنین باگ‌هایی می‌تواند بسیار دشوار باشد.

برای مثال ممکن است برنامه دارای ایرادی باشد که داده‌هایی را به اشتباه خارج از محل صحیح خود در heap بنویسد اما برای ورودی‌های کوچک این داده‌ها در فضای بدون مالک قرار بگیرند و خطایی رخ ندهد. اما با بزرگ شدن ورودی به تدریج از فضای بدون مالک نیز بیرون بزنیم و به فضای بیرون از page وارد شویم و خطای segfault رخ دهد. نتایج از این دست می‌تواند بسیار گیج‌کننده باشد.

۲.۶ بهبود

این قسمت اختیاری و فاقد نمره امتیازی است و صرفاً مخصوص علاقه‌مندان است.

شما می‌توانید اختصاص‌دهنده حافظه خود را از نقطه‌نظرهای بسیاری بهبود دهید. توجه داشته باشید که نمره‌دهنده خودکار (جاج) انتظار دارد شما الگوریتم first-fit را پیاده‌سازی کنید. بنابراین اگر تصمیم به پیاده‌سازی قسمت های اضافه دارید، آن را پس از ارسال قسمت اصلی انجام دهید.

- اختصاص‌دهنده خود را Safe Thread کنید! منظور این نیست که یک قفل دور کل دستور malloc خود قرار دهید، بلکه باید داده‌ساختارهای خود را طوری طراحی کنید که دسترسی چند ریسه به صورت هم‌زمان به آنها امکان‌پذیر باشد.

یک راه خوب برای این مقصود، این است که داده‌ساختاری استفاده کنید که قطعه‌های هم‌اندازه حافظه را در یک لیست (سبد) قرار دهد و برای هر سبد یک قفل جدا در نظر بگیرید.

به این ترتیب دو ریسه که malloc را هم‌زمان صدا کنند تنها در صورتی بلاک می‌شوند که قطعه‌هایی هم‌اندازه را درخواست کنند.

- الگوریتم اختصاص‌دهی خود را بهبود دهید. الگوریتم first-fit یکی از ساده‌ترین روش‌های اختصاص‌دهی حافظه است. یکی از روش‌های پیشرفته‌تر، Allocator Buddy نام دارد که می‌توانید درباره آن تحقیق کنید.
- پیاده‌سازی realloc را بهبود دهید به طوری که در صورت امکان از قطعه حافظه فعلی استفاده کند و از آزاد کردن، اختصاص دوباره و کپی کردن بی‌مورد اجتناب کند.

۳.۶ پیاده‌سازی‌های دیگر

میتوان برای پیاده‌سازی از داده‌ساختارهای دیگر هم استفاده کرد که برای دو مورد از آن ها توضیحی در زیر آورده شده است:

۱. یک لیست از **اندازه های حافظه** ، که هر یک شامل یک linked list از قطعه‌های حافظه با آن اندازه باشند. (در واقع می‌توان آن را به لیستی از سبدهای حافظه تعبیر کرد که در هر سبد تکه‌هایی هم‌اندازه از حافظه نگه‌داری می‌شود)

۲. یک درخت بازه (**`Interval Tree`**). برگ های این درخت قطعه‌های آزاد حافظه هستند و هر گره از درخت بازه‌ای را به صورت (شروع، اندازه) بیان می‌کند. به این ترتیب اگر N بایت حافظه درخواست شود و درخت به خوبی متوازن شده باشد (مثلاً از Tree Red-Black به جای BST عادی استفاده شود) می‌توان درخت را برای قطعه‌هایی با اندازه بزرگتر از N در زمان $O(\log n)$ جست‌وجو کرد.

در صورت داشتن هرگونه سوال در رابطه با درس و تمرینات، سوال خود را به لیست ایمیلی درس به آدرس ce۴۲۴@lists.sharif.ir ارسال کنید. همچنین اگر در استفاده از سامانه طرشت به مشکلی برخوردید، آن را از طریق ایمیل hossein.moghaddas۲۶@student.sharif.edu مطرح کنید.