第十七章-空闲空间管理

预备知识

malloc.py 允许您查看简单的内存分配器的工作方式。 以下是您可以使用的选项：

```
  -h, --help            帮助信息
  -s SEED, --seed=SEED  随机种子
  -S HEAPSIZE, --size=HEAPSIZE
                        堆大小
  -b BASEADDR, --baseAddr=BASEADDR
                        堆的开始地址
  -H HEADERSIZE, --headerSize=HEADERSIZE
                        header块大小
  -a ALIGNMENT, --alignment=ALIGNMENT
                        align allocated units to size; -1->no
                        分配的地址空间大小是否对齐
  -p POLICY, --policy=POLICY
                        空闲空间搜索算法 (BEST, WORST, FIRST)
  -l ORDER, --listOrder=ORDER
                        空闲列表排序 (ADDRSORT, SIZESORT+, SIZESORT-, INSERT-
FRONT, INSERT-BACK)
  -C, --coalesce        合并空闲列表

  -n OPSNUM, --numOps=OPSNUM
                        要生成的随机操作的数量
  -r OPSRANGE, --range=OPSRANGE
                        最大分配空间大小
  -P OPSPALLOC, --percentAlloc=OPSPALLOC
                        分配空间的操作的百分比
  -A OPSLIST, --allocList=OPSLIST
                        不随机分配操作，指定操作列表(+10,-0,etc)
  -c, --compute         计算答案
```

一种使用它的方法是让程序生成一些随机分配/释放(allocation/free)操作, 检查你能不能弄清楚空闲列表是什么样子的,以及每个操作的成功或失败。

Problem1

问题描述

首先运行 flag -n 10 -H 0 -p BEST -s 0 来产生一些随机分配和释放。你能预测 malloc()/free()会返回什么吗？你可以在每次请求后猜测空闲列表的状态吗？随着时间的推移，你对空闲列表有什么发现？

问题分析

最优匹配原则是遍历空闲列表，找到满足申请的最小空闲块。本题-H 0，不需要考虑头块大小。同时分配的地址空间大小不需要对齐。

问题解答

```
python2 malloc.py -n 10 -H 0 -p BEST -s 0
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3)  returned ?
List?

最开始是一整块空闲空间，遍历空闲列表（searched 1 elements），选择空闲列表的一个表项，分配成功，返回堆的开始地址1000，
更新空闲列表为从1003开始，大小为97的一个表项。

Free(ptr[0]) returned ?
List?

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为97的两个表项。（空闲列表按地址递增排序，下文不再赘述）

ptr[1] = Alloc(5)  returned ?
List?

按照最优匹配原则，遍历空闲列表（searched 2 elements），选择空闲列表的第二个表项，分配成功，返回1003，
更新空闲列表为从1000开始，大小为3、从1008开始，大小为92的两个表项。

Free(ptr[1]) returned ?
List?

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为92的三个表项。

ptr[2] = Alloc(8)  returned ?
List?

按照最优匹配原则，遍历空闲列表（searched 3 elements），选择空闲列表的第三个表项，分配成功，返回1008，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1016开始，大小为84的三个表项。

Free(ptr[2]) returned ?
```

```
List?

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

ptr[3] = Alloc(8)  returned ?
List?

按照最优匹配原则，遍历空闲列表（searched 4 elements），选择空闲列表的第三个表项，分配成功，返回1008，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1016开始，大小为84的三个表项。

Free(ptr[3]) returned ?
List?

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

ptr[4] = Alloc(2)  returned ?
List?

按照最优匹配原则，遍历空闲列表（searched 4 elements），选择空闲列表的第一个表项，分配成功，返回1000，
更新空闲列表为从1002开始，大小为1、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

ptr[5] = Alloc(7)  returned ?
List?

按照最优匹配原则，遍历空闲列表（searched 4 elements），选择空闲列表的第三个表项，分配成功，返回1008，
更新空闲列表为从1002开始，大小为1、从1003开始，大小为5、从1015开始，大小为1、从1016开始，大小为84的四个表项。
```

答案验证

```
python2 malloc.py -n 10 -H 0 -p BEST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
```

```
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [
addr:1016 sz:84 ]
```

经验证，结果正确。

由于没有合并，随着时间的推移，空闲空间碎片会越来越多。如果加入-C参数，即每次free后合并空闲列表，那么空闲空间还是一整块。最终的分配结果也有所改变。

```
python2 malloc.py -n 10 -H 0 -p BEST -s 0 -C -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce True
numOps 10
```

```
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[1] = Alloc(5)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1005 sz:95 ]

Free(ptr[1]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[2] = Alloc(8)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]

Free(ptr[2]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[3] = Alloc(8)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1008 sz:92 ]

Free(ptr[3]) returned 0
Free List [ Size 1 ]:  [ addr:1000 sz:100 ]

ptr[4] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1002 sz:98 ]

ptr[5] = Alloc(7)  returned 1002 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1009 sz:91 ]
```

Problem3

问题描述

如果使用首次匹配（ - p FIRST)会如何？使用首次匹配时，什么变快了？

问题分析

首次匹配原则只需要找到第一个满足申请的空闲块，不需要遍历空闲列表。

问题解答

```
python2 malloc.py -n 10 -H 0 -p FIRST -s 0
seed 0
size 100
baseAddr 1000
```

```
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute False

ptr[0] = Alloc(3)  returned ?
List?
```

最开始是一整块空闲空间，查找空闲列表的一个表项（searched 1 elements），分配成功，返回堆的开始地址1000，
更新空闲列表为从1003开始，大小为97的一个表项。

```
Free(ptr[0]) returned ?
List?
```

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为97的两个表项。（空闲列表按地址递增排序，下文不再赘述）

```
ptr[1] = Alloc(5)  returned ?
List?
```

按照首次匹配原则，选择空闲列表的第二个表项（searched 2 elements），分配成功，返回1003，
更新空闲列表为从1000开始，大小为3、从1008开始，大小为92的两个表项。

```
Free(ptr[1]) returned ?
List?
```

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为92的三个表项。

```
ptr[2] = Alloc(8)  returned ?
List?
```

按照首次匹配原则，选择空闲列表的第三个表项（searched 3 elements），分配成功，返回1008，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1016开始，大小为84的三个表项。

```
Free(ptr[2]) returned ?
List?
```

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

```
ptr[3] = Alloc(8)  returned ?
List?
```

按照首次匹配原则，选择空闲列表的第三个表项（searched 3 elements），分配成功，返回1008，

```
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1016开始，大小为84的三个表项。

Free(ptr[3]) returned ?
List?

释放刚分配的空间，释放成功，返回0，
更新空闲列表为从1000开始，大小为3、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

ptr[4] = Alloc(2)  returned ?
List?

按照首次匹配原则，选择空闲列表的第一个表项（searched 1 elements），分配成功，返回1000，
更新空闲列表为从1002开始，大小为1、从1003开始，大小为5、从1008开始，大小为8、从1016开始，大小为84的四个表项。

ptr[5] = Alloc(7)  returned ?
List?

按照首次匹配原则，选择空闲列表的第三个表项（searched 3 elements），分配成功，返回1008，
更新空闲列表为从1002开始，大小为1、从1003开始，大小为5、从1015开始，大小为1、从1016开始，大小为84的四个表项。
```

不难发现，首次匹配原则只需要找到第一个满足申请的空闲位置就可以了，不需要遍历整个空闲列表，遍历时间变短。

答案验证

```
python2 malloc.py -n 10 -H 0 -p FIRST -s 0 -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy FIRST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]
```

```
Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [
addr:1016 sz:84 ]
```

经验证，结果正确。

Problem4

问题描述

对于上述问题，列表在保持有序时，可能会影响某些策略找到空闲位置所需的时间。使用不同的空闲列表排序（-l ADDRSORT,-l SIZESORT+,-l SIZESORT-)查看策略和列表排序如何相互影响。

问题解答

按照地址递增排序:

```
python2 malloc.py -n 10 -H 0 -p BEST -s 0 -l ADDRSORT -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy BEST
listOrder ADDRSORT
coalesce False
numOps 10
range 10
percentAlloc 50
```

```
allocList
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ] [
addr:1016 sz:84 ]
```

该方式会让合并空闲列表变得容易，减少了首次匹配算法在列表头部产生的空闲碎片，使首次匹配算法搜索更快。

按照空闲块大小递增排序:

```
python2 malloc.py -n 10 -H 0 -p WORST -s 0 -l SIZESORT+ -c
seed 0
size 100
baseAddr 1000
headerSize 0
alignment -1
policy WORST
listOrder SIZESORT+
coalesce False
```

```
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1016 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1024 sz:76 ]

Free(ptr[3]) returned 0
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:8 ] [ addr:1024 sz:76 ]

ptr[4] = Alloc(2)  returned 1024 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:8 ] [ addr:1026 sz:74 ]

ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ] [
addr:1016 sz:8 ] [ addr:1033 sz:67 ]
```

因为最优匹配算法是遍历空闲列表，找到满足申请的最小空闲块，将空闲列表按照空闲块大小递增排序会让最优匹配算法搜索更快。

按照空闲块大小递减排序:

```
python2 malloc.py -n 10 -H 0 -p WORST -s 0 -l SIZESORT- -c
seed 0
size 100
baseAddr 1000
headerSize 0
```

```
alignment -1
policy WORST
listOrder SIZESORT-
coalesce False
numOps 10
range 10
percentAlloc 50
allocList
compute True

ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1003 sz:97 ] [ addr:1000 sz:3 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1008 sz:92 ] [ addr:1000 sz:3 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1008 sz:92 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1016 sz:84 ] [ addr:1003 sz:5 ] [ addr:1000 sz:3 ]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1016 sz:84 ] [ addr:1008 sz:8 ] [ addr:1003 sz:5 ] [
addr:1000 sz:3 ]

ptr[3] = Alloc(8)  returned 1016 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1024 sz:76 ] [ addr:1008 sz:8 ] [ addr:1003 sz:5 ] [
addr:1000 sz:3 ]

Free(ptr[3]) returned 0
Free List [ Size 5 ]:  [ addr:1024 sz:76 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [
addr:1003 sz:5 ] [ addr:1000 sz:3 ]

ptr[4] = Alloc(2)  returned 1024 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1026 sz:74 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [
addr:1003 sz:5 ] [ addr:1000 sz:3 ]

ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1033 sz:67 ] [ addr:1008 sz:8 ] [ addr:1016 sz:8 ] [
addr:1003 sz:5 ] [ addr:1000 sz:3 ]
```

因为最差匹配算法是遍历空闲列表，找到满足申请的最大空闲块，将空闲列表按照空闲块大小递减排序会让最差匹配算法搜索更快。

三种排序方式在 free 时会变慢，因为插入空闲块时需要遍历空闲列表找到插入位置，来达成某种排序方式。