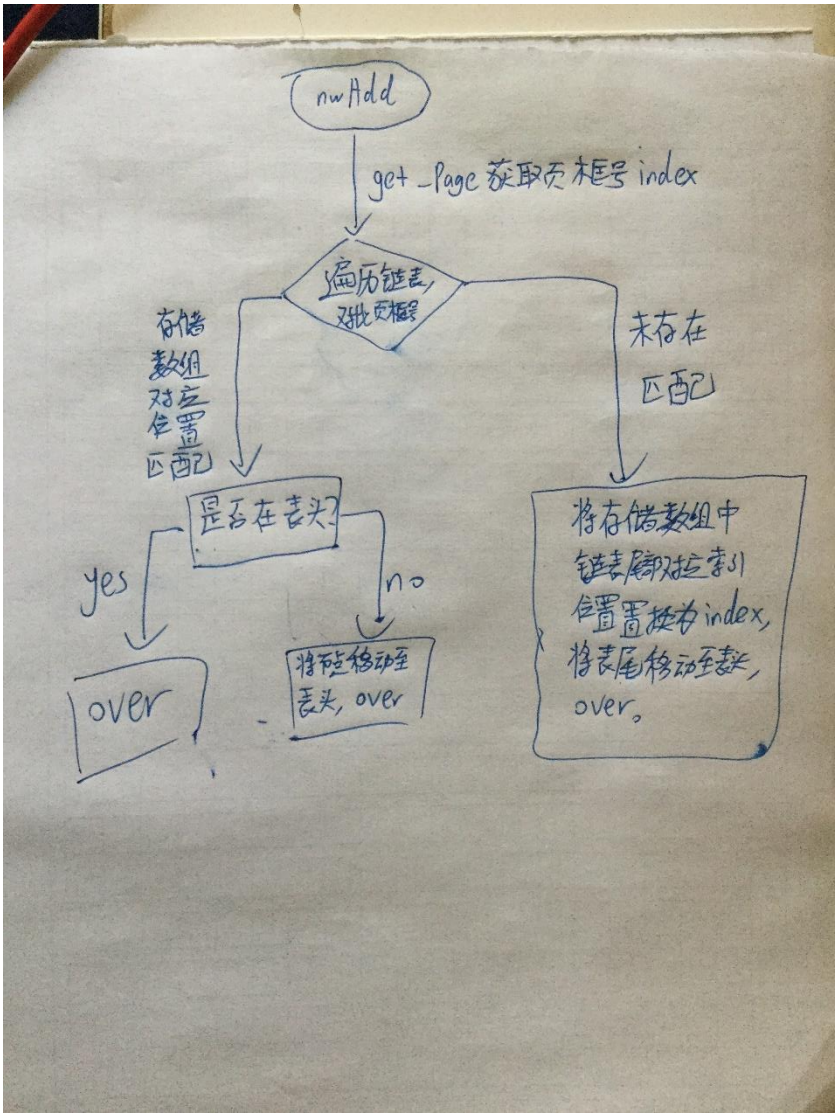
一、设计思想

LRU 算法，使用一个含 64 个节点的双向链表维护存储数组索引，每次将对应的索引移动至链表头部。如图所示。




二、实现技巧

查阅网上的算法，感觉 LRU 是效率最高的，然而大多都是采用 HashMap 进行存储的。这样在查找时效率极高。然而本题限定了使用单个纯数组进行存储，必须进行改编。哈希存储主要是实现了在数组中查找时惊人的效率，换句话说，在数组中定位极其简单。在不能采用哈希的情况下，如何解决在数组中的定位问题，成了本次算法优化的关键。我采取了双向链表的方式，每个存储数组索引，这样就无需记录时间，每次将最新被使用的位置换至表头即可。

三、优化与改进

我前后实现了 3 版函数。

1.纯数组模拟，每次遍历搜索，代码如下。

```
//
// Created by Conno on 2020/4/20.
//

#include "pageReplace.h"

#define MAX_PHY_PAGE 64
#define MAX_PAGE 12
#define get_Page(x) (x>>MAX_PAGE)

struct MC {
        long time;
        long num;
};

struct MC FGSB[MAX_PHY_PAGE];

int flag = 1;

int locate(long *memory, long VA) {
        for (int i = 0; i < MAX_PHY_PAGE; i++)
{
```

```
            if (get_Page(VA) == memory[i]) {
                return i;
            }
        }
        return -1;
}

void pageReplace(long *physic_memery,
long nwAdd) {
        int i, current = 0;
        if (flag) {
            for (i = 0; i < MAX_PHY_PAGE;
i++) {
                FGSB[i].num = 0;
                FGSB[i].time = -1;
            }
            flag = 0;
        }
        int pos = locate(physic_memery,
nwAdd);
        if (pos == -1) {
            for (int j = 0; j < MAX_PHY_PAGE;
```

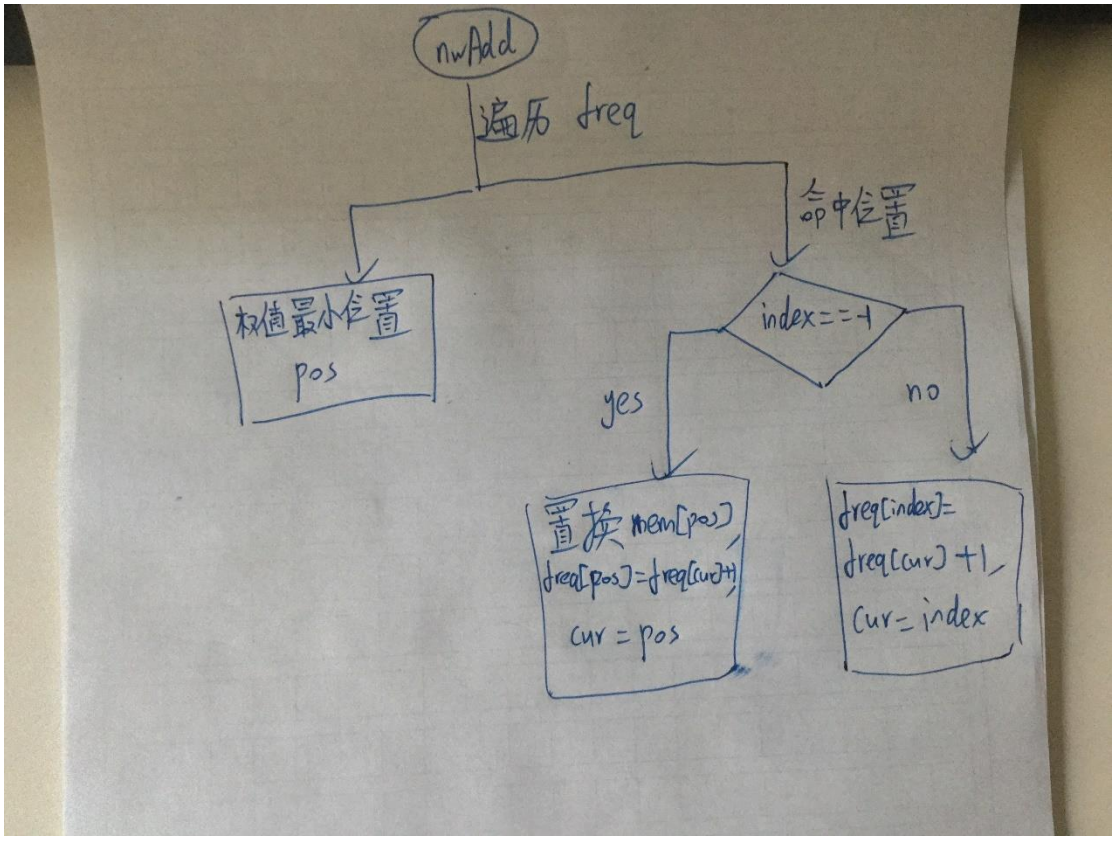
```
j++) {
                    if (FGSB[j].time == -1) {
                        current = j;
                        break;
                    } else if (FGSB[j].time >
FGSB[current].time) {
                        current = j;
                    }
                }
                physic_memery[current] =
get_Page(nwAdd);
                FGSB[current].num =
get_Page(nwAdd);
                FGSB[current].time = 0;
            } else {
                FGSB[pos].time = 0;
            }
            for (int j = 0; j < MAX_PHY_PAGE; j++)
{
                if (FGSB[j].num > 0) {
                    FGSB[j].time++;
                }
```

```
        }

        return;
}
```

初版基本就是 LRU 算法的直接翻译，其巨大问题在于基本没有任何优化，每次查找都要进行遍历，时间复杂度极高。我用该版本成功跑出了 1S+的佳绩，在所有提交中成功成为后 10%。

2.将存储数组映射到另一个 freq 数组上。Freq 记录的是存储数组对应位置的权值。每次遍历查找存储数组的同时遍历 freq，记录下 freq 中权值最小的位置 pos。若未命中，则将存储数组 pos 位置置换，并将 pos 位置权值+1，同时将该位置记录为 cur；若命中，则直接将 freq 中命中位置的权值置为 cur 位置权值+1，同时将命中位置记录为 cur。流程图与代码如下。




```
        //
// Created by Conno on 2020/4/20.
//
//#include "stdio.h"

#include "pageReplace.h"
#define MAX_PHY_PAGE 64
#define MAX_PAGE 12
#define get_Page(x) (x>>MAX_PAGE)

int freq[MAX_PHY_PAGE];
```

```
int cur, pos;

int locate(long *memory, long VA){
        for(int i = 0; i < MAX_PHY_PAGE; i++) {
                if (freq[i] < freq[pos]) {
                        pos = i;
                } // enough to go through the
whole freq[]???????
                if (get_Page(VA) == memory[i]) {
                        return i;
                }
        }
        return -1;
}

void pageReplace(long *physic_memery,
long nwAdd) {
        int index = locate(physic_memery,
nwAdd);
        if (index == -1 ) {
                physic_memery[pos] =
```

```
get_Page(nwAdd);
                freq[pos] = freq[cur] + 1;
                cur = pos;
        } else {
                freq[index] = freq[cur] + 1;
                cur = index;
        }

//        for (int i = 0; i < MAX_PHY_PAGE;
i++) {
//                printf("%ld ",
physic_memery[i]);
//        }
//        printf("\n");

        return;
}


//long fyou[MAX_PHY_PAGE];
//int main() {
//        long s;
```

```
//        while (scanf("%ld", &s)) {
//            if (s == -1) break;
//            pageReplace(&fyou, s);
//        }
//        return 0;
//}
```

该版本采取了记录时间戳与位置的理念，大幅提升了效率，一般的算法是采取每次置换则置换位置时间戳为 1，其他所有时间戳+1 的方法，这样每次置换的是时间戳最高的。然而这种方法每次都要再遍历数组进行自增操作，效率低下，因此我采用了相反的方法，每次只记录被置换位置与时间戳最低的位置，免除再次遍历，降低耗时。该版本排名为 50%左右，仍然不达标。为提升效率，降低耗时，最终采取了双向链表的方法。

3.代码如下

```
        //
// Created by Conno on 2020/4/22.
//
#include "time.h"
```

```
#include <stdio.h>
#include <stdlib.h>
//#include "pageReplace.h"
#define MAX_PHY_PAGE 64
#define MAX_PAGE 12
#define get_Page(x) (x>>MAX_PAGE)

struct FGSB{
	int pos;
	struct FGSB *prev;
	struct FGSB *next;
} *head;

int flag = 1;

void init(struct FGSB *p, struct FGSB *q,
long *memory, int *flag) {
	if ((*flag) == 1) {
		int i;
		for (i = 0; i < MAX_PHY_PAGE;
i++) {
			p = (struct FGSB
```

```
*)malloc(sizeof(struct FGSB));
                    p->pos = i;
                    p->prev = NULL;
                    p->next = NULL;
                    memory[i] = p->pos;
                    if (!head) {
                        head = p;
                        q = p;
                    } else {
                        p->prev = q;
                        q->next = p;
                    }
                    q = p;
                }
                (*flag) = 0;
        }
}

void pageReplace(long *physic_memery,
long nwAdd) {
        struct FGSB *p, *q, *temp;
        init(p, q, physic_memery, &flag);
```

```
        temp = head;
        long index = get_Page(nwAdd);
        while (temp->next) {
                if(physic_memery[temp->pos] ==
index){
                        if(temp == head) {
                                return;
                        } else {
                                temp->next->prev =
temp->prev;
                                temp->prev->next =
temp->next;
                                temp->next = head;
                                head->prev = temp;
                                temp->prev = NULL;
                                head = temp;
                                return;
                        }
                }
                temp = temp->next;
        }
        physic_memery[temp->pos] = index;
```

```
        head->prev = temp;
        temp->prev->next = NULL;
        temp->prev = NULL;
        temp->next = head;
        head = temp;
}

long fyou[MAX_PHY_PAGE];

int main() {
        long s;
        while (scanf("%ld", &s)) {
                pageReplace(&fyou, s);
                for (int i = 0; i < MAX_PHY_PAGE;
i++) {
                        printf("%ld ", fyou[i]);
                }
                printf("\n");
        }
}
```

四、本地测试情况

```
67786687
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
3 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 16549 2194
2147483647
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
3 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 524287 16549 2194
```