【18-4班作业】孙铭泽

一、证明线性筛使用一个Prime数组，不影响程序的正确性

is_prime数组起到了标记合数区分素数的作用

而只用prime[]单数组后通过

```
prime[++prime[0]]
```

这一步操作通过对prime[0]的更新起到了计数（质数个数的作用）

而后因为质数判断是从prime[2]开始的进行标记的而质数存储从prime[1]

开始，已用过标记prime数组不会影响到质数的存储所以不影响程序的正确性（完美错开一个身位QWQ）。

二、欧拉第10题

```
#include<stdio.h>
#include<inttypes.h>
#define MAX_N 2000000
int prime[MAX_N] = {0};
void init() {
    for(int i = 2; i * i <= MAX_N; i++) {
        if(prime[i]) continue;
        for(int j = i * i; j <= MAX_N; j += i) {
            prime[j] = 1;
        }
    }
}
int main() {
    init();
    int64_t ans = 0;
    for(int i = 2; i <= MAX_N; i++) {
        if(!prime[i])
        ans += i;
    }
    printf("%"PRId64 "\n",ans);
    return 0;
}
```

三、学习【缓存命中率】报告

存储器在计算机内部的组织方式如下图所示：

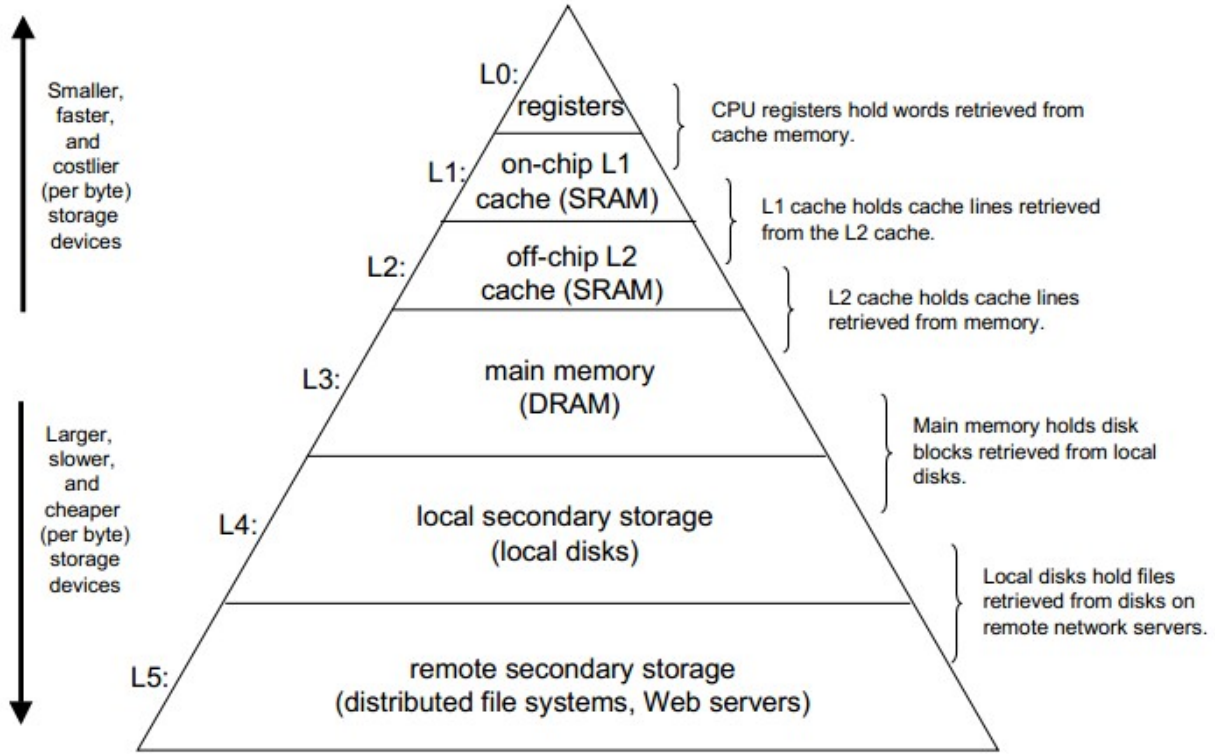


Figure 6.21: **The memory hierarchy.**

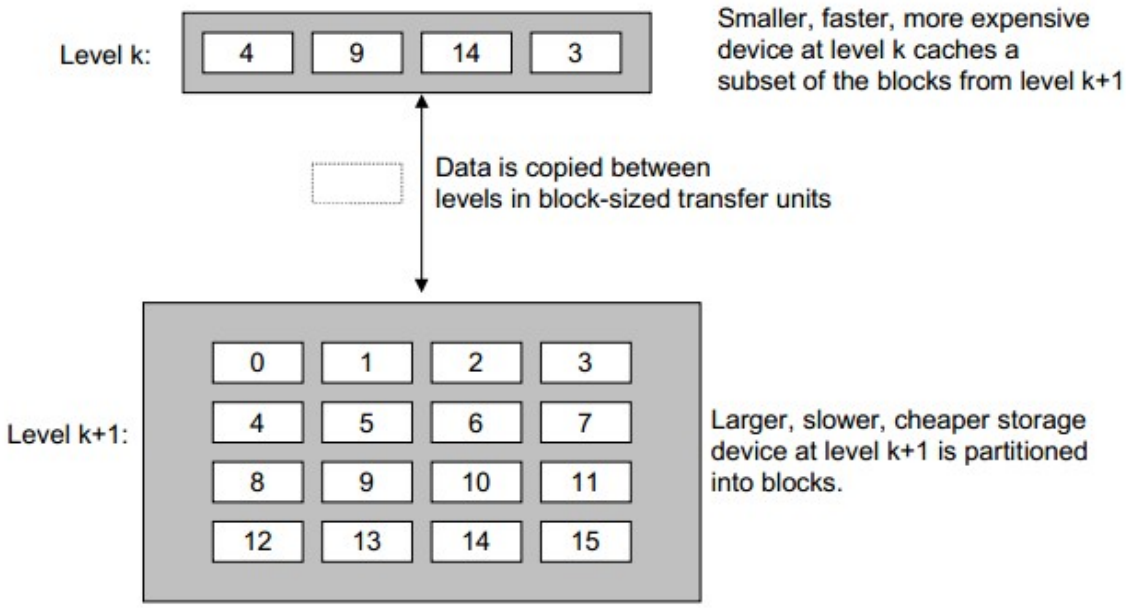
由上图可知：越往上，存储器的容量越小、成本越高、速度越快。

早期的时候存储器层次结构只有三层：cpu寄存器、DRAM主存以及磁盘存储。

由于CPU和主存之间巨大的速度差异，系统设计者被迫在CPU寄存器和主存之间插入了一个小的SRAM高速缓存存储器称为L1缓存，大约可以在2--4个时钟周期内访问。再后来发现L1高速缓存和主存之间还是有较大差距，又在L1高速缓存和主存之间插入了速度更快的L2缓存，大约可以在10个时钟周期内访问。

于是，在这样的模式下，在不断的演变中形成了现在的存储体系。速度快的存储器缓存了速度慢存储器的数据，对于每个k ，位于k层的更快更小的存储器设备作为第k+1层的更大更慢存储设备的缓存。就是说，k层存储了k+1层中经常被访问的数据。在缓存之间，数据是以块为单位传输的。当然不同层次的缓存，块的大小会不同。一般来说是越往上，块越小。

下图示例



k是k+1的缓存，他们之间的数据传输是以块大小为单位的。如上图中，k中缓存了k+1中块编号为 4、9、14、3的数据。

当程序需要这些块中的数据时，可直接冲缓存k中得到。这比从k+1层读数据要快

缓存命中

当程序需要第k+1层中的某个数据时d，会首先在它的缓存k层中寻找。如果数据刚好在k层中，就称为缓存命中。

缓存不命中

当需要的数据对象d不再缓存k中时，称为缓存不命中。当发生缓存不命中时，第k层的缓存会从k+1层取出包含数据对象d的那个块，如果k层的缓存已经放满的话，就会覆盖其中的一个块。至于要覆盖哪一个块，这是有缓存中的替换策略决定的，比如说可以覆盖使用频率最小的块，或者最先进入缓存的块。在k层从k+1层中取出数据对象d后，程序就能在缓存中读取数据对象d了。

关于缓存命中率的思考

$$\text{缓存命中率} = \frac{\text{缓存命中}}{\text{缓存命中}+\text{缓存不命中}}$$

结合欧拉地14题中记忆化搜索来看，随着宏定义的#define MAX_N值增大，被标记过直接输出结果的值更多，按惯性思维思考程序执行效率本应该提升，但事实上却出现了效率下降的结果，经过对系统缓存命中的学习我做出如下猜测：

1.在MAX_N足够大时，因为较小的第k层缓存本身大小不变，所储存数据的块有限，所以在进行记忆化搜索操作时可能想查找的值不在k层而在k + 1层的概率增加，公式分母中缓存不命中次数增加，缓存命中率降低。程序运行效率变低。

2.在MAX_N足够大时，因为较小的第k层缓存本身大小不变，所储存数据的块有限，如果存储在k层的缓存已经放满的话，就会覆盖其中的一个块。至于要覆盖哪一个块，这是有缓存中的替换策略决定的，所以说可能覆盖使用频率最小的块，或者最先进入缓存的块，这就造成了之后的某一次记忆化搜索结果不再是从k层寻找而是在k + 1层中。由此造成了分子中缓存命中率下降，分母中缓存命中下降缓存不命中增加，从而缓存命中率降低。程序运行效率的变低。

四、欧拉第17题

```c
#include<stdio.h>
int length20[20] = {
    0, 3, 3, 5, 4, 4, 3, 5, 5, 4, 3,
    6, 6, 8, 8, 7, 7, 9, 8, 8 };//前二十位数量
int length10_bit[10] = {
    0, 0, 6, 6, 5, 5, 5, 7, 6, 6};//整十位数量
int exchenge_length(int x) {
    if(x < 20) {
        return length20[x];
    } else if(x < 100) {
        return length10_bit[x / 10] + length20[x % 10];
    } else if(x < 1000) {
        //大于等100 小于1000这一部分存在两种情况，一种是整百不需要加and，另一种需要加and所以加3
        if(exchenge_length(x % 100)) {
            return exchenge_length(x % 100) + 3 + length20[x / 100] + 7;
        } else {
            return  exchenge_length(x % 100) + length20[x / 100] + 7;
        }
    } else {
        return 11;
    }
}
int main() {
    int ans = 0;
    for(int i = 1; i <= 1000; i++) {
        ans +=  exchenge_length(i);
    }
    printf("%d\n",ans);
    return 0;
}
```

五、欧拉第16题

```c
#include<stdio.h>
int main() {
    int ans[3500];
    ans[0] = 4;
    ans[1] = 4;
    ans[2] = 2;
    ans[3] = 0;
```

```
    ans[4] = 1;
    for(int i = 1 ; i <= 99; ++i) {
        for(int j = 1; j <= ans[0]; ++j) {
            ans[j] *= 1024;
         }
        for(int k = 1; k <= ans[0]; ++k) {
            if(ans[k] >= 10) {
                ans[k + 1] += ans[k] / 10;
                ans[k] %= 10;
                if(k == ans[0])
                ans[0] += 1;
            }
        }
    }
     int sum = 0;
    for(int i = 1; i <= ans[0]; i++) {
        sum += ans[i];
    }
    printf("%d\n",sum);
    return 0;
}
```