Lab7 问题分析

赵耀

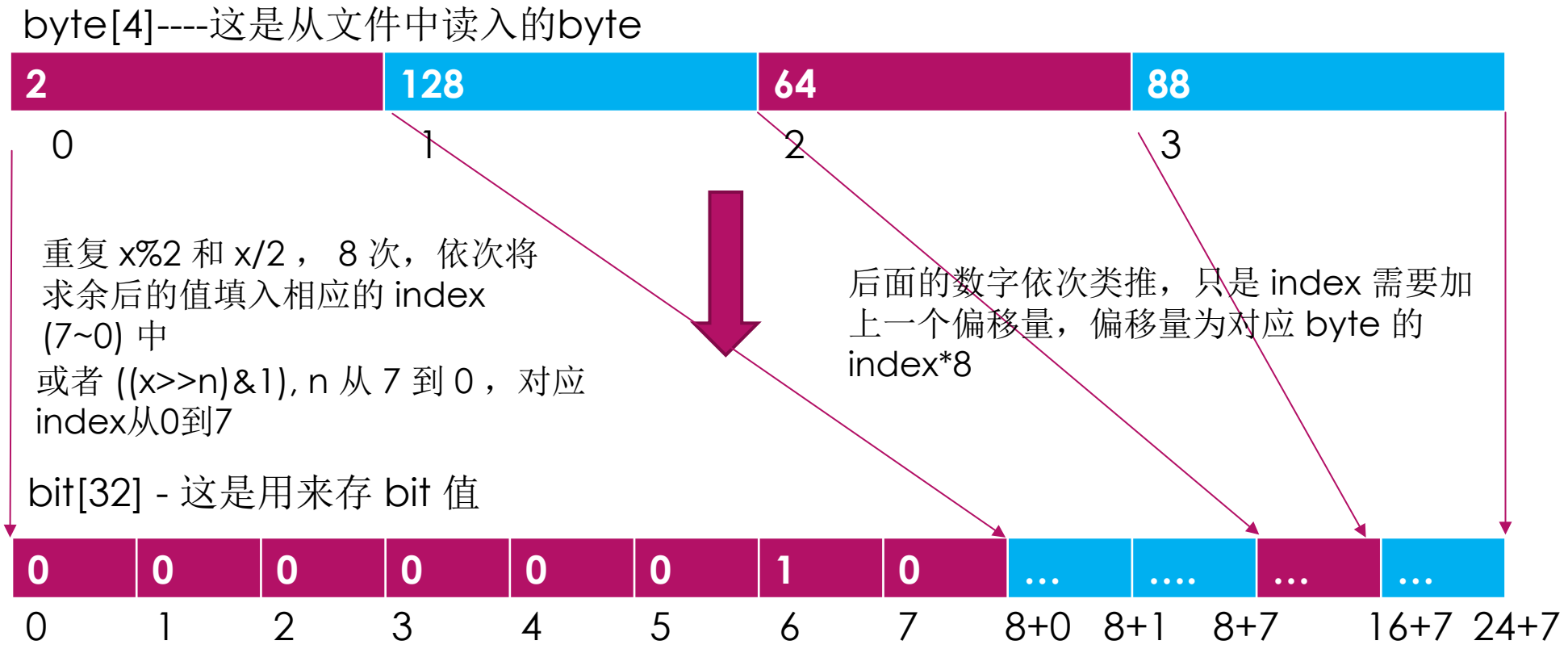
Huffman 压缩容易出现的问题

- Byte (8 bits) 的范围是 0~255，有的同学认为是 ASCII 码，写成了范围 0~127，虽然例子举的是英文文本，然而实际被压缩的文件也许是图片，也许是 pdf 。测试数据是按照 Byte 读待测文件的。
- 统计完 Byte 出现的次数，有的同学使用一个 a[256] 的数组，每个 Byte 值对应数组下标，元素值为统计次数，此时需注意，统计次数为 0 的 Byte 值不需要为其进行 Huffman 编码
- 编码过程《Huffman编解码》有详细描述
- https://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/

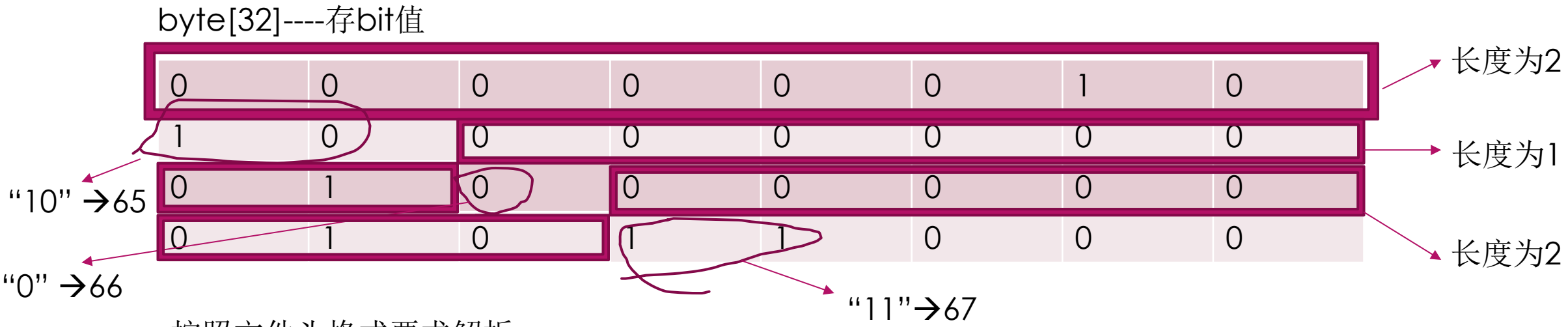
Huffman 解压容易出现的问题

- 将 Byte 流转为 bit 流
- 从 bit 流正确读出 Byte 值到编码的映射，构建码表
- 正确读出压缩后的 bit 流长度，并将压缩后的 bit 码表还原原始信息

示例：byte 到 bit


byte[4]----这是从文件中读入的byte
2
128
64
88
0
1
2
3
重复 x%2 和 x/2， 8 次，依次将求余后的值填入相应的 index (7~0) 中
或者 ((x>>n)&1), n 从 7 到 0，对应index从0到7
后面的数字依次类推，只是 index 需要加上一个偏移量，偏移量为对应 byte 的 index*8
bit[32] - 这是用来存 bit 值
0
0
0
0
0
0
1
0
...
....
...
...
0
1
2
3
4
5
6
7
8+0
8+1
8+7
16+7
24+7

Bit 到 Int 和编码:提取码表



按照文件头格式要求解析：
1、依次读出8个值，换算成整数：
2、根据该整数x，依次取x个值，转成对应的bit string
3、重复该过程

上述省略了前 65 个 byte 和后 188 个 byte 对应编码的解析

长度及加密后 bit 的解析

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1
0	0	0	0	1	1	0	0

0*2^31+0*2^30+….+1*2^3+0*2^2+0*2^1+1*2^0 = 9

10	65
0	66
11	67

接下来解析长度为 9 的加密后 bit

读入第一个bit，转成字符

Map 中是否有该字符？没有则读入下一个字符继续判断，有则 byte 数 ++

Lab6 代码问题

- Prim 邻接矩阵赋值及更新典型问题

```
}
for(int i=0;i<m;i++){
    int a = in.nextInt();
    int b = in.nextInt();
    map[a-1][b-1] = in.nextInt();
    k[a-1]++;
    k[b-1]++;
}
```

```
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        if(map[i][j]!=MAX){
            map[i][j] = map[i][j]*(Math.abs(v[i]-v[j])+1)*(Math.abs(k[i]-k[j])+1);
        }
    }
}
```

- 正确

```
    for(int i=0;i<m;i++){
        int a = in.nextInt();
        int b = in.nextInt();
        map[a-1][b-1] = in.nextInt();
        map[b-1][a-1] = map[a-1][b-1];
        k[a-1]++;
        k[b-1]++;
    }
```

```
for(int i=0;i<n;i++){
    for(int j=i;j<n;j++){
        if(map[i][j]!=MAX){
            map[i][j] = map[i][j]*(Math.abs(v[i]-v[j])+1)*(Math.abs(k[i]-k[j])+1);
            map[j][i] = map[i][j];
        }
    }
}
```

Kruskal 超时 1

- Kruskal 对边排序

```
// 冒泡排序
for (int j = i - 1; j >= 0; j--) {
    if (finishsort[j+1].weight < finishsort[j].weight) {
        int node1temp = finishsort[j].node1;
        int node2temp = finishsort[j].node2;
        int dtemp = finishsort[j].d;
        int weighttemp = finishsort[j].weight;
        finishsort[j].node1=finishsort[j+1].node1;
        finishsort[j].node2=finishsort[j+1].node2;
        finishsort[j].d=finishsort[j+1].d;
        finishsort[j].weight=finishsort[j+1].weight;
        finishsort[j+1].node1=node1temp;
        finishsort[j+1].node2=node2temp;
        finishsort[j+1].d=dtemp;
        finishsort[j+1].weight=weighttemp;
    }

}
```

冒泡排序：O(n^2)可以优化到O(nlogn)

Kruskal 超时 2

- 以下流程将检查所有的边，实际只需要 n-1条边就可以终止

```
for(int i=0;i<m;i++){
    root1=finishsort[i].node1;
    root2=finishsort[i].node2;
    while(roottable[root1]!=0){
        root1=roottable[root1];
    }
    while(roottable[root2]!=0){
        root2=roottable[root2];
    }

    if(root1==0&&root2==0){

        roottable[root2]=root1;
        w[root1]++;
        result+=finishsort[i].weight;
    }else if(root1!=0&&root2==0){
        roottable[root2]=root1;
        result+=finishsort[i].weight;
    }else if(root1==0&&root2!=0){
        roottable[root1]=root2;
        result+=finishsort[i].weight;
    }else{
```

```
int cost = 0;
group = new int[n+1];
while(!heap.isEmpty()) {
    Edge e = heap.poll();
    int v = e.v;
    int w = e.w;
    int groupv = findGroup(v);
    int groupw = findGroup(w);
    if(groupv!=0 && groupw!=0 && groupv==groupw)
        continue;
    else if(groupv==0 && groupw==0) {
        group[v] = v;
        group[w] = v;
        cost += e.value;
    }
    else if(groupv==0 && groupw!=0) {
        group[v] = groupw;
        cost += e.value;
    }
    else if(group[v]!=0 && group[w]==0) {
        group[w] = groupv;
        cost += e.value;
    }
    else if(groupv != groupw) {
        group[groupv] = groupw;
        cost += e.value;
    }
}
```

Kruskal 超时 2

```
int cost = 0;
group = new int[n+1];
while(!heap.isEmpty()) {
    Edge e = heap.poll();
    int v = e.v;
    int w = e.w;
    int groupv = findGroup(v);
    int groupw = findGroup(w);
    if(groupv!=0 && groupw!=0 && groupv==groupw)
        continue;
    else if(groupv==0 && groupw==0) {
        group[v] = v;
        group[w] = v;
        cost += e.value;
    }
    else if(groupv==0 && groupw!=0) {
        group[v] = groupw;
        cost += e.value;
    }
    else if(group[v]!=0 && group[w]==0) {
        group[w] = groupv;
        cost += e.value;
    }
    else if(groupv != groupw) {
        group[groupv] = groupw;
        cost += e.value;
    }
}
```

```
int count = 0;
while(!heap.isEmpty()  count<n-1)
    Edge e = heap.poll();
    int v = e.v;
    int w = e.w;
    int groupv = findGroup(v);
    int groupw = findGroup(w);
    if(groupv!=0 && groupw!=0 && groupv==groupw)
        continue;
    else if(groupv==0 && groupw==0) {
        group[v] = v;
        group[w] = v;
    }
    else if(groupv==0 && groupw!=0) {
        group[v] = groupw;
    }
    else if(group[v]!=0 && group[w]==0) {
        group[w] = groupv;
    }
    else if(groupv != groupw) {
        group[groupv] = groupw;
    }
    count++;
    cost += e.value;
}
```

通过

Kruskal 死循环

```
    parent = new int[n + 1];
    for (int i = 0; i < n + 1; i++) {
        parent[i] = -1;
    }
```

```
public static int Find(int x) {
    int s;
    for (s = x; parent[s] > 0; s = parent[s]) {
        while (s != x) {
            int tmp = parent[x];
            parent[x] = s;
            x = tmp;
        }
    }
    return s;
}
```

```
public static void Union(int R1, int R2, int r1, int r2) {
    int tmp = parent[R1] + parent[R2];
    if (parent[R1] > parent[r2]) {
        parent[r1] = r2;
        parent[r2] = tmp;
    } else {
        parent[r2] = r1;
        parent[r1] = tmp;
    }
}
```

Union-Find algorithm

正确

```
int Find(int x)
{
    if (x == fa[x])
        return x;
    else
        return fa[x] = Find(fa[x]);
}

void Union(int x, int y)
{
    x = Find(x);
    y = Find(y);
    fa[x] = y;
}
```

Union-Find algorithm to detect cycle. So we recommend to read following post as a prerequisite.
Union-Find Algorithm | Set 1 (Detect Cycle in a Graph)
Union-Find Algorithm | Set 2 (Union By Rank and Path Compression)

- 其他算法流程相关错误，建议好好读下发出的 Good Code