作业讲解（一）

贪心算法、递归与搜索、链表

陈宇琪

2020 年 4 月 20 日

Content

Greedy

More About Haffman Coding

- 如何计算哈夫曼编码之后的长度。
- 核心思想：哈夫曼每次合并可以理解为将两个单词合并成一个新的单词/词组，这个单词/词组的词频就是原先两个单词的词频的和。
- 算法实现：一个优先级队列即可，没有必要建立哈夫曼树进行求解。
- 优先级队列的使用：由于需要每次选择词频最小的两个节点，所以想到了优先级队列。
- 为什么不用 multiset：multiset 当然可以，就是会慢一点。

Greedy

More About Haffman Coding

- 注意一些特殊情况：由于算法是每次选择两个单词来处理的，所以如果只有一种单词呢》
- 对于只有一种单词的情况，答案就是它的词频！
- 注意优先级队列要改成小根堆。

Greedy

More About Haffman Coding

Listing 1: **Haffman.cpp**

```
int huffman() {
        priority_queue <int, vector<int>, greater<int> > q;
        // read string and calculate word frequency
        if(q.size() == 1) {
                return q.top();
        }
        int ans = 0;
        while(q.size() > 1) {
                int a = q.top(); q.pop();
                int b = q.top(); q.pop();
                q.push(a + b);
                ans += a + b;
        }
        return ans;
}
```

Greedy

More About Haffman Tree

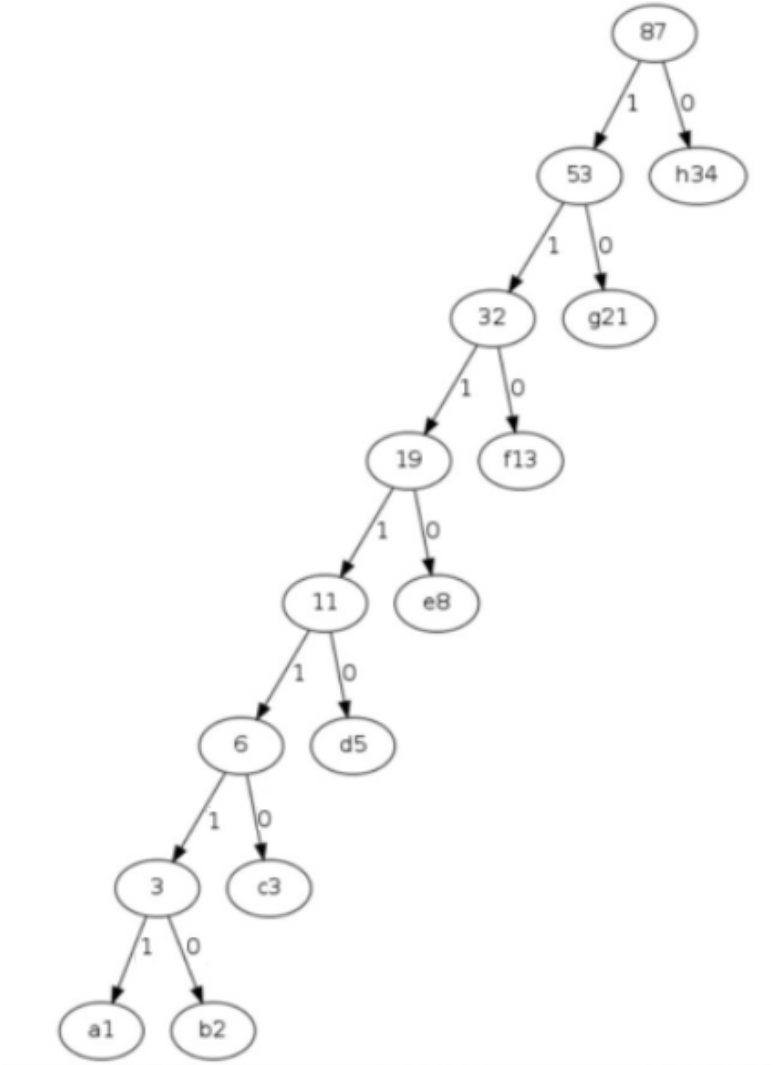

图: Haffman Tree

Greedy

More About Haffman Tree

- 哈夫曼树是一颗字典数。
- 哈夫曼树对应哈夫曼编码。
- 每个字母对应的哈夫曼编码为根到这个叶子节点的路径上的值组成的二进制串。
- 在这个例子中，a 对应的哈夫曼编码为 1111111。
- 哈夫曼编码通常不唯一。

Greedy

More About Greedy

- 同时有 n 个人同时开始排队做某件事情，为了使排队浪费的所有人的总时间最少，所以让简单的做的快的先做
- 贪心思想：每一次都让还没做正在等待的人等的时间最短，也就是在剩下的时长中选择时间最短的先执行。
- 证明：假设对于最优解 o，队列每个人完成时间为 t_i，假设存在 $t_i \geq t_{i+1}$，令 $o^{'} = \{t_1, t_2, ...t_{i+1}, t_i, ...t_n\}$，假设 $T(o)$ 表示队列为 o 时，所有人等待时间的总和，由于 o 是最优解，所以 $T(o) \leq T(o^{'})$，由于 $T(o) - T(o^{'}) = t_i - t_{i+1} \geq 0$，所以 $T(o) = T(o^{'})$，所以**交换序列中的逆序对不会使得结果变差**，所以一个合理的贪心解为使得队列升序排列后的结果。
- 分析：为什么选择相邻的交换？因为这样影响的只有被交换的两个数。
- **注意：不是所有贪心问题都可以容易得到证明！**

Greedy

More About Greedy

- 做作业，为了做完更多的作业，先做比较简单的，很快就很写完的，
- 例如：EOJ 刷水题?

Greedy

More About Greedy

- 背景：当考试的时候，**每道题目要么满分，要么 0 分**。假设在都能做对的前提下，如何选择题目的顺序从而使得在考试时间内获得更高的分数。
- 贪心思想：**可以通过单位时间获得的分数来得到最优解**。每次从剩下的题目中选择单位时间内得分更高的题目，便可达到局部最优。
- 如果有一题性价比很高，但是做不完怎么办？
- 如果按照这样找到的最优解导致有空余时间怎么办？
- 解决：将问题转成一个背包问题（需要用到动态规划）。
- 问题抽象：假设每个物品 (x_i, y_i)，最优化下面的目标函数：

$$\max \sum_{i=1}^{N} w_i * y_i$$

$$s.t. \sum_{i=1}^{N} w_i * x_i \leq T, w_i \in \{0, 1\}$$

Greedy

More About Greedy

- 背景：当考试的时候，**每道题目有部分分，且部分分的多少和在这道题上消耗的时间成正比。**假设在都能做对的前提下，如何选择题目的顺序从而使得在考试时间内获得更高的分数。
- 贪心思想：**可以通过单位时间获得的分数来得到最优解。**每次从剩下的题目中选择单位时间内得分更高的题目，便可达到局部最优。
- 问题抽象：假设每个物品 (x_i, y_i)，最优化下面的目标函数：

$$\max \sum_{i=1}^{N} w_i * y_i$$

$$s.t. \sum_{i=1}^{N} w_i * x_i \leq T, 0 \leq w_i \leq 1$$

Greedy

More About Greedy

- 给定 n 个数 $a_1, a_2, ...a_n$，输出 $2n$ 个数 $b_1, b_2, ..., b_{2n}$，满足 $a_i = \min(b_{2i-1}, b_{2i})$ 并且 $b_1, b_2, ..., b_{2n}$ 是 1 到 $2n$ 的一个排列，如果有多解，请输出字典序最小的解，如果无解，请输出-1。
- 考虑如果 $b_{2i-1} > b_{2i}$ 则交换 b_{2i-1} 和 b_{2i} 条件仍然满足，而且字典序变小。所以 $b_{2i-1} < b_{2i}$，又因为 $a_i = \min\{b_{2i-1}, b_{2i}\}$，所以 $b_{2i-1} = a_i$。
- 其次为了使得 b_{2i} 尽可能小，所以 b_{2i} 应该选择剩下的可以填的数中第一个大于 a_i 的数。
- 我们需要一个能够求出第一个大于 x 的数，并且支持删除一个数，所以我们想到了使用 set 来完成。
- 注意：输入的数可能会有重复的数。
- 提示：**根据题目的需要设计使用什么数据结构！**

Search

More about BFS

- 对于一个简单的 BFS 问题，如果一个状态可以延展出 n 个状态，并且我们需要穷举所有 m 步可以到达的状态，那么整个算法复杂度是多少?
- 复杂度为 $O(1 + n^1 + n^2 + ... + n^{m-1})$，对于 $n = 1$ 的时候复杂度为 $O(m)$，对于 $n \geq 2$ 的时候复杂度可以认为是 $O(n^m)$。
- 搜索可以理解为暴力，所以复杂度一般不低，需要一些剪枝技巧!

Search

More about BFS

- 双向广搜：假设起始状态和终止状态已知，如果优化广搜的算法复杂度。
- 分析：从起点和终点分别进行搜索，每次从起点搜索一步，在从终点搜索一步，直到某个状态相遇。
- 这样假设从起点到终点需要 m 步，每个状态可以延展出 n 个状态，复杂度为 $O(n^{m/2})$
- 状态记录：set 或者 map 都可以。
- 应用：八数码问题（也可以用启发式搜索实现）

Search

6.2 迷宫问题

- 经典的 BFS 问题：使用队列实现 BFS 的过程，每次从队列中取出一个状态，将这个状态所能拓展的 4 个状态加入队列(在加入之前判断新状态是否合法)，并更新到起点的距离。
- 注意：每个状态只能拓展一次，**需要一个 bool 数组记录拓展过哪些状态。**

- 关于内存分配问题，由于题目中只有 $m \times n \leq 10^6$，所以一种做法是使用 vector。

Listing 2: **Vector.cpp**

```
vector <int> v[1000000+5];

int bfs(int n,int m) {
        for (int i=0;i<n;i++) {
                v[i].resize(m);
        }
}
```

- 关于内存分配问题，题目中给出 $m \times n \leq 10^5$（由于出题的失误，所以稍作修改），但是仍然可以用数组。
- 由于 $m \times n \leq 10^5$，所以 $\min(m, n) \leq \sqrt{10^5}$。
- 看上去无聊，但也是一种思想。

Listing 3: **Array.cpp**

```
const int maxn=1e5+5;
const int maxm=sqrt(maxn)+1;

int a[maxn][maxm];
void bfs(int n,int m) {
        if (n<m) swap(n,m);
}
```

Search

More about DFS

- 给出 n 个正整数，问有多少种方法在这 n 个数字的中取其中一些数字，使得这些数字之和超过 k。若答案超过 2000000，输出 -1。
- 暴力复杂度是多少？$O(2^n)$。
- 如何优化?

Listing 4: **DFS.cpp**

```
int tot;

void dfs(int dep,int sum) {
        if (dep==n) {
                if (sum>k)
                        tot++;
                return;
        }
        dfs(dep+1,sum+a[dep]); // Select a[dep]
        dfs(dep+1,sum);
}
```

Search

More about DFS

- 给出 n 个正整数，问有多少种方法在这 n 个数字的中取其中一些数字，使得这些数字之和超过 k。若答案超过 2000000，输出 -1。
- 1、tot 大于 2000000，程序直接终止。
- 2、从大到小排序：尽可能使得和超过 k。
- 3、尽可能先考虑选这个数的情况：同样尽可能使得和超过 k。
- 4、假设和已经超过 k 了，这个时候还剩 m 个数，则这 m 个数可取可不取，所以对答案贡献为 2^m，特别的如果 $m > 25$，则 2^m 已经超过上界了。
- 还有吗?

Search

More about DFS

- 给出 n 个正整数，问有多少种方法在这 n 个数字的中取其中一些数字，使得这些数字之和超过 k。若答案超过 2000000，输出 -1。
- 如果所有数很小怎么办？那么复杂度还是 $O(2^n)$。
- 5、如果剩下的数加上现在的和还小于等于 k，则这种状态不可能对结果有影响。(可行性剪枝)
- 什么时候剪枝算完成？实在想不出来了后者 AC 了就算完成了！
- 这样剪枝后复杂度是多少呢？不知道，应该还是 $O(2^n)$ 吧。

Search

A Knight's Journey

- 如何在搜索过程中找到字典序最小的解?
- 解法 1：找到最后解之后排序（效果太低)。
- 解法 2：贪心地搜索：没有优先考虑字典序最小的状态进行拓展，这样找到一个解就是字典序最小的解。
- 例如：当前状态为 1，后续状态为 2,3,4，则先搜索 2，如果 2 有解的话，直接返回，否则再搜索 3，以此类推。

Linked Structure

More about Linked Queue

- 用链接方式存储的队列，在进行删除运算时（　）。
- A. 仅修改头指针 B. 仅修改尾指针
- C. 头、尾指针都要修改 D. 头、尾指针可能都要修改
- 分析：这里的队列就是传统的链式队列。在这种情况下，当链表只有一个元素时，删除头部元素的时候，需要同时修改 head 和 tail 为空指针。

Linked Structure

More about Linked Queue

- 如何实现 Linked Queue?
- **在尾部插入，在头部删除。**
- 由于这是单链表，所以只能在头部删除，在尾部删除是不可以的!
- 可以维护尾部的前一个元素的位置？可以应该可以，但是不会这么做!

Linked Structure

More about Linked Queue

- 重载运算符时候需要特别注意!
- if (&other != this) 如果没有的话，后果是什么?

Listing 5: **operator.cpp**

```
Queue<T>& operator = (const Queue<T>& other){
        if (&other != this) {
                if (head != NULL) {
                        clear();
                }
                auto p = other.head;
                for (uint32_t i = 0; i < other._size; i++) {
                        push(p->data);
                        p = p->next;
                }
                return *this;
        }
        else {
                return *this;
        }
}
```

Linked Structure

More about Linked Queue

- swap 只要交换指针就可以了!

Listing 6: **swap.cpp**

```
void swap (const Queue<T>& other){
        swap(this->head,other.head);
        swap(this->tail,other.tail);
        swap(this->_size,other._size);
}
```

Linked Structure

More about Linked Queue

出现的错误：

- 1、重载运算符封装错误。
- 2、pop，size 复杂度错误（pop 复杂度错误由于出栈方式错误，size 复杂度错误由于没有动态维护 size 的值）。
- 3、删除元素之后没有 delete，导致内存泄漏。

不是很好的做法：

- 1、只写 main 函数（这样代码重复很厉害，不利于代码调试）。
- 2、使用 list 或者 queue 或者数组模拟（不符合要求）。
- 3、swap 使用标记实现（因为现在只有两个，如果有很多呢?）。

Linked Structure

More about Linked Queue

- 使用单链表实现“删除链表的倒数第 N 个节点”。
- 使用快慢指针算法，先让快指针前进一段距离，然后快慢指针同时开始向前移动，当快指针到最后时，慢指针指向的下一个位置就是要删除的位置。

主要错误：

- 1、删除要分为在头部删除和中间删除，或者可以采用悬空头指针的做法，先在头指针前加一个空指针。
- 2、没有考虑 $N \leq 0$ 和 N 过大的情况。
- 3、不允许使用数组或者队列等数据结构辅助完成。
- 4、删除节点后忘记释放内存导致内存泄漏问题。

Quick Sort

More about Quick Sort

- Divide: Partition the array into two subarrays around a pivot x such that elements in lower subarray $\leq x \leq$ elements in upper subarray
- Conquer: Recursively sort the two subarrays.
- Combine: Trivial (because in place).

Quick Sort

More about Quick Sort

图: Quick Sort

从图中可以看出快排每次分治都选择数组中的一个数作为 pivot，将 $\leq$pivot 的数放在 pivot 左边，将 $\geq$pivot 的数放在 pivot 后边，当然对于等于 pivot 的数放在哪里都不影响结果，之后递归求解图中蓝色和黄色部分的数组即可。**也可以理解为每次将 pivot 放到它该在的位置上。**

Quick Sort

More about Quick Sort

快排的关键是寻找 $O(N)$ 的 Partition 算法。

Listing 7: **quick sort.cpp**

```
Quicksort(A,p,r)
{
        if (p<r)
        {
                q = Partition(A,p,r);
                Quicksort(A,p,q-1);
                Quicksort(a,q+1,r);
        }
}
```

代码中的 q 就是 pivot 所在的位置。注意代码中假设当前待排序区间为 $[p, r]$。

Quick Sort

More about Quick Sort

在一个数组中找到 $\leq$pivot 的数和 $>$pivot 的数可以使用辅助 vector 数组通过一个遍历求得。

Listing 8: **naive partition.cpp**

```
int partition(vector <int> &a,int p,int r) {
    int k=a[r],tmp=a[r];
    vector <int> lhs;
    vector <int> rhs;
    for (int i=p;i<=r;i++) {
        if (i==k) continue;
        if (a[i]<=a[k]) lhs.push_back(a[i]);
        else rhs.push_back(a[i]);
    }
    int i=p;
    for (int j=0;j<lhs.size();j++) a[i++]=lhs[j];
    int q=i;
    a[i++]=tmp;
    for (int j=0;j<rhs.size();j++) a[i++]=rhs[j];
    return q;
}
```

Quick Sort

More about Quick Sort

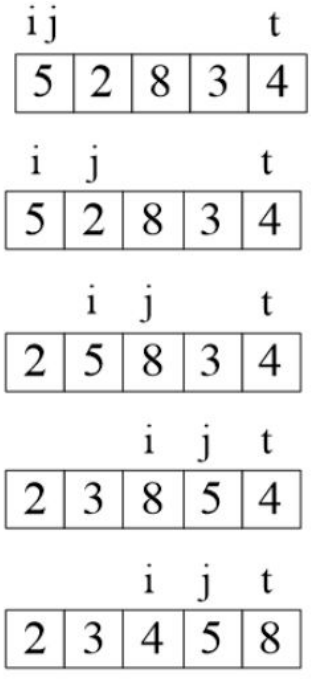

图: Quick Sort