基础数据结构和STL杂题选讲

Saya

洛谷
www.luogu.com.cn

std::string

动态分配空间

下标从 0 开始

常用函数:

```
s.find(str,pos)
s.substr(pos,len)
s.insert(index,count,ch)/s.insert(index,str)
s.erase(index,count)
s.replace(pos,count,str)/s.replace(first,last,str)
```

std::string

重载了加法运算符和比较运算符

```
a="abc"; b="def"
s=a+b; // s == "abcdef"
```

==：两个串完全相等

比较符号：比较两个串的字典序（即从前往后一个个字符比较）

B3997 小洛的字符串分割

给定一个字符串 S，按顺序将其切割成长度分别为 $1,2,3\cdots$ 的片段，除最后一段外，均符合第 i 段长度为 i，最后一段为这样切割后剩下的段。计算这些段中有多少是回文串。

对于所有数据，$|S| \le 10^6$。$|S|$表示字符串 S 的长度。

例如说，对于字符串 aaababcaacd，会被分为如下的五个字符串：

a/aa/bab/caac/d

这五个字符串都是回文的。

Solution

使用C++中 string 容器的 substr 函数可以方便地取出一个子串，并且该函数还有自动截断的功能，这样可以方便地处理最后剩下的一个段。

```cpp
int main(){
    string s = "abcdef";
    cout << s.substr(0, 3) << "\n";
    cout << s.substr(5, 3) << "\n";
    return 0;
}
```

```
abc
f

--------------------------------
Process exited
请按任意键继续
```

使用该函数取出一个段后再判断这个段是否为回文串。

判断是否为回文串：枚举i，检查是否每个 t[i] 都等于 t[t.size()-1-i]

代码

```cpp
bool check(string s) {
    for (int i = 0; i < s.size() - 1 - i; i++)
        if (s[i] != s[s.size() - 1 - i])
            return false;
    return true;
}

int main() {
    string s; cin >> s;
    int ans = 0;
    for (int i = 0, j = 1; i < s.size(); i += j, j++) {
        if (check(s.substr(i, j)))
            ans++;
    }
    cout << ans << "\n";
    return 0;
}
```

B3990 通配符匹配

我们称字符 a, b 匹配，当且仅当 $a = b$ 或 a, b 的其中一者为 ?。

我们称两个字符串 s, t 匹配，当且仅当 $|s| = |t|$，并且对应位置上的字符匹配。

比如说字符串“a?c” 会与 “ab?” 匹配。

给定两个串 s, t，规定 $s(l, r)$ 表示 s 的第 l 个字符到第 r 个字符组成的字符串（$1 \le l \le r \le |s|$）。例如：若 s="luogu"则 s(2,4)="uog"。

请你求出所有的 l, r，满足 t 和 $s(l, r)$ 匹配。

Solution

按顺序枚举 s 中每一个和串 t 等长的子串，判断这个子串和 t 是否匹配，若匹配则按格式输出。

时间复杂度 $O(n^2)$。

```
bool check(string s, string t) {
    for (int i = 0; i < s.size(); i++)
        if (s[i] != t[i] && s[i] != '?' && t[i] != '?')
            return false;
    return true;
}
int main() {
    string s, t;
    cin >> s >> t;
    for (int i = 0; i + t.size() - 1 < s.size(); i++) {
        if (check(s.substr(i, t.size()), t))
            cout << i + 1 << " " << i + t.size() << "\n";
    }
    return 0;
}
```

B3843 密码合规

网站注册需要有用户名和密码，编写程序以检查用户输入密码的有效性。合规的密码应满足以下要求：

1. 只能由 $a \sim z$ 之间 26 个小写字母、$A \sim Z$ 之间 26 个大写字母、$0 \sim 9$ 之间 10 个数字以及 !@#$ 四个特殊字符构成。
2. 密码最短长度 6 个字符，密码最大长度 12 个字符。
3. 大写字母，小写字母和数字必须至少有其中两种，以及至少有四个特殊字符中的一个。

输入一行不含空格的字符串。约定长度不超过 100。该字符串被英文逗号分隔为多段，作为多组被检测密码。输出所有合规密码。

输入样例

seHJ12!@,sjdkffH$123,sdf!@&12HDHa!,123&^YUhg@!

输出样例

seHJ12!@

sjdkffH$123

Solution

先将原字符串按照逗号划分为若干个密码。

对于每个划分出来的字符串 s，分别进行判断。

按照题目中的要求检查密码长度，检查字符是否都合法，用变量 a,b,c,d 的 0/1 值表示四种字符是没出现/有出现，

特殊字符有出现即 d==1，剩下三种字符出现至少两种即 a+b+c>=2。

```
void check(string s) {
    if (s.size() < 6 || s.size() > 12) return ;
    int a = 0, b = 0, c = 0, d = 0;
    for (int i = 0; i < s.size(); i++) {
        if ('A' <= s[i] && s[i] <= 'Z') a = 1;
        else if ('a' <= s[i] && s[i] <= 'z') b = 1;
        else if ('0' <= s[i] && s[i] <= '9') c = 1;
        else if (s[i] == '!' || s[i] == '#' || s[i] == '@' || s[i] == '$')
            d = 1;
        else return ;
    }
    if ((a + b + c) >= 2 && d) cout << s << "\n";
}
```

std::vector

- `vector<T> a(n)`
- `vector<vector<T>> a(n, vector<T> m)`
- `a.push_back()`
- `a.pop_back()`
- `a.front()`
- `a.back()`
- `a.erase(first, end)`
- `a.resize(size, value)`

B3665 小清新数据结构题

- n 条数据，第 i 条数据有 $s[i]$ 个数，分别为 $a[i][1]$~$a[i][s[i]]$
- q 次询问，询问 $a[x][y]$ 的值

- $n \leq 3 \times 10^6, \sum_{i=1}^{n} s[i] \leq 5 \times 10^6$。

Solution

直接开静态数组，大小为 $3 \times 10^6 \times 5 \times 10^6$，开不下。

需要使用动态数组 vector，可以使用 push_back 一个个把元素放到a[i]末尾，也可以使用 a[i].resize(s[i]) 得到一个大小为 s[i] 的数组。

注意vector的下标从0开始。

代码

```cpp
const int N = 3e6 + 10;
int n, q, s[N]; unsigned int ans = 0;
vector<unsigned int> a[N];

int main() {
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> s[i];
        a[i].resize(s[i] + 1);
        for (int j = 1; j <= s[i]; j++) cin >> a[i][j];
    }
    for (int i = 1; i <= q; i++) {
        int x, y; cin >> x >> y;
        ans ^= a[x][y];
    }
    cout << ans << "\n";
    return 0;
}
```

std::queue

- `q.push()`
- `q.front()`
- `q.pop()`
- `q.empty()`
- `q.size()`

P5661 公交换乘

旅游城市 B 市推出了一种地铁换乘公交车的优惠方案：

1. 在搭乘一次地铁后可以获得一张优惠票，有效期为 45 分钟，在有效期内可以消耗这张优惠票，免费搭乘一次票价不超过地铁票价的公交车。在有效期内指开始乘公交车的时间与开始乘地铁的时间之差小于等于 45 分钟，即：$t_{bus} - t_{subway} \leq 45$。

2. 搭乘地铁获得的优惠票可以累积，即可以连续搭乘若干次地铁后再连续使用优惠票搭乘公交车。

3. 搭乘公交车时，如果可以使用优惠票一定会使用优惠票；如果有多张优惠票满足条件，则优先消耗获得最早的优惠票。

现在你得到了小轩最近的公共交通出行记录，你能帮他算算他的花费吗?

数据范围：$n \leq 10^5$，$t_i \leq 10^9$，$1 \leq price_i \leq 1000$。

Solution

存一个队列，按时间先后顺序存储地铁的乘车记录。

坐公交时，弹出队列中时间距离当前时间超过 45 的乘车记录。

由于同一分钟只会有一条乘车记录，所以队列中最多只有 45 条乘车记录。接着枚举所有乘车记录，找到其中最前面的超过当前公交乘车费用的记录即可。

代码

```
for (int i = 1; i <= n; i++) {
    int op;
    cin >> op >> p[i] >> t[i];
    if (op == 0) ans += p[i], q.push(i);
    else {
        while (!q.empty() && t[i] - t[q.front()] > 45) q.pop();
        queue<int> tmp; bool flag = false;
        while (!q.empty()) {
            int id = q.front(); q.pop();
            if (flag == 0 && p[id] >= p[i]) flag = true;
            else tmp.push(id);
        }
        swap(q, tmp);
        if (flag == 0) ans += p[i];
    }
}
```

std::stack

- S.push()
- S.top()
- S.pop()
- S.size()

P1241 括号序列

给定一个包含 “[]()” 四种字符的字符串，将其补充为一个合法的括号序列（未配对的括号，需要在它旁边加一个和它匹配的字符）。

对于全部数据，$|s| \leq 100$。

Solution

使用栈存储所有未被匹配的左括号，最近的未被匹配的左括号即栈顶元素。

我们每匹配成功一对括号，就将它们打上标记，然后左括号出栈。

最后输出时，有标记的括号可以直接输出，没有标记的括号就要补充成一对合法的括号。

代码

```
cin >> s;
n = strlen(s);
for (int i = 0; i < n; i++) {
    if (s[i] == '(' || s[i] == '[')
        stk.push(i);
    else {
        if (stk.size() == 0)
            continue;
        int j = stk.top();
        if (s[j] == '(' && s[i] == ')')
            vis[j] = vis[i] = true, stk.pop();
        else if (s[j] == '[' && s[i] == ']')
            vis[j] = vis[i] = true, stk.pop();
    }
}
```

P1901 发射站

某地有 N 个能量发射站排成一行，每个发射站 i 都有不相同的高度 H_i，并能向两边（两端的发射站只能向一边）同时发射能量值为 V_i 的能量，发出的能量只被两边最近的且比它高的发射站接收。

请计算出接收最多能量的发射站接收的能量是多少。

对于 100% 的数据，$1 \leq N \leq 10^6, 1 \leq H_i \leq 2 \times 10^9, 1 \leq V_i \leq 10^4$。

Solution

找左边第一个比 i 高的发射站和找右边第一个比 i 高的发射站对称的，所以从左往右和从右往左各做一次。

假设 $h[0] = h[n+1] = +\infty$。以从左到右为例：

维护一个栈，从栈底到栈顶元素单调递减。初始时，栈中只放入一个元素 0。

之后，当我们要查询左边第一个比 i 高的发射站时，就在栈中不断弹出高度小于等于它的元素。剩下的栈顶就是第一个比它高的发射站。

计算完答案之后，需要把元素 i 放入栈中。

注意使用 long long，答案最大为 10^{10}。

时间复杂度 $O(n)$。

代码

```
cin >> n;
for (int i = 1; i <= n; i++) cin >> h[i] >> v[i];
h[0] = h[n + 1] = INT_MAX;
stk.push(0);
for (int i = 1; i <= n; i++) {
    while (h[i] >= h[stk.top()]) stk.pop();
    ans[stk.top()] += v[i];
    stk.push(i);
}
while (stk.size()) stk.pop();
stk.push(n + 1);
for (int i = n; i >= 1; i--) {
    while (h[i] >= h[stk.top()]) stk.pop();
    ans[stk.top()] += v[i];
    stk.push(i);
}
```


线性表
链表
顺序表
单向链表
双向链表
循环链表

洛谷P1160 队列安排

一个学校里老师要将班上 N 个同学排成一列，同学被编号为 $1 \sim N$，他采取如下的方法：

1. 先将 1 号同学安排进队列，这时队列中只有他一个人；

2. $2 \sim N$号同学依次入列，编号为 i 的同学入列方式为：老师指定编号为 i 的同学站在编号为 $1 \sim i-1$ 中某位同学（即之前已经入列的同学）的左边或右边；

3. 从队列中去掉 M 个同学，其他同学位置顺序不变。

在所有同学按照上述方法队列排列完毕后，老师想知道从左到右所有同学的编号。

对于 100% 的数据，$1 < M \leq N \leq 10^5$。

Solution

使用双向链表快速进行插入和删除。

记 $nex[i]$ 为 i 右边的同学的编号，$las[i]$ 为 i 左边的同学的编号。

在队列中插入一个同学 i 时，需要修改编号为 k、i、$las[k]$/$nex[k]$ 的相应信息（它们的 nex/las 值）。

在队列中删除一个同学 i 时，需要先判断它是否已被删除，然后修改它左边同学的 nex 值和右边同学的 las 值。

但是这样在队列第一个同学和最后一个同学位置时需要特殊判断，比较麻烦。

可以假想存在两个编号分别为 0 和 $n+1$ 的同学，它们始终在队列最左边和最右边。这样就规避了边界的处理。

总时间复杂度为 $O(n+m)$。

代码

```
nex[1] = n + 1, las[n + 1] = 1;
las[1] = 0, nex[0] = 1;
for (int i = 2; i <= n; i++) {
    int k, p; cin >> k >> p;
    if (p == 0) {
        las[i] = las[k]; nex[las[k]] = i;
        las[k] = i; nex[i] = k;
    }
    else {
        nex[i] = nex[k]; las[nex[k]] = i;
        nex[k] = i; las[i] = k;
    }
}
// 建立初始队列
```

```
cin >> m;
for (int i = 1; i <= m; i++) {
    int x; cin >> x;
    if (vis[x]) continue;
    vis[x] = true;
    nex[las[x]] = nex[x];
    las[nex[x]] = las[x];
}
int now = nex[0];
while (now != n + 1) {
    cout << now << " ";
    now = nex[now];
}
// 删除 m 个元素并输出
```

P7912 小熊的果篮

给定一个长度为 n 的 01 串，所有相邻的 0 或者 相邻的 1 组成一个块。

执行若干次操作，每次操作从每个块中取出最前面的一个元素，并输出这些元素在最开始的 01 串中的位置。直到将所有的元素全部取完，操作停止。

如果一个块被删光了，和它相邻的两个块会合并。

对于所有数据，$1 \leq n \leq 2 \times 10^5$。

Solution

使用链表套链表的形式来维护块。

每个块内部使用链表连起来，删除的时候只需要把链表的头指针后移即可。

然后块与块之间也用链表连起来（给每个块一个编号），删除某个块时需要把相邻的两个块合并成一个块。

合并时注意边界情况！！！若额外维护了头指针，不能在合并时把头指针给合并掉。

代码

```
for (int l = 1; l <= n; l++) {
    head[++cnt] = l, las[cnt] = l;
    int r = l + 1;
    while (r <= n && a[r] == a[l])
{
        nex[r - 1] = r;
        las[cnt] = r;
        r++;
    }
    NEXT[cnt - 1] = cnt;
    l = r - 1;
}
// 建立链表
```

```
while (NEXT[0] != 0) {
    int now = NEXT[0];
    while (now != 0) {
        cout << head[now] << " ";
        head[now] = nex[head[now]];
        now = NEXT[now];
    }
    cout << "\n";
    now = 0;
    while (NEXT[now] != 0) {
        if (!head[NEXT[now]]) {
            NEXT[now] = NEXT[NEXT[now]];
            if (now && NEXT[now]) {
                nex[las[now]] = head[NEXT[now]];
                if (head[NEXT[now]])
                   las[now] = las[NEXT[now]];
                NEXT[now] = NEXT[NEXT[now]];
            }
        }
        else now = NEXT[now];
    }
}
```

std::priority_queue

- 默认大根堆
- 小根堆可以写成`priority_queue<int, vector<int>, greater<int>>`
- `Q.top()` 返回堆顶的值
- `Q.pop()` 弹出堆顶
- `Q.push(x)`
- `Q.size()`
- `Q.empty()`

P1090 合并果子

有 n 堆果子，每堆数量为 $a[i]$，合并两堆果子消耗的体力为这两堆的果子数量总和。

经过 $n-1$ 次合并后，果子会变成一堆。问把果子合并成一堆所需要消耗的最小体力值是多少。

对于全部的数据，保证有 $n \leq 10000$。

Solution

最优方案一定是每次选取两个大小最小的堆，将它们合并成一个堆。

若将每堆果子对应到一棵哈夫曼树的节点上，初始时每堆果子对答案的贡献就是它的大小乘以它到根的路径长度。

所以我们一定是把大小更小的堆先合并。

时间复杂度 $O(n \log n)$。

代码

```cpp
int n; cin >> n;
priority_queue<int, vector<int>, greater<int> > heap;
for (int i = 1; i <= n; i++) {
    int x; cin >> x;
    heap.push(x);
}
long long ans = 0;
for (int i = 1; i < n; i++) {
    int p = heap.top();
    heap.pop();
    int q = heap.top();
    heap.pop();
    ans += p + q;
    heap.push(p + q);
}
cout << ans << "\n";
```

P1168 中位数

https://www.luogu.com.cn/problem/P1168

给定一个长度为 N 的非负整数序列 A，对于前奇数项求中位数。

对应 100% 的数据，$1 \leq N \leq 100000, 0 \leq A_i \leq 10^9$。

Solution

假设我们要计算前 2i + 1 个数的中位数。

维护一个大根堆 p 存其中前 i 小的数，一个小根堆 q 存其中前 i 大的数，再维护一个变量 mid 表示中位数。

采用递推的方式依次计算前奇数项的中位数，由前 $2i-1$ 项维护的信息推出前 $2i+1$ 项的答案。

将 $A[2i]$ 和 $A[2i+1]$ 分别与 mid 进行比较，

1. 若一大一小，则将小的放入 p，大的放入 q，mid 保持不变。
2. 若两者都小于等于 mid，则将两者都放入 p 中，将 mid 放入 q 中，最后取出 p 的堆顶来更新 mid。
3. 若两者都大于等于 mid，则将两者都放入 q 中，将 mid 放入 p 中，最后取出 q 的堆顶来更新 mid。

总时间复杂度为 $O(n \log n)$。

代码

```
cin >> n;
for (int i = 1; i <= n; i++) cin >> A[i];
mid = A[1]; cout << mid << "\n";
for (int i = 3; i <= n; i += 2) {
    if (A[i - 1] <= mid && A[i] <= mid) {
        p.push(A[i - 1]); p.push(A[i]);
        q.push(-mid);
        mid = p.top(); p.pop();
    }
    else if (A[i - 1] >= mid && A[i] >= mid) {
        p.push(mid);
        q.push(-A[i - 1]); q.push(-A[i]);
        mid = -q.top(); q.pop();
    }
    else {
        if (A[i - 1] > A[i]) swap(A[i - 1], A[i]);
        p.push(A[i - 1]);
        q.push(-A[i]);
    }
    cout << mid << "\n";
}
```