鸡尾酒QAQ
编辑于 10-15 15:39 上海

+ 关注

10.11 OI 集训营普及组第四场题解

T1 制裁复读机

要注意的是：每次重复是当前数字之前的所有字符，而不是只重复单个字符或部分字符。

剩下的还是考察模拟能力：

开一个答案字符串。

每次循环，若为小写字母，则在答案字符串上加上当前字符。

若为数字，向后统计，并移动当前循环指针 i，将答案字符串复制拼接 $x-1$ 次即可，其中 x 是数字大小。

```
#include<bits/stdc++.h>
using namespace std;

int t,n,tmp,pos;
string a,b,c;

int main(){
    cin >> t;
    while(t--){
        cin >> n >> a;
        a = " " + a, b = "";
        for(int i = 1; i <= n; ++i){
            if(a[i] >= 'a' && a[i] <= 'z') b += a[i];
            else{
                tmp = 0,pos = i;
                while(a[pos] >= '0' && a[pos] <= '9'){
                    tmp *= 10;
                    tmp += a[pos] - '0';
                    pos++;
                }
                i=pos-1;
                c=b,tmp--;
                while(tmp--)b+=c;
            }
        }
        cout<<b<<endl;
    }
    return 0;
}
```

T2 攻与防

思路 1：前缀和

求所有 0 的位置的前缀和，所有 1 的位置的前缀和。然后枚举断点，进行比较。

思路 2：递推

假设一开始分界线在位置 0（即所有人都在第二阵营），然后逐渐移动分界线，维护双方战斗力的变化。

注意特殊情况：分界线有可能在位置 0 或位置 n

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(){
    int n;
    string s;
    cin >> n >> s;
    s = " " + s;
    vector<long long>pre1(n+5, 0), pre0(n+5, 0);
    for(int i = 1; i < s.size(); i++){
        pre1[i] = pre1[i-1] + (s[i] == '1') * i;
        pre0[i] = pre0[i-1] + (s[i] == '0') * i;
    }
    long long ans = 1e9;
    for(int i = 1; i <= n; i++){
        ans = min(ans, abs(pre0[i-1] - pre1[n] + pre1[i-1]));
    }
    cout << ans << endl;
} // 前缀和思路
```

T3 四月是你的谎言

首先，n非常小，并且单词长度非常小，我们在字符串寻找单词时可以使用BF算法暴力匹配。

其次，当只有一个单词时，对于可以在字符串中找到的单词，我们可以将其任意一位改成'*'，即可使单词无法被找到，显然（明明就是不会严格证，非要说显然），改动最后一位最优。

感性理解一下，在一个单词中有可能前缀和后缀重复，例如"aabaa"，而字符串是"ooooaabaaabaaa"。在[1,10]的前缀中可以找到第一个"aabaa"的位置为[6,10]，修改其中的任意一个位置都可以使得[1,10]的前缀中无法找到"aabaa"。我们可以发现改动[6,9]之间的任意一位，[10,14]依然可以找到"aabaa"，需要再次更改。而改动[10]的位置，则只通过一次改动，就可以使[10,14]也找不到"aabaa"。

那么，对于多个单词的情况，也是将最后一位进行更改，此时有可能出现的问题是一个单词是另一个单词的前缀。例如单词有"abcd"、"abc"、"ab"，在字符串"abcde"中，若我们先寻找"abcd"，并将最后一位改成'*'则会得到"abc*e"，此时依然可以找到"abc"，则再将字符串变成"ab**e"，依此类推，再将字符串变成"a***e"，需要三次操作。实际上，将字符串变成"a*cde"，所有单词都无法再被找到，只需要一次操作。因此，我们需要寻找一种合理的枚举方式，

因此，一个容易的写法是将单词按长度排序，之后从短到长在字符串中寻找，并在每次找到后更改最后一位。 **然后这个做法就被hack了！** 例如单词为"ab"、"baa"，字符串为"baab"，若按"ab"、"baa"顺序枚举，则字符串会依此变为"baa*"，"ba**"，共修改2次，实际上只需要修改1次变成"ba*b"即可。

另一个很直观而且容易想到的写法是枚举字符串每一位，往后从短到长寻找每一个单词是否存在，将找到的最短的单词进行修改。 **然后这个做法又被hack了！** 例如单词为"asdsa"、"sds"，字符串为"asdsa"时，按此写法在第一位时会先找到"asdsa"，将其改成"asds*"，那么在第二位时会找到"sds"，导致错误。

正确的枚举写法是对字符串每一位往前寻找，然后若是找到任意一个以这一位结尾的单词，则更改这一位（别再被hack了）。

证明如下：改 $[i]$ 的前提是区间 $[0,i-1]$ 之间找不到任意一个单词，然后 $[i]$ 第一次出现单词，不妨设单词为区间 $[i_0,i]$ ，同时区间 $[i,j]$ 也是一个单词，此时改 $[i]$ 相比改 $[k](k\in[i0,i-1])$ 是更优的，因为如果改 $[k]$ 无法保证区间 $[i,j]$ 里的单词可以被改掉。

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    int T=1;
    cin>>T;
```

```
    while(T--){
        int n;
        cin>>n;
        vector<string> ve(n);
        for(auto &i : ve){
            cin>>i;
        }
        string s;
        cin>>s;
        n = s.size();
        s = " " + s + " ";
        for(int i=1;i<=n;i++){
            for(auto &j : ve){
                if(i-int(j.size())<0) continue;
                if(s.substr(i-j.size()+1,j.size())==j) s[i] = '*';
            }
        }
        cout<<s.substr(1,n)<<endl;
    }
    return 0;
}
```

T4 嘤嘤的珂朵莉树

首先，可以发现，我们只需要查询一次，并且每个操作之间无关，那么我们可以在做完所有操作后再进行计算。

其次，若是对同一个点进行多次操作一，可以将这多次操作一合成一次操作一，操作一的权值为多次操作一的权值和，而由其他结点传递过来的操作一也可以和当前结点的操作一进行合并。操作二同理。

对于操作一，可以发现，深度最大的结点不会受到对其他结点进行操作一的影响，那么深度最大的结点操作完后，深度次大的结点就不会再次受到影响。依此类推，我们可以根据结点深度从大到小进行操作一，这样只需要进行一次遍历即可完成所有的操作一操作。

对于操作二，类似于操作一，可以根据结点深度从小到大进行操作。

因此，我们只需要先dfs或bfs处理出各个结点的深度，然后将操作进行记录，在up和down数组中记录操作一和操作二累积的权值，再根据操作按深度从小到大或从大到小进行传递，最后将up、down数组与原本的权值数组进行求和输出即可。

```
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
const int N = 1e6 + 10;
const ll mod = 1e9 + 7;

ll a[N], up[N], down[N];
int n, m, q, dep[N];
int cnt, head[N];
struct edge {
    int to, nxt;
} e[N << 1];
vector<pair<int,int> >s;

void ins(int u, int v) {
    e[++cnt] = (edge) {v, head[u]};
    head[u] = cnt;
}

void dfs(int u, int fa) {
    s.push_back(make_pair(dep[u], u));
    for(int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == fa) continue;
        dep[v] = dep[u] + 1;
        dfs(v, u);
    }
}
```

```cpp

int main() {
    cin >> n >> m >> q;
    for(int i = 1; i <= n; ++i) cin >> a[i];
    for(int i = 1; i < n; ++i) {
        int u, v;
        cin >> u >> v;
        ins(u, v), ins(v, u);
    }
    dfs(1, 1);
    for(int i = 1; i <= m; ++i) {
        int u, v;
        cin >> u >> v;
        ins(u, v), ins(v, u);
    }
    for(int i = 1; i <= q; ++i) {
        int u, c, op;
        cin >> op >> u >> c;
        if(op == 1) (up[u] += c) %= mod;
        else (down[u] += c) %= mod;
    }
    sort(s.begin(), s.end());
    for(int cur = 0; cur < s.size(); ++cur) {
        int u = s[cur].second;
        for(int i = head[u]; i; i = e[i].nxt) {
            int v = e[i].to;
            if(dep[v] > dep[u]) (down[v] += down[u]) %= mod;
        }
    }
    reverse(s.begin(), s.end());
    for(int cur = 0; cur < s.size(); ++cur) {
        int u = s[cur].second;
        for(int i = head[u]; i; i = e[i].nxt) {
            int v = e[i].to;
            if(dep[v] < dep[u]) (up[v] += up[u]) %= mod;
        }
    }
    for(int i = 1; i <= n; ++i) cout << (a[i]+up[i]+down[i])%mod << " ";
    puts("");
}
```