

# 2025-2026 秋《数据结构与算法》第一次实验题解

助教-陈孙一硕

## Difficulties:

- Easy: A,B,C
- Medium-Easy: D
- Medium: E
- Medium-Hard: F
- Hard: G

## A 吃不完！

经典约瑟夫环问题，用队列去模拟解决。

```
#include <iostream>
using namespace std;
int josephus(int n, int m) {
    int last = 0;
    for (int i = 2; i <= n; i++) {
        last = (last + m) % i;
    }
    return last + 1;
}
int main() {
    int n, m;
    while (cin >> n >> m) {
        if (n == 0 && m == 0) break;
        cout << josephus(n, m) << endl;
    }
}
```

```
    return 0;  
}
```

## B 火车进站

用一个变量 *expected* 去记录下一个需要接上去的编号。从前往后遍历列车，如果可以接上去就接上去，否则放到栈里去。如果无法接上，那就判断栈顶的元素能否接上。按照这个思路模拟即可。

```
#include <iostream>  
#include <stack>  
#include <vector>  
using namespace std;  
const int N=2e6+101;  
int a[N];  
void solve() {  
    int n; cin >> n;  
    for (int i = 1; i <= n; i++) cin >> a[i];  
    stack<int> st;  
    int expected = 1;  
    int count = 0;  
    for (int i = 1; i <= n; i++) {  
        if (a[i] == expected) {  
            count++;  
            expected++;  
        } else {  
            st.push(a[i]);  
        }  
        while (!st.empty() && st.top() == expected) {  
            st.pop();  
            expected++;  
        }  
    }  
    while (!st.empty() && st.top() == expected) {  
        st.pop();  
        expected++;  
    }
```

```

    }

    if (expected == n+1) {
        cout << "Yes" << endl;
        cout << count << endl;
    } else {
        cout << "No" << endl;
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int T; cin >> T;
    while (T--) solve();
    return 0;
}

```

## C 用两个栈实现队列

上学期，我把这道题作为了弘深班期末考试的第一题，这里拿出来给大家练一练。难度并不大，按照题意进行模拟即可，主要考察 STL 中 stack 的操作。

```

#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n; cin>>n;
    stack <int> s1,s2;
    while (n--){
        char op; cin>>op;
        if (op=='I'){
            int num; cin>>num;
            s1.push(num);
        }else{
            if (s2.empty()){
                if (s1.empty()) cout<<"ERROR"<<endl;

```

```

        else{
            int t=0;
            while (!s1.empty()){
                s2.push(s1.top());
                s1.pop();
                t+=2;
            }
            t++;
            cout<<s2.top()<<" "<<t<<endl;
            s2.pop();
        }
    }
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int T=1; //cin>>T;
    while (T--) solve();
}

```

## D 字典序最大出栈序列

首先从后往前遍历，记录到位置  $i$  时数组的后缀最大值。在每个数入栈后，如果当前的栈顶元素大于  $b[i + 1]$ ，说明此后要入栈的元素已经没有比当前栈顶元素大的了。由于需要维护出栈序列字典序最大，所以此时把栈顶元素出栈就可以了。按照这个思路去模拟，代码实现并不困难。

```

#include <iostream>
#include <stack>
using namespace std;

```

```

#define int long long
int a[1000005], b[1000005];
stack<int> q;

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    for (int i = n - 1; i >= 0; i--)
        b[i] = max(a[i], b[i + 1]);
    for (int i = 0; i < n; i++) {
        q.push(a[i]);
        while (!q.empty() && q.top() > b[i + 1]) {
            cout << q.top() << " ";
            q.pop();
        }
    }
    return 0;
}

```

## E 点兵点将

题目等价于求原序列两个不相交的非空字段的最大和。这是一个经典的 Kadane 算法。名字听起来很高级，其实就是对于前缀和后缀的灵活运用。

用  $predp[i]$  表示以  $a[i]$  为结尾的子段的最大和， $prebest[i]$  表示  $a[1] \sim a[i]$  的子段的最大和。这两个数组的值可以在  $O(n)$  时间内预处理得到。

同样地，用  $sufdp[i]$  表示以  $a[i]$  开头的子段的最大和， $sufbest[i]$  表示  $a[i] \sim a[n]$  的子段的最大和。也可以  $O(n)$  预处理。

此时只需  $O(n)$  枚举断点，不断更新答案最大值即可。

时间复杂度  $O(n)$ 。

```
#include<bits/stdc++.h>
```

```

using namespace std;
#define int long long
const int N=2e6+101;
const int inf=1e18;
int a[N],predp[N],sufdp[N],prebest[N],sufbest[N];
void solve(){
    int n; cin>>n;
    for (int i=1;i<=n;i++) cin>>a[i];
    for (int i=0;i<=n+1;i++) predp[i]=sufdp[i]=prebest[i]=sufbest[i]=-inf;
    for (int i=1;i<=n;i++){
        predp[i]=max(a[i],predp[i-1]+a[i]);
        prebest[i]=max(prebest[i-1],predp[i]);
    }
    for (int i=n;i>=1;i--){
        sufdp[i]=max(a[i],sufdp[i+1]+a[i]);
        sufbest[i]=max(sufbest[i+1],sufdp[i]);
    }
    int ans=-inf;
    for (int i=2;i<=n;i++) ans=max(ans,prebest[i-1]+sufbest[i]);
    cout<<ans<<endl;
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int T=1; //cin>>T;
    while (T--) solve();
    return 0;
}

```

## F 幂的 gcd

首先，若  $a, c$  互素，显然对于任意的  $b, d \geq 1$ ，均有  $\gcd(a^b, c^d) = 1$ 。

不妨设  $b \leq d$ ，那么  $\gcd(a^b, c^d) = \gcd(a^b, c^b \cdot c^{d-b})$ 。设  $g = \gcd(a, c)$ ，那么  $\gcd(a^b, c^b \cdot c^{d-b}) = g \cdot \gcd\left(\left(\frac{a}{g}\right)^b, \left(\frac{c}{g}\right)^b \cdot c^{d-b}\right)$ 。由于  $\frac{a}{g}, \frac{c}{g}$  互素，所以  $\gcd\left(\left(\frac{a}{g}\right)^b, \left(\frac{c}{g}\right)^b \cdot c^{d-b}\right) = \gcd\left(\left(\frac{a}{g}\right)^b, c^{d-b}\right)$ 。综上， $\gcd(a^b, c^d) = g \cdot \gcd\left(\left(\frac{a}{g}\right)^b, c^{d-b}\right)$ 。递归求解即可。由于每次的  $g \geq 2$ ，所以递归次数不会超过

$\log a$  次。算上每次计算快速幂的时间，单组测试时间复杂度  $O(\log(\max(a, c)) \cdot \log(\max(b, d)))$ 。

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int mod=998244353;
int gcd(int a,int b){
    if (a<b) swap(a,b);
    if (a%b==0) return b;
    return gcd(b,a%b);
}
int ksm(int a,int b){
    a%=mod;
    int res=1;
    while (b){
        if (b&1) res=res*a%mod;
        a=a*a%mod;
        b>>=1;
    }
    return res;
}
int calc(int a,int b,int c,int d){
    if (d<b) swap(b,d),swap(a,c);
    int g=gcd(a,c);
    if (g==1) return 1;
    return ksm(g,b)*calc(a/g,b,c,d-b)%mod;
}
void solve(){
    int a,b,c,d; cin>>a>>b>>c>>d;
    cout<<calc(a,b,c,d)<<endl;
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int T=1; cin>>T;
```

```

while (T--) solve();
}

```

## G みんなで作ろう！簡単な電卓

**解读：**先翻译一下题面：大家一起来做吧！简单的计算器。

这个题是李老师提出的想法，由我进行题面的撰写和测试数据的编写。我承认这个题目确实有些麻烦，我自己调试的时候也花了一些时间。值得一提的是，由于内存限制只有 64MB，所以我最开始写的 std 在 PTA 上提交之后出现了 MLE（我相信一定有同学也遇到了这个问题）。后来经过调试，到 TLE 再到 AC，再到写出一版耗费时间和空间都更少的代码，这道题也是让我经历了很多心路历程。下面我会把我两版的代码都贴出来，大家可以参考。

### 问题 1：MLE

评测详情				
测试点	提示	内存(KB)	用时(ms)	结果
0		63076	38	答案正确
1		63148	43	答案正确
2		63076	40	答案正确
3		63016	39	答案正确
4		63012	39	答案正确
5		63160	40	答案正确
6		65536	59	内存超限

图 1: 第 7 题-MLE

出现这个问题的原因大概率是最开始就把字符串  $s$  直接读入并且存起来了（离线处理）。正确的做法应该是一个一个字符去读入，也就是读一个处理一个（在线处理）。

### 问题 2：TLE

评测详情				
测试点	提示	内存(KB)	用时(ms)	结果
0		62952	40	答案正确
1		62964	41	答案正确
2		63136	40	答案正确
3		62948	40	答案正确
4		62936	40	答案正确
5		63084	41	答案正确
6		63088	400	运行超时

图 2: 第 7 题-TLE

解决了上述 MLE 的问题之后，有一种做法是先把后缀表达式求出来，存起来再处理。这种做法几乎是卡着空间没有 MLE（印象当中内存到了 60000+KB 了）。由于这道题的输入量非常大，经过我的测试发现，这种 TLE 是可以通过关闭输入流同步来避免的。

## 参考代码 1：常规做法，需要关闭流同步避免 TLE

```
#include<iostream>
#include<stack>
#include<map>
using namespace std;
#define int long long
stack<char> opstk;
stack<int> numstk;
map<char,int> pri;
string lst[2000005];
int idx=0,ans1=-1,ans2=-1,opsz=0,numsz=0;
bool last=1;
string ans="";
void make(){
    char c;
    while (cin>>c){
        if (c>='0'&&c<='9'){
            if (last){
                ans+=c;
            }else{
                ans+=c;
            }
            last=0;
        }else{
            last=1;
            if (ans!=""){
                idx++;
                lst[idx]=ans;
                ans="";
            }
            if (c=='('){
                opstk.push(c);
                opsz++;
                ans1=max(ans1,opsz);
            }else if (c==')'){

```

```

        while (opstk.top() != '('){
            idx++;
            lst[idx] = "";
            lst[idx] += opstk.top();
            opstk.pop();
            opsz--;
        }
        opstk.pop();
        opsz--;
    }else{
        while (!opstk.empty() && pri[opstk.top()] >= pri[c]){
            idx++;
            lst[idx] = "";
            lst[idx] += opstk.top();
            opstk.pop();
            opsz--;
        }
        opstk.push(c);
        opsz++;
        ans1 = max(ans1, opsz);
    }
}
if (ans != ""){
    idx++;
    lst[idx] = ans;
    ans = "";
}
while (!opstk.empty()){
    idx++;
    lst[idx] = "";
    lst[idx] += opstk.top();
    opstk.pop();
    opsz--;
}
}

```

```

int getnum(string s){
    int a=0;
    for (size_t i=0;i<s.size();i++){
        a=a*10+s[i]-'0';
    }
    return a;
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    pri['+']=1,pri['-']=1,pri['*']=2,pri['/']=2;
    make();
    for (int i=1;i<=idx;i++){
        if (lst[i][0]>='0'&&lst[i][0]<='9'){
            numstk.push(getnum(lst[i]));
            numsz++;
            ans2=max(ans2,numsz);
        }else{
            char op=lst[i][0];
            int num2=numstk.top();
            numstk.pop();
            int num1=numstk.top();
            numstk.pop();
            numsz-=2;
            if (op=='+' ){
                numstk.push(num1+num2);
            }else if (op=='-' ){
                numstk.push(num1-num2);
            }else if (op=='*' ){
                numstk.push(num1*num2);
            }else if (op=='/' ){
                if (num2==0){
                    cout<<"error:<<num1<<"/<<num2<<endl;
                    return 0;
                }
                numstk.push(num1/num2);
            }
        }
    }
}

```

```

    }
    numsz++;
}
}

cout<<numstk.top()<<endl;
cout<<ans1<<" "<<ans2<<endl;
return 0;
}

```

评测详情				
测试点	提示	内存(KB)	用时(ms)	结果
0		63100	40	答案正确
1		63184	44	答案正确
2		63096	42	答案正确
3		63168	41	答案正确
4		63048	42	答案正确
5		63024	42	答案正确
6		63060	209	答案正确

图 3: 第 7 题朴素版（关闭流同步答案正确）

后来我想了一种优化的做法，只要读取了数字，就立马放到栈里；只要读取到运算符也是如此。如果出现运算符出栈，那就从数字栈顶取出两个数执行运算，再将运算结果入栈。这样的做法在时间和空间上都是优于上一种做法的，且不需要关闭流同步也不会 TLE.

## 参考代码 2：时空优化版

```

#include<iostream>
#include<stack>
#include<map>
using namespace std;
#define int long long
stack<char> opstk;
stack<int> numstk;
map<char,int> pri;
int idx=0,ans1=-1,ans2=-1,opsz=0,numsz=0;
bool last=1,flag=1;
string ans="";
int getnum(string s){
    int a=0;
    for (size_t i=0;i<s.size();i++){

```

```

        a=a*10+s[i]-'0';
    }
    return a;
}

void cal(char op){
    int num2=numstk.top();
    numstk.pop();
    int num1=numstk.top();
    numstk.pop();
    if (op=='+') {
        numstk.push(num1+num2);
    }else if (op=='-') {
        numstk.push(num1-num2);
    }else if (op=='*') {
        numstk.push(num1*num2);
    }else if (op=='/') {
        if (num2==0) {
            cout<<"error:"<<num1<<"/"<<num2<<endl;
            flag=0;
            return;
        }
        numstk.push(num1/num2);
    }
    numsz--;
}

void solve(){
    char c;
    while (cin>>c){
        if (c>='0'&&c<='9'){
            ans+=c;
            last=0;
        }else{
            last=1;
            if (ans!=""){
                numstk.push(getnum(ans));
                numsz++;
            }
        }
    }
}

```

```

        ans2=max(ans2,numsz);
        ans="";
    }
    if (c=='('){
        opstk.push(c);
        opsz++;
        ans1=max(ans1,opsz);
    }else if (c==')'){
        while (opstk.top()!='('){
            char op=opstk.top();
            opstk.pop();
            opsz--;
            cal(op);
            if (!flag) return;
        }
        opstk.pop();
        opsz--;
    }else{
        while (!opstk.empty()&&opstk.top()!=''&&pri[opstk.top()]>=pri[c]){
            char op=opstk.top();
            opstk.pop();
            opsz--;
            cal(op);
            if (!flag) return;
        }
        opstk.push(c);
        opsz++;
        ans1=max(ans1,opsz);
    }
}
if (ans!=""){
    numstk.push(getnum(ans));
    numsz++;
    ans2=max(ans2,numsz);
}

```

```

while (!opstk.empty()){
    char op=opstk.top();
    opstk.pop();
    opsz--;
    cal(op);
    if (!flag) return;
}

signed main(){
//ios::sync_with_stdio(false);
//cin.tie(nullptr);
pri['+']=1,pri['-']=1,pri['*']=2,pri['/']=2;
solve();
if (flag){
    cout<<numstk.top()<<endl;
    cout<<ans1<<" "<<ans2<<endl;
}
return 0;
}

```

评测详情					
测试点	显示	内存(KB)	用时(ms)	结果	得分
0		556	2	答案正确	1 / 1
1		608	2	答案正确	2 / 2
2		564	2	答案正确	2 / 2
3		560	2	答案正确	3 / 3
4		572	2	答案正确	3 / 3
5		572	2	答案正确	4 / 4
6		608	140	答案正确	5 / 5

图 4: 第 7 题-时空优化版